Computer vision Homework3

Team: 20

Member: 0856633 湛興達

0856645 黃品捷

0856116 黃偉嘉

Introduction

Image stitching is the process of combine multiple photographic images with overlapping fields of view to produce a high-resolution image.
In modern application, image stitching is widely used.
The image stitching process can be divided into five main components:

1:Feature points detection of two images.

2:Accroding to those detection point which are belongs to each image, find a correspondence feature point from image_1 to image_2.

3:Use RANSAC algorithm, sample correspondence feature points which has the largest number of inlier point to model a Homography matrix.

4:After get the Homography matrix H, warp image to create panoramic image and get the result.

5:Result may have some acne problem, used blending function to enhance the result.

Below we will describe the implement's details.

Implementation procedure

Step1:

Read the image.

```
#read img
picture_name1 = '1.jpg'
picture_name2 = '2.jpg'

img1_Ori = cv2.imread('./data/'+ picture_name1 , cv2.IMREAD_COLOR)
img2_Ori = cv2.imread('./data/'+ picture_name2 , cv2.IMREAD_COLOR)

img1 = cv2.imread('./data/'+ picture_name1 , cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('./data/'+ picture_name2 , cv2.IMREAD_GRAYSCALE)
```

Step2:

SIFT

Interest points detection & feature description by SIFT, The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images.

Here, we just simply call the function which offer by OpenCV.

```
#Interest points detection & feature description by SIFT
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
```

Draw the key points on given image (img_1, img_2)

Now we have key points sets of two images.

Step3:

Finding Matches

```
#Find Matches
matches = Find_Matches(des1,des2,ratio = 0.5,k=2)
```

Find the most similar corresponding point according to the vector.
“k”: For each point in image 1 find the k points that are most similar in image 2.
Usually k is 2.

“ratio”: Among the most similar points, if the distance between the first similar point and the distance between the second similar point is less than a certain ratio, the first similar point will be the corresponding point we want.

```python
def Find_Matches(des1,des2,ratio = 0.1,k=2):
    res = []
    for v1 in des1:
        list_distance = []
        for v2 in des2:
            Distence = np.sqrt( (np.square(v1-v2)).sum() )
            list_distance.append(Distence)
        list_distance = np.array(list_distance)
        Sort_index = np.argsort(list_distance)
        match = []
        for K in range(k):
            Index = Sort_index[K]
            match.append([list_distance[Index],Index])  #distance,index
        res.append(match)

    #Pick good
    good = []
    for (i,(m,n)) in enumerate(res):
        if m[0] < ratio * n[0]:       #distance
            good.append([i,m[1]])   #queryIdx,trainIdx
    return good
```

First, find each point in image1 to find the two most similar points in image2.

Similar: Euclidean distance.

Then we will compare the distance between the two points found if they are less than a ratio.

If it is less than a ratio, we will find that point as the match point found for us.

If not, ignore the point and continue process until check all points.

The function will return a list of a pair of indexes, pair of indexes is like (idx1, idx2), idx1 is index for the key point 1, idx2 is index for the key point 2.

Step4:

Draw line between corresponding feature points.

```python
#Make Feature image
Feature_img = FeatureMatchingImg(img1_Ori,img2_Ori,matches)
```

According to the matches obtained in the third step, connect corresponding feature points.

```python
def FeatureMatchingImg(img1,img2,matches):
    connect_img = np.hstack((img1,img2))
    for (queryIdx,trainIdx) in matches:
        p1 = np.array(kp1[queryIdx].pt)
        p2 = np.array(kp2[trainIdx].pt)
        p2[0]+=img1_Ori.shape[1]

        Img1RowIdx = p1[1]
        Img1ColIdx = p1[0]
        Img1Index = np.array([Img1RowIdx,Img1ColIdx])
        Img2RowIdx = p2[1]
        Img2ColIdx = p2[0]
        Img2Index = np.array([Img2RowIdx,Img2ColIdx])
        direction = np.array([Img2RowIdx-Img1RowIdx,Img2ColIdx-Img1ColIdx])
        direction = direction/np.linalg.norm(direction)
        k = 0
        color = np.random.rand(3)*255
        while True:
            curren_piexl = Img1Index + (k*direction).astype(int)
            k+=1
            if curren_piexl[1]> Img2Index[1]:
                break
            connect_img[int(curren_piexl[0]),int(curren_piexl[1]),:] = color
    return connect_img
```

First, merge the two image together, and second, draw the line according to the position.

