Computer Vision and Image Process

Project-2 Image Stitch

UBID:50291971

Name: tingting wang

Email:twang49@buffalo.edu

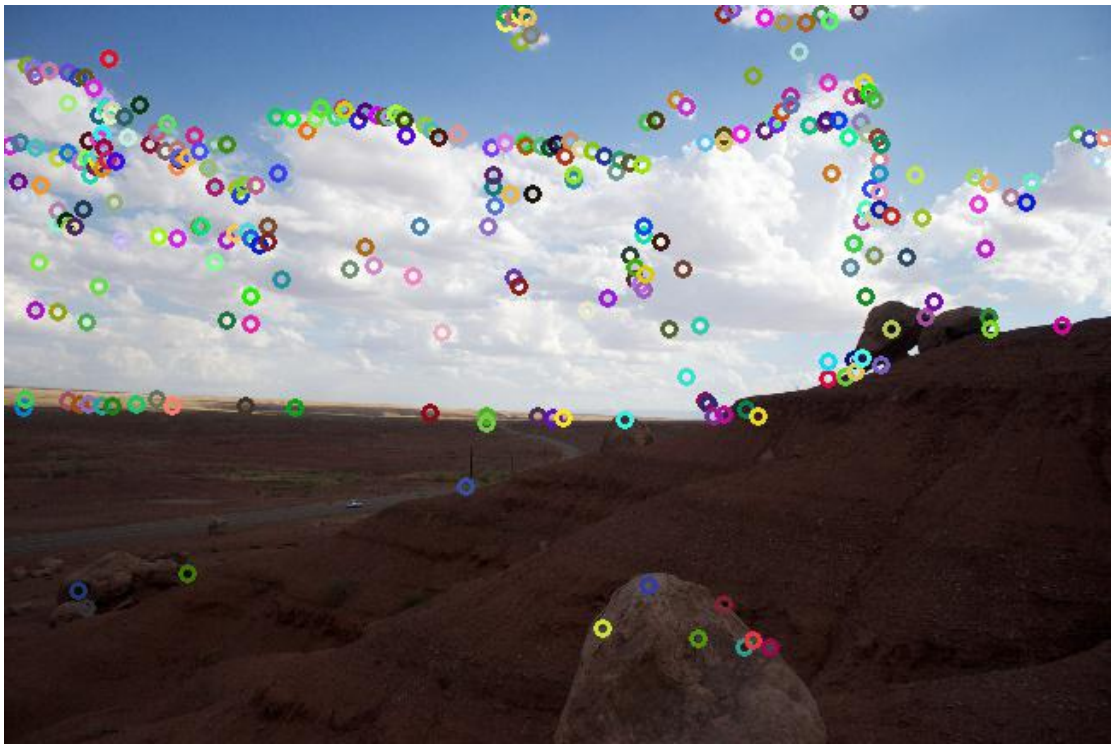
Algorithm Design:

1. Feature Detect and Describe
2. Keypoint Match or Feature Match
3. Use RANSAC to Remove Bad Matches
4. Use RANSAC to Compute Projection Matrix
5. Stitch Image with some tricks
6. Decide Spacial Arrangement
7. Something Else

Algorithm Detail:

1. Feature Detect and Describe

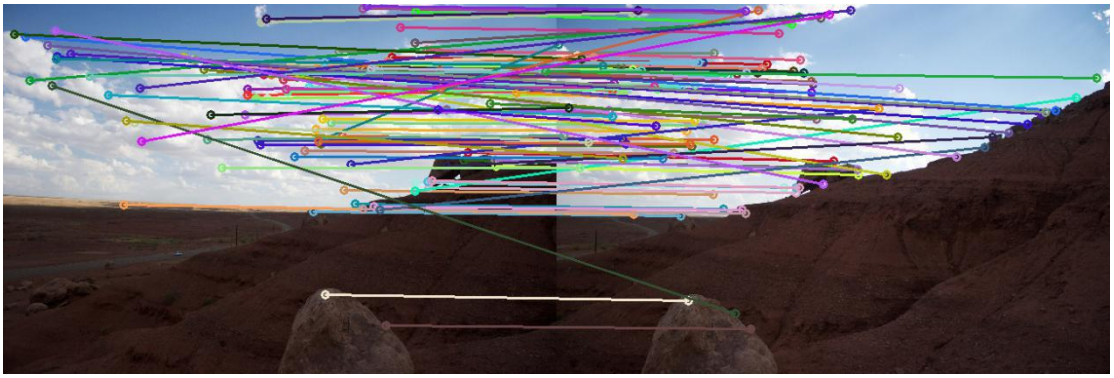
In my project, I used SIFT from opencv to detect and compute feature, and I use cv2.circle() to visualize the result, like below 2 pictures:



