EN2550 – Assignment 02

Name: Abeywansa C.N.
Index: 190005E

Question-1

(a) This question is about to estimate the circle using RANSAC coded by own. For that, the **"Circle_generator()**" function is defined that select the three (x, y) coordinates values of the given noisy points randomly and calculate the g, f ,c coefficients in every iteration, then return best of them and sample points, inliers.

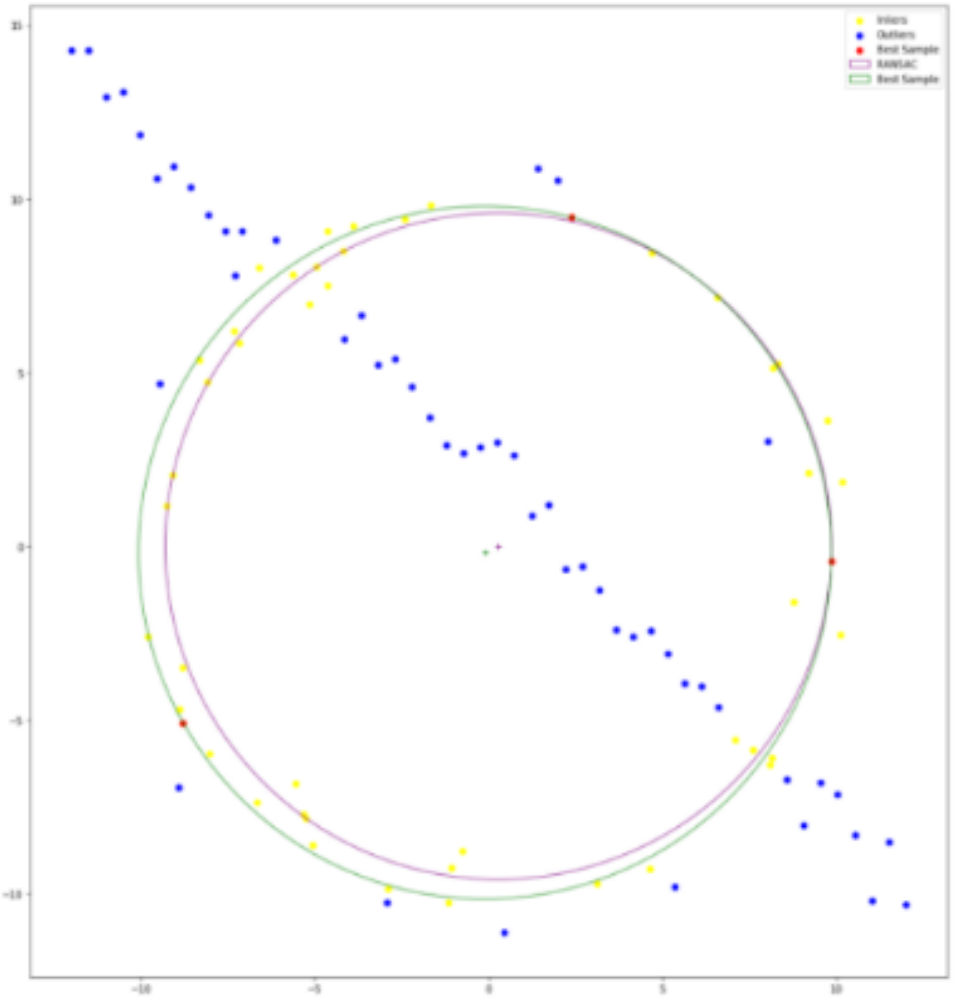
```
s = 3   #three points are used to fixed the model
t = 1       #treshold
e = 0.5     #relavant e value for 3 points
p = 0.99    #probality of at least one random sample is free from outliers
N = int (np.log(1-p) / np.log(1 -(1-e)**s)) #Number of samples(RANSAC equation)

ransac_coeff, ransac_sample, ransac_inlier_count = Circle_generator(X,N,t)
F,G,C = ransac_coeff[0], ransac_coeff[1], ransac_coeff[2]
R = np.sqrt(G**2 + F**2 -C)

# Finding Inliers and Outliers of the RANSAC Estimated Circle to identifying them seperately
Ins,Outs = [],[]
for point in X:
    d = abs(np.sqrt((point[0]+G)**2 + (point[1]+F)**2) - R) #Check where the point is,whether
                                                            # it is near to the circle or not
    if d < t:
        Ins.append(point)
    else:
        Outs.append(point)
#take the Transposes
Inliers = np.array(Ins).T
Outliers = np.array(Outs).T
Samples = ransac_sample.T

print("Number of Iterations =",N)
print("RANSAC Estimated circle Inlier Count =",ransac_inlier_count)
```

In the figure, first we must calculate number of samples N (iterations) that are needed for get accurate circle.

