Project 1
My Autopano

Lih-Narn Wang
Master of Engineering in Robotics
University of Maryland, College Park
Email: ytcdavid@terpmail.umd.edu

I. Phase1: Traditional Approach

In this section, we'll present a pipeline that can merge multiple images without knowing there orders. Our approach can also reject irrelevant images. Basically, We followed the traditional approach, which can be divided into 6 steps: 1) Use corner detection to locate feature points. 2) Use adaptive non-maximal suppression to get uniformly distributed feature points. 3) Construct the feature descriptors. 4) Match the feature points based on the feature descriptors. 5) Use RANSAC to refine the matches and extract robust homography. 6) Warp and blend the image. Besides these steps, we added an additional step at the head of the pipeline, which basically pairs the image and rejects the outliers. The details and results are presented in the following sections.

A. Corner Detection

The first step is to locate the feature points. Because corners usually encode rich information, we used corner detection to locate them. We used two corner detection algorithms, Harris and Shi-Tomasi corner detection. The example results are shown in figure 1&2. We chose Shi-Tomasi's results to develop the rest of the projects.

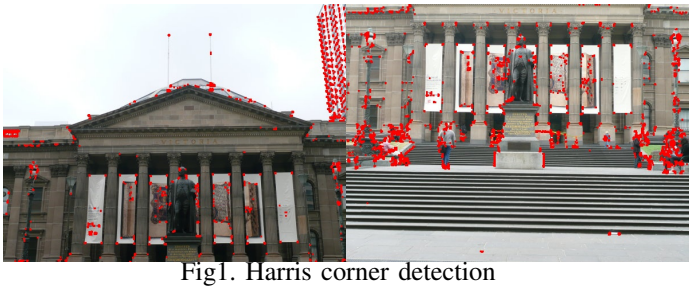


Fig1. Harris corner detection

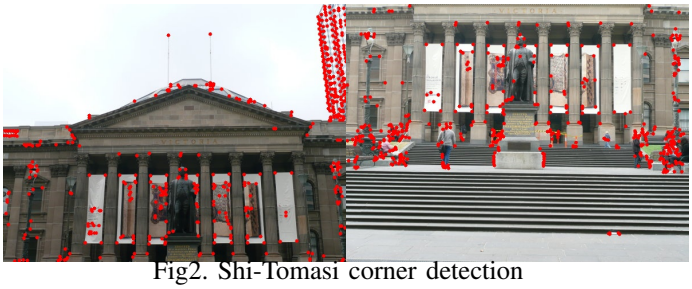


Fig2. Shi-Tomasi corner detection

B. Adaptive Non-Maximal Suppression

Because there may be multiple feature points concentrated at a relatively small area, we applied ANMS to make them evenly distributed. By doing this, we can have a better representation of the image and a higher accuracy in matching features. The pseudo code of the algorithm is provided in the reference[1], and for the purpose of the completeness, it's also shown in figure 3. The example results after applying ANMS are shown in figure 4.

Input : Corner score Image (C_{img} obtained using `cornermetric`), N_{best} (Number of best corners needed)
Output: (x_i, y_i) for $i = 1 : N_{best}$

Find all local maxima using `imregionalmax` on C_{img};
Find (x, y) co-ordinates of all local maxima;
((x, y) for a local maxima are inverted row and column indices i.e., If we have local maxima at $[i, j]$ then $x = j$ and $y = i$ for that local maxima);
Initialize $r_i = \infty$ for $i = [1 : N_{strong}]$

```
for i = [1 : N_strong] do
    for j = [1 : N_strong] do
        if (C_img(y_j, x_j) > C_img(y_i, x_i)) then
            ED = (x_j − x_i)^2 + (y_j − y_i)^2
        end
        if ED < r_i then
            r_i = ED
        end
    end
end
```

Sort r_i in descending order and pick top N_{best} points

Fig3. Adaptive Non-Maximal Suppression

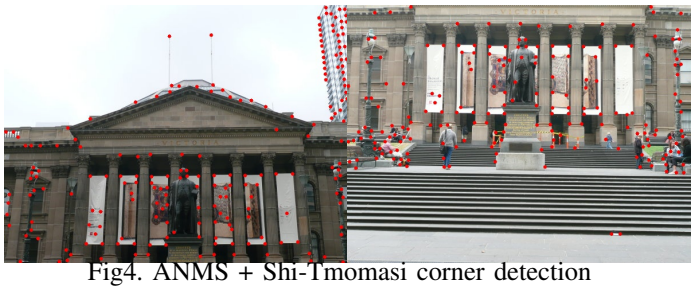


Fig4. ANMS + Shi-Tmomasi corner detection

C. Feature Descriptor

Before we matched the features, we had to construct a descriptor for each of them. Then, we can quantify the relativity between features based on their descriptors. In this project, we took a patch of the image that centered at each features. We blurred the patch and sub-sample it into $8 * 8$.

Finally, we reshaped it into a $64*1$ vector, which is the feature descriptor.

D. Feature Matching

With the feature descriptors, we can match the features. We followed the traditional pipeline. Firstly, we calculated the relativity between a point in image 1 to all points in image 2, using the L2-Norm of the difference of their descriptors. Then, we calculated the ratio of the largest relativity to the second largest relativity. If the ratio is bigger than the threshold value, we considered it a good match. The example result is shown in figure 5.

