Image Stitching and Blending

Xiang Wang
Department of Intelligent
Peking University
wx@stu.pku.edu.cn

1 Introduction

In this project, I achieved image Stitching and blending step by step, and finally generated the panorama.[1] After that, I kept on adjusting my hard-coded parameters to analyze their influence and finally got higher quality of panorama. Finally, I used different feature detectors & descriptors[2] and advanced blending techniques[3] to improve the quality of panorama.[4]

2 Implement

2.1 Harris Corner Detector with non-maximal suppression

2.1.1 Calculate spatial derivative

Spatial derivative can be simply produced with Soble filter. But before we produce spatial derivative, we need to do some important preprduce. The original data type is uint8, and it will lead to data truncation, which means you can't get a negative derivatives. But when we calculate the direction of derivative, we do need negative derivatives. In order to prevent data truncation, I reset the data type to float. For simplification, the input image needs to be converted to grayscale. As usual, I use Gaussian filter to smooth the image, which can remove part of noise. Finally, we can calculate spatial derivative with Soble filter of 'reflect' mode, which gives a zero gradient at the edges.

2.1.2 Calculate Harris response

Harris response defines as:

$$R = det(\mathbf{M}) - \alpha trace(\mathbf{M})^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

I choose to calculate $det(\mathbf{M}) - \alpha trace(\mathbf{M})^2$ because I can make full use of matrix operation in numpy to remove for loop to improve the efficiency. As for gaussian weight, I just reuse the function in $hybrid.py$. The process is as below:

$$\mathbf{Ix} = gradient\_x, \mathbf{Iy} = gradient\_y,$$
$$\mathbf{A} = Ix * Ix \otimes Guassian_filter,$$
$$\mathbf{B} = Ix * Iy \otimes Guassian_filter,$$
$$\mathbf{C} = Iy * Iy \otimes Guassian_filter,$$
$$det(\mathbf{M}) = \mathbf{AC} - \mathbf{B}^2$$
$$trace(\mathbf{M}) = \mathbf{A} + \mathbf{C}$$

Here operator $\otimes$ means convolve.

[1] $main\_project.py$
[2] $detectors\&descriptors.py$
[3] $blend.py$
[4] $panoramas\_modes.py$ combines these parts to generate panoramas.

2.1.3 Select candidate corners and non-maximal suppression

First, I set all numers that are smaller than threshold to zero. Then I use maximum filter to achieve non-maximum suppression and then set those which aren't local maximum to zero. Finally, the interesting points are non-zero numbers left in matrix.

2.2 Histogram of gradients and feature vector construction

Rather than process original image, I use padding to prevent 'index out of range'. For each interesting point, I choose the 4×4 blocks around it, and each of blocks is a 4×4 pixels matrix, that's 16×16 in total. I divide the region into 8 directions . For each blocks, I calculate the number of derivatives in those directions and normalize the Histogram. After that, I can select the prominent gradient and take it as principle orientation. Then I rotate it's neighbor to fit principle orientation and calculate the histogram again. Note: the neighbor is a 32×32 block, because if you just rotate 16×16 block, you are bound to miss several pixels, which lead to errors. In this part, I refer to SIFT descriptor, so I use the histogram to built feature vector. It's a vector of $4 \times 4 \times 8 = 128$ dimensions.

2.3 Local feature matching

After we get feature vectors corresbonding to interestng points, we can calculate the distance between each pair of interesting ponit of two images. For simplification, I calculate euclidean distance. However, I take some tricks to prevent errors when matching images with similar patterns. I take the image with smaller amount of interesting points as enumerating image. For each interesting point in enumerating image, if the result of minimum distance dividing the second minimum distance is smaller than threshold, I consider it as a successful matching and set them as a pair. That means the minimum distance should be 'prominent' enough to be considered as successful matching. After matching is finished, if rate of successful matching is less than 0.3, I consider these two pictures as 'Fail to Match' and throw an error.

2.4 Image stitching and blending

2.4.1 Compute the homography matrx of image pairs

Our goal is to calculate:

$$minimize \, \|\mathbf{A}_{2n\times 9} h_{9\times 1}\|$$

The interesting points we got above are in Cartesian coordinates, so we have to convert it to homogeneous coordinates first. Then, construct $\mathbf{A}_{2n\times 9}$ using slicing operations with numpy. Finally, find the eigenvector of $\mathbf{A}^T\mathbf{A}$ with the smallest eigenvalue using the Singular Value Decomposition (SVD) and reshape it to 3×3 matrix. That's exactly what we want.