2. Keypoints Match or Feature Match

After finding keypoints/features, I need to match them point to point according their Euclidean distance of descriptor (128-degree

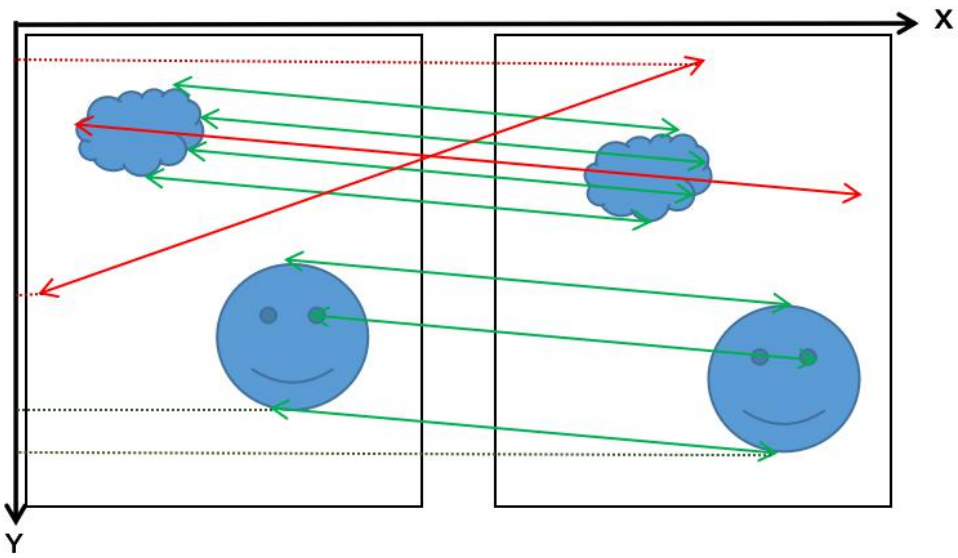
vector). In the left image(assuming we have already finished spacial arrangement, which will be discussed in part-5), for every keypoint in left image, say a1, we try to find another keypoint b1 in right image, with which a1 will have lowest Euclidean distance. And I also cross check to see if a1 is the lowest distance keypoint for b1, If it's, then we keep a1,b1's coordinate and their distance in my class(MyMatch); If it isn't, then we abandon keypoint a1, and this is implemented in my_match(). In order to finish the rest part, I also sort my matches in the order of non-decreasing Euclidean distance. And I use cv2.circle() and cv2.line() to visualize it, like below:



3. Use RANSAC to Remove Bad Matches

From the upper match image, we could find there are a lot of bad-match, which should be removed. We use the idea of approximate RANSAC to do it, we try to find as much as possible inliners/good-matches, and try to remove outliners/bad-matches. From the original match image, we could find that good-match

always have the same angle and length(line between 2 keypoints), but bad-match will have very different angle and length. In order to reduce computation problem, we use ‘xgap’ and ‘ygap’ to describe the angle and length(they are same thing), ‘xgap’ is the match line’s projection in the X-axis direction, ‘ygap’ is the match line’s projection in the Y-axis direction. As you can see it from below picture, the green line means good matches and red one are bad. You can also see that all the green line ‘xgap’ is almost equal and ‘ygap’ is almost equal too. But red line’s gap (project in axis direction) is very different, often very large, which you can find in below:



According this property, we can just find the most-frequent ‘xgap’ and ‘ygap’, so if one match’s/line’s gap is too larger or too smaller than it, we should consider it as bad batch. This is how my RANSAC works, and it work very well, you can see from below

picture, it abandon all bad batches, which is implement in my_ransac() function:

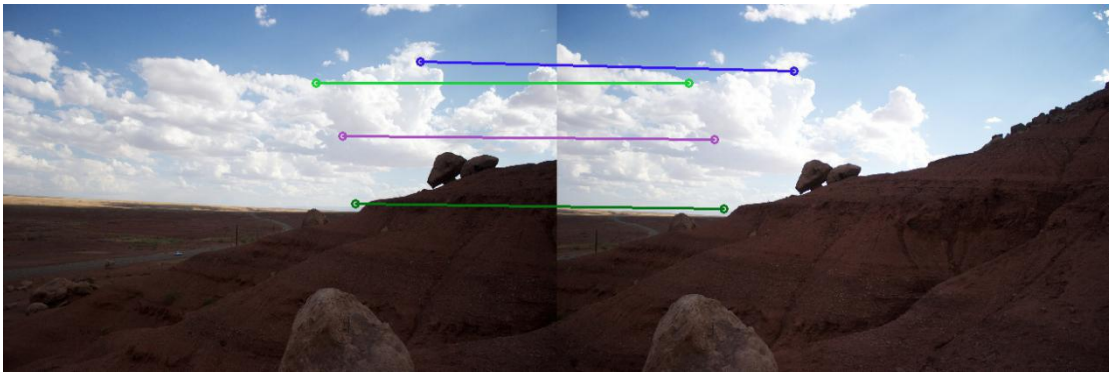
4. Use RANSAC to Compute Projection Matrix

Although we have remove bad matches, but we still need to find best match/keypoints in order to compute projection/homography matrix.

