CS 378 / CS 395T Computer Vision
Fall 2007
Problem set 3
Due Thursday 11/13

Submit both as hardcopy and electronically through the turnin program. If you have multiple files, tar them together, and use your first initial and last name to name the tar file. Then submit with:

turnin --submit svnaras pset3 <yourfilename>

In your hardcopy, include the code as well as any relevant figures.

Useful matlab functions: 'impixelinfo', 'imagesc', 'imfilter'.
Provided functions: 'display_features.m', 'example_file_reading.m', 'dist2.m'

1. Stereo matching

In order to compute a 3-D reconstruction, a stereo system must first solve the correspondence problem. Matching algorithms for stereo can produce *sparse* or *dense* disparity maps, depending on exactly what features are used as a basis for the correspondence. In dense stereo matching, the disparity is estimated for every pixel; the elements matched are image windows of a fixed size. In contrast, for sparse stereo matching, the disparity is computed at a subset of "interesting" points in the image; the search for correspondence is restricted to a sparse set of detected features. (See Trucco and Verri, Section 7.2)

For this exercise, you will implement a dense and sparse stereo matching algorithm. The basic inputs in both cases will be a left and right stereo image pair, and the output will be an image (matrix) of the computed disparity estimates, as well as a quantitative measure of the error in the estimates.

Download the data from the class webpage. There are four rectified stereo image pairs (named 'poster', 'venus', 'tsukuba', and 'sawtooth'); the data for each is in its own directory. This data set is from the Middlebury Stereo project (http://vision.middlebury.edu/stereo/data/). Important: because the images are rectified, we know that epipolar lines correspond to scanlines. For each stereo pair, there is also a ground truth image giving the true disparity. This is what you will use to quantify how good your method's disparity estimates are.

We have also provided an example script ('example_file_reading.m') to demonstrate how to read and display the data. It in turn calls the function 'display_features.m' to display sparse features detected for part (b) below. Before you write any code, open example_file_reading.m, set the path ('basepath') to the correct place in your home directory, and run this script. You should see the stereo pairs displayed one after another with ellipses plotted on top of them, and the associated disparity maps. This file also contains a few comments about the data provided (dimensions, etc.) For all your matching computation, use grayscale left/right images, and cast to doubles before processing.

If you are interested in reading more about how the Middlebury benchmark data was generated, see this paper: D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1/2/3):7-42, April-June 2002.

a) dense stereo matching (40 points)

For this problem you will need the following files in each of the four directories:

- disparity.pgm
- leftim.ppm
- rightim.ppm

To measure how similar two fixed-sized image subwindows are, we need a cost function. Then for every window along a scanline in the left image, we can identify the best matching window along the same scanline in the right image. (Remember, we are dealing with rectified pairs here). The disparity at each point is the distance between the matching windows' centers (in the *x*-coordinate).

The SSD between two image windows is the sum of the squared intensity differences within them. For this problem, you will write a dense stereo matching algorithm using SSD as the cost function. This can be done efficiently via convolution.

First write a function called 'sqdiff_images' that computes a set of squared-difference images. Given the left and right images and a list of possible disparities, it should return a stack of squared-difference images, one for each disparity (shift). For each disparity amount d_i, shift the entire left image by d_i, and compute the squared intensity difference relative to the right image for each pixel. The result should be n images of the form: $S(x, y, d_i) = [I_{left}(x + d_i, y) - I_{right}(x, y)]^2$, for $i = 1, \ldots n$. When you call this function, select a range of disparities that is roughly on par with the amount of displacement you see in the image pairs.

Next write a function called 'windowed_ssd' that convolves each of the squared difference images computed above with an *N* x *N* box filter. This will efficiently yield the SSD within windows of size *N* (why?). The result is of the form: $SSD(x, y, d_i) = S(x, y, d_i) * F(x, y)$, for the box filter F centered at (x, y). The size of the filter *N* should be a parameter to the function. You can use 'imfilter' to do the convolution.

Finally, write code to determine the disparity associated with the best matching window for each point in the left image, i.e., the disparity at which the SSD is minimized.

At the boundaries the window may fall partially outside of the image, so you can limit your output disparity estimates to the region for which windows are always complete. That is, only process (x, y) positions which do not involve boundary problems. You should have disparity estimates at each pixel except for these bordering sections.

Quantitatively measure the error of your disparity estimates relative to the ground truth. Display your estimate and the ground truth with 'imagesc'.

