University of Moratuwa

EN2550 - Fundamentals of Image Processing and Machine Vision

Assignment II

Name - Weerasinghe K.N.

Index No. - 190672T

GitHub link - https://github.com/KavinduWeerasinghe/Assignement-2

Question 1: *The code snippet in Listing 1 shows the code to generate a noisy point set X amounting to a circle and the code to estimate a circle—center and the radius—from a set of inliers in X.*

(*a*) *Estimate the circle using the RANSAC algorithm* (*must be coded on your own*).

(*b*) *Show in the same plot, the point set, the circle estimated from the sample leading to the best estimate, this sample of three points, inliers, and the best-fit circle. See Figure 1 for an example.*

Answer: The end goal of the code is to get the circle with a maximum number of inliers. To select a circle 3 points needs to be selected. In this code this procedure has been done through a for loop. After selecting 3 points, the circle (coordinates of the center and the radius) is created by get_circle function indicated in Code 1. Using that T_check function is used to check the error function. A value of 1 is used as the threshold and checked with the error value to identify the inliners. The function is coded with brute force. So that the circle with maximum inliners can be returned from the function. The Result is shown in Figure 1.

```python
def get_circle(x1,y1, x2,y2, x3,y3):
    x1y1 =x1**2 +y1**2
    x2y2 =x2**2 +y2**2
    x3y3 =x3**2 +y3**2

    mat = np.array([[x1y1, x1, y1, 1],[x2y2, x2, y2, 1],[x3y3, x3, y3, 1]])
    det_1 = np.round(np.linalg.det(np.hstack((mat[:,1].reshape(3,1),mat[:,2].reshape(3,1),mat[:,3].reshape(3,1)))) ,5)
    det_2 =  np.round(-np.linalg.det(np.hstack((mat[:,0].reshape(3,1),mat[:,2].reshape(3,1),mat[:,3].reshape(3,1)))) ,5)
    det_3 =  np.round(np.linalg.det(np.hstack((mat[:,0].reshape(3,1),mat[:,1].reshape(3,1),mat[:,3].reshape(3,1)))) ,5)
    det_4 =  np.round(-np.linalg.det(np.hstack((mat[:,0].reshape(3,1),mat[:,1].reshape(3,1),mat[:,2].reshape(3,1)))) ,5)

    x_c = (det_2/det_1)/(-2)
    y_c = (det_3/det_1)/(-2)
    r = np.sqrt(x_c**2 + y_c**2 - (det_4/det_1))

    return (x_c, y_c, r)
```

Code 1 - Code to generate a circle given 3 points

```python
def RANSAC_circ(X,thrsh):
    inliners_max=[]
    def T_check(x,c):
        return np.sqrt((x[0]-c[0])**2+(x[1]-c[1])**2)-c[2]
    for i in range(len(X)-2):
        for j in range(i+1,len(X)-1):
            for k in range(j+1,len(X)):
                inliners=[]
                C=get_circle(X[i][0],X[i][1],X[j][0],X[j][1],X[k][0],X[k][1])
                if C==0:
                    continue
                else:
                    for x in X:
                        if abs(T_check(x,C))<thrsh:
                            inliners.append(list(x))
                    if len(inliners)>len(inliners_max):
                        print(C)
                        Output,inliners_max,Output_2=C,inliners.copy(),[X[i],X[j],X[k]]
    return Output,Output_2,inliners_max
```

Code 2 - RANSAC function

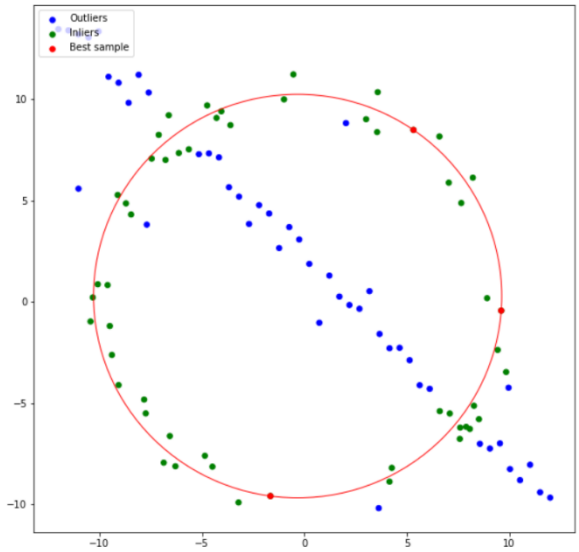


Figure 1 : Result

Question 2: Figure 2 shows an architectural image1 with a flag 2 superimposed. This is done by clicking four points on a planar surface in the architectural image, computing a homograph that maps the flag image to this plane, and warping the flag, and blending on to the architectural image. Carry this out for a couple of image pairs of you own choice. You may explain the (non-technical) rationale of your choice.

Figure 2

Answer: To compute the homograph the coordinates of the vertices of the respective overlay, the background and the respective mouse click points are needed. The custom points are obatined as indicated by the Code – 3. The transformation from the original to the warped image can be explained using the following set of equations.