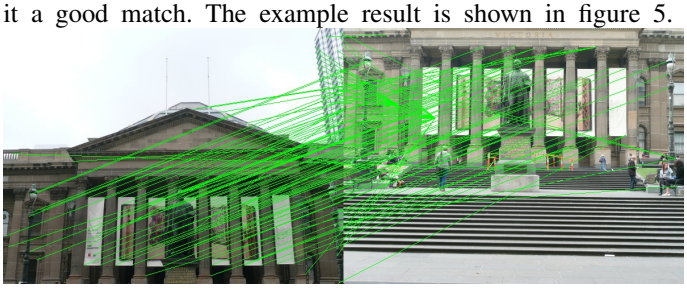


Fig5. Feature matching between two images

E. Random Sample Consensus

As you can see, there are some good matches along with more bad matches, which could be viewed as outliers. As a consequence, we had to remove the outliers and refine the matches. There's a classical algorithm, known as RANSAC, that removes the ouliers efficiently. The pseudo code of the algorithm[1] is shown in figure 6, and the example result is shown in figure 7. The details of the implementation are as follows:

1) Outlier Rejection: Firstly, we picked 8 points randomly to estimate the homography between the feature points. Because a good match means that they are the same point, if we warped the features of one image, the good matches should have very small difference. Therefore, we can use this fact to decide whether it's a good match or an outlier. We repeated these steps for about 20000 times, then we rejected almost all bad matches. The results are shown in ...

2) Robust Homography Estimation: After we rejected all the outliers, we can use all the good matches and least-square method to estimate a robust homography. In this way, we can minimize the error due to the perspective transform and produce a seamless blending image.

1. Select four feature pairs (at random), p_i from image 1, p_i' from image 2.
2. Compute homography H between the previously picked point pairs.
3. Compute inliers where $SSD(p_i', Hp_i) < \tau$, where τ is some user chosen threshold and SSD is sum of square difference function.
4. Repeat the last three steps until you have exhausted N_{max} number of iterations (specified by user) or you found more than some percentage of inliers (Say 90% for example).
5. Keep largest set of inliers.
6. Re-compute least-squares $\hat{H}$ estimate on all of the inliers.

Fig6. RANSAC algorithm

Fig7. Robust feature matching using RANSAC

F. Blending Image

In this section, we proposed two approaches that blend the images. For simplicity, we only discuss the case that has two images, img_1 and img_2, and a homography H_{12} that warps img_1 to img_2. Firstly, we need to calculate the size of img_1 after the warping. So we took the four corners of the image, and applied homography to calculate the new four corners. Then, we constructed another homography matrix H_{offset} that offset the warping image so that every points in the image can be shown in the frame. Due to linear algebra, we produced an image img_{warp} that contains all the pixels in img_1. An example result is shown in figure 8.

$$img_{warp} = H_{offset} * H_{12} * img_1$$

Then, based on the coordinates of the matches, we calculated the size of the blending image and enlarged the img_{warp} as $img_{template}$so that it could merge with img_2. The example result is shown in figure 9. As you can see in the figure, we extra black space is the space for img_2 that we calculated before we do any blending operations.

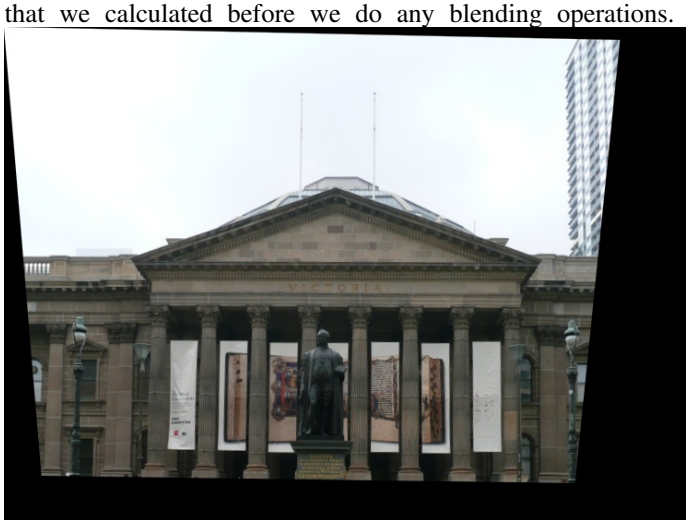


Fig8. img_{warp}

Fig9. $img_{template}$

1) Naive Approach: In the naive approach, we simply pasted img_2 onto $img_{template}$. The example result is shown in figure 10. We are not satisfied to this result. As you can see, there's a huge difference in brightness around the edge, so we proposed another approach.

Fig10. Merge result of the naive approach

2) Seamless Approach: In this approach, we used $cv2.seamlessClone()$ function, which applied the poisson filter. However, we can't just use this function to clone img_2 onto the template, because the poisson filter would try to smooth all the second order differential of the pixel values. There's a large black area, so the whole image would be pretty dark. The example output of the failure using the function is shown in figure 11. On the other hand, we cut the img_2 so that it won't overlap the $img_{template}$ and paste it onto the template. The example result is shown in figure 12. Then, we use the function to clone the whole img_2 onto the $img_{template}$, which no longer has black areas. The example output is shown in figure 13.

