COL780 Assignment 1

Rishit Singla

March 2024

1 Introduction

This is a report for the computer vision assignment on Automated Panorama Creation. The code is written in two files, namely main.py and helper.py. main.py imports helper.py and calls the functions from it. Numpy is used for data processing. OpenCV is used only for reading and writing images and videos. Os and sys are used for reading directories and arguments passed to python code.

2 Preprocessing Image

First each image is preprocessed. Contrast of each image is increased and then a gaussian kernel of size 5 is used for image smoothing. Two sample images are shown before and after preprocessing in Figure 1.

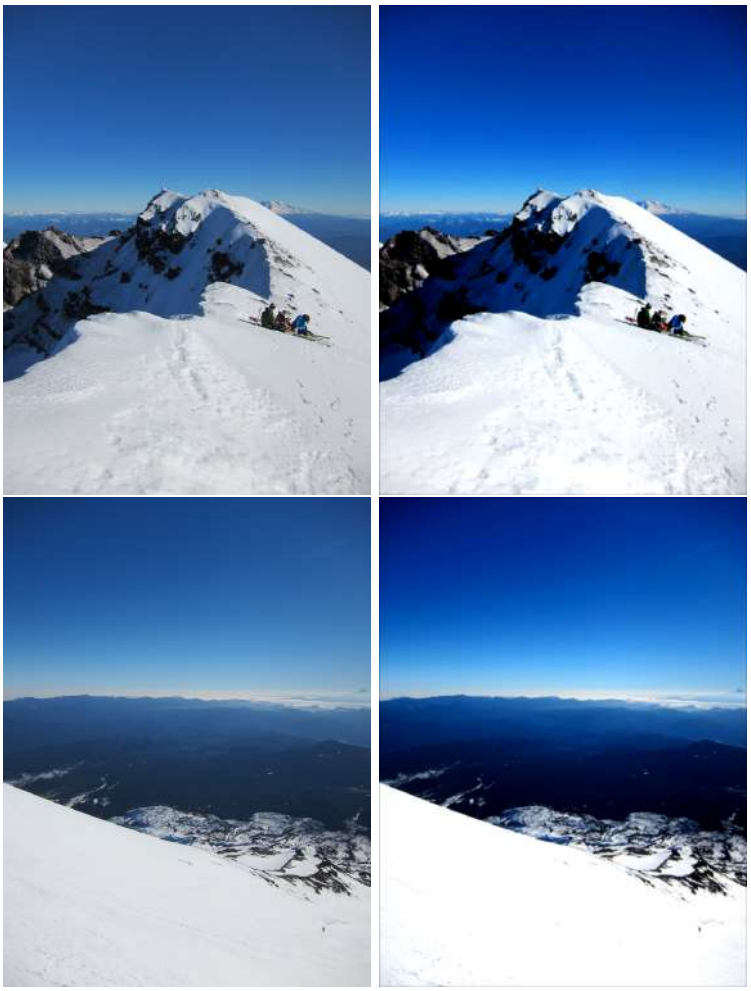

Figure 1: Image before preprocessing (Left), Image after preprocessing (Right)

3 Feature Detection

3.1 Experimentation

For detecting features, I experimented with Harris and Laplace Harris detectors. Since, Laplace Harris Detector is size invariant it gave better result as well as more features to map.
Also, I tried breaking the image into rectangular regions of identical dimensions and finding at least a specified number of features in each region. This was done to ensure features are selected from each region of image so that the overlapping region between two images is not scarce of features. Panoramas were generated

for different number of partitions and it was observed the best results were obtained when image was divided into just two partition, namely, left and right. Having more features resulted into counting not so good points as features resulting into increased noise in further processing.

3.2 Conclusion

Finally, Laplace Harris Detector was used with a three layered scale having the ratio of std deviation between consecutive scales to be 1.5. Image is divided into two parts, namely, left and right while applying threshold on R value. A high maximum threshold was set so that any point with R value more than it should be counted as a feature. If number of features in a single partition were lower than 2000 then the threshold is reduced to a lower value up to a specified value (minimum threshold) so that each partition has at least 2000 features. Features detected in each image is shown as dots in Figure 2.

