CSE 559A: Computer Vision

Fall 2018: T-R: 11:30-1pm @ Lopata 101
Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Zhihao Xia, Charlie Wu, Han Liu

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Oct 11, 2018

GENERAL

- Still missing a few problem set 2 submissions
  - Make sure you have git push-ed.
  - Do a git pull; git log and make sure the latest log message confirms your submission.
- Problem set 3 due two weeks from today.
- No Class Tuesday (Fall Break)
- No office hours tomorrow or Monday. Recitation next Friday.

GLOBAL OPTIMIZATION

Last Time

- We define a cost volume $C$ of size $W \times H \times D$
  - $C[x, y, d]$ measures dis-similarity between $(x, y)$ in left image and $(x - d, y)$ in right image
- Simplest Approach: $d[x, y] = \arg\min_d C[x, y, d]$
  - Too noisy
- Want to express that disparity (and therefore depth) of nearby pixels is similar
- Ad-hoc Method: Cost Volume Filtering
- Only encodes that nearby pixel disparities are exactly equal.

GLOBAL OPTIMIZATION

\[
d = \arg\min_d \sum_n C[n, d[n]] + \lambda \sum_{\{n, n'\} \in \mathbb{E}} S(d[n], d[n'])
\]

- $n = (x, y)^T$ for pixel location.
- $C$ is cost-volume as before. Gives us "local evidence"
- $\mathbb{E}$ is a set of all pairs of pixels that are "neighbors" / adjacent in some way.
  - Can include all un-ordered pairs of pixels with $\{(x, y), (x - 1, y)\}$ and $\{(x, y), (x, y - 1)\}$ (four connected)
  - Or diagonal neighbors as well.
- $S$ is a function that indicates a preference for $d[n]$ and $d[n']$ to be the same.
GLOBAL OPTIMIZATION

\[ d = \arg \min_d \sum_n C[n, d[n]] + \lambda \sum_{(n,n') \in \mathcal{R}} S(d[n], d[n']) \]

- \( S \) is a function that indicates a preference for \( d[n] \) and \( d[n'] \) to be the same.

- Choice 1: 
  0 if \( d[n'] = d[n] \), 1 otherwise.

- Choice 2: \(|d[n'] - d[n]|\)

- Choice 3: 
  0 if \( d[n'] = d[n] \)
  \( T_1 \) if \(|d[n'] - d[n]| < \epsilon \)
  \( T_2 \) otherwise.

How do we solve this?

Note that this is a discrete minimization. Each \( d[n] \in \{0, 1, \ldots, D-1\} \).

GLOBAL OPTIMIZATION

One approach: Iterated Conditional Modes

- Begin with \( d_0 = \arg \min_d C[n, d[n]] \)
- At each iteration \( t \), compute \( d_{t+1} \) from \( d_t \) by solving for each pixel in \( d_{t+1} \) assuming neighbors have values from \( d_t \).

\[ d_{t+1}[n] = \arg \min_{d} C[n, d[n]] + \lambda \sum_{(n,n') \in \mathcal{R}} S(d[n], d[n']) \]

- So for each pixel,
  - Take matching cost.
  - Add smoothness cost from its neighbors, assuming values from previous iteration.
  - Minimize.

Does it converge?

No Guarantee: We are changing all pixel assignments simultaneously.

GLOBAL OPTIMIZATION

- These kind of cost functions / optimization problems are quite common in vision.
- The cost can be interpreted as a log probability distribution:

\[ p(d) \propto \prod_n \exp(-C[n, d[n]]) \prod_{(n,n') \in \mathcal{R}} \exp(-\lambda S(d[n], d[n'])) \]

- Joint distribution over all the \( d[n] \) values.
GLOBAL OPTIMIZATION

Joint distribution over all the $d[n]$ values.

Graphical Model: Probability Distribution Represented as a "Graph" $(V, E)$

$$p(\{v \in V\}) = \prod_{v \in V} \Psi_v(v) \prod_{(v_1, v_2) \in E} \Phi_{v_1, v_2}(v_1, v_2)$$

Unary term for each node, pair-wise term for each edge.