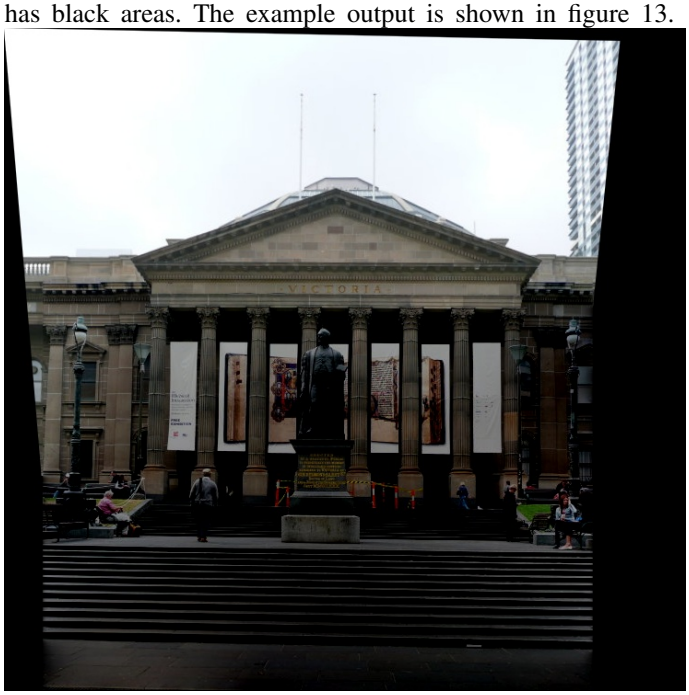


Fig11. Failure using poisson blending

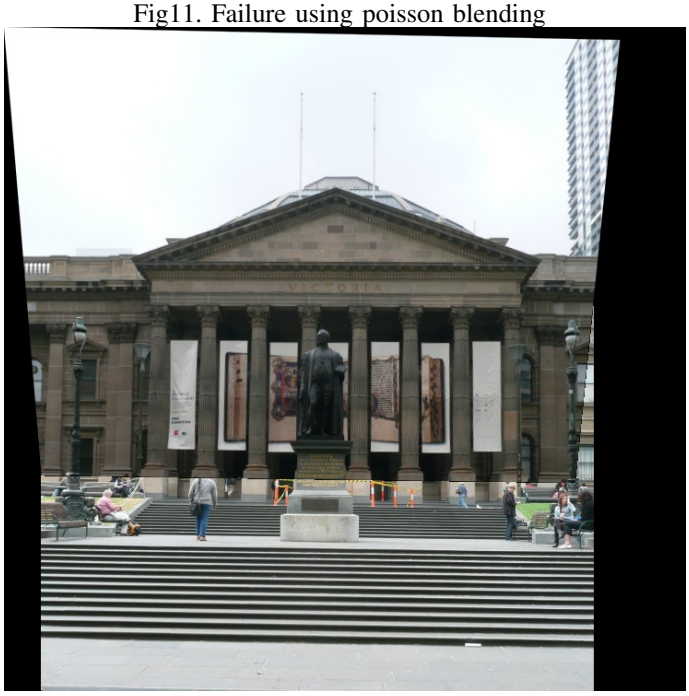


Fig12. Cut and paste the img_2

Fig13. Final output of merging img_1 and img_2

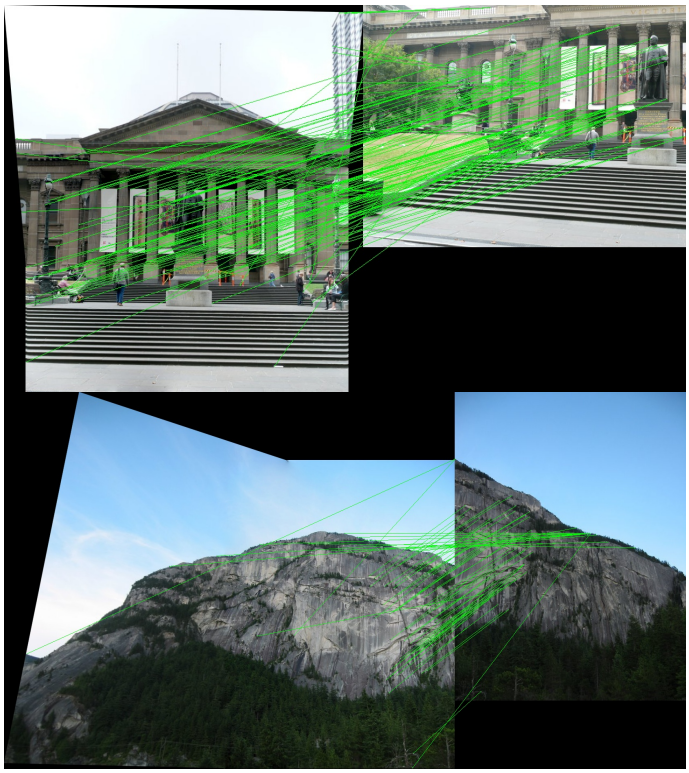


Fig15. Matching features

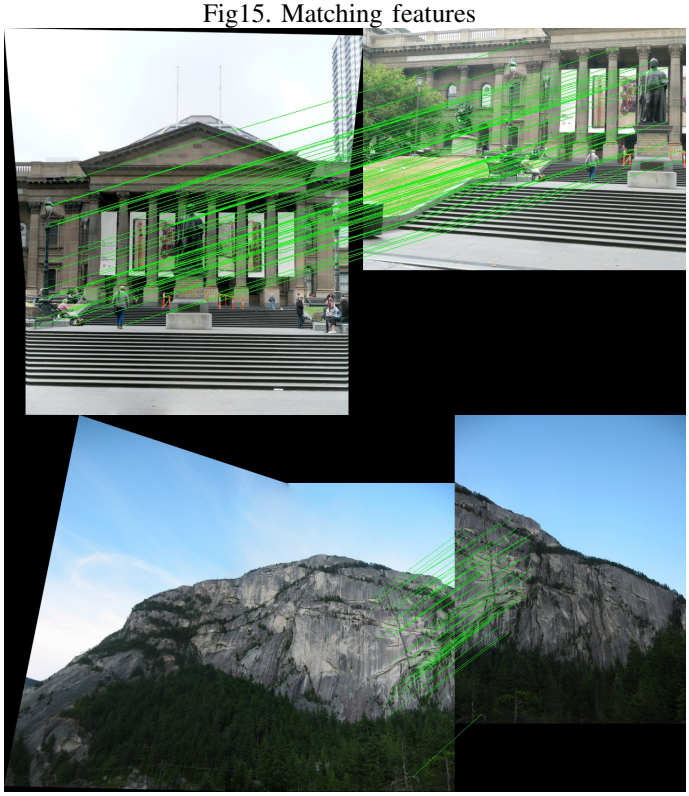


Fig16. Robust matching feature

G. Generalization

After merging two images successfully, our next step is to generalize the algorithm into merging multiple images. Our first approach is simple and efficient, which achieved great results on 80% of the training sets. We first picked two images, img_1 and img_2 to blend. We used the algorithm described above to produce img_{merge}. Then, we used this image as the img_1 and picked another image as img_2. Finally, we looped over the steps until every images is merged into a single image. The procedures is illustrated in figure 14, and the results of each steps shown in figure 15, 16&17.

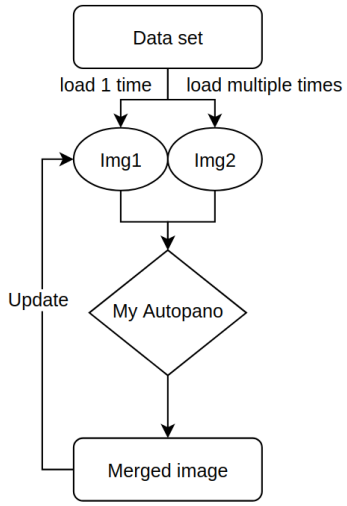


Fig14. Generalized algorithm

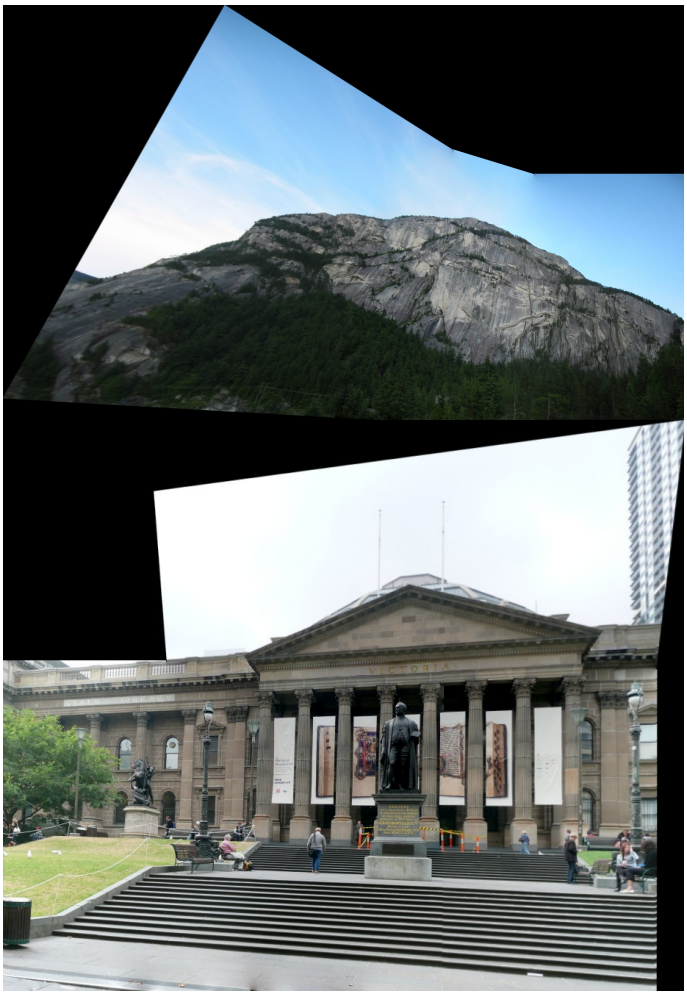


Fig17. Final result of the training set1&2

