CMSC733 Project1: Myautopano!

Xiaoxu Meng
Department of Electrical and Computer Engineering
UMCP
Email: xiaoxumeng1993@gmail.com

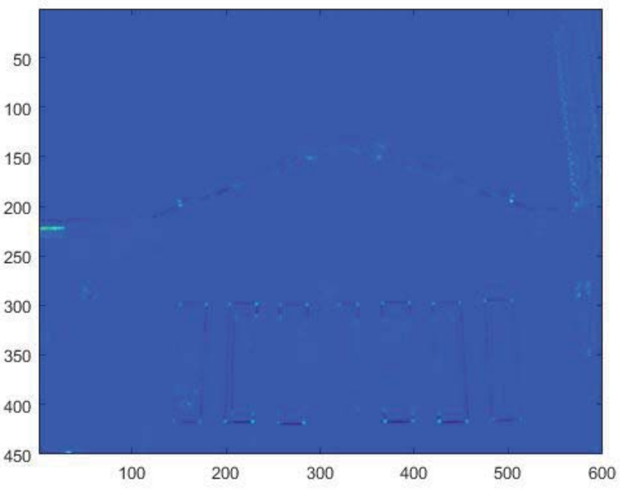

Fig. 1: corner

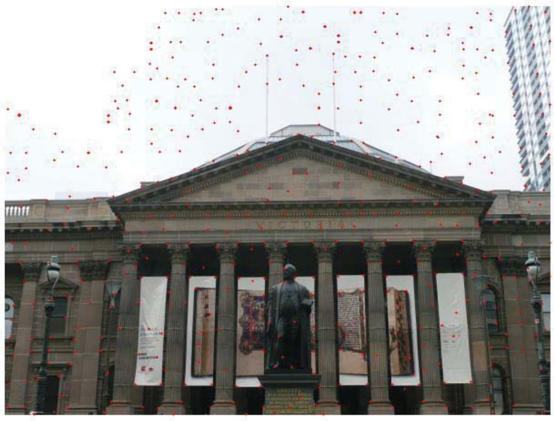

Fig. 2: Best corners (1)

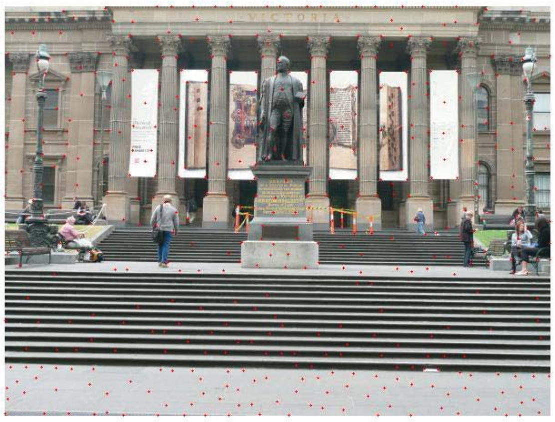

Fig. 3: Best corners (2)

***Abstract*—The aim of this project is to implement an end-to-end pipeline to do image panorama stitching of unordered images.**

I. Algorithm

A. Capture Images

Simply capture a set of unordered images.

B. Find the Corners

The aim of this step is to detect corners spread all across the image to avoid weird artifacts in warping. Implement the following steps for ANMS:

1) Detect corner features: Detect corner features using cornermetric(). The output is a matrix of corner scores. Visualize the output using imagesc(). The result is shown in Fig. 1.

2) Find Strong Corners: Find the Nstrong strongest corners using the Matlab function imregionalmax(). More than 3000 "corners" could be found, some of which are not real corners. Then ANMS algorithm is used to choose the best corners in the image. I chose the strongest 490 corners and the results are shown in Fig. 2, Fig. 3, Fig. 4. f

C. Feature Descriptor

Next step is to describe each feature point by a feature vector. Take a patch of size 40 x 40 centered (this is very important) around one corner point. Then apply gaussian blur. Now, sub-sample the blurred output to 8 x 8. Then reshape to obtain a 64 x 1 vector. Standardize the vector to have zero mean and variance of 1.

