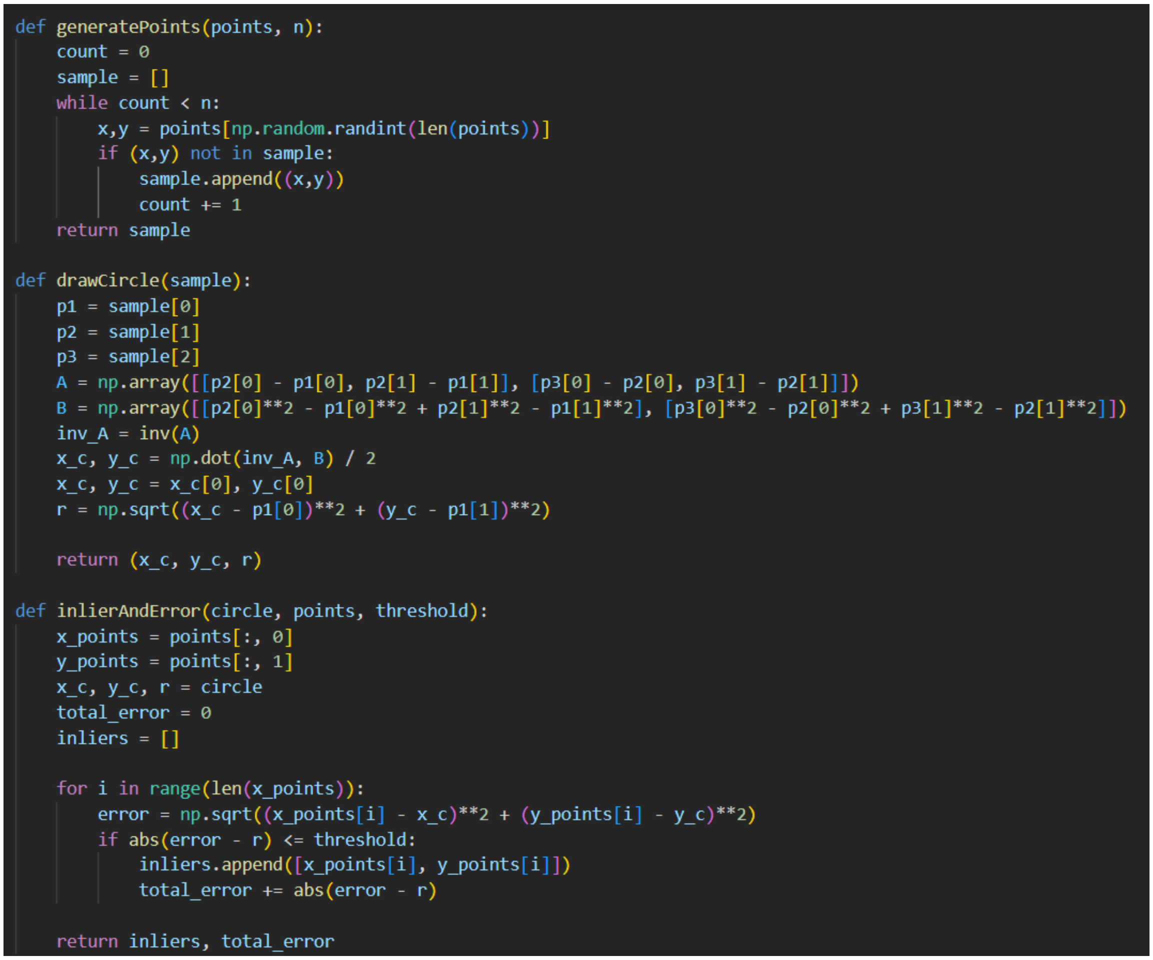
NAME: KRISHNAKANTH N.