The pipeline above seems to be robust and performed well. However, We ran into a lot of troubles doing the set_3 in the training sets. After a closed examination, we believed that the set_3 is hard for this approach due to a simple reason: 1) The angel between the first image and the last image. However, to solve this problem, another problem arises: 2) The order of the images. And in the process of solving these two problems, we found and solved the last problem: 3) Some images may be irrelevant. The pipeline of the new approach is illustrated in figure 18.

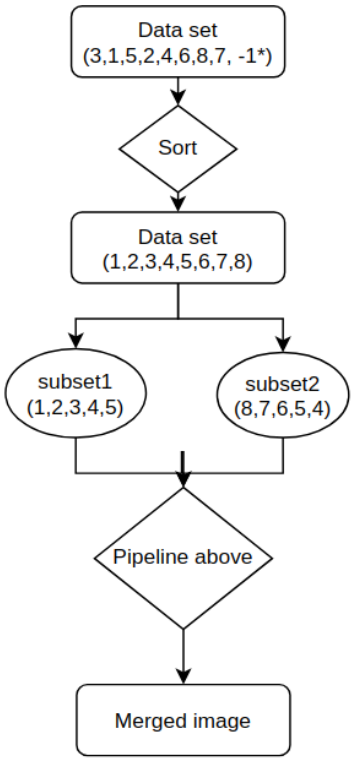


Fig18. Generalized robust algorithm

1) Angles: In the previous strategy, we warped the first image into the angle of the second image. Then, we warped the merged image into the angle of the third image. However, if the angles changes a lot from the first image to the last image, we may warp the image too far that the first image part is at the back of the camera view. So, the best strategy is to warp the image to the middle of all the images. However, to achieve this, we'll have to calculate the order of the images, which we'll explain in the next section.

