CS512 Assignment 2: report

Gady Agam
Illinois Institute of Technology

Abstract

The assignment is about Harris corner detection, corner localization, calculate feature of the corners and match the feature point between two images.

1. Problem statement

Corner detection is an important component of Computer Vision. The corners of an image usually related to some important or interest point of the image, and the set of these point can represent the image in a certain extent. So the corner detection is a base component of some higher and more complex Computer Vision algorithm.

In this assignment, after we detect corner sets in two images, we will try to match them. This operation can be the base to tell if two images are about a same object or same scene. This can be used to do image matching/searching or other interesting stuffs.

2. Proposed solution

At first we will use Harris corner detection algorithm to detect the corners in two images. Since the algorithm just tells "whether there's a corner inside an area", we have to suppress and localized the corners. Now we get the corners we need. For each corner we use SIFT to calculate its feature vector. By comparing the feature vectors between these two images, we can tell if there are some point matching between them.

The first and mainly algorithm in this assignment is Harris corner detection. Actually this algorithm is a merge of edge detection and corner detection, but we only use it to detect corner. The mainly idea of this algorithm is to look into the distribution of the gradients of points in a certain window. If the gradients are small, meaning that there's no corner or edge in it; If the gradients (roughly) have same orientation, meaning that there may be an edge in the window; If the gradients points to different orientation, there maybe a corner in it. Notice that a "corner" actually means "a point with the gradient of its neighbors has totally different orientation".

According to [1]:

$$E_{AC}(\Delta u)=\sum w(x_i)[I_0(x_i+\Delta u)-I_0(x_i)]^2=\Delta u^T A \Delta u$$

represent "how stable a point is respect to a small variation Δu". "w" is the window. Here:

$$A=w*\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

The larger uncertainty depends on the smaller eigenvalue of matrix A, so what a window has a big "smallest eigenvalue of matrix A" will has a large uncertainty, implying that there's a corner in it. So we want to find some window with big "smallest eigenvalue of matrix A".

But to do this, we have to find the eigenvalue, and it will cost large compute time. So Harris corner detection gives a simpler equation:

$$det(A) - \alpha\, trace(A)^2 = \lambda_0 \lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

With α=0.06. When this equation's absolute result is small, there's nothing in the window; negative means an edge; positive means a corner. So if the result of this equation is larger than a threshold, we will think there's a corner in the window.

The second part is to suppress and localized the corner.

Since the window in the Harris corner detection can be partly overlap with each other, one real corner can cause several corner detected. By finding a local maximum in a window and delete all other detected corner in this window, it will suppress the “multiple detect” problem in a certain extent.

Since Harris corner detection can only tell if a window has a corner in it, we want to know the exactly location of the corner in the window. The algorithm is: find a point P in the window, which minimize a number X. X equals {the sum(for all edge points) of square of ((gradient of edge point E) dot product (the vector that equals (P-E))}.

Now we have find some nice corners. The next step is to calculate its feature and match them. What I use is SIFT. According to [1], it will separate a window into 16 blocks(not overlap with each other). For each block, it will normalize and add the weighted(the further from the center of the window, the smaller the weight is) gradient to 8 values which representing 8 orientations. This 8*16=128 values will be the feature vector (or histogram) of the point. To make the feature rotation-invariance, SIFT will rotate the histogram (maybe according to the max/min value or distribution of this 128 values). So, SIFT will calculate the feature of each corners.

Finally, we need to match the points in two images. I use a naïve algorithm: for each corner in image A, find the corner in image B which has the minimum Euclid-distance relate to it. Of course if the Euclid-distance is too large, consider that there's no matching. If two corners match to a same corner P, P will choose the one with smaller Euclid-distance to it as its match result.

There's a better algorithm which I didn't use because of its complexity: for each corner in image A find several candidate corner in image B. If P1 is a candidate of P2 and P2 is also a candidate of P1, they are matched. If conflict happens, find a way to optimize the result (it can be difficult to do this).

3. Implementation details

The program use the same support code as assignment 1, so there's no design issues.

Instead using Sobel in Harris corner detection, I implement GaussianBlur on the image first and use [-1,0,1; -1,0,1; -1,0,1] and [-1,-1,-1; 0,0,0; 1,1,1] to get x and y derivative. So my result is a little different from the cornerHarris() implemented by OpenCV. When I try to use the same Sobel filter in my implement, the result is same with cornerHarris(). So I consider my implement is correct. The final code doesn't include this part because I only know what Sobel filter is with size=3.

