CS696, Applied Computer Vision

Homework Assignment 4

Instructor: Xiaobai Liu

1. **How to submit**

* Submit your source codes and writeup through SDSU Blackboard, before 11:59pm, April 17, 2020

1. **Overview**

The goal of this assignment is to create a local feature matching algorithm using techniques described in Szeliski chapter 4.1. The pipeline we suggest is a simplified version of the famous [SIFT](http://www.cs.ubc.ca/~lowe/keypoints/) pipeline. The matching pipeline is intended to work for instance-level matching -- multiple views of the same physical scene.



Figure.1 Two images of Notre Dame.

1. **Details**

For this project, you need to implement the three major steps of a local feature matching algorithm:

* Interest point detection in get\_interest\_points.m (see Szeliski 4.1.1)
* Local feature description in get\_features.m (see Szeliski 4.1.2)
* Feature Matching in match\_features.m (see Szeliski 4.1.3)

There are numerous papers in the computer vision literature addressing each stage. For this project, we will suggest specific, relatively simple algorithms for each stage. You are encouraged to experiment with more sophisticated algorithms!

**2.1 Interest point detection (get\_interest\_points.m )**

You will implement the Harris corner detector as described in the lecture materials and Szeliski 4.1.1. See Algorithm 4.1 in the textbook for pseudocode. The starter code gives some additional suggestions. You do not need to worry about scale invariance or keypoint orientation estimation for your baseline Harris corner detector.

*Options*. Try detecting keypoints at multiple scales or using a scale selection method to pick the best scale. You can also try the adaptive non-maximum suppression. Finally, you can try an entirely different interest point detection strategy like that of MSER. If you implement an additional interest point detector, you can use it alone or you can take the union of keypoints detected by multiple methods.

**2.2 Local feature description (get\_features.m)**

In this function, you will need to extract the histogram of oriented gradients (HOG). You might follow the design of SIFT-like features. Normalize the histogram to be unit 1.

See the placeholder get\_features.m for more details. If you want to get your matching pipeline working quickly (and maybe to help debug the other algorithm stages), you might want to start with (template based) normalized intensities, histogram of intensity, or histogram of RGB, as your local feature descriptor.

*Options.* The simplest thing to do is to experiment with the numerous histogram parameters: how big should each feature patch have (16 pixels by 16 pixels , 32 pixels by 32 pixels etc. ) ? How many cells should each feature descriptor have (2 by 2 cells, 4 by 4 cells etc.)? How many orientations should each histogram have (8 bins, 16 bins etc.)? How well the orientation normalization work against rotations? Don't get lost: You are only required to implement the basic version of HOG. Extra works to answer the above questions will be encouraged with extra credits.

**2.3 Feature matching (match\_features.m)**

You will implement the Nearest Neighbor method of matching local features across two images as described in Szeliski 4.1.3. Given two set of features descriptors, your goal is to find the matched feature pairs across the two sets. To do so, for each descriptor, simply compute its similarities to all the descriptors in the other set. Save the pair of matched points and their similarities (as matching confidences) . See the matlab file for more instructions.

*Tips* . Simply applying the above procedure might lead to many-to-one or a one-to-many correspondence, i.e. an interest point in the first image is matched to two or more interest points in the second image. In the starter code, this wouldn’t be a problem, because the codes will always pick up the most confident correspondences. You are encouraged to apply the non-maximal suppression (NMS) strategy to prune the correspondences.

**2.4 Using the starter code (proj4.m)**

*You DON’T NEED to modify the scripts mentioned in this subsection.*

The top-level proj4.m script provided in the starter code includes file handling, visualization, and evaluation functions for you as well as calls to placeholder versions of the three functions listed above. Running the starter code without modification will visualize random interest points matched randomly on the particular Notre Dame images shown in Figure 1. For these two images there is a ground truth evaluation in the starter code, as well. The script evaluate\_correspondence.m will classify each match as correct or incorrect based on similarity to hand-provided matches ( run show\_ground\_truth\_corr.m  to see the ground truth annotations). The starter code only includes ground truth for this image pair, but you can create additional ground truth matches with collect\_ground\_truth\_corr.m, although it is not required.

As you implement your feature matching pipeline, you should see your performance according to evaluate\_correspondence.m increase. Note that a correspondence is bad if there's no ground truth point within 150 pixels or if nearest ground truth correspondence offset isn't within 25 pixels of the estimated correspondence offset. See the script for more details.

1. **Resources**

**Potentially useful MATLAB functions**: imfilter(), fspecial(), bwconncomp(), colfilt(), sort().

**Forbidden functions** you can use for testing, but not in your final code:

extractFeatures(), detectSURFFeatures().

1. **Optional Programming**

In addition to the specific suggestions above for each stage of the pipeline, you can try running a Structure from Motion algorithm based on the feature matches you find. For example, try configuring the widely used **[VisualSFM package](http://homes.cs.washington.edu/~ccwu/vsfm/)** or [**Bundler**](http://www.cs.cornell.edu/~snavely/bundler/) to accept the feature matches you find rather than their default (SIFT-based) matches. If you find many matches between many photos you can achieve a sparse 3d reconstruction of the physical scene. You can then use a stereo algorithm to generate a dense reconstruction. You'll be dealing with complicated software packages.

1. **Writeup**

In the report you will describe your algorithm and any decisions you made to write your algorithm a particular way. Then you will show and discuss the results of your algorithm.

In the case of this project, show how well you’re matching method works on both the Notre Dame image pair and ONE additional pair of images captured on your own. For the additional case, you are not required to provide groundtruth nor numberric results (i.e. no need to run correspondence.m). You are encouraged to do so, though.

For the Notre Dame images, you can show eval.jpg which the starter code generates.

For other image pairs, there is no ground truth evaluation (you can make it!) so you can show vis.jpg instead.

A good writeup will assess how important various design decisions were. E.g. by using SIFT-like features instead of normalized patches, I went from 70% good matches to 90% good matches.

You should clearly demonstrate how your additions changed the behavior on particular test cases.