D. Feature Matching

Pick one point in image 1, compute sum of square difference between all points in image 2. Take the ratio of best match

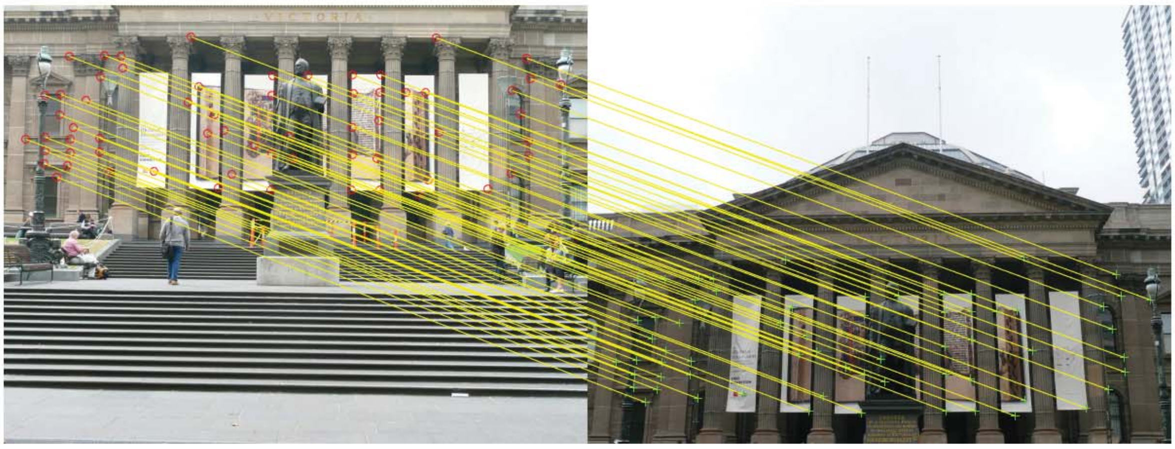

Fig. 5: Match result (1)

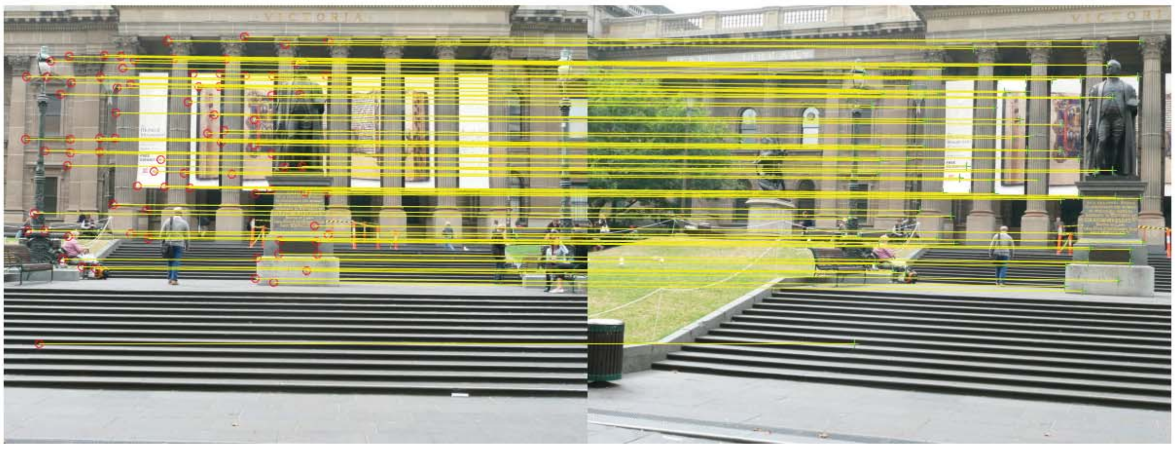

Fig. 6: Match result (2)

(lowest distance) to the second best match (second lowest distance) and if this is below some ratio keep the matched pair or reject it. Repeat this for all points in image 1. I am left with only the confident feature correspondences and these points will be used to estimate the transformation between the 2 images also called as Homography. Finally use the function dispMatchedFeatures() to visualize the result as shown in Fig. 5 and Fig. 6.

E. RANSAC to estimate robust Homography

Use RANSAC to compute homography. The steps are:

1) : Select four feature pairs (at random), pi from image 1 , p1i from image 2.

2) : Compute homography H (exact). Use the function est homography given to you.

3) : Compute inliers where SSD(p1i;Hpi) ¡ thresh. Here, Hpi computed using the apply homography() function given to you.

4) : Repeat the last three steps until you have exhausted Nmax number of iterations (specified by user) or you found more than a percentage of inliers (say 90% for example).

5) : Keep largest set of inliers.

6) : Re-compute least-squares H estimate on all of the inliers. Use the function est homography() given to you.