INDEX NO: 190323C

Question 1

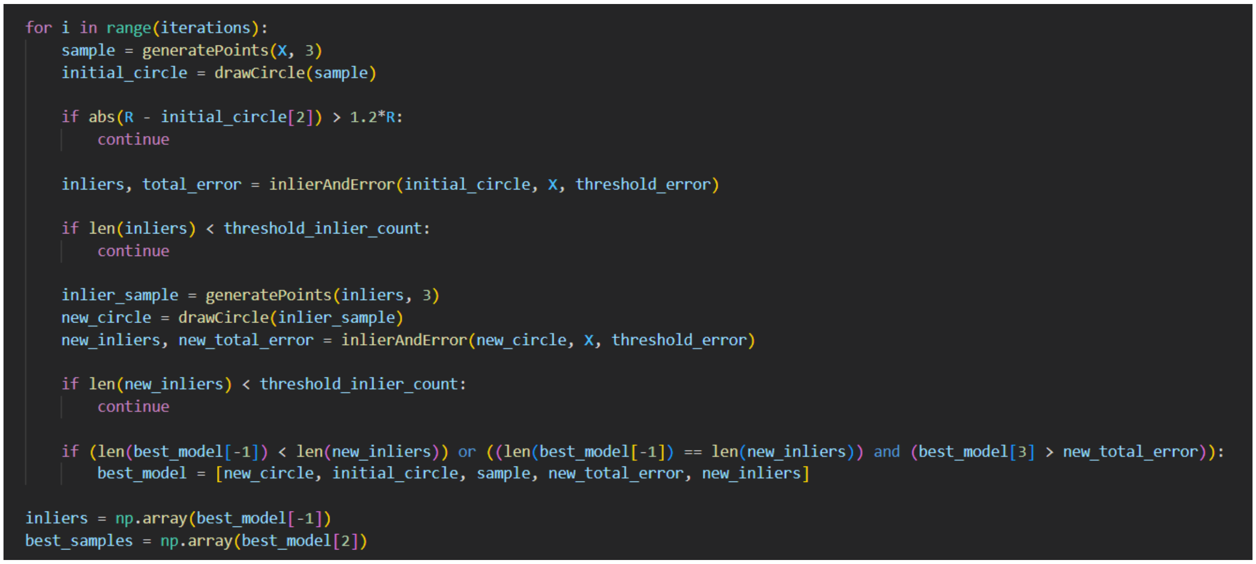
The algorithm as follows

1. Select 3 points randomly and draw a circle through them.
2. Observe the number of inliers within the given threshold and go to step 1 if the number of inliers less than the given threshold.
3. If the number of inliers calculated in step 2 is greater than the given threshold go to next step.
4. Select 3 random points among the inliers.
5. Draw a circle passing through the new circle and observe the number of inliers. If the number of inliers less than the given threshold go to step 1.
6. If the number of inliers calculated in step 5 is greater than the given threshold go to next step.
7. If the number of inliers in the current model is greater than the previously stored model, make this one as the best model.
8. If the number of inliers in the current model is equal to the previously stored model the one with the less threshold error is considered as the best model.
9. Continue these steps for a given number of times.

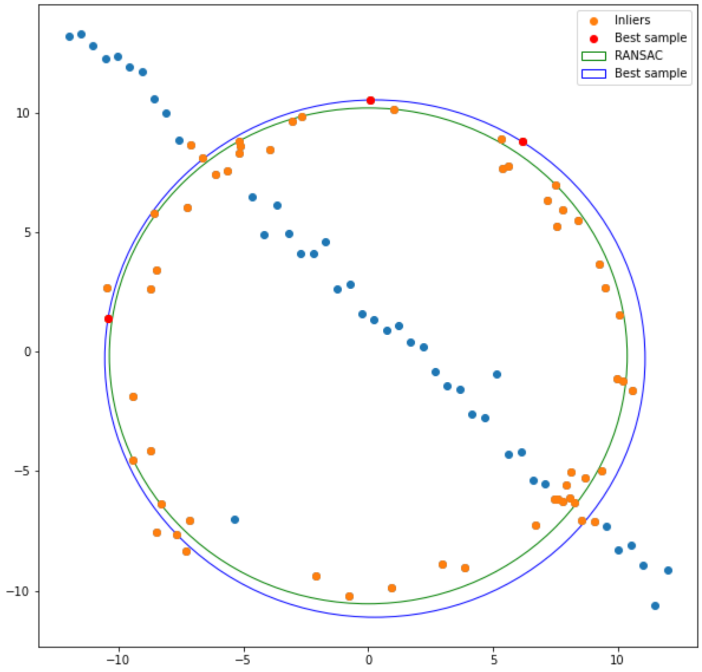


```
def generatePoints(points, n):
    count = 0
    sample = []
    while count < n:
        x,y = points[np.random.randint(len(points))]
        if (x,y) not in sample:
            sample.append((x,y))
            count += 1
    return sample

def drawCircle(sample):
    p1 = sample[0]
    p2 = sample[1]
    p3 = sample[2]
    A = np.array([[p2[0] - p1[0], p2[1] - p1[1]], [p3[0] - p2[0], p3[1] - p2[1]]])
    B = np.array([[p2[0]**2 - p1[0]**2 + p2[1]**2 - p1[1]**2], [p3[0]**2 - p2[0]**2 + p3[1]**2 - p2[1]**2]])
    inv_A = inv(A)
    x_c, y_c = np.dot(inv_A, B) / 2
    x_c, y_c = x_c[0], y_c[0]
    r = np.sqrt((x_c - p1[0])**2 + (y_c - p1[1])**2)

    return (x_c, y_c, r)

def inlierAndError(circle, points, threshold):
    x_points = points[:, 0]
    y_points = points[:, 1]
    x_c, y_c, r = circle
    total_error = 0
    inliers = []

    for i in range(len(x_points)):
        error = np.sqrt((x_points[i] - x_c)**2 + (y_points[i] - y_c)**2)
        if abs(error - r) <= threshold:
            inliers.append([x_points[i], y_points[i]])
            total_error += abs(error - r)

    return inliers, total_error
```

