**Homework 2 : Image Stitching**

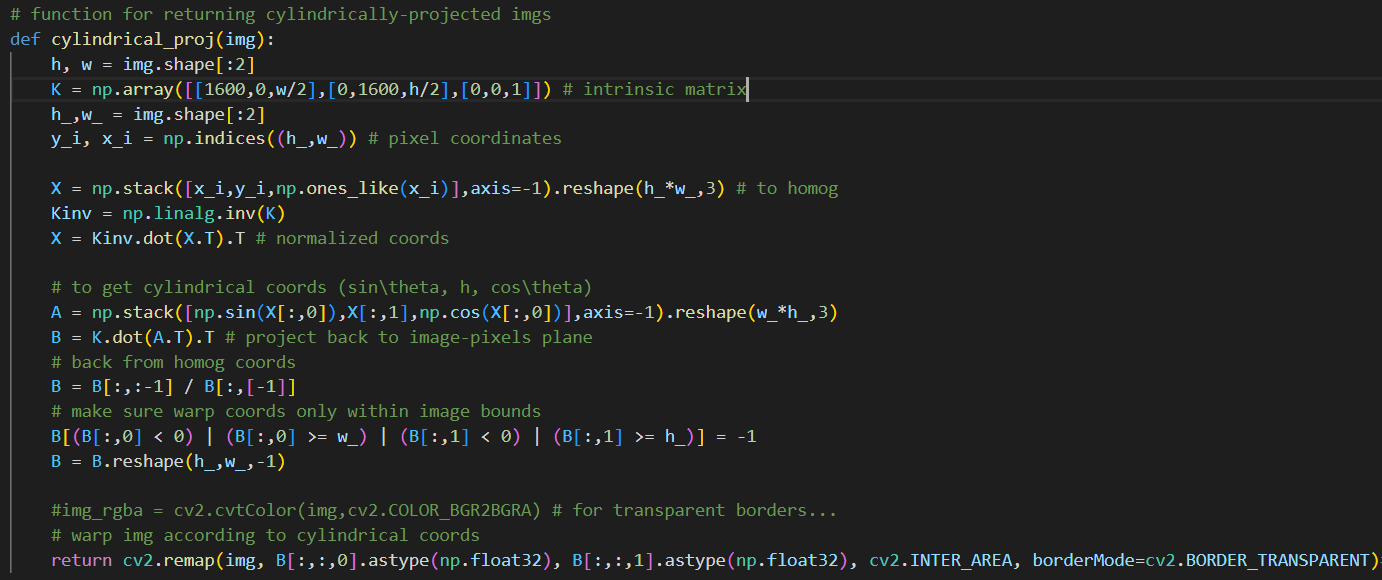
109550135 范恩宇

1. **Implementation :**
2. **Cylindrical Projection**

If we don’t do cylindrical projection to the images, it may be distorted like the following. For “Base” part, the left most one could be distorted a lot.



As for cylindrical projection, I extract height & width of the images, create an intrinsic matrix, and get cylindrical coordinates. In the end, warp the images according to cylindrical coordinates we just get.

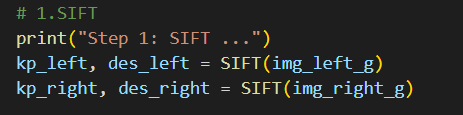


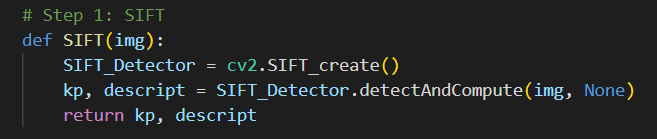
The level of image-bending depends on the constant in intrinsic matrix, the smaller the constant, the more it bends. After several trials, 1600 seems to be the best choice.

|  |  |
| --- | --- |
| Constant = 1000 | Constant = 1600 |
|  |  |

1. **SIFT**

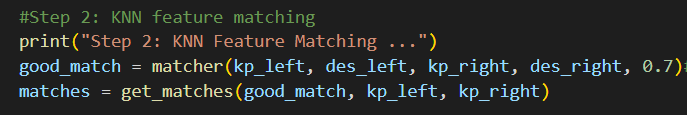
Here I just followed the slides, using OpenCV functions to extract SIFT features. In this case, “kp” means OpenCV key points, which stores the point information. While “descript” is an array storing gradient information.





