I400/B457 Intro to Computer Vision - Programming assignment #5

This assignment heavily relies on lecture7_1_localfeats.ppt . We will use the opencv library for the first time. This is an important package.

Problem 0: OpenCV installation

1. Follow Manish's instructions to install opencv for python.

Problem 1: SIFT local feature extraction and matching [70%]

0. Download the images from Canvas: "box.png" and "box_in_scene.png".

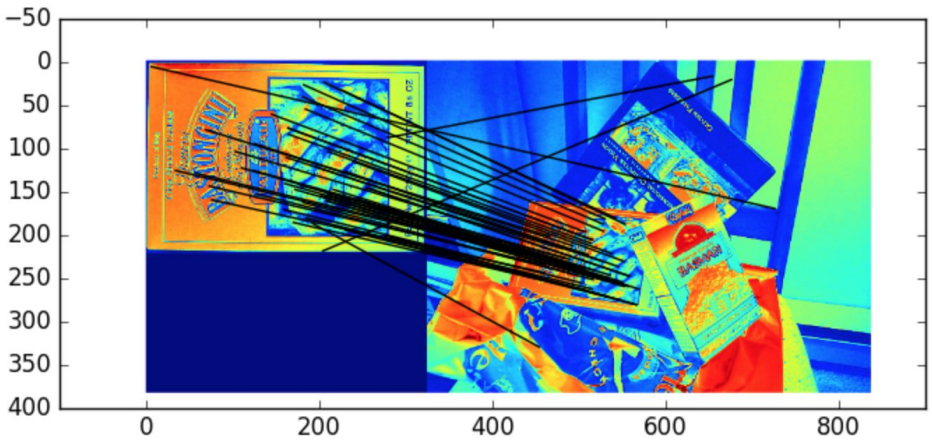
1. Extract SIFT features from the images using OpenCV. This URL will help you: http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html#gsc.tab=0 . Python textbook pages 36-44 (pdf version pages 55-62) will also help. Save the results as "box_sift.jpg" and "box_in_scene_sift.jpg".

2. Implement a function to measure "Euclidean distance" between two features (i.e., L2 norm). If we have two SIFT descriptors f1 and f2, and if they are n-dimensional vector, it can be defined as: `sqrt( (f1[0]-f2[0])^2 + (f1[1]-f2[1])^2 + … + (f1[n]-f2[n])^2 ).`

3. Use the implemented function to measure the pairwise distance between each feature from box.png and each feature from box_in_scene.png. Create a m1-by-m2 array, where m1 is the number of features in box.png and m2 is the number of features in box_in_scene.png. This is called 'distance' matrix or array.

4. Based on the distance matrix, for each feature in box.png, find the corresponding feature in box_in_scene.png. Visualize the matching results with lines, similar to pages 42-44 of the python textbook (pdf version pages 60-62). Only display 50 best matches (i.e., 50 features with smallest distances). Save the matching results as "matching.jpg".

It should (somewhat) look like this:



5. Use BFMatcher of OpenCV to do the same matching: http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html . Save the matching results as “matching2.jpg”.

Problem 2: SIFT matching spatial verification [30%]

Note: if you are confused about this problem, we will have a chance to talk about this in the Tuesday’s class next week as well, so please attend it.

0. Use the Euclidean distance SIFT feature matching from above.

1. Use the OpenCV’s `cv2.findHomography` function to find the homography between SIFT features in one image (box.png) to SIFT features in the other image (box_in_scene.png) using RANSAC.
http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findhomography#findhomography
http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html

Check slides 40-45 of lecture8_1_indexing.pptx. The `cv2.findHomography` function will not only give you the homography matrix, but also let you know which matches are inliners (i.e., spatially consistent) and which are not. For example, if the code is `M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)` , the returned array `mask` lets you know which matches are good. Please be careful with the format of `src_pts` and `dst_pts`.

2. Visualize the bounding box of the object in box_in_scene.png using the homography matrix. Check the 2nd URL of the step 1. Also visualize the valid SIFT matches yourself. Save the result as “box_match_RANSAC.jpg”.