CSCI 4831/5722 Computer Vision, Spring 2022
Instructor: Fleming
Homework 4, Due Friday, March 4th, by 11:55pm

Stereo Vision. Dynamic programming

For this homework assignment, we are asking you to implement some of the functions in a stereo pipeline example provided by mathworks.com

Provided files:

- **Files for students using Matlab**:

  In your Matlab Command Window, type

```
>> openExample('vision/DepthEstimationFromStereoVideoExample')
```

  A Matlab Live Code file will open: *DepthEstimationFromStereoVideoExample.mlx*. Save it as a .m file. You will keep this template, but you will have to write your own version of some of the functions called from this script.

  Files needed:

  1. DepthEstimationFromStereoVideoExample.m
  2. Four images for task 6

- **Files for students using Python**:
  1. DepthEstimationFromStereoVideoExample.m – for reference
  2. frameLeftgray.png – rectified, grayscale, left image
  3. frameRightgray.png – rectified, grayscale, right image
  4. disparityMAP_Matlab.png – the disparity map, as calculated by the Matlab function, for reference and for plotting in some homework tasks
  5. stereo parameters (for both cameras): *stereoParams.mat* – you will need to unpack the data in this cell array; some values will be needed for computing depth (task 5)
  6. Four images for task 6

What You Have to Do:

Task 1: Calculate disparity using the SSD algorithm
Implement the SSD matching algorithm for a range of window sizes, and create a disparity image D(x, y) such that Left(x, y) = Right(x + D(x, y), y) when matching from left to right.

Write a function *disparitySSD* to replace the *disparity* built-in function on line 50:

```
disparityMap = disparity(frameLeftGray, frameRightGray);
```

Your function needs to accept an additional input value: the size of the search window. You can assume the size of the search window in an odd integer. At the very least, your function should work for the window size values of 1, 3 and 5.

Call the *disparitySSD* function three times for the following window sizes: 1, 3 and 5. Create a 2x2 subplot and display the disparity map results from these three function calls, plus the one obtained using Matlab's built-in *disparity* function (this is the image file *disparityMap.png* for students working in Python).

Notes:

1. Remember to reduce the search only over epipolar lines.
2. You should also restrict the search using the maximum disparity range. The maximum disparity value as calculated by using the built-in Matlab *disparity.m* function is 64 (see line 52 in the original script). Feel free to try a lower or a higher threshold and notice if your disparity maps improve.
3. When using a window size bigger than 1, it is common practice to use a Gaussian weighting of the window prior to computing the sum of squared differences. The size of the Gaussian kernel should be the same as the window size.
4. For this task, you do not need to impose any constraint on the matching algorithm. But, if your matching algorithm finds more than one "best match", you will want to come up with a rule for selecting just one match and stick to it. Don't forget to let us know in your comments, what rule did you use.

Task 2: Uniqueness constraint. So far, you have found matches based on an *epipolar constraint* (matches lie on the same scanline) and on the *similarity constraint* (because each unique pair of visual features originate from the same physical surface or object, they naturally look very similar and should have similar intensity values). Modify your implementation of the `disparitySSD` function to incorporate the **uniqueness constraint**, defined as follows: one visual feature in one retinal image can have at most one corresponding counterpart in the other image. This is explained by the fact that one physical point/surface always 'causes' at most one feature in each retinal image, and one physical point/surface cannot project at two retinal locations at the same time. Your algorithm should enforce a rule that, for each left-/right- eye feature, there will be only one corresponding feature in the right-/left- eye image.

Note: There is no unique correct answer here. Think about it, be creative and come up with your own solution to this constraint. This is not a competition with your peers.

Task 3: Disparity smoothness constraint. So far, you have found matches based on an *epipolar constraint*, a *similarity constraint*, and a *uniqueness constraint*. Modify your implementation of the `disparitySSD` function to take into account the **disparity smoothness constraint**, defined as follows: adjacent points (based on their location along each scanline) should have similar disparities.
Note: You do not need to enforce smoothness between scanlines.