1. **KNN Feature Matching**

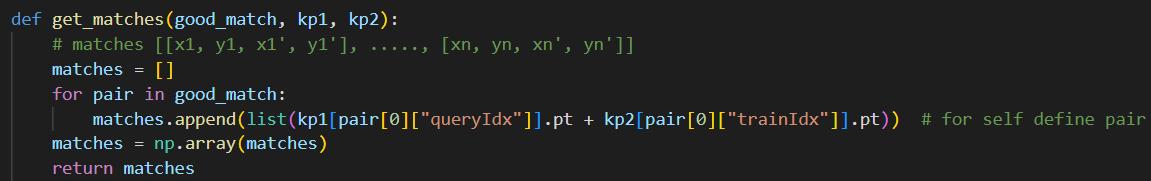
Here I apply KNN algorithm (with brutal force) & Lowe’s Ratio test to find good matches, then debug with cv2.BFMatcher( ) as the slide suggests. Motivated by “dmatch” in OpenCV which stores match information, I create some dictionaries for the same purpose.



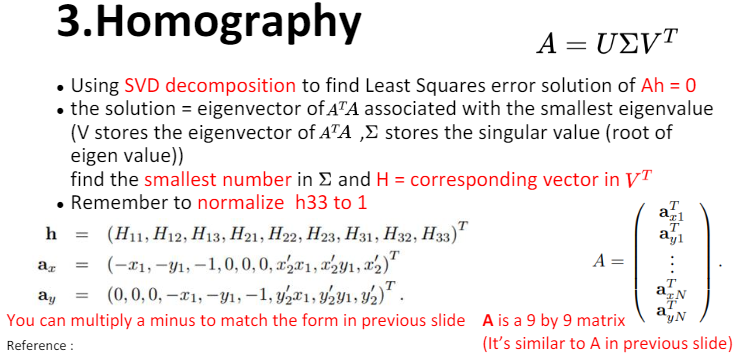
In order to find the least and 2nd least distance, I make two dictionaries(dmatch & dmatch\_2nd) store information and use np.linalg.norm( ) to accelerate the calculation.

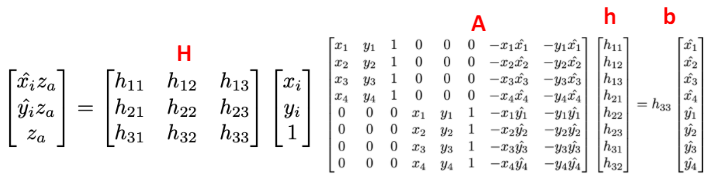
After getting the distance, do Lowe’s Ratio test. In the end, store information of matches in an array and obtain information of it through get\_matches( ).