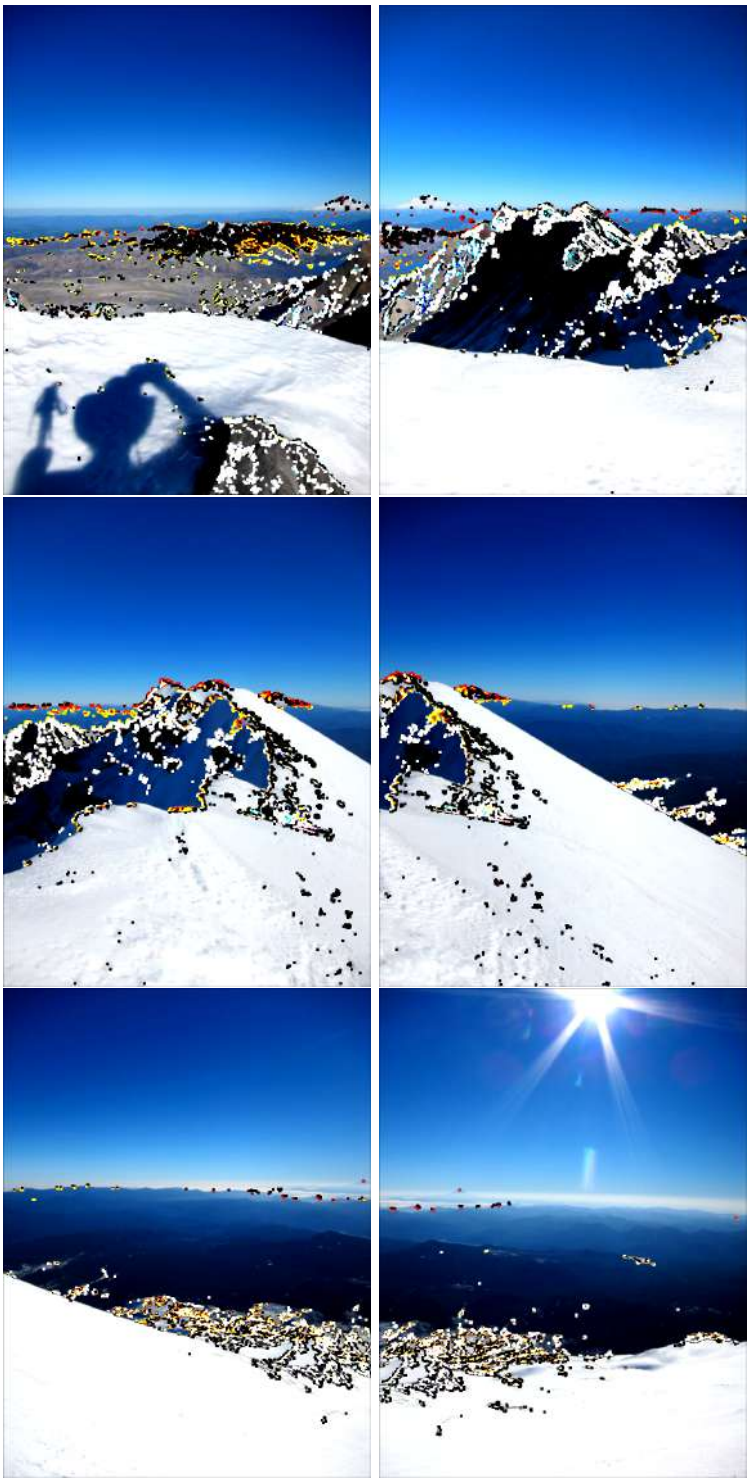

Figure 2: Images with features detected as dots. Color of dot is chosen based on local background colour for easy visibility.

4 Patch Descriptor

Each of above feature is described using a Patch Descriptor. Each descriptor is a 128 dimensional vector generated using SIFT. For generating patch descriptor, first the a (16,16) shaped patch is selected around the feature point. The patch is converted into gray scale. Voting is done using SIFT with 8 bins in histogram for each smaller (4,4) shaped patch.
The descriptor so obtained is normalized to have unit norm. Further thresholding is done, so that any value greater than 0.2 is set to 0.2. Again the descriptor is normalized to have unit norm. This kind of normalization with thresholding allows SIFT to be illumination invariant.

