University of Moratuwa

Department of Electronic and Telecommunication Engineering

EN2550 - Fundamentals of Image Processing and Machine Vision

Transformations and Neighborhood Filtering

W. A. D. K. De Silva	190128H

github.com/devindi99

March 5 2022

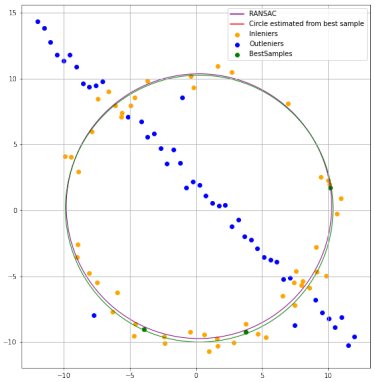
1. QUESTION 1

```python
def RANSAC_circle(Data_set,No_of_iterations,t):
    max_in_count=0
    best_fit_circle_coeff=[]
    best_sample_points=[]

    for sample in range(0,No_of_iterations+1):
        p1,p2,p3=random.randint(0,len(Data_set)-1),random.randint(0,len(Data_set)-1),random.randint(0,len(Data_set)-1)
        x1,x2,x3=Data_set[p1][0],Data_set[p2][0],Data_set[p3][0]
        y1,y2,y3=Data_set[p1][1],Data_set[p2][1],Data_set[p3][1]
        P = np.array([[2*x1 , 2*y1 , 1] , [2*x2 , 2*y2 , 1] , [2*x3 , 2*y3 , 1]])
        if (np.linalg.det(P)==0):
            continue
        K=np.array([[x1**2 + y1**2] , [x2**2 + y2**2] , [x3**2+y3**2]])*(-1)
        answer=np.linalg.inv(P)@ K
        g,f,c=answer[0][0],answer[1][0],answer[2][0]
        radius=np.sqrt(g**2+f**2-c)
        if radius>15:
            continue
        center=[-g,-f]
        in_count=0
        for i in range(0,len(Data_set)):
            d=abs(np.sqrt((Data_set[i][0]-center[0])**2 + (Data_set[i][1]-center[1])**2) - radius)
            if d < t:
                in_count+=1
        if in_count>max_in_count:
            max_in_count=in_count
            best_fit_circle_coeff=[g,f,c]
            best_sample_points=np.array([Data_set[p1],Data_set[p2],Data_set[p3]])
    return(best_fit_circle_coeff,best_sample_points,max_in_count)
```

Figure 1: Calculating values for best fitting circle and RANSAC estimated circle

```python
s=3
t=1.96
e=0.5
p=0.99
iterations=round(np.log(1-p)/np.log(1-(1-e)**s))

ransac_circle_coeff,ransac_sample,ransac_in_count=RANSAC_circle(X,iterations,t)
F,G,C=ransac_circle_coeff[0],ransac_circle_coeff[1],ransac_circle_coeff[2]
R=np.sqrt(G**2+F**2-C)

In,Out=[],[]
for p in X:
    d=abs(np.sqrt((p[0]+G)**2+(p[1]+F)**2)-R)
    if d<t:
        In.append(p)
    else:
        Out.append(p)
Inliners = np.array(In).T
Outliners = np.array(Out).T
Samp = ransac_sample.T
best_circle_coeff,best_sample,best_in_count=RANSAC_circle(X,10000,t)
best_F,best_G,best_C=best_circle_coeff[0],best_circle_coeff[1],best_circle_coeff[2]
best_R=np.sqrt(best_G**2+best_F**2-best_C)
best_sample_T=best_sample.T
```

(a)



(b) Output image

Figure 2

In this code snippet the values for the:

- number of points(s) = 3
- outlier ratio(e) = 0.5
- Threshold value(t) = 1.96
- Probability at least one random sample is free from outliers(p) = 0.99

The number of iterations are calculated using the equation:

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

When estimating the circle larger circles with radius larger than 15 have been eliminated for obvious reasons.

