DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION ENGINEERING
UNIVERSITY OF MORATUWA

EN2550 FUNDAMENTALS OF IMAGE PROCESSING AND MACHINE VISION

ASSIGNMENT 02 - Fitting and Alignment

Thanushan K. 190621M

This is submitted as a partial fulfilment for the module EN2550.

April 28, 2022

1 RANSAC Implementation

RANSAC is a general framework used for fitting models when there are outliers present in the images. In this question, we have to fit a circle using the RANSAC algorithm for a randomly generated set of points. The probability that atleast one sample is free from outliers (p) is set to be 0.99. The outlier ratio (e) was set to 0.5, considering the worst case. The threshold of selecting inliers was set to 1. The Samples N was calculated using the following equation.

$$N = \frac{\log(1-p)}{\log(1-(1-e)^{min_s})} \tag{1}$$

Three random samples were selected in each iteration. The circle was found corresponding to the point at each sample. The inliers are found using the threshold. The current set of inliers is compared with the set of inliers obtained from the previous sample and the set with maximum number of inliers is selected. The iteration goes upto N. Then The RANSAC algorithm is run by giving the final set of inliers obtained as the input inorder to obtain the best fit circle.
When considering the results, the best fit circle and the circle drawn using the sample points, which give the set with maximum inliers, are almost similar to each other. The code and Output are shown in figure 1 and 2 respectively.

```python
def RANSAC_fit(X):
    t = 1
    N = np.inf
    Bestfit = None
    Inliers = []
    sample_points =[]
    p = 0.99 #probability p, at least one random sample is free from outliers
    iterations = 0
    min_s = 3 #Minimum no of points required to find the equation of the circle
    while N > iterations:
        random_indices= np.random.randint(X.shape[0], size=min_s)
        point1, point2, point3 = X[random_indices]

        #Calculation of the center coordinates and the radius of the circle passing through the sample points
        coefficientMatrix = np.array([[point2[0] - point1[0], point2[1] - point1[1]], [point3[0] - point1[0], point3[1] - point1[1]]])
        constantMatrix = np.array([[point2[0]**2 - point1[0]**2 + point2[1]**2 - point1[1]**2], [point3[0]**2 - point1[0]**2 + point3[1]**2 - point1[1]**2]])
        invCoefficientMatrix = np.linalg.pinv(coefficientMatrix)

        center_x, center_y = (invCoefficientMatrix@constantMatrix) / 2
        center_x, center_y = center_x[0], center_y[0]
        r = np.sqrt((point1[0]- center_x)**2 + (point1[1] - center_y)**2)

        Inlier_test = []
        #Checking for inliers and appending them into the inlier set.
        for x, y in X:
            distance = np.sqrt((x - center_x)**2 + (y - center_y)**2)
            if (np.abs(distance - r) < t):
                Inlier_test.append([x,y])


        #Checking whether the number of current inliers is greater than the past inliers
        if (len(Inlier_test) > len(Inliers)):
            Bestfit = [center_x, center_y, r] #Getting the center coordinates and radius of the best fit circle
            Inliers = Inlier_test
            sample_points = [point1, point2, point3]  #Collecting the sample points of the best fit
            e = 1 - ((len(Inliers)/len(X))) #Outlier ratio 0.5 was taken for the worst case
            N = np.log(1-p)/np.log(1-(1-e)**min_s)  #calculation of samples
        iterations+=1
    return Bestfit, Inliers, sample_points
```

Figure 1: Code for RANSAC Algorithm

2 Image Warping and Blending by calculating a hormography

In this question we have to warp and blend one image to another. Initially we have to select 4 points (We get fout points because it is the minimum number of points required to calculate the homography matrix) in the background image and the image to be warped. This can be done by the setMouseCallback function in the opencv library. After getting the four coordinate points of both images, we create a matrix A which is the coeffiecient matrix of the homography vector. We calculate the homography matrix using equation 2. The eigen vector corresponding to the minimum eigen value is taken and constructed into $3X3$ matrix. Then the image is warped according to the corresponing homograpghy matrix and added to the

