Recognizing Panoramas

Kevin Luo
Stanford University
450 Serra Mall, Stanford, CA 94305
kluo8128@stanford.edu

Abstract

This project concerns the topic of panorama stitching. Given a set of overlapping photos, we want to stitch them together into a panorama. I implemented the method described by Brown and Lowe and tested my method on a set of 46 unordered input images. My method correctly identifies and outputs four panoramas.

1. Introduction

Sometimes, a single image alone cannot fully capture an object of interest. For example, a tourist may have taken multiple photos of a mountain range from the same location, but none of these images alone can completely cover the extent of the mountain range. Using panorama stitching, we can combine these overlapping photos into a single panorama, enabling the tourist to picture the mountain range all at once. There are many commercial applications that provide panorama stitching capabilities. In fact, even smart phones now have the capability to take panoramas instead of ordinary photos. I implemented in MATLAB a simplified version of the invariant feature based approach described by Brown and Lowe [2, 3].

2.1. Review of previous work

Some methods for panorama stitching require users to place the images in the approximate regions of the panorama, and then they proceed to stitch together the images. Other approaches require a fixed ordering to the images. For example, the images might be from left to right in the panorama, or vice versa. However, these methods do not scale well since they require human input and are not fully automated.

On the other hand, direct methods attempt to minimize an error function of the intensity differences in the region of overlap [5]. These methods are not robust to illumination changes.

2.2. Description of method

My approach relies on Lowe's Scale Invariant Feature Transform (SIFT) [7] to identify feature descriptors. The keypoints associated with each feature descriptor have a characteristic scale and orientation in addition to the feature location, hence SIFT is insensitive to zoom as well as rotation in the input image. In addition, normalization of the vector of gradients in each frame makes SIFT invariant to affine changes in intensity.

My method is fully automated and does not make any assumptions about the ordering of input images. It can handle multiple panoramas simultaneously and can filter out noise images that do not belong to any panorama. Consequently, it can potentially accept any set of images as input, such as the photos on a camera flash card.

3.1. Summary of the technical solution

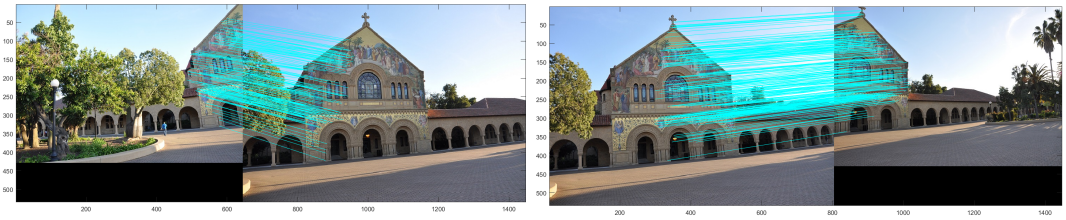
Given a set of unordered images as input, we first extract SIFT features from all of the images. We find matches between pairs of images and use RANSAC to detect and filter out outliers. Next, we verify the image matches using a probabilistic model and find the connected components of the image matches. For each connected component with at least two images, we estimate the homographies from each image to the center image. Then we perform bundle adjustment to minimize the sum of squared projection error over all matches, using the homography estimates as initial values of the parameters. Finally, we apply multi-band blending and render the panorama.

3.2. Technical details

Feature Matching

I downloaded the SIFT demo code by David Lowe [6]. For each image, Lowe's function `sift.m` runs the executable `siftWin32.exe` to extract the SIFT features from that image.

Then for each pair of images, I adapted the SIFT demo code in the function `getMatches.m` to obtain the indices



(a) SIFT matches

(b) Images aligned according to homographies

(c) Rendered with multi-band blending

Figure 1. Memorial Church. There are 170 and 263 correct feature matches between the two pairs of images. Input image sizes are 644×428, 804×534, and 644×428 pixels^2. Resulting panorama has size 2644×1135 pixels^2.

of the feature matches between each pair of images. Figure 1(a) shows two examples of the feature matches identified between pairs of images. The matches are plotted by the functions `plotMatches.m` and `appendImages.m`, both of which have been adapted from the SIFT demo code.

Image Matching

For each pair of images with feature matches, I applied the RANSAC algorithm in my implementation of `refineMatches.m` from Problem 3 of PS3 to detect and filter out outliers in each pair of matching images. RANSAC separates the original feature matches into inliers,

which are geometrically consistent, and outliers, which occur inside the area of overlap but are inconsistent. Since any two random images may potentially have some feature matches, I used the same probabilistic model from the paper by Brown and Lowe to verify correct image matches.

