An Implementation on Recognizing Panoramas

Cansın Yıldız

Dept. of Computer Engineering
Bilkent University
Ankara,Turkey
cansin@cs.bilkent.edu.tr

***Abstract*— I implemented a panorama creation tool using [2] and [3]. Although this implementation gives excellent result on narrow panoramas of few images, it has some drawbacks when it comes to wider panoramas of multiple images. Causes for it will be discussed.**

I. Introduction

THIS report is a detailed presentation of the conducted research on *Recognizing Panoramas*. **Fig. 1** shows some results of implemented Panorama Creator. After presenting the details of the method used for image stitching, the obtained results will be explained.

II. Overview of the Method

The implemented application is a simplified version of the method proposed by *Lowe* [1] for creating panoramas. Similar to *Lowe*'s approach, the application first matches *SIFT features* between two given images. Then using those *SIFT feature pairs*, a *homography matrix* is calculated via *RANSAC*. This homography matrix is then used to warp one of the images onto other one.

Section III below explains the *Feature Matching* process in detail. Similarly, Section IV talks about the details of *Image Matching* technique.

III. Feature Matching

Feature matching process consists of two steps. First, SIFT features are extracted for two input images. Then, using those SIFT features some interest points are matched between those two images.

A. Compute SIFT Features

Scale-invariant feature transform (or *SIFT*) is an algorithm to detect local features in an image. SIFT features are very well-suited for image stitching problem because they are invariant to scale, orientation and affine distortion.

To compute SIFT features of each input images, I have directly used a MATLAB function `sift(imageName)` from *Lowe*'s *SIFT Keypoint Detector*. Demo software is provided at [3].

B. Match Interest Points

After SIFT features of two images are calculated, they must be matched. Those matched SIFT pairs will be used to compute homography later on.

To match SIFT features, I have adapted some code from a MATLAB function `match(imageName1,imageName2)`, similarly from *Lowe*'s work. In that code, they accepted two SIFT features to be a pair, if the angle between those features are less than a threshold. They used the dot products of SIFT features' coordinates to compute the angle.

IV. Image Matching

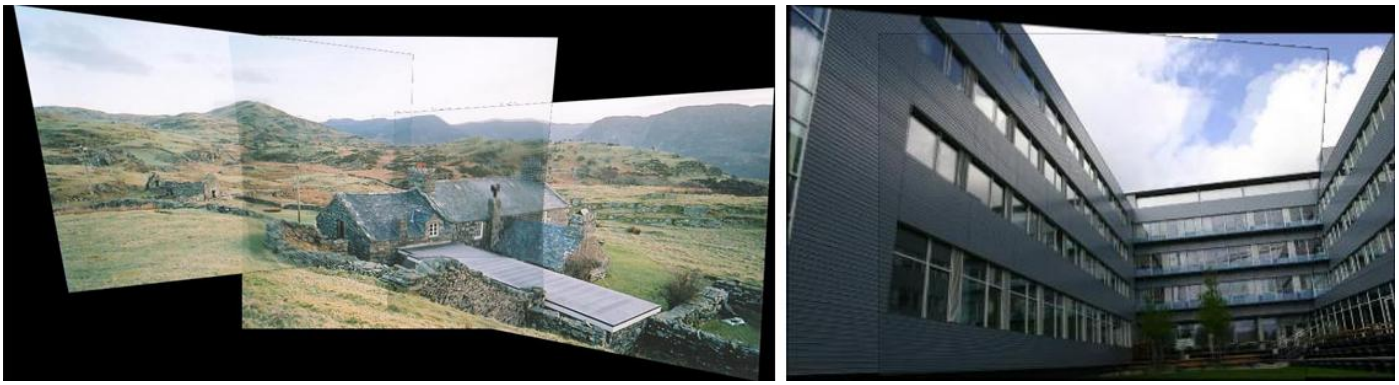

Fig. 1. Some Results of Panorama Creator.

After feature matching process is done, two images can be stitched together. Similar to feature matching, this image stitching process is also consist of two steps. First, a homography should be calculated via RANSAC. Then, using this computed homography the two input images can be stitched and blended into final output panorama.

A. Apply RANSAC to Compute Homography

To stitch two images together, a homography matrix between them should be calculated first. *Homography matrix* is a conversion matrix between two images;

$$p_b^{'} = H_{ab} p_a \quad (1)$$

where p_b and p_a are points from *Image B* and *Image A* respectively.

To estimate a robust homography matrix between two images, I have implemented *RANdom SAmple Consensus* (or *RANSAC*) method, which is summarized at **Fig. 2**.

