Targil 9:
Optical Flow

Lecture Outline

- Defining the optical flow problem
- Patch matching (correlation)
- The aperture problem
- Optical flow equation (gradient descent on the error function)
- Lukas-Kanade flow
- Iterative refinement
- Course to Fine solution (pyramids)
- Multi-resolution Lucas Kanade Algorithm

Motion estimation: Optical flow

We have seen how to estimate the image of the entire image, now we want to estimating motion of each pixel separately

Problem definition: optical flow

How to estimate pixel motion from image H to image I?

Why is it useful
- Depth (3D) reconstruction
- Motion detection - tracking moving objects
- Compression.
- …..

Classes of Techniques

Feature-based methods
- Extract visual features (corners, textured areas) and track them over multiple frames
- Sparse motion fields, but possibly robust tracking
- Suitable especially when image motion is large (10s of pixels)

Direct-methods
- Directly recover image motion from spatio-temporal image brightness variations
- Motion vectors directly recovered without an intermediate feature motion calculation
- Dense motion fields, but more sensitive to appearance variations
- Suitable for video and when image motion is small (< 10 pixels)
Patch matching (revisited)

How do we determine correspondences?

- block matching or SSD (sum squared differences)

$$E(x, y; d) = \sum_{(x', y') \in N(x, y)} [I_1(x'+d, y') - I_2(x', y')]^2$$

Matching Criteria - Difference

Common matching criteria:

- SSD - Sum of Squared Differences
  $$d(p_1, p_2) = \sum_{j=0}^{k} (I_1(x_1 + i, y_1 + j) - I_2(x_1 + i, y_1 + j))^2$$
  where:
  - $k$ - the size of the window.
  - $p_1 = (x_1, y_1)$ is the center of the window in $I_1$.
  - $p_2 = (x_2, y_2)$ is the center of the window in $I_2$.

- SAD - Sum of Absolute Differences

$$d(p_1, p_2) = \sum_{j=0}^{k} |I_1(x_1 + i, y_1 + j) - I_2(x_1 + i, y_1 + j)|$$

Matching Criteria - Correlation

Max Cross Correlation is similar to Min SSD, but can be implemented more efficiently:

$$d(p_1, p_2) = \sum_{j=0}^{k} I_1(x_1 + i, y_1 + j) \cdot I_2(x_2 + i, y_2 + j)$$

Handling illumination changes

SSD, SAD and Cross Correlation assume constant brightness.

On order to handle illumination changes, Normalized cross-correlation can be used.

Normalized Cross Correlation

When ordering the pixels in the windows in vectors $v_1, v_2$:

- $\text{ssd}(\vec{v}_1, \vec{v}_2) = \sum_{i} |\vec{v}_1(i) - \vec{v}_2(i)|^2$  \hspace{1cm} $\text{Squared Vector norm}$
- $\text{corr}(\vec{v}_1, \vec{v}_2) = \sum_{i} \vec{v}_1(i) \cdot \vec{v}_2(i) = (\vec{v}_1, \vec{v}_2)$ \hspace{1cm} $\text{inner product}$

The Normalized Cross Correlation is:

$$\text{NorCorr}(\vec{v}_1, \vec{v}_2) = \frac{\sum_{i} \vec{v}_1(i) \cdot \vec{v}_2(i)}{\sqrt{\sum_{i} \vec{v}_1(i)^2} \sqrt{\sum_{i} \vec{v}_2(i)^2}}$$

Where we are subtracting the average of the patch from the vector

$$\vec{v}_1 = \vec{v}_1 - \bar{v}_1$$
$$\vec{v}_2 = \vec{v}_2 - \bar{v}_2$$
Aperture problem

Direct methods

gradient descent on the error function

Same assumption we used in finding global motion (image alignment)

Optical flow constraints (grayscale images)

Optical flow equation

Combining these two equations:

\[ 0 = I(x + u, y + v) - H(x, y), \]

shorthand: \( I_x = \frac{\partial I}{\partial x} \)

\[ \approx I(x, y) + I_u u + I_v v - H(x, y), \]

\[ \approx (I(x, y) - H(x, y)) + I_u u + I_v v, \]

\[ \approx I + I_u u + I_v v, \]

\[ \approx I + \nabla I : [u \ v] \]

In the limit as \( u \) and \( v \) go to zero, this becomes exact

\[ 0 = I + \nabla I : \left[ \begin{array}{c} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{array} \right] \]
Optical flow equation

\[ 0 = I_t + \nabla I \cdot [u \ v]. \]

Q: how many unknowns and equations per pixel?