Using Circle_generator function, circle coefficients, samples and inliers are determined.

(b) The best estimate of the circle is obtained by the same process with increasing number of iterations (N=10000) that gives a more accurate estimation. Here inliers which are chosen with the threshold distance t (in this case, t is taken as 1), are in the range of radius ± t.

```
best_fitting_coeff, best_selected_pts, best_inlier_count = Circle_generator(Ins,10000,t)
best_F, best_G ,best_C = best_fitting_coeff[0], best_fitting_coeff[1], best_fitting_coeff[2]
best_R = np.sqrt(best_G**2 + best_F**2 -best_C)

b_samp = best_selected_pts.T

print("best fitting circle Inlier Count =",best_inlier_count)
✓ 7.3s
best fitting circle Inlier Count = 46
```



The final circle is drawn with the randomly chosen 3 points that give maximum number of inliers. In this question, the best fitting circle has 46 inlier count.

Question-2

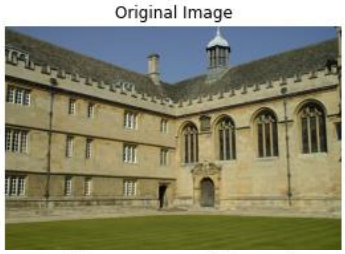
In this question, homography is used to warp an image and blend onto another image. Homography needs at least 4 points to do the transformations. So, points of the source image and destination points are necessary to find. Doing it with test and trial point indexes is hard. Therefore **"MouseHandling()"** function is defined for getting the indexes of the needed points. Here what happen is, OpenCV window of the destination image will be appear and by clicking the points that we need to blend the other image, that point indexes will be taken.

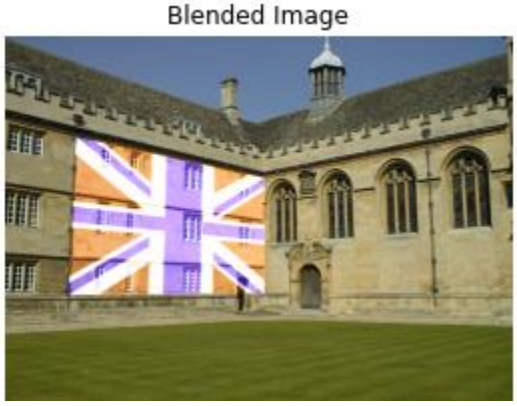
```
def MouseHandling(event,x,y,f,pram):
    global im_temp,pts_src
    if event== cv.EVENT_LBUTTONDOWN:
        cv.circle(im_temp ,(x,y),3,(0,255,255),3,cv.LINE_AA)
        cv.imshow("Image",im_temp)
        if len(pts_src)<4:
            pts_src = np.append(pts_src,[(x,y)],axis=0)
```

In this example, the **"findHomography ()"** built-in function is used to get homography matrix of the destination image (Architecture image). Warping and blending the source image (flag) are processed from the **"warPerspective ()"** and **"addWeighted ()"** respectively.

```
Homograpy_Matrix , status = cv.findHomography(pts_src, pts_dst)
transformed_flag = cv.warpPerspective(flag, np.linalg.inv(Homograpy_Matrix), (width, height))
blended_image = cv.addWeighted(architecture_img, 1, transformed_flag, 0.8, 0)
```

In "**addWeighted ()**" function, weighted the two images and combine. Here, for architecture image its 1(α) and for flag its 0.8(β). The 0 value is for gamma that is added to the final output image.

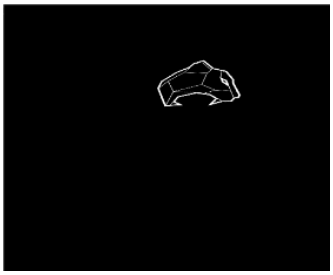




The blended image is the architectural image with flag superimposed. By changing the α, β and ϒ values, weights of the images (transparency, brightness, contrast...) will change.

Examples:

Here, the wrapped image (the tiger logo) has black background. Therefore, with ϒ=0, it will automatically remove. Unless it's not black there will be some colored background around the logo on the final image.



