Lecture 13
Optical Flow and Tracking

COS 429: Computer Vision

Slides credit:
Many slides adapted from Michael Black, James Hays, Derek Hoeim, Lana Lazebnik, Silvio Saverse, Steve Seitz, Rick Szeliski, Martial Hebert, Mark Pollefeys, and others
Recall: Feature Matching

1. Find a set of distinctive keypoints
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors
Optical Flow and Tracking

https://youtu.be/GIUDAZLfYhY?t=63
https://youtu.be/jg6Nz6BfoSQ
Illusory Snakes
Motion field = 2D motion field representing the projection of the 3D motion of points in the scene onto the image plane
Optical Flow Field

\[ u(x, y) \] Horizontal component
\[ v(x, y) \] Vertical component

Optical flow = 2D velocity field describing the apparent motion in the images.
Assumption

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

\[ I(x + u, y + v, t + 1) = I(x, y, t) \]
Assumption
* Neighboring points in the scene typically belong to the same surface and hence typically have similar motions.
* Since they also project to nearby points in the image, we expect spatial coherence in image flow.

=> this allows us to assume a small patch will move together:
Distance Metric

• Goal: find \( \mathbf{B} \) in image, assume translation only: no scale change or rotation, using search (scanning the image)

• What is a good similarity or distance measure between two patches?
  – Correlation
  – Zero-mean correlation
  – Sum Square Difference
  – Normalized Cross Correlation
Matching with filters

Goal: find \( \begin{align*} &\text{in image} \\
&\text{Method 0: filter the image with eye patch} \\
&\quad h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l] \\
&f = \text{image} \quad g = \text{filter} \\
&\text{What went wrong?} \\
&\text{response is stronger for higher intensity} \\
\end{align*} \)
0-mean filter

- Goal: find image
- Method 1: filter the image with zero-mean eye

$$h[m, n] = \sum_{k, l} (f[k, l] - \bar{f}) (g[m + k, n + l])$$

mean of $f$
Sum of Squared error (L2)

- **Goal:** find an eye in the image.
- **Method 2:** SSD
  
  $$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2$$

---

Input | 1- sqrt(SSD) | Thresholded Image
--- | --- | ---

**True detections**
Sum of Squared error (L2)

- **Goal**: find 🔮 in image
- **Method 2: SSD**
  \[
  h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2
  \]

One potential downside of SSD:

**Brightness Constancy Assumption**
Normalized Cross-Correlation

- Goal: find in image
- Method 3: Normalized cross-correlation
  (= angle between zero-mean vectors)

\[
h[m, n] = \frac{\sum_{k,l} (g[k,l] - \bar{g})(f[m-k,n-l] - \bar{f}_{m,n})}{\sqrt{\sum_{k,l} (g[k,l] - \bar{g})^2 \sum_{k,l} (f[m-k,n-l] - \bar{f}_{m,n})^2}}
\]

Matlab: \texttt{normxcorr2(template, im)}
Normalized Cross-Correlation

- Goal: find 🌍 in image
- Method 3: Normalized cross-correlation

![Input](image1)

![Normalized X-Correlation](image2)

![Thresholded Image](image3)
Normalized Cross-Correlation

• Goal: find $\_\_\_\_$ in image
• Method 3: Normalized cross-correlation
Search vs. Gradient Descent

• Search:
  – Pros: Free choice of representation, distance metric; no need for good initial guess
  – Cons: expensive when searching for sub-pixel accuracy or over complex motion models (scale, rotation, affine)

• If we have a good guess, can we do something cheaper?
  – Gradient Descent
Lucas-Kanade Object Tracker

• **Key assumptions:**
  
  • **Brightness constancy:** projection of the same point looks the same in every frame (uses SSD as metric)
  
  • **Small motion:** points do not move very far (from guessed location)
  
  • **Spatial coherence:** points move in some coherent way (according to some parametric motion model)
    
    • For this example, assume whole object just translates in (u,v)
The brightness constancy constraint

• Brightness Constancy Equation:
  \[ I(x, y, t) = I(x + u, y + v, t + 1) \]

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,

\[
I_x \cdot u + I_y \cdot v + I_t \approx 0 \quad \rightarrow \nabla I \cdot [u \ v]^T + I_t = 0
\]
How does this make sense?

