**EN2550 - Assignment 2 on Fitting and Alignment**

**Name**: C. J. Kurukulasuriya

**Index Number**: 190337X

**GitHub**: <https://github.com/chira99/image-processing-opencv-python.git>

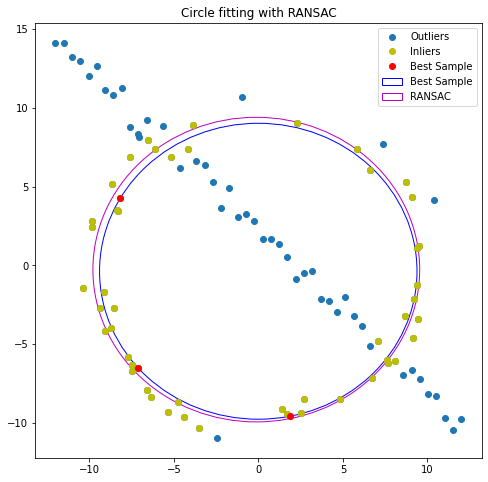
**Question 01**

1. RANSAC algorithm for circle estimation is implemented as follows.
2. def RANSAC\_circ(X):
4. e = 0.5     # outlier ratio
5. s = 3       # Number of points needed to create the estimated model
6. p = 0.99    # probability that at least 1 sample is free from outliers
7. t = 1.96 \* 10/16   # treshold
8. d = 50      # expected inlier count
10. iters = int(np.ceil(np.log(1-p)/np.log(1-(1-e)\*\*s)))
12. best\_inlier\_count = 0
13. best\_samples = None
14. best\_fit\_inliers = None
16. for \_ in range(iters):
17. # Choose 3 distinct points from dataset
18. [p1, p2, p3] = np.random.choice(len(X), size=3, replace=False)
19. [p1, p2, p3] = X[p1, :], X[p2, :], X[p3, :]
21. # Get circle through the 3 points
22. f, g, r = getCircle(p1, p2, p3)
24. if r == None:
25. continue
27. inlier\_count, inliers = getInlierCount(f, g, r, X, t)
29. if inlier\_count > best\_inlier\_count:
30. best\_inlier\_count = inlier\_count
31. best\_fit\_inliers = inliers
32. best\_samples = [p1, p2, p3]
33. best\_fit\_circle = [f, g, r]
35. if best\_inlier\_count < d:
36. # Repeat RANSAC if no model found
37. RANSAC\_circ(X)
39. ransac\_circle = bestFitCircle(best\_fit\_inliers) # returns f,g,r
41. return ransac\_circle, best\_fit\_circle, best\_samples, best\_fit\_inliers

Parameters of the algorithm:

* The minimum number of points needed to estimate the circle, **s = 3**
* A threshold of **t = 1.96\*(r/16)** gives the required 95% probability of capturing all inliers since the dataset is corrupted by mean-zero variance-one gaussian noise. (r = 10)
* Consensus size, **d = 50**, since 50 points are inliers out of the given 100 dataset points.

1. The resulting circle fitting using RANSAC algorithm is as follows.



**Question 02**

1. Classical painting projected onto the wall of a display room.

****

1. Movie poster on billboard display.

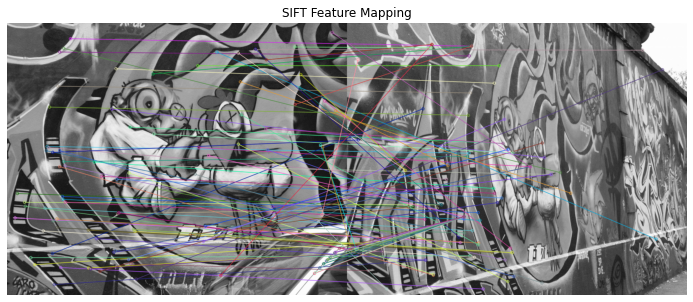


1. Sri Lankan flag projected/painted onto Sigiriya.



**Question 03**

* + 1. The SIFT feature mapping carried out between the two images are as follows.



* + 1. Due to the high perspective difference between image 1 and image 5 the number of “good” SIFT feature matches were insufficient to calculate a satisfactory homography transformation between them. Therefore, an intermediate homography was calculated using image 4 and by composition, a satisfactory homography was obtained for image 1 to image 5. The relevant code for calculating the homography is as follows.

1. def get\_homography(X, Y):
2. A = []
3. zeros = np.array([0,0,0])
5. # create matrix A
6. for i in range(4):
7. A.append(np.hstack((X[i, :], zeros, (-1\*Y.T[0, i]\*X[i,:]))))
8. A.append(np.hstack((zeros, X[i, :], (-1\*Y.T[1, i]\*X[i,:]))))
10. A = np.array(A).squeeze().astype(np.float64)
12. # find the eigen vector H corresponding to the smallest eigen value
13. eigen\_values, eigen\_vectors = np.linalg.eig(A.T @ A)
14. col\_idx = np.argmin(eigen\_values)
15. H = eigen\_vectors[:, col\_idx]
17. # rearrange H to obtain the Homography transformation matrix
18. H = H.reshape(3, -1)
20. return H
21. # Calculate homography in two parts
22. H1to4 = RANSAC("Images/graf/img1.ppm", "Images/graf/img4.ppm", 1, 20, 10000)
23. H4to5 = RANSAC("Images/graf/img4.ppm", "Images/graf/img5.ppm", 1, 20, 10000)
24. H1to5 = H4to5 @ H1to4

The transformations carried out using the computed homography and the actual homography have a high resemblance, as shown below

A picture containing text

Description automatically generated

The two homography matrices are given below, and their Sum Squared Difference, SSD = 10.08.

• Actual Homography

• Computed Homography after accounting for λ difference

* + 1. The image 1 stitched onto image 5 using the homography calculated is as follows.

