CSCI 4830/5722 Computer Vision, Fall 2016
Instructor: Fleming
Assignment 4, Due Wednesday, November 2nd, by 11:55pm

Dynamic Programming

Task 1 (20 points): Implement the Longest Common Subsequence (LCS)
Given two sequences, X ={x_1; ... ; x_m} and Y ={y_1; ... ; y_n}. Find a subsequence common to both whose length is longest. A subsequence doesn't have to be consecutive, but it has to be in order.

H O R S E B A C K
S N O W F L A K E

} LCS = OAK

The LCS problem has 2 versions:

- The *Simple* version, requesting only to find out the length of the longest common subsequence.
- The *Complete* version, requesting to find out the sequence itself.

For this assignment, you need to implement the *Complete* version using the Dynamic Programming approach.

X = {x_1, ... x_m}
Y = {y_1, ...,y_n}
X_i = the prefix subsequence {x_1, ... x_i}
Y_i = the prefix subsequence {y_1, ... y_i}
Z ={z_1, ... z_k} is the LCS of X and Y .
LCS(i,j) = LCS of X_i and Y_j

We can recursively define the LCS as:

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ LCS(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(LCS(i-1, j), LCS(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

Use the recursive definition to set up your memorization table and compute the size of the longest subsequence and the characters that are part of it.

Task 2 (40 points): Calculate stereo disparity using the DP (as outlined below)

Here, you will implement a stereo algorithm that uses dynamic programming. This algorithm enforces the ordering constraint, the uniqueness constraint, and matches individual pixels based on a cost function. Every pixel in one image can either match one pixel in another image, or be marked as occluded.

Note: this algorithm assumes the image intensities in the left and right image fall in the range 0 to 1.

Part A (20 points): Disparity matching along each epipolar lines
Just like in Assignment 3, you will use the files provided with the script `DepthEstimationFromStereoVideoExample`. Your DP algorithm will replace the *disparity.m* built-in Matlab function. For each image pair, you seek to output a disparity map for the left image indicating the disparity to the right image. The cost function is defined such that the penalty for matching two pixels is the square of the difference in their intensity. The penalty for a pixel being occluded is a fixed value, *occ*.

The images (frames in the video) are already rectified, so that the epipolar lines are horizontal scan lines of the input images. Thus, you just need to run the DP stereo algorithm on each corresponding line/row in the two images. You will need to call your function once for each epipolar line. Your function should have the form:

```
D = stereoDP(e1, e2, maxDisp, occ)
```

The recommended value for `occ` is `0.01`. For `maxDisp`, you can start with the value `64` (this is the maximum disparity value as resulting from using the built-in Matlab *disparity.m* function). Feel free to try a lower value and notice if your disparity maps improves.

Algorithm:
Consider two scanlines $I_l(i)$ and $I_r(j)$, $1 \le i, j \le N$, where N is the number of pixels in each line (the process will be repeated for each row of the image). Pixels in each scanline may be matched or skipped if they are considered to be occluded, in either the left or right image).

Let d_{ij} be the cost associated with matching pixel $I_l(i)$ with pixel $I_r(j)$. Here we consider a squared error measure between pixels given by:

$$d_{ij} = (I_l(i) - I_r(j))^2$$

The cost of skipping a pixel (in either scanline) is given by a constant `occ`.

We can compute the optimal (minimal cost) alignment of two scanlines recursively as follows:

1. D(0,j) = j * occ
2. D(i,0) = i * occ
3. D(1, 1) = d_{11},
4. D(i, j) =
 = min(D(i−1,j−1) + d_{ij} , D(i−1,j) + occ, D(i,j−1) + occ)

Note: just like in the LCS complete problem, you will need to save which "direction" was used for the calculation of each `D(i, j)` value: North, West, or Northwest.

Part B (10 points): Backtracking
The goal is to find the optimal alignment (and thus the disparity) by backtracking. Starting at (i, j) = (N, N), trace your path of minimum cost all the way to (1,1). You will need to use the "directions" saved in part A. Selecting (i − 1, j) corresponds to skipping a pixel in I_l (a unit increase in disparity), while selecting (i, j − 1) corresponds to skipping a pixel in I_r (a unit decrease in disparity). Selecting (i − 1, j − 1) matches pixels (i, j), and therefore leaves disparity unchanged.

Part C (10 points): Displaying the disparity map
For display purposes, the disparity values should be normalized and 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. Here is pseudo-code for scaling the values appropriately and showing occlusions in a different color:

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

% 1.  Map disparity into the range [0,1]
% max_d = the maximum calculated value of disparity;
% min_d = the minimum calculated value of disparity;

% scale the disparity values by subtracting the minimum
% value min_d and dividing by the difference beween max_d
% and min_d

% 2. Colorize occluded pixels to be red
% dColor = color image where each RGB layer is equal to the
% scaled disparity matrix (values between 0 and 1)

% find the positions/indices where each of the 3 values of
% dColor is equal to NaN, and store them in a variable

% replace the values of these positions with:
% dColor(at position in R layer) = 1;
```

```
% dColor(at position in G layer) = 0;
% dColor(at position in B layer) = 0;

% 3. Display dColor image using imshow
```

Part C (extra credit 20 points): 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:

1. **(10 points)** Sum of Squared Differences (SSD) and
2. **(10 points)** Normalized Cross Correlation (NCC).

We wish to compute the SSD/NCC 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. The size of the two windows should be the same. Use the same definition of window matching with SSD and NCC as in Assignment 3.

For this assignment (extra credit), evaluate your algorithm using the SSD and NCC match costs with window sizes of 3x3 and 5x5 (four disparity maps total, 5 points each). You can start with the same occlusion penalty (`occ`) as for 1-pixel SSD, and then try to change this value and see how it affects your disparity maps.

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:

- Adjust the values for the minimum and maximum disparity for each epipolar line and only evaluate the match cost for pixels in this disparity range. You can use the values obtained at the first pass and then reduce the search range accordingly.
- Think of ways to compute any part of the match cost using vectorized code rather than loops.
- Take advantage of Matlab's profiling tool (available under Desktop → Profiler in the menu). This will isolate the slowest parts of your code.

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

Task 3 (20 points): Calculate stereo disparity using the DP (cones images)
Task 4 (20 points): Calculate stereo disparity using the DP (teddy images)
The left and right images for the *cones* and the *teddy* sets, as well as the left and right disparity maps (ground truth) can be found here:
http://vision.middlebury.edu/stereo/data/scenes2003/

Submitting the assignment:
Make sure each script or function file is well commented and it includes a block comment with your name, course number, assignment number and instructor name. Zip all the .m files together and submit the resulting *.zip* file through Moodle as Assignment 4 by Wednesday, November 2nd, by 11:55pm.