Computer Vision I
CSE 252A, Fall 2009
David Kriegman

Name :
Student ID :
E-Mail :

Assignment #3 : Dense Stereo Matching

(Due date: 11/24/09)

---

1 Introduction

In this assignment you will implement an algorithm for binocular stereo vision. Your data will be pairs of stereo images that are available on the course website. You will hand in results for the three stereo pairs from that page. The images have been rectified so that the scan lines are epipolar lines. For each image pair, you will ultimately seek to output a disparity map for the left image indicating the disparity to the right image.

You can download all images for this assignment from the link in the HW3 section of the course website.

Stereo with Dynamic Programming

Here, you will implement a stereo algorithm that uses dynamic programming. This algorithm enforces the ordering constraint (or else you couldn't use dynamic programming) and matches individual pixels. Every pixel in one image can either match one pixel in another image, or be marked as occluded. The penalty for matching two pixels is the square of the difference in their intensity. The penalty for being occluded is a fixed value, c_{occ}.

In addition to the class lecture notes, you should consult the paper: "A Maximum Likelihood Stereo Algorithm", by Cox, Hingorani, Rao, and Maggs, from the journal Computer Vision and Image Understanding, 63, 3, pp. 542-567 (available from the course web page). You should focus on section 2.1, which covers dynamic programming; the pseudo-code provided in figures 2 and 3 will provide a good starting point for your own code[1]. Please note that for consistency with that paper, all thresholds below are given assuming that image intensities fall in the range 0 to 1.

Part A: 1D Disparity Matching

Implement the dynamic programming stereo algorithm in 1D. You should write a function of the form

```
d = stereo_1d(v1,v2,c_occ)
```

where v1 and v2 are vectors to be matched, and d contains the disparity for every pixel in v1. For this assignment, we will define disparity such that if $v_1(i)$ corresponds to $v_2(j)$, then the disparity is given by $d = i - j$. If $v_1(i)$ has no corresponding pixel in v_2, then this is considered an *occluded* pixel. You should write your code such that the "disparity" of any occluded pixels are set to NaN. Test your function on the following input,

```
v1 = [3 2 6 8 7 0 2 9 3 0 4];
v2 = [3 2 0 6 8 2 0 0 9 0 3 4];
c_occ = .01;
```

Turn in your code and the result of this test.

[1]The rest of the paper may be useful, but you don't need to implement or understand their maximum likelihood cost function or anything of that nature.

Part B: 2D Disparity Matching

Extend this algorithm to 2D. We assume the images are rectified, so that the epipolar lines are horizontal scan lines of the input images. Thus, you just need to run stereo_1d on each corresponding line in the two images. Your function should have the form: D = stereo_2d(I1, I2, c_occ). Test your algorithm on the three image pairs (DOTS,BARN1,TSUKUBA) using an occlusion penalty of $c_{occ} = 0.01$ and display the resulting disparity maps as images. For display purposes, the disparity values should be mapped to the range [0,1] and, to distinguish valid disparities from occlusions, you should *colorize* the image so that occluded pixels are shown in color while the rest of the disparity map is shown in grayscale. The following code will scale the values appropriately and show occlusions in blue,

```
function [d_color] = display_dmap(d)

% Map disparity into the range [0,1]
max_d = max(d(:)); min_d = min(d(:));
d_scaled = (d-min_d)/(max_d-min_d);

% Colorize occluded pixels to be blue
d_color = repmat(d_scaled(:),[1 3]);
occ_inds = isnan(d(:));
d_color(occ_inds,1) = 0;
d_color(occ_inds,2) = 0;
d_color(occ_inds,3) = 1;
d_color = reshape(d_color,[size(d) 3]);

% Display image
image(d_color);
```

Turn in your code and an image showing the resulting disparity map for each input image pair (DOTS,BARN1,TSUKUBA).

Part C: Different Cost Metrics

Up to this point, we have used the squared difference in pixel value as our matching cost. In this section we will extend our definition of matching cost to include neighborhoods of pixels. We will evaluate two different cost metrics: Sum of Squared Differences (SSD) and Normalized Sum Squared Differences (NSSD), defined in Table 1.

