190026T

AHMAED M.I.I

GitHub: https://github.com/ishrath99/S4_EN_2550/tree/main/Assignments/Assignment_2

EN 2550

Assignment 2 – Report

Question 1

Algorithm:

In addition to RANSAC algorithm, a new condition was passed to eliminate the bigger circles caused by points in the straight line.

Choosing parameters:

$s = 3$ $(minimum\ number\ points\ needed\ to\ fit\ the\ circle)$

$p = 0.99$ $(at\ least\ one\ random\ sample\ is\ free\ from\ outliers)$

$e = 0.5$ $(outlier\ ratio\ (50\ circle\ points, 50\ line\ points))$

$N = \log\frac{1-p}{\log(1-(1-e)^s)} \cong 35$ (Number of iterations)

$d = 50$ $(consensus\ set\ size\ (expexted\ inlier\ ratio = 0.5))$

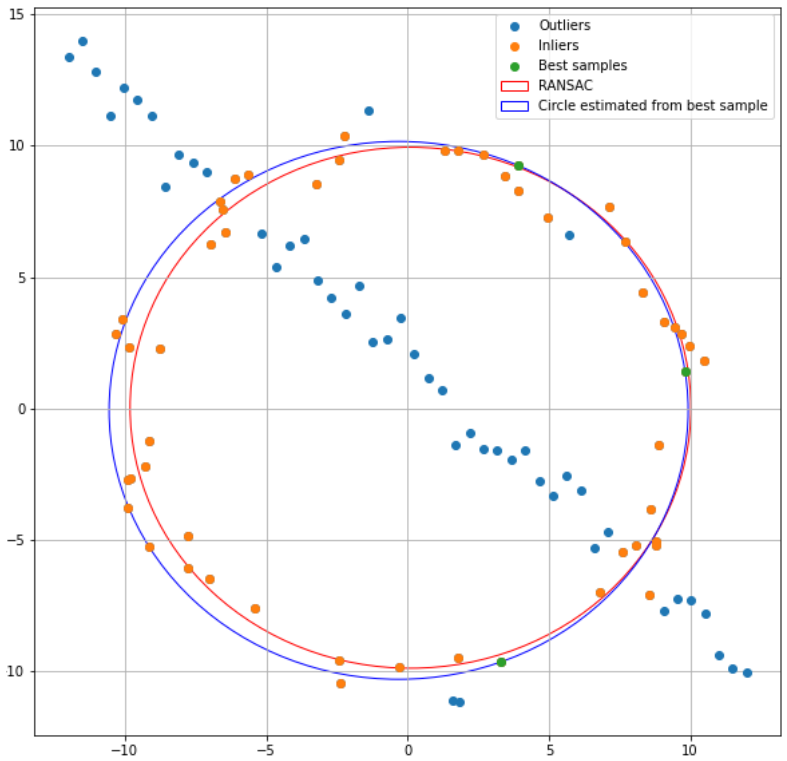
$t = 0.2 * radius$ $(distance\ threshold\ for\ inliers)$

Functions:

```
def getInliersAndDistance(circle, data, threshold):
    #return the inliers and the total absolute ditance by the inliers
    # parameters: circle parmaeters, all data points, inliers threshold
    x_data = data[:, 0]
    y_data = data[:, 1]

    a = circle[0]
    b = circle[1]
    r = circle[2]

    total_distance = 0
    inliers = []
    for i in range(len(x_data)):
        distance = np.sqrt((x_data[i] - a)**2 + (y_data[i] - b)**2)

        if abs(distance - r) <= threshold:
            inliers.append([x_data[i], y_data[i]])
            total_distance += abs(distance - r)

    return inliers, total_distance
```

