1. Implementation:

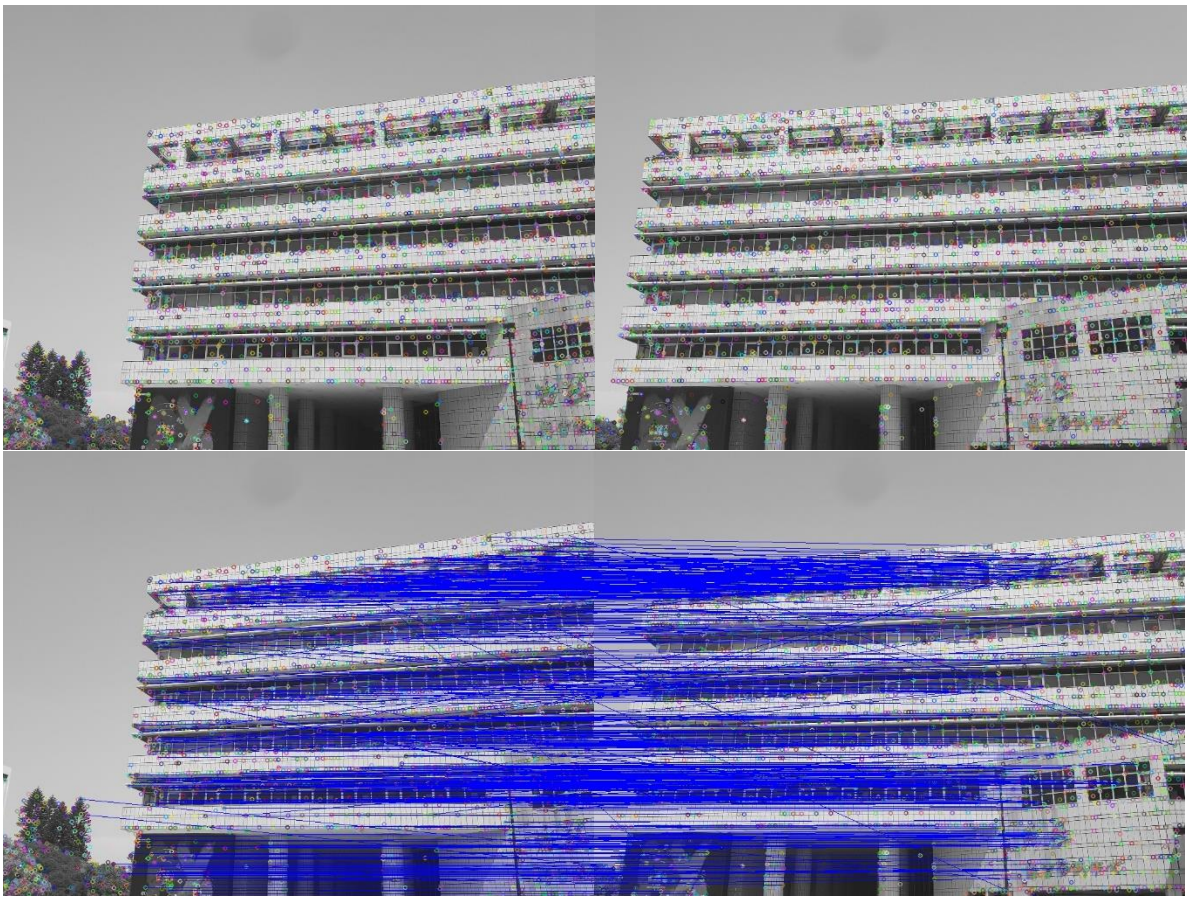
Image stitching algorithm can be divided by following 4 steps.

1. Detecting key point (feature) on the image
2. Finding feature correspondences (feature matching)
3. Computing homography matrix
4. Stitching image (warp images into same coordinate system)

For the first step of this algorithm is we need to find the key point for each image, we can use the SIFT algorithm (Scale-invariant feature transform) to find the feature, the idea is quiet simple, we use 4 * 4 * 8 dimension to describe one key point, and this step we can use the OpenCV function to do that.

And the second step is feature matching, feature matching first we use KNN (K-Nearest Neighbor) to find the k closet neighbors to the target, we the use the simple brute-force algorithm to do this, time complexity is $O(n^2)$ or use the advance method K-d tree, reduce time complexity to $O(nlogn)$, after the KNN then we need to do Lowe's Ratio test, since we may have bed match in the KNN, with the Lowe's ratio test, we can set a threshold to separate good match and bad match, a good match should be able to be distinguished from noise, after the testing we can eliminate the bad match. By the way OpenCV tutorial suggest that threshold should be set as 0.7 ~ 0.8.

The following figure show two image and they key point, and the draw two same point in a line.



The third step is computed homography matrix, the propose of this place is that we need to put the image in the same planar surface, and that is related by a homography. The detail is show at the following screenshot.

