# Fitting and Alignment

## Assignment 2

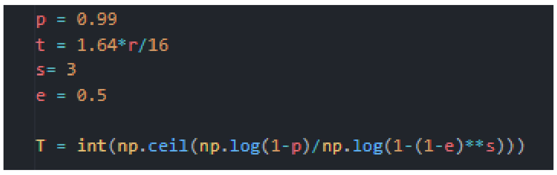
Biyon Fernando - 190178J

March 1, 2022

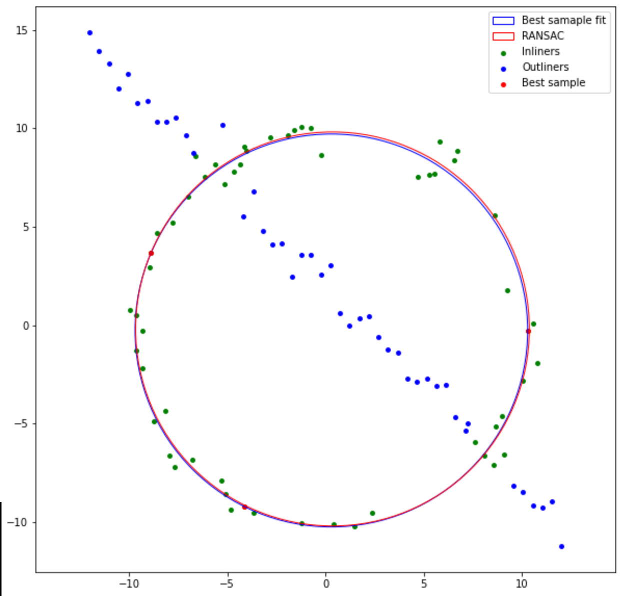
GitHub Link

### Question 1

To get the best fitting circle first we have to find an inlier point set to fit the curve. As we are fitting a circle at least we need three points to fit the circle. We can calculate the number of times the algorithm has to randomly select 3 points from the data set to achieve the targeted number of inliers as below. As this gave us 35 as result other than setting a threshold value for the number of inlier points we can use iteration where point count becomes maximum to fit the curve. Finding circle for chosen 3 points is



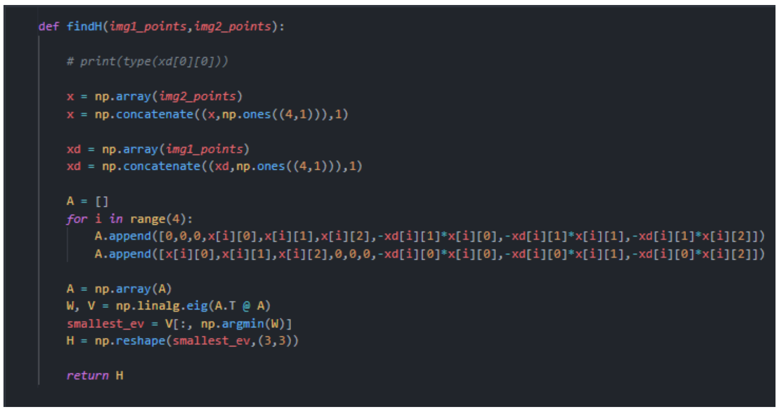
```
p = 0.99
t = 1.64*r/16
s= 3
e = 0.5

T = int(np.ceil(np.log(1-p)/np.log(1-(1-e)**s)))
```



Finding the circle for chosen 3 points is calculated using geometry. Some corner cases are handled in the algorithm omit extremely high and low values for the radius of the circle. After estimating inlier points fitting the best circle to data points is calculated according to an algebraic method which is given on the SciPy website. Code for the question can be found in the GitHub repository. The results of the implementation are bellowed.

| Results | Actual value | Estimated value |
|---|---|---|
| Radius of circle | 10.0 | 10.26998 |
| X coordinate of center | 0.0 | 0.432139 |
| Y coordinate of center | 0.0 | -0.04015 |

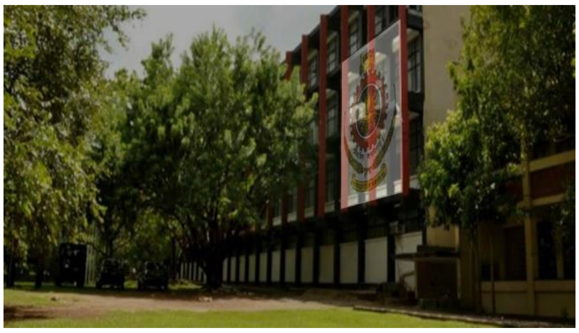
## Question 2

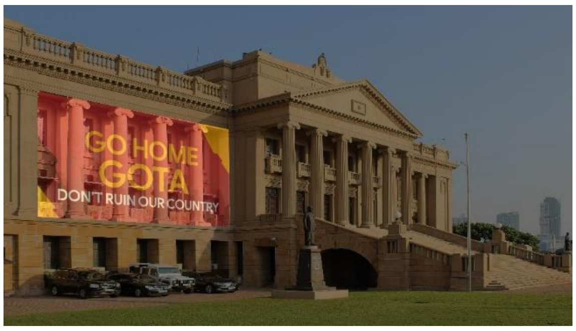


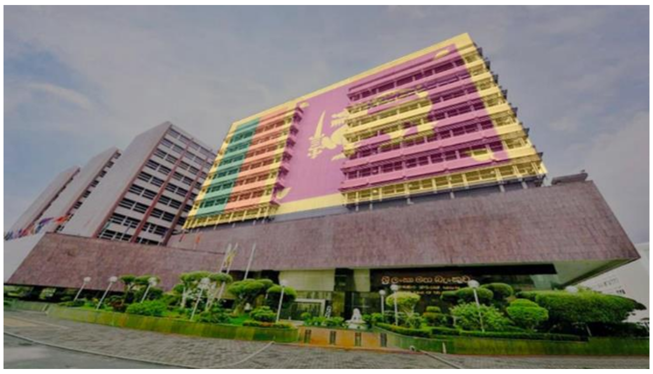
```python
def findH(img1_points,img2_points):

    # print(type(xd[0][0]))

    x = np.array(img2_points)
    x = np.concatenate((x,np.ones((4,1))),1)

    xd = np.array(img1_points)
    xd = np.concatenate((xd,np.ones((4,1))),1)

    A = []
    for i in range(4):
        A.append([0,0,0,x[i][0],x[i][1],x[i][2],-xd[i][1]*x[i][0],-xd[i][1]*x[i][1],-xd[i][1]*x[i][2]])
        A.append([x[i][0],x[i][1],x[i][2],0,0,0,-xd[i][0]*x[i][0],-xd[i][0]*x[i][1],-xd[i][0]*x[i][2]])

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

    return H
```

To get the user-selected points to align the second image left button mouse-clicking event is used. After the user click, 4 points on the first image algorithm uses those points to calculate the homography needed to transform the second image conners onto those points. Calculating holography is achieved from the code given below..

OpenCV function cv.warpPerspective can use to transform the second image. Here we can give the size of the first image to the resultant image so the new image can fit the first image. Then we can combine both images using the cv.addWeighted function in OpenCV with weights of 50% for both. The resultant images are shown below.





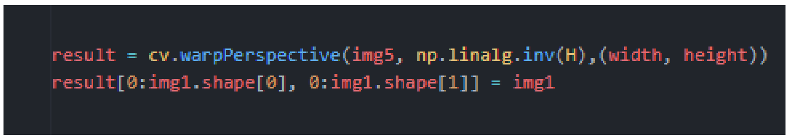


## Question 3

After reading both images we can get sift features of both images using libraries in OpenCV. Using these features matching points are calculated using BFMatcher.knnMatch() to get the best matches. In this, we will take k=2 so that we can apply the ratio test explained by D.Lowe in his paper to extract good matches.

Then we are applying RANSC to get the best aligning homography for these two images. In each iteration, we choose 4 points randomly from the good matches list and calculate homography using the code in question 2. Then we can transform matching points indexes of the second image to the first image using that homography matrix and calculate the distance between the first image's original point and the transformed ones. Points within a distance of 1 pixel are selected as inliers for each iteration. Then we can select the homograph which gives the maximum number of inliers. A set of inliers given by that homography transform can be used to get the best fitting homographic transform for two images.

After giving the cv.warpPerspective to the second image with the best-fitting homograph we can stitch two images as given in the code..



```
result = cv.warpPerspective(img5, np.linalg.inv(H),(width, height))
result[0:img1.shape[0], 0:img1.shape[1]] = img1
```

But when if we obtain homography for img1.ppm and img5.ppm most of the matches found from the BFMatcher.knnMatch() are garbage values and it does not contain proper inliers set to generate the best homography. Therefore we can get homography from img1.ppm to img4.ppm then from img4.ppm to img5.ppm and multiply them to get homography for img1.ppm to img5.ppm.

Results of stitching for img1.ppm and img5.ppm has given bellow