The principle is for each pair of points, calculate the direction of the point in image1 and the point in image2, change the color of all connected lines. Colors are set randomly.

Step5:

Get the pixel coordinates of corresponding key points in their own image

```python
points_in_img1,points_in_img2 = GetPointFromImg(kp1,kp2,matches)
```

According to the points in matches, find their pixel coordinates in their own image.

```
def GetPointFromImg(kp1,kp2,matches):
    dst__pts = []
    src__pts = []
    for matche in matches:
        p1 = np.float32( kp1[matche[0]].pt + (1,) )
        p2 = np.float32( kp2[matche[1]].pt + (1,) )
        dst__pts.append(p1)
        src__pts.append(p2)
    points_in_img1 = np.array(dst__pts)
    points_in_img2 = np.array(src__pts)
    return points_in_img1,points_in_img2
```

“Key points (kp)” have pixel coordinates with corresponding points. Because we will compute the homography matrix later, so we need to translation 2-dimension coordinate to 3- dimension coordinate.
The function return the corresponding key point, but the value is the true pixel coordinate.

Step6:

Finding homography matrix

```
H = homomat(points_in_img2, points_in_img1)
```

According to the corresponding coordinates of the two list, to calculate homography matrix.

```python
def homomat(src__pts,dst__pts):
    sigma = 2
    t = np.sqrt(5.99*sigma*sigma)
    s = 8   # Minimum number needed to fit the model
    p = 0.95 #prob=0.95
    e = 0.05 #outlier ratio: e
    N = np.log(1-p)/np.log(1-(1-e)**s)
    k = int(N) # number of iterations
    H = np.eye(3)
    MaxInliers = -1
    Index = np.arange(len(src__pts))    #[0,1,2,...]
    for i in range(k):
        np.random.shuffle(Index)
        choice_idx = Index[0:s]
        unchoice_idx = Index[s:]
        p1 = np.array([ src__pts[index] for index in choice_idx])
        p2 = np.array([ dst__pts[index] for index in choice_idx])
        Hi = getHi(p1,p2)
        nInliers = ComputeInliers(Hi,t,src__pts,dst__pts,unchoice_idx)
        if MaxInliers < nInliers:
            MaxInliers = nInliers
            H = Hi
    #print(MaxInliers)
    return H
```

“t”: If a score is less than t, it is classified as inlier.
“s”: Minimum number needed to fit the model
“p”: Desired probability of success
“e”: Outlier ratio
“k”: The number of iterations
Because we have to use RANSAC to find homography matrix, so we need to randomly select some points.

```python
np.random.shuffle(Index)
choice_idx = Index[0:s]
unchoice_idx = Index[s:]
```

we record the selected index.

```python
p1 = np.array([ src__pts[index] for index in choice_idx])
p2 = np.array([ dst__pts[index] for index in choice_idx])
Hi = getHi(p1,p2)
```

After getting the index, save the coordinates of the corresponding point. Then can compute homography matrix.

```python
def getHi(p1,p2):
    A = []
    for i in range(0, len(p1)):
        x, y = p1[i][0], p1[i][1]
        u, v = p2[i][0], p2[i][1]
        A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
        A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
    A = np.asarray(A)
    U, S, Vh = np.linalg.svd(A)
    L = Vh[-1,:] / Vh[-1,-1]
    Hi = L.reshape(3, 3)
    return Hi
```

This algorithm is exactly the same as the first homework (Copy the code of the first homework), so no more explanation here.

```python
nInliers = ComputeInliers(Hi,t,src_pts,dst_pts,unchoice_idx)
if MaxInliers < nInliers:
    MaxInliers = nInliers
    H = Hi
```

When get homography matrix, we need to calculate their scores from the points that have not been selected, if the score is small or equal than t, classify it as inlier, otherwise, classify it as outlier.
We want to find the corresponding matrix with the most inliers.

```python
def ComputeInliers(Hi,t,img1_p,img2_p,unchoice_idx):
    res = 0
    for index in unchoice_idx:
        p0 = img1_p[index]
        p1 = img2_p[index]
        p2 = Hi.dot(p0.T)
        Score = np.linalg.norm(p2.T - p1)
        if(Score<=t):
            res+=1
    return res
```

If the score is small or equal than t, classify it as inlier.
The function return the number of inlier in the not selected points.