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

- What do the static image gradients have to do with motion estimation?
Tracking in the 1D case:

\[ I(x,t) \quad I(x,t + 1) \]
Tracking in the 1D case:

Assumptions:
- Brightness constancy
- Small motion

$$I_x = \frac{\partial I}{\partial x} \bigg|_t$$

$$I_t = \frac{\partial I}{\partial t} \bigg|_{x=p}$$

$$\vec{v} \approx -\frac{I_t}{I_x}$$
Tracking in the 1D case:

Iterating helps refining the velocity vector

\[ I(x, t) \quad I(x, t + 1) \]

Temporal derivative at 2\textsuperscript{nd} iteration

Can keep the same estimate for spatial derivative

\[ \vec{v} \leftarrow \vec{v}_{\text{previous}} - \frac{I_t}{I_x} \]

Converges in about 5 iterations
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

\[ \nabla I \cdot [u' \ v']^T = 0 \]

- Spatial coherence constraint: solve for many pixels and assume they all have the same motion
- In our case, if the object fits in a 5x5 pixel patch, this gives us 25 equations:

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

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

- Least squares problem:

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

\[
A \quad d = b
\]

\[
25 \times 2 \quad 2 \times 1 \quad 25 \times 1
\]
Solving for free parameters \((u,v)\)

- Over-constrained linear system

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

\[
A d = b
\]

Least squares solution for \(d\) given by

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

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

The summations are over all pixels in the \(K \times K\) window.
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. Repeat steps 2-4 until small change

• Use interpolation to warp by subpixel values

\[
I_t = I(x', y', t+1) - I(x, y, t)
\]

Original \((x, y)\) position

Shift window by \((u, v)\):

\[
x' = x' + u; \quad y' = y' + v
\]
Schematic of Lucas-Kanade
Dealing with larger movements

- How to deal with cases where the initial guess is not within a few pixels of the solution?
Dealing with larger movements: coarse-to-fine registration

- Gaussian pyramid of image 1 (t)
- Gaussian pyramid of image 2 (t+1)
- Run iterative L-K
- Upsample
- Run iterative L-K

Slide Credit: Image I

Image J
Coarse-to-fine optical flow estimation

Gaussian pyramid of image 1

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

Gaussian pyramid of image 2

- $u=10$ pixels
Coarse-to-fine optical flow estimation
Coarse-to-fine optical flow estimation

Downscale image and rectangle

Downscale image and rectangle
Coarse-to-fine optical flow estimation
Coarse-to-fine optical flow estimation

- Track Iter 1
- Track Iter 2
- Track Iter 3

Upscale rect
Coarse-to-fine optical flow estimation

Track Iter 1
Track Iter 2
Track Iter 3

Upscale rect
Feature Tracking

- Similar to feature matching, but track instead of match:
  - Track small, good features using translation only \((u,v)\)
  - Use RANSAC to solve more complex motion model
    (Scale, Rotation, Similarity, Affine, Homography, ... Articulated, non-rigid)
The barber pole illusion

http://en.wikipedia.org/wiki/Barberpole_illusion
The aperture problem

Perceived motion
The aperture problem
Edges cause problems

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

- large gradients, all the same
- large \( \lambda_1 \), small \( \lambda_2 \)
Low texture regions don’t work

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

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

$$\sum \nabla I(\nabla I)^T$$
- gradients are different, large magnitudes
- large $\lambda_1$, large $\lambda_2$
Conditions for solvability

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

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

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 (\(\lambda_1 =\) larger eigenvalue)

Recall: This is the Harris Corner Detector!
Feature Point tracking

• Find a good point to track (harris corner)
• Track small patches (5x5 to 31x31) (e.g. using Lucas-Kanade)
• For rigid objects with affine motion: solve motion model parameters by robust estimation (RANSAC)
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)
  – 15x15 to 31x31 seems typical

• Weighting the window
  – Common to apply weights so that center matters more (e.g., with Gaussian)
Dense Motion field

