CMPEN/EE 454 Computer Vision I
Project Assignment 2
Due Friday October 27

Computing Dense Disparities with Simple Stereo

In this assignment we will compute dense stereo disparity maps using patch-based matching along horizontal epipolar lines. We have gone over all the relevant concepts in class during lectures 11 and 12 (about stereo) and also lecture 7 (about patch matching).

Your program will need to know the following things:
- Filename of the Left image
- Filename of the Right image
- Max disparity
- Patch width

Left and Right images are from a stereo pair where corresponding image patches must lie along horizontal scan lines (rows of each image). This is consistent with the "simple stereo" geometry we have studied in class, with the right camera having been displaced some distance along the positive X axis, while keeping the right and left camera optical axes parallel.

Max disparity is the maximum disparity value in pixels that you can expect to have. Recall that for a patch in the left image centered at column x0, if the matching patch in the right image is centered at column x1, then the disparity value will be x0-x1 and will be nonnegative. Max disparity is a conservative estimate in that it may be larger than the maximum disparity you will actually find in practice for the given image pair. However, you can rely on it providing a guaranteed bound on the range of disparities you need to consider, specifically 0 <= disparity <= max disparity , which narrows down your search range making your matching procedure both more efficient and more likely to find correct matches.

Patch width will specify the width (and height) of a square patch size to use for matching. You should assume that it is always an odd number, e.g. 5 or 7 or 11, etc. Make sure you code this as a parameter that you can easily change – we are going to ask you to experiment with different patch sizes and compare results.

With the above information, the output of your program will be an image the same size as the Left image, containing disparity values for each pixel in that image, or -1 to

flag that you could not compute a disparity for that pixel (recall that disparities are always nonnegative, so -1 can never be confused for a valid disparity value).

What your program should do

1) Read in each image using imread. They are color images, and VERY large. I suggest that you first convert them to floating point, single channel images, since all of our patch matching scores in Lecture 7 were defined just for intensity (aka greyscale aka single channel) images. For example, you could get a single channel image from 3-channel color by computing (r+g+b)/3 or by just taking the green channel. You probably also want to reduce the size of each image, at least for testing. I suggest using matlab's imresize command to do this. Hint: if you reduce the width of the image by N, you also want to divide the max disparity value by N, to maintain an equivalent disparity range constraint at the new scale.

2) Compute disparity image. It makes sense to organize this computation row by row, working your way down the rows of the left image and for each patch along the row, finding an estimated matching patch centered along the same row in the right image. Thus, as each row in the left image is considered, a new row of disparities will be computed and stored in the output disparity image.

3) To do the actual patch matching, recall that we talked about 3 different kinds of patch matching functions: raw correlation (after first subtracting the mean of the patch from the left image that you are trying to match), SSD, and NCC. Use at least one of these. Since for each patch along the row in the left image, you are comparing to several candidate patches along the same row in the right image, it behooves you to make this comparison as fast as possible, ideally using imfilter or normxcorr2 to compare the left patch with a whole swath of candidate right patches by leveraging these efficient, vectorized matlab functions. How you do this will be totally up to you. This is the main area of cleverness needed for this project. Also keep in mind that you should constrain the locations of matching patches to be only those with disparities in the correct range 0 <= disparity <= max disparity.

4) Once your estimated disparity image is filled in, save it as a Matlab file. If your disparity image is called dispimage, do something like

"save outputfilename.mat dispimage"

to save it to a file. You will be handing in this file. Also display the file using imagesc, and take a screenshot for putting in your report.

5) Rerun the experiment using different patch sizes and compare results. Try at least 5x5, 7x7 and 11x11.