An additional if condition is written to avoid the case where a bigger circle is drawn taking samples from the line with satisfying the condition of maximum inliers within the threshold.



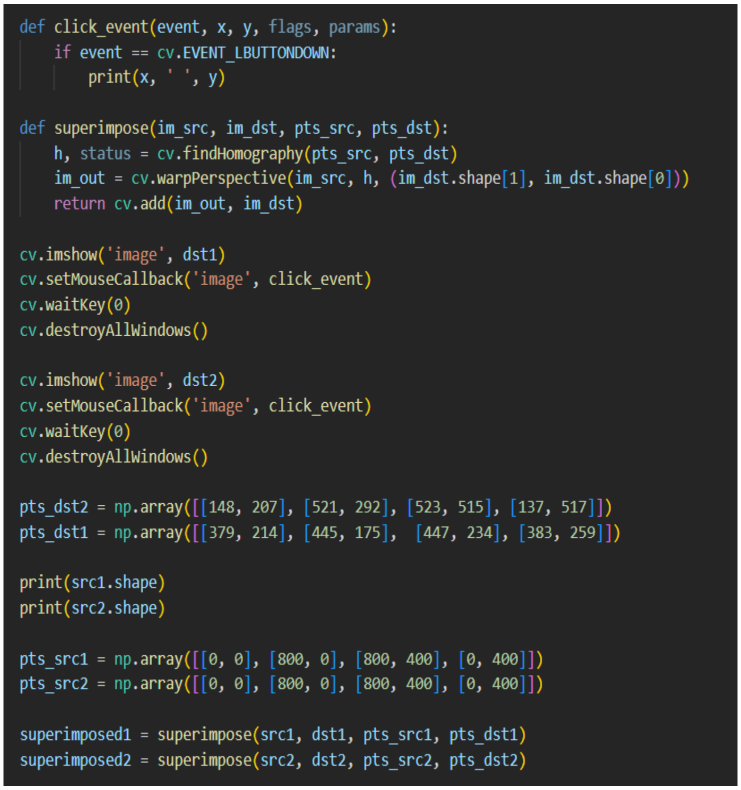
```
for i in range(iterations):
    sample = generatePoints(X, 3)
    initial_circle = drawCircle(sample)

    if abs(R - initial_circle[2]) > 1.2*R:
        continue

    inliers, total_error = inlierAndError(initial_circle, X, threshold_error)

    if len(inliers) < threshold_inlier_count:
        continue

    inlier_sample = generatePoints(inliers, 3)
    new_circle = drawCircle(inlier_sample)
    new_inliers, new_total_error = inlierAndError(new_circle, X, threshold_error)

    if len(new_inliers) < threshold_inlier_count:
        continue

    if (len(best_model[-1]) < len(new_inliers)) or ((len(best_model[-1]) == len(new_inliers)) and (best_model[3] > new_total_error)):
        best_model = [new_circle, initial_circle, sample, new_total_error, new_inliers]

inliers = np.array(best_model[-1])
best_samples = np.array(best_model[2])
```