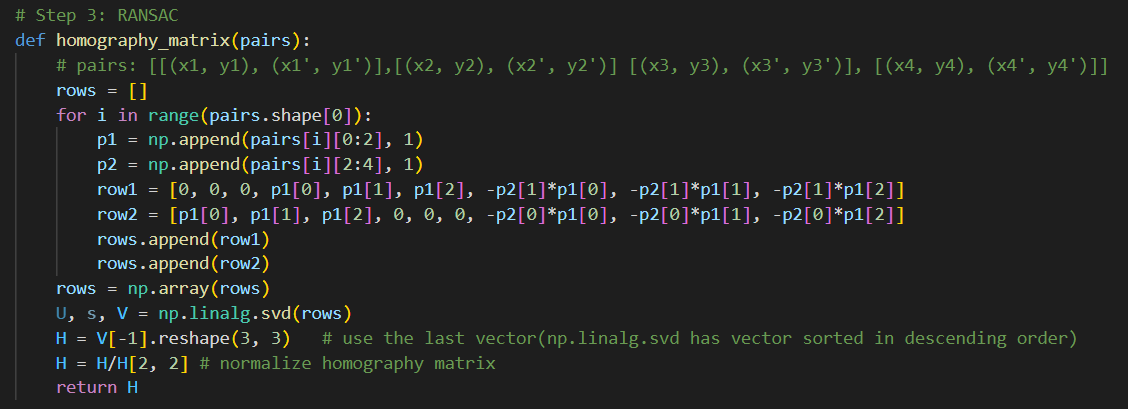


1. **Homography Matrix**



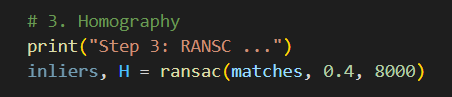


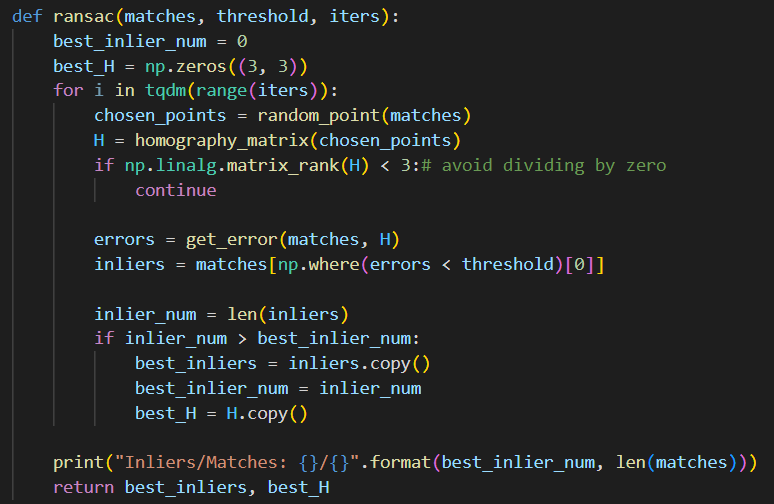
Through the hints we get from slides, I use np.linalg.svd( ) to solve the least squares problem by pseudo inverse. Of course, also normalize the matrix after getting the first result.

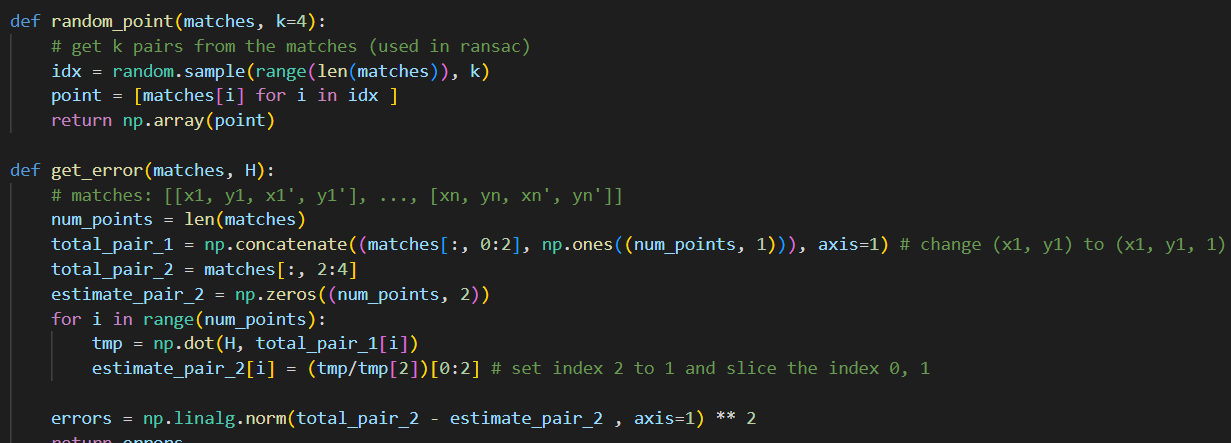


1. **RANSAC**

Here I basically follow the pseudo code in slide, while the iterations and error threshold are decided through several tests. To get random pair of matches and error value, I also write functions for them.