Question-3

```python
def sift_keypoint(src,dst):# detect features from the image
    sift = cv.SIFT_create(nOctaveLayers=3, contrastThreshold=0.1, edgeThreshold=28, sigma=1)
    keypoints_1, descriptors_1 = sift.detectAndCompute(src, None)
    keypoints_2, descriptors_2 = sift.detectAndCompute(dst, None)

    # matching the sift detected key points
    bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
    matches = bf.match(descriptors_1,descriptors_2)
    matches = sorted(matches, key = lambda x:x.distance)

    # Lists for keypoints
    list_kp1 = []
    list_kp2 = []

    for mat in matches:
        # Get the matching keypoints for each of the images as (x,y) coordinates
        img1_id = mat.queryId
        img2_id = mat.trainId

        # Get the coordinates
        (x1, y1) = keypoints_1[img1_id].pt
        (x2, y2) = keypoints_2[img2_id].pt

        # Append to relavant list
        list_kp1.append((int(x1+0.5), int(y1+0.5)))
        list_kp2.append((int(x2+0.5), int(y2+0.5)))
    return(list_kp1,list_kp2)
```

In this question, two Graffiti images should be stitched. To compute and match SIFT features between 2 images, "**cv.SIFT_create ()**" in-built function is used. Detect key points & descriptors of both images and matching them is done by below code part.

"**sift_keypoint()**" is a function defined to matching two images and get the key points lists separately as x, y coordinates ((x1, y1) from image1 and (x2, y2) from image2 matched with image1(x1,y1) coordinates).

In the part(b), to compute homography within RANSAC, we define "**ransacHomography()**" function. At first, take the 4 indexes and corresponding points. Then solve the homography matrix, solve the below matrix equation. (Here the P matrix is equal to A matrix)

$$PH = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y_2' & y_2y_2' & y_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x_3' & y_3x_3' & x_3' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y_3' & y_3y_3' & y_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x_4' & y_4x_4' & x_4' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y_4' & y_4y_4' & y_4' \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \\ h9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

```python
h = np.linalg.inv(A) @ np.array([[0,0,0,0,0,0,0,0,1]]).T
h = np.reshape(h,(3,3))#turn into a 3x3 matrix
X = np.vstack([src_keylist.T , np.ones( (1, len(src_keylist)) )])#add 1 as an element, vertically

transformed_keys = h @ X
z= np.array([transformed_keys[-1]]).T
transformed_keys = (transformed_keys[:2].T) / z
```

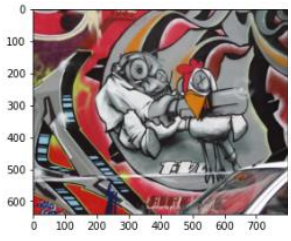
As in the question 1, RANSAC is used to determine the maxinliers and homography in the definition.

(c)The homography is generated using the **ransacHomography** function, is not almost same with its given homography. And stitch img1 onto img5 is done by "**warPerspective()**" function used in question 2. Even some values have bit or large difference between the 2 matrices, we can get better result.

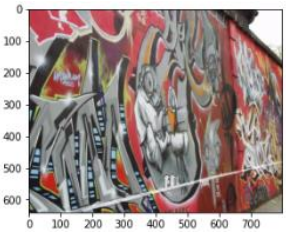
```
Dataset homography matrix
[[ 6.2544644e-01  5.7759174e-02  2.2201217e+02]
 [ 2.2240536e-01  1.1652147e+00 -2.5605611e+01]
 [ 4.9212545e-04 -3.6542424e-05  1.0000000e+00]]
```

```
10
maxinliers , generated homography matrix
[[ 6.12016279e-01  2.80653536e-02  2.24141030e+02]
 [ 2.26083985e-01  1.10681389e+00 -2.08710333e+01]
 [ 4.78018646e-04 -1.14815351e-04  1.00000000e+00]]
```

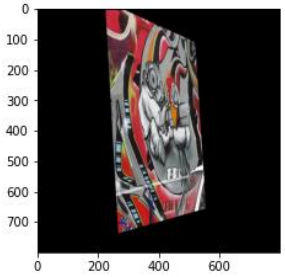
```python
warp= cv.warpPerspective(img1, G, (800,800))
result = cv.warpPerspective(img1, H, (800,800))

result[0:img5.shape[0],0:img5.shape[1]] = img5
```



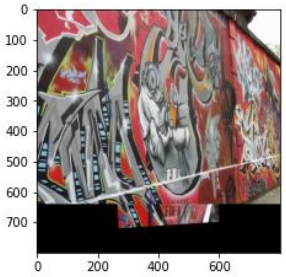
Img1



Img5



Warp image



Result image (stitch img1 onto img5)

GitHub link: https://github.com/chalindunis