Computer Vision
Object and People Tracking
Feature Tracking and Optical Flow

Prof. Didier Stricker
Kaiserlautern University
http://ags.cs.uni-kl.de/
DFKI – Deutsches Forschungszentrum für Künstliche Intelligenz
http://av.dfki.de
Outline of the lecture

**Feature-tracking**
- Extract visual features (corners, textured areas) and “track” them over multiple frames

**Optical flow**
- Recover image motion at each pixel from spatio-temporal image brightness variations (optical flow)

→ Two problems, one registration method

Some examples

Video examples (external viewer)
Feature tracking

- Many problems, such as object tracking or structure from motion require tracking points

- If motion is small, tracking is an easy way to get them
Feature tracking

Challenges

- Figure out which features can be tracked
- Efficiently track across frames
- Some points may change appearance over time (e.g., due to rotation, moving into shadows, etc.)
- Drift: small errors can accumulate as appearance model is updated
- Points may appear or disappear: need to be able to add/delete tracked points
Three assumptions

- Brightness consistency
- Spatial coherence
- Temporal persistence
Brightness consistency

Image measurement (e.g. brightness) in a small region remain the same although their location may change.
Spatial coherence

- Neighboring points in the scene typically belong to the same surface and hence typically have similar motions.
- Since they also project to nearby pixels in the image, we expect spatial coherence in image flow.
Temporal persistence

- The image motion of a surface patch changes gradually over time.
Feature tracking

Given two subsequent frames, estimate the point translation

- **Key assumptions of Kanade-Lucas Tracker (KLT)**
  - **Brightness constancy**: projection of the same point looks the same in every frame
  - **Temporal persistence ("small motions")**: points do not move very far
  - **Spatial coherence**: points move like their neighbors
The brightness constancy constraint

\[ I(x, y, t) = I(x + u, y + v, t + 1) \]

- Take Taylor expansion of \( I(x + u, y + v, t + 1) \) at \((x, y, t)\) to linearize the right side: 

\[
I(x + u, y + v, t + 1) \approx I(x, y, t) + I_x \cdot u + I_y \cdot v + I_t
\]

\[
I(x + u, y + v, t + 1) - I(x, y, t) = I_x \cdot u + I_y \cdot v + I_t
\]

Hence, \( I_x \cdot u + I_y \cdot v + I_t \approx 0 \to \nabla I \cdot [u \ v]^T + I_t = 0 \)
The brightness constancy constraint

- Can we use this equation to recover image motion \((u, v)\) at each pixel?
  \[
  \nabla I \cdot [u \ v]^T + I_t = 0
  \]

- How many equations and unknowns per pixel?
  - One equation (this is a scalar equation!), two unknowns \((u, v)\)
  - The component of the motion perpendicular to the gradient (i.e., parallel to the edge) cannot be measured
  - If \((u, v)\) satisfies the equation, so does \((u + u', v + v')\) if
  \[
  \nabla I \cdot [u' \ v']^T = 0
  \]
The aperture problem

Actual motion
The aperture problem

“In the absence of additional information the visual system prefers the slowest possible motion: i.e., motion orthogonal to the moving line”

http://en.wikipedia.org/wiki/Barberpole_illusion
How to get more equations for a pixel?

- Spatial coherence constraint
- Assume the pixel’s neighbors have the same \((u, v)\)
- If we use a 5x5 window, that gives us 25 equations per pixel

\[
0 = I_t(p_i) + \nabla I(p_i) \cdot [u \ v]
\]

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

**How to get more equations for a pixel?**

- Basic idea: impose additional constraints
- If we use a 5x5 window, that gives us 25*3 equations per pixel!

\[ 0 = I_t(p_i)[0,1,2] + \nabla I(p_i)[0,1,2] \cdot [u\ v] \]

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

\[ A \quad 75x2 \quad d \quad 2x1 \quad b \quad 75x1 \]

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
Lucas-Kanade flow

- Problem: we have more equations than unknowns
  \[ A \ d = b \]
  \[
  \begin{align*}
  &25 \times 2 \quad 2 \times 1 \\
  &25 \times 1
  \end{align*}
  \]
  minimize \( \|Ad - b\|^2 \)