2. Homography Estimation

To estimate H, we start from the equation $\mathbf{x}_2 \sim H\mathbf{x}_1$. Written element by element, in homogenous coordinates we get the following constraint:

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \Leftrightarrow \mathbf{x}_2 = H\mathbf{x}_1 \tag{3}$$

In inhomogenous coordinates ($x_2' = x_2/z_2$ and $y_2' = y_2/z_2$),

$$x_2' = \frac{H_{11}x_1 + H_{12}y_1 + H_{13}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1} \tag{4}$$

$$y_2' = \frac{H_{21}x_1 + H_{22}y_1 + H_{23}z_1}{H_{31}x_1 + H_{32}y_1 + H_{33}z_1} \tag{5}$$

Without loss of generality, set z_1 = 1 and rearrange:

$$x_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{11}x_1 + H_{12}y_1 + H_{13} \tag{6}$$

$$y_2'(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{21}x_1 + H_{22}y_1 + H_{23} \tag{7}$$

We want to solve for H. Even though these inhomogeneous equations involve the coordinates nonlinearly, the coefficients of H appear linearly. Rearranging equations 6 and 7 we get,

$$\mathbf{a}_x^T\mathbf{h} = \mathbf{0} \tag{8}$$

$$\mathbf{a}_y^T\mathbf{h} = \mathbf{0} \tag{9}$$

where

$$\mathbf{h} = (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T \tag{10}$$

$$\mathbf{a}_x = (-x_1, -y_1, -1, 0, 0, 0, x_2'x_1, x_2'y_1, x_2')^T \tag{11}$$

$$\mathbf{a}_y = (0, 0, 0, -x_1, -y_1, -1, y_2'x_1, y_2'y_1, y_2')^T . \tag{12}$$

Given a set of corresponding points, we can form the following linear system of equations,

$$A\mathbf{h} = \mathbf{0} \tag{13}$$

where

$$A = \begin{pmatrix} \mathbf{a}_{x1}^T \\ \mathbf{a}_{y1}^T \\ \vdots \\ \mathbf{a}_{xN}^T \\ \mathbf{a}_{yN}^T \end{pmatrix} . \tag{14}$$

Equation 13 can be solved using homogeneous linear least squares, described in the next section.

After computed the homography matrix, then we use RANSAC algorithm to find the best matrix, The idea of RANSAC is also simple, we try a large number to iterate and each time we get 4 different key point, we find the matrix which have the maximum inlier and assume it is the BEST homography matrix.

The last step is stitching the two image, this part we can use the OpenCV function to do that, given the image and homography H then concatenate two image, but it maybe some problem on the result, for example for the two image overlap part it may have different color or ghost issues, so we need to add blending to solve this problem, blending can be implemented by different way, the simplest way is to do the linear way, for instance in the overlay part of two image, the position in pixel that have the large ratio to the nearest image, also the farer image have the smaller ratio, and two ratio should add to 1. That is the linear blending.

And the following are some experiences.

Exp1

Merge from 1 to 5

Merge from 6 to 10

Merge from 1 to 10 (have some error)

Exp2

Merge from 1 to 2

Merge from 3 to 4

Merge from 1 to 4

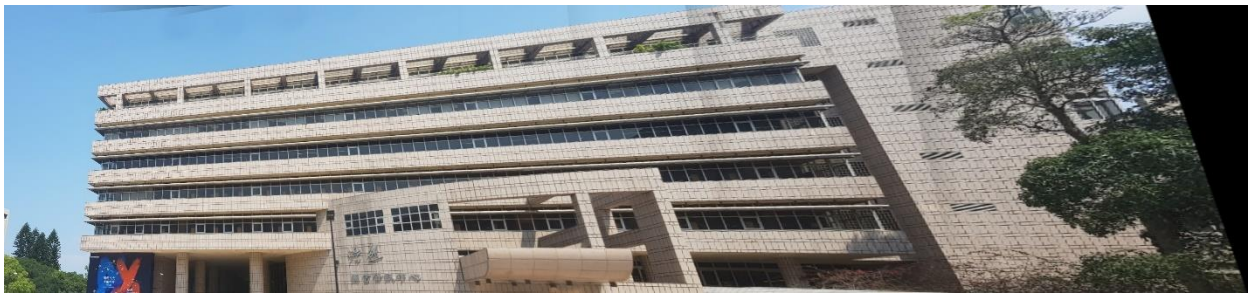
Exp3

Merge from 1 to 2

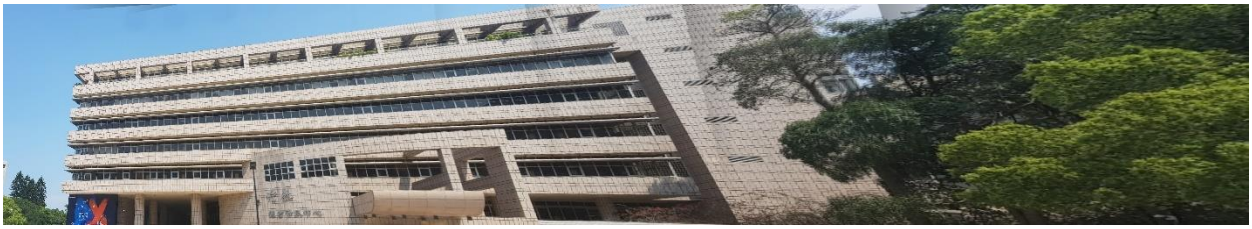
Merge from 1 to 3

Merge from 1 to 4

Merge from 1 to 5



Merge from 1 to 6



Merge from 1 to 7

Merge from 1 to 8

Merge from 1 to 9

Merge from 1 to 10

In the exp3 I think my algorithm is not better enough to merge too many images, and the maximum number seen like 5 or 6 images, and I didn't figure out what is the problem for my implementation.

And in those experiences I have the conclusion is that, the method of merge or we say the merge order will have difference result and different display performance, in exp1 and exp2 I use the way like divide and conquer, just like the merge sort to merge the whole image, we say from 1 to 10, we divide it to two image, the image merge from 1 to 5 and the image merge from 6 to 10 and so on. In the exp3 I just merge the image by the order step by step, that is first merge image 1 and image 2 to get image_12, and then merge with image 3 and so on ….