Let m be a binary variable denoting whether two images match correctly or not. For each i from 1 to the number of feature matches n_f, let $f^{(i)}$ be a binary variable denoting whether the ith feature match is an inlier or an outlier. Assuming that the $f^{(i)}$'s are independent Bernoulli, the total number of inliers $n_i = \sum_{i=1}^{n_f} f^{(i)}$ can be modeled by a Binomial distribution

$$p(f^{(1:n_f)}|m=1) = Binomial(n_i; n_f, p_1)$$
$$p(f^{(1:n_f)}|m=0) = Binomial(n_i; n_f, p_0)$$

where p_1 is the probability that a feature match is an inlier given a correct image match, and p_0 is the probability that a feature match is an inlier given an incorrect image match. Then the posterior probability that the two images match correctly given the set of feature matches can be calculated using Bayes' Rule

$$\begin{aligned} p(m=1|f^{(1:n_f)}) &= \frac{p(f^{(1:n_f)}|m=1)p(m=1)}{p(f^{(1:n_f)})} \\ &= \frac{1}{1 + \frac{p(f^{(1:n_f)}|m=0)p(m=0)}{p(f^{(1:n_f)}|m=1)p(m=1)}} \end{aligned}$$

An image match is accepted if $p(m=1|f^{(1:n_f)}) > p_{min}$. If we assume a uniform prior $p(m=1) = p(m=0)$, then this simplifies into a likelihood ratio test

$$\frac{Binomial(n_i; n_f, p_1)}{Binomial(n_i; n_f, p_0)} > \frac{1}{\frac{1}{p_{min}} - 1}$$

I used the same parameter values $p_1 = 0.7$, $p_0 = 0.01$, and $p_{min} = 0.97$ as in the paper by Brown and Lowe, which results in accepting an image match as correct if and only if

$$n_i > 5.9 + 0.22 n_f$$

This filters out cases with too few feature matches as well as those with relatively many outliers. If the image match is accepted, then the inlier feature matches are saved and the outliers are disregarded. I implemented the image match verification check in `main.m`.

The input images can be treated as vertices in an undirected graph, and for each correct match between two images, there is an edge with a weight equal to the number of feature matches. Then we can use depth-first search (DFS) to identify the connected components of the graph. The set of images in each connected component form a panorama. Any noise images will fail to match other images and form isolated components, which this method subsequently ignores. I implemented the functionality of depth-first search in the functions `dfs.m` and `visit.m`.

Finding the Center Image

In order to obtain a rough approximation of the homographies, we can find the maximum spanning tree T of the connected component, which eliminates edges with low number of feature matches in favor of those with more matches. I implemented a naïve version of finding the maximum spanning tree in the function `getMST.m`.

Then we can compute the pairwise projective homographies between pairs of matching images in T. I implemented this in `getTform.m`, which uses the MATLAB function `estimateGeometricTransform` with the "projective" option. Since we can view the resulting panorama from any orientation, we can minimize the total panorama area by using the orientation of an image that is close to the center of the panorama to reduce the stretch of images at the ends of the panorama when rendering in two-dimensional Cartesian coordinates.

Given a potential center image c, we initialize its transformation H_c to the identity matrix. Then we perform DFS on the maximum spanning tree T of the connected component, starting from image c, to estimate the transformations for the rest of the images. When we visit image i, we can compute the transformation for image j for each edge (i, j) in T where j has not yet been visited

$$H_j = H_{ji} H_i$$

where H_{ji} is the pairwise homography from image i to image j and H_i is the transformation corresponding to image i. By propagating the pairwise homographies throughout T, we can generate estimates of the homographies $H_i = H_{ci}$ from each image i to image c. I implemented this in the functions `getTforms.m` and `updateTforms.m`.

Then we can use these homographies to estimate the area of the resulting panorama, which is implemented in the function `getPanoramaSize.m`. The center image is chosen as the potential image that minimizes the panorama area.

Bundle Adjustment

Given the estimated transformations from the previous subsection as initial points, we can perform bundle adjustment to refine the homographies from each image to the center image by minimizing the error function, which is the sum of the squared projection error across all matches.