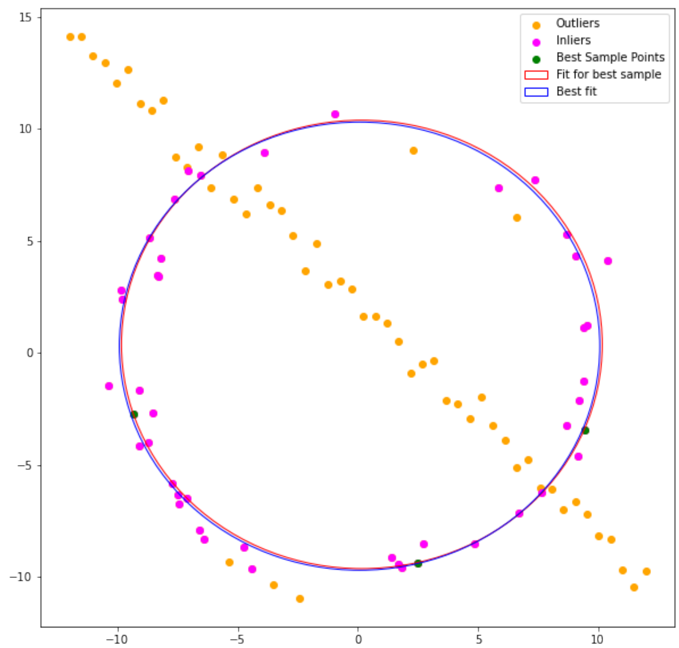

Figure 2: RANSAC Circle Fitting

```python
# mouse callback function
def mousePoints(event,x,y,flags,array):
    global counter
    point_matrix = array[0]
    if event == cv.EVENT_LBUTTONDOWN: # Left button mouse click event opencv
        cv.circle(array[1],(x,y),5,(255,0,0),-1)
        point_matrix[counter] = [x,y]
        counter += 1
```

(a) mousePoints function

```python
while True:
    cv.imshow("Archi Image", archiImage_copy)
    if counter == points:
        break
    if cv.waitKey(20) & 0xFF == 27:
        break
```

(b) Loop for selecting points

Figure 3: Mouse click algorithm

background image. Two other pairs of images were and the algorithm was tested. The essential codes are shown in figures 3 and 4. The results are shown in figure 5.

$$A.H = 0 \tag{2}$$

```python
#A is the coeffient matrix of the homography matrix such that A.H = 0. H is the 3 X 3 homography matrix stretched out as a 9 x 1 vector
A = np.array([[point_matrix_Flag[0][0], point_matrix_Flag[0][1], 1, 0, 0, 0, (-point_matrix_archiImage[0][0]*point_matrix_Flag[0][0]), (-point_matrix_archiImage[0][0]*point_matri
              [0, 0, 0, point_matrix_Flag[0][0], point_matrix_Flag[0][1], 1, (-point_matrix_archiImage[0][1]*point_matrix_Flag[0][0]), (-point_matrix_archiImage[0][1]*point_matri
              [point_matrix_Flag[1][0], point_matrix_Flag[1][1], 1, 0, 0, 0, (-point_matrix_archiImage[1][0]*point_matrix_Flag[1][0]), (-point_matrix_archiImage[1][0]*point_matri
              [0, 0, 0, point_matrix_Flag[1][0], point_matrix_Flag[1][1], 1, (-point_matrix_archiImage[1][1]*point_matrix_Flag[1][0]), (-point_matrix_archiImage[1][1]*point_matri
              [point_matrix_Flag[2][0], point_matrix_Flag[2][1], 1, 0, 0, 0, (-point_matrix_archiImage[2][0]*point_matrix_Flag[2][0]), (-point_matrix_archiImage[2][0]*point_matri
              [0, 0, 0, point_matrix_Flag[2][0], point_matrix_Flag[2][1], 1, (-point_matrix_archiImage[2][1]*point_matrix_Flag[2][0]), (-point_matrix_archiImage[2][1]*point_matri
              [point_matrix_Flag[3][0], point_matrix_Flag[3][1], 1, 0, 0, 0, (-point_matrix_archiImage[3][0]*point_matrix_Flag[3][0]), (-point_matrix_archiImage[3][0]*point_matri
              [0, 0, 0, point_matrix_Flag[3][0], point_matrix_Flag[3][1], 1, (-point_matrix_archiImage[3][1]*point_matrix_Flag[3][0]), (-point_matrix_archiImage[3][1]*point_matri

w, v = np.linalg.eig((A.T)@A)
V = v[:, np.argmin(w)]
V = np.reshape(V,(3,3))
#print(V)
Flag_warped_Image = cv.warpPerspective(Flag, V, (archiImage.shape[1], archiImage.shape[0]))
Final_Image = cv.addWeighted(archiImage, 1, Flag_warped_Image, 0.8, 0) # Adding the warped image to the background image
```

Figure 4: Homography calculation

3 Stitching Two Images by computing a hormography using RANSAC

Histogram equalization is also an intensity transformation that can be performed in an image such that the pixels are distributed all over the region that the pixels could take values. The result will be an improved

(a) Output 1 (b) Output 2 (c) Output 3

Figure 5: Outputs for Warping and blending

version of the image. In this question we have to use a manual function for equalizing the image. Initially, the histogram of the input image is obtained. Then the cdf value is calculated at each point. The calculated cdf value is multiplied by the maximum possible pixel value (L-1) and divided by the product of the dimensions of the input image. The transformation is given by equation 2.

$$s_k = \frac{L-1}{MN}\sum_{j=0}^{k} n_j \tag{3}$$

Code Files : All relevant codes and files can be found in **GitHub**.