2) Orders: To find the orders of the image, we need to determine the relationship between images. To achieve this goal, we constructed a table that saved the number of matching points between each images. Then, we observed that the image at far left or far right has only one significantly higher number of matching points. After we located the image at the edge, we used back-tracking algorithm to track which image is overlapped with the edge image to construct the orders. In the three training sets and two custom sets, the algorithm achieved 100% accuracy, so we believed it's robust enough.

3) Outliers: Lastly, we can us the algorithm above to reject outlier images. Intuitively, if an image has a number of matching points lower than a threshold value between all images, we treated it as an outlier and reject it. As a conclusion, we took two opposite ways to merged a sequence of the images, so that the angle of the camera of the output would be at the center of the image. In this way, we can merged a wider range of images. To illustrate the steps, we took set3 in the training sets as an example. The steps are shown in figure 19 22.

Fig19. First merge

Fig20. Second merge

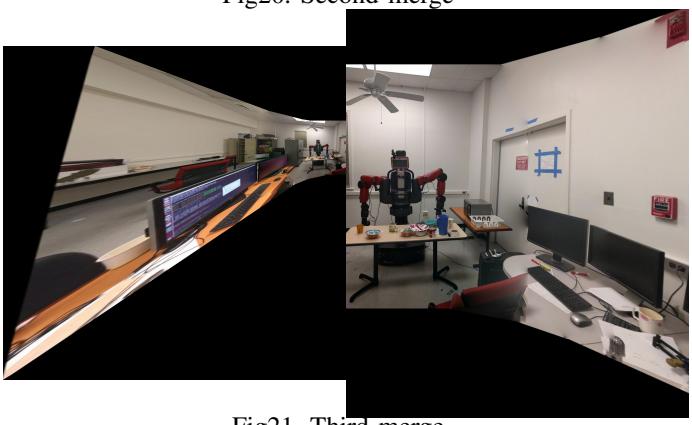


Fig21. Third merge

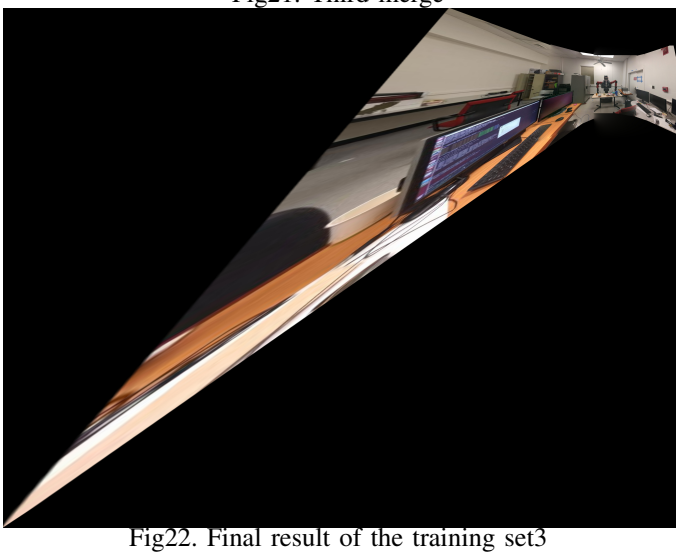
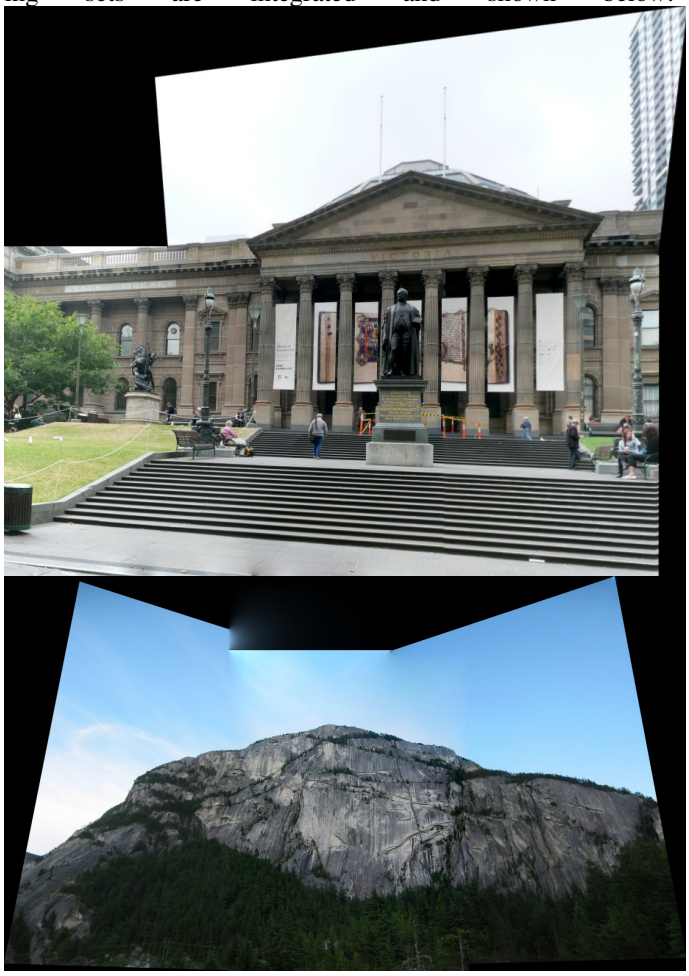


Fig22. Final result of the training set3

H. Results and Conclusions

Though we've tried our best to generalize the algorithm, there are still many cases that required further procedures. For example, due to the perspective transform, the first frame would become extremely large that the computer would run out of memory. However, if we down-sized the image, the features would vanish because the overlapped region is very small compare to the others. So we may need to use sliding-window algorithm to solve this issue. Because of the time and the scope of this project, we didn't go further to refine the algorithm.