Step7:

Warping image

```
res = warp(img1_Ori,img2_Ori,H)
```

When get homography matrix, we need to warp image to create panoramic image.

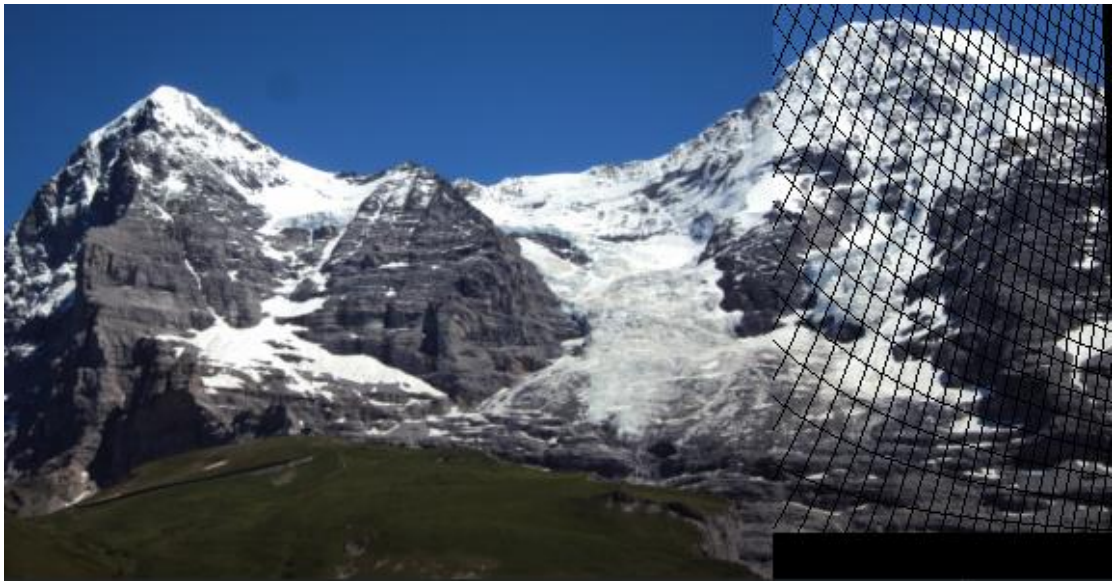
```
def warp(img1_Ori,img2_Ori,H):
    res = np.zeros((img1_Ori.shape[0] , img1_Ori.shape[1] + img2_Ori.shape[1],3))
    res[:] = np.nan
    for row in range(img2.shape[0]):
        for col in range(img2.shape[1]):
            #p = np.array([row,col,1])
            p = np.array([col,row,1])
            new_p = (H.dot(p.T)).T
            new_p = new_p/new_p[-1]
            new_p = new_p.astype(int)

            new_p[1] = min(max(0,new_p[1]),res.shape[0]-1)
            new_p[0] = min(max(0,new_p[0]),res.shape[1]-1)
            res[new_p[1],new_p[0],:] = img2_Ori[row,col,:]

    res[0:img1_Ori.shape[0],0:img1_Ori.shape[1]] = img1_Ori
    return res
```

First create an array, the size of array is two images combine size. (parallel)
Set “nan” to each value of array.
Then convert each pixel coordinate in image 2 to the pixel coordinate of image 1 by homography matrix.
If the converted coordinates exceed the size of the array, ignore it.
After the conversion of all the coordinates in image2 is completed, fill in the values of the original coordinates in image1.

Step8:

Fill in non-integer coordinate pixels

After warp image, some pixels from original img2 could be non-integer coordinates. So, the result will be very strange like the following image. We use linear interpolation to fill in those non-integer coordinate pixels.