In this part, I also use the RANSAC algorithm to do it: assuming we have 500 good matches after removing bad matches, and then we orderly choose 4 matches from it, and use cv2.getPerspectiveTransform() to compute a temporary projection matrix H. And second, according this H, I transform the original right (or left) keypoints from good matches to expected left(or right) keypoints. And finally, we compare the Euclidean distance between expected left keypoints coordinate and real left keypoints coordinate. In my program, I will not compute all possible permutation, which is too slow and work not very well. I

will just compute at most 100000 permutation and combination of Homography matrix(which is very very enough) to find the best matches/keypoints with which we will have the lowest Euclidean distance between expected left keypoints and real left keypoints. And this is implemented in ransacChoise().

As you can see from below, we choose this 4 best matches to compute our final homography matrix:



5. Stitch Image with some tricks

This part is easy. After getting projection/homography matrix H, we use cv2.warpPerspective() to transform right image to left image's coordinate(or vise verse), but if we do it directly, we may lose some pixel in the top-right corner of right image, because some pixel in right image, after projection, may not fit in left image's coordinate. So before that, we need to combine the original right image shape and the homography matrix to compute a new

homography, which is implemented in my_perspective().

And now we need to stitch 2 image together, we cann't just put left image in left-position of projected-right image, because it will occur a crack between this two. However, we try to find the overlap part of 2 images(value in 2 images are all not (0,0,0)), and according the formula: result[i,j] = alpha*right[i,j] + (1-alpha)*left[i,j], alpha means the ratio of distance from this point to the begin-overlap in X-axis, this part is implemented in my_stitch(). And except the overlap part, if it's belong to left image, then the value is the same as left image; otherwise, it's belong to projected-right image. The result is very good, you can see it from below picture:

6. Decide Spacial Arrangement

Say we have 2 pictures: img1 and img2, we first detect SIFT feature, and then use the result of brute force match(part-2), if most matched-points are in the left then this picture is right picture, and vise versa. So if we can find relative position of 2 pictures, then of course we can find left, middle, and right in 3 picture: we just compare img1 to img2 and img3, if most matched-point for match1-2(img1 and img2) in img1 is in the left, then img1 is in the right of img2, and then we keep consider match1-3(img1 and img3), process are the same. This is implemented in find_position().

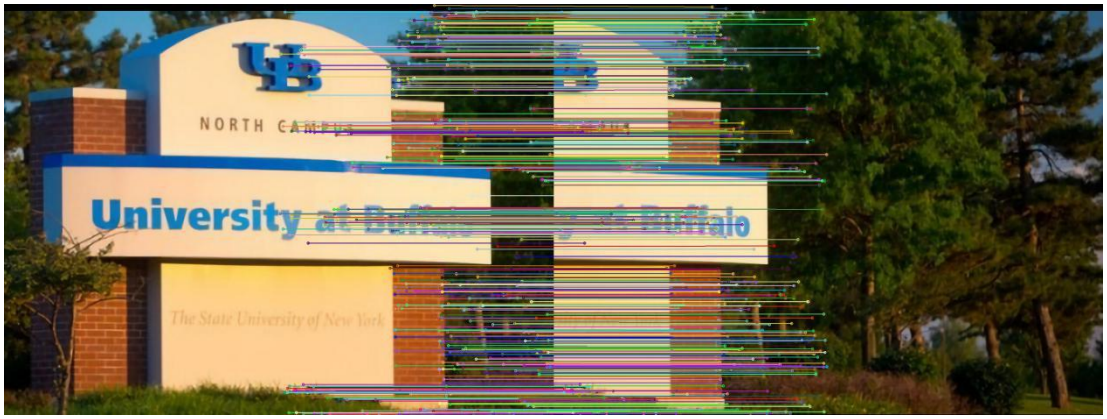
7. Something Else

The original image is too big, so I reduced it and saved them in [newdata] directory, you can see from my code zip file. And I also keep some middle-produced images in [process] directory, if you want, you can see the whole process, including:finding keypoints, brute force match, RANSAC, spacial arrangement, and so on. And finally, of course, you can change any directory, as long as this directory is in the same directory with [src], which directory I stored my stitch.py file. You can run command line like: python stitch.py data, python stitch.py newdata,python stitch.py ubdata. I strongly recomend you to run [newdata], which image is resized but look just like the original image. Below are some data from UB.

1.original images need to be stitched:

2.keypoints:

3.RANSAC match image:



4.matches we choose to compute projection matrix:

5.final result:

For more detail, you can see [process] directory.