1. Select four feature pairs (at random)
2. Compute homographyH (exact)
3. Compute *inliers* where $dist(p_b^{'}, H_{ab} p_a) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares H estimate on all of the inliers

Fig. 2. RANSAC Loop

1st step selects four random SIFT pairs. Then, 2nd step computes the homography matrix for those randomly selected SIFT pairs. I have used a MATLAB function `homography2d(sifts1,sifts2)` from *Kovesi*'s *Matlab Functions for CV and IP* [2] to compute homography matrix. The 3rd step computes an error measure between SIFT pairs after transforming the points using homography. To transofrm points, I have first derived *tform* object of homography using `maketform('projective',H')`, then using `tformfwd(tform,X,Y)`, I have transformed SIFT points. Then to calculate error, I simply take squared distance between each pair.

B. Stitch Images

After a homography matrix is estimated for input image pair, they can be stitched together. First, one of the images are transformed according to that homography matrix, and then this transformed image is blend with the other input image, while keeping the coordinate constraints of both images.

To achieve that I have first derived *tform* of homography matrix using `maketform('projective',H')`, then I transformed the image using `imtransform(im,tform)`.

After transformation is done, I have stitched images together using a simple linear blending. First, I am creating a mask for intersection section between images. Then with the help of that mask, I am blending images together.

V. Demonstration

This section demonstrates a sample run for two input images. Let's say, we have given two input images (see **Fig. 3**).

Fig. 3. Input Images

First step of application computes SIFT features of input images (see **Fig. 4**).

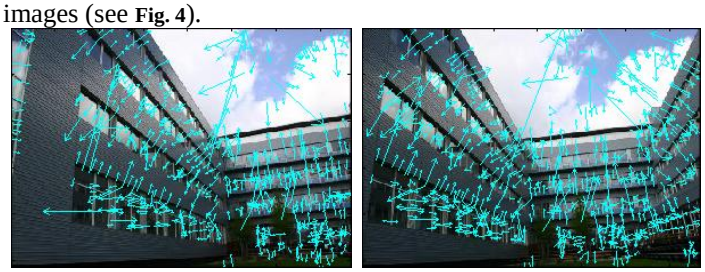

Fig. 4. SIFT Features

Then matches between those SIFT features are extracted (see **Fig. 5**).

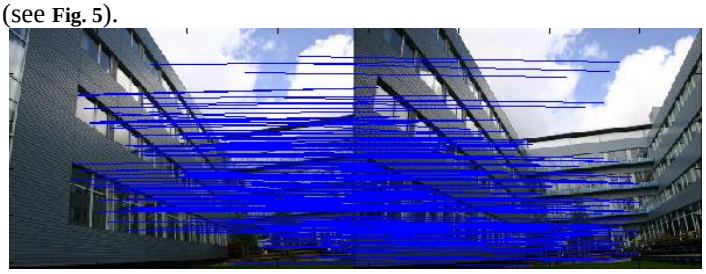

Fig. 5. Extracted Matches

Next step in application is to compute a homography matrix between images using those extracted matches via RANSAC. See **Fig. 6** for this sample's best four matches (which have most inliers).

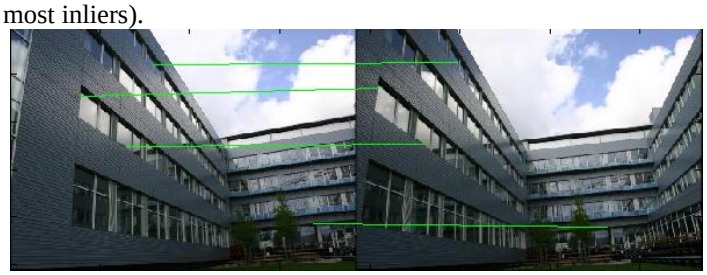

Fig. 6. Four Best Matches

The inlier and outlier pairs of those four best matches can be seen at **Fig. 7**. Red pairs are outliers and green pairs are inliers.

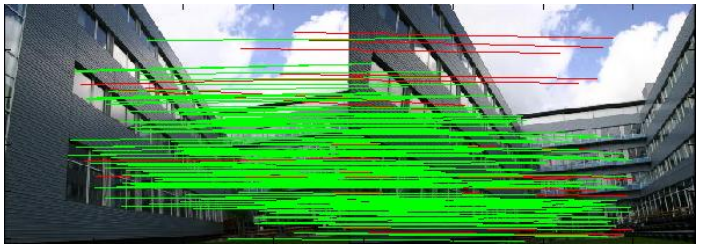

Fig. 7. Inliers and Outliers

Using all inliers of the four best matches, a homography matrix is calculated. It can be seen at **Fig. 8**.

$$\begin{bmatrix} 0.6055 & -0.0278 & -32.0494 \\ 0.0502 & 0.5703 & -9.5209 \\ 0.0002 & -0.0000 & 0.5349 \end{bmatrix}$$

Fig. 8. Homography Matrix

And finally the first image is warped according to this homography matrix on top of second image. Also a simple blending is performed at this step. You can see the output of the whole process at **Fig. 9**.