We set those non-integer coordinate pixels to np.nan. Cropping image by removing rows and columns that are all np.nan.

```
def trim(frame):
    cols = np.isnan(frame).all(axis = 0)
    rows = np.isnan(frame).all(axis=1)
    cols = cols[: , 0]
    rows = rows[: , 0]
    x = cols.size - 1
    y = rows.size - 1
    for i in reversed(range(cols.size)):
        if(not cols[i]):
            x = i
            break
    for i in reversed(range(rows.size)):
        if(not rows[i]):
            y = i
            break
    return frame[:y , :x]
```

Use for loop to find those unknown pixels and use linear interpolation to fill in the color.

```
for r in range(res.shape[0]):
    for c in range(res.shape[1]):
        if np.isnan(res[r , c , 0]):
            final_img[r,c,:] = f(r,c)
            res[r,c,:] = final_img[r,c,:]
        else:
            final_img[r,c,:] = res[r,c,:]
```

Find the closest point that is not unknown on the right and left.

```
def f(r,c):
    x1 = x2 = y1 = y2 = np.nan
    for i in range(c , -1 , -1):
        if( np.isnan(res[r , i , 0])):
            continue
        else:
            x1 = i
            break
    for i in range(c , res.shape[1] , 1):
        if( np.isnan(res[r , i , 0])):
            continue
        else:
            x2 = i
            break

    for i in range(r , -1 , -1):
        if( np.isnan(res[i , c , 0])):
            continue
        else:
            y1 = i
            break
    for i in range(r , res.shape[0] , 1):
        if(np.isnan(res[i , c , 0])):
            continue
        else:
            y2 = i
            break
```

Use those two points to do the linear interpolation. Do the same thing along y axis and average the result. Use this result as it's color.

```
if(np.isnan(x1) or np.isnan(x2)):
    if(np.isnan(y1) or np.isnan(y2)):
        return np.nan
    else:
        Ry = ((y2 - r) / (y2-y1)) * res[y1 , c] + ((r - y1) / (y2-y1)) * res[y2 , c]
        return (Ry).astype(int)
elif(np.isnan(y1) or np.isnan(y2)):
    Rx = ((x2 - c) / (x2-x1)) * res[r , x1] + ((c - x1) / (x2-x1)) * res[r , x2]
    return (Rx).astype(int)
else:

    Rx = ((x2 - c) / (x2-x1)) * res[r , x1] + ((c - x1) / (x2-x1)) * res[r , x2]
    Ry = ((y2 - r) / (y2-y1)) * res[y1 , c] + ((r - y1) / (y2-y1)) * res[y2 , c]
    return ((Rx + Ry) / 2).astype(int)
```

After fill in the color, we get the result like following image.

Step9:

Blending image

There are obvious borders in the picture, so we use bilateral Filter function in the OpenCV library to handle this problem. Bilateral Filter will consider the difference in luminosity, color and geometric proximity between pixels. So, it can keep the edges of different object in the picture while doing blending.