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


def getCircleParams(sample):
    #return circle parameters from sample
    pt1 = sample[0]
    pt2 = sample[1]
    pt3 = sample[2]
    A = np.array([[pt2[0] - pt1[0], pt2[1] - pt1[1]],
                  [pt3[0] - pt2[0], pt3[1] - pt2[1]]])
    B = np.array([[pt2[0]**2 - pt1[0]**2 + pt2[1]**2 - pt1[1]**2],
                  [pt3[0]**2 - pt2[0]**2 + pt3[1]**2 - pt2[1]**2]])
    inv_A = inv(A)

    a, b = np.dot(inv_A, B) / 2
    a, b = a[0], b[0]
    r = np.sqrt((a - pt1[0])**2 + (b - pt1[1])**2)
    return (a, b, r)
```

Results:



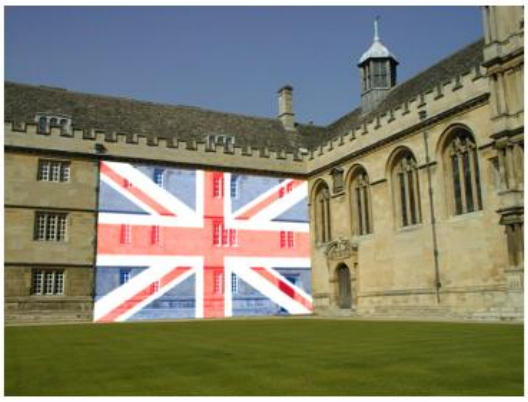
Question 2

Functions:

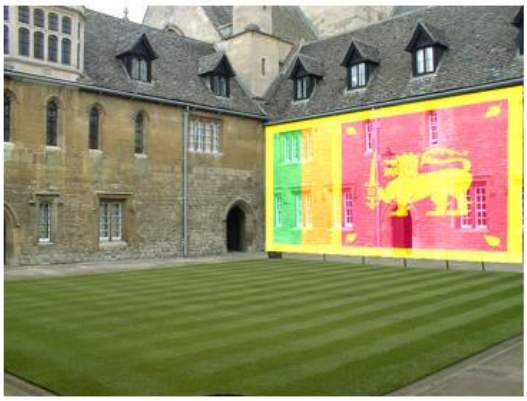
```
def click_event(event, x, y, flags, params):
    #function for getting co-ordinates by mouse click
    if event == cv.EVENT_LBUTTONDOWN:
        print(x, ' ', y)

def superimpose(im_src, im_dst, pts_src, pts_dst):
    #function to superimpose
    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.addWeighted(im_out, 0.3, im_dst, 0.7, 0.0)
```

Results:

superimposed 1



superimposed 2



Non-technical rationale of choice of images:

This kind of warping can be used for advertising and photo editing. When warping, we can consider the weightage of the source and the destination images. We must also consider the alignment and the points of contact of the images.

Question 3

(a)

Sift features are detected of the 2 images. Then features are matched using Flann based KNN matcher. Good matches are extracted as per Lowe's ration test. Ratio is selected to be **0.7**.

Since the good matching points between ***img1*** and ***img5*** are not enough to come up with a good homography. Homography is calculated associatively by starting from ***img1*** and ***img2*** and propagating the homography until ***img4*** and ***img5.***

(b)

Homography is calculated from the good matches. The function ***cv.findhomography()*** is used to obtain the homography with RANSAC implementation with the confidence of **0.95**.

Comparison:

Homography calculated:

$$\begin{matrix} 6.22389620e-01 & 4.90051516e-01 & 2.20882134ee+02 \\ 2.22416106e-01 & 1.14261758 & -2.34633970e+02 \\ 4.95739539e-04 & -6.21289797e-05 & 0.993217260 \end{matrix}$$

Homography given:

$$\begin{matrix} 6.2544644e-01 & 5.7759174e-02 & 2.2201217e+02 \\ 2.2240536e-01 & 1.1652147 & -2.56-5611e+01 \\ 4.9212545e-04 & -3.6542424e-05 & 1 \end{matrix}$$

Calculated homography is almost the same as the given homography in the dataset.

(c)

Following images shows the resulting image after stitching and the matching features between the images. The images are not perfectly stitched. The reason is, calculated homography is slightly different than the one in the dataset.

images stitched

features matched