Fig. 9. Output Panorama Image

VI. Observations

Since the algorithm is well-thought and straight-forward, there are actually little or no parameters. The two most important parameters were the threshold to decide inlier counts and the number of times that RANSAC will run.

To observe the effect of inlier threshold, I have re-calculate panorama images for sample run at VI with thresholds 1, 0.1 and 0.001. As expected, the inlier count was dropped while threshold get smaller. For threshold 1, 0.1 and 0.001 the inlier counts were 184, 97 and 24, respectively. But surprisingly, I have found that an inlier count even as low as 24 does not affect the final result that much. Only change is observed, as I said, in inlier counts. You can see the inliers of those runs at **Fig. 10**, and the results can be seen at previous **Fig. 9**, since their results are visually indistinguishable.

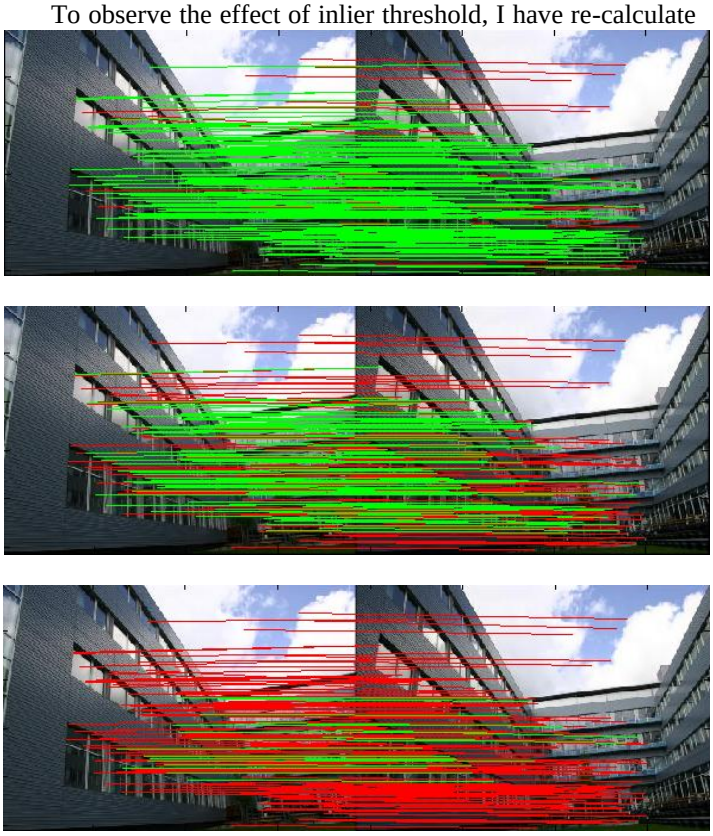

Fig. 10. Inliers/Outliers for Thresholds 1, 0.1 and 0.001.

One thing to note about thresholds is that, for too small and too high thresholds, the probability of getting a best four matches which are not actually very suitable for other pairs, will increase. Although, I cannot observe that phenomenon, I believe that there exists a threshold which should be called suitable. So, by having some empirical observations on well-paired images of mountains from *Lowe*'s work, I have decided that this 'suitable' threshold which has low risk of randomness should be 1, in my case.

The second parameter, number of times that RANSAC will run is similarly does not produce any remarkable changes in final result. All I can say that, a reasonable amount like 1.000, even 100 is enough for RANSAC to come up with a decent four best matches almost all of the time. And similarly, if RANSAC can't find a good four best matches at 1.000 loops, it cannot find any better matches at 100.000 loops either.

VII. Conclusion

The application that I have implemented had almost excellent results for panoramas of two images, even with perpendicular ones (see **Fig. 12**). Similarly, it also produced a good panorama of four images of a hut as well (see **Fig. 11**).

One deficiency of the implementation though, is about wide panoramas. If there are lots of images which are connected to each other like train cars, it cannot stitch those together. I believe that this happens because of the linear behavior of the homography matrix calculation. The perspective angles become so absurd that it cannot produce any good result after 3rd image.

I have researched how one achieved such good wide panoramas and found another article by *Lowe* [4]. In that article they have lots of post/pre-processing which are not included in this application, as I suspected. So, I assume that this is normal.