The results of the training sets and testing sets are integrated and shown below:



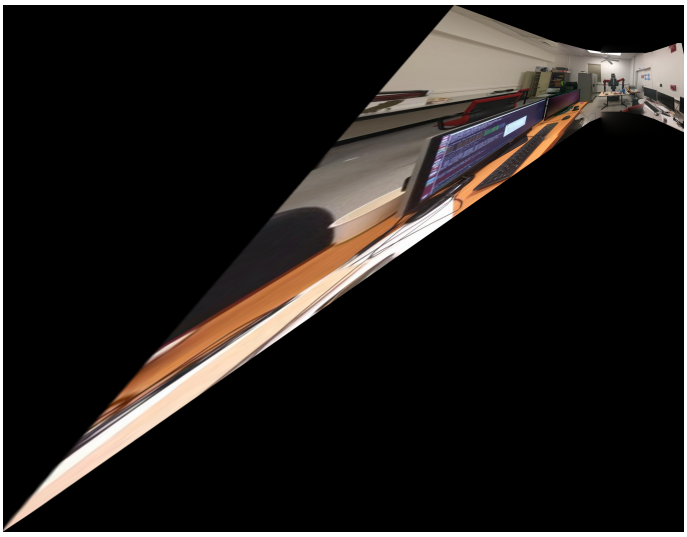


Fig23. Result of the training sets

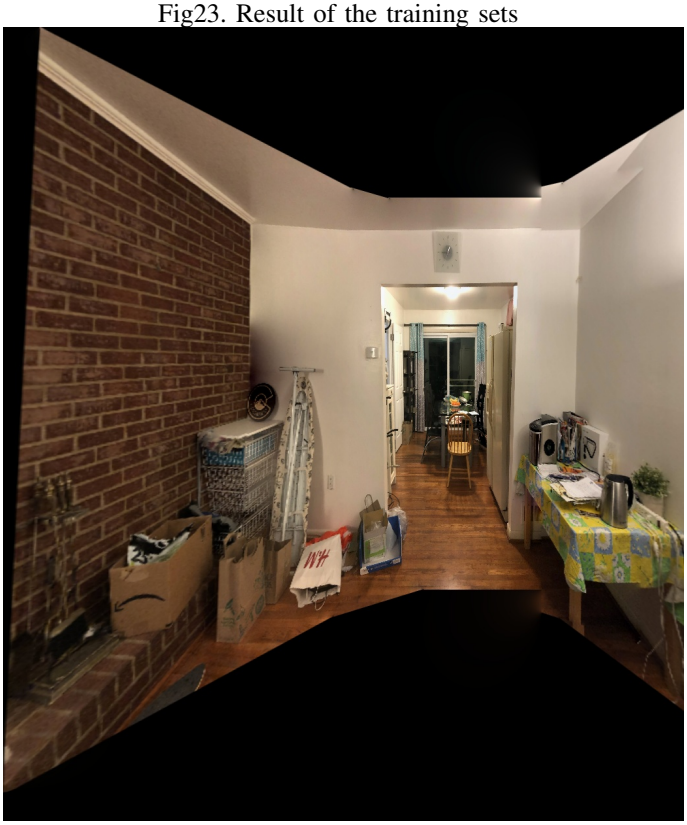


Fig24. Result of the Custom Set1

1) Test Set1: We believe that this test set can't be completed with this pipeline. Though there are many feature points that is easy to match, for computers, they can't distinguish this corner from that corner. They all look the same to the computers, so it's easy to fool them. An example failure of the matching even with RANSAC is shown in figure 25. We still had some good match, but surprisingly they're not even on the board. An example of good match is shown in figure 26, and the final result is shown in figure 27.

2) Test Set2: This test set is also super challenging not just because they cover a wide range of angles, but also they are not in orders. Fortunately, our program sorted the images