One implement choice is: when doing the minimum dot product to localized the corner, if we should only count edge point or count all points in the window. My choice is calling canny() first to detect the edge and only count the edge point. It will get a better localization in theory.

One problem I faced is: the SIFT feature extractor(SiftDescriptorExtractor) in OpenCV

doesn't give the rotation-invariance feature vector. The document of it is not that clear, so I spent a lot of time and finally found that I have to tell it the angle of the corner which it use to "rotate the vector". So I give it the gradient of the corner and the problem solved.

This is not an implementation problem but it cost me a lot of time so I still want to mention it. When I test the program using a chessboard image, I found that when I make the neighbor size (the window size in Harris corner detection) larger than a certain value, the corner is not localized correctly. At first I think it's a bug. But in fact it's not. When the neighbor size is larger than the grid of the chessboard, it will cover several corners. Of course it can't localize correctly.

When a match pair is found, the program will number them with identical numbers and draw a line between them.

To run the program, you need to give two parameters as image paths. For example: "./program a.jpg b.jpg".

The program has 5 track-bars.

1) Weight of Trace (/100): This parameter controls α in the Harris corner detection equation which was described before. The program initialize it to 4, means α=0.04, you can control it from 0 to 50 (0.5).
2) Match Threshold(*10): When doing the match, we calculate the Euclid-distance of two vectors. Empirically, when point A is the one has minimum Euclid-distance to point B and the value of the Euclid-distance is below 300-500, point A may be match to point B. So we initialize the match threshold to 35 (350), meaning if we find two points that has minimum Euclid-distance but the value is larger than 350, we don't count them as matched points. You can control it from 0 to 100 (1000).
3) Neighbor size: The window size of Harris corner detection. Actually I also use it to suppress corner points and localization. It's initialized to 5, and you can control it from 2 to 50.
4) Corner Threshold: The difference of equation value in Harris corner detection can be very large. So I use normalize() function in OpecCV to normalize it to [0,255]. The normalize is base on NORM_MINMAX. If an equation value is negative, I will make it to 0, or the value 0 will be normalized to about 120. It's initialized to 100, and you can control the corner threshold from 0 to 255.
5) Gaussian kernel size. The program will reduce the image noise before doing other steps by implement GaussianBlur to the image. The sigma will be valued as (Gaussian kernel size/5.0). It's initialized to 3, and you can control it from 0 to 31.

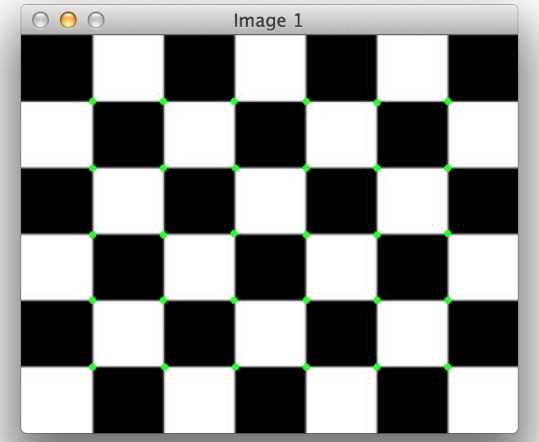
4. Results and discussion

First, here gives a comparision of my implement of Harris corner detection with cornerharris(). Notice that the release vison of the program does not have this function, because I only know that Sobel filter can have size=3,5,7, but cornerharris() can take any size as parameter.

(All the parameters are the initialized value in the below two picture).