- Solution: solve least squares problem

- Recall: least squares solution:
  \[ Ax = b \]
  \[
  \begin{align*}
  &\min((Ax - b)^T (Ax - b)) \\
  \text{Set derivative to zero:} & \quad 2A^T Ax - 2A^T b = 0 \\
  & \quad x = (A^T A)^{-1} A^T b
  \end{align*}
  \]
  Also called pseudo-inverse

Matlab operator:
  \[ x = A\backslash b \]
Lucas-Kanade flow

- Problem: we have more equations than unknowns
  \[ A \begin{bmatrix} d \end{bmatrix} = b \]
  with dimensions 25x2 2x1 25x1
  \[ \longrightarrow \]
  minimize \( \|Ad - b\|^2 \)

- Solution: solve least squares problem
  - Minimum least squares solution given by solution (in \( d \)) of:
    \[ \begin{bmatrix} A^T A \end{bmatrix} \begin{bmatrix} d \end{bmatrix} = A^T b \]
    \[ \begin{bmatrix} 2 \times 2 \end{bmatrix} \begin{bmatrix} 2 \times 1 \end{bmatrix} \begin{bmatrix} 2 \times 1 \end{bmatrix} \]

\[ \begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_xI_y & \sum I_yI_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_xI_t \\ \sum I_yI_t \end{bmatrix} \]

\[ A^T A \]
\[ A^T b \]

- The summations are over all pixels in the K x K window
- This technique was first proposed by Lucas & Kanade (1981)
Conditions for solvability

- Optimal \((u, v)\) satisfies Lucas-Kanade equation:
  \[
  \begin{bmatrix}
  \sum I_x I_x & \sum I_x I_y \\
  \sum I_x I_y & \sum I_y I_y
  \end{bmatrix}
  \begin{bmatrix}
  u \\
  v
  \end{bmatrix}
  = -
  \begin{bmatrix}
  \sum I_x I_t \\
  \sum I_y I_t
  \end{bmatrix}
  \]
  \[
  A^T A
  \]
  \[
  A^T b
  \]

- When is this solvable? I.e., what are good points to track?
  - \(A^T A\) 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 (\(\lambda_1 = \text{larger eigenvalue}\))

- Does this remind you of anything?
  - \(\rightarrow\) Criteria for corner detector (see also: Harris)
Edge

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

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

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
Low texture region

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

- gradients have small magnitude
- small \( \lambda_1 \), small \( \lambda_2 \)

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
High textured region

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

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
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 and which are hard
  - Very useful later on when we do feature tracking...

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
The aperture problem resolved
The aperture problem resolved
Errors in Lucas-Kanade

- 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?

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
Revisiting the small motion assumption

- Is this motion small enough?
  - Probably not—it’s much larger than one pixel ($2^{nd}$ order terms dominate)
  - How might we solve this problem?
Reduce the resolution!
Coarse-to-fine optical flow estimation

Gaussian pyramid of image $I_{t-1}$

- $u=10$ pixels
- $u=5$ pixels
- $u=2.5$ pixels
- $u=1.25$ pixels

Gaussian pyramid of image $I_t$

- $u=1.25$ pixels
- $u=2.5$ pixels
- $u=5$ pixels
- $u=10$ pixels
Dealing with larger movements: coarse-to-fine registration

Gaussian pyramid of image $I_t$

run iterative L-K

upsample

run iterative L-K

Gaussian pyramid of image $I_{t+1}$
Dealing with larger movements: Iterative refinement

1. Initialize \((x', y') = (x, y)\)
2. Compute \((u, v)\) by
   \[
   \begin{bmatrix}
   \sum I_x I_x & \sum I_x I_y \\
   \sum I_x I_y & \sum I_y I_y
   \end{bmatrix}
   \begin{bmatrix}
   u \\
   v
   \end{bmatrix}
   = -
   \begin{bmatrix}
   \sum I_x I_t \\
   \sum I_y I_t
   \end{bmatrix}
   \]
   2\textsuperscript{nd} moment matrix for feature patch in first image
3. Shift window by \((u, v)\): \(x' := x' + u; y' := y' + v\);
4. Recalculate \(I_t\)
5. Repeat steps 2-4 until small change
   - Use interpolation for subpixel values

