Dynamic Programming for Stereo Correspondence and Disparity

Wednesday, October 18, 2017 4:40 AM

Stereo Imagery.

One goal in computer vision within the context of stereo imagery is the goal of depth computation.

- An object in the scene may have a corresponding projections (pixels) in each of the stereo images. The goal is to compute the depth of the object.
  - This computation may be relative or absolute based on the information given.
- If pixel correspondences can be identified, then triangulation can be used to determine the depth of the image object associated with the pixel pair.

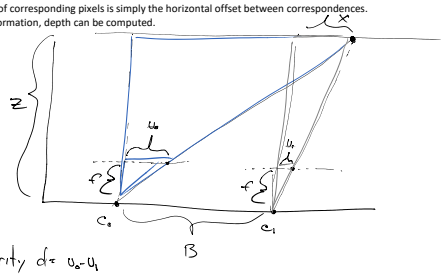
Thus the problem of computing the depth of objects is reduced to finding pixel correspondences in our stereo images. The search of pixel correspondences is reduced if we know corresponding epipolar lines or if the images have been rectified.

Rectification.

Assume two corresponding stereo images I0 and I1 have been rectified such that corresponding epipolar lines are horizontal and collinear (similar rows correspond.) Thus we need only compare scanlines between the stereo pair. Thus the problem of computing disparity is simplified.

Disparity.

The disparity of corresponding pixels is simply the horizontal offset between correspondences. Given this information, depth can be computed.



disparity $d = u_0 - u_1$

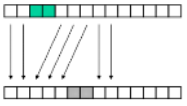
By similar triangles: $\frac{u_0}{f} = \frac{B+x}{Z}$ & $\frac{u_1}{f} = \frac{x}{Z}$

Thus $\frac{u_0 - u_1}{f} = \frac{B+x-x}{Z}$

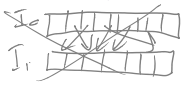
$\therefore \quad Z = f\frac{B}{d}$

Finding pixel correspondences in corresponding Image rows (scanlines).

Two pixels are in correspondence when (x,y) in image 0 corresponds to (x',y') in image 1,
where y' = y (same row)
and
where x' = x + d, where d is the disparity



Observe: this is simply a sequence of correspondences. Without constraints the number of potential sequences is large; however, we can impose some intuitive constraints to as to reduce our search space.

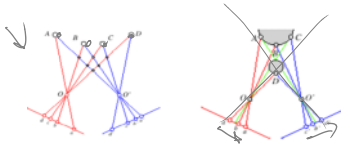


Constraints.

- Ordering Constraint
- Uniqueness*: each pixel in Image 0 can only correspond to 1 pixel in Image 1

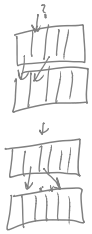
Ordering Constraint: monotonic ordering

If pixel i in image 0 corresponds to pixel j in image 1, then pixel i+1 in image 0 can only match pixel k in image 1 where k > j



How can we determine the "best" matching correspondence sequence?

- Energy minimization approach (we investigate dynamic programming herein)
  - Pixel matching criterion
    - Simple: Based on intensities
    - Feature matching
    - Windowed (neighborhood) matching.
  - "smoothness" constraint -- **order**.
    - Smoothness penalty for discontinuous (undefined) disparity
    - This penalty can be relaxed given the context of the image.



$E_{data} = [I_0(x+d,y) - I_1(x,y)]^2$

(cost for pixel match)

$E_{smooth} = |d_0 - d_1|$ Or something similar

(discontinuity penalty)

Given our constraints, given a correspondence for pixel i in image 0, there are 3 cases for correspondences for pixel i+1.

Case 1: Correspondence is found!

Ordering Constraint: monotonic ordering

If pixel i in image 0 corresponds to pixel j in image 1, then pixel i+1 in image 0 can only match pixel k in image 1 where k > j

Given our constraints, given a correspondence for pixel i in image 0, there are 3 cases for correspondences for pixel i+1.

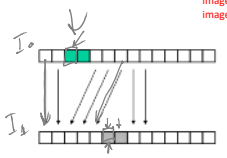
Case 1: Correspondence is found!

Case 2:
Correspondence is not found in image 0

Case 3:
Correspondence is not found in image 1

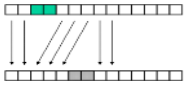
Case 4: Repeated cases 2 + 3

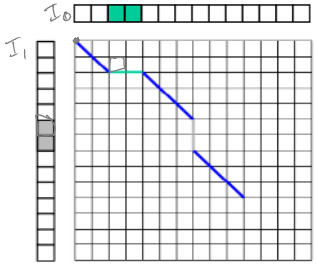
Ordering Constraint: monotonic ordering
If pixel i in image 0 corresponds to pixel j in image 1, then pixel i+1 in image 0 can only match pixel k in image 1 where k > j

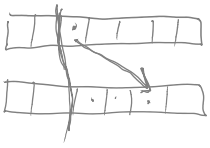


Observe: Since our sequence is a monotonic listings of (i,j) pairs we can view all possible matches as a move on a grid or table.

Ordering Constraint: The ordering constraint restricts which paths are permitted upon this table. All possible moves on the grid are
- Case 1: "diagonal": ... (i,j), (i+1, j+1) ...
- Case 2: "down": ... (i,j), (i+1,j) ...
- Case 3: "to the right": ... (i,j), (i,j+1) ...
- Case 4: Repeated case 2 or 3: ... (i,j), (i,j+1), (i+1,j) ... OR ... (i,j), (i+1,j), (i,j+1) ... OR ...