- The motion field is the projection of the 3D scene motion into the image
Lucas-Kanade Optical Flow

• Same as Lucas-Kanade feature tracking, but densely 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
Example
Multi-resolution registration
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
Object Detection and Tracking
Once target has been located, and we “learn” what it looks like, should be easier to find in later frames... this is object tracking.
Approaches to Object Tracking

- Motion model (translation, translation+scale, affine, non-rigid, ...)
- Image representation (gray/color pixel, edge image, SIFT, HOG, wavelet...)
- Distance metric (L1, L2, normalized correlation, Chi-Squared, ...)
- Method of optimization (gradient descent, naive search, combinatoric search...)
- What is tracked: whole object or selected features
Approaches to Object Tracking

Can we use Lukas-Kanade?

Yes, but need to deal with more than translation \((u,v)\)

\[ \Rightarrow \text{Affine Motion} \]
Affine Motion

\[ E(a) = \sum_{x,y \in R} (\nabla I^T u(x; a) + I_1)^2 \]

\[ u(x; a) = \begin{bmatrix} u(x; a) \\ v(x; a) \end{bmatrix} = \begin{bmatrix} a_1 + a_2 x + a_3 y \\ a_4 + a_5 x + a_6 y \end{bmatrix} \]
Affine Motion

\[
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}^* = \begin{bmatrix}
a_1 & a_2 \\ a_4 & a_5
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix} + \begin{bmatrix}
a_3 \\ a_6
\end{bmatrix}
\]

Linear transformation

Homogeneous coordinates

Translation
Affine Motion Optimization

\[ E(a) = \sum_{x, y \in R} (I_x u + I_y v + I_t)^2 \]

\[ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \]

\[ E(a) = \sum_{x, y \in R} (I_x a_1 x + I_x a_2 y + I_x a_3 + I_y a_4 x + I_y a_5 y + I_y a_6 + I_t)^2 \]
Recall: Solving for free parameters \((u,v)\)

- Over-constrained linear system

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

Least squares solution for \(d\) given by

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

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

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

The summations are over all pixels in the \(K \times K\) window
Affine Motion Optimization

\[ E(\mathbf{a}) = \sum_{x,y \in \mathbb{R}} (I_x a_1 x + I_x a_2 y + I_x a_3 + I_y a_4 x + I_y a_5 y + I_y a_6 + I_t)^2 \]

Differentiate wrt the \( a_i \) and set equal to zero.

\[
\begin{bmatrix}
\Sigma I_x^2 x^2 & \Sigma I_x^2 xy & \Sigma I_x^2 x & \Sigma I_x I_y x^2 & \Sigma I_x I_y xy & \Sigma I_x I_y x \\
\Sigma I_x^2 xy & \Sigma I_x^2 y^2 & \Sigma I_x^2 y & \Sigma I_x I_y xy & \Sigma I_x I_y y^2 & \Sigma I_x I_y y \\
\Sigma I_x^2 y & \Sigma I_x^2 y & \Sigma I_x^2 y & \Sigma I_x I_y xy & \Sigma I_x I_y y & \Sigma I_x I_y y \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
a_3 \\
a_4 \\
a_5 \\
a_6 \\
\end{bmatrix}
= 
\begin{bmatrix}
-\Sigma I_x I_t x \\
-\Sigma I_x I_t y \\
-\Sigma I_x I_t \\
-\Sigma I_y I_t x \\
-\Sigma I_y I_t y \\
-\Sigma I_y I_t \\
\end{bmatrix}
\]
Summary

• L-K works well when:
  – Have a good initial guess
  – L2 (SSD) is a good metric
  – Can handle more degrees of freedom in motion model (scale, rotation, affine, etc.), which are too expensive for search

• But has problems with:
  – Changes in brightness
  – ...

LK Problem: Change in Brightness

Possible Solutions:
- Subtract mean intensity (based on current estimate before iteration)
- Transform gray values into some features that are not affected by brightness
  - Any filter that is zero-mean
  - Example: vertical, horizontal edge filters
  - Example: Non-parametric filters (Rank, Census Transforms)
More Problems

- Outliers: bright strong features that are wrong
- Complex, high dimensional, or non-rigid motion