Motion Estimation

http://www.sandlotscience.com/Distortions/Breathing_objects.htm

http://www.sandlotscience.com/Ambiguous/barberpole.htm

Today’s Readings
• Trucco & Verri, 8.3 – 8.4 (skip 8.3.3, read only top half of p. 199)
• Numerical Recipes (Newton-Raphson), 9.4 (first four pages)

http://www.sandlotscience.com/Distortions/Breathing_objects.htm
http://www.sandlotscience.com/Ambiguous/barberpole.htm

Why estimate motion?

Lots of uses
• Track object behavior
• Correct for camera jitter (stabilization)
• Align images (mosaics)
• 3D shape reconstruction
• Special effects

Optical flow

Problem definition: optical flow

How to estimate pixel motion from image H to image I?
• Solve pixel correspondence problem
  – given a pixel in H, look for nearby pixels of the same color in I

Key assumptions
• color constancy: a point in H looks the same in I
  – For grayscale images, this is brightness constancy
• small motion: points do not move very far

This is called the optical flow problem
Optical flow constraints (grayscale images)

Let’s look at these constraints more closely

- Brightness constancy: What’s the equation?

- Small motion: (u and v are less than 1 pixel)
  - Suppose we take the Taylor series expansion of \( I \):
  
  \[
  I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \text{higher order terms}
  \]

  \[
  \approx I(x, y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v
  \]

Optical flow equation

Combining these two equations

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
Aperture problem

Solving the aperture problem

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 we use a 5x5 window, that gives us 25 equations per pixel!

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

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 we use a 5x5 window, that gives us 25*3 equations per pixel!

\[
\begin{bmatrix}
I_x(p_1)[0] & I_x(p_1)[1] & I_x(p_1)[2] \\
I_y(p_1)[0] & I_y(p_1)[1] & I_y(p_1)[2] \\
I_x(p_{25})[0] & I_x(p_{25})[1] & I_x(p_{25})[2] \\
I_y(p_{25})[0] & I_y(p_{25})[1] & I_y(p_{25})[2]
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
= \begin{bmatrix}
I_x(p_1) \\
I_x(p_2) \\
\vdots \\
I_x(p_{25})
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
\]

Lukas-Kanade flow

Prob: we have more equations than unknowns

Solution: solve least squares problem

• minimum least squares solution given by solution (in d) of:

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

- The summations are over all pixels in the K x K window
- This technique was first proposed by Lukas & Kanade (1981)
  – described in Trucco & Vern reading
Conditions for solvability

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

\[
\begin{bmatrix}
\sum I_x I_x & \sum I_x I_y \\
\sum I_x I_y & \sum I_y I_y
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
=
-
\begin{bmatrix}
\sum I_x I_y \\
\sum I_y I_y
\end{bmatrix}

\begin{bmatrix}
u \\
v
\end{bmatrix}

ATA

ATA should be invertible

ATA should not be too small due to noise
- eigenvalues \(\lambda_1\) and \(\lambda_2\) of ATA should not be too small

ATA should be well-conditioned
- \(\lambda_1/\lambda_2\) should not be too large (\(\lambda_1\) = larger eigenvalue)

When is This Solvable?

- \(ATA\) should be invertible
- \(ATA\) should not be too small due to noise
- eigenvalues \(\lambda_1\) and \(\lambda_2\) of \(ATA\) should not be too small
- \(ATA\) should be well-conditioned
- \(\lambda_1/\lambda_2\) should not be too large (\(\lambda_1\) = larger eigenvalue)

Eigenvectors of \(ATA\)

\[
ATA = \begin{bmatrix}
\sum I_x I_x & \sum I_x I_y \\
\sum I_x I_y & \sum I_y I_y
\end{bmatrix}
= \begin{bmatrix}
I_x & I_y \\
I_y & I_y
\end{bmatrix}
= \nabla I(\nabla I)^T
\]

\(ATA\) should be invertible
- gradients along edge all point the same direction
- gradients away from edge have small magnitude

\[
\sum \nabla I(\nabla I)^T \approx k \nabla I \nabla I^T
\]

\(\sum \nabla I(\nabla I)^T \nabla I = k \|\nabla I\|^2 \nabla I\)

