CMSC 733
Project1 Report

Siqin Li
Department of Electrical and Computer Engineering
University of Maryland
Email: siqinli@terpmail.umd.edu

***Abstract*—This is a report of the Project 1: Myautopano. In this project, we implemented an end-to-end pipeline to do image panorama stitching of images. This report will describe the basic idea of the algorithm using in this project and the corresponding results.**

I. Introduction

This project can be separated by 5 steps. Sequentially, they should be cylindrical projection, corner feature detection, feature description, feature matching, Homography estimation using RANSAC and images blending. Since cylindrical projection is not an indispensable step, we will describe it after Homography estimation. According to the project description, the algorithm of first 4 steps are given, hence, we will quick go over them then elaborate the algorithm of image blending here. This algorithm is robust in the case that we need to blend a large number of images.

A. Feature Detection

In this step, our aim is to detect corners spread all across the image. First, we detect corner features using **cornermetric**. This function will give us the probability of every pixel being a corner features. Then we use **imregionalmax** to find out N_{strong} strongest corners. Lastly, by doing Adaptive Non-maximal Suppression(ANMS), N_{best} corners which are local maxima will be found out.The input of ANMS are corner score image and number of best corners needed. Here, we set $N_{best} = 800$. The output is a $N_{best} \times 2$ matrix and each row stores the coordinates of the best corner.

The corner feature score can be found in $Fig.1$.

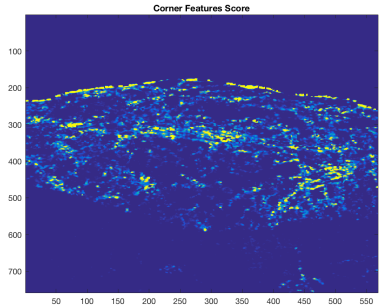

Fig. 1. Corner Feature Score

The local maxima feature points are shown in $Fig.2$

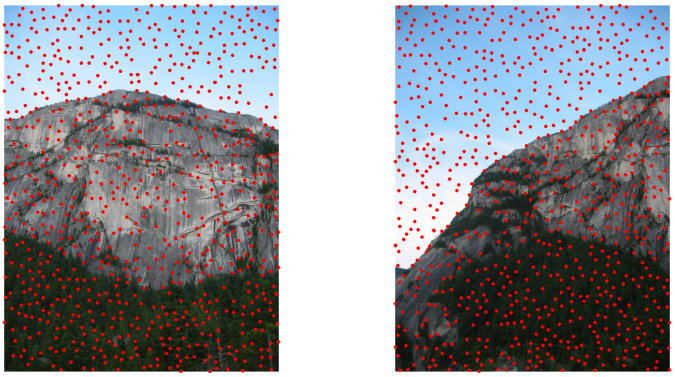

Fig. 2. Output of ANMS

B. Feature Descriptor

In this part, we describe each feature point by a feature vector. We first take a 40×40 patch around the keypoint. Then we apply a gaussian blur on the patch and down sample blurred patch to a 8×8 matrix. Later on, we just simply reshape the matrix to a column vector and standardize it to remove bias and illumination effect.

C. Feature Matching

After we encode each keypoint by a 64×1 vector in every image, now we need to match the feature points among a pair of images. The input of this function are feature descriptors and feature coordinate matrix from two images. First, we pick a feature point in image 1, then we calculate the sum of square distance (SSD) of feature descriptor with every feature point in image 2. Then we sort the SSD and set a $threshold = 0.8$ on the ratio of two lowest SSD to determine if we should keep the matched pair. If the ratio is below the threshold, we will regard the feature point with lowest SSD as the matched feature in image 2. Hence the output of this function are the coordinates of the match points in images. Noted that we also output the number of matched pair here in order to determine if this two images have enough matched point to do the homography estimation. If the number of matched pairs is less than 100, an error will show up and declare that this two images cannot be stitched together.

