ELEC474 PROJECT

NOVEMBER 30, 2021

XIANGMAN LI
Student #: 20119884

1. Source Modules

Source modules	Originality
def Matches(image1,image2):	Yes
def read_directory(directory_name):	Yes
def Homography(ref,tar,i)	Yes
def find_postion(ref,tar,i)	Yes
def Stich(ref, tar)	Yes

2.ORIGINALITY STATEMENT

‘The signature below attest that this submission is our original work. Following professional engineering practice, I bear the burden of proof for original work. I have read the Policy on Academic Integrity posted on the Faculty of Engineering and Applied Science web site (http://engineering.queensu.ca/policy/Honesty.html) and confirm that this work is in accordance with the Policy.

Signature: Xiangman Li Date: 01/12/2021
Name: Xiangman Li ID #: 20119884

3. Step_One: Match Feature

```
        def read_directory(directory_name)
        def Matches(image1, image2)
        def Homography(ref,tar, i)
        def find_postion(ref,tar,i):
        def Stiching(ref, tar)
```

The first step is to read images from the directory, the list keeping images called img_list[ ]. The starting point is an unordered set of *N* images. Therefore, I use *Matches(image1, image2)* function to check the keypoint and descriptors between two images and use the ratio to find a good match. The feature I used is SIFT. Then, this function will be used in *Homography(ref,tar, i)*, and *Stiching(ref, tar)*.
Homography(ref,tar, i) calculates the Homography and returns the mask and the number of good matches I got from *Matches(image1, image2)* function.
The purpose of *find_postion(ref,tar,i)* is to filter the bad match and find the position of each image.
The first judging condition is the number of good matches should be larger than THRESHOLD. The image will not be added to the list if the number of good matches is less than THRESHOLD.
The second one is using *cv2.perspectiveTransform(points, f)* to find the relative position of each image with the reference image. If Output is larger than 0, this image will be added to the left list; otherwise, add to the right list.
In conclusion, step one aims to find the good match of images and sort them based on the good match.

4. Step_Two: Estimate Transformation

```
            def Matches(image1, image2)
            def Homography(ref,tar, i)
            def radiometric(l_r,output,j)
            def Stiching(ref, tar)            inherit
```

The purpose of *Stiching(ref, tar)* is calculate the transformation and warp the image. The function we calculate the transformation is *cv2.warpPerspective (source_image, result, outputimage_size) .* where source_image is the image whose perspective is to be changed using the getPerspectiveTransform() function, result is the inverse of the f from homography and outputimage_size is the size of the output image that fits the size of the original image using the warpPerspective() function.

At the same time, the method I used as above base on geometric transformation, and I define a function called radiometric(l_r,out) to do the radiometric transformation. The principle of this function is to compare the reference image's pixels with left image's pixels. The first step was doing the same thing with geometric transformation, then I build a new background to merge the reference image and target image.

5. Step_Three: Merge Image:

```
            def Matches(image1, image2)
            def Homography(ref,tar)
            def radiometric(l_r,output,j)
            def Stiching(ref, tar)
```

In Step Two, I have already gotten the output of the *warpPerspective()* function. My second variable in *warpPerspective()* is f, which is a matrix. Therefore, the output of this function is also a matrix. In order to show the transformation into the image, I use *" warpImg[0:ref.shape[0], 0:ref.shape[1]] = ref"* to get the warp image.
Then, the output image after we warp should be an irregular shape, which means this image would have the black border if we use this image to warp the next image. My solution to this problem is using cv2.findContours to find the edge of the black border, which we studied in Lab 3.
In *radiometric(l_r,out)*, we build a new canvas after we warp the images. Then I compared the reference image's pixels with left image's pixels using *np.clip()* to get the new pixel values and write the new pixel into the canvas.

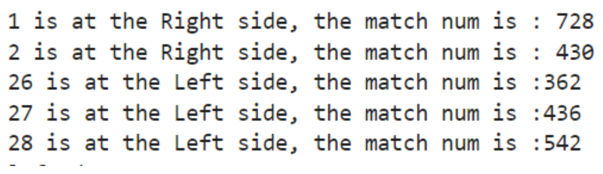
6. Test Result

In my test part, I used the "office2" folder and the "WLH" Folder to do the test. My design idea is to merge the panoramic image that the photographer stands at the center of the room. The threshold I used to filter the match is 300, finding the best three left images ad three right images. I tried to merge 28 images, but we do not have related parameters, so the images will be distorted. I divide the dataset into three parts, which can get three images to include four walls in the room.

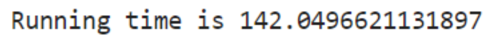
The result output image will be shown in D3, which the name is “office1.jpg”, “office2.jpg”, “office3.jpg”. The running time of each separated dataset is 142 seconds, 156 seconds, 143 seconds. Then the listing of the matching metric values between each image (after filtering) is shown below.
In addition, running time of geometric transformation is less than radiometric transformation.