Intuitively, what does this constraint mean?
- The component of the flow in the gradient direction is determined
- The component of the flow parallel to an edge is unknown

This explains the Barber Pole Illusion
http://www.sandlotscience.com/Ambiguous/barberpole.htm

Getting more Equations

How to get more equations for a pixel?
- Basic idea: Impose additional constraints
  - most common is to assume that the flow field is smooth locally
  - one method: pretend the pixel’s neighbors have the same (u,v)
  - if you use a 5x5 window, that gives us 25 equations per pixel

\[
\begin{align*}
0 &= I_t(p_1) + \nabla I(p_1) \cdot [u \ v] \\
0 &= I_t(p_2) + \nabla I(p_2) \cdot [u \ v] \\
0 &= I_t(p_{25}) + \nabla I(p_{25}) \cdot [u \ v] \\
\end{align*}
\]

\[
\begin{bmatrix}
I_x(p_1) & I_y(p_1) \\
I_x(p_2) & I_y(p_2) \\
I_x(p_{25}) & I_y(p_{25}) \\
\end{bmatrix}
\begin{bmatrix}
u \\
v \\
\end{bmatrix}
= 
\begin{bmatrix}
-I_t(p_1) \\
-I_t(p_2) \\
-I_t(p_{25}) \\
\end{bmatrix}
\]

\[
A \ d = b
\]

25x2 \times 1

25x1

RGB version

How to get more equations for a pixel?
- Basic idea: Impose additional constraints
  - most common is to assume that the flow field is smooth locally
  - one method: pretend the pixel’s neighbors have the same (u,v)
  - if you use a 5x5 window, that gives us 25/3 equations per pixel

\[
\begin{align*}
0 &= I_t(p_1)_{[0,1,2]} + \nabla I(p_1)_{[0,1,2]} \cdot [u \ v] \\
0 &= I_t(p_2)_{[0,1,2]} + \nabla I(p_2)_{[0,1,2]} \cdot [u \ v] \\
0 &= I_t(p_{25})_{[0,1,2]} + \nabla I(p_{25})_{[0,1,2]} \cdot [u \ v] \\
\end{align*}
\]

\[
\begin{bmatrix}
I_x(p_1)_{[0]} & I_y(p_1)_{[0]} \\
I_x(p_1)_{[1]} & I_y(p_1)_{[1]} \\
I_x(p_1)_{[2]} & I_y(p_1)_{[2]} \\
I_x(p_{25})_{[0]} & I_y(p_{25})_{[0]} \\
I_x(p_{25})_{[1]} & I_y(p_{25})_{[1]} \\
I_x(p_{25})_{[2]} & I_y(p_{25})_{[2]} \\
\end{bmatrix}
\begin{bmatrix}
u \\
v \\
\end{bmatrix}
= 
\begin{bmatrix}
-I_t(p_1)_{[0]} \\
-I_t(p_1)_{[1]} \\
-I_t(p_1)_{[2]} \\
-I_t(p_{25})_{[0]} \\
-I_t(p_{25})_{[1]} \\
-I_t(p_{25})_{[2]} \\
\end{bmatrix}
\]

\[
A \ d = b
\]

75x2 \times 1

75x1

Lukas-Kanade flow

Prob: we have more equations than unknowns

\[ A \ d = b \]

25x2 \times 1

25x1

Solution: solve least squares problem
- minimum least squares solution given by solution in \( d \) of:

\[
(A^T A) \ d = A^T b
\]

\[
\sum I_x I_x \sum I_y I_y \sum I_z I_z \\
\sum I_x I_y \sum I_y I_y \sum I_z I_z \\
\sum I_x I_z \sum I_y I_z \sum I_z I_z \\
\]

\[
A^T A \\
\]

\[
\]

\[
\sum I_x I_x \sum I_y I_y \sum I_z I_z \\
\sum I_x I_y \sum I_y I_y \sum I_z I_z \\
\sum I_x I_z \sum I_y I_z \sum I_z I_z \\
\]

\[
A^T b \\
\]

- The summations are over all pixels in the \( K \times K \) window
- This technique was first proposed by Lukas & Kanade (1981)

Conditions for solvability

- Optimal \( (u, v) \) satisfies Lucas-Kanade equation

\[
\sum I_x I_x \sum I_y I_y \sum I_z I_z \\
\sum I_x I_y \sum I_y I_y \sum I_z I_z \\
\sum I_x I_z \sum I_y I_z \sum I_z I_z \\
\]

\[
A^T A \\
\]

\[
A^T b \\
\]