As I always do, I want to conclude the report by showing some results. Please find some examples of panoramas below.

References

[1] Brown, M. and Lowe, D. G. 2003. Recognising Panoramas. In *Proceedings of the Ninth IEEE international Conference on Computer Vision - Volume 2* (October 13 - 16, 2003). ICCV. IEEE Computer Society, Washington, DC, 1218.

[2] P. D. Kovesi. MATLAB and Octave Functions for Computer Vision and Image Processing. *Univ. Western Australia.* Available from: http://www.csse.uwa.edu.au/~pk/research/matlabfns/.

[3] D. Lowe. Demo Software: SIFT Keypoint Detector. *The University of British Columbia.* Available from: http://people.cs.ubc.ca/~lowe/keypoints/.

[4] Brown, M. and Lowe, D. G. 2007. Automatic Panoramic Image Stitching using Invariant Features. *Int. J. Comput. Vision* 74, 1 (Aug. 2007), 59-73.

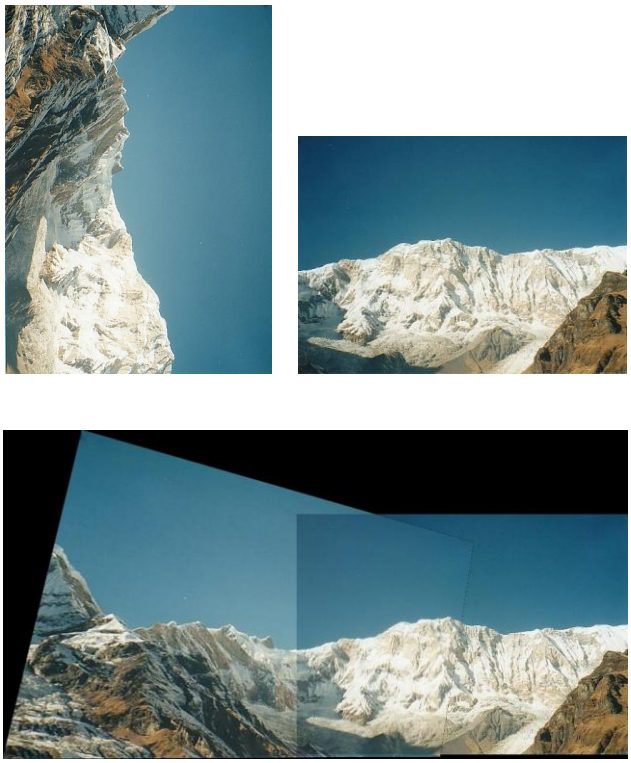

Fig. 12. Sample Run II – Input images are perpendicular.

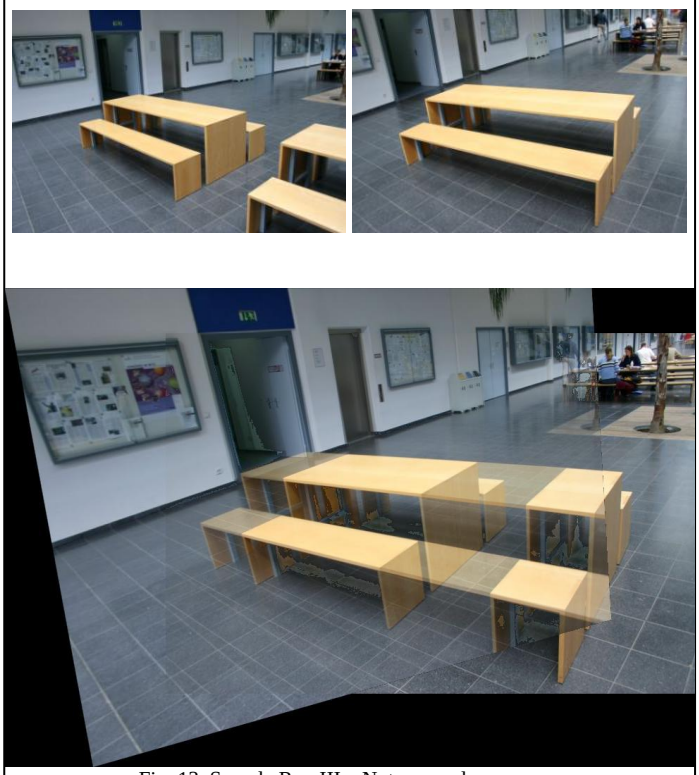

Fig. 13. Sample Run III – Not so good panorama.

Fig. 11. Sample Run I – Two level stitching is applied.