2.4.2 Align the image with RANSAC

Considering the limitation of accuracy of feature matching, I suppose that there are 70% outliers, so I need about random selection for $\left\lceil \frac{\log(1-0.99)}{\log(1-0.3^4)} \right\rceil = 567$ times to get an accuracy of 99% For each estimated model, I use euclidean distance to decide whether each points 'agree with' this model. The model with most 'agreement' will be my estimated homo-matrix.

2.4.3 Stitch and blend the image

First, project four corner pixels with estimated homo-matrix and converting them back to Cartesian coordinates after normalization. Together with four corner pixels of the other image, we can get the size of new image plane. Then, remap both image to new image plane and blend two images using Alpha Blending.

2.4.4 Generate a panorama

After all things done above, generating a panorama is quite simple. The only thing we need to do is choose the middle one in ordered image list as principle image, and then blend its neighbor images to its plane one by one until we blend all images to principle image. And the final principle image is exactly the panorama we want.

3 Analysis

The quality of the stitched image is determined by lots of factors, Below I will analyze the impact from four aspects: some hard-code parameters, the way I move the camera during shooting, different kinds of feature detectors & descriptors, different kinds of blending techniques. And last two aspects will be presented in Appendix A.

3.1 Hard-code parameters

Sigma of Gaussian filter There are two parts where I used Gaussian filter. Before calculating spatial derivatives, I use Gaussian filter to remove noise. In this function, the larger sigma is, the smoother image is. As a result, when processing images full of trees or grasslands or other images which are lack of corners, we can't set sigma too large, otherwise, we can't get enough interesting points. On the contrary, when processing images full of obvious corners, we should set sigma larger to remove more noise. When calculating Harris response, I use Guassian filter as weights. The smaller sigma is, the more the center of corner contributes to response function.

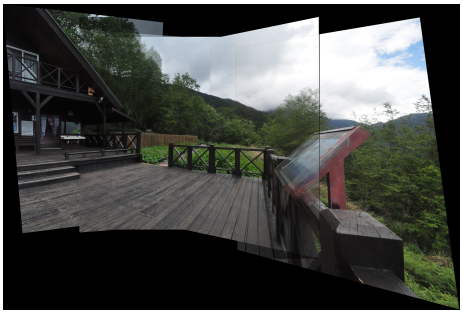
Threshold In function $corner\_selection$, the larger threshold is, the less candidates are chosen. In this part, threshold is normally settled as 0.01 times of maximum numbers in response matrix, so did I. What's more, the parameter $min\_distance$ works as threshold for non-maximal suppression. The larger threshold is, the less candidates are chosen, and the distance between two candidates are larger. In function $feature\_matching$, threshold determines whether the matching is successful. Smaller threshold means the minimum distance should be more 'prominent' to be successful matching. In this way, we can prevent errors when matching images with similar patterns. Smaller threshold usually contributes to higher accuracy, but if it is too small, it will lead to quite low rate of successful matching. In function $align\_pair$, threshold affects the amount of inliers. The smaller threshold is, the less inliers are.

Adjust hard-code parameters Considering the difference between inputs sets, I made targeted adjustments for each set of inputs and ended up with better results below:

(a) **grail**

(b) **library**

(c) **parrington**



(d) **xue-mountain-entrance**

Figure 1: **Harris Corner detector** & **SIFT descriptor** & **Alpha Blending**

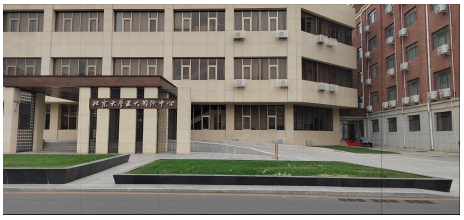
The source images of panoramas in Fig. 1 are listed in Tab. 1.

Table 1: **Source of panoramas**

Figure	Source	amount
Fig. 1a	grail 07 to 11	5
Fig. 1b	library 10 to 13	4
Fig. 1c	prtn 00 to 04	5
Fig. 1d	DSC 186 to 173	5

3.2 The way how you move the camera during shooting

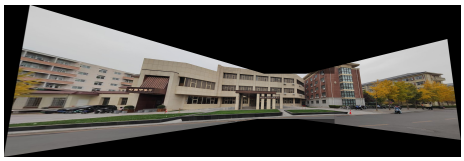
Translation and rotation. In order to make comparison, panoramas are generated from three images, corresbonding to different settings for shooting the image sequences, including translating only(small & large), rotating only(small & large), translating and rotating. The result are showed in Fig. 2. As expectation, translation has less influence on quality than rotation. And smaller movements lead to better panoramas. The movement of camera means movement of objects in images. The larger the movements, the more different objects and features are, which makes it harder to match images.