To illustrate the process, suppose we wish to compute the SSD match cost between pixel (x_1, y) in image 1 and pixel (x_2, y) in image 2. We first extract a window W_1 centered at pixel (x_1, y) from image 1. We then extract a window W_2 centered at pixel (x_2, y) from image 2. Plugging into the formula for the SSD cost we get $c = \sum_{i,j} |W_1(i,j) - W_2(i,j)|^2$. Evaluate your algorithm using the SSD, and NSSD match costs with window sizes of 3x3 and 5x5 (4 disparity maps in total). The occlusion penalty you should use for each match cost is provided in Table 1.

Since evaluating these cost functions can be computationally intensive, you may find it helpful to optimize your implementation to get acceptable run-times. The following list of suggestions may be of help,

- Choose reasonable values for the minimum and maximum disparity (you should be able to roughly determine this by looking at the input images) and only evaluate the match cost for pixels in this disparity range.
- Think of ways to compute the match cost in batch (i.e., without using for loops).
- Take advantage of Matlab's profiling tool (available under Desktop → Profiler in the menu). This will isolate the slowest parts of your code.
- Implement the algorithm in C (only if you are unable to get acceptable performance via other means – for comparison, the reference implementation takes about about 2 minutes on the Tsukuba pair using NSSD and with a 5x5 window).

Implementing the first suggestion is probably sufficient to make your algorithm run in a reasonable amount of time.

What to hand in: (1) Disparity maps for DOTS, BARN1, and TSUKUBA using SSD, and NSSD with windows of 3x3 and 5x5. You can use the recommended occlusion penalty indicated in Table 1; however, you may find that some tweaking of the occlusion penalty on the different image sets is helpful.

	Match Cost Function	Recommended c_occ
SSD	$\sum_{i,j} \left\|W_1(i,j) - W_2(i,j)\right\|^2$	$0.01N$
NSSD	$\sum_{i,j} \left\|\widetilde{W}_1(i,j) - \widetilde{W}_2(i,j)\right\|^2$	1.5

Table 1: The different matching costs used in this assignment. W refers to a window of pixels and $\widetilde{W} = \frac{W - \overline{W}}{\sqrt{\sum_{k,l}(W(k,l) - \overline{W})^2}}$ is a mean-shifted and normalized vesion of the window. N refers to the number of pixels in each window.

Part D: Epipolar Rectification

In Part B, we assumed that images have been rectified as a preprocessing step, such that the epipolar lines are horizontal scan lines of the input images. Suppose that we have captured two images I_A and I_B from identical calibrated cameras separated by a rigid transformation

$$ {}^B_A\mathcal{T} = \begin{bmatrix} \mathcal{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{1} $$

Recall from lecture that for each image a rectifying transform should map each epipole to the point infinitely far away in the horizontal direction $H_A\mathbf{e_A} = H_B\mathbf{e_B} = [1, 0, 0]^T$. Consider the following special cases:

1. Pure horizontal translation $\mathbf{t} = [t_x, 0, 0]^T$, $\mathcal{R} = I$
2. Pure translation orthogonal to the optical axis $\mathbf{t} = [t_x, t_y, 0]^T$, $\mathcal{R} = I$
3. Pure translation along the optical axis $\mathbf{t} = [0, 0, t_z]^T$, $\mathcal{R} = I$
4. Pure rotation $\mathbf{t} = [0, 0, 0]^T$, $\mathcal{R}$ is an arbitrary rotation matrix

For each of these cases, determine whether or not epipolar rectification is possible. Include the following information for each case

- The equation of the epipolar line ℓ_B in I_B corresponding to the point $[x_A, y_A, 1]^T$ in I_A (if one exists)
- The epipoles $\mathbf{e}_A$ and $\mathbf{e}_B$
- A plausible solution to the rectifying transforms H_A and H_B (if one exists) that attempts to minimize distortion (is as close as possible to a rigid-body transformation). Note that the above 4 cases are special cases; a simple solution should become apparent by looking at the epipolar lines.

One or more of the above rigid transformations may be a degenerate case where rectification is not possible or epipolar geometry does not apply. If so, explain why.

What To Turn In

1. Email your source code to sbranson AT cs.ucsd.edu.
2. Hand in a report which contains
   (a) A hardcopy of your code.
   (b) A report which includes the output described above and your answers to Part D. Also, include a short summary of your conclusions, based on your experiences while completing this assignment.