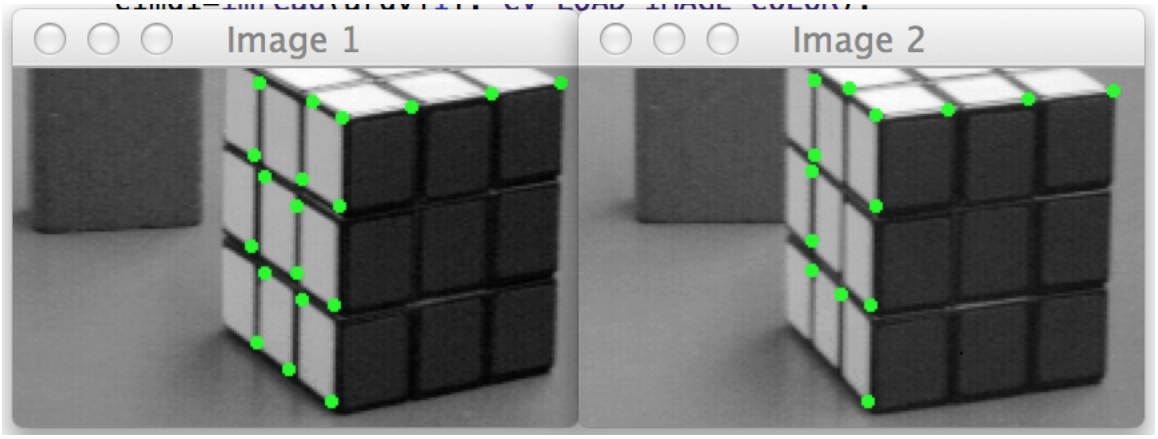
My version **OpenCV version**

This result is not suppressed and localized. Here's the result after these two operation. And I also gives a result on the chessboard to certify the result is correct. (All the parameters are the initialized value in the below two picture).

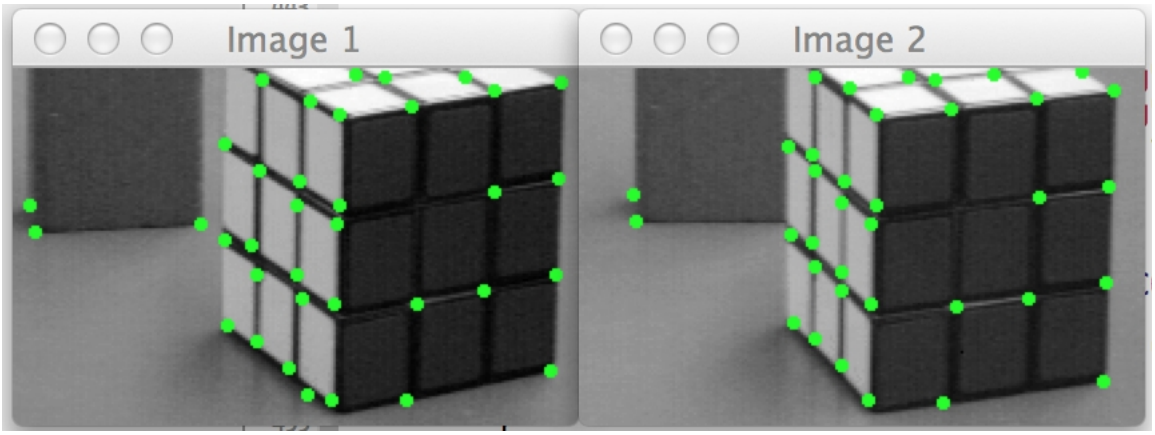


After suppress and localization **Result on chessboard**

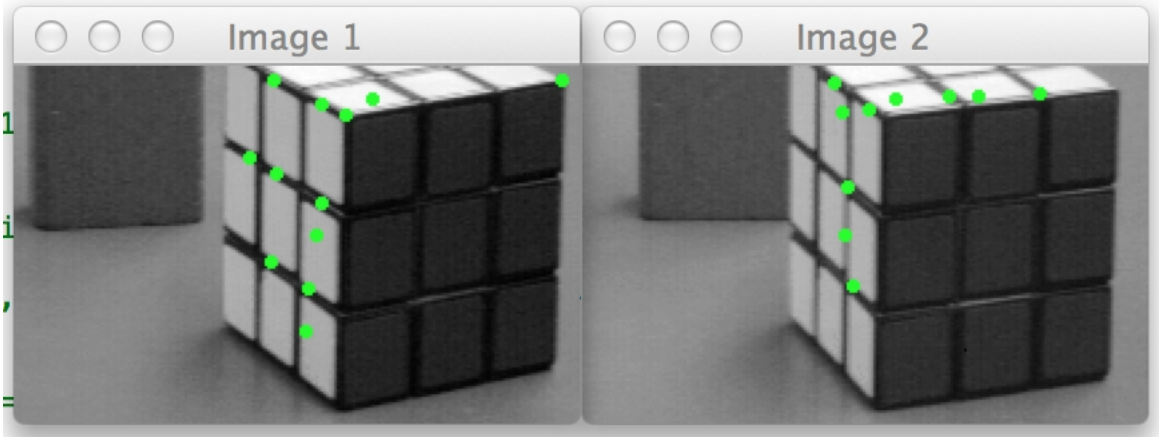
Now we gives the result in different parameters, the two images are just a stereo pair:



