EN2550 Assignment 2 on Fitting and Alignment

Name: K. D. S. D. Kuruppu
Index no: 190338C
GitHub: https://github.com/SupunDK/EN2550-Assignment-2.git

Question 1

Parameters of the RANSAC algorithm

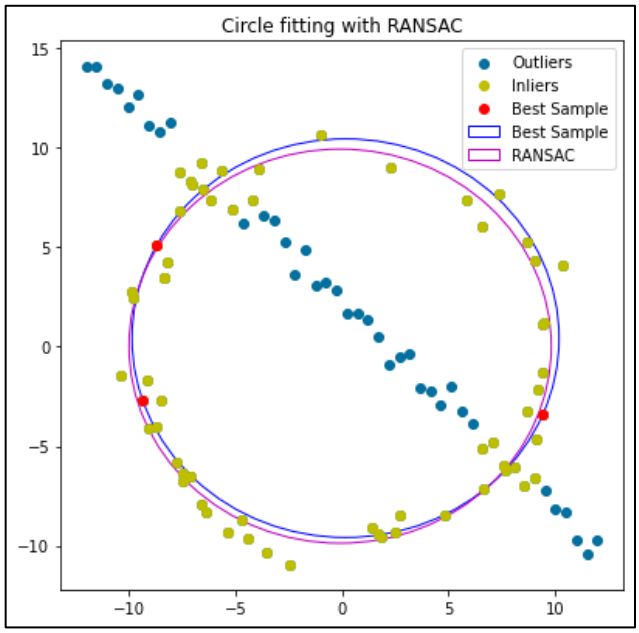
- s = 3
  - The minimum number of points needed to estimate a circle is three. Hence, "S" is taken as 3.
- t = 1.96
  - The set of points corresponding to a circle has been corrupted by mean-zero variance-one Gaussian noise. Therefore, a threshold of 1.96 gives a 0.95 probability of capturing all inliers.
- d = 50
  - The entire dataset consists of 100 points in which 50 are outliers and the remaining 50 are inliers. Therefore, when an accurate circle estimation has been done, it should contain all the 50 inlier points. Therefore, the consensus set size has been set to 50.
- N = 35
  - Since the initial number of points (s) is chosen as 3, by the Number of Samples vs. Outlier Ratio table, the number of samples (N) should be chosen as 35 to have a probability of 0.99 for selecting at least one outlier free random sample.

a) The implementation of the RANSAC algorithm for circle estimation is as follows,

```
def RANSAC_circle(X):
    # Parameter for the RANSAC algorithm
    s = 3
    t = 1.96
    d = 50
    N = 35

    best_fit_circle = None
    best_fit_x = None
    best_inlier_count = 0

    for _ in range(N):
        x = []

        for i in range(s):
            hold = X[np.random.randint(0, 100), :]

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

                x.append(hold)
            else:
                x.append(hold)

        a, b, r = estimate_circle(x[0], x[1], x[2])

        if a == None:
            continue

        count, inliers = get_inlier_count(a, b, r, X, t)

        if count > best_inlier_count:
            best_fit_circle = plt.Circle((a, b), r, color='b', fill=False, label="Best Sample")
            best_fit_x = x
            best_fit_inliers = inliers
            best_inlier_count = count

    if best_inlier_count < d:
        print("The RANSAC algorithm did not find a suitable model")
```

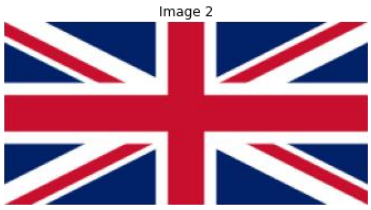
```
        return None, None, None, None

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

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

    return ransac_circle, best_fit_circle, best_fit_x, best_fit_inliers
```

b) A circle fitting for the given set of points through the RANSAC algorithm is as follows,

