ASSIGNMENT 2

NAME: M.N.F.NIFLA

INDEX NO: 190413D

Question 1

In [ ]:
```
# determine a circle using 3 points (radius and center)
def circleRadius(b, c, d):
  temp = c[0]**2 + c[1]**2
  bc = (b[0]**2 + b[1]**2 - temp) / 2
  cd = (temp - d[0]**2 - d[1]**2) / 2
  det = (b[0] - c[0]) * (c[1] - d[1]) - (c[0] - d[0]) * (b[1] - c[1])
  if abs(det) < 1.0e-10:
    return None
  # Center of circle
  cx = (bc*(c[1] - d[1]) - cd*(b[1] - c[1])) / det
  cy = ((b[0] - c[0]) * cd - (c[0] - d[0]) * bc) / det
  radius = ((cx - b[0])**2 + (cy - b[1])**2)**.5
  return ((cx,cy) , radius)
```

In [ ]:
```
X_new = np.concatenate((X_circ, X_line),axis=0)
candidates = [] # circles after ransac and MAE

#RANSAC
maximum_iterations = 30
for iter in range(maximum_iterations):
  # 1.Select any 3 points
  one,two,three=np.random.choice(50),np.random.choice(50),np.random.choice(50)
  first,second,third= X_circ[one],X_circ[two],X_circ[three]
  best_circles = [] # from ransac
  # 2.Finding the circle that passes through these three points
  center , radius = circleRadius(first,second, third)
  # Define threshold distance to find inliers
  threshold_distance = 1
  # 3.Finding inliers
  inliers = []
  sq_distance = 0
  for i in range(100):
      sq_distance = (X_new[i][0] - center[0])**2 + (X_new[i][1] - center[1])**2
      sq_distance = np.sqrt(sq_distance)
      if sq_distance > radius - threshold_distance and sq_distance < radius + threshold
        inliers.append(X_new[i])
  # 4. define a threshold for inlier count to distinguish circles(best)
  inlier_threshold = 40
  if len(inliers) > inlier_threshold:
      best_circles.append([center, radius])
      # 5. determine a new candidate circle using all the inlier points
      # Least Square Circle Fit
      x_bar = sum(inliers[i][0] for i in range(len(inliers))) /len(inliers)
      y_bar = sum(inliers[i][1] for i in range(len(inliers))) /len(inliers)

      u_i = X_circ[:,0] - x_bar
      v_i = X_circ[:,1] - y_bar
      s_u = sum(u_i)
      s_uu = sum(u_i ** 2)
      s_uuu = sum(u_i ** 3)
```

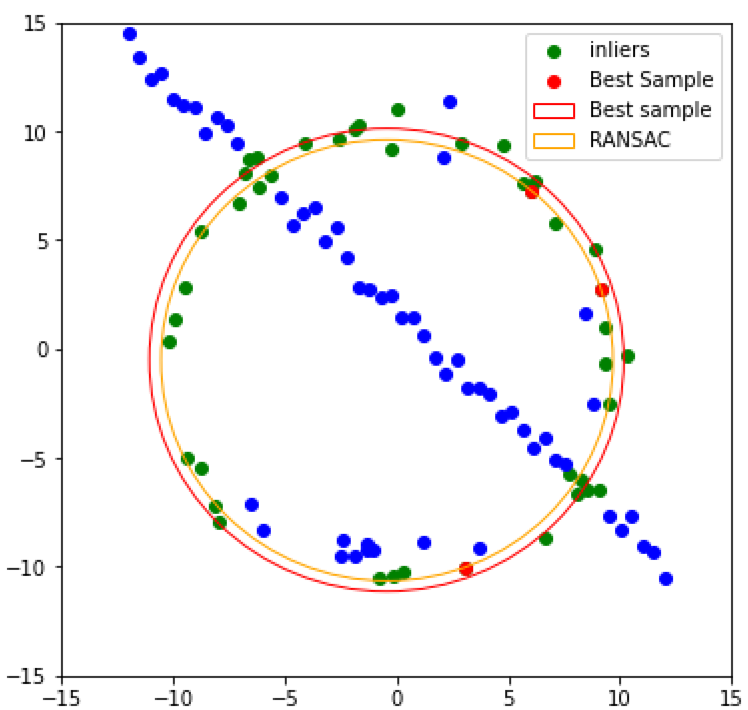
```
        s_v = sum(v_i)
        s_vv = sum(v_i ** 2)
        s_vvv = sum(v_i ** 3)
        s_uv = sum(u_i * v_i)
        s_uvv = sum(u_i * v_i * v_i)
        s_vuu = sum(v_i * u_i * u_i)
        u_c = 0.5 * ((s_uuu + s_uvv) * s_vv - (s_vvv + s_vuu) * s_uv) / (s_uu * s_vv - s_
        v_c = 0.5 * ((s_uuu + s_uvv) * s_uv - (s_vvv + s_vuu) * s_uu) / (s_uv ** 2 - s_uu
        x_c = u_c + x_bar # center
        y_c = v_c + y_bar # center
        Radius = np.sqrt(u_c**2 + v_c**2 + (s_uu + s_vv)/len(inliers))
        # 6.Find all new inliner points and new  outlier points for new candidate circle
        new_inliers=[]
        new_outliers=[]
        for i in range(100):
          sq_distance2 = (X_new[i][0] - x_c)**2 + (X_new[i][1] - y_c)**2
          sq_distance2 = np.sqrt(sq_distance2)

          if sq_distance2 > Radius - threshold_distance and sq_distance2 < Radius + thres
            new_inliers.append(list(X_new[i]))
          else: new_outliers.append(list(X_new[i]))
    #7.If the count of inlier points is less than threshold inlier count then skip this n
    # If the count exceeds threshold inlier count then move on to next step
        if len(new_inliers) > inlier_threshold:
      # 8. Calculate the mean absolute error for the new candidate circle using the new i
            MAE = sum(np.sqrt((np.array(new_inliers)[:,0]-x_c)**2 + (np.array(new_inliers
      # 9. Add this candidate circle to the shortlist along with count of inlier points a
            candidates.append([(x_c,y_c), Radius, len(new_inliers), MAE, new_inliers, new_
      # 10. Go back to the first step and repeat for max iterations number of times
  #11.When max iterations is completed, examine the shortlist of candidate circles and pi
  # If more than candidate circles with same inliner count then pick the candidate circle
```