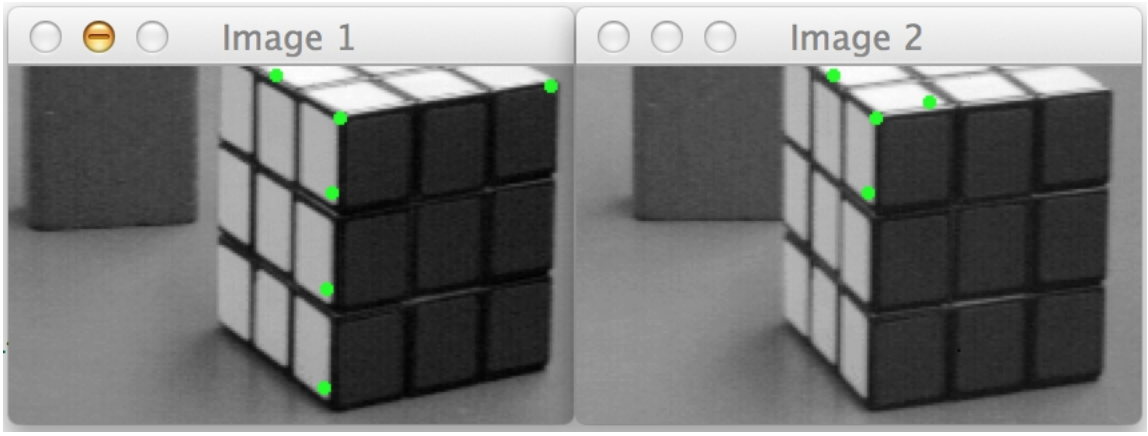
A: Corner threshold=100, neighbor size=5, Gaussian kernel size=3, weight of trace=0.04



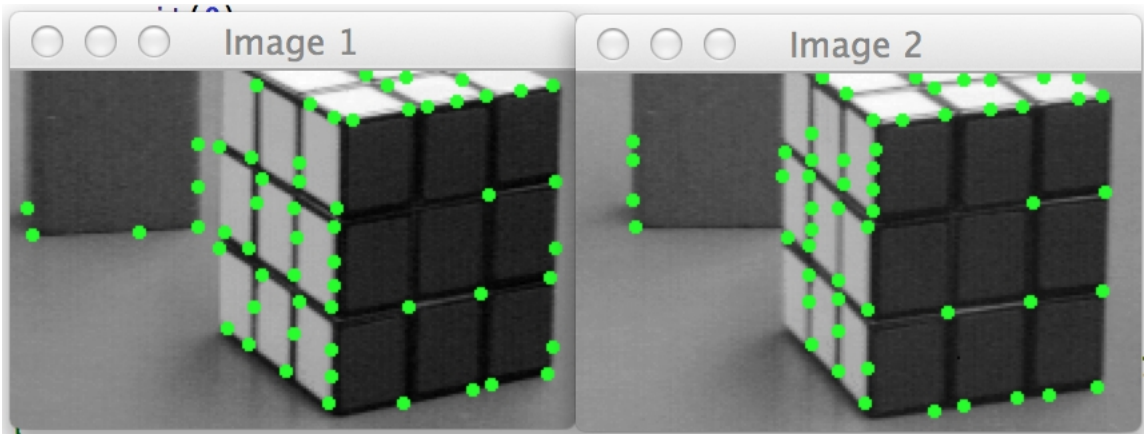
B: Corner threshold=0, neighbor size=5, Gaussian kernel size=3, weight of trace=0.04



C: Corner threshold=100, neighbor size=30, Gaussian kernel size=3, weight of trace=0.04



D: Corner threshold=100, neighbor size=5, Gaussian kernel size=18, weight of trace=0.04



E: Corner threshold=0, neighbor size=5, Gaussian kernel size=3, weight of trace=0