Note: the disparities in the image files are scaled for display purposes. For the Tsukuba pair they are scaled by 16; for the rest, the scale factor is 8. See the 'disparityvals' matrix in 'example_file_reading.m'.

After processing all of the stereo pairs, write answers to each of the following:

i) Explain why the squared differencing followed by the box filter yields the match scores we want.

ii) For a decent selection of *N*, where do most errors in your disparity estimates occur? Include figures of the disparity maps your code generates for each of the four cases.

iii) What impact does the window size have on the estimated disparities? Try a few values, check the average quantitative error, and display the disparity maps. Explain how and why things differ for different window sizes.

iv) What is the computational complexity of computing the dense matches, in terms of *N*, *n*, and the size of the image? What is the computational savings, if any, over a naïve implementation that does not use convolution?

b) sparse stereo matching (30 points)

For this problem you will need the following files in each directory:

- disparity.pgm (as above)
- leftim.ppm (as above)
- rightim.ppm (as above)
- sift_left.mat (SIFT features and their positions for the left image)
- sift_right.mat (SIFT features and their positions for the right image)
- detections_left.harraff & detections_right.harraff (needed for visualizing features with display_features.m)

Now implement a sparse stereo matcher. The SIFT features in the provided files were extracted from interest points detected with Schmid et al.'s Harris-Affine detector. While above we compared pixel intensities in regular fixed-size windows, here we will compare SIFT descriptors extracted from sparse elliptical regions of varying sizes. While the ellipse regions may have different scales, though, each SIFT descriptor is a 128-dimensional vector.

Note that the sift_{left,right}.mat files contain both the descriptors as well as the associated image positions (x, y) for each feature. There may be multiple features extracted at a given position with different scales or orientations. You will need the descriptors to find similar features, and the positions to compute the disparities. You can round the positions to integer values to simplify the row-by-row computation below.

Write a function called 'sparse_match' that takes the SIFT and position data for a left/right pair, and returns the sparse disparity estimates. For each row in the left image, identify any sparse features that are located on it, i.e., have an associated image position with the current *y* coordinate. For each of those features, determine which of the features from the right image on that scanline best matches. For a matched pair, compute the disparity as the distance in the *x*-coordinates of the associated image positions.

Use the Euclidean distance (L_2) between SIFT descriptors as the cost function (an efficient implementation is in 'dist2.m'). It may be helpful to place a limit on the maximum disparity at which you will consider matches, as is essentially done above in (a). Note, you will not have a disparity estimate for every pixel in the input now.

In your code, compare the errors obtained with the dense matching to the errors obtained with this sparse matching----*but only at the positions where the sparse stereo matching gave a result.*

After processing all of the stereo pairs, write answers to each of the following:

i) How do the sparse and dense disparity estimates compare?

ii) Which sparse feature matches are good for the depth computation, and which may be problematic? It may be helpful to check some examples to see which sparse features in the left image are matching which in the right image.

c) Extra credit: robust matching constraints (Optional, worth up to 15 extra points)

For either your sparse or dense stereo matching implementation, improve the robustness by incorporating one or more of the constraints for correspondences discussed in class (e.g., continuity, ordering, smooth disparity, etc.) Describe your addition clearly, and evaluate the amount of improvement relative to ground truth.

2. Indexing scenes with local invariant features (30 points)



For this problem, use the image data in the directory named 'boston'. There are a few images each from several different scenes, and each view contains some amount of viewpoint change, occlusion, and/or scale change. The goal is to use matches between local invariant features to identify a query scene.

Implement a simple voting scheme, relying on the premise that many similar local features in two images indicate that the same object is present. First, create a pool of feature vectors from all of the examples. Then treat each image as a query, in turn. For a given query input, look at each one of its features, and find the nearest feature within the pool from some other image. Cast a vote for that image. After going through all the features in the query image, tally the votes for the remaining images, and rank them based on how many votes they receive.

The SIFT features associated with each image are saved in the associated files: 'imageXXXX.pgm' goes with the SIFT data in "imageXXXX.pgm.sift.mat'. As before, each SIFT file contains a 'features' matrix and 'position' matrix.

For each query, display the image for that query together with the other images that are ranked highest based on this voting.

Answer the following:

i) If you use the top ranked image as the location for this scene, how often is it correct?

ii) What could be adjusted to make this more robust for those cases where it is incorrect?