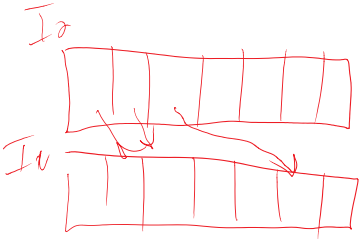


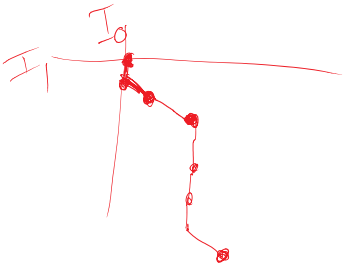




Observe: Case 4 simply consists of sequentially application of Case 2 and 3. Further note: all other "complicated but legal" correspondences are composed of sequences of cases 1, 2, and/or 3

Thus, *we need not search ALL* inferior neighbors (only three immediate inferior neighbors.)





Side Note: Horizontal and vertical moves are be interpreted as discontinuities when the uniqueness constraint is applied or monotonicity is strict, otherwise these moves may be interpreted as part of a multi-match that also incurs a discontinuity penalty.

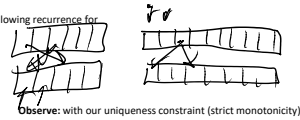
To use a dynamic programming solution, we assume that we can compute the total energy (or approximate) using an overlapping recurrence.

Assuming we want the best path, given our constrained problem, we have the following recurrence for cost of subsequence up to decision C(i,j)

$$C(i,j) = \begin{cases} C(i-1,j-1) + E_{data}, & \text{case 1} \\ C(i-1,j) + E_{smooth}, & \text{case 2} \\ C(i,j-1) + E_{smooth}, & \text{case 3} \end{cases}$$

$$C(0,j) = C(0,j-1) + E_{smooth}$$
$$C(i,0) = C(i-1,0) + E_{smooth}$$

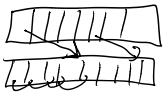
Assuming Uniqueness*
- select 1 of the 3 cases for the current cost selection



Observe: with our uniqueness constraint (strict monotonicity), we either have a match or discontinuity (not both), thus

We will never add the Match Cost (E_data) and Discontinuity Penalty (E_smooth) at the same (i,j).

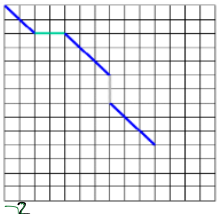
Further Observe: Given our previous definition of E_smooth, the resulting value was either 0 or 1. That is, it was 1 for horizontal or vertical moves and 0 otherwise. Thus, \lambda would be the penalty for discontinuity. We can simply view this as a discontinuity penalty: we either incur the penalty or not based on the case.



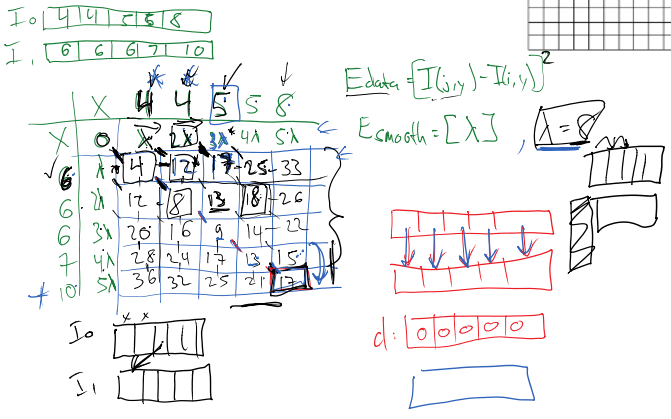
(Side note: In our previous example in class -- we solved the problem without the uniqueness constraint, thus allowing multi-matching. As a result the penalty and match cost were added for horizontal and vertical moves rather than only adding a penalty for such moves.)

Given our ordering constraint, we can pose this problem as a recurrence and solve for the "best" sequence (the sequence with minimum accrued error) in terms of the best (sub)sequences previously determined (based on the ordering constraint). *Thus, we need to keep a "backpointer" the best previous subsequence; this is analogous to identifying an inferior neighbor.*

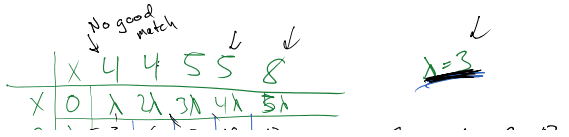
Some final notes: if we wish to incur a penalty for not mapping all pixels in *both* images, we should begin our reverse scan of backpointers from the lower right corner of the matrix. Otherwise, (if we expect the images are slightly offset) it may be a good idea to simply choose the min cost on the last row or last column. Note however, if the discontinuity penalty is high enough, we may be forced to pick the lower right corner (or near it) anyway.

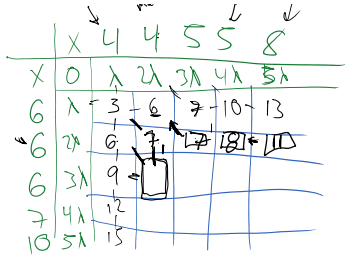


Example (HW #2)



Lets try this example again with a lower smoothness penalty.





$\lambda = 3$

Case 1: 6 + 17
Case 2: 7 + 3
Case 3: 7 + 3

Try this one ...

$I_0 = [0\ 1\ 2\ 3\ 4]$ $\lambda = 5$

$I_1 = [2\ 3\ 3\ 4\ 5]$