Lecture 18: Fitting geometric models
Announcements

- Section this week: project office hours
- PS8: panorama stitching
Today

- Finding correspondences
- Fitting a homography
- RANSAC
- Triangulation
Panorama stitching (PS9)
Panorama stitching

Warp using homography
Panorama stitching

We’ll estimate the homography from correspondences!
Finding correspondences with local features

1) **Detection:** Identify the interest points (a.k.a. keypoints), the candidate points to match.

2) **Description:** Extract vector feature descriptor surrounding each interest point.

3) **Matching:** Determine correspondence between descriptors in two views.

Source: K. Grauman
What are good regions to match?
What are good regions to match?

- How does the window change when you shift it?
- Shifting the window in any direction causes a big change

“flat” region: no change in all directions

“edge”: no change along the edge direction

“corner”: significant change in all directions

Source: S. Seitz, D. Frolova, D. Simakov, N. Snavely
Finding good key points to match

Compute difference-of-Gaussians filter (approx. to Laplacian).

Find local optima in space and scale using Laplacian pyramid.
Feature descriptors
We know how to detect good points
Next question: How to match them?

Come up with a descriptor (feature vector) for each point, find similar descriptors between the two images

Source: N. Snavely
Simple idea: normalized image patch

We want invariance to rotation, lighting, and tiny spatial shifts.

Take 40x40 window around feature
- Find dominant orientation
- Rotate to horizontal
- Downsample to 8x8
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window
Scale Invariant Feature Transform (SIFT)

- Compute histograms of oriented gradients
- Take 16x16 square window around detected feature
- Compute edge orientation for each pixel
- Looks like a small, hand-crafted CNN
Scale Invariant Feature Transform

Create the descriptor:
- Rotation invariance: rotate by “dominant” orientation
- Spatial invariance: spatial pool to 2x2
- Compute an orientation histogram for each cell
- (4 x 4) cells x 8 orientations = 128 dimensional descriptor

Source: N. Snavely, D. Lowe
SIFT invariances

Source: N. Snavely
Today

- Finding correspondences
- Computing local features
  - **Matching**
- Fitting a homography
- RANSAC
How can we tell if two features match?

Source: N. Snavely
Feature matching

Given a feature in $I_1$, how do we find the best match in $I_2$?

1. Define distance function that compares two descriptors
2. Test all the features in $I_2$, find the closest one.

Source: N. Snavely
Finding matches

How do we know if two features match?
- Simple approach: are they the nearest neighbor in $L_2$ distance, $||f_1 - f_2||$?

Source: N. Snavely
Finding matches

How do we know if two features match?

- Simple approach: are they the nearest neighbor in $L_2$ distance, $||f_1 - f_2||$?
- Can give good scores to ambiguous (incorrect) matches

Source: N. Snavely
Finding matches

Throw away matches that fail tests:

- **Ratio test:** this *by far* the best match? Compare best and 2nd-best matches.
  - Ratio distance = $\|f_1 - f_2\| / \|f_1 - f_2'\|$
  - $f_2$ is best SSD match to $f_1$ in $I_2$
  - $f_2'$ is 2nd best SSD match to $f_1$ in $I_2$
- **Forward-backward consistency:** $f_1$ should also be nearest neighbor of $f_2$

Source: N. Snavely
Feature matching example

51 feature matches after ratio test

Source: N. Snavely
Feature matching example

58 feature matches after ratio test

Source: N. Snavely
Today

- Finding correspondences
- Computing local features
- Matching
- **Fitting a homography**
- RANSAC
From matches to a homography

\[ (x_1, y_1) \]

\[ (x'_1, y'_1) \]

\[
\begin{pmatrix}
    x'_1 \\
    y'_1 \\
    w_1
\end{pmatrix} =
\begin{pmatrix}
a & b & c \\
    d & e & f \\
g & h & i
\end{pmatrix}
\cdot
\begin{pmatrix}
x_1 \\
y_1 \\
1
\end{pmatrix}
\]
From matches to a homography

\[
J(H) = \sum_i \| f_H(p_i) - p'_i \|^2
\]

where \( f_H(p_i) = H p_i / (H^T_3 p_i) \) applies homography

Remember: homogenous coordinates.

\( H_3 \) is the third row of \( H \)
Option #1: Direct linear transform

\[
\begin{bmatrix}
  x'_1 \\
  y'_1 \\
  w_1
\end{bmatrix} =
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  g & h & i
\end{bmatrix} \cdot
\begin{bmatrix}
  x_1 \\
  y_1 \\
  1
\end{bmatrix}
\]

Leaving homogeneous coordinates:

\[
x'_1 = \frac{ax_1 + by_1 + c}{gx_1 + hy_1 + i}
\]

\[
y'_1 = \frac{dx_1 + ey_1 + f}{gx_1 + hy_1 + i}
\]

Re-arranging the terms:

\[
 gx_1x'_1 + hy_1x'_1 + ix'_1 = ax_1 + by_1 + c
\]

\[
 gx_1y'_1 + hy_1y'_1 + ix'_1 = dx_1 + ey_1 + f
\]