Original \((x, y)\) position
\[
I_t = I(x', y', t + 1) - I(x, y, t)
\]
Shi-Tomasi feature tracker

- Find good features using eigenvalues of second-moment matrix (e.g., Harris detector or threshold on the smallest eigenvalue)
  - Key idea: “good” features to track are the ones whose motion can be estimated reliably

- Track from frame to frame with Lucas-Kanade
  - This amounts to assuming a translation model for frame-to-frame feature movement

- Check consistency of tracks by affine registration to the first observed instance of the feature
  - Affine model is more accurate for larger displacements
  - Comparing to the first frame helps to minimize drift

Tracking example

Figure 1: Three frame details from Woody Allen’s *Manhattan*. The details are from the 1st, 11th, and 21st frames of a subsequence from the movie.

Figure 2: The traffic sign windows from frames 1, 6, 11, 16, 21 as tracked (top), and warped by the computed deformation matrices (bottom).

Summary of KLT tracking

- Find a good point to track (harris corner)
- Use intensity second moment matrix and difference across frames to find displacement
- Iterate and use coarse-to-fine search to deal with larger movements
- When creating long tracks, check appearance of registered patch against appearance of initial patch to find points that have drifted
Implementation issues

Window size

- Small window more sensitive to noise and may miss larger motions (without pyramid)
- Large window more likely to cross an occlusion boundary (and it’s slower)
- Size of 15x15 to 31x31 seems typical

Weighting the window

- Common to apply weights so that center matters more (e.g., with Gaussian)
Tracking over many frames

- Select features in first frame

- For each frame:
  - Update positions of tracked features
    - Discrete search (e.g. cross-correlation) or Lucas-Kanade (or a combination of the two)
  - Terminate inconsistent tracks
    - Compute similarity with corresponding feature in the previous frame or in the first frame where it’s visible
  - Find more features to track
Optical flow

Vector field function of the spatio-temporal image brightness variations

Picture courtesy of Selim Temizer – Learning and Intelligent Systems (LIS) Group, MIT
Uses of motion

- Estimating 3D structure
- Segmenting objects based on motion cues
- Learning and tracking dynamical models
- Recognizing events and activities
- Improving video quality (motion stabilization)
Motion field

- The **motion field** is the projection of the 3D scene motion into the image.
Definition: optical flow is the *apparent* motion of brightness patterns in the image.

Ideally, optical flow would be the same as the motion field.

Have to be careful: apparent motion can be caused by lighting changes without any actual motion.

- Think of a uniform rotating sphere under fixed lighting vs. a stationary sphere under moving illumination.
Lucas-Kanade Optical Flow

- Same as Lucas-Kanade feature tracking, but for each pixel
  - As we saw, works better for textured pixels

- Operations can be done one frame at a time, rather than pixel by pixel
  - Efficient
Iterative Refinement

- **Iterative Lucas-Kanade Algorithm**
  1. Estimate displacement at each pixel by solving Lucas-Kanade equations
  2. Warp $I(t)$ towards $I(t+1)$ using the estimated flow field
     - Basically, just interpolation
  3. Repeat until convergence

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
Coarse-to-fine optical flow estimation

Gaussian pyramid of image $I_t$

run iterative L-K

warp & upsample

... 

run iterative L-K

Gaussian pyramid of image $I_{t+1}$
Example
Multi-resolution registration

* From Khurram Hassan-Shafique  
CAP5415 Computer Vision 2003
Optical Flow Results

Lucas-Kanade without pyramids
Fails in areas of large motion

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
Optical Flow Results

Lucas-Kanade with Pyramids

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003
Summary

- Major contributions from Kanade, Lucas, Tomasi
- We refer to it as “KLT-Tracker”
  - Tracking feature points
  - Optical flow
  - Stereo
  - Structure from motion

- Key ideas
  - By assuming brightness constancy, truncated Taylor expansion leads to simple and fast patch matching across frames
  - Coarse-to-fine registration
Thank you!