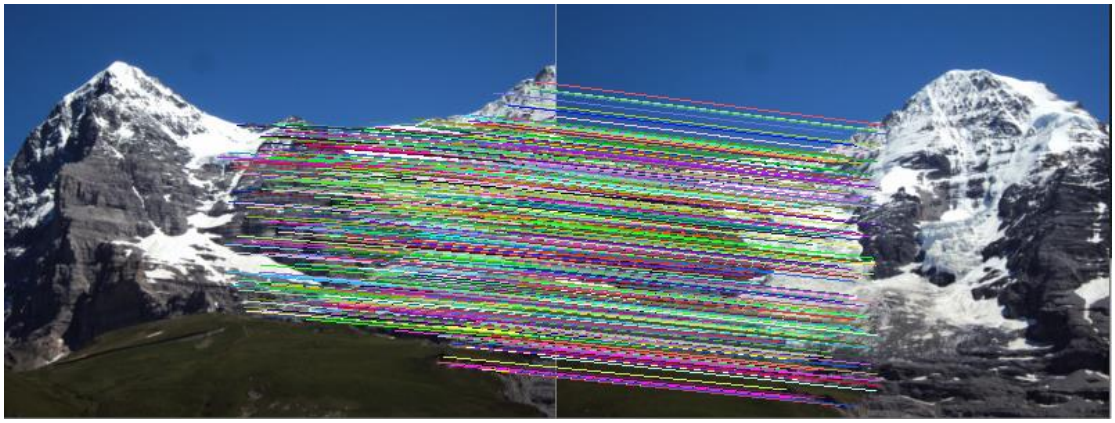
```
def showImg(img):
    sigma = 75
    blur = cv2.bilateralFilter(img,9,sigma,sigma)

    cv2.imshow('My Image', blur)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Following is the result image after doing blending.

Experimental results

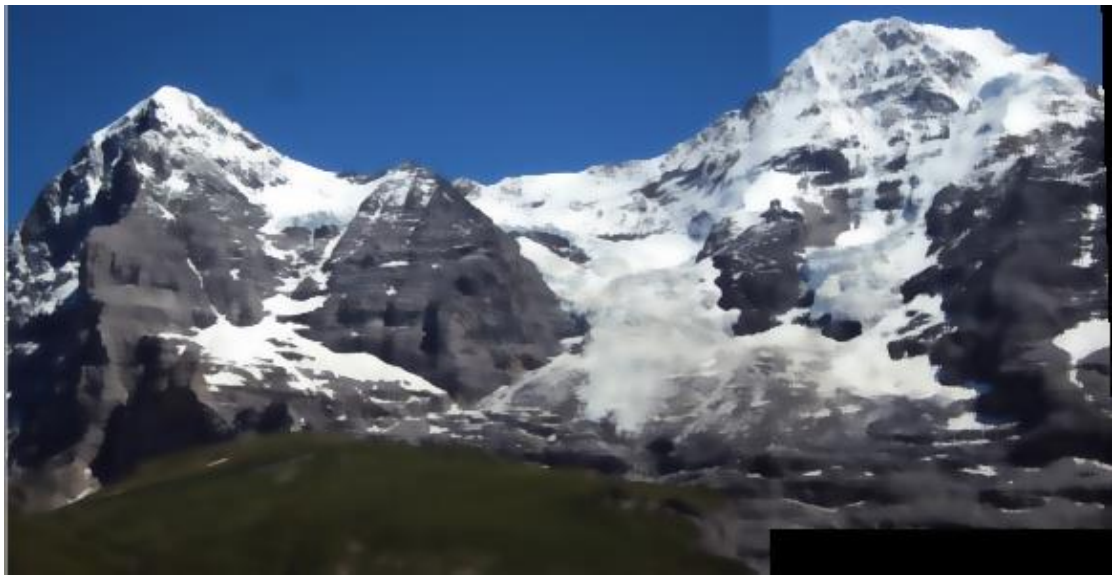
"hill1 & hill2"

(original image)



(Feature matching)

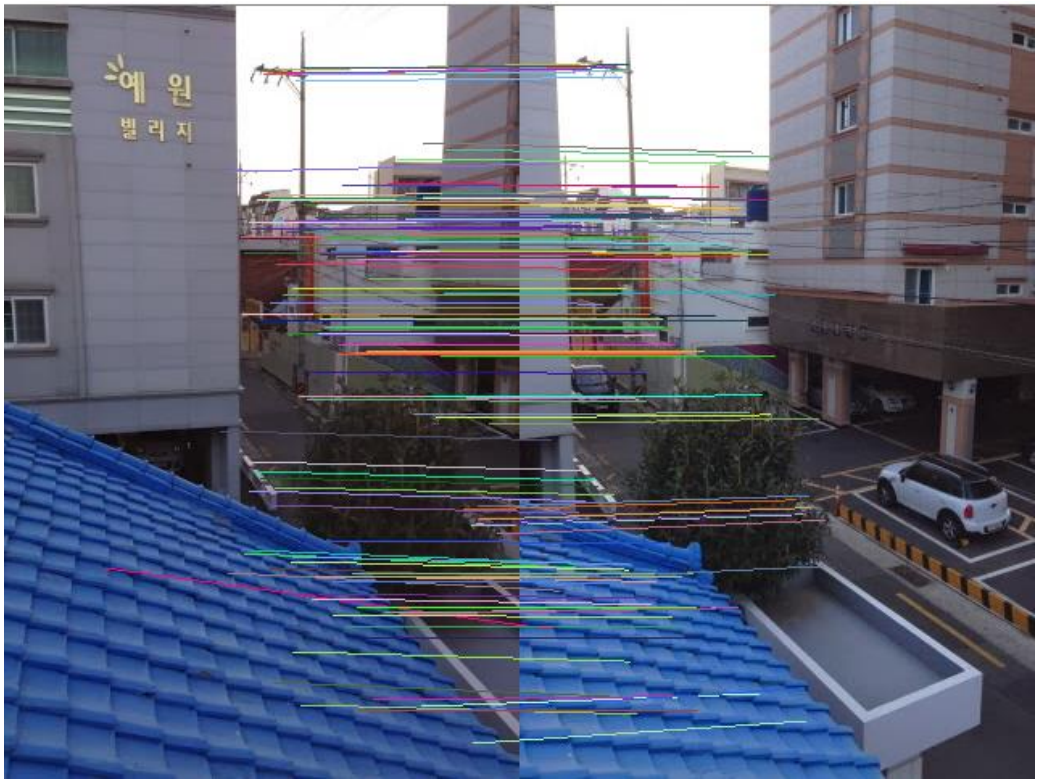
(Result without blending)



(Result with blending)

“S1 & S2”

(Original image)



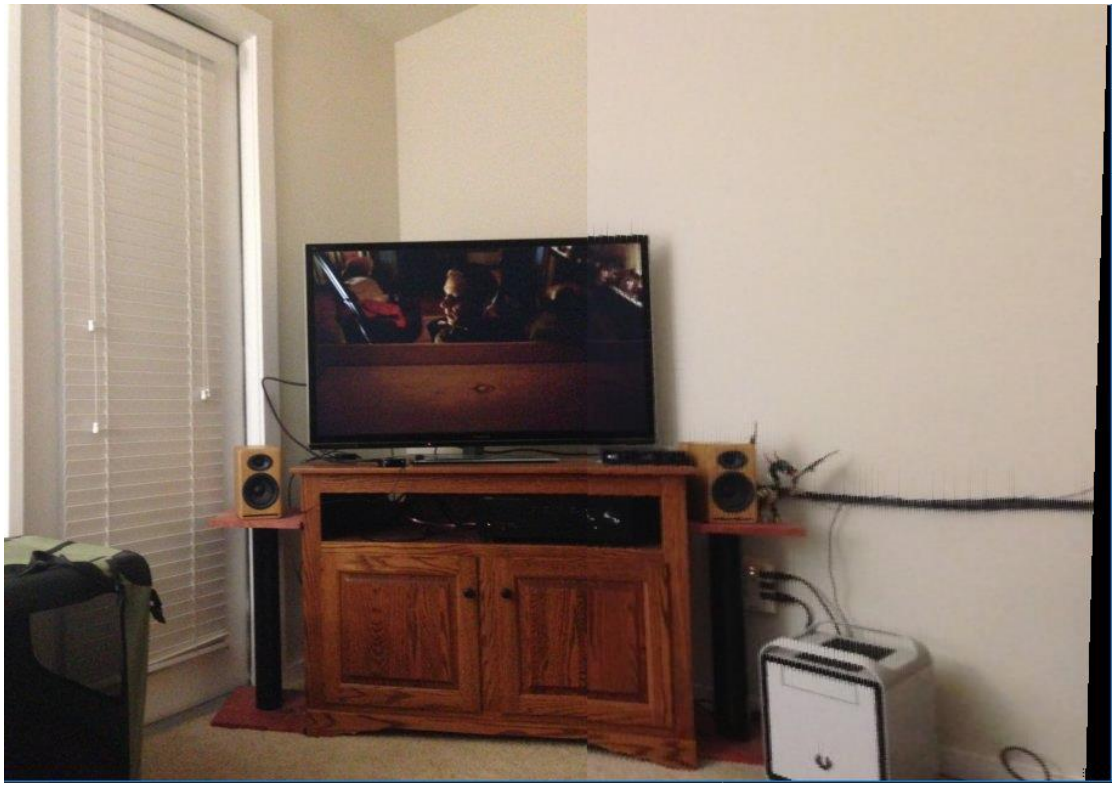
(Feature matching)

(Result without blending)

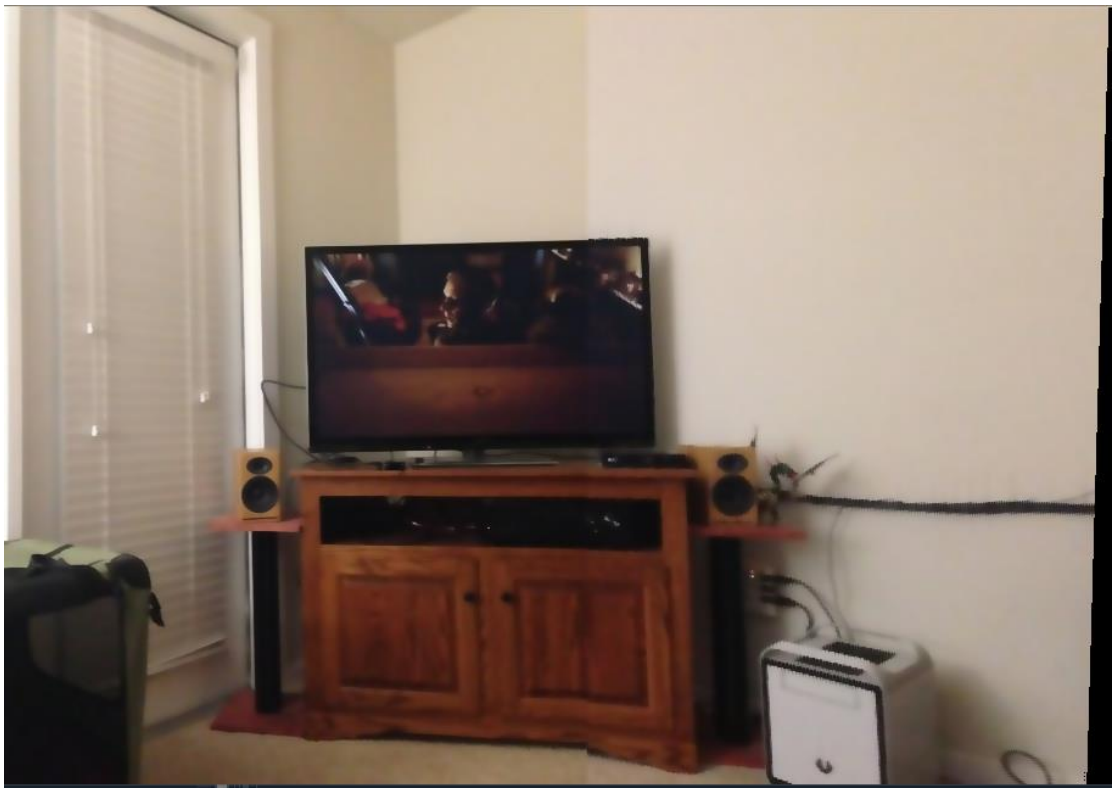
(Result with blending)
“1 & 2”

(Original image)

(Feature matching)



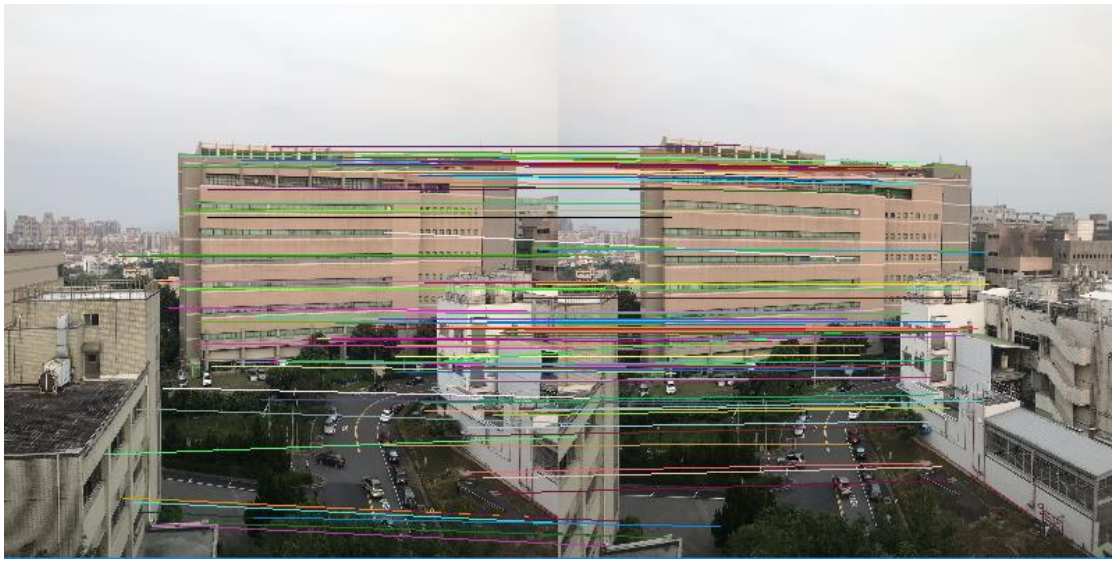
(Result without blending)



(Result with blending)

“Photos taken by ourselves”

(Original image)



(Feature matching)

(Result without blending)

(Result with blending)

Discussion