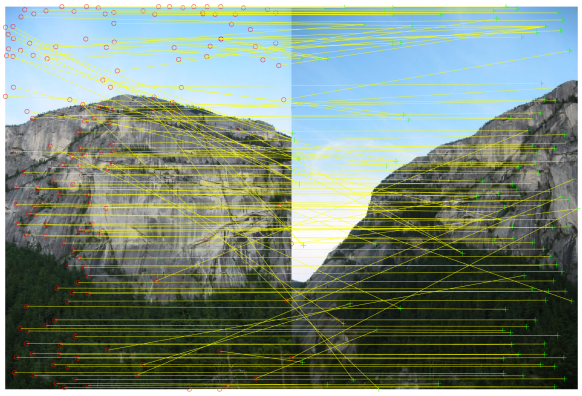

Fig. 3. Output of Feature Matching

D. RANSAC

After we get a set of matching pairs, we are going to select a subset of them to estimate a robust Homography. RANSAC algorithm provides a method to compute it. The input of RANSAC are the coordinates of matched pairs got from previous step, desired number of robust pairs (we used $percentage = 90\%$ here), a $threshold = 10$ representing the square distance between estimated feature location and actual feature location, and the maximum number of iterations $N_{max} = 5000$. We follow the steps: randomly selecting 4 feature pairs, computing exact the Homography H, computing the SSD between estimated points and actual point in image 2. Repeating this algorithm for N_{max} times to pick the Homography with the largest set of inliers or breaking out from the iteration once the number of pairs reached.

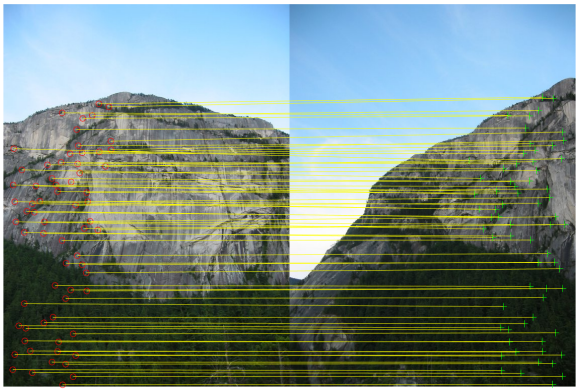

Fig. 4. Output after RANSAC

E. Cylindrical Projection

When we try to stitch images with translation, we find out that we should project the original image to a cylinder in order to get a better result. This is a pre-processing step before we do the feature detection. We first initialize a cylindrical projective image as the same size of the original image. Then just simply follow the projection equations to fill out the projective image by finding the corresponding pixel in the original image.

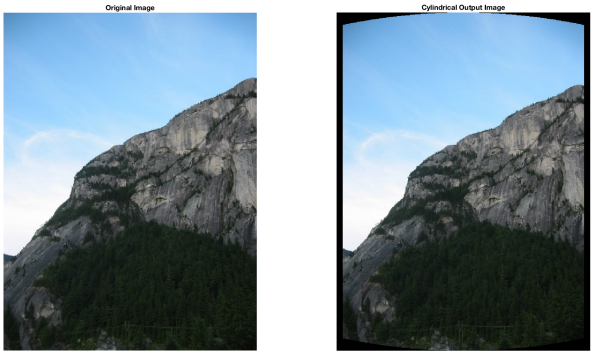

Fig. 5. Performance of Cylindrical Projection

F. Stitching and Blending

In this part, we do images stitching and blending alternately. The method of them will be described separately below.

1) Stitching: In the case that we want to stitch a lot of images, let's say image 1 to N, we can first stitch two of them as a panorama then stitch the third image with that panorama. However, this is not a good idea for feature matching and it is very time consuming. Therefore, we adopted another way to solve this problem. In this part, we assume that image 1 to N can form a panorama and image $N-1$ and image N can be stitched together. The algorithm is below:

a. Compute the Homography from image n to image $n-1$, $H_{n,n\text{-}1}$, where $n = 2...N$

b. Assign an image as the reference image (here we use the image $\lceil N/2 \rceil$ as reference). Compute the Homography from every image I_n to reference image I_{ref}, where $n = 1...N$. This can be done by simply multiplying the homography obtained in previous step.

c. Initialize a panorama image by computing the height and width using function **outputLimits**. Then paste image 1 to N into the panorama using the build in function **imwarp**. At the meantime, we do the image blending to blend the common region between images.