F. Cylindrical Projection

When the FoV is large, to overcome the distortion problems at edges, we will be using cylindrical projection on the images before performing other operations. An comparison between the image and the cylindrical image is shown in Fig. 7 and Fig. 8. For this project. We don't need cylindrical projection for Set1 and Set2. But for Set3, cylindrical projection is necessary

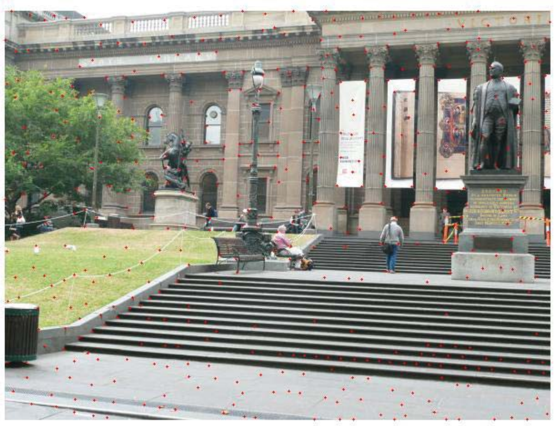

Fig. 4: Best corners (3)

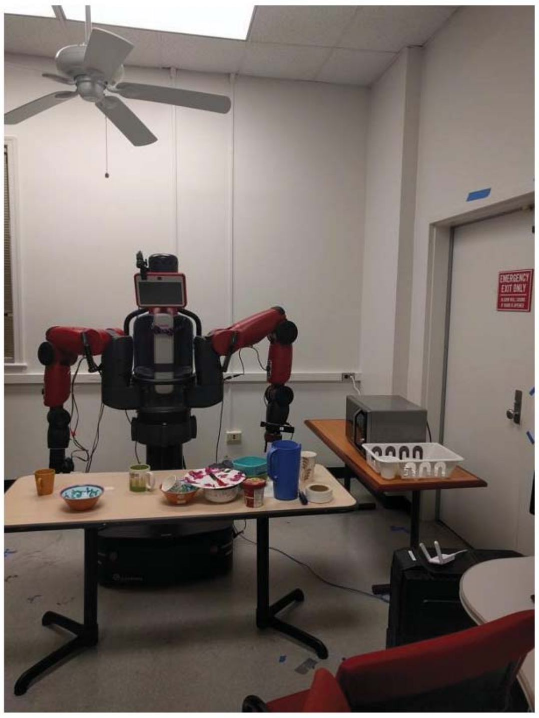

Fig. 7: Original image

because FoV is large. However, I still use the original images to find the inliers, perform cylindrical projection for the inliers and use the projected inliers to calculate homography H.

G. Blending images to get a seamless panorama

To blend the image, I firstly perform transformation for the images. Instead of using Guassian filter, I use the nearest-neighbor algorithm to fill the black lines of the transformed image. When transforming, I calculate the offset of coordinates

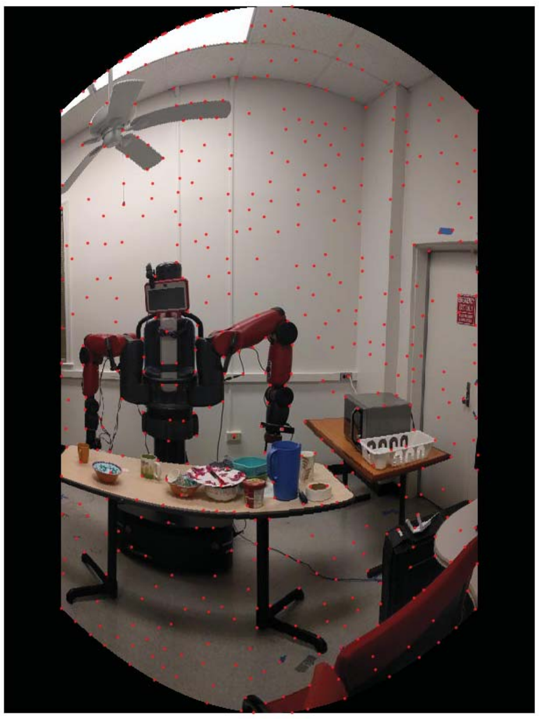

Fig. 8: Cylindrical image

of different images. And apply this offset when blending the images. Then I build a mask to mark where the pixel is not black. Add the mask of all the images to get where the pixels overlap. The final image is the sum of all the images by the mask. Results are shown in Fig. 9, Fig.10 and Fig. **??**. Results for custom sets are shown in Fig. 12, Fig.13. Results for test sets are shown in Fig. 14, Fig.15, Fig.16 and Fig. **??**.

H. Analysis and Problems

Most of the results look good and TestSet4 could report an error. Some of the overlapping part is a little bit blurred because of pixel-level error. The result of TestSet1 is not as good as others because the grid is too dense. One problem I have met is that when generating result image, the overlaping parts are always gray as shown in Fig. 18. The reason is that I used a mask to calculate the number of overlap. When I perform color = (sum of color) / (number of overlap). The color format I used is uint8. However, the sum of uint8 has a upper bound of 255, thus causing the wrong color.

