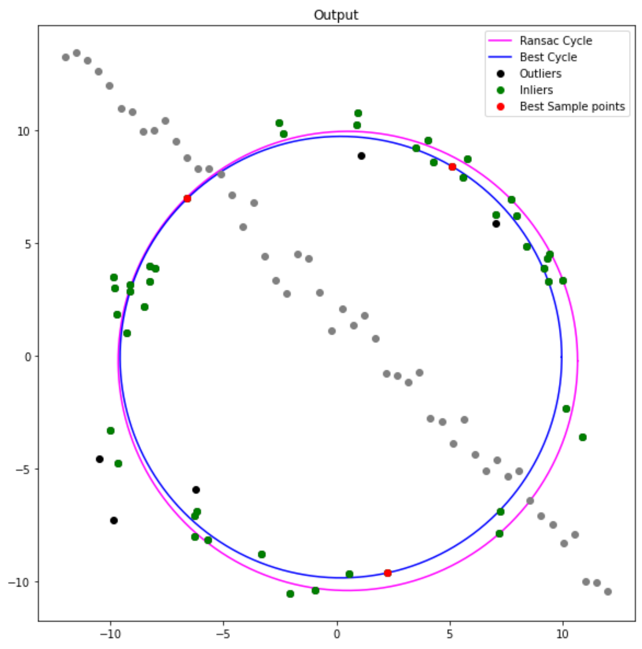
Ranathunga R.A.C.D.

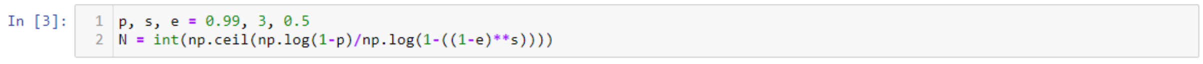
190501V – BME

GitHub link to the repository - [Assignment 02 (GitHub Repo)]

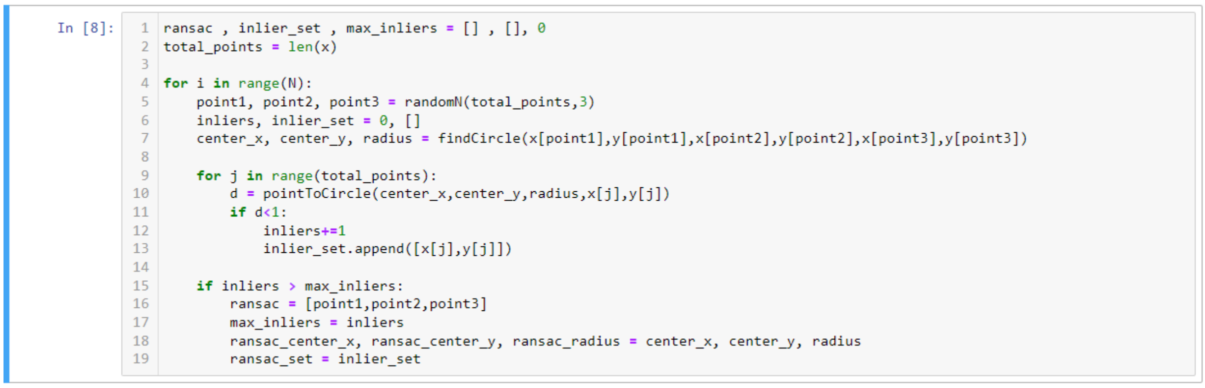
1) RANSAC algorithm and its results

To estimate a circle we need three points. First I select three random points, draw the circle, count the inliers and repeat. The best RANSAC circle in the below figure is the circle that had the most number of inliers up to that point. Then again running the above steps choosing inliers as the point cloud, I got the best circle. The best circle has less error than the RANSAC cycle.





```
In [3]:
p, s, e = 0.99, 3, 0.5
N = int(np.ceil(np.log(1-p)/np.log(1-((1-e)**s))))
```