Notes and Suggestions:

1. *This task is required for graduate students. If you are an undergraduate student and you attempt this task, it will count as Extra Credit.*
2. There is no unique correct answer here. Think about it, be creative and come up with your own solution to this constraint. This is not a competition with your peers.
3. In order to enforce a disparity smoothness constraint, you need to have a starting point; which point on each line should be the first one you want to calculate the disparity for? The first disparity value will determine the disparity values of the rest of the points.
4. You will need to define a *threshold of smoothness* parameter. What do you think would be an acceptable disparity gradient value? Feel free to experiment with multiple values.
5. You are allowed to use Matlab or Python built-in functions for keypoint detections (i.e. Harris corners, SURF features). If you wish, you can also use different descriptors for the SSD computation.

Task 4: Generate outliers map - Refine the disparity by performing a left-to-right consistency check.

The disparity map $d_{LR}(x)$ is acquired considering as reference image the left image of the stereo pair. If the right image is considered as reference, then the disparity map $d_{RL}(x)$ is acquired. The disparity maps $d_{LR}(x)$ and $d_{RL}(x)$ can be useful in detecting problematic areas, especially outliers in occluded regions and depth discontinuities. One strategy for detecting outliers is the Left–Right consistency check introduced by (1). In this strategy, the outliers are disparity values that are not consistent between the two maps and therefore, they do not satisfy the relation:

$| d_{LR}(x) - d_{RL}(x - d_{LR}(x)) | \leq T_{LR}$, where T_{LR} is a user-defined threshold.

Write a function *detectOutliers* that accepts as inputs two disparity maps (LR and RL) and a value for T_{LR} and returns a binary image where the outliers have the value 1 and the inliers have the value 0.

Call this function twice, passing as inputs the LR and RL disparity maps from SSD matching with window size of 3 and 5. Use a T_{LR} value of 1. With this value for T_{LR}, only pixels with a difference greater than 1 in the Left–Right consistency check will be considered outliers. Plot the two resulting binary images side by side.

Task 5: Compute depth from disparity
Now that we have a disparity image, computing depth at every pixel in the left image should be very straightforward. Use equation 11.1 from the Szeliski textbook.

Write a function `reconstructSceneCU` to replace the `reconstructScene` built-in function on line 60. Your function should have the same input arguments as the built-in version and it should return a matrix of depth values for each pixel location from the left image.

Task 6: Synthetic stereo sequences
Note: *This task will count as Extra Credit, for both undergraduate and graduate students.*

As you're developing stereo matching algorithms, it might prove useful to test them on synthetic stereo sequences, and to compare with a ground truth disparity map.

Use the provided pair of synthetic images (the *Teddy* folder) and your disparity implementation functions from Tasks 1-3 to generate your disparity map(s). Compare your result to the corresponding ground truth (provided):

1. Compare disparity maps visually by plotting side by side
2. Calculate a map of errors and display it
3. Calculate a histogram of disparity differences and display it – the bin size is up to you.

Dynamic Programming

Task 7: 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 (i.e. the image is grayscale).

Part A: Disparity matching along each epipolar lines
Write a function *disparityDP* that uses the DP algorithm. The function will replace the *disparity.m* built-in Matlab function, just like the ones for Task 1 and 2. For each image pair, you seek to output a 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 improve.

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: Backtracking
Given D we 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: Displaying the disparity map, again

For display purposes, the disparity values should be normalized and mapped to the range [0,1]. In order 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
```

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:

1. 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 values obtained at the first pass
2. Think of ways to compute any parts of the match cost using vectorized code rather than loops

3. 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.

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. Save all the figures/plots your program generates as image files. Zip all the .m (or .ipynb or .py) files and image files together and submit the resulting *.zip* file through Canvas as H4 by Friday, March 4th, by 11:55pm.

(1) M. Humenberger, T. Engelke, W. Kubinger, *A census-based stereo vision algorithm using modified semi-global matching and plane-fitting to improve matching quality*, CVPR ECV Workshop, 2010.

Rubric:

	Undergraduate	Graduate
Q1	25	20
Q2	30	25
Q3	10 EC	25
Q4	25	20
Q5	20	10
Q6	10 EC	20 EC
Q7 A.	50	50
B.	25	25
C.	25	25
TOTAL	**100**	**100**
EC	20	20