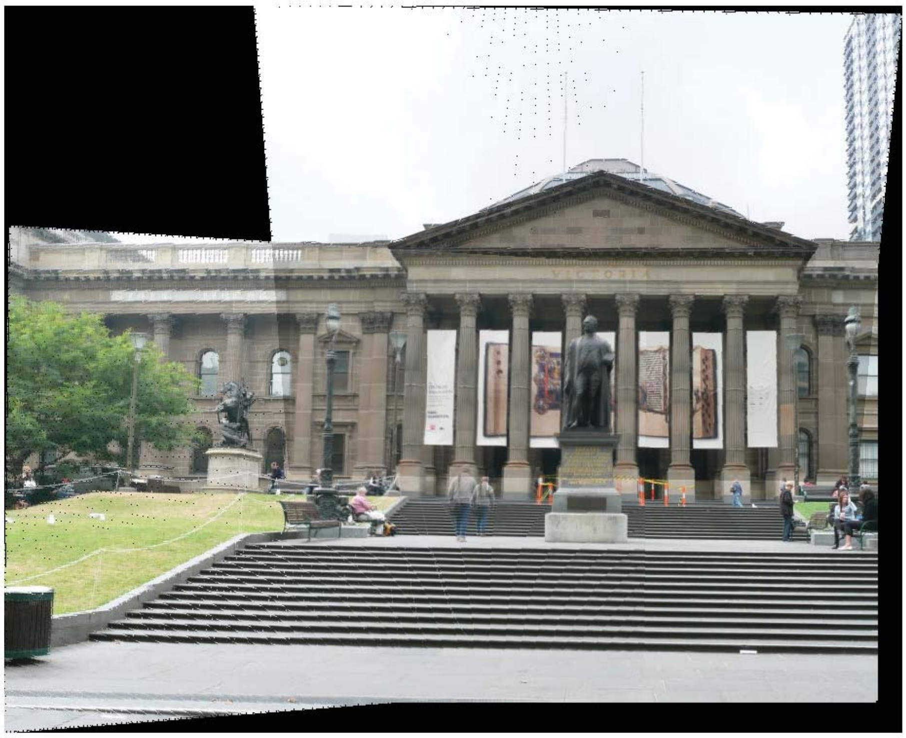

Fig. 9: Result of Set1

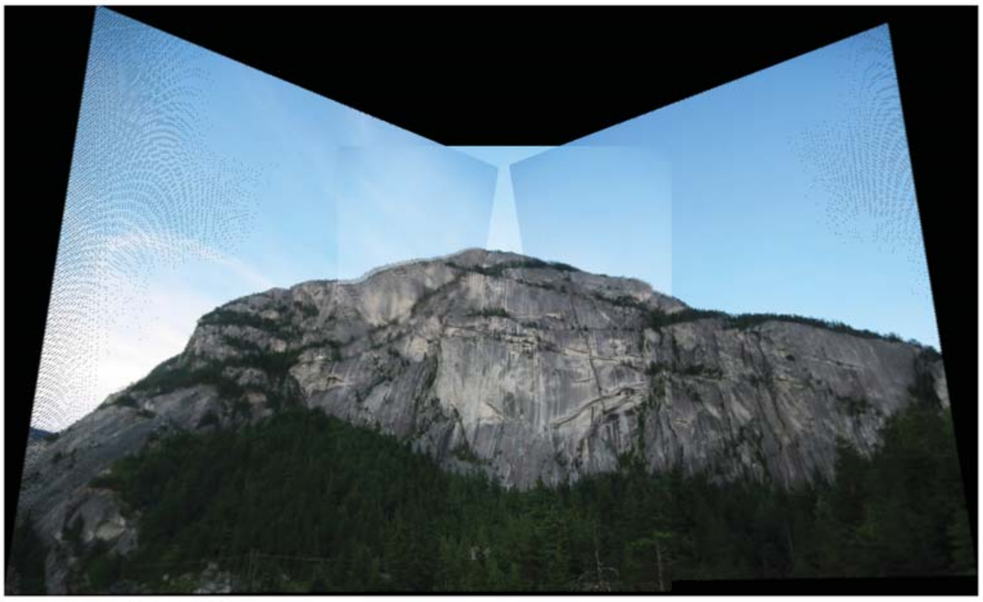

Fig. 10: Result of Set2