In [ ]:

```
# get the best condidate circle with highest inliers
```

In [ ]:

```
import matplotlib.patches as mpatches
figure, axes = plt.subplots(figsize=(6,6))
```



Question 2

In [ ]:
```
def mouse_handler(event, x, y, flags, data) :
    if event == cv.EVENT_LBUTTONDOWN :
        cv.circle(data['im'], (x,y),3, (0,0,255), 5, 16)
        cv.imshow("Image", data['im'])
        if len(data['points']) < 4 :
            data['points'].append([x,y])
image = [('Images/Go_Home_Gota.jpg',"Images/Galle_Fort.jpg"),("Images/I_lv_u2.jpg","Ima
for (src, dst) in image:
    # Read source image.
    # Create a vector of source points.
    # Read destination image
    # Get four corners
    # Calculate Homography between source and destination points
    h, status = cv.findHomography(pts_src, pts_dst)
    # Warp source image
    im_temp = cv.warpPerspective(im_src, h, (im_dst.shape[1],im_dst.shape[0]))
    # Black out polygonal area in destination image.
    # Add warped source image to destination image.
    im_dst = im_dst + im_temp
```

There are three pairs of images and here is the non-technical rationale of them :D,

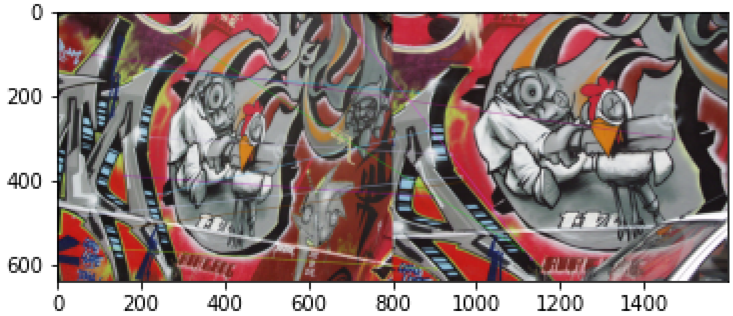
1) For the country and respect the people who are fighting for #GoHomeGota

2) For my love

3) Just for fun we took over the civil department under our custedy.

Question 3

In [ ]:
```
# read images
#sift
sift = cv.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)
#feature matching
bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)
```



Out[ ]: (<matplotlib.image.AxesImage at 0x1e609227ee0>, None)

In [ ]:
```
# find homography between images
good_kp_l = np.array([keypoints_1[m.queryIdx].pt for m in matches[10:20]])
good_kp_r = np.array([keypoints_2[m.trainIdx].pt for m in matches[10:20]])
H, masked = cv.findHomography(good_kp_r, good_kp_l, cv.RANSAC, 5.0)
```

In [ ]:
```
def warpTwoImages(img1, img2, H):
    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]
    pts1 = np.float32([[0, 0], [0, h1], [w1, h1], [w1, 0]]).reshape(-1, 1, 2)
    pts2 = np.float32([[0, 0], [0, h2], [w2, h2], [w2, 0]]).reshape(-1, 1, 2)
    pts2_ = cv.perspectiveTransform(pts2, H)
    pts = np.concatenate((pts1, pts2_), axis=0)
    [xmin, ymin] = np.int32(pts.min(axis=0).ravel() - 0.5)
    [xmax, ymax] = np.int32(pts.max(axis=0).ravel() + 0.5)
    t = [-xmin, -ymin]
    Ht = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]])  # translate
    result = cv.warpPerspective(img2, Ht@H, (xmax-xmin, ymax-ymin))
    result[t[1]:h1+t[1], t[0]:w1+t[0]] = img1
    return result
result = warpTwoImages(img1, img2, H)
```