2. QUESTION 2

```python
def MouseHandling(event,x,y,f,pram):
    global im_temp,pts_src
    if event== cv.EVENT_LBUTTONDOWN:
        cv.circle(im_temp ,(x,y),2,(0,0,255),5,cv.LINE_AA)
        cv.imshow("Image",im_temp)
        if len(pts_src)<4:
            pts_src = np.append(pts_src,[(x,y)],axis=0)
```

(a)Function for selecting points

```python
def blending(name1,name2):
    global img1,img2,blend,im_temp,source_points

    img1 = cv.imread(name1)
    assert img1 is not None
    RGB_img1 = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
    im_temp = cv.cvtColor(RGB_img1, cv.COLOR_RGB2BGR)
    h, w = img1.shape[0] , img1.shape[1]

    img2 = cv.imread(name2)
    assert img2 is not None
    img2_w,img2_h = img2.shape[1] , img2.shape[0]
    points = np.array([[0,0] , [img2_w-1,0] , [img2_w-1,img2_h-1] , [0,img2_h-1] ])

    source_points = np.empty((0,2))

    cv.namedWindow("Image",1)
    cv.setMouseCallback("Image", MouseHandling)
    cv.imshow("Image",im_temp)
    cv.waitKey(0)

    H,S = cv.findHomography(source_points,points)
    t_img2 = cv.warpPerspective(img2, np.linalg.inv(H), (w, h))
    blend = cv.addWeighted(img1, 1, t_img2, 0.9, 0)
```

(b)Blending the two images

Figure 3

By using the mouse clicking function the user can select the corner points of the source image that he/she needs to blend the second image onto. In the following examples I have used images with a black screen in order to get a clear image output.

Figure 4

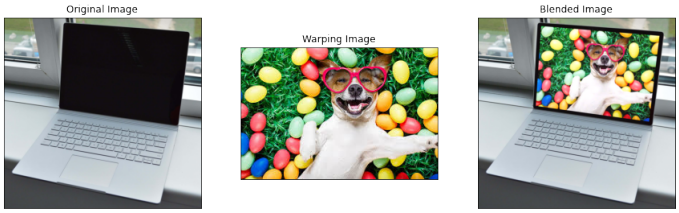


Figure 5

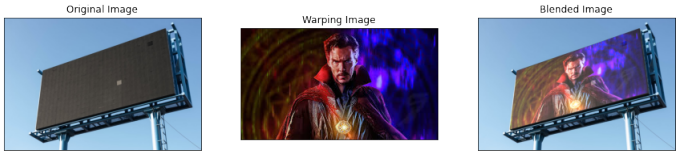


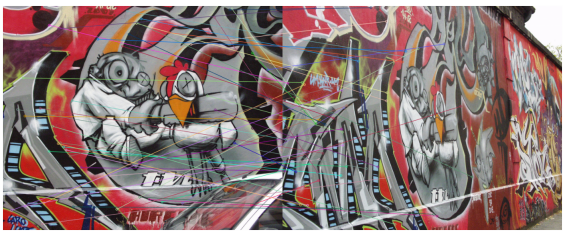
Figure 6

3. QUESTION 3

3.1 Matching SIFT features between images

```python
img1 = cv.imread(r'img1.ppm')
assert img1 is not None

img2 = cv.imread(r'img5.ppm')
assert img2 is not None

sift = cv.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(img1, None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2, None)

bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)

matched_img = cv.drawMatches(img1, keypoints_1, img2, keypoints_2, matches[:50], img2, flags=2)
```

(a)Code snippet



(b)Output image

Figure 7

3.2 Compute the homography using your own code within RANSAC

```python
import random as rnd
def sift_keypoint(src,dst):
    sift = cv.SIFT_create()
    keypoints_1, descriptors_1 = sift.detectAndCompute(src, None)
    keypoints_2, descriptors_2 = sift.detectAndCompute(dst, None)
    bf = cv.BFMatcher(cv.NORM_L1, crossCheck=True)
    matches = bf.match(descriptors_1,descriptors_2)
    matches = sorted(matches, key = lambda x:x.distance)
    list_kp1 = []
    list_kp2 = []

    for mat in matches:
        img1_idx = mat.queryIdx
        img2_idx = mat.trainIdx
        (x1, y1) = keypoints_1[img1_idx].pt
        (x2, y2) = keypoints_2[img2_idx].pt
        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)
```