5 Matching Descriptors

5.1 Experimentation

First, I tried to match descriptors by computing distance between all the descriptors of two image. This was very slow when there were more than a thousand of descriptors. So, I switched to use KDTree. Since, we were not allowed to use scipy library, I implemented KDTree from scratch. The performance of my implemented KDTree is lower than of scipy's KDTree but far better than my previous approach.
For Finding the distance between two descriptors, we can use either the sum of absolute difference between coordinates or euclidean distance. By generating panoramas using both the techniques, it was found that using absolute difference is better than euclidean distance.

5.2 Conclusion

For matching features of two images, 2 nearest descriptors were found in second image for each descriptor of first image. This was done using KDTree and using absolute difference for finding distance. If the ratio of distance is more than 0.7 (i.e., best match and second best match are similar) then the corresponding feature point is discarded and not used for any further processing. This ensures selection of only prominent feature matches.

5.3 Plotting Feature Matches in Image

Features matching for two pair of images is shown in Figure 3 and 4. For displaying feature match in image, I tried a different approach. I placed two copy of each image in rectangular fashion such that, first image is adjacent to second image horizontally as well as vertically and vice versa. Now all the matching features are joined using lines. Each match is visualized as trapezium. Thick, parallel and similar sized trapezium shows that matches are consistent with each other, i.e. less fraction of outliers.