(a) **Smaller translation**

(b) **Larger translation**

(c) **Smaller rotation**

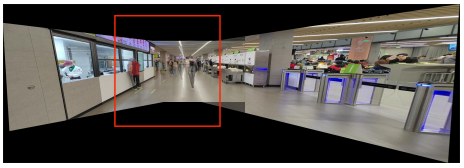


(d) **Larger rotation**

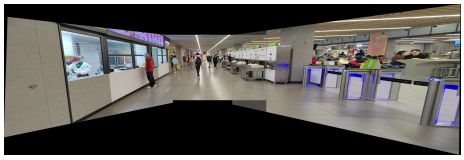
(e) **Translation** & **rotation**

Figure 2: **Movements**

Panorama with some objects moving. To generate panorama from images with something moving is much more difficult. There might be ghosts in the areas where two images blend. However, it also means that ghosts can only appear in those area. This phenomenon provides us with a potential solution. When shooting a sequence with some objects moving, try to contain the moving objects as closer to center as you can. Or preproduce the images to avoid the moving object appear in blending area at the same time. This little trick works well, and the result are showed in Fig. 3 and inputs after preprduce are showed in Fig. 4. In terms of algorithm, we can use Temporal Difference, GMM, ViBe or other advaced techniques to remove ghosts.



(a) **Ghosts**



(b) **Without ghosts**

Figure 3: **Handling ghosts**

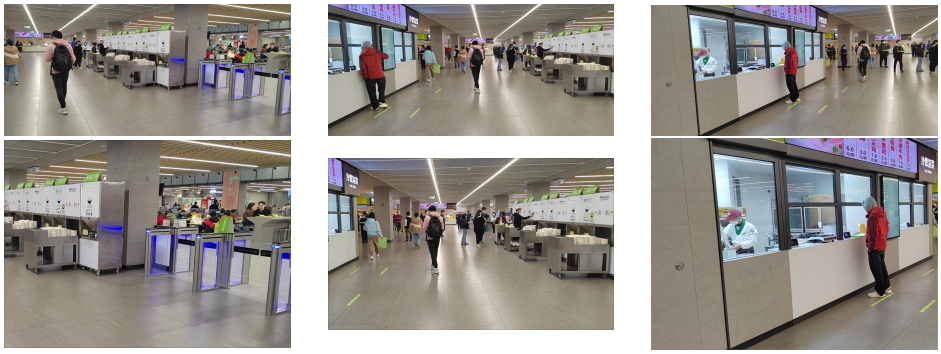


Figure 4: **Inputs of ghosted panorama**

Panorama with the same person appears multiple times. Based on what we have done above, this task is easy to implement. But we need to pay attention not to let this person to be ghost!

Figure 5: **I'm everywhere!**

A Appendix

A.1 Feature detectors & descriptors

What I've implemented above is Harris Corner detector and SIFT detector, and it performs well in most situations. However, when processing images in $library$ image set, it performs worse and it can only stitch up to four images. And it may works badly when you try images full of grasslands. That's easy to understand. Harris Corner detector detects corners as interesting points, however, the grasslands and trees are too 'noisy'. if I set a large sigma to smooth the image, I'll lose most of corners at the same time. But if I set a small sigma, I'll get too many corners and their patterns are nearly the same, which will lead to bad matching.

In order to improve the quality of panoramas, I try to use more kinds of feature detectors and descriptors, including SIFT, Speeded-Up Robust Features(SURF) [5], Accelerated-KAZE(AKAZE), Binary Robust Invariant Scalable Keypoints(BRISK) and Oriented FAST and Rotated BRIEF(ORB).

[5]SURF needs lower version of OpenCV and its performance is similar to SIFT but more faster. For simplification, I just use the results of SIFT to represent its performance.

With these detectors & descriptors, I try to blend five images for each panoramas sets. The results are showed in Fig. 6. [6]

As you can see, SIFT&SURF and BRISK can works great even when precessing images with large rotation or massive trees and grasslands. AKAZE performs well in most of situations unless the images are shooted with large rotation. However, ORB performs worse, even worse than my program. Sometimes it goes totally wrong. I was surprised even if I had anticipate it would perform worse, as the relavant materia showed that ORB is faster but worse than SIFT & SURF. I checked the usage of ORB again on website of OpenCV, and I didn't make mistakes when calling the function.