First, When I try to use SIFT function which is offered by OpenCV, The IDE give me a message that this OpenCV version I installed don't support SIFT and SURF algorithm because the patent problem.

I search some information on network, I found a solution that I should downgrade my opencCV version and openCV contribute version.

I do so, it works.

After solve the version problem, I started to do the works that we need to implement in this homework.

At first, I need to do the feature matching of two images, but I have no idea where I should start. After I found some information in internet and the lecture power point, I need to compute the eigenvector's L2 distance between two image features descriptor.

Then, I notice that OpenCV SIFT function will return the features descriptor which is contain some eigenvector's information in two image features, then, I solve this problem.

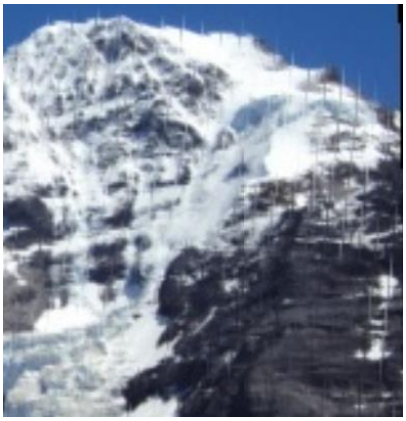
Next, I encountered many problems while doing RANSAC.