Question 2

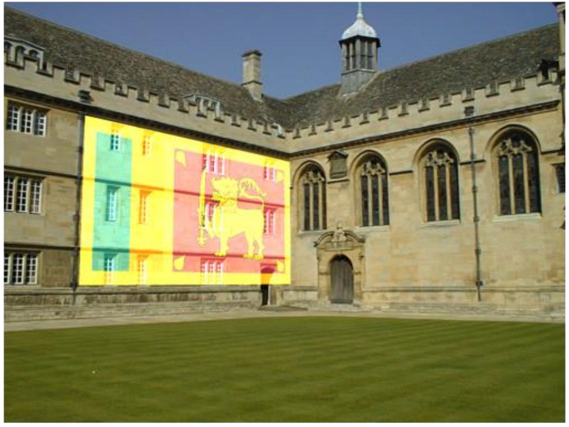
OpenCV default function ***cv.setMouseCallback()*** was used with a manual function ***click_event()*** is used to select vertices to place the source image on the destination image. The coordinate points of the destination image where the source image should be placed is written to a numpy array by observing the output after running the cell (it is printed in the output as written in the ***click_event()*** function).

The selected region in the destination image is a planar area. The superimpose done here doesn't align it according to the 3D nature in the figure. It is rather doing a linear transformation. If it placed in a non-planar area it would not adapt to it but gives a 2D perspective transformation.

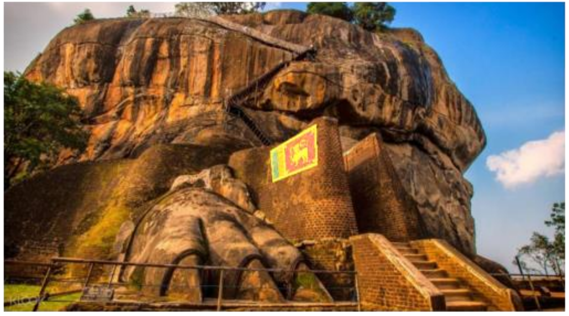


```python
def click_event(event, x, y, flags, params):
    if event == cv.EVENT_LBUTTONDOWN:
        print(x, ' ', y)

def superimpose(im_src, im_dst, pts_src, pts_dst):
    h, status = cv.findHomography(pts_src, pts_dst)
    im_out = cv.warpPerspective(im_src, h, (im_dst.shape[1], im_dst.shape[0]))
    return cv.add(im_out, im_dst)

cv.imshow('image', dst1)
cv.setMouseCallback('image', click_event)
cv.waitKey(0)
cv.destroyAllWindows()

cv.imshow('image', dst2)
cv.setMouseCallback('image', click_event)
cv.waitKey(0)
cv.destroyAllWindows()

pts_dst2 = np.array([[148, 207], [521, 292], [523, 515], [137, 517]])
pts_dst1 = np.array([[379, 214], [445, 175],  [447, 234], [383, 259]])

print(src1.shape)
print(src2.shape)

pts_src1 = np.array([[0, 0], [800, 0], [800, 400], [0, 400]])
pts_src2 = np.array([[0, 0], [800, 0], [800, 400], [0, 400]])

superimposed1 = superimpose(src1, dst1, pts_src1, pts_dst1)
superimposed2 = superimpose(src2, dst2, pts_src2, pts_dst2)
```

Superimposed Image 2



Sperimposed Image 1

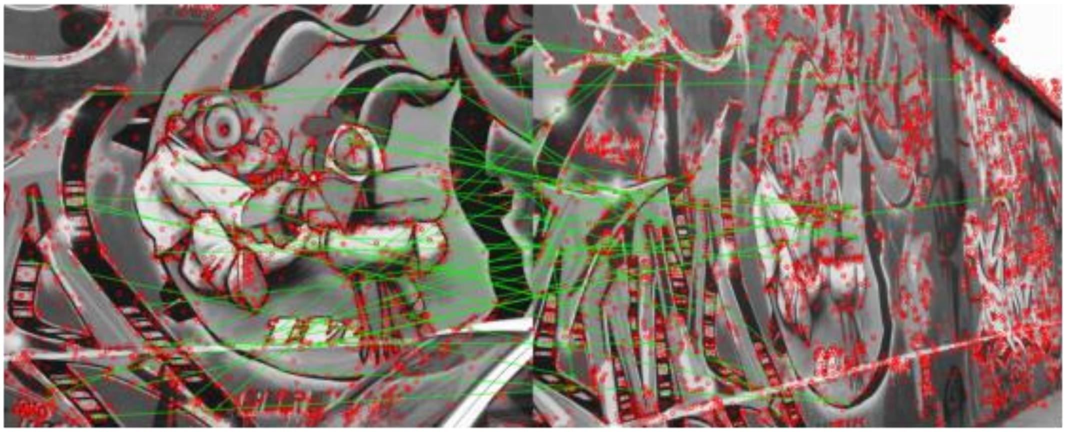


Question 3

a)

Flann based KNN matcher is used to match the features which were done as per the Low's ration test selecting 0.8 as the ratio.