The results shows that SIFT is still more powerful and robust. But it is too slow, and SURF can do it faster and get similar quality of panoramas. It seems that AKAZE and BRISK are worse. but actually AKAZE is based on nonlinear interpolation so it can get better quality sometimes. And BRISK performs best when registering images with large blurs. SIFT & SURF just look best on these datasets, it doesn't mean they are always the best. So when we choose detectors and descriptors, we'd better to take the details of inputs into consideration, and then we can choose proper algorithm to get better quality of panoramas.

A.2 Blending techniques

What I've implemented above is alpha blending. Actually it's just take a weighted average where images overlap. As a result, if the stitching is not perfect, the panorama may be ghosting in areas where images overlap. What's more sometimes, the stitching boundary is obvious. That means I need more powerful blending techniques, so I explored to use some advanced blending techniques to improve the quality of panoramas. The results is showed in Fig. 7. [7]

As you can see, multiband blending performs great, it totally removed the stitching boundary. And it mitigates image ghosting. This technique is easy to implement, I didn't use third-party codes. Maybe I should use this technique when I begin this project, so that I can get better quality. Considering academic integrity, I won't use this technique to rewrite my program to pretend I get great quality at the first time.

Then comes poisson blending. It performs well as it totally removed the ghosting. You can see the details clearly rather than see something double. But the stitching boundary is still obvious. The advantages of poisson blending are not illustrated well in this input set. Actually it is poweful when the images have different illumination. Poisson blending can blend these images very naturally while other techniques can't do that.

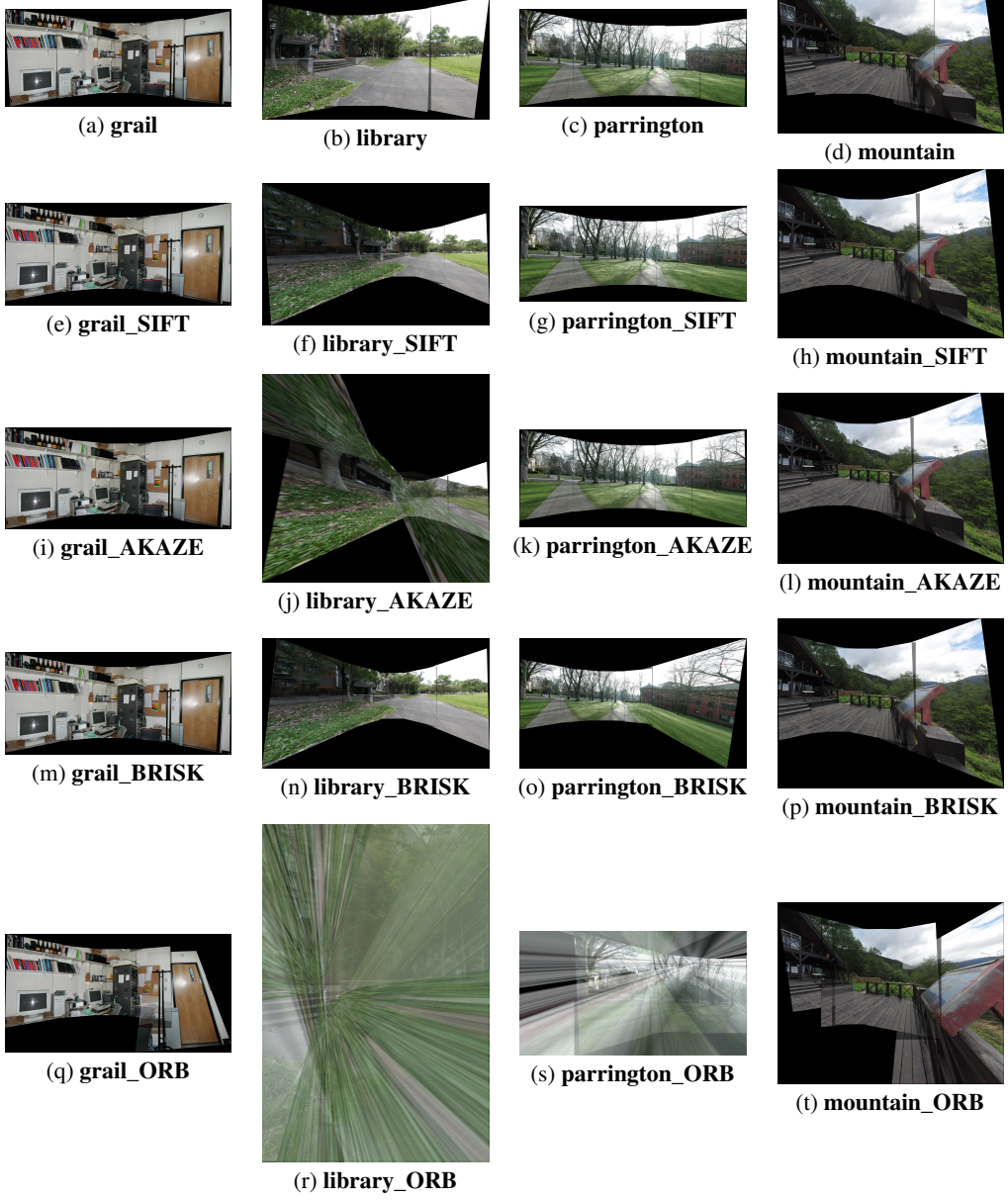
The last technique I use is pyramid blending, when processing two images, it performs better than any other blending techniques above. It totally removed stitching boundary and image ghosting. What's more, the blended looks so smooth that maybe you think it as the original image with no blending. However, the disadvantage is that it will perform worse when blending more images to generate panoramas. With more and more images blended, the front blended image will be so blur that we can't even extract features from it, which lead to wrong matching and stitching.