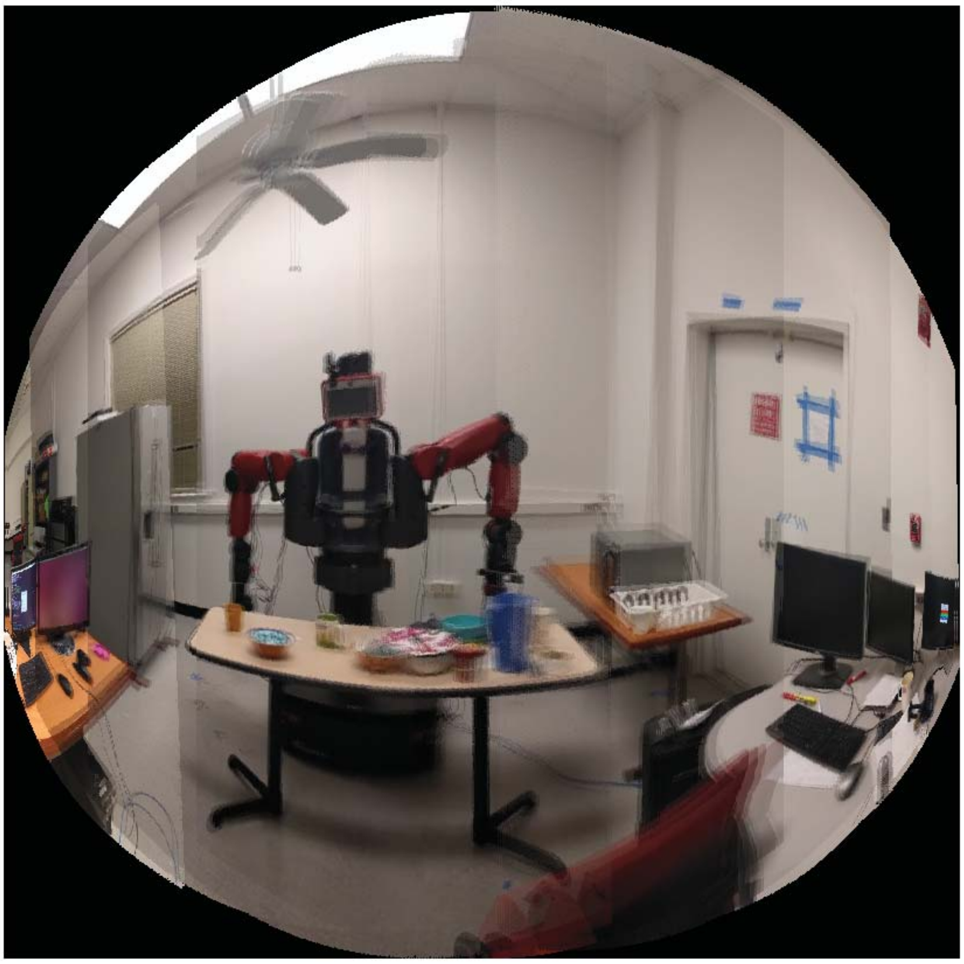

Fig. 11: Result of Set3

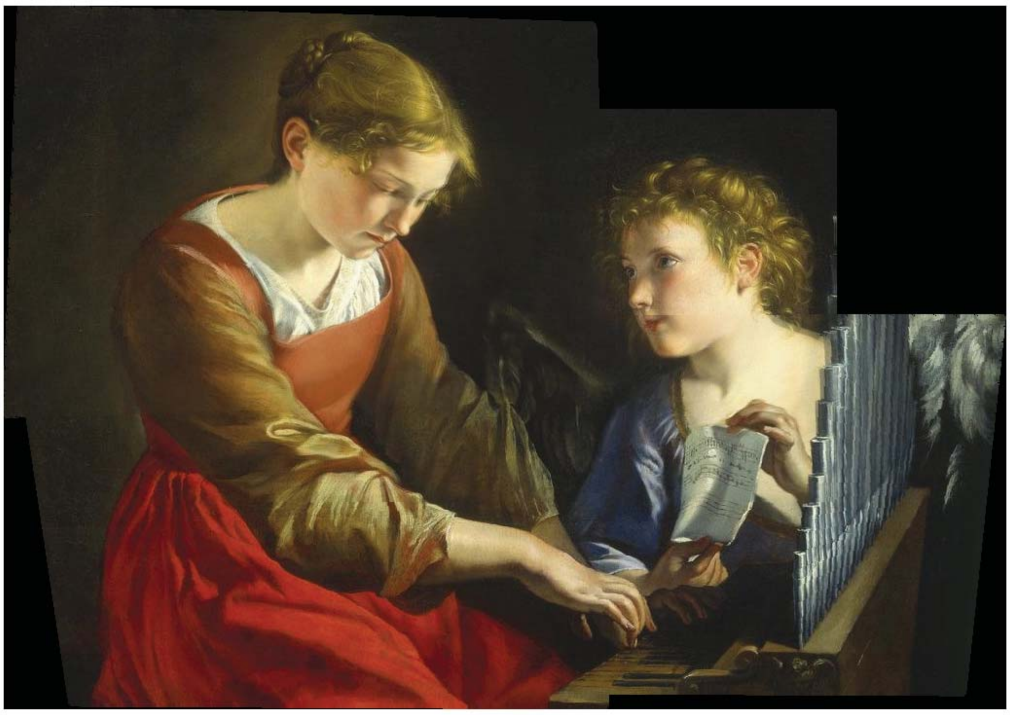

Fig. 12: Result of Custom Set1