Here we can see:

a) From A and B, we get that the smaller threshold gives more corners. Since the program delete all points that has negative value, and the magnitude of the positive values can be very different, normalize the value to [0,255] means that a point with value 1 also can be a corner (the threshold=0 means that the value must >0, namely that at least 1)

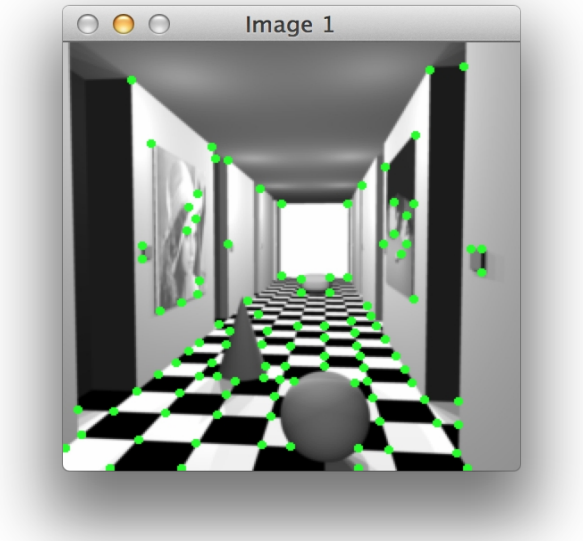
b) From A and C, we find that C does not find the correct corners. The reason of it is that the neighbor size is too big, causing that one window contains several exact corners. This will invalidate the localization algorithm we used. If the window is too small, some corner may not be detected because the corner may not sharp enough that it can't be cover in a single window. To reduce the figures I didn't give the result of small window, but the result correspond to my inference.

c) From A and D, we find that big Gaussian kernel size can reduce the number of detected corner and decrease the precision of localization. This is obviously, because a smoothing operation will blur the image and make the corner less sharp, which make it more difficult to detect and localize the corners.

d) E is to show the affect of “weight of trace”. The reason why we make “Corner threshold=0” is that there's not enough number of corners to make the comparison clearly enough if we make “Corner threshold=100”. By comparing E to B, we can find that “weight of trace=0.04” reduce some fault negative result which are in fact edge points. The reason is that it will make the Harris equation value of edge points smaller and lower than the threshold. The conclusion is that: a suitable “weight of trace” will makes the corner

detection more precise.
(it's not necessary to show the affect of “matching threshold”, because it will only reduce the number of matching, not even change the matching pair)

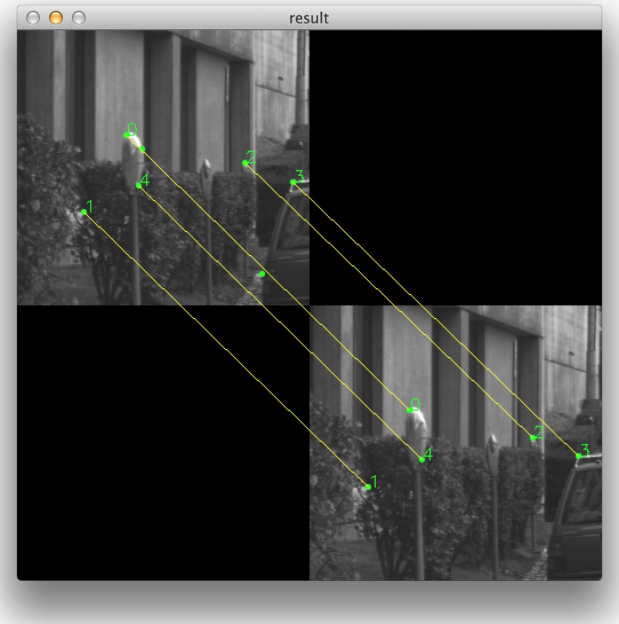
Since I haven't show how well the corner detection result can be if we control the parameter to get an optimal result, I show it here. Here's the corner detect result in a perspective picture:



Corner threshold=0, neighbor size=2, Gaussian kernel size=0, all other parameters are the initialized value (neighbor size=2 can cause this nice result only because the image is sharp enough)

We can find that: Because of the large difference of the value of the Harris corner detection equation and the normalize function we used, a point is usually a corner if its value is larger than 0 after normalization. Although we use the smallest “corner threshold”, “neighbor size” and “gaussian kernel size”, it still can not detect the corner of distant floor grid. But it's reasonable because that grids already can be treat as “lines”, because Harris corner detection consider two inverse-orientation gradients has same orientation.