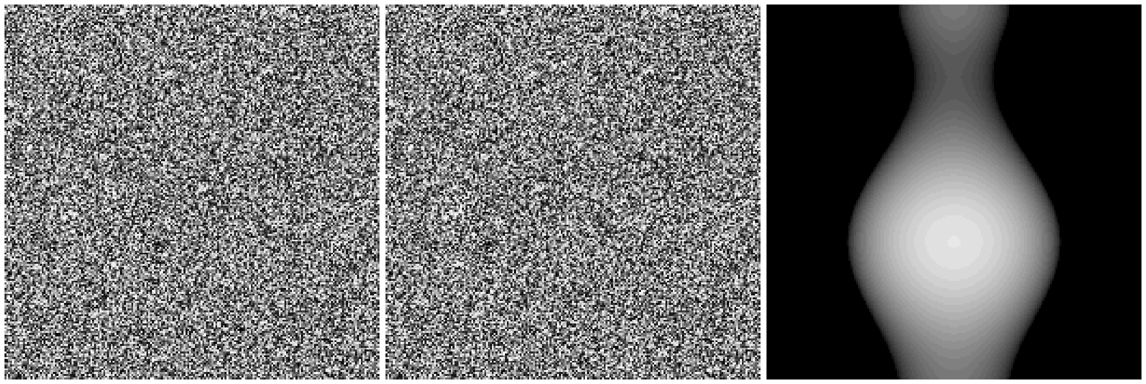

Figure 1: Random dot stereogram image pair and true depth map.

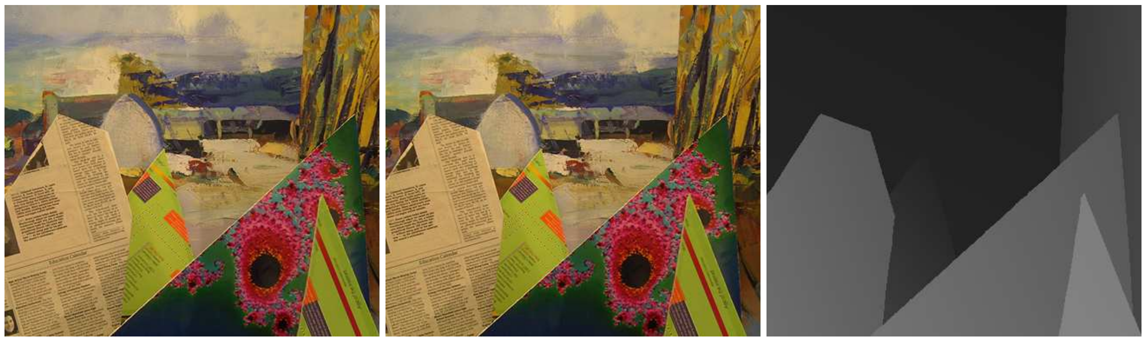

Figure 2: Barn1 image pair and true depth map (courtesy Middlebury College).

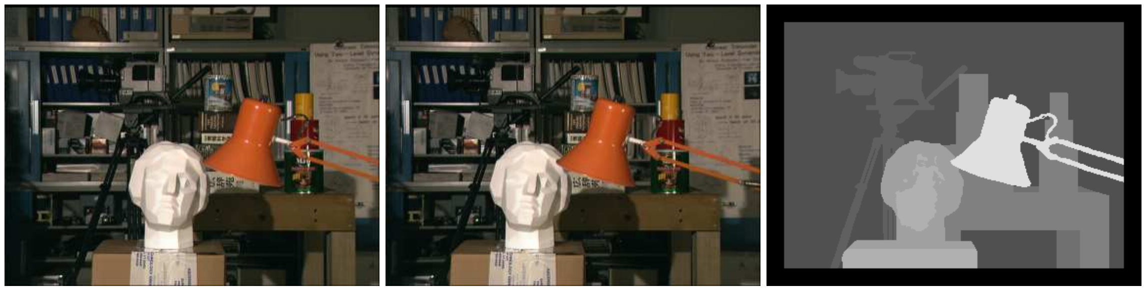

Figure 3: Tsukuba image pair and true depth map (courtesy University of Tsukuba).