6) Run your algorithm for at least one other pair of images from the same dataset the motorcycle images that I am giving you as input come from. To do this, go to http://vision.middlebury.edu/stereo/data/scenes2014/ , scroll down to "10 evaluation training sets with GT" or "13 additional datasets with GT" and select an example image you like. Click on the "perf" link above the picture you chose. You will see a bunch of files. Download im0.png and im1.png to use as Left and Right images, respectively. Click on the calib.txt file and find the line that starts with "ndisp=". The number on that line will be the max disparity value to use for this image pair. By the way, the direct link to the motorcycle data that we are using is: http://vision.middlebury.edu/stereo/data/scenes2014/datasets/Motorcycle-perfect/

What to hand in

Half of your grade will be based on submitting a runable program, and the other half will be based on a written report discussing your program, design decisions, and experimental observations. The report should include at least the following things:

a) Summarize *in your own words* what you think the project was about. What were the tasks you performed; what did you expect to achieve?

b) Present an outline of the procedural approach along with a flowchart showing the flow of control and subroutine structure of your Matlab code. Explain any design decisions you had to make. Pay particular attention to explaining what patch matching score function you used, and how you implemented it to be as efficient as you could make it.

c) Experimental observations. What do you observe about the behavior of your program when you run it? Does it seem to work the way you think it should? How long does it take to run? How does runtime relate to size of the stereo images used and size of the patches used? What differences do you observe in the output disparities computed for different patch sizes? Are there visually obvious errors in the output disparities (hint: there will be many!) and where do they seem to occur most? Can you think of reasons why matching failed in those image locations?

f) Document what each team member did to contribute to the project. It is OK if you

divide up the labor into different tasks. This is also where you let us know if any of your teammates were slacking off on this project.

Optional Additions

Here are some ideas of additional things to try. This is purely optional, but if you do one or more of them well you can get additional points for it.

A1) (+5) Implement all three patch matching scores and compare/discuss the output results using the same image pair. Also download the companion images im1E.png and im1L.png from the appropriate Middlebury dataset page. The "E" image is taken with a different camera exposure; the "L" image is taken with different lighting conditions. Replace im1.png with each of these to run another round of experiments, trying all three patch matching scores and comparing results. Do any of the three match score functions do a better job than the others when the camera exposure or lighting conditions are different?

A2) (+5) Quantitatively evaluate your results using the known ground truth (GT) disparity file called "disp0.pfm" posted with the original dataset. This is a floating point image the same size as the original Left image, containing very accurately measured disparity values for each pixel. I have posted a function called parsePfm.m that can read this file. You will have to figure out and justify in your report a method for comparing your disparity results with the GT results. Something like mean squared difference error would be a useful place to start. However, you need to be aware that there are places where GT data is not available (values are "inf" in the ground truth files) and that there are places in your results where you were perhaps not able to compute valid values. You should only quantitatively evaluate accuracy of the disparity values that are valid in common across your results and the GT results. Also remember that if you reduced the size of the input images, you should also divide the ground truth disparity values by the same amount.

A3) (+10) Use the dynamic programming technique presented in lecture 12 to compute "optimal" matches for each row within the context of a disparity space image (DSI) representation. Specifically, for a given row in the left and right images, compute all pairs of match scores to form the DSI. Then use DP to determine a path from cell 1,1 in the disparity space image to cell M,N , thus determining for each pixel

in the left image row: i) whether it matches a pixel in the right hand row, and which pixel it matches; ii) whether it is occluded from the left; or iii) whether it is occluded from the right (note, only one of these three possibilities holds at each pixel). Then compute a disparity for the matched pixels along the row and fill in the corresponding row in the output disparity image. Do this for each row of patches in the left image, thus computing a full set of disparities for the whole left image. That is, as each row in the left image is considered, a new row of disparities will be computed and stored in the output disparity image. More details on the DSI data structure and the DP path finding algorithm are in Lecture 12, and the DP algorithm details (including pseudocode!) can be found in Section 2.1 of Cox, Hingorani, Rao, Maggs, "A Maximum Likelihood Stereo Algorithm," Computer Vision and Image Understanding, Vol 63(3), May 1996, pp.542-567.