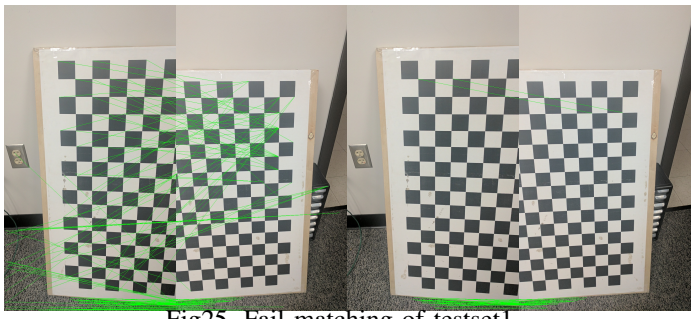


Fig25. Fail matching of testset1

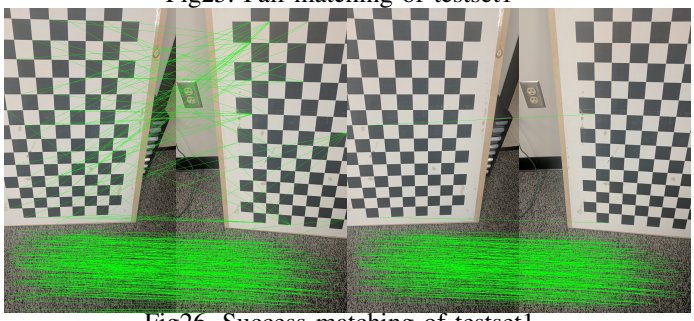


Fig26. Success matching of testset1

Fig27. Final result of the testing set1

successfully. Then, the program divided the images into two sets and merged them separately. Unfortunately, our program failed at the last stage. Because we assumed that the center view is at the center of the images, but in this set, there are multiple images that looks identical. So the center of the view is left-shifted or right-shifted. As a consequence, our result has extremely large distortion. But the program worked well before the last merging. The results are shown in figure 29. If we enlarge the final result, it's actually blended pretty well. The enlarged result is shown in figure 30.

3) Test set3: This test set is the easiest among others. The results are shown in figure 30.

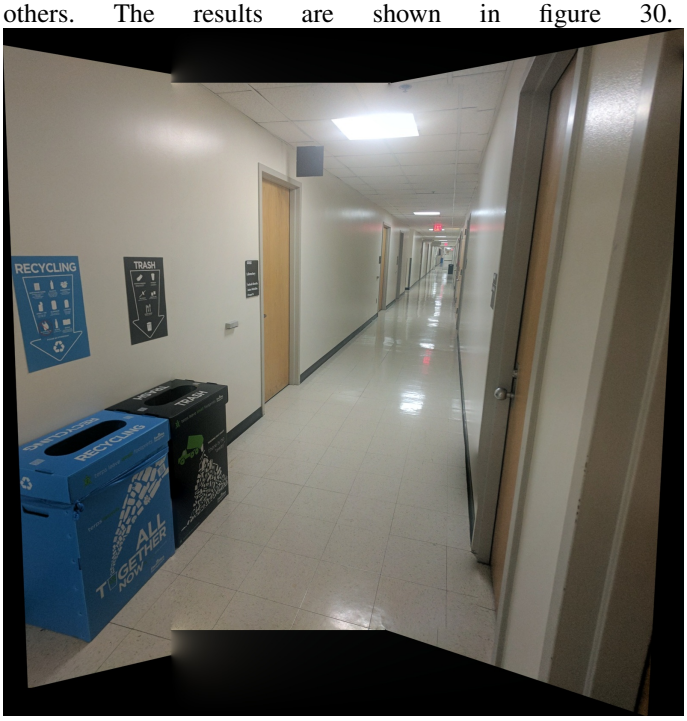


Fig30. Result of the test set3

4) Test set4: The challenge in this set is that it has two irrelevant images. Though we have designed a sentinel in our sorting algorithm, it fails at this set. We believe that the reason our algorithm fails is because of the feature descriptor. The descriptor in this project is too naive that even they are two irrelevant images, they still have about 40 feature points even with RANSAC. So, we can't reject the outliers. On the other hand, our sentinel didn't work is because it believe that the images can still merged together, and actually we still got a pretty good result. And it's because the irrelevant images are all shrink to a small region. That's also the reason why RANSAC isn't working and sentinel fails to reject them. The matching results are shown in figure 31&32, and the results are shown in figure 33.