Matched Image



b)

Since the direct matching between image 1 and image 5 was not as accurate as expected the matching is done by making a match between consecutive images in a chain formation starting from image 1 and image 2 and ending up at image 4 and image 5. This same as matching the features between image 1 and image 5.

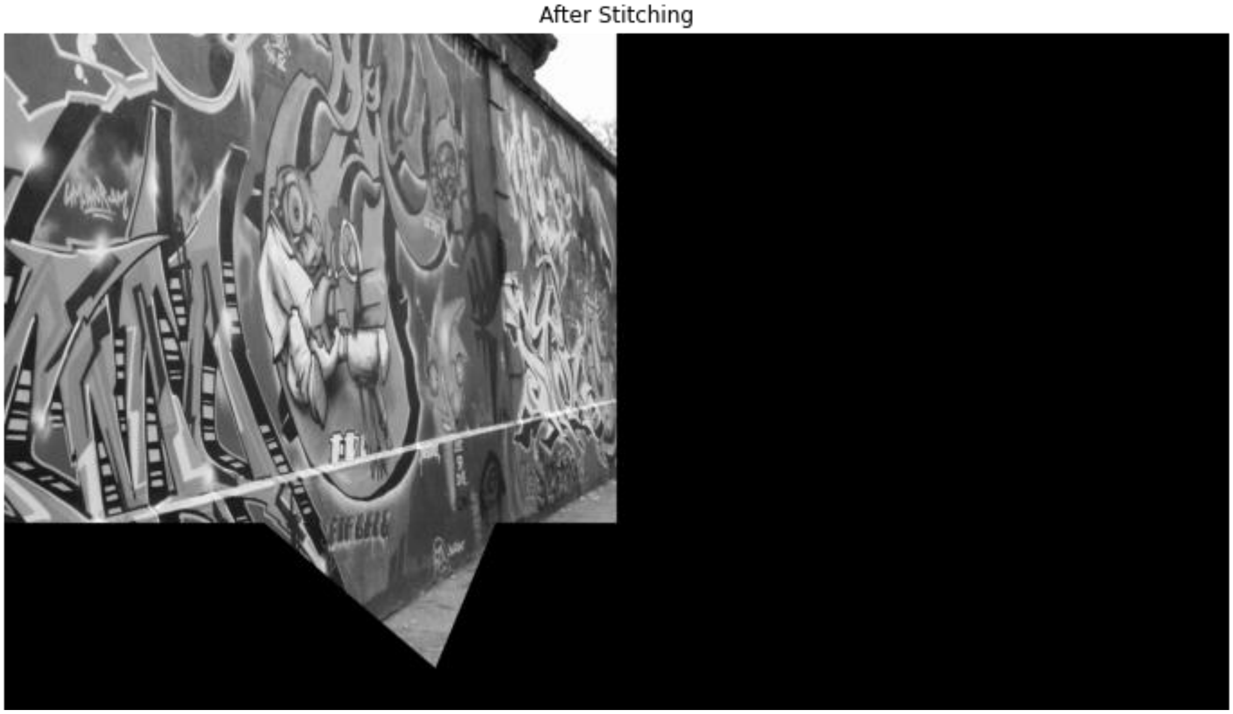
Obtained Homography:

$$\begin{matrix} 6.22594685e-01 & 6.32229252e-02 & 2.20364448e+02 \\ 2.20156353e-01 & 1.1638969e+00 & -2.57695451e+01 \\ 4.88791378e-04 & -2.87804193e-05 & 9.95040822e-01 \end{matrix}$$

Given Homography:

$$\begin{matrix} 6.254464e-01 & 5.7759174e-02 & 2.2201217e+02 \\ 2.2240536e-01 & 1.1652147e+00 & -2.5605611e+01 \\ 4.9212545e-04 & -3.6542424e-05 & 1.0000000e+00 \end{matrix}$$

c)



GitHub link: https://github.com/Krishnakanth-lab/EN2550_Assignment2.git