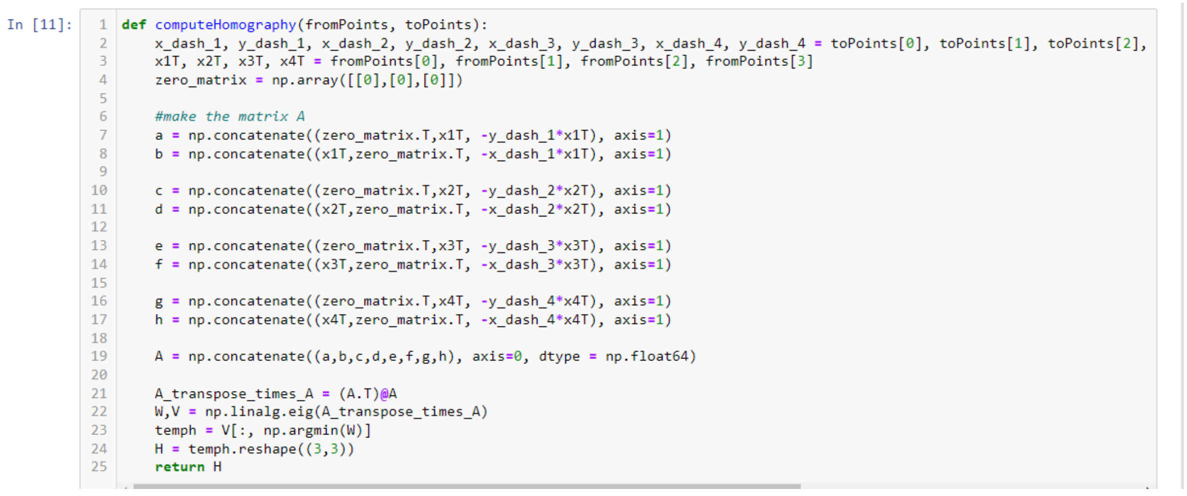


```
In [8]:
ransac , inlier_set , max_inliers = [] , [], 0
total_points = len(x)

for i in range(N):
    point1, point2, point3 = randomN(total_points,3)
    inliers, inlier_set = 0, []
    center_x, center_y, radius = findCircle(x[point1],y[point1],x[point2],y[point2],x[point3],y[point3])

    for j in range(total_points):
        d = pointToCircle(center_x,center_y,radius,x[j],y[j])
        if d<1:
            inliers+=1
            inlier_set.append([x[j],y[j]])

    if inliers > max_inliers:
        ransac = [point1,point2,point3]
        max_inliers = inliers
        ransac_center_x, ransac_center_y, ransac_radius = center_x, center_y, radius
        ransac_set = inlier_set
```

The code snippets here show some important steps in getting the RANSAC cycle. To view the full code with my own defined functions, please go to my GitHub repository that I have mentioned above.

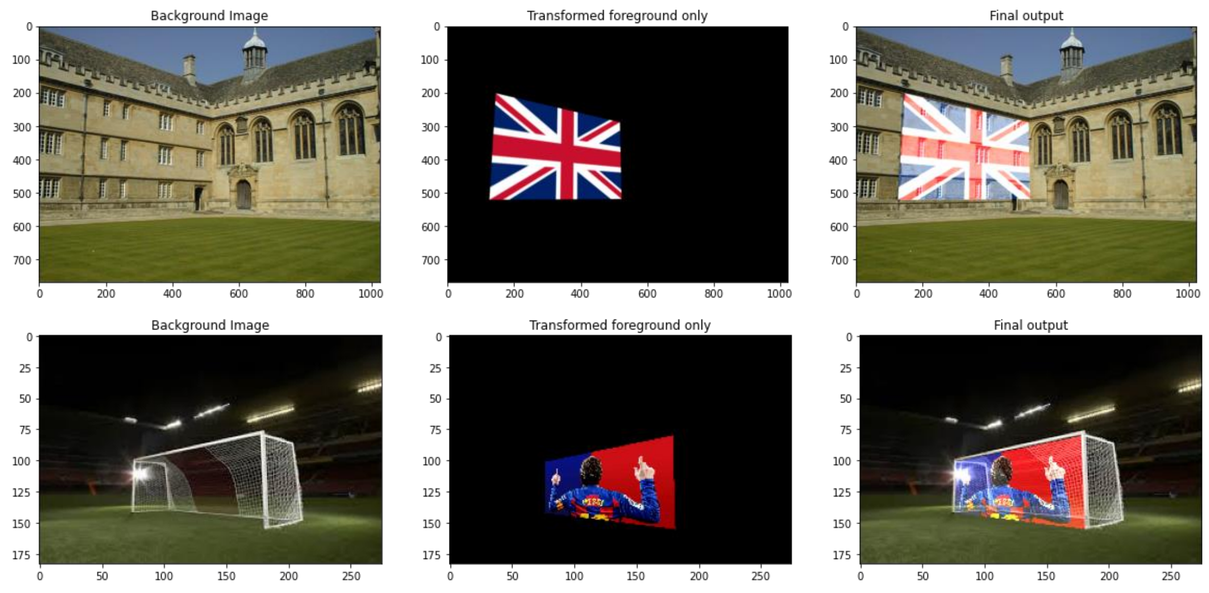
2) Compute homography and plot one image on top of another

Clicking four points on the image, I get their coordinates which are the destination locations. Then take the starting locations from the foreground image. Here, this function calculates the H matrix.



```
In [11]:
def computeHomography(fromPoints, toPoints):
    x_dash_1, y_dash_1, x_dash_2, y_dash_2, x_dash_3, y_dash_3, x_dash_4, y_dash_4 = toPoints[0], toPoints[1], toPoints[2],
    x1T, x2T, x3T, x4T = fromPoints[0], fromPoints[1], fromPoints[2], fromPoints[3]
    zero_matrix = np.array([[0],[0],[0]])

    #make the matrix A
    a = np.concatenate((zero_matrix.T,x1T, -y_dash_1*x1T), axis=1)
    b = np.concatenate((x1T,zero_matrix.T, -x_dash_1*x1T), axis=1)

    c = np.concatenate((zero_matrix.T,x2T, -y_dash_2*x2T), axis=1)
    d = np.concatenate((x2T,zero_matrix.T, -x_dash_2*x2T), axis=1)

    e = np.concatenate((zero_matrix.T,x3T, -y_dash_3*x3T), axis=1)
    f = np.concatenate((x3T,zero_matrix.T, -x_dash_3*x3T), axis=1)

    g = np.concatenate((zero_matrix.T,x4T, -y_dash_4*x4T), axis=1)
    h = np.concatenate((x4T,zero_matrix.T, -x_dash_4*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
```

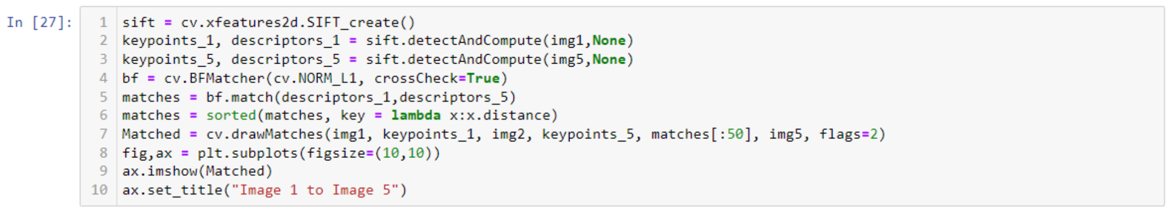
Now let’s see the results.



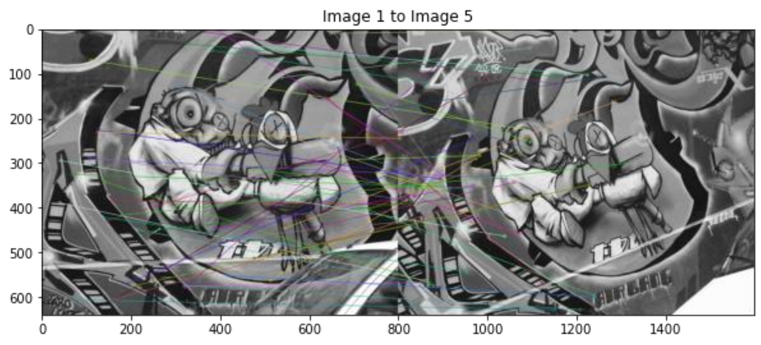
Here the transforms that are applied to the flag image and the image of Messi are different from each other. That means, their homography matrices are also different. The homography matrix depends on how we want to rotate, scale, and align the images. The original image used as Messi is on the right side of this text. We can see, how that transformation has changed the image.

3) Stitch image 1 to image 5 using Homography and RANSAC together.

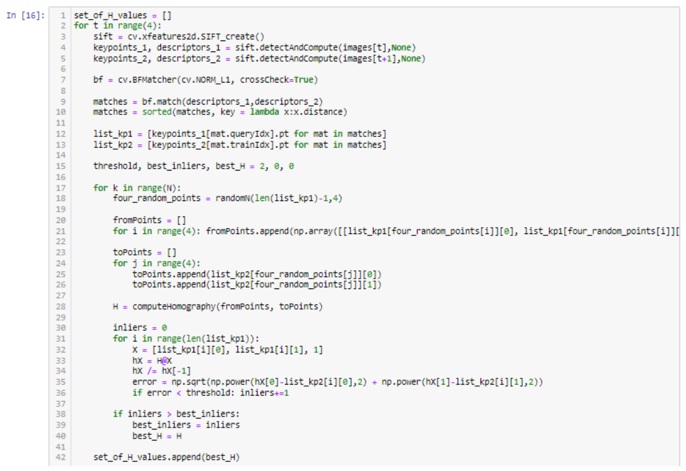
When we match the SIFT features of image 1 and image 5, in the result we can see that a considerable number of points are not correct. The result came from OpenCV's built-in function here.



```
In [27]:
sift = cv.xfeatures2d.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_5, descriptors_5 = sift.detectAndCompute(img5,None)
bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
matches = bf.match(descriptors_1,descriptors_5)
matches = sorted(matches, key = lambda x:x.distance)
Matched = cv.drawMatches(img1, keypoints_1, img2, keypoints_5, matches[:50], img5, flags=2)
fig,ax = plt.subplots(figsize=(10,10))
ax.imshow(Matched)
ax.set_title("Image 1 to Image 5")
```



Therefore, I computed, the homography matrices for image 1 to Image 2 and then from Image 2 to Image 3 and so on.



```
In [16]:
set_of_H_values = []
for t in range(4):
    sift = cv.xfeatures2d.SIFT_create()
    keypoints_1, descriptors_1 = sift.detectAndCompute(images[t],None)
    keypoints_2, descriptors_2 = sift.detectAndCompute(images[t+1],None)

    bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)

    matches = bf.match(descriptors_1,descriptors_2)
    matches = sorted(matches, key = lambda x:x.distance)

    list_kp1 = [keypoints_1[mat.queryIdx].pt for mat in matches]
    list_kp2 = [keypoints_2[mat.trainIdx].pt for mat in matches]

    threshold, best_inliers, best_H = 2, 0, 0

    for k in range(N):
        four_random_points = randomN(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]][

        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 = computeHomography(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

    set_of_H_values.append(best_H)
```

Img2_points = H[1 to 2] x Img1_points

Img3_points = H[2 to 3] x Img2_points

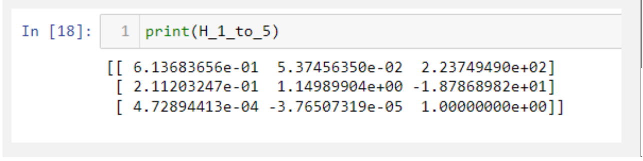
Therefore, we can say that,

Img5_points = (H[4 to 5] x H[3 to 4] x H[2 to 3] x H[1 o 2]) x Img1_points

Then the homography matrix from image 1 to image 5 is the multiplication of Homography matrices, 4 to 5 and 3 to 4 and 2 to 3 and finally 1 to 2.

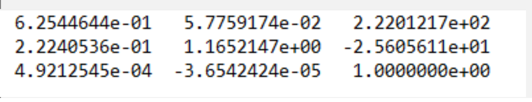
Let's compare the Homography matrix I got for image 1 to image 5 and the matrix is given on the website.

Homography matrix calculated by my code



```
In [18]:
print(H_1_to_5)

[[ 6.13683656e-01  5.37456350e-02  2.23749490e+02]
 [ 2.11203247e-01  1.14989904e+00 -1.87868982e+01]
 [ 4.72894413e-04 -3.76507319e-05  1.00000000e+00]]
```

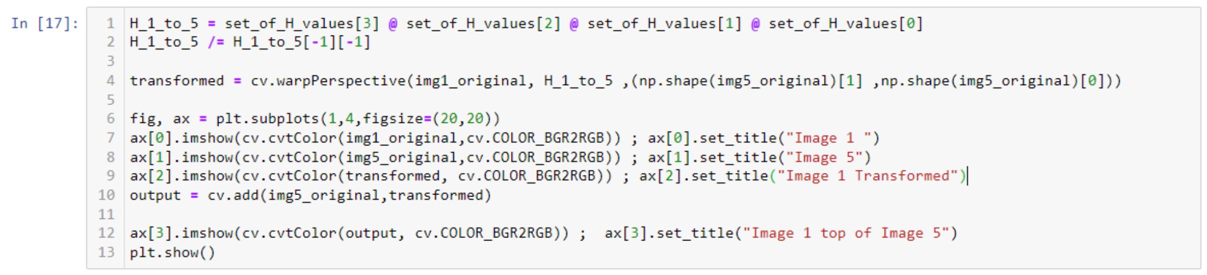
Homography matrix by the website



```
6.2544644e-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
```

We can see that the two homography matrices are so much close to each other. The homography matrix can be changed from one execution to another because it depends on the random points that are selected. But every time it gets values close to the above matrices.

Now let's see how the output is received from the homography matrix that I computed.



```
In [17]:
H_1_to_5 = set_of_H_values[3] @ set_of_H_values[2] @ set_of_H_values[1] @ set_of_H_values[0]
H_1_to_5 /= H_1_to_5[-1][-1]

transformed = cv.warpPerspective(img1_original, H_1_to_5 ,(np.shape(img5_original)[1] ,np.shape(img5_original)[0]))

fig, ax = plt.subplots(1,4,figsize=(20,20))
ax[0].imshow(cv.cvtColor(img1_original,cv.COLOR_BGR2RGB)) ; ax[0].set_title("Image 1 ")
ax[1].imshow(cv.cvtColor(img5_original,cv.COLOR_BGR2RGB)) ; ax[1].set_title("Image 5")
ax[2].imshow(cv.cvtColor(transformed, cv.COLOR_BGR2RGB)) ; ax[2].set_title("Image 1 Transformed")
output = cv.add(img5_original,transformed)

ax[3].imshow(cv.cvtColor(output, cv.COLOR_BGR2RGB)) ;  ax[3].set_title("Image 1 top of Image 5")
plt.show()
```

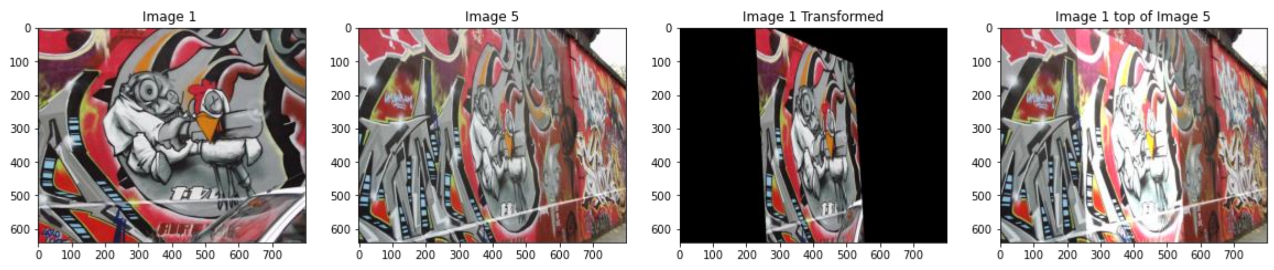
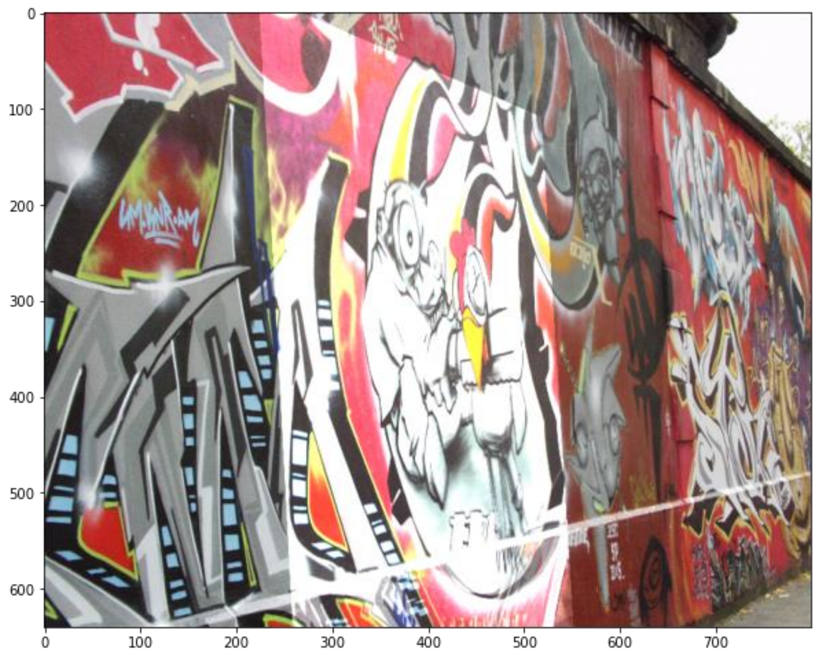


Image 1 is nicely transformed and stitched on Image 5. The homography result is fine.



Please refer the code from my GitHub repo for clear understanding of my works.