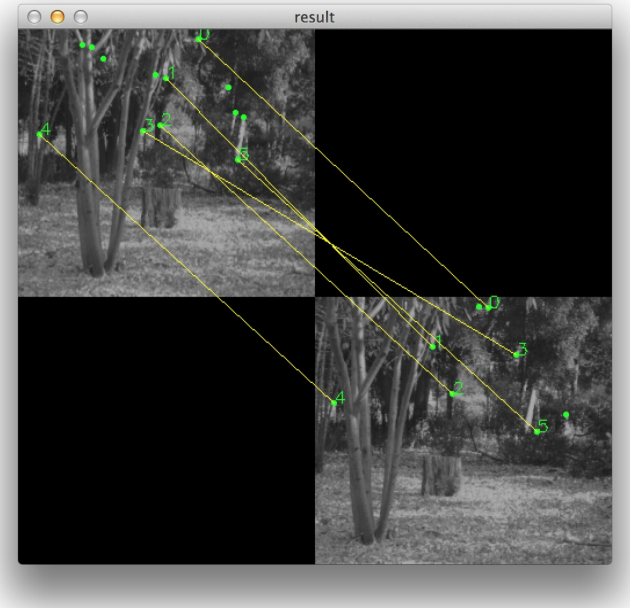
Now we show the matching result:



All parameter is the initialized value (see the describe above. Matching threshold=350)

We can see that although two pictures has different number of corners, the matching is still nice. The matching is nice because the corners are very different in the pictures.

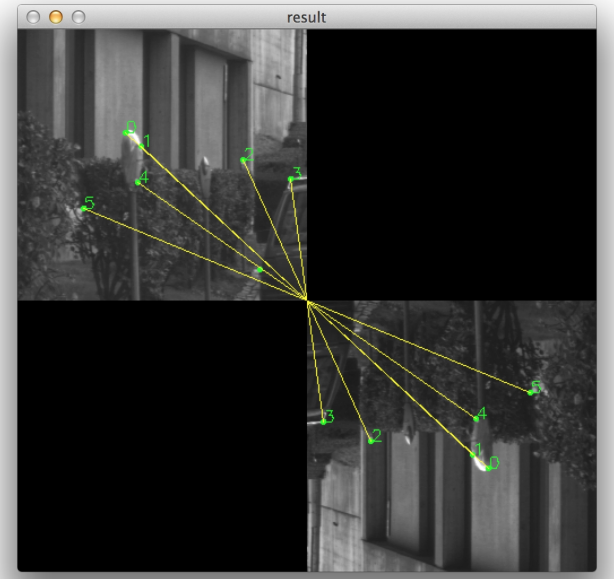
Now we gives a stereo pair that the corners in it is not that different:



Corner threshold=85, Match threshold=330, other parameter is the initialized value

We see that: for a picture that the corners in it is not that different, it's harder to match the corners. But if we choose the parameters right, the result will not be very wrong. Actually, if we make the “match threshold” to be 310, the only wrong in this pairs (match 3) will be discarded.

What if a picture is rotated?



All parameters are the initialized value (see the describe above. Matching threshold=350)

We see that the match is perfect. The SIFT does well in rotation invariance.

Summary: The Harris corner detection can detect most of the corners. But you have to choose the weight of trace carefully to let the algorithm distinguish between edge points and corners. One corner will result to several corners to be detected. If so, it's hard for suppression algorithm to perfectly suppress the result because it may suppress a nearby point which belong to a different corner. If the suppression is not good, the localization will be affected as well. These all related to the problem of how to choose the window size, which can be conflictive. The localization algorithm only works well if a window has exactly one corner in it. Using only the edge points in the localization process will help increase the precision.

Since the program has some "for loop" and sort operation which can't be done through matrix operation, the running speed of the program is not that fast: takes about 600-650ms to do the Harris corner detection+suppress+localization on a 1024*1024 image with all parameters to be the initialized value. The matching part(not include the feature calculating) is not that slow: 800 match to 800 points takes less than 100ms. The processor is 2.3GHz Inter Core i7, with 16GB memory and SSD, OS is OS X 10.9.1.

5. References

[1] Szeliski Book, http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf