A System of Image Matching and 3D Reconstruction

CS231A Project Report

Xianfeng Rui

1. Introduction

Given thousands of unordered images of photos with a variety of scenes in your gallery, you will find it is very interesting to organize your photos according to scenes, and create a 3D reconstruction for each of scenes. The most important and challenging part in this task is the matching of images. The popularity of mobile devices has brought a great demand for computer vision application on platforms with limited computation and storage resources. The main focus of this project is to perform some interesting evaluations to help decide which keypoint descriptor is more suitable for image matching before 3D reconstruction and/or image panorama given the restricted resources. After matching the images with the same scene, I am going to use the results to reconstruct a 3D view for each scene. The flow of the system is shown in Figure 1.

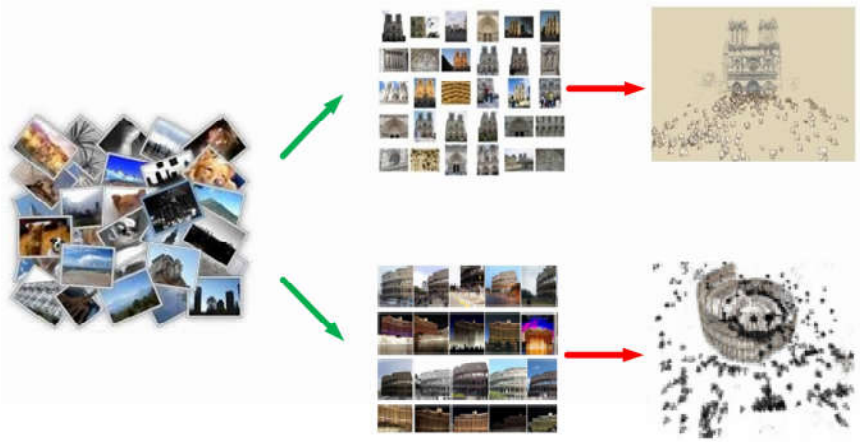

Figure 1. The flow of the system

2. Background/Related Work

A lot of algorithms/methods have been developed to detect distinctive invariant features in images, and the most famous one is SIFT developed by David Lowe[1]. It was left unchallenged before SURF was developed. To compensate the complexity involved in SIFT and SURF, some novel binary descriptors such as BRISK[2], FREAK[3] was discovered and were proven promising in some cases. All these are good candidates to detect distinctive invariant features for various purposes. Also a lot of comparison was made between them for performance and computation cost; however none of them is perfect for any specific task.

3D reconstruction is based on matches between multiple images of the same scene. The most famous 3D reconstruction project was “Photo Tourism” [4] developed by the collaboration of University of Washington-Seattle and Microsoft, where a large collection of photos either from personal photo collections or internet photos sharing the same site is the input, and each photo’s viewpoint and a sparse 3D model are computed using incremental bundle adjustment approach. CMVS and PMVS[5] were further developed to create a 3D dense reconstruction.

Since 3D reconstruction or image panorama or other applications assumes the images from the same scene, there is a lot of manual work done by us to group images belonging to the same scene together firstly. To my knowledge, there is no much work on this region and I want to develop an effective method to group these images belonging to the same scene automatically. Our approach here is trying to use various descriptors, in particular SURF and BRISK in this project, to evaluate their performance on a real dataset[6] with the consideration of resources. 3D reconstruction is the second step in our project by using the results from the first step.

3. Technical Approaches

This section details image database and approaches I will implement or use in our work.

3.1.Image Database

Our initial database comes from Hyojin Kim[6] at University of California-Davis, and includes 7 scenes which include around 7-10 different views for each scene. After I downloaded these images, I purposely mixed them together to form the whole dataset for image matching step where I try to group images of the same scene.

Figure 2. Sample images from dataset

3.2. SURF and BRISK

SURF is based on the same principles as SIFT, but it improves on SURF by using a box filter approximation to the convolution kernel of the Gaussian derivative operator with the using of integral images. SURF uses a blob detector based on the Hessian matrix to find points of interest. The descriptor is formed by concatenating the histograms of gradients of sub-grids around the keypoint into a 64 dimension vector.

BRISK is a 512 bit binary descriptor with sampling pattern composed out of concentric rings. It distinguishes between short pairs and long pairs by their distance. The short pairs are used for building the descriptor by comparing the smoothed intensity of the first point in the pair is larger than that of the second point. This builds the binary descriptor that uses hamming distance instead of Euclidean, which can speed up computation a lot.

In our project, I used these two detectors to detect features for image matching, and compared their performance over the dataset.

3.3. Selection of KeyPoints

There are many keypoints detected when I use SURF and BRISK to detect, and some images have even more than 10,000 keypoints. Suppose I have **n** number of images to match, and **m** number of keypoints in each image, then the complexity of keypoints matching is O(**n*n*m*m**). If I use these keypoints directly, tremendous computation will be incurred, and therefore not practical in real applications. In order to solve this problem, I only took the most important keypoints for our task, and here I measured the importance of each keypoint by their individual strength since it is roughly an indictor of how good the keypoint is. Although the strength of some keypoints will change due to the change of light and there is no guarantee that a good keypoint is still there, it is not that usual, especially in our dataset. In our project, I ran our matching algorithm by considering all the keypoints in each image as a reference, and sorted these keypoints in terms of their strength, and compared the performance by selecting the different numbers of the most important keypoints for SURF and BRISK cases, which is the main focus of our project. Section 3.6 talks out the metric for performance measure.

3.4. RANSAC

Some matches extracted by comparing the descriptors of keypoints, even with our selection, of two images are actually outliers. In order to remove these invalid matches between two images to build up a more robust matching, I implemented RANSAC to help us. I calculated the fundamental matrix between two images using 8-point algorithm, and defined a threshold so that one pair of matches is classified as outlier if the error of this pair is larger than it based on the calculated fundamental matrix from the selected 8 points. I randomly took 8 points for each iteration and used the iteration with the least number of outliers as our model, and removed outliers. Figure 3 shows the difference before and after RANSAC.

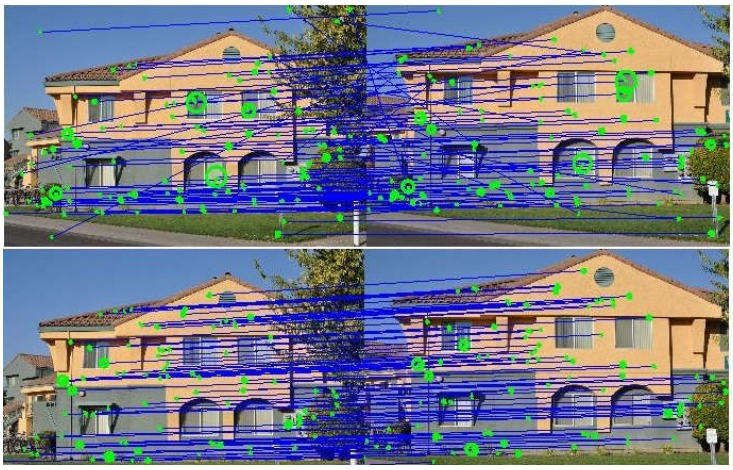

Figure 3. Keypoints matching before and after RANSAC

3.5. Clustering

The last step is to cluster images. After calculating key point matching between every pair of images with RANSAC, I have a match matrix (nxn, n is the number of images) to store the number of matches between any two images. Clustering algorithm was implemented to group images based on the match matrix. Different views of the same scene don't necessarily have the large number of matching points considering that they might be taken from relatively big different viewpoints and they don't have many visible points in common. Even though they don't have many key points in common, they might have much more matching points with an intermediate image which has a not much different viewpoint from both images.

I designed a different clustering algorithm to group these images from the same scene. I looped through every element of match matrix (actually I only cover the half matrix since the other half is almost symmetric to the first half). If the element, i.e. the number of matching points, is larger than a threshold (25 in our experiments), I consider them match, and add them to the cluster as follows, otherwise the whole pair is dropped. If one image is already in one cluster, and the other one isn't in any cluster, then I add the second image into the cluster the first image belongs to. If both of them are already in clusters, then I merge these two clusters. If none of them are in clusters, a new cluster is created to hold these two images. In this way, the images from the same scene can hopefully be classified into the same cluster. Based on these clusters, I measured the performance based on metric discussed in Section 3.6.

3.6. Metric

There are three important metrics involved in performance measure for clustering. I am expecting after the clustering algorithms, all the images should be got grouped, and also correctly, and the number of clusters should be the same as the exact number of scenes. Therefore I have metrics below to measure performance.

3.6.1. Misclassification Rate

It measures whether the images of each cluster belong to the same scene. In order to measure this, I assume that the images of one scene that occurs most frequently in one cluster are grouped correctly, and other images of the other scenes in this cluster are clustered wrongly. Suppose one image of scene 1, two images of scene 2, and three images of scene 3 are in one cluster, then I think this cluster is scene 3, so the misclassification number is 1+2. I looped all the clusters, added these misclassification numbers of each cluster together, and divided it by the images which got clustered to achieve the misclassification rate.

3.6.2. Clustering Rate

Smaller misclassification rate doesn't necessarily mean a better clustering since some images might not get clustered at all, which means that these images don't appear in any cluster. It is possible since there is chance that one image has no good defined match with any image. It is calculated by the percentage of images which get clustered over all the images.

3.6.3. Over-Cluster Rate

0% misclassification rate and a clustering rate of 1 don't mean the clustering is perfect. Clustering might create more scenes than expected, although all the images get clustered and all the images of one cluster belong to the same scene. I measure over-cluster rate by the percentage of the number of extra clusters created over the expected number of scenes. It is less important measure compared to the other two since it is still acceptable if it is not perfect. In our context, I mainly focus on the first two metrics.

3.7. 3D reconstruction

3D reconstruction takes images of the same scene as input to generate camera postions and 3D views. Bundle adjustment is used to perform a 3D sparse reconstruction, and CMVS(Clustering Views for Multi-view Stereo) and PMVS(Patch-based Multi-view Stereo) are used to create a dense 3D view. CMVS is a preprocess for PMVS. Here I complied the libraries of SIFT, Bundle Adjustment, CMVS, and PMVS separately and integrate them into Visual SFM Package[7] to form both a sparse and a dense 3D reconstruction. The lib requires one point should be seen by three images in order to be reconstructed in 3D.

4. Experiments

4.1. Image Matching

Image matching is the most important part in our project. In our dataset I have totally 67 images and 7 scenes for experiments. As I talked earlier, I tuned the number of important keypoints where importance is measured by the strength of keypoints. In order to get an

idea of the performance degradation by limiting the number of keypoints for matching purpose, I ran the clustering with all the keypoints considered for all the images for both SURF and BRISK without the storage of keypoints, and got the performance metric in table 1 below:

	SURF	BRISK
Misclassification Rate	0.015873	0
Clustering Rate	0.980299	0.925373
Over-Cluster Rate	0.285714	0.428571

Table 1. Performance of SURF and BRISK with all the keypoints considered.

SURF has a higher clustered rate than BRISK due to its high accurate description of descriptors. However, the time spent on SURF (around 3 hours) is 100x longer than BRISK, and it is expected since the matching part of BRISK is simply hamming distance instead of Euclidean, and also there are more keypoints detected with SURF in our dataset. And it is not practical for SURF to use all the keypoints, and here it is only to extract the performance for reference.

The next step is to vary the number of important keypoints to compare with the golden one above. I ran our codes at 108, 216, 396, 540, and 640 for both SURF and BRISK with the storage of keypoints, which means that I only need to extract the keypoints for each image once and store there for future usage, and it is practical since the number of keypoints got stored are limited, and there is no re-computation of keypoints. I found that misclassification rate is 0 for all the cases, so I showed the clustering rate plot in Figure 4 for various numbers of keypoints. The computation time of SURF is 2x slower than BRISK, and both of them are a couple minutes with our dataset of 67 images.

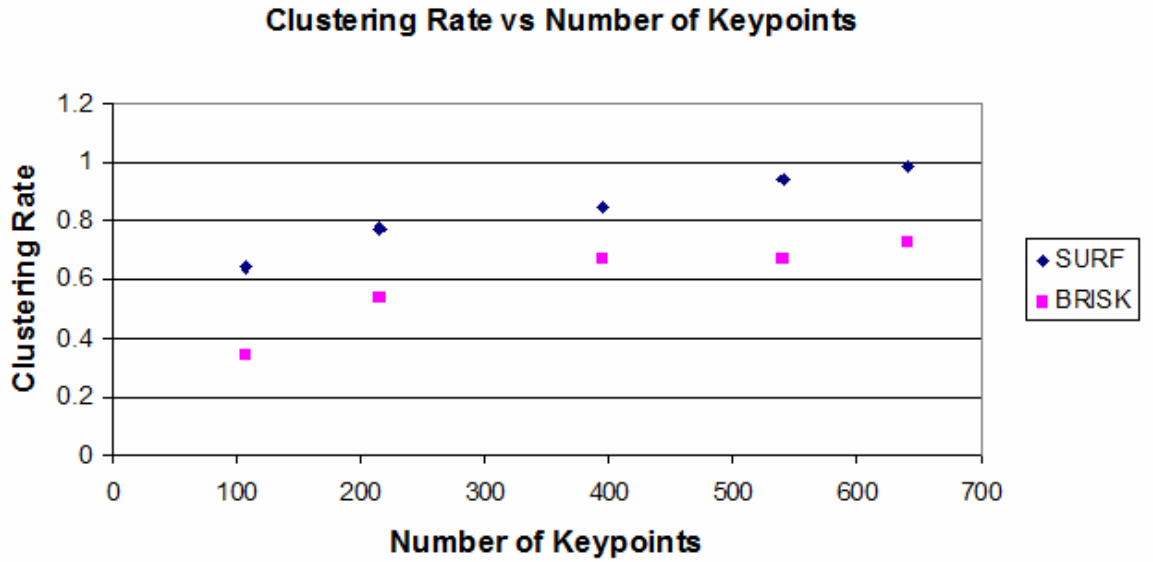

Figure 4. Clustering Rate versus Number of Keypoints for SURF and BRISK

With the increase of number of keypoints used for image matching, an improvement of clustering rate was observed, which is expected since in the limit it is the case of golden

case where all the keypoints are used. In all the cases, SURF always outperforms BRISK. The most exciting result is in the case of 640, the performance of SURF is roughly same as the golden case, which means that these 640 keypoints are really those eventually used in the golden case in our dataset. This result is really encouraging since we can achieve the same performance with much less computation cost (here several minutes vs 3 hours). The improvement of performance for SURF implies that the strength of keypoints is a good indication of distinctiveness/goodness for our dataset. However, the experiments on BRISK exhibit the different behavior. Even though clustering rate improves with the increase of the number of keypoints, it is not that apparent, and there is 30% difference with SURF at 108, and performance at 640 is the same as that of SURF at 108. These indicate that the strength of BRISK keypoints can't fully express the distinctiveness/goodness of keypoints in our dataset. In order to further investigate the case of BRISK, I actually took a step further to divide the whole image into nxn grids, and extracted the important keypoints grid by grid instead of extracting them from the whole image, and I saw around 5% improvement, but still much lower than SURF.

The sign of the Laplacian for keypoints in SURF can be used further to speed up image matching (only compare when their signs are the same), and is also quite useful when using k-d trees. Given a 64-dimension feature of SURF, there are still a lot of storage and computation involved in image matching, and 36-dimension PCA versioned feature can be useful for saving of computation and storage, which is comparable to BRISK.

4.2. 3D Reconstruction

After the investigation of performance over the change of the number of keypoints, I achieved the best results at 540 at acceptable computation and storage cost. Base on the clustering results from 540, I performed 3D sparse and dense reconstruction by integrating the libraries of SIFT, Bundle Adjustment, CMVS, and PMVS into Visual SFM Package. In our best case of clustering, actually I have three extra clusters formed for scene 1, so I reconstructed the other 6 scenes, and below shows the sparse and dense reconstruction for scene 0 (Figure 5) and scene 2(Figure 6) with satisfactory results. We also observed some failures of reconstruction in other datasets, and it is due to the repeatable structure. Figure 7 shows invalid matches between two views of Huang because of repeated small trees and lawns, leading to failure of 3D reconstruction.

Figure 5. Sparse and dense reconstruction of Scene 0

Figure 6. Sparse and dense reconstruction of Scene 2

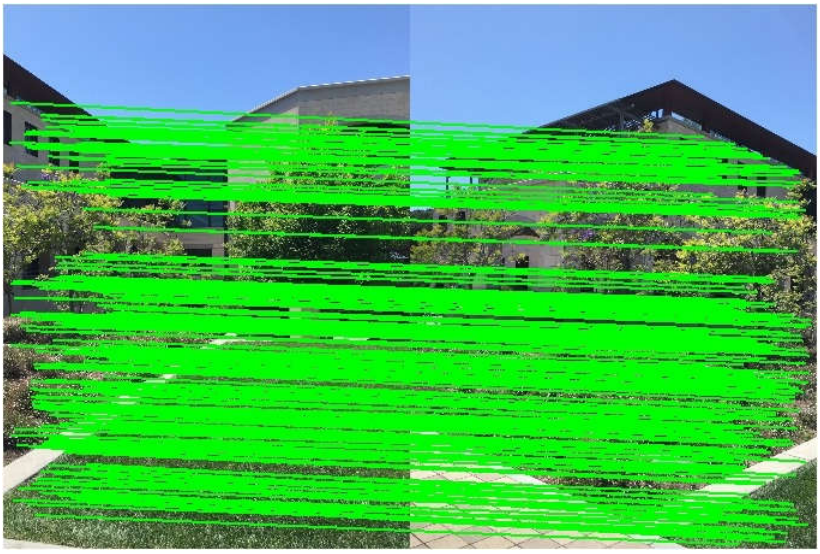

Figure 7. Wrong matches between two images of Huang

5. Conclusions

In conclusion, I have compared the performance of SURF and BRISK on the image clustering by considering limited number of keypoints for saving of computation cost of computer vision applications. I discovered that the strength of SURF keypoints is a good measurement of distinctiveness/goodness and can be used for keypoints selection in our dataset for image matching, and achieved a comparable performance with much less computation cost. The sign of the Laplacian for the keypoints in SURF can help speed up image matching. A more aggressive approach might be taken to use PCA SURF to reduce the dimension of descriptor for more saving of computation and storage. SURF

might be a promising candidate for mobile platform applications. My analysis might provide some information to someone trying to select limited keypoints for their own applications. I hope in the next few years, there are more and more compute vision applications on the mobile platform, and necessary performance analysis will contribute to this advent.

The codes and dataset are accessible through the link below:
https://drive.google.com/folderview?id=0B_eHunfjZfIbTkRtQVJYMzdQWWM&usp=sharing

6. References

1. "Distinctive image features from scale-invariant keypoints" DG LoI - International journal of computer vision, 2004
2. Bay H, Ess A, Tuytelaars T, et al. Speeded-up robust features (SURF). Computer vision and image understanding, 2008, 110(3): 346-359.
3. Alahi, Alexandre, Raphael Ortiz, and Pierre Vandergheynst. "Freak: Fast retina keypoint." Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012
4. N.Snavely, S. Seitz, R. Szeliski. Photo Tourism: Exploring Photo Collections in 3D. ACM Transactions on Graphics (2006)
5. http://www.di.ens.fr/pmvs/pmvs-1/index.html
6. http://idav.ucdavis.edu/~hkim/mvs/dataset/
7. http://ccwu.me/vsfm/