Let u_i^k denote the Euclidean coordinates of the kth feature in image i. Given a correspondence $u_i^k \leftrightarrow u_j^\ell$ between images i and j, the residual is

$$r_{ij}^k = u_i^k - p_{ij}^k$$

Figure 2. 46 unordered input images consisting of four panoramas and three noise images. Six images have been resized, and six others have been rotated. Most of the images have size 644×428 pixels^2.

where p_{ij}^k is the projection (in Euclidean coordinates) of point u_j^ℓ from image j onto image i. In homogeneous coordinates,

$$\tilde{p}_{ij}^k = H_{ij}\tilde{u}_j^\ell$$

where

$$H_{ij} = H_{ic}H_{cj} = H_{ci}^{-1}H_{cj} = H_i^{-1}H_j$$

is the homography from image j to image i. Then the error function is the sum over all images of the squared projection error

$$e = \sum_{i=1}^{n} \sum_{j \in \mathcal{I}(i)} \sum_{k \in \mathcal{F}(i,j)} ||r_{ij}^k||_2^2$$

where n is the number of images, $\mathcal{I}(i)$ is the set of images matching image i, and $\mathcal{F}(i,j)$ is the set of feature matches between images i and j. This is a non-linear least squares problem that can be solved using the Levenberg-Marquardt algorithm.

As a starting point, I used the estimated projective homographies $H_i = H_{ci}$ from each image i to the center image c, which have been computed in the previous subsection. I rescaled each homography so that its last component is 1, and then organized the remaining 8 elements of each homography into a vector of parameters `Phi`. I implemented the above error function in `projectionError.m` with `Phi` as one of the arguments, then called `lsqnonlin` with the Levenberg-Marquardt option to optimize the error with respect to `Phi`.

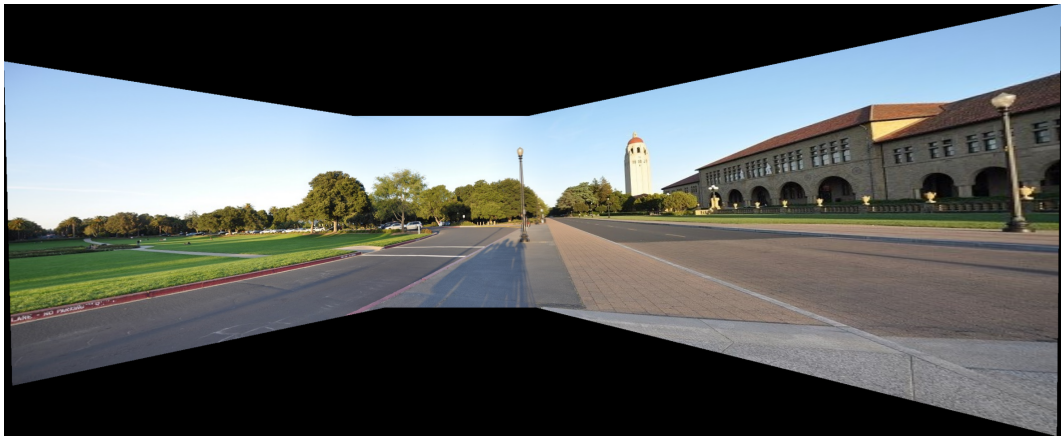
In the first round, I started with the two images with the highest number of matches and called `lsqnonlin` to minimize the error. Then I added in another image with the most matches to one of those two images, called `lsqnonlin`, and repeated until all images in the component had been added. I implemented selection of the ordering of the images in the functions `getOrdering.m` and `getEdges.m`.

Multi-band Blending

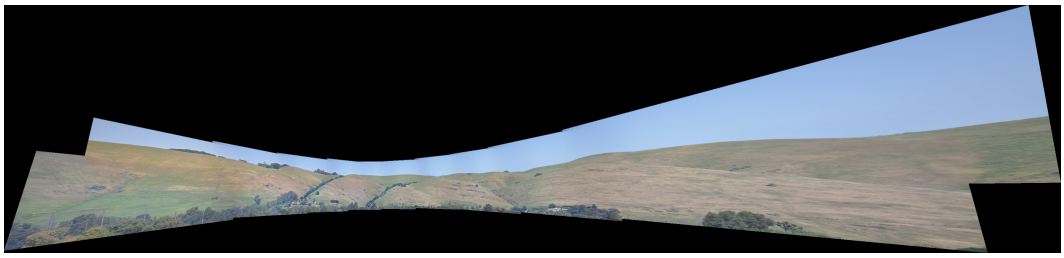
For multi-band blending, I followed the general approach outlined in the Image Pyramids and Blending lecture of the course Computational Photography at CMU [4]. I used a weight function $w(x,y) = w(x)w(y)$, where $w(x)$ and $w(y)$ vary linearly from 1 at the center of the image to 0 at the edges. This is implemented in my function `getWeight.m`.