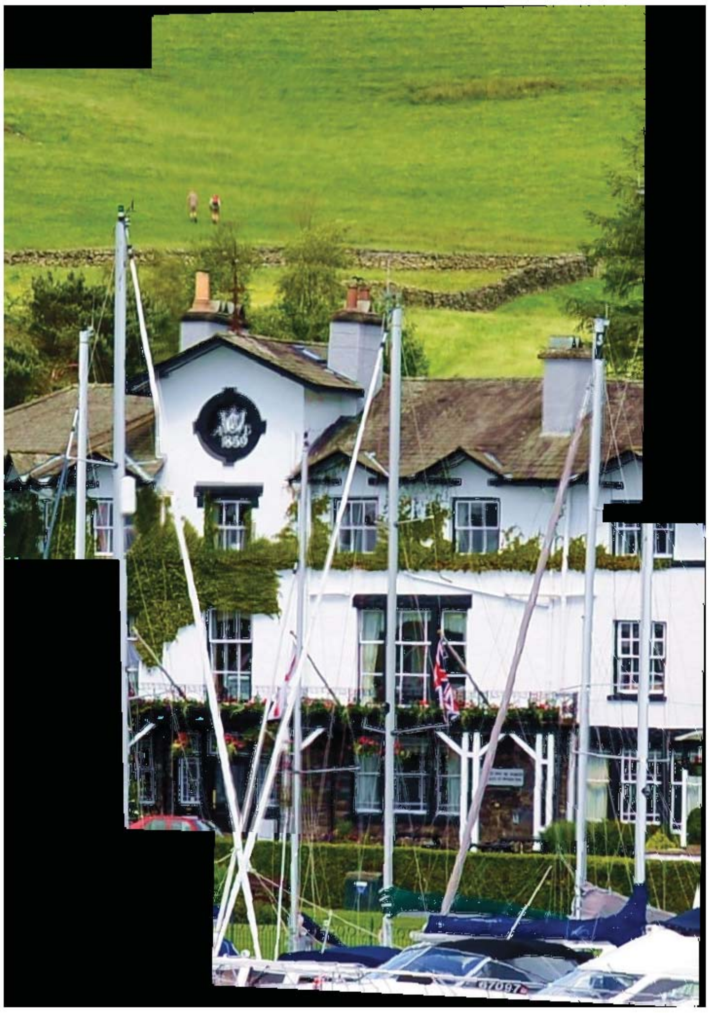

Fig. 13: Result of Custom Set2

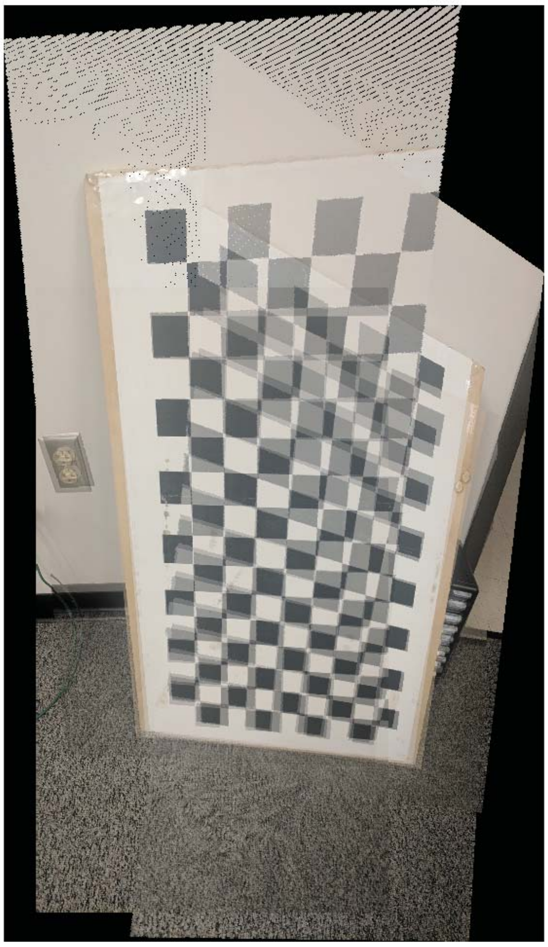