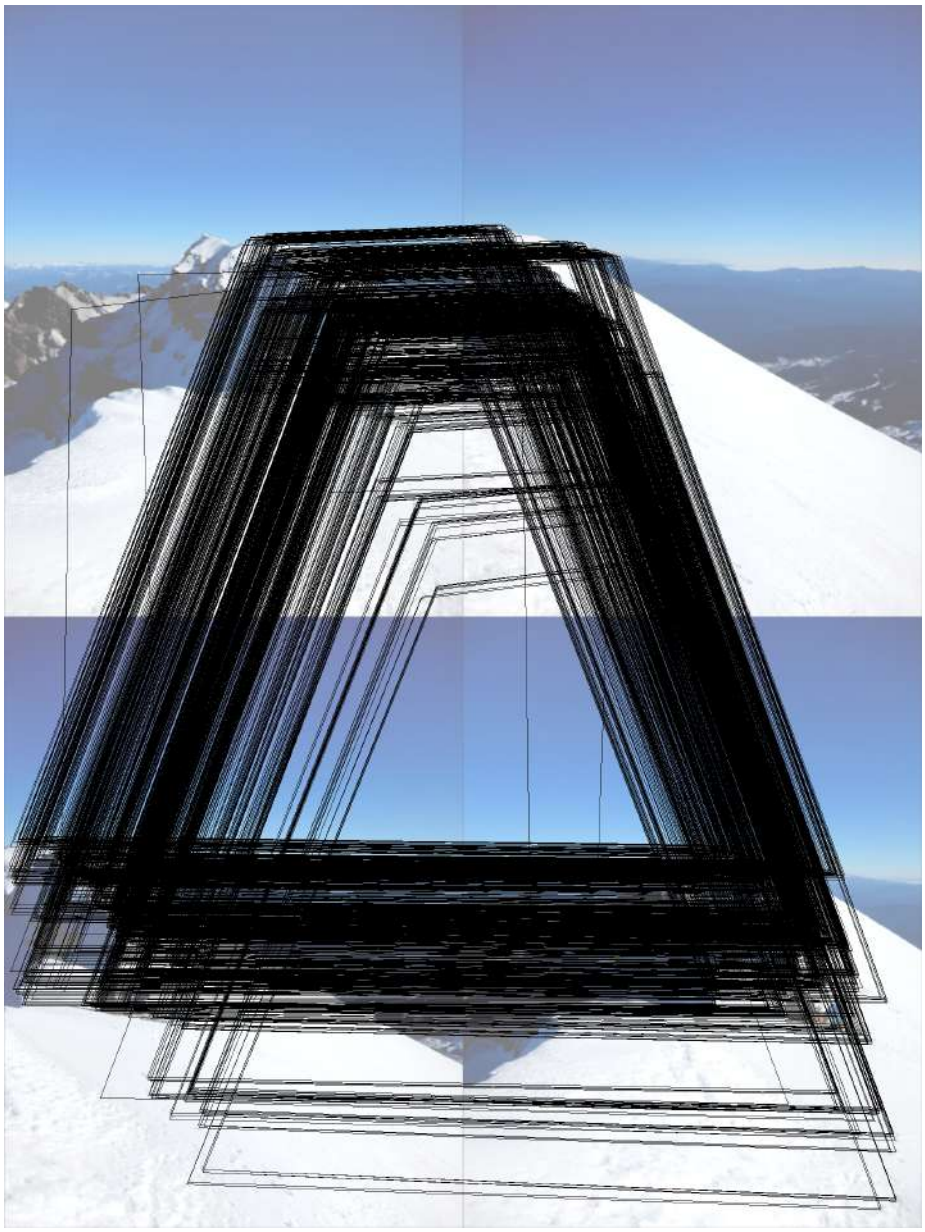

Figure 3: Thick trapezium shows a good consistent match between images and less number of outliers.