I applied the homographies H_{ci} on both the images and the weights to transform them onto the plane of the panorama. Then for each transformed image I constructed a Laplacian pyramid L_i with two levels, corresponding to low and high frequencies, as well as a two-level Gaussian pyramid G_i for each transformed weight. Next, I formed a combined pyramid where each level is the weighted average of the images. Finally, I collapsed the two levels of the

(a) Output panorama 1: Mission Peak, Fremont, 18 images, 4090×733 pixels^2.



(b) Output panorama 2: The Oval, 3 images, 2362×964 pixels^2.



(c) Output panorama 3: Fremont hills, 19 images, 3575×839 pixels^2.

(d) Output panorama 4: Memorial Church, 3 images, 2644×1135 pixels^2.

Figure 3. Panoramas produced from the 46 input images.

combined pyramid to obain the final blended image.

I found an example of multi-band blending two images in MATLAB online by Hao Jiang, Boston College [1]. The weights are computed in my function `getWeight.m`. For rendering the panorama, I adapted the panoramic image stitching example from MathWorks [8] in `getPanorama.m`.

The effects of applying multi-band blending are shown in Figure 1. The panorama in Figure 1(b) is rendered without any blending, and there are visible seams between the individual images in the panorama. In Figure 1(c), the same panorama has been rendered with multi-band blending, and the seams are no longer visible.

4. Experiments

To acquire data for my experiment, I took overlapping pictures at Lake Elizabeth in Fremont, the Stanford Inner Quad Courtyard, and the Stanford Oval, using a Nikon D90 camera. Since my approach can handle the constituent photos of multiple panoramas simultaneously, I combined all of my images into one set. I also included several noise images taken by the same camera. Then I resized all images from 3216×2136 square pixels down to 644×428 square pixels to make matching more manageable. I enlarged three images up to 804×534 square pixels and reduced three images down to 483×321 square pixels, to simulate differences in zoom. I also rotated two images by $90°$, two by $180°$, and two by $270°$, to simulate differences in rotation. Then I scrambled the order of the images and placed them in the same directory, `data46`. The images are shown in Figure 2 (plotting handled by my functions `plotImages.m` and `appendImages.m`). There are 46 total images consisting of four panoramas and three noise images. The dimensions of the panoramas are 4090×733, 2362×964, 3575×839, and 2644×1135 pixels^2.

Then I ran my method on MATLAB on my personal laptop (2.4GHz Windows PC) with the Computer Vision, Image Processing, and Optimization Toolboxes installed, using my function `main` with the directory name as input. The algorithm completed after about 250 seconds, ignored the noise images, and correctly identified the four panoramas, shown in Figure 3.

For the most part, the panoramas seem to be well-aligned, although there are some misalignments in the arches of the Memorial Church panorama in Figure 3(d). Since the panoramas are rendered in two-dimensional Cartesian coordinates, the images at the ends of the panorama are stretched horizontally much more than the ones in the center, especially in the Fremont hills panorama in Figure 3(c).

5. Conclusions

As seen in the previous section, this fully-automated approach is robust to scale and orientation of the input images, as well as to noise images that are not part of any panorama. However, since I rendered my panoramas in Cartesian coordinates, this method cannot handle wide-angle panoramas, since the images along the edge of the panoramas get stretched more and more as the field of view increases. Therefore, one future work could involve cylindrical or spherical mapping so that 180 degree panoramas can be rendered on a plane.

In addition, my approach is rather slow and does not scale well since it considers all pairwise matches between images. In reality, each image only matches a small number of others, even if they are all in the same panorama. Therefore, another future work could involve finding the k nearest-neighbors for each feature and only considering the top candidate matching images to each image, as opposed to all of them. Implementing this would significantly improve the efficiency of this approach.

The code for my project can be found online at GitHub: `https://github.com/kluo8128/cs231_project`. The directory `data46` contains the 46 input images. There is also a second data directory, `data8`, which contains 8 images, 2 panoramas and 1 noise image.

References

[1] Multi-band image blending in compact matlab codes — computer vision notes, 2015. [Online; accessed 2-June-2016].

[2] M. Brown and D. Lowe. Recognising panoramas. In *Proceedings of the 9th International Conference on Computer Vision*, volume 2, pages 1218–1225, Nice, October 2003.

[3] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, 2007.

[4] A. Efros. Cmu 15-463: Computational photography, 2005. [Online; accessed 2-June-2016].

[5] M. Irani and P. Anandan. *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, chapter About Direct Methods, pages 267–277. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[6] D. Lowe. Keypoint detector, 2005. [Online; accessed 2-June-2016].

[7] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[8] MathWorks. Feature based panoramic image stitching, 2016. [Online; accessed 2-June-2016].