Automatic Image Alignment + Optical Flow

with a lot of slides stolen from Steve Seitz and Rick Szeliski

CS194: Intro to Comp. Vision and Comp. Photo
Alexei Efros, UC Berkeley, Fall 2022
Feature descriptors

We know how to detect points
Next question: How to match them?

Point descriptor should be:
1. Invariant
2. Distinctive
Feature Descriptor – MOPS
Multi-Scale Oriented Patches

Interest points

• Multi-scale Harris corners
• Orientation from blurred gradient
• Geometrically invariant to rotation

Descriptor vector

• Bias/gain normalized sampling of local patch (8x8)
• Photometrically invariant to affine changes in intensity

[Brown, Szeliski, Winder, CVPR’2005]
Detect Features, setup Frame

Orientation = blurred gradient

Rotation Invariant Frame

• Scale-space position \((x, y, s)\) + orientation \((\theta)\)
Detections at multiple scales

Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.
MOPS descriptor vector

8x8 oriented patch
- Sampled at 5 x scale

Bias/gain normalisation: $l' = (l - \mu)/\sigma$
Automatic Feature Matching
Feature matching
Feature matching

• Pick best match!
  • For every patch in image 1, find the most similar patch (e.g. by SSD).
  • Called “nearest neighbor” in machine learning

• Can do various speed ups:
  • Hashing
    – compute a short descriptor from each feature vector, or hash longer descriptors (randomly)
  • Fast Nearest neighbor techniques
    – $kd$-trees and their variants
  • Clustering / Vector quantization
    – So called “visual words”
What about outliers?
Feature-space outlier rejection

Let’s not match all features, but only these that have “similar enough” matches?

How can we do it?

• $SSD(\text{patch1}, \text{patch2}) < \text{threshold}$
• How to set threshold?
Feature-space outlier rejection: symmetry

Let’s not match all features, but only those that have “similar enough” matches?

How can we do it?

- Symmetry: x’s NN is y, and y’s NN is x
Feature-space outlier rejection: Lowe’s trick

A better way [Lowe, 1999]:

- 1-NN: SSD of the closest match
- 2-NN: SSD of the second-closest match
- Look at how much better 1-NN is than 2-NN, e.g. 1-NN/2-NN
- That is, is our best match so much better than the rest?
Can we now compute H from the blue points?

- No! Still too many outliers…
- What can we do?
Matching features

What do we do about the “bad” matches?
Random Sample Consensus

Select one match, count inliers
RA
dom SA
mple C
onsensus

Select *one* match, count *inliers*
Least squares fit

Find “average” translation vector
RANSAC loop:
1. Select four feature pairs (at random)
2. Compute homography $H$ (exact)
3. Compute *inliers* where $\text{dist}(p'_i, H p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares $H$ estimate on all of the inliers
Limitations of Alignment

We need to know the global transform (e.g. affine, homography, etc)
Optical flow

Will start by estimating motion of each pixel separately
Then will consider motion of entire image
Why estimate motion?

Lots of uses

• Track object behavior
• Correct for camera jitter (stabilization)
• Align images (even if no global transform)
• 3D shape reconstruction
• Special effects
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: Q: what’s the equation?
  \[ 0 = I(x + u, y + v) - H(x, y) \]
- 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

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

\[ \approx I(x, y) + I_xu + I_yv - H(x, y) \]

\[ \approx (I(x, y) - H(x, y)) + I_xu + I_yv \]

\[ \approx I_t + I_xu + I_yv \]

\[ \approx I_t + \nabla I \cdot [u \ v] \]

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

\[ 0 = I_t + \nabla I \cdot [\frac{\partial I}{\partial t} \ \frac{\partial I}{\partial t}] \]
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
Aperture problem
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!

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

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

\[ A \quad 25 \times 2 \quad d \quad 2 \times 1 \quad b \quad 25 \times 1 \]
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 $$25 \times 3$$ equations per pixel!

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

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

\[
A_{75 \times 2} \quad d_{2 \times 1} \quad b_{75 \times 1}
\]
Lukas-Kanade flow

Prob: we have more equations than unknowns

\[ \begin{array}{ccc}
A & d & = b \\
25x2 & 2x1 & 25x1 
\end{array} \quad \text{minimize } \|Ad - b\|^2 \]

Solution: solve least squares problem

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

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

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

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

- The summations are over all pixels in the K x K window
- This technique was first proposed by Lukas & 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?

- \(A^T A\) 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

\[
A^T A = \left[\begin{array}{cc}
\sum I_x I_x & \sum I_x I_y \\
\sum I_x I_y & \sum I_y I_y
\end{array}\right] = \sum \begin{bmatrix}
I_x \\
I_y
\end{bmatrix} [I_x \ I_y] = \sum \nabla I (\nabla I)^T
\]
Local Patch Analysis
Edge

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

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

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

- 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?
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 ($2^{nd}$ 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

$u=1.25 \text{ pixels}$

$u=2.5 \text{ pixels}$

$u=5 \text{ pixels}$

$u=10 \text{ pixels}$
Coarse-to-fine optical flow estimation

Gaussian pyramid of image H

Gaussian pyramid of image I

run iterative L-K

warp & upsample

run iterative L-K