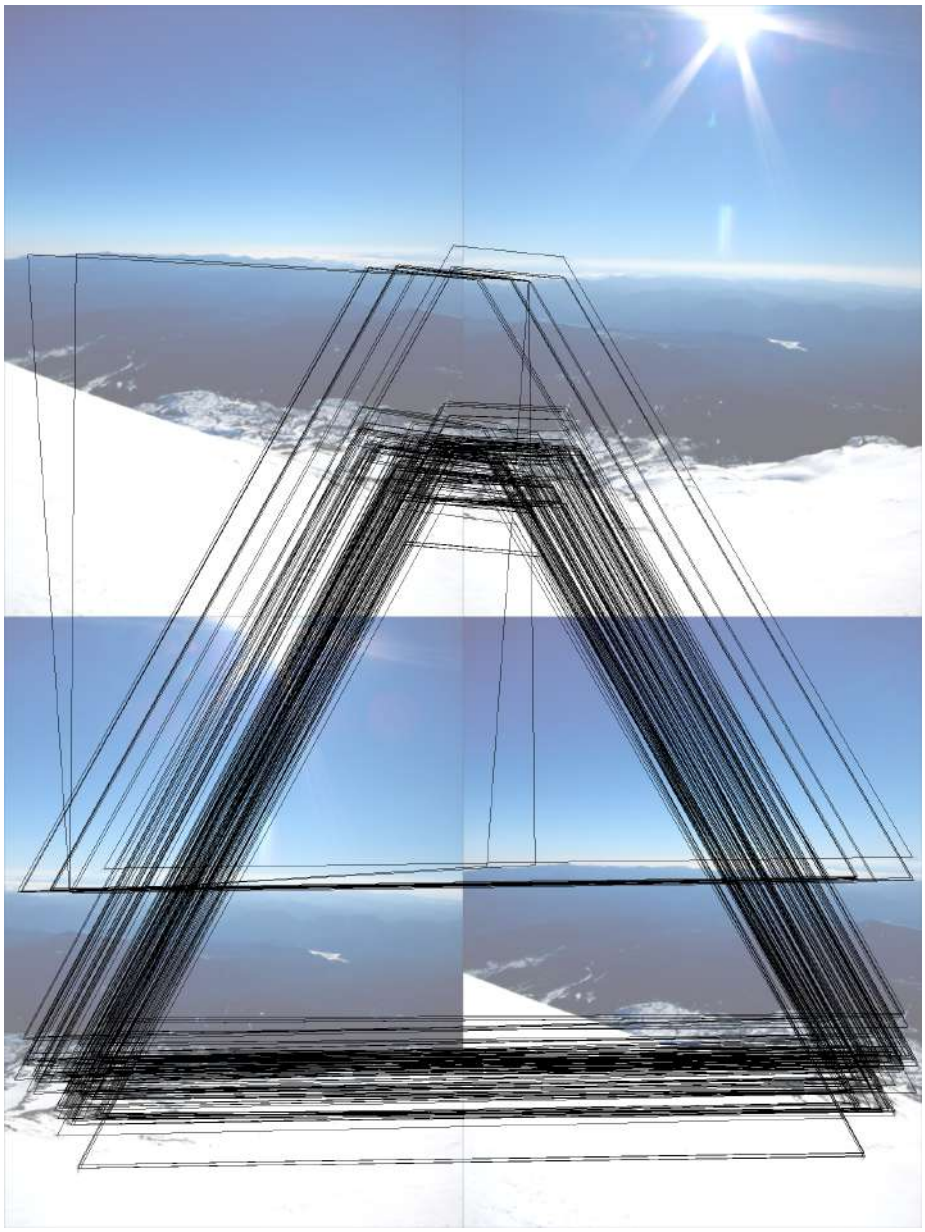

Figure 4: A comparatively thinner trapezium shows presence of more outliers as compared to previous figure.

6 Computing Homography

6.1 Experimentation

I tried calculating homography using calculus method (for affine transformation), linear algebra method and SVD method. Each of them gave similar result, so I used SVD, since it is most common method used.
I experimented with constraining homography to be only affine as well as in general perspective homography. For the case of affine transformation, all of the images had significant contribution to panorama but the panorama was a little rounded up and was difficult to crop in a rectangular fashion. General perspective transform resulted in better results, but at the cost of lost information at the end of images.

6.2 Conclusion

Homography corresponding to general perspective transform was calculated via SVD method. Before calculating homography, the feature data was normalized and corresponding changes were done in the homography matrix. To deal with outliers, RANSAC algorithm was used with each model selected from random 6 matching pair of descriptors. For each run, 1000 iteration were performed and if still, no suitable model is found the error tolerance for inlier classification is increased and another run of 1000 iteration is done.

7 Warping

7.1 Experimentation

I tried both forward and backward warping. As expected, forward warping leaves holes in the image and was difficult to deal with. So, finally I used backward warping with bilinear interpolation to do the job.

7.2 Conclusion

Backward warping with bilinear interpolation was used to warp image. Each image is warped into the plane of center image. The height of warped image is chosen to be same as height of input image. The width of warped image is chosen to be number of image (-1 if number of images are even) times the width of input image. Warping of each image is shown in figure 5.