The main problem is we don't know how to set the parameters, such as

the number of iterations and how much to set the “t” value.
I couldn't find the answer on the Internet, so I realized that these parameters should be set by myself.
There are also many parameters to be set, and different parameters must be set for different photos when doing experiments. This will take a lot of time to change the parameter settings.

I encountered many problems while doing RANSAC.
The main problem is we don’t know how to set the parameters, such as the number of iterations and how much to set the “t” value.
I couldn't find the answer on the Internet, so I realized that these parameters should be set by myself.
There are also many parameters to be set, and different parameters must be set for different photos when doing experiments. This will take a lot of time to change the parameter settings.

In the fill in non-integer coordinate pixels part, we need to take those pixels we have updated before when we calculate the unknown pixels. If we do not consider the updated pixels, there will be a little burr on the image like the following image.



Conclusion

In this assignment, we learn how to get the key points between two images by using SIFT. Use these key points to find the transform matrix and compute the corresponding coordinate from one image to another image. Merge two image and fill in color to those non-integer coordinate pixels. Finally, merge two images and blending it to get a better result.

In our system, there are too many parameters need to adjust to get a nice result. For example, we should not use blending in hill image to get a better result. But in other images, use blending can get the better result. Maybe we can try to develop a more general system to reduce the parameters in next time.

Work assignment plan

We do this assignment all together.