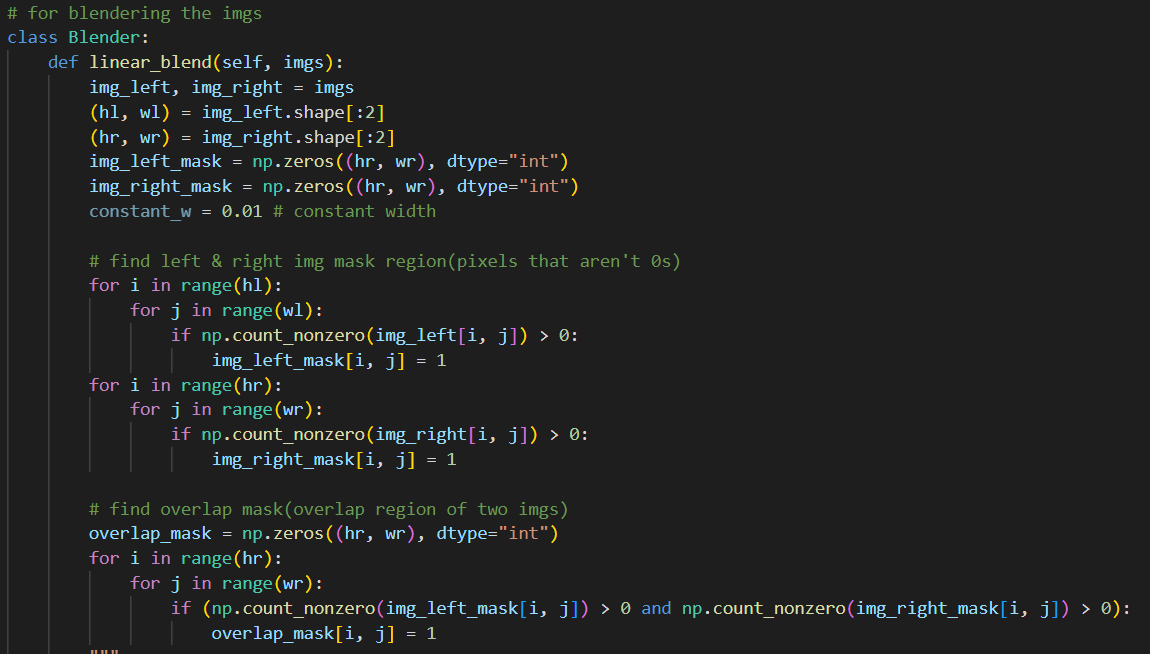




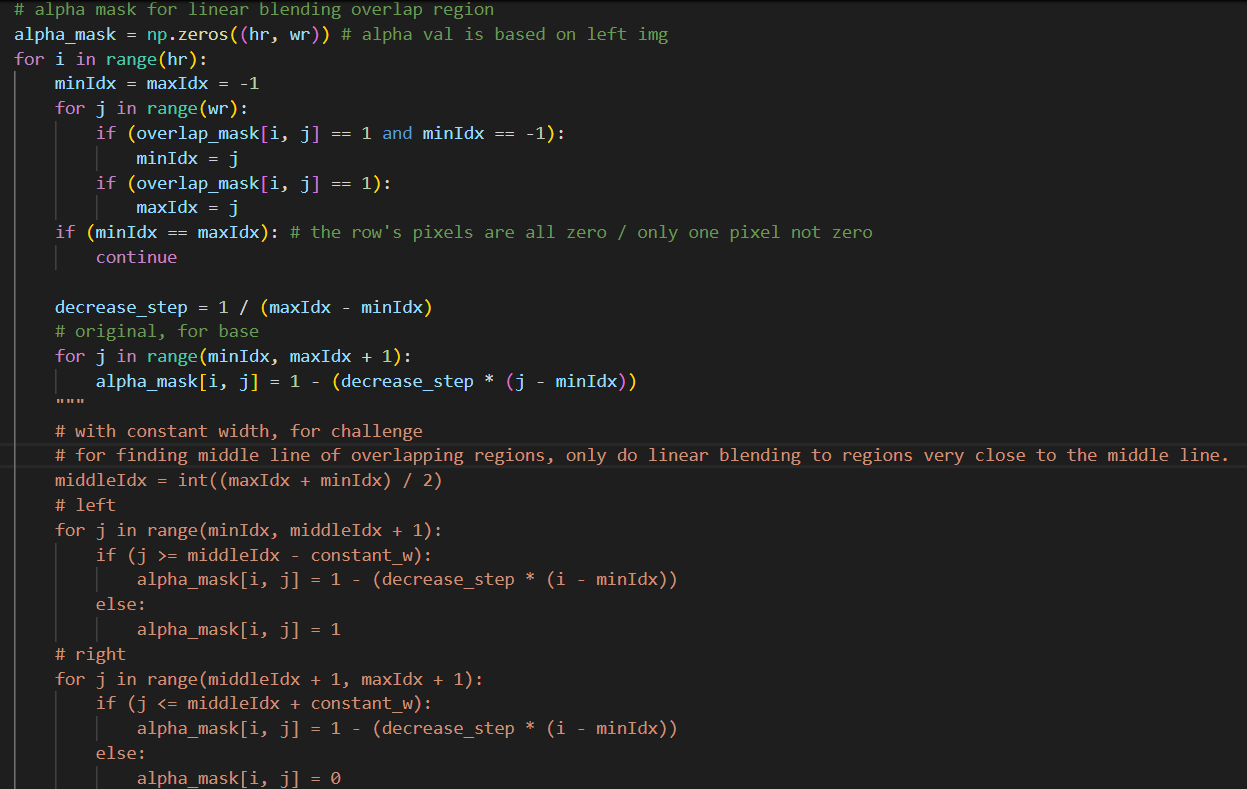
1. **Blending Image**

When stitching images, I apply linear blending on the images, which means I give each pixel different weights in the overlap region. The corresponding direction weight (left/right) will be larger if the pixel is closer to left/right. And this do remove some boundaries.

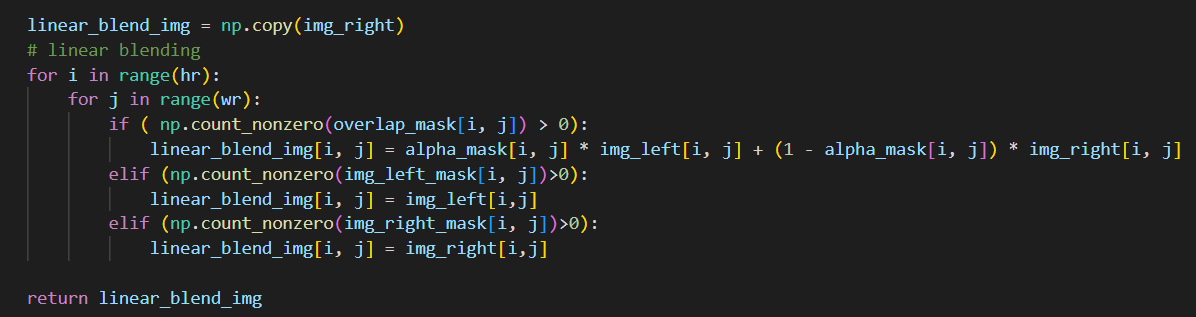
So in my implementation, I find left & right image mask region and their overlap mask first.