Fig. 14: Result of Test Set1

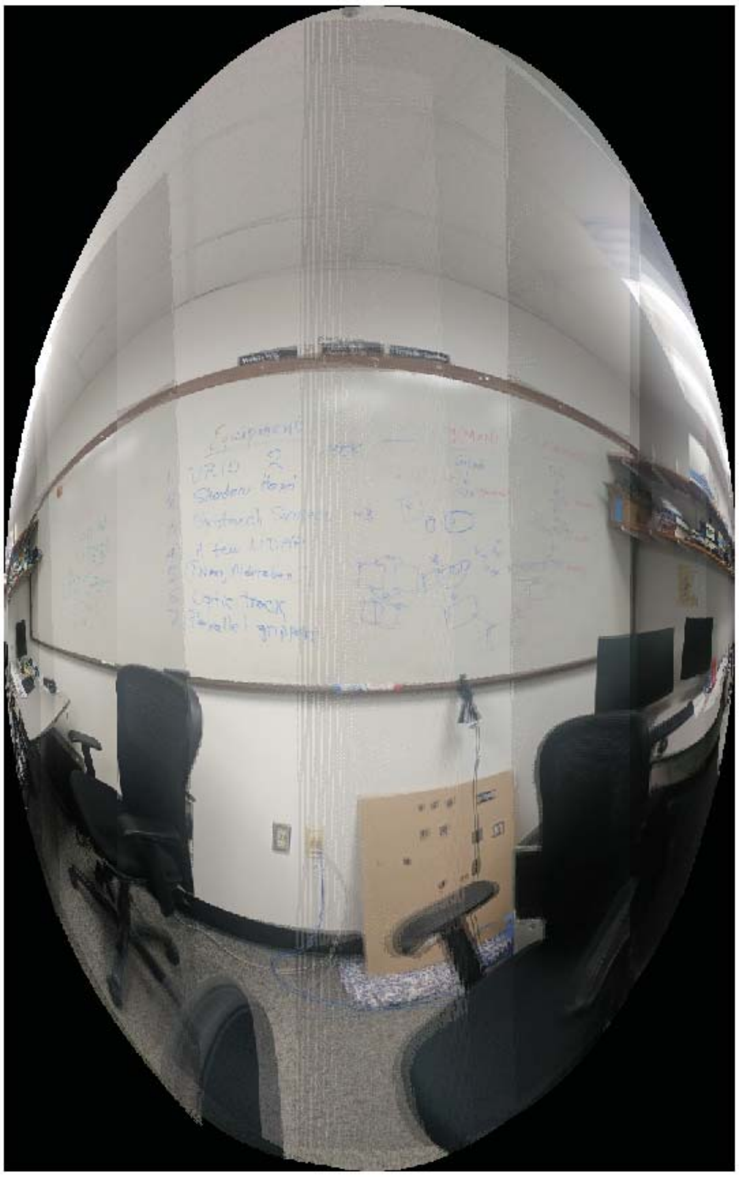

Fig. 15: Result of Test Set2

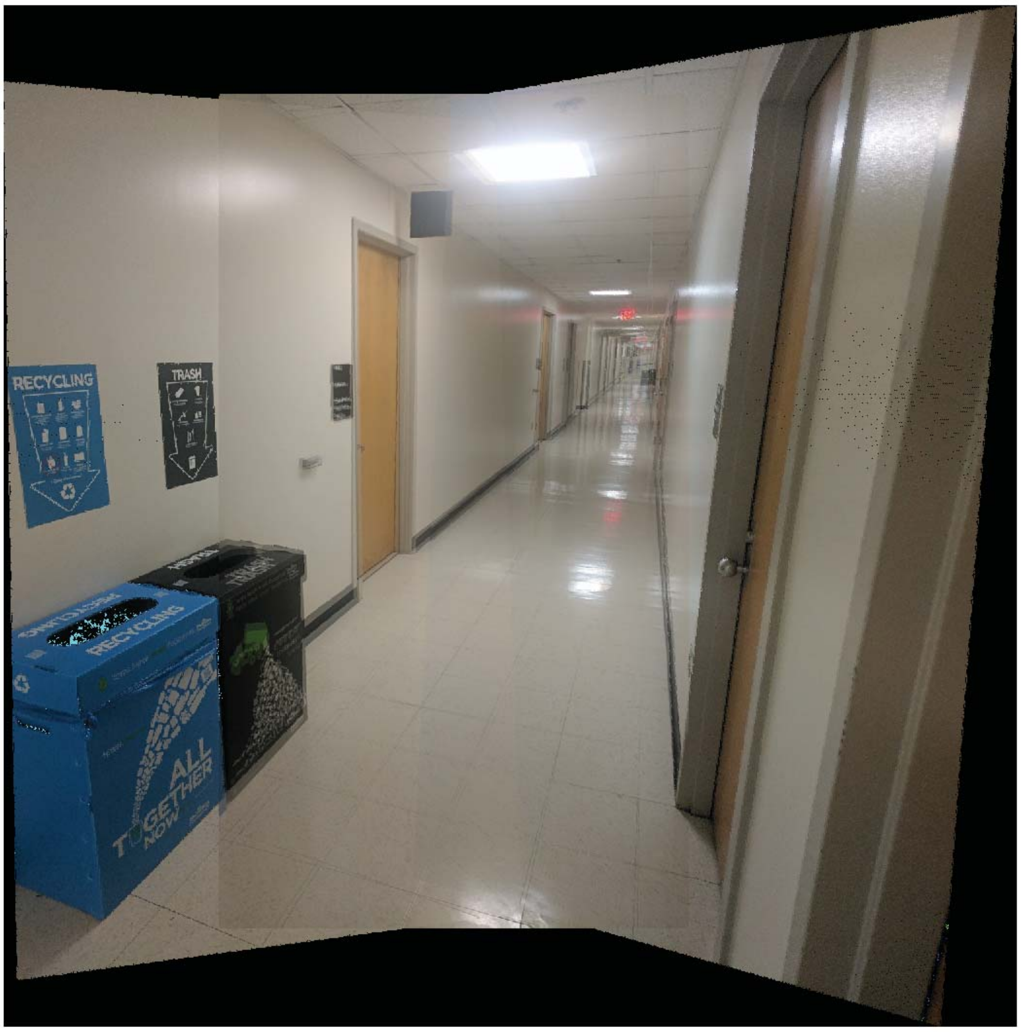