- \(\nabla I\) is an eigenvector with eigenvalue \(k \|\nabla I\|^2\)
- What's the other eigenvector of \(ATA\)?
  - let \(N\) be perpendicular to \(\nabla I\)
  - \(\sum \nabla I(\nabla I)^T \nabla I = C\)
  - \(N\) is the second eigenvector with eigenvalue 0

The eigenvectors of \(ATA\) relate to edge direction and magnitude

Edge

- large gradients, all the same
- large \(\lambda_1\), small \(\lambda_2\)

Low texture region

- gradients have small magnitude
- small \(\lambda_1\), small \(\lambda_2\)
High textured region

\[ \sum \nabla I(\nabla I)^T \]
- gradients are different, large magnitudes
- large \( \lambda_1 \), large \( \lambda_2 \)

Observation

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...

Errors in Lukas-Kanade

What are the potential causes of errors in this procedure?
- Suppose \( A^T A \) 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?

Improving accuracy

Recall our small motion assumption

\[ 0 \approx I(x, y) + I_x u + I_y v - I(x', y') \]

This is not exact
- To do better, we need to add higher order terms back in:
  \[ = I(x, y) + I_x u + I_y v + \text{higher order terms} - H(x, y') \]

This is a polynomial root finding problem
- Can solve using Newton’s method
  - Also known as Newton-Raphson method
- Today’s reading (first four pages)
- Approach so far does one iteration of Newton’s method
  - Better results are obtained via more iterations
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
3. Repeat until convergence

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

Gaussian pyramid of image H

Gaussian pyramid of image I
Coarse-to-fine optical flow estimation

Optical flow result

Motion tracking

Suppose we have more than two images
- How to track a point through all of the images?
  - In principle, we could estimate motion between each pair of consecutive frames
  - Given point in first frame, follow arrows to trace out it’s path
  - Problem: DRIFT
    » small errors will tend to grow and grow over time—the point will drift way off course

Feature Tracking
- Choose only the points ("features") that are easily tracked
- How to find these features?
  - windows where $\sum \nabla^2 (\nabla^2 f)^2$ has large eigenvalues
  - Called the Harris Corner Detector

Feature Detection
Tracking features

Feature tracking
- Compute optical flow for that feature for each consecutive H, I

When will this go wrong?
- Occlusions—feature may disappear
  - need mechanism for deleting, adding new features
- Changes in shape, orientation
  - allow the feature to deform
- Changes in color
- Large motions
  - will pyramid techniques work for feature tracking?

Handling large motions

L-K requires small motion
- If the motion is much more than a pixel, use discrete search instead

\[ \min_{(u,v)} \sum_{(x,y) \in W} (I(x+u, y+v) - H(x, y))^2 \]
- Given feature window W in H, find best matching window in I
- Minimize sum squared difference (SSD) of pixels in window
- Solve by doing a search over a specified range of (u,v) values
  - this (u,v) range defines the search window

Tracking Over Many Frames

Feature tracking with m frames
1. Select features in first frame
2. Given feature in frame i, compute position in i+1
3. Select more features if needed
4. i = i + 1
5. If i < m, go to step 2

Issues
- Discrete search vs. Lucas Kanade?
  - depends on expected magnitude of motion
  - discrete search is more flexible
- Compare feature in frame i to i+1 or frame 1 to i+1?
  - affects tendency to drift.
- How big should search window be?
  - too small: lost features. Too large: slow

Incorporating Dynamics

Idea
- Can get better performance if we know something about the way points move
- Most approaches assume constant velocity
  \[ x_{i+1} = x_i \]
  \[ x_{i+1} = 2x_i - x_{i-1} \]
  or constant acceleration
  \[ \ddot{x}_{i+1} = \ddot{x}_i \]
  \[ x_{i+1} = 3x_i - 3x_{i-1} + x_{i-2} \]
- Use above to predict position in next frame, initialize search
Feature tracking demo

Oxford video

MPEG—application of feature tracking
  • http://www.pixelfools.com/pixweb2.html

Image alignment

Goal: estimate single \((u,v)\) translation for entire image
  • Easier subcase: solvable by pyramid-based Lukas-Kanade

Application: Rotoscoping (demo)