The results shows that these techniques have their own advantages and disadvantages. When handling spcific problems, we should use targeted techniques to get a better quality of panoramas. And that's how science goes forward. When facing a problem that we can't solve it well with old techniques, we will try to make targeted adjustments until we come up with a new powerful technique.

Finally, I applied multiband blending to improve the quality of other intput sets. The results are showed in Fig. 8.

[6] Some performs well when blending less images, but in order to make comparison, the results are forcibly blended from 5 images, except the first row, which are results in Fig. 1.

[7] In this part, panoramas need to be showed in larger size, and I can't spend too much space to cover four input sets. So I just choose to illustrate with panoramas produced from the first input sets.



(a) **grail** (b) **library** (c) **parrington** (d) **mountain**

(e) **grail_SIFT** (f) **library_SIFT** (g) **parrington_SIFT** (h) **mountain_SIFT**

(i) **grail_AKAZE** (j) **library_AKAZE** (k) **parrington_AKAZE** (l) **mountain_AKAZE**

(m) **grail_BRISK** (n) **library_BRISK** (o) **parrington_BRISK** (p) **mountain_BRISK**

(q) **grail_ORB** (r) **library_ORB** (s) **parrington_ORB** (t) **mountain_ORB**

Figure 6: **More detectors** & **descriptors**

(a) **Alpha blending**

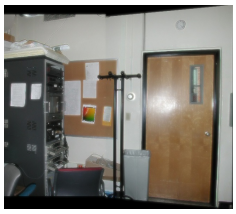
(b) **Multiband blending**

(c) **Poisson blending**

(d) **Pyramid blending**

(e) **Pyramid blending**

(f) **Pyramid blending**

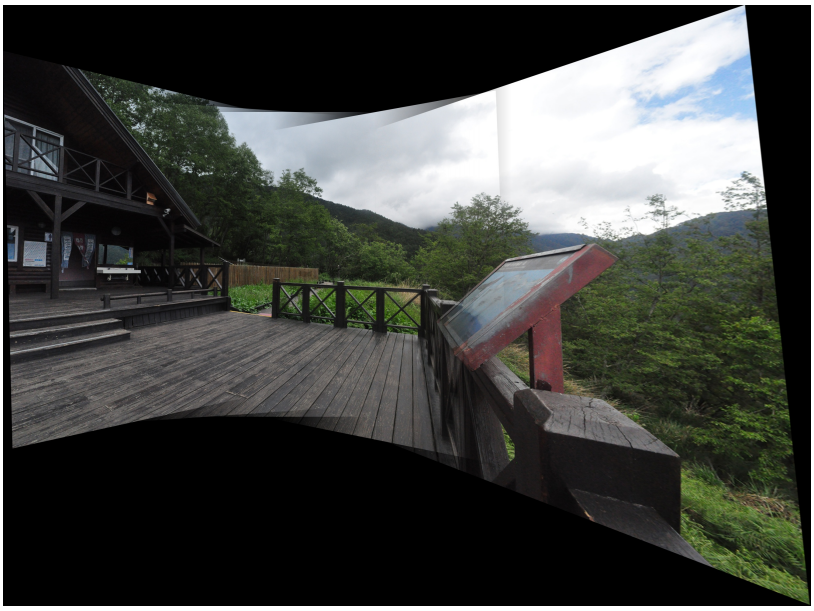


(g) **Pyramid blending**

Figure 7: **Blending Techniques**

(a) **Library**

(b) **Parrington**



(c) **Mountain**

Figure 8: **Apply multiband blending to other three input sets**