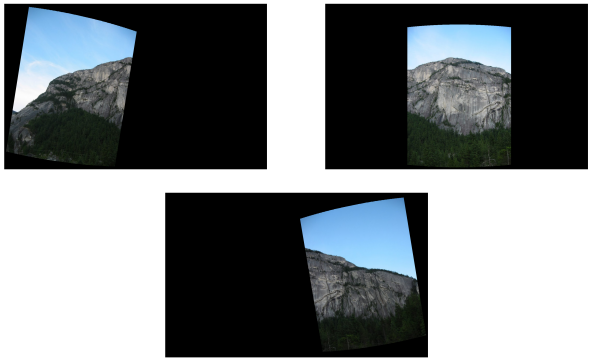

Fig. 6. Output of *imwarp*: Upper left: warped image 1; Upper right: warped image 2; Lower: warped image 3

2) Blending: To implement images blending, the intuition come to our mind is that we should give two weighted masks on the overlap region in both stitching images. But setting the weight is challenging. The idea here is that dealing with a pixel in the overlap region, we compute the distances from that pixel to image1 and to image 2 separately. For the image that the pixel is closer to, we will give a higher weight to it than the other. Another problem here is to determine the distance from a pixel to a image. To solve this, we adopt the basic idea from Breath-First Search (BFS) algorithm. Here, assuming overlapped image 1 and image 2, we first divide them into 3 regions. To simplify the explanation, let's define $mask1$ as the mask of image 1, $mask2$ as the mask of image2, $overlapMask$ as the overlapping region, $rest1$ = $mask1$ - $overlapMask$, $rest2$ = $mask2$ - $overlapMask$. Consider a pixel P, we define that the $distance_{P,1}$ from P to image 1 equal to the closest distance (smallest number of pixels) from P to the boundary of $rest1$. Hence, we can first find out all pixels around $rest1$ that with $distance_{P,1} = 1$ pixel. Then, mark those pixels who are also in the $overlapMask$ with a distance value $distance_{P,1} = 1$. Do this procedure iteratively by adding the distance until each pixels in $overlapMask$ is assigned by a value. Then we repeat the process on image 2. After we obtain the distance data, we can simply apply a $Softmax$ function on $distance_{P,1}$ and $distance_{P,2}$ to get a standardized $weightMask$. We can implement this idea by convolving $rest1$ with a 3×3 mask with all ones. The algorithm is:

Input: $mask1$ and $mask2$
output: $weightMask1$ and $weightMask2$
Compute the $overlapMask$, $rest1$ and $rest2$. Initialize a $reduceMask = overlapMask$ and a 3×3 all ones matrix $onesFilter$. Then do the for loop as follow:

for i = 1 : number of overlap pixels **do**
  if reduceMask not zero matrix **then**
    $rest1 = convolution(rest1, onesFilter)$
    set non-zero elements in $rest1$ to 1
    $stepMask = rest1\ AND\ reduceMask$ (do the logical operation)
    $distanceMask = distanceMask + i * stepMask$
    set $reduceMask = reduceMask - stepMask$
  else
    Break;
  end if
end for
Repeat the loop on $rest2$
Compute $weightMask$ by applying Softmax on $distanceMask1$ and $distanceMask2$

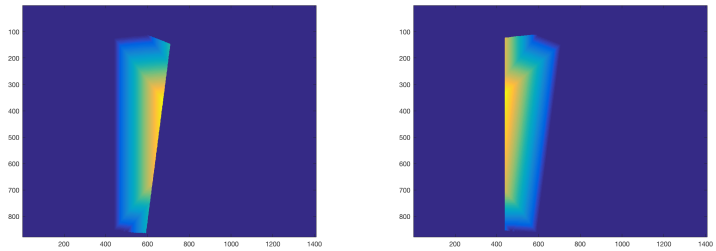

Fig. 7. Distance Mask: Left: distance mask of image 1; Right: distance mask of image 2. Note: the color change from cold to warm means the distance from small to large

Fig. 8. Result of Blending Image 1 and Image 2

G. Dealing with unordered input image

So far, we considered all of our input images are taken in the order. What if the inputs are unordered? We came up with a method to solve this problem. The idea is as follow:
For each image, we keep tracking on its reference image and the projective matrix with respect to the reference image. Initially, all images set their reference images as themselves and their projective matrix as identity. The algorithm will

terminate when all images set the first image as their reference image.

Repeat
Randomly pick a pair of images $(Image_i, Image_j)$, $i \leq j$
If $(Image_i, Image_j)$ do not match
go back to the beginning
else
for all the images whose reference image is $Image_j$,
let their reference image be the reference image of $Image_i$

II. Result and Discussion

Now, let's see the robustness of this algorithm.