(Directed Graphs represent Bayesian Networks)

Question: Are $d[n]$ and $d[n']$ independent if:

- If $(n, n') \notin E$ -- pixels are not neighbors? NO.
- If $(n, n') \notin E$ -- pixels are not neighbors? NO.
Question: Are \( d[n] \) and \( d[n'] \) independent if:

- If \((n,n') \notin E\), "conditioned" on all the neighbors of \( n \) being observed, \( p(d[n], d[n']|\{d[n'']\})\)

YES. This is the Markov property. And these kinds of graphical models are called Markov random fields.

Graph structure encodes "conditional independence".

\[
p(d) \propto \prod_n \exp(-C(n, d[n])) \prod_{(n,n') \in E} \exp(-\lambda S(d[n], d[n']))
\]

Compute assignment with highest probability

\[
d = \arg \max_d p(d) = \arg \min_d \sum_n C(n, d[n]) + \lambda \sum_{(n,n') \in E} S(d[n], d[n'])
\]

- Iterated Conditional Modes really slow.
- No guaranteed solution for arbitrary graphs.
- But could solve it our graph were a chain (or more generally a tree).

\[
d = \arg \min_d \sum_x C(x, d[x]) + \lambda \sum_x S(d[x], d[x+1])
\]

- Consider where we optimize each epipolar line separately.
You have costs stored for each individual allocation, as well as cost for edges

$$\sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x + 1])$$

The total cost of those blocks and edges was the least.

Say we only had two nodes:

$$d_1, d_2 = \arg \min_{d_1, d_2} C[1, d_1] + C[2, d_2] + \lambda S(d_1, d_2)$$

$$d_2 = \arg \min_{d_2} C[1, d_1] + C[2, d_2] + \lambda S(d_1, d_2)$$

This is the $d_2$ corresponding to the optimal path

$$\sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x + 1])$$

Say we only had two nodes:

$$d_1, d_2 = \arg \min_{d_1, d_2} C[1, d_1] + C[2, d_2] + \lambda S(d_1, d_2)$$

$$d_2 = \arg \min_{d_2} C[2, d_2] + \min_{d_1} (C[1, d_1] + \lambda S(d_1, d_2))$$

This is a function of $d_2$ or a table of values for each possible value of $d_1$

$$\sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x + 1])$$
\[
d_1, d_2, d_3 = \arg\min C[1, d_1] + C[2, d_2] + C[3, d_3] + \lambda S(d_1, d_2) + \lambda S(d_2, d_3)
\]
\[
d_3 = \arg\min_{d_3} C[3, d_2] + \min_{d_2} \left[ \lambda S(d_2, d_3) + C[2, d_2] + \min_{d_1} \left[ \lambda S(d_1, d_2) + C[1, d_1] \right] \right]
\]
This is precisely what we computed for the 2 node case.
Also note that once you have this, you don’t care about what the value of \(d\) was in the inner minimization.
\[
\sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x + 1])
\]
\[
d_1, d_2, d_3 = \arg\min C[1, d_1] + C[2, d_2] + C[3, d_3] + \lambda S(d_1, d_2) + \lambda S(d_2, d_3)
\]
\[
d_3 = \arg\min_{d_3} C[3, d_2] + \min_{d_2} \left[ \lambda S(d_2, d_3) + C[2, d_2] + \min_{d_1} \left[ \lambda S(d_1, d_2) + C[1, d_1] \right] \right]
\]
\[
\bar{C}[3, \cdot] = \bar{C}[2, \cdot]
\]
\[
\bar{C}[x + 1, d] = C[x + 1, d] + \min_{d'} \lambda S(d, d') + \bar{C}[x, d']
\]

---

We go from left to right, and doing an arg min on the last \(\bar{C}\) gives us the disparity of the last node.
And then we backtrack to find the full chain.