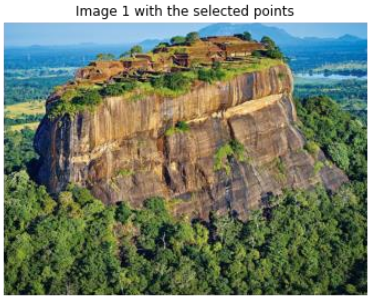


Question 2

1) Oxford university building and British flag



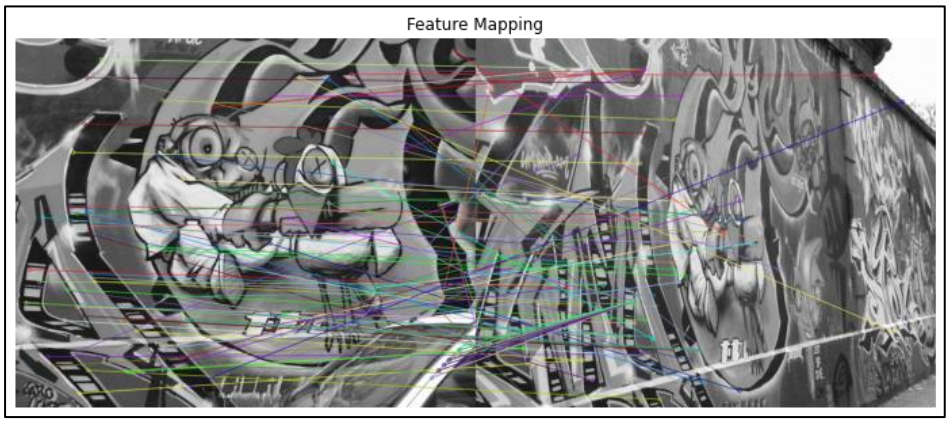
2) Sigiriya and Sri Lanka flag



3) Galle Face beach and Sri Lanka flag

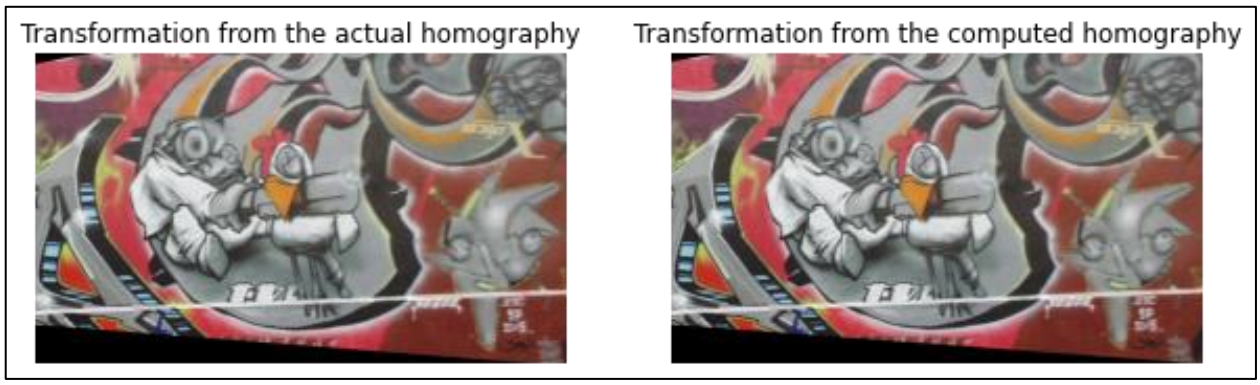
Question 3

a) SIFT feature mapping



b) Homography computation using the RANSAC algorithm

The number of good sift feature matches between image 1 and image 5 were not sufficient due to the significant perspective difference between the images. Therefore, homographies between consecutive images were computed via the RANSAC algorithm and they were multiplied together to obtain the homography from image 1 to 5.



The transformation carried out by the computed homography is identical to the transformation carried out by the actual homography given in the dataset.

The two homography matrices are as follows,

- Actual Homography

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

- Computed Homography

$$\begin{pmatrix} 8.27212794\text{e}-11 & 6.71870717\text{e}-12 & 3.03233982\text{e}-08 \\ 2.90487388\text{e}-11 & 1.55010572\text{e}-10 & -2.96662588\text{e}-09 \\ 6.40661394\text{e}-14 & -7.10375518\text{e}-15 & 1.35034033\text{e}-10 \end{pmatrix}$$

The drastic difference in the two matrices is due to the λ variable that correspond to the z-dimension of a 3-dimensional scene. The computed homography after making the λ variable in the two matrices equivalent is as follows,

- Computed Homography after accounting for λ

$$\begin{pmatrix} 6.25446440\text{e}-01 & 5.07994014\text{e}-02 & 2.29271858\text{e}+02 \\ 2.19634300\text{e}-01 & 1.17201779\text{e}+00 & -2.24303300\text{e}+01 \\ 4.84396990\text{e}-04 & -5.37107068\text{e}-05 & 1.02097738\text{e}+00 \end{pmatrix}$$

The sum of squared differences of the computed homography (after λ adjustment) and the actual homography is,

$$= 62.786$$

The code for general homography calculation is as follows,

```
def get_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)
    H = eigen_vectors[:, np.argmin(eigen_values)]
    H = H.reshape(3, -1)

    return H
```

The code for the calculating the homography from image 5 to 1 is as follows,

```
path = r"Images/graf/"
H_db_inv = []

for i in range(1, 5):
    img1_path = path + "img" + str(i) + ".ppm"
    img2_path = path + "img" + str(i+1) + ".ppm"

    H, count, count_db, best_fit_X_inliers, best_fit_Y_inliers = RANSAC(img1_path, img2_path, 1, 20, 10000)

    H_db_inv.append(np.linalg.inv(H))

H5to1 = H_db_inv[-1]

for i in range(len(H_db_inv)-2, -1, -1):
    H5to1 = H_db_inv[i] @ H5to1
```

c) Image Stitching