“Office2 DataSet”

When the centre image is 0:

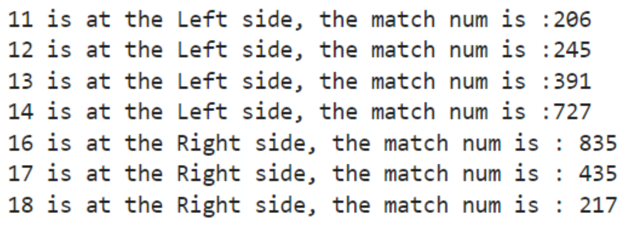


```
1 is at the Right side, the match num is : 728
2 is at the Right side, the match num is : 430
26 is at the Left side, the match num is :362
27 is at the Left side, the match num is :436
28 is at the Left side, the match num is :542
```

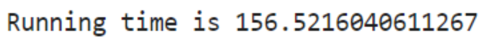


```
Running time is 142.0496621131897
```

When the centre image is 15:

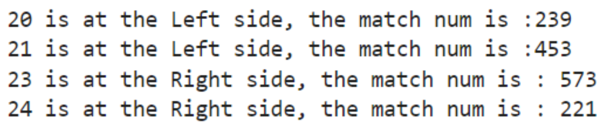


```
11 is at the Left side, the match num is :206
12 is at the Left side, the match num is :245
13 is at the Left side, the match num is :391
14 is at the Left side, the match num is :727
16 is at the Right side, the match num is : 835
17 is at the Right side, the match num is : 435
18 is at the Right side, the match num is : 217
```

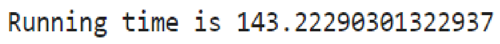


```
Running time is 156.5216040611267
```

When the centre image is 22:

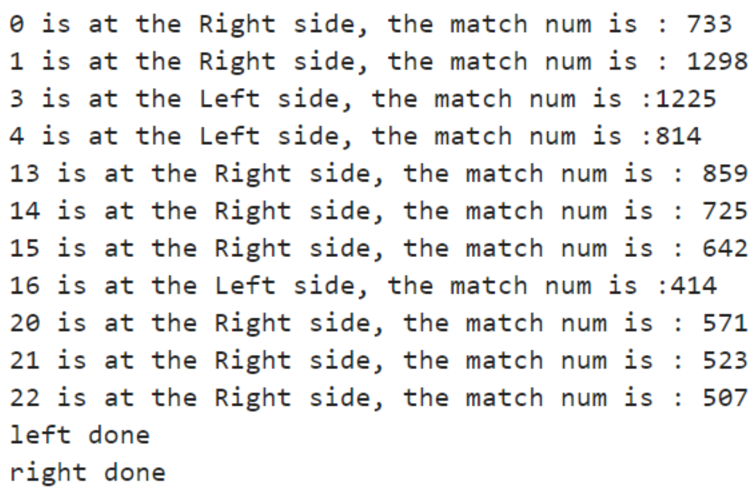


```
20 is at the Left side, the match num is :239
21 is at the Left side, the match num is :453
23 is at the Right side, the match num is : 573
24 is at the Right side, the match num is : 221
```



```
Running time is 143.22290301322937
```

“WLH Dataset”



```
0 is at the Right side, the match num is : 733
1 is at the Right side, the match num is : 1298
3 is at the Left side, the match num is :1225
4 is at the Left side, the match num is :814
13 is at the Right side, the match num is : 859
14 is at the Right side, the match num is : 725
15 is at the Right side, the match num is : 642
16 is at the Left side, the match num is :414
20 is at the Right side, the match num is : 571
21 is at the Right side, the match num is : 523
22 is at the Right side, the match num is : 507
left done
right done
```

```
Running time is 217.6526803970337
```

6. Correctness and Effectiveness

The design solution can be shown clearly and merge images with no black crevice. The position of each image in one level can be found. At the same time, the depth of the panoramic image can be shown. Code is designed and implemented correctly and is cleanly organized. The design idea bases on the lab description and previous labs. Output is presented and correct.
The calculation speed is in the standard range; my computer is Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz, 16GB. I used the other people's laptop with RAM 64GB, and this laptop ran 60 seconds per image.

7. Improvement

In the last part, it is talked that the code is just proper for one level panoramic image, it is hard to use in the third database, St. James. The reason is St. James need to find the upper, down, left, and right side.
My basic idea is assuming the whole image of St. James can be separated into NxN, which has N columns and N rows. Therefore, I can rotate the whole images dataset into 90 degrees and classify 25 images in N parts; each part is one column. Then, merge them into N images. Finally, rotate these N images 270 degrees and repeat the last step.
However, it is hard to come true because this method needs strong computer power. At the same time, this method used in the dataset includes N-level images, which will not achieve the perfect result.
Based on my research, it should be finding the center of the whole image. Then, find the surrounding images, which means I need to use the 2 times Euclidean distance and find the good match between upper, down, left, and right images.
In addition, all the outputs have a little bit of distortion, and the perfect image should be smoothed. Therefore, the output should be adjusted to the shape.