Store best \(d'\) for each \(d\).
\[
z[x + 1, d] = \arg\min_{d'} \lambda S(d, d') + \bar{C}[x, d']
\]
\[
\bar{C}[x + 1, d] = C[x + 1, d] + \min_{d'} \lambda S(d, d') + \bar{C}[x, d']
\]

---

Forward
\[
\bar{C}[0, d] = C[0, d]
\]
\[
z[x + 1, d] = \arg\min_{d'} \lambda S(d, d') + \bar{C}[x, d']
\]
\[
\bar{C}[x + 1, d] = C[x + 1, d] + \min_{d'} \lambda S(d, d') + \bar{C}[x, d']
\]

Backward
\[
d[x_{end}] = \arg\min_{d} \bar{C}[x_{end}, d]
\]
\[
d[x] = z[x + 1, d[x + 1]]
\]
GLOBAL OPTIMIZATION

We could apply this on individual epipolar lines.

That’s why we want to use a full 2D grid.
But forward-backward only works on chains (or graphs without cycles).
One flavor of approximate algorithms apply the same idea of forming a $\bar{C}[x, d]$

- TRW-S
- Loopy Belief Propagation
- SGM

GLOBAL OPTIMIZATION

Semi-Global Matching

$\bar{C}[x, d] = C[x, d] + \min_{d'} \bar{C}[x - 1, d'] + \lambda S(d, d')$

This is going left to right in the horizontal direction.

Idea: Compute different $\bar{C}$ along different directions ...

and average.

GLOBAL OPTIMIZATION

Semi-Global Matching

$\bar{C}_{lb}[n, d] = C[n, d] + \min_{d'} \bar{C}_{lb}[n - [1, 0]^T, d'] + \lambda S(d, d')$
$\bar{C}_{rd}[n, d] = C[n, d] + \min_{d'} \bar{C}_{rd}[n + [1, 0]^T, d'] + \lambda S(d, d')$
$\bar{C}_{rd}[n, d] = C[n, d] + \min_{d'} \bar{C}_{rd}[n - [0, 1]^T, d'] + \lambda S(d, d')$
$\bar{C}_{ul}[n, d] = C[n, d] + \min_{d'} \bar{C}_{ul}[n + [0, 1]^T, d'] + \lambda S(d, d')$

$d[n] = \arg \min_d \bar{C}_{lb}[n, d] + \bar{C}_{rd}[n, d] + \bar{C}_{ul}[n, d] + \bar{C}_{rd}[n, d]$
Consider the case when:

\[ S(d, d') = \begin{cases} 
0 & \text{if } d = d' \\
D_1 & \text{if } |d - d'| = 1 \\
D_2 & \text{otherwise.}
\end{cases} \]

Can we do this efficiently?

- Need to go through each line sequentially.
- But can go through all lines in parallel.
- But what about \( d' \)? Do we need to do minimization for every \( d' \) independently?

\[
\tilde{C}[x, d] = C[x, d] + \min_{d'} \tilde{C}[x - 1, d'] + \lambda S(d, d')
\]

Note: It doesn't matter if we add / subtract constants to all \( d' \)’s:

- \( C[x, d] \) with \( C[x, d] + C_0[x] \)
- \( \tilde{C}[x, d] \) with \( \tilde{C}[x, d] + C_0[x] \)

Why not?

- Because the minimization will always be over \( d \). You are never comparing \( C[x_1, d_1] \) with \( C[x_2, d_2] \).

Step 1 (Simplify): Replace \( \tilde{C}[x - 1, d'] \) with \( \tilde{C}[x - 1, d'] = \tilde{C}[x - 1, d'] + \min_{d'} \tilde{C}[x - 1, d''] \)

The MAXIMUM value for \( \min_{d'} \tilde{C}[x - 1, d'] + S(d, d') \) is \( P_2 \).