Fig. 16: Result of Test Set3

Fig. 17: Result of Test Set4

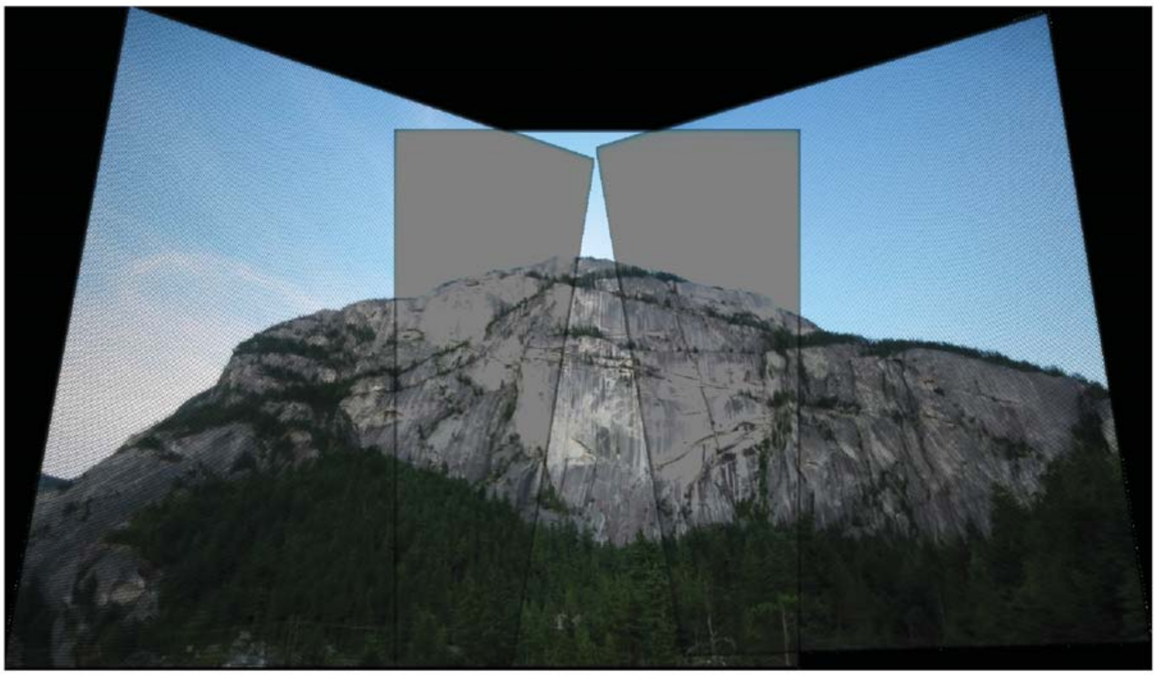

Fig. 18: Problem