$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$ and with H being equal to $H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$ matrix. This matrix can be aquired from opencv as indicated by code – 4 (21st line).

```
Cordinates=[]
def click_event(event, x, y, flags, params):
    global Cordinates
    # checking for left mouse clicks
    if event == cv2.EVENT_LBUTTONDOWN:
        Cordinates.append([x,y])
```

Code 3 - Code to store the coordinates of clicks

After that, blending is done using the following equation,

$$h(x) = \alpha f(x) + \beta g(x) + \gamma$$

Results are indicated in figure 3-5.

```
if __name__=="__main__":
    wall = cv2.imread("Images/uom1.jpg")
    logo = cv2.imread("Images/entc1.png")
    cv2.imshow('image', wall)

    print("Click on 4 points with the edges of the morph and close the window")
    cv2.setMouseCallback('image', click_event)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    size = logo.shape
    logo_corners = np.array([[0,0],[size[1] - 1, 0],[size[1] - 1, size[0] -1],[0, size[0] - 1 ]],dtype=float)
    homography_C=np.array([Cordinates[0],Cordinates[1],Cordinates[2],Cordinates[3]],dtype=float)
    h, status = cv2.findHomography(np.array(logo_corners),np.array(homography_C))
    warped = cv2.warpPerspective(logo, h, (wall.shape[1],wall.shape[0]))
    #cv2.fillConvexPoly(wall, homography_C.astype(int), 0, 16)
    #alpha = 0.5   #----------------->variable
    #beta = (1.0 - alpha)
    wall = cv2.addWeighted(wall,1, warped, 1,0.5)
    cv2.imshow('image', wall)
    cv2.imwrite("Final image - wall.jpg",wall)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Code 4 - main code snippet

Figure 3 - wall

Figure 4 - Logo

Figure 5 - Result

Question 3: In these questions, we will stitch the two Graffiti image3 img1.ppm onto img5.ppm.

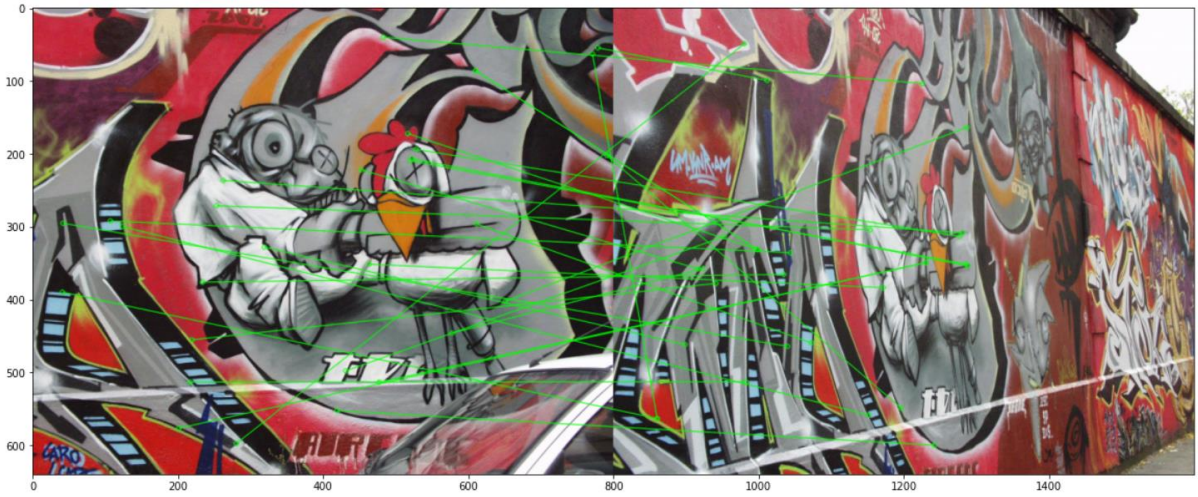
(a) Compute and match SIFT features between the two images.

(b) Compute the homography using your own code within RANSAC and compare with the homography given in the dataset.

(c) Stitch img1.ppm onto img5.ppm

Answer:

(a) The result of sift matching is as follows. The feature matching caused errors to some extent because some features of the second image didn't have as much as an intensity level as the first image.



(b) From the key points acquired in part (a) a known matrix A which is defined as,

$$A = \begin{bmatrix} \vdots & & & & & & & & \\ x_s{}^i & y_s{}^i & 1 & 0 & 0 & 0 & -x_d{}^i x_s{}^i & -x_d{}^i y_s{}^i & -x_d{}^i \\ 0 & 0 & 0 & x_s{}^i & y_s{}^i & 1 & -y_d{}^i y_s{}^i & -y_d{}^i y_s{}^i & -y_d{}^i \\ & & & & & & \vdots & & \end{bmatrix}$$

And this matrix connecting with the rearranged homography shows the following property,

$AA^T h = h\lambda$ which means by finding the eigen value of AA^T we can acquire the matrix h which can be rearranged to get the required homography.

The resultant homography from the manual function is as follows,

$$H = \begin{bmatrix} 1.26899491e+00 & 1.26899491e+00 & 1.92343057e+02 \\ -2.43801212e+00 & 2.43801212e+00 & 3.69532375e+02 \\ -6.59756029e-03 & 6.59756029e-03 & 1.00000000e+00 \end{bmatrix}$$

The given homography went like this,

$$H = \begin{bmatrix} 6.2544644e-01 & 5.7759174e-02 & 2.2201217e+02 \\ 2.2240536e-01 & 1.1652147e+00 & -2.5605611e+01 \\ 4.9212545e-04 & -3.6542424e-05 & 1.0000000e+00 \end{bmatrix}$$

