Dendrite graph penalized matching algorithm

In this project, we want to design a light-weight graph matching image identification/ authentication algorithm. The code is developed by Ali and Zaoyi. In the following section, I will describe the usage of the most of function file and the folders. The most of functions do have the comments in the very beginning. I hope it could help anyone who wants to understand the code. :>

The tutorial and the description of the current repository

I will describe the usage of folders. Some folders are duplicated with minor changes, I may use them for different test before. I would suggest to include all files within the sub-folders in MATLAB before run any of codes.

`./faster-code`: In this folder, I developed a newer version of graph penalized algorithm. In this version, the searching and matching algortihm is at least 10 times faster than the older version. The functions are named as `$Functions_Fast(er).m`, which means we also have the older version of the same function named as `$Functions.m`

`./faster-code/Distance_Fast.m`: This function computes the distance score between two nodes from `test` and `ref` separately.

`./faster-code/mappingAndLink_Faster.m`: The function map the linkage between two trees `test` and `ref`.

`./faster-code/mappingTest_Fast.m`: This function calls the previous two functions and can be used directly. The example of how to use it, you can see `./scratch_FinalTest.m`

`./artificial_colored:` The self-generated denderite image samples with color. All the functions inside this folder are used to do the test describe in the paper, e.g., SNR test and compressed image estimate test. You can use the samples in this folder.

`./artificial_colored/step-by-step`: This folder is unnecessary, I create it for the image preprocessing step and only just for showing every step processed image.

`./high-resolution:` Not required.

`./pp-image`: Not required, for generating the thesis figure purpose.

`./PUF_DATA_COLORFLIPED:` The function and the images that do flip the original white and black background.

`./results`: The tested results figure sources and figures.

`./results_skewness:` This folder does the test on the skew images (also included in this folder for different angles). You can see demo for file `skew_feature_map.m`

`./results-mat/`: a very early version of 50 samples matching results in `.mat` file

`./results-mat-2/`: the test results obtained by Monsoon for different noise level results.

`./test-sample-og/`: original low resolution samples

`./tex-reports/`: My latex report

`./algfixTesting.m`: The function I used to debug, this function can visualze the matching by horizontal lines

`./build_Database.m`: Build a datebase to store all the trees

`./buildTreeStruct.m`: Create the tree structure based on the extracted input information

`./calc_homography.m`: The function used to compute the homography score between two points set

`./checkClosePoints.m`: The function that called by random dendrite generation to check if we are generating a loop.

`./checkFutureOverlap.m`: Does the similar function as the previous one to check the overlap when generating the points.

`./colorextract_(artificial).m`: a start function that extract the skleton from the raw input image. **artificial* is for self-generated image.

`./consistencyScore.m`: a part of the penalized algorithm that computes the consistency score between two nodes

`./DendriteDicGen.m`: Create and save the image to a desgined `.mat` file

`./dendriteModelGenrator.m`: The main function of random denderite model generation, the input is wanted number of branch. The typical output is less than the number branch you input because of the initial start chance.

`./denoiseHandle.m`: the function that denoise the image based on different noise scenarios. such as guassian, salt & pepper and motion.

`./DistanceScore.m`: compute the distance between two nodes

`./fasle_magic.m`: give the random generated dendrite sample a similar color scheme as the orginal sample

`./featurepoint_(n)test.m`: One of the demo function that matching the feature points by different methods. Such as SIFT, Harris, etc. In the end, that would be our proposed method. *(n)* represents the noise test

`./findInitialDots.m`: the sub-function that calll by `graph_based_rdGen.m` to find out the initial start points from the root.

`./findmiddlecircle2.m`: A depreacted function that was designed to find out the location of the middle root, the current method uses `./hough_circle.m`

`(general)_graph_based_rdGen.m`: the main feature extraction function of our proposed method as well as processing the data into a desired format.

`./get_children.m` : find the children of a node when we do penalized matching

`./hungarian.m`: the underground source code to do quick matching in $O(n^3)$

`linkchildren.m` and `linksibling.m` : The method that does the function to link the children and siblings for the matching algorithm

`mappingMain.m` : The overall main function that maps two data storages.

`matchedNodeDrawLine.m`: Draw line between the two images for matched feature points.

`munkres.m`: older version of hugarian, deprecated.

`scratch_FinalTest.m`: A demo to do scratch test

`quickScripts_(sh).m`: A script to generate amount of sh code and source code for running Monsoon.

`subTreeGen.m` : Called by `./dendriteModelGenrator.m` to generate the subtree.

`randNode_Type.m` and `randPath_Choose.m` : Called by `./dendriteModelGenrator.m` to select the next node type and path direction.