Automatic Joiners - Report for Project 1

Name: Jian Ru

Student ID: 10389231

Dept.: School of Software

Introduction

This is a report about project 1, Automatic Joiners. Joiner is an artistic genre proposed by David Hockney. Joiners are a reconstruction of related-photos captured from different viewpoints and they are distinguished by the inconsistency across image borders which introduce a strong artistic visual effect. Joiners are great since they provide multiple benefits: artistic photo composition, panorama, and a new way of organizing images. Currently, however, joiners are composed by photographers or artists manually where the whole process is long and tedious. Furthermore, it requires lots of experience and skills to compose joiners when the number of photos grows large. In this context, an application for automatic joiners is preferable which can enable non-professionals to compose joiners using their own photos and share then with their friends, and provide a start point for professionals.

In this project, I have done three things:

- Implement a C++ version of automatic joiners;
- Experiment both the MATLAB and C++ implementations using photos captured or found by myself and make a comparison of the results and the two implementations;
- Make a little improvement to my C++ implementation.

All in all, this work implements the application of automatic joiners proposed by [L. Zelink-Manor and P. Perona 2007] with some simplification.

Implementation

Well. It's time to talk about procedures and algorithms. According to the original paper, the application of automatic joiners includes three steps: image alignment, ordering images, and iterative refinement. I my implementation, however, I abandoned the iterative refinement step for several reasons that I will discuss further in the following. I actually tried iterative refinement and if you look into my code, you will see there is a large amount of experimental code dedicated to iterative refinement. Whereas, the problem is that both my implementation and the MATLAB implementation always deteriorate results instead of improving them. Thus I ditched the iterative refinement process. I have contemplated the reason for degradation for a long time and I have come up with several possible catalysts for this problematic

phenomenon that I will further explain later. For now, I will focus on how I implement the application.

Image Alignment

In the original paper, image alignment is done by the following steps. First, they extract and match SIFT features between all pairs of images. Then, they use RANSAC to select a set of inliers (valid matches). Next, they apply the probabilistic model suggested in [Brown and Lowe 2003] to verify the match. Finally, they use bundle adjustment to solve all the transformations. There are similar to the work done in [Brown and Lowe 2003] but the difference is that the photos are no longer taken from the same viewpoint. In this context, it is impossible to avoid inconsistency across image borders and thus given all the key-points the same weight can cause misalignment and distortion. As a result, they gave different weights to key-points according to their distances to the visible image borders where heavier weights are given to the key-points close to visible borders while lighter weights are given to the key-points farther from visible borders. The objective is to minimize the weighted sum of distances between each pair of matched key-points. This is a non-linear least square problem and they solve this using Levenberg-Marquardt algorithm.

In my and the MATLAB implementation, a simplified method [S. Umeyama 1991] for calculating similarity transforms are used in lieu of the more complex and powerful bundle adjustment. The reason is that bundle adjustment is too hard for me to implement. I think the authors of the papers assume that the readers have enough pre-knowledge to figure out the implementation details by themselves and thus the algorithm is not clearly explained. I tried to implement bundle adjustment and I read a lot of materials to help my understanding but nevertheless my implementation is incorrect since the results of alignment after my bundle adjustment implementation is applied are much worse than before. Therefore I adopted the simplified method which is also adopted in the MATLAB implementation. The disadvantage is that the simplified method always assumes the same weight for all the matches, which, as explained above, can cause misalignment somehow.

Speaking of the difference between my implementation and the MATLAB implementation, I use all the matches to compute similarity transforms while the MATLAB implementation randomly selects a pair of matched images and uses their key-points to compute similarity transforms. The advantage of my choice is that all the matches are considered so that there is no bias that in favor of one match rather than others. This method generates better results than considering the information of only one image match when there is no mismatch between images. The precondition is important since all matches are considered of the same importance, a noise match (mismatch) can cause severe problem. Nevertheless, the precondition is usually true since the presence of RANSAC. Random Sample Consensus (RANSAC) is a non-deterministic algorithm for finding a good model based on a set of noisy data points. RANSAC is capable to sift most of the outliers and find a good model that fits the inliers well. I hypothesize that the data points given to RANSAC are not very noisy. That is, most of the points are inliers with a small number of outliers. Based on this hypothesis, I gave a threshold to the number of inliers found by RANSAC. If the number is higher than the threshold, the two images are considered as a valid match. Otherwise, they are treated as mismatch and won't be used for alignment.

It is time to summarize my work. I implement image alignment in the following steps:

- Extract and match SIFT key-points between each pair of images;
- The matched key-points of an image match are passed to RANSAC to sift outliers and a threshold, twenty in my implementation, is used to decide whether the match is valid;

- The information of all the valid matches is recorded while images with no match are pruned;

- Finally, similarity transforms that transform each image to the matched images on canvas are computed and stored. I start from the image that has the most image matches.

Ordering images

The implementation of image reordering is the follows the steps introduced in the original paper. The important thing is that this reordering is not optimal since evaluating all the permutations is too costive. Image reordering is done as follows:

- Start from a randomly chosen image and put it on the canvas (transform it using its similarity transformation obtained from the last phase);

- Next, we enter iterations. An image that is not on the canvas is selected in each round. Try all the orders of the images that are already on canvas and the image selected with the order of the images that are already on canvas unchanged.

- In each attempt, compute the energy of the current order and choose the order that has least energy. The energy of an order is represented by the sum of the magnitudes of gradients on visible image borders.

The reason why we need reordering is that photos are captured from different viewpoints. As explained before, inconsistency across image borders are unavoidable. Nevertheless, it is possible that inconsistency is occluded by other images. Thus it is meaningful to find an order that exposes those borders with better consistency and hides the borders that are more inconsistent.

Iterative Refinement

Iterative refinement is abandoned in my implementation and the MATLAB implementation used only one round (that is, realignment after reordering). As explained before, the reason why this phase is derelict is that it usually deteriorates results. In my opinion, the radical reason is the strategies of realignment of my and the MATLAB implementation cannot ensure convergence. This difficulty is introduced by the simplified method for calculating similarity transforms. The simplified method assumes equal weight for all the key-point matches. In order to simulate the effect of the weighted approach used in the original paper, both my and the MATLAB implementation choose a set of key-point matches based on some criteria and totally ignore those key-point matches that are not selected. This is equivalent to assigning one as weight to those selected matches and zeros as weight to those that are excluded. The ignorance of some key-point matches is a bad idea since those matches passed the test of RANSAC and hence they are valid matches. By ignoring them totally, we lose too much information or, in order words, we overestimated the importance of those selected key-point matches. As we can see, the original paper gives a minimum weight of 0.1 to those key-point matches whose weight are lower than 0.1 and uses a robust error function to prevent prejudice. Based on the reason above, I decided to abandon iterative refinement in my implementation.

Result

In the following pages, I will show the results of my implementation and the results of the MATLAB implementation. By convention, the results of my implementation is always shown on the left-most column, the results of the MATLAB implementation before realignment is shown on the middle, and that after realignment is shown on the right. After showing the results, a short discussion and comparison is made. One experiment result is not shown but it is provided in the delivery package.

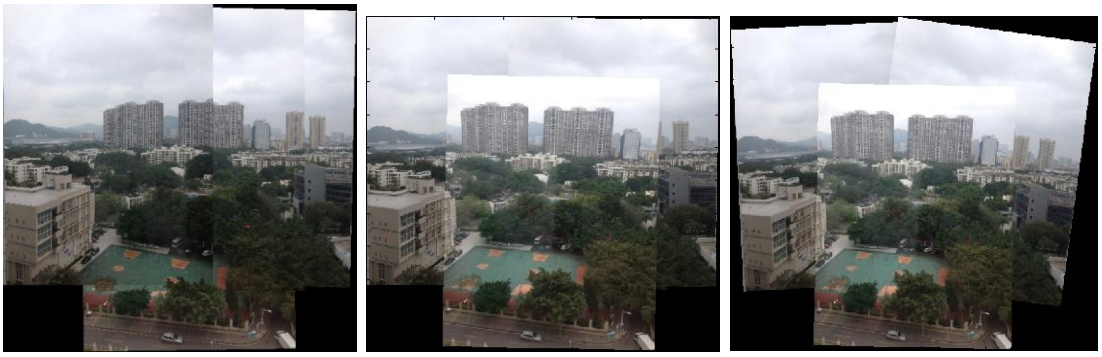

Fig.1.1. Wide-angle landscape (4/4)

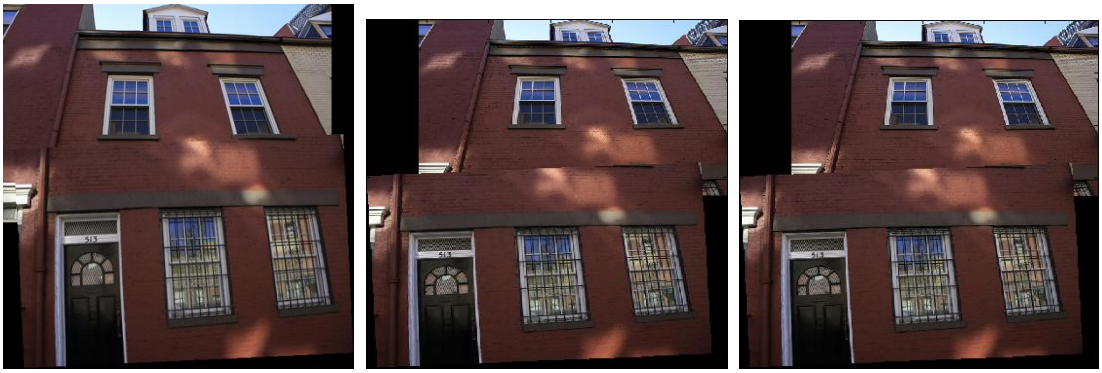

Fig.1.2. Red building (2/5)

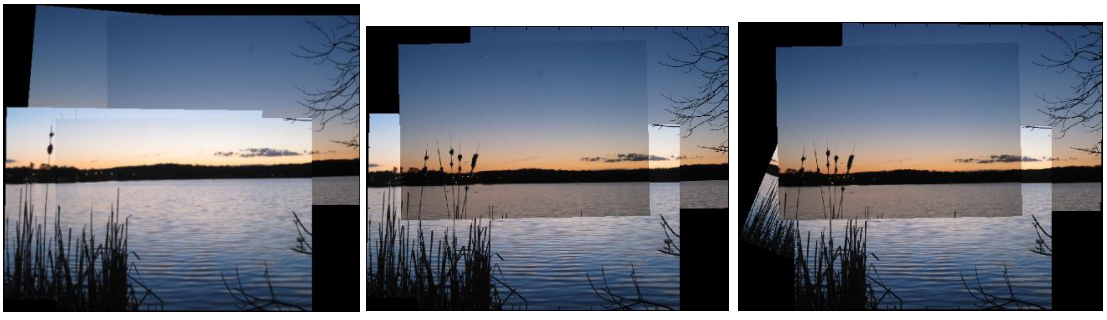

Fig.1.3. Lake (4/6)

Fig.1.4. Inner space (6/6)

Fig.1.5. Man (11/11)

Well. There are some results above and let's discuss them.

The result in Fig.1.1. is reconstructed from four images. As we can see, the result is relatively pleasing and things are matched well but the third image which is the result of the MATLAB implementation after realignment presents some distortion. Those images were captured by me and when I shot those images, I held the camera upright. So I now there is no reason to rotate any images. This is caused by the total ignorance of key-point matches.

The result in Fig.1.2. is reconstructed using two out of five images. This result shows that the algorithm of automatic joiners is not very robust against moderate or large change in viewpoint orientation. The three images that are considered irrelevant by the application of automatic joiners present relatively strong yaw of viewpoint about the red building and the application is not capable to deal with this condition. Furthermore, this result also shows my implementation gives better result than the MATLAB implementation does.

The result in Fig.1.3. is reconstructed using four out of six images. This set of testing images is downloaded from a website on which there is also a joiner composed by these six images. But the joiner on the website used all six images and the result is much better than mine and result of the MATLAB implementation. This shows that improvement is needed for our image matching mechanism.

The result in Fig.1.4. is a "happy ending" result, which means that all the images are recognized and matched relatively good. But this result shows important things. As you see, the third joiner is much worse than the second one but the third one is the result after realignment. This fact shows that our approach of realignment is bad and justifies my believe of abandoning iterative refinement phase.

The result in Fig.1.5. is presented here to show the ability of the application of handling larger amount of images. The images are very suitable for automatic joining and thus the resulting images are fairly good regardless of the deployment of iterative refinement. But the important point is the efficiency of the programs. Although exact time is not measured, the efficiency

between my implementation and the MATLAB implementation is so large such that there is sensible time difference in the experiment. Roughly speaking, my implementation needs one to two minutes, on my machine, to complete the joining process while the MATLAB implementation takes three to four times of time to get the same thing done. There are two reasons: generally speaking, C++ emphasizes on efficiency more than that MATLAB does; and the iterative refinement process is ditched in my implementation, which saves some time.

Finally, I want to discuss about an important experiment result. The application of automatic joiners is not adept at joining images of small scene. In the experiment, I took several photos of a computer. There was no large change in viewpoint orientation but the application thought they are irrelevant. This is also justified by the result in Fig.1.2. Look at the result in Fig.1.1., it is also taken by me using the same camera but all the images are recognized and joined relatively good. This shows that we need improvement or another algorithm for joining images of close scene.

Conclusion

In this project, I implemented an application of automatic joiners using C++ and I experiment both my and the MATLAB implementation using several set of images. The general procedures of my implementation and my understanding over the algorithms are discussed. The results of the experiment is also compared and discussed. Nevertheless, my implementation presents several deficiencies and thus needs improvements.