Then I get alpha mask for linear blending the overlap region, whose alpha value is based on left image. For “Challenge” task, since the original blending method somehow creates more doubled scenes, I check middle lines of overlapping regions and only do linear blending to regions that are close to the middle line (within a custom-constant width).

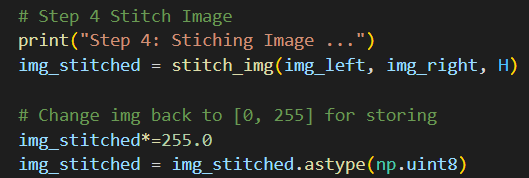


In the end, do linear blending through things I just get in this part.



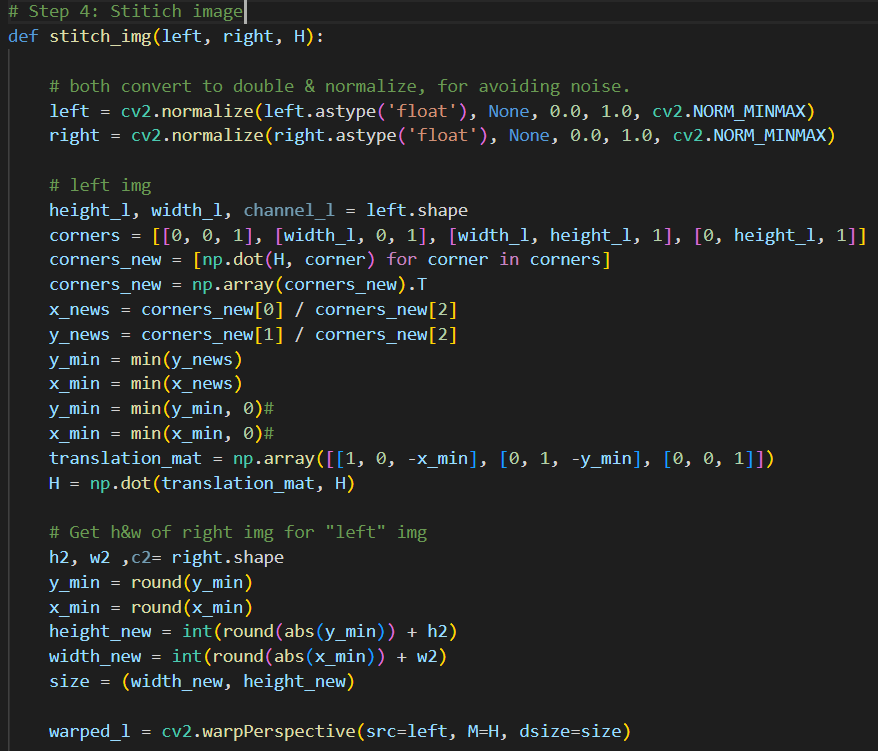
1. **Image Stitching**

In this part, I follow the slide’s instruction as well.