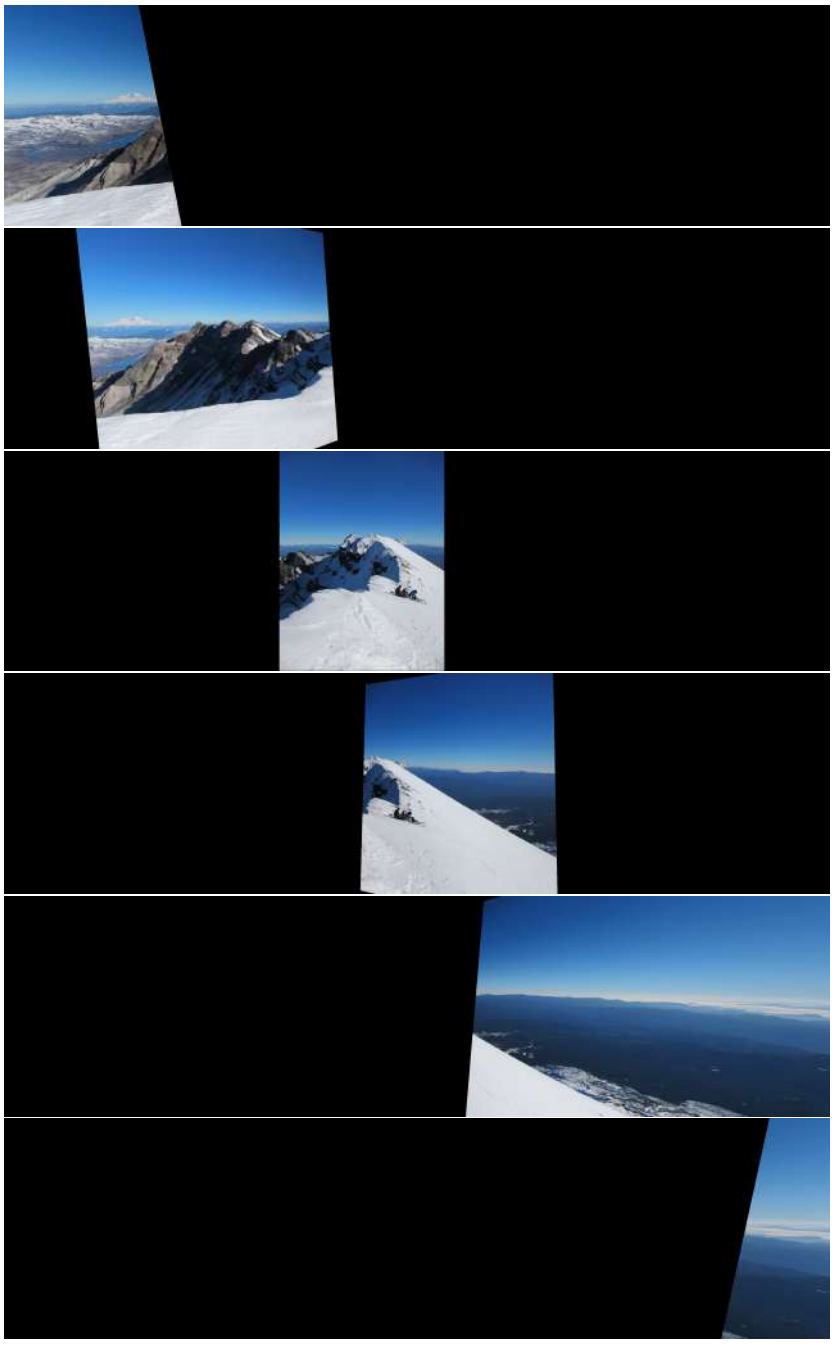

Figure 5: Each image is warped into the plane of second image.

8 Blending

8.1 Experimentation

First, I tried simple averaging which resulted into obvious brightness seams and ghosts. So I tried blending with weighted averaging, where weight is proportional to the distance from image edge. It dealt with brightness seams but the ghosts remain. To deal with ghosts, I tried using graph cut for blending. It gracefully dealt with ghosts but the brightness seams were back (though very less visible this time). The brightness seems could be removed if poisson blending is used with graph cut. But since it is computationally expensive without using external libraries, I did not perform poisson blending. Instead, I tackled this issue by changing the brightness of image so that mean of difference of brightness in overlapped region is zero. Since overlapped region contains many ghosts, instead of overlapping region, I used its boundary.

8.2 Conclusion

Blending was done using graph cut method. The remaining brightness seams were dealt with by changing the overall brightness of images to have same mean brightness in overlapping region (basically its boundary, to be precise). Graph Cut method was implemented using dynamic programming. Figure 6 shows graph cut blending in two of the panoramas.

Figure 6: Blended Panorama. The seam used for blending via graph cut is highlighted using red color

9 Panorama Creation From Videos

For Creating Panorama from Videos, I select 6 frames uniformly distributed over all frames of video (both first and last frames are included). Then use same algorithm used for images on these selected frames to generated the panorama.

10 Drawbacks

The method demonstrated in this report fails to create a panorama with wide angle view. The information in the images at both the ends are lost. I tried to tackle this issue by warping the image over cylindrical surface instead of a plane, but it did not help either. The problem could be sorted out if homography is restricted to only affine transformation. But due to the mentioned reasons in homography computation, a general perspective transformation is used.

Another drawback is that while creating panorama from videos, all the information in frames which were not selected from panorama generation is wasted. Also if the frame picked for panorama generation is blurry, it can lead to a bad blurry panorama.

Due to computationally expensive operations, generating a single panorama takes from 2 to 10 minutes.

11 Generated Panoramas

The generated Panorams for each of the sample data provided is displayed in following figures:

Figure 7: Field

Figure 8: Mountain

Figure 9: Office

Figure 10: Video 1

Figure 11: Video 2

Figure 12: Video 3