```python
def Homography(pts_1,pts_2):
    mean_1,mean_2 = np.mean(pts_1,axis=0) , np.mean(pts_2,axis=0)
    s_1 = len(pts1)*np.sqrt(2)/np.sum(np.sqrt(np.sum((pts_1-mean_1)**2,axis = 1)))
    s_2 = len(pts2)*np.sqrt(2)/np.sum(np.sqrt(np.sum((pts_2-mean_2)**2,axis = 1)))

    tx_1 , ty_1 , tx_2, ty_2 = -s_1*mean_1[0], -s_1*mean_1[1] , -s_2*mean_2[0] , -s_2*mean_2[1]
    T_1 = np.array(((s_1,0,tx_1),(0,s_1,ty_1),(0,0,1)))
    T_2 = np.array(((s_2,0,tx_2),(0,s_2,ty_2),(0,0,1)))

    x_i = []

    for i in range (len(pts_1)):
        x_11 = T_1 @ np.concatenate((pts_1[i],[1])).reshape(3,1)
        x_21 = T_2 @ np.concatenate((pts_2[i],[1])).reshape(3,1)

        x_i.append((-x_11[0][0],-x_11[1][0],-1,0,0,0,x_21[0][0]*x_11[0][0],x_21[0][0]*x_11[0][0],x_21[0][0]))
        x_i.append((0,0,0,-x_11[0][0],-x_11[1][0],-1,x_21[1][0]*x_11[1][0],x_21[1][0]*x_11[1][0],x_21[1][0]))

    x_i = np.array(x_i)
    U , S , V = np.linalg.svd(x_i, full_matrices=True)
    h = np.reshape(V[-1],(3,3))
    H = np.linalg.inv(T_2) @ h @ T_1
    H = (1 / H.item(8)) * H
    return H
```

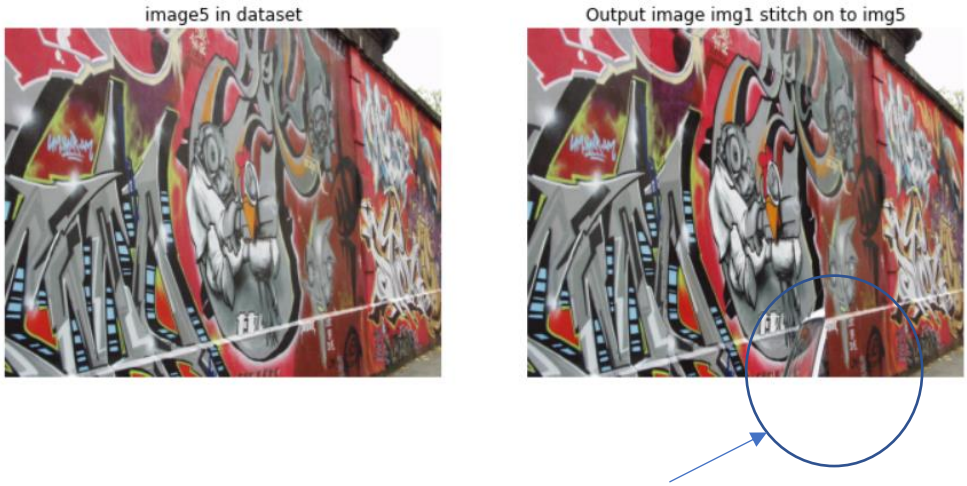
```python
def Homography_Ransac(pts1,pts2,s,error):
    tot_inliers = 0
    selected = None
    pts = np.hstack((pts1,pts2))
    req_iterations = np.log(1-(1-error))/np.log(1-(1-0.5)**s)

    for i in range(int(np.ceil(req_iterations))):
        np.random.shuffle(pts)
        points1,points1_rest = pts[:s,:2],pts[s:,:2]
        points2,points2_rest = pts[:s,2:],pts[s:,2:]

        H = Homography(points1,points2)
        inliers = [(points1_rest[j],points2_rest[j]) for j in range(len(points1_rest)) if distance_calc(points1_rest[j],points2_rest[j],H)<100]

        if len(inliers) > tot_inliers :
            tot_inliers = len(inliers)
            selected = np.array(inliers)
    H2 = Homography(selected[:,0],selected[:,1])
    return H2
```

(c) The result of the image stitching was as follows.



A little difference can be identified here.