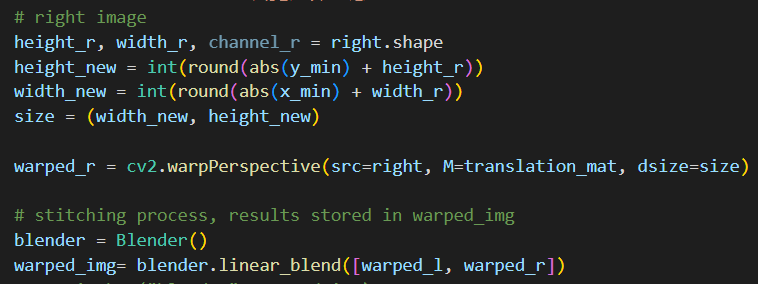


In the beginning, I convert left & right image into double then normalize them for latter calculation. Later, I try to get new corners, sizes, and affine matrix, then get new height and width and do matrix multiplication for those two images by cv2.warpPerspective( ).

For the left image, I want it to fit in with the right one, so I use the height and width of right image for calculating the left one’s new shape.



For the right image, I basically apply the same procedure but not use height & width of left image for calculation. In the end, apply blender method to the new-stitched image.



1. **Results :**

|  |
| --- |
| **Base** |
|  |
| **Base (blend with constant width)** |
|  |
| **Challenge** |
|  |
| **Challenge (blend with constant width)** |
|  |

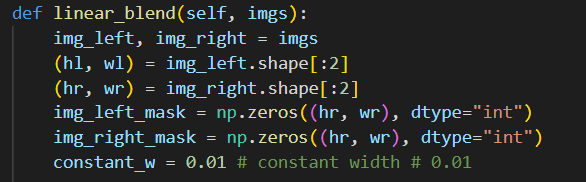
1. **Discussion(different blending method) :**

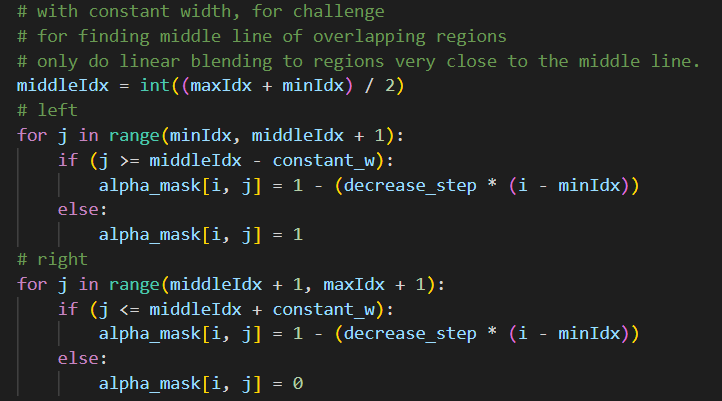
**Boundary Problem**

This is the first problem I encountered. At the beginning, I thought of the aliasing problem and tried to solve it with inverse mapping. But after trying blending, not only the problem seems to be solved, I also don’t have to do inverse mapping.

**Ghost Problem**

Ghost problem means that some scenes are doubled, due to the blending process, which is apparent in “Challenge” task. Here I try “linear blending with constant width” instead, which only blend constant width instead of all the overlapped parts. To implement this, I find the middle line of overlapping regions first, then do linear blending only to regions close to middle line(distance <= constant width).





This method does solve the ghost problem, but it makes the boundary more apparent, since it only blend certain width around the boundary.

|  |  |
| --- | --- |
| Original | With Constant Width |
|  |  |

**Light Problem**

For “Challenge”, the brightness of each image differs a lot, which could make the stitching process harder. So I try to make their brightness close to each other.

When reading the images, I also record their brightness then calculate the average brightness of them. In the stitching part, I assign the average brightness to each image part, which somehow makes them stitched better.