A. Training set result

Fig. 9. Panorama of dataset 1

Fig. 10. Panorama of dataset 2

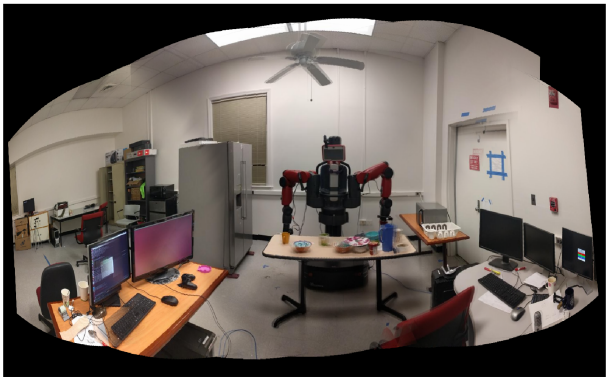

Fig. 11. Panorama of dataset 3

Fig. 12. Images of dataset 4

Fig. 13. Panorama of dataset 4

Fig. 14. Images of dataset 5

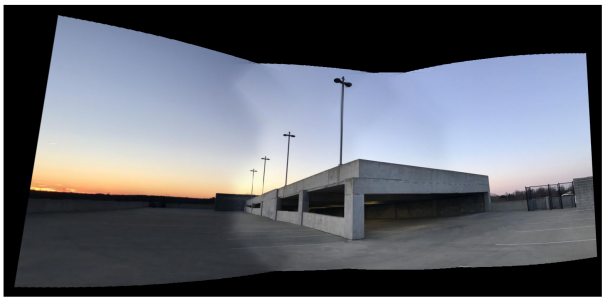

Fig. 15. Panorama of dataset 5

B. Testing set result

For testing set 1, images are very similar to each others. Doing cylindrical projection will destroy a lot of feature pairs. Hence, we did not do cylindrical projection here. Due to the similarity, most of the input pairs to RANSAC algorithm are not really matched, especially between image 2 and 3. So the result of RANSAC are not very acceptable. By adjusting the parameters of RANSAC, we finally got this result shown in Fig.16.

Fig. 16. Panorama of testing set 1

In the testing set 2, there are 9 unordered images. To do this, we came up with a randomly selection algorithm. Details of this algorithm is described in $Introduction$. Also, there are two images are absolutely the same. We also consider this situation in our blending algorithm. When two images are totally the same or one image cover the another one totally, we will pick either one or the bigger image then simply paste it on the panorama frame. Since blending is not in the order, for the case that two image do not have any overlap, we will paste both of them on the panorama. The result is shown in Fig.17.

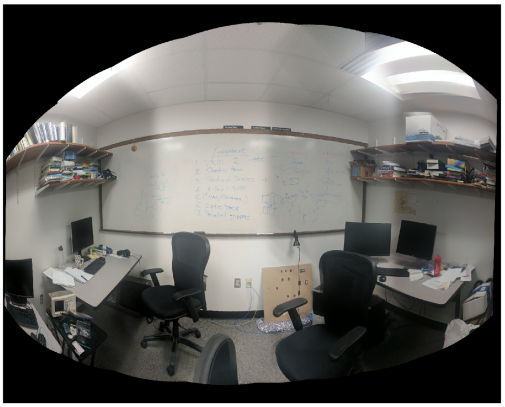

Fig. 17. Panorama of testing set 2

For testing set 3, adjusting the focal length and the threshold of RANSAC gives us a very nice result shown in Fig.18.

Fig. 18. Panorama of testing set 3

For testing set 4, since there are 2 images that cannot be matched to any other images, our algorithm will return an

error says ' Some images are not included in the dataset'. To further improvement, we should throw the irrelevant images and stitch the rest as a panorama.

C. Discussion

As we see, for dataset1,3,4, this algorithm performs pretty well. Images can be blended seamlessly. For dataset 2 and 5, it can still estimate the homography very well even though the images do not have many corners. However, this algorithm cannot do the brightness compensation. If images between each other have obviously different brightness, the blending region will be detected easily.

III. Conclusion

This project is very interesting and challenging. Blending two images is not a trivial work. And also, to deal with the unordered image set, there are still lots of improvement needed to be done. This pipeline of panorama is not robust for a dataset having many unordered images. It will take a long time to do matching and lots of parameters need to be adjust in order to get an acceptable performance.

Acknowledgment

The authors would like to thank Jiahao Su who points out the blending algorithm and randomly image ordering. The way to determine the distance between pixel and image is suitable for this case. And randomly selecting images and ordering do save much more time than simply computing Homography between every pair of images.

References

[1] Brown, Matthew, and David G. Lowe. "Automatic panoramic image stitching using invariant features." International journal of computer vision 74.1 (2007): 59-73.