Figure 8: Getting the key points of two images

```python
def Homography_by_ransac(src,dst):
    N = 10000
    threshold = 0.3
    max_inlinears = 0
    H=[]
    src_keylist , dst_keylist = sift_keypoint(src,dst)
    src_keylist, dst_keylist = np.array(src_keylist) , np.array(dst_keylist)
    for samp in range(0,N):
        ind1  =rnd.randrange(0,len(src_keylist))
        ind2 = rnd.randrange(0,len(src_keylist))
        ind3 = rnd.randrange(0,len(src_keylist))
        ind4 = rnd.randrange(0,len(src_keylist))
        s_pt1 ,d_pt1 = src_keylist[ind1]  , dst_keylist[ind1]
        s_pt2 ,d_pt2 = src_keylist[ind2]  , dst_keylist[ind2]
        s_pt3 ,d_pt3 = src_keylist[ind3]  , dst_keylist[ind3]
        s_pt4 ,d_pt4 = src_keylist[ind4]  , dst_keylist[ind4]
        colinearity =  bool(np.cross(s_pt1,s_pt2)) and bool(np.cross(s_pt1,s_pt3)) and bool(np.cross(s_pt1,s_pt4))
        colinearity =  colinearity and bool(np.cross(s_pt2,s_pt3)) and bool(np.cross(s_pt2,s_pt4))
        colinearity =  colinearity and bool(np.cross(s_pt3,s_pt4))
        if (not colinearity): continue
        A = np.array([[ -s_pt1[0], -s_pt1[1], -1, 0, 0, 0, d_pt1[0]*s_pt1[0], d_pt1[0]*s_pt1[1], d_pt1[0]],
                      [ 0, 0, 0, -s_pt1[0], -s_pt1[1], -1, d_pt1[1]*s_pt1[0], d_pt1[1]*s_pt1[1], d_pt1[1]],
                      [ -s_pt2[0], -s_pt2[1], -1, 0, 0, 0, d_pt2[0]*s_pt2[0], d_pt2[0]*s_pt2[1], d_pt2[0]],
                      [ 0, 0, 0, -s_pt2[0], -s_pt2[1], -1, d_pt2[1]*s_pt2[0], d_pt2[1]*s_pt2[1], d_pt2[1]],
                      [ -s_pt3[0], -s_pt3[1], -1, 0, 0, 0, d_pt3[0]*s_pt3[0], d_pt3[0]*s_pt3[1], d_pt3[0]],
                      [ 0, 0, 0, -s_pt3[0], -s_pt3[1], -1, d_pt3[1]*s_pt3[0], d_pt3[1]*s_pt3[1], d_pt3[1]],
                      [ -s_pt4[0], -s_pt4[1], -1, 0, 0, 0, d_pt4[0]*s_pt4[0], d_pt4[0]*s_pt4[1], d_pt4[0]],
                      [ 0, 0, 0, -s_pt4[0], -s_pt4[1], -1, d_pt4[1]*s_pt4[0], d_pt4[1]*s_pt4[1], d_pt4[1]],
                      [0, 0, 0, 0, 0, 0, 0, 0, 1]])
        h = np.linalg.inv(A) @ np.array([[0,0,0,0,0,0,0,0,1]]).T
        h = np.reshape(h,(3,3))
        X = np.vstack([src_keylist.T , np.ones( (1, len(src_keylist)) )])
        transformed_keys = h @ X
        z= np.array([transformed_keys[-1]]).T
        transformed_keys = (transformed_keys[:2].T) / z
        inlinears_count = 0
        for  i in range(0 , len(transformed_keys)):
            distance = np.sqrt( (transformed_keys[i][0] - dst_keylist[i][0])**2 + (transformed_keys[i][1] - dst_keylist[i][1])**2 )
            if abs(distance) < threshold:
                inlinears_count += 1
        if inlinears_count > max_inlinears:
            max_inlinears = inlinears_count
            H = h
    return(H)
```

Figure 9: Calculating H matrix of two images

3.3 Stiching images

```
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
```

(a) Actual H matrix

```
6.27501634e-01  5.19878005e-02  2.22598834e+02
2.24996057e-01  1.15696696e+00 -2.57669893e+01
5.03288696e-04 -4.46994208e-05  9.95349119e-01
```

(b)Calculated H matrix

Figure 10

The calculated homography matrix and actual matrix are approximately equal.

(a) Transformed image

(b)Stitched image

Figure 11

Since the number of sift feature matches between image 1 and image 5 were insufficient, homographies between two consecutive images were computed using the RANSAC algorithm and were multiplied together to obtain the homography from image 1 to 5.

4. REFERNECES

- A Detailed Guide to the Powerful SIFT Technique for Image Matching
- Adding (blending) two images using OpenCV
- Image Stitching with OpenCV and Python