Step 2: This means that for every value of \( d \), we just need to consider four values.

\[
\min_{d'} \tilde{C}[x - 1, d'] + S(d, d')
\]

is the min of

- \( P_2 \) (for \( d' = \arg \min \tilde{C}[x - 1, d'] \))
- \( \tilde{C}[x - 1, d - 1] + P_1 \) (for \( d' = d - 1 \))
- \( \tilde{C}[x - 1, d + 1] + P_1 \) (for \( d' = d + 1 \))
- \( \tilde{C}[x - 1, d] \) (for \( d' = d \))

Full algorithm in paper:

\[
\tilde{C}[x, d] = C[x, d] + \min_{d'} \tilde{C}[x - 1, d'] + S(d, d')
\]
\[
S(d, d') = \begin{cases} 
0 & \text{if } d = d' \\
P_1 & \text{if } |d - d'| = 1 \\
P_2 & \text{otherwise}
\end{cases}
\]

- \(\min_{d'} \tilde{C}[x - 1, d'] + S(d, d')\) is the min of
  - \(P_2\) (for \(d' = \arg \min \tilde{C}[x - 1, d']\))
  - \(\tilde{C}[x - 1, d - 1] + P_1\) (for \(d' = d - 1\))
  - \(\tilde{C}[x - 1, d + 1] + P_1\) (for \(d' = d + 1\))
  - \(\tilde{C}[x - 1, d]\) (for \(d' = d\))

Can do this in parallel with matrix operations for all \(d\) and all lines.

SGM Algorithm Averages along four directions:

\[
\begin{align*}
\tilde{C}_b[n, d] &= C[n, d] + \min_{d'} \tilde{C}_b[n - [1, 0]^T, d'] + \lambda S(d, d') \\
\tilde{C}_l[n, d] &= C[n, d] + \min_{d'} \tilde{C}_l[n + [1, 0]^T, d'] + \lambda S(d, d') \\
\tilde{C}_{db}[n, d] &= C[n, d] + \min_{d'} \tilde{C}_{db}[n - [0, 1]^T, d'] + \lambda S(d, d') \\
\tilde{C}_{dl}[n, d] &= C[n, d] + \min_{d'} \tilde{C}_{dl}[n + [0, 1]^T, d'] + \lambda S(d, d')
\end{align*}
\]
\(d[n] = \arg \min_d \tilde{C}_b[n, d] + \tilde{C}_l[n, d] + \tilde{C}_{db}[n, d] + \tilde{C}_{dl}[n, d]\)

Bur \(\tilde{C}_{rl}\) is still smoothing the original cost.

Wouldn’t this be better?
But then ...

Wouldn’t this be better?
Why not this?
Because this is a circular definition.
Loopy Belief Propagation (one version)

\[
\bar{C}_{n}^{t+1}[n, d] = (C[n, d] + \bar{C}_{n}^{t}[n, d] + \bar{C}_{ad}^{t}[n, d] + \bar{C}_{db}^{t}[n, d]) + \min_{d'} \bar{C}_{n}^{t+1}[n - [1, 0]^T, d'] + \lambda \mathcal{S}(d, d')
\]

\[
\bar{C}_{db}^{t+1}[n, d] = (C[n, d] + \bar{C}_{db}^{t}[n, d] + \bar{C}_{ad}^{t}[n, d] + \bar{C}_{db}^{t}[n, d]) + \min_{d'} \bar{C}_{db}^{t+1}[n - [1, 0]^T, d'] + \lambda \mathcal{S}(d, d')
\]

\[
\bar{C}_{ad}^{t+1}[n, d] = (C[n, d] + \bar{C}_{ad}^{t}[n, d] + \bar{C}_{db}^{t}[n, d] + \bar{C}_{ad}^{t}[n, d]) + \min_{d'} \bar{C}_{ad}^{t+1}[n - [1, 0]^T, d'] + \lambda \mathcal{S}(d, d')
\]

Do this iteratively

More generally, at time step \( t \), pass a message from node \( n \) to \( n' \), based on all messages \( n \) has at that time, except for the message from \( n' \).

Read more:

- Other methods for discrete minimization---based on "Graph Cuts".
- SGM / Loopy BP: Generalize that there is an exact solution for a chain.
- Graph Cuts (with expansions / swaps): Generalize that there is an exact solution if only two values of \( d \).


http://vision.middlebury.edu/stereo/