When is This Solvable?
- \( A^T A \) should be invertible
- \( A^T A \) should not be too small due to noise
  - eigenvalues \( \lambda_1 \) and \( \lambda_2 \) of \( A^T A \) should not be too small
- \( A^T A \) should be well-conditioned
  - \( \lambda_1/\lambda_2 \) should not be too large (\( \lambda_1 \) = larger eigenvalue)
- \( A^T A \) is solvable when there is no aperture problem

\[
A^T A = \sum \sum I_x I_x \sum I_y I_y \sum I_z I_z = \sum \sum I_x I_y \sum I_y I_y \sum I_z I_z = \sum \sum I_x I_z \sum I_y I_z \sum I_z I_z \\
\]

\[
= \sum \sum \nabla I(\nabla I)^T \\
\]
Local Patch Analysis

Edge

Low texture region

High textured region

Observation

Errors in Lukas-Kanade

This is a two image problem BUT

• Can measure sensitivity by just looking at one of the images!
• This tells us which pixels are easy to track, which are hard
  – very useful later on when we do feature tracking...

What are the potential causes of errors in this procedure?

• Suppose $A^TA$ is easily invertible
• Suppose there is not much noise in the image

When our assumptions are violated

• Brightness constancy is not satisfied
• The motion is not small
• A point does not move like its neighbors
  – window size is too large
  – what is the ideal window size?
**Limits of the gradient method**

Fails when intensity structure in window is poor
Fails when the displacement is large (typical operating range is motion of 1 pixel)
Linearization of brightness is suitable only for small displacements
Also, brightness is not strictly constant in images
   actually less problematic than it appears, since we can pre-filter images to make them look similar

---

**Iterative Refinement**

**Iterative Lukas-Kanade Algorithm**
1. Estimate velocity at each pixel by solving Lucas-Kanade equations
2. Warp H towards I using the estimated flow field
   - use image warping techniques (easier said than done)
3. Repeat until convergence

---

**Optical Flow: Iterative Estimation**

Initial guess: $d_0 = 0$
Estimate: $d_1 = d_0 + \hat{d}$

(Using $\hat{d}$ for displacement here instead of $u$)

---

Initial guess: $d_1$
Estimate: $d_2 = d_1 + \hat{d}$

---

Initial guess: $d_2$
Estimate: $d_3 = d_2 + \hat{d}$

---

$\hat{d} = d_3$
Optical Flow: Iterative Estimation

Some Implementation Issues:
- Warping is not easy (ensure that errors in warping are smaller than the estimate refinement)
- Warp one image, take derivatives of the other so you don’t need to re-compute the gradient after each iteration.
- Often useful to low-pass filter the images before motion estimation (for better derivative estimation, and linear approximations to image intensity)

Optical Flow: Aliasing

Temporal aliasing causes ambiguities in optical flow because images can have many pixels with the same intensity.

i.e., how do we know which ‘correspondence’ is correct?

- nearest match is correct (no aliasing)
- nearest match is incorrect (aliasing)

To overcome aliasing: coarse-to-fine estimation.

Revisiting the small motion assumption

Is this motion small enough?
- Probably not—it’s much larger than one pixel (2nd order terms dominate)
- How might we solve this problem?

Reduce the resolution!

Coarse-to-fine optical flow estimation

- run iterative L-K
- warp & upsample

Coarse-to-fine optical flow estimation

- Gaussian pyramid of image H
- Gaussian pyramid of image I
- u=10 pixels
- u=5 pixels
- u=2.5 pixels
- u=1.25 pixels
- u=10 pixels
- u=5 pixels
- u=2.5 pixels
- u=1.25 pixels
Multi-resolution Lucas Kanade Algorithm

Compute iterative LK at highest level
• For each level i
  • Take flow u(i-1), v(i-1) from level i-1
    • Upsample the flow to create u'(i), v'(i) matrices of twice resolution for level i.
    • Multiply u'(i), v'(i) by 2
  • Compute I_t from a block displaced by u'(i), v'(i)
    • Apply LK to get u'(i), v'(i) (the correction on flow)
  • Add corrections u'(i), v'(i) to obtain the flow u(i), v(i) at ith level, i.e., u(i)=u(i)+u'(i), v(i)=v(i)+v'(i)

Optical Flow: Iterative Estimation

Some Implementation Issues:
• Warping is not easy (ensure that errors in warping are smaller than the estimate refinement)
• Warp one image, take derivatives of the other so you don’t need to re-compute the gradient after each iteration.
• Often useful to low-pass filter the images before motion estimation (for better derivative estimation, and linear approximations to image intensity)

Beyond Translation

So far, our patch can only translate in (u,v)
What about other motion models?
• rotation, affine, perspective

Same thing but need to add an appropriate Jacobian (see Table 2 in Szeliski handout):

\[ A^T = \sum_{i} J \nabla I (\nabla I)^T J^T \]

\[ A^T b = -\sum_{i} J^T I (\nabla I)^T \]

Optical Flow Results

Lucas-Kanade with Pyramids

Optical flow Results