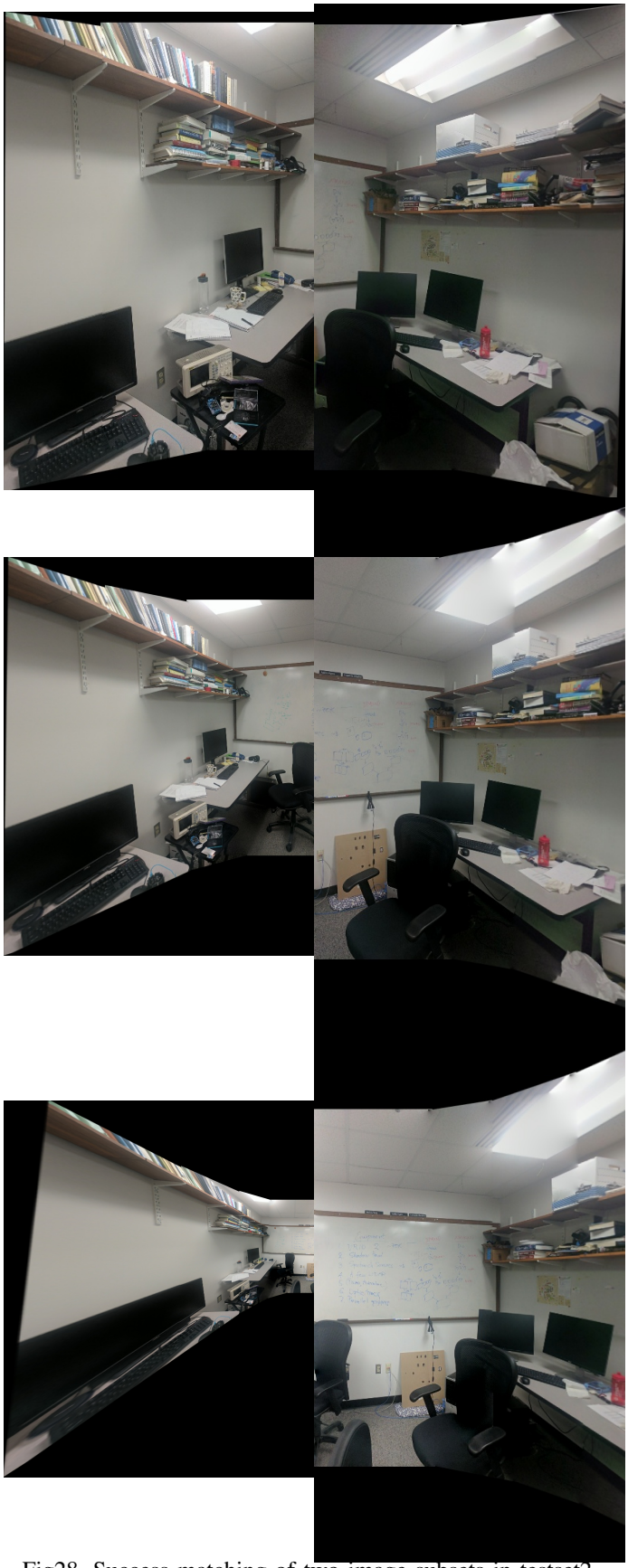


Fig28. Success matching of two image subsets in testset2

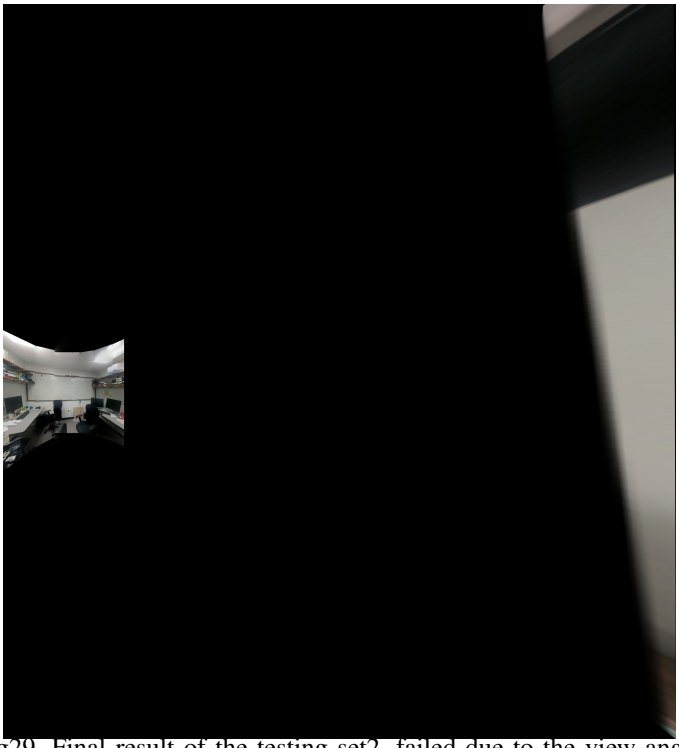


Fig29. Final result of the testing set2, failed due to the view angle

Fig29. Enlarged result of the testing set2

Fig31. Matching and merging of the subset1 in testset4

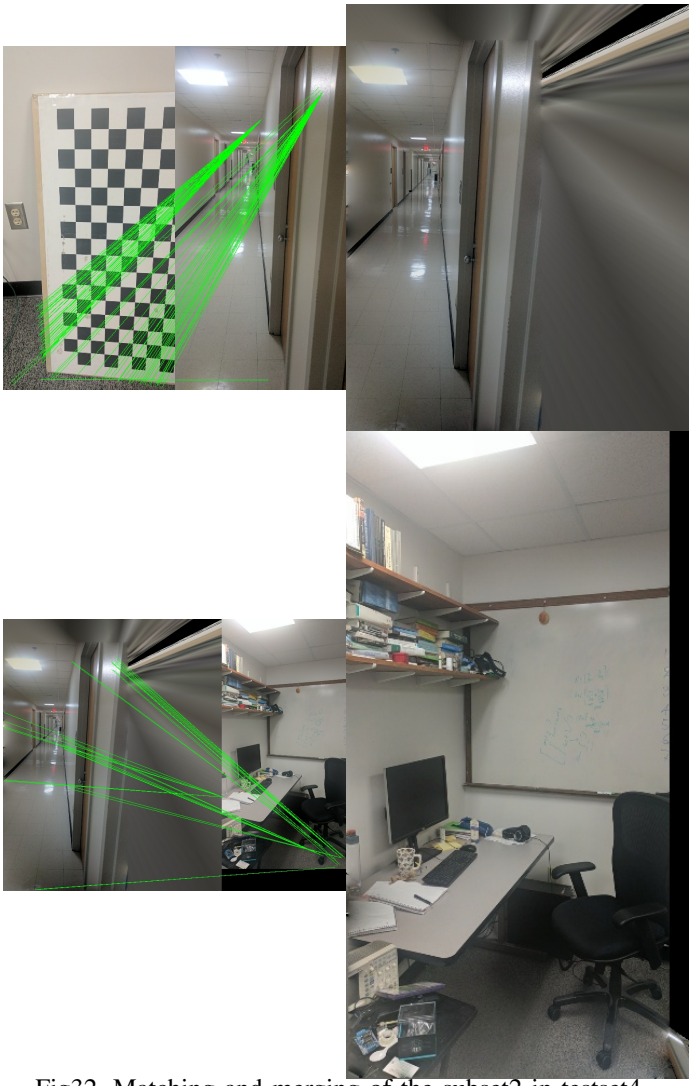


Fig32. Matching and merging of the subset2 in testset4

Fig33. Final result of the tesing set 4

5) Conclusion: This is a very fun project. We had a great time playing with this algorithm, testing its limitation, and producing funny results. Our CustomSet 2: *Shadow Clone Jutsu!* result is at the end of the report.

Fig38. Shadow Clone Jutsu!