Source: Torralba, Freeman, Isola
Option #1: Direct linear transform

\[ \begin{align*}
    gx_1x' + hy_1x' + ix' &= ax_1 + by_1 + c \\
    gx_1y' + hy_1y' + ix' &= dx_1 + ey_1 + f
\end{align*} \]

More rearranging:

\[ \begin{align*}
    gx_1x' + hy_1x' + ix' - ax_1 - by_1 - c &= 0 \\
    gx_1y' + hy_1y' + iy' - dx_1 - ey_1 - f &= 0
\end{align*} \]

In matrix form:

\[
\begin{bmatrix}
-x_1 & -y_1 & -1 & 0 & 0 & x_1x' & y_1x' & x' \\
0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y' & y_1y' & y'
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d \\
e \\
f \\
g \\
h \\
i \\
j
\end{bmatrix}
= \begin{bmatrix} 0 \\
0 \end{bmatrix}
\]

Can solve using Singular Value Decomposition (SVD).

Fast to solve (but not using “right” loss function). Uses an algebraic trick. Often used in practice for initial solutions!

Source: Torralba, Freeman, Isola
Option #2: Optimization

\[
J(H) = \sum_i \| f_H(p_i) - p'_i \|^2
\]

minimize

- \( J(H) = \sum_i ||f_H(p_i) - p'_i||^2 \)
Optimization

\[ J(H) = \sum_i \| f_H(p_i) - p'_i \|^2 \]

- Can use gradient descent, just like when learning neural nets
- These problems are **smaller scale** than deep learning problems but have **more local optima**:
  - Use 2nd derivatives to improve optimization
  - Can use finite differences or autodiff
- Can use special-purpose **nonlinear least squares** methods.
  - Exploits structure in the problem for a sum-of-squares loss.
Problem: outliers

Source: N. Snavely
One idea: robust loss functions

minimize $J(H) = \sum_{i=1}^{N} \sum_{j=1}^{2} \rho(f_H(p_{ij}) - p'_{ij})$

where $\rho(x)$ is a **robust** loss.

Special case: $\rho(x) = x^2$ is L2 loss (same as before)
Robust loss functions

L1 loss: $\rho(x) = |x|$
Robust loss functions

Truncated quadratic: \( \rho(x) = \min(x^2, \tau) \)
Robust loss functions

Huber loss:

\[ \rho(x) = \begin{cases} 
\frac{1}{2}x^2, & \text{if } |x| \leq \tau, \\
\tau(|x| - \frac{1}{2}\tau), & \text{else}
\end{cases} \]
Robust loss functions

Source: [Barron 2019, “A General and Adaptive Robust Loss Function”]
Handling outliers

- Can be hard to fit a robust loss, e.g., due to local minima
- Another idea: trial and error!
- Let’s consider the problem of linear regression

Problem: Fit a line to these data points

Least squares fit

Source: N. Snavely
Counting inliers

Source: N. Snavely
Counting inliers

Inliers: 3

Source: N. Snavely
Counting inliers

Inliers: 20

Source: N. Snavely
RANSAC

• Idea:
  – All the inliers will agree with each other on the solution; the (hopefully small) number of outliers will (hopefully) disagree with each other
  • RANSAC only has guarantees if there are < 50% outliers

  – “All good matches are alike; every bad match is bad in its own way.”
  – Tolstoy via Alyosha Efros

Source: N. Snavely
RANSAC: random sample consensus

RANSAC loop (for N iterations):

• Select four feature pairs (at random)
• Compute homography $H$
• Count inliers where $\|p_i' - f_H(p_i)\| < \varepsilon$

Afterwards:

• Choose $H$ with largest set of inliers
• Recompute $H$ using only those inliers (often using high-quality nonlinear least squares)

Source: Torralba, Freeman, Isola
Simple example: fit a line

• Rather than homography $H$ (8 numbers)
  fit $y=ax+b$ (2 numbers $a$, $b$) to 2D pairs

Source: Torralba, Freeman, Isola
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers

3 inliers

Source: Torralba, Freeman, Isola
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers

Source: Torralba, Freeman, Isola
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers

Source: Torralba, Freeman, Isola
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers

8 inliers

Source: Torralba, Freeman, Isola
Simple example: fit a line

- Use biggest set of inliers
- Do least-square fit

Source: Torralba, Freeman, Isola
Example: fitting a translation

Source: N. Snavely
RAndom SAample Consensus

Select one match at random, count inliers

Source: N. Snavely
RAndom SAmple Consensus

Select one match at random, count inliers

Source: N. Snavely
**R**andom **S**Ample **C**onsensus

Select *one* match at random, count *inliers*

Source: N. Snavely
RAnDom SAmple Consensus

Select another match at random, count inliers

Source: N. Snavely
Select another match at random, count inliers

Source: N. Snavely
RAndom SAmple Consensus

Choose the translation with the highest number of inliers

Then compute average translation, using only inliers

Source: N. Snavely
Warping with a homography (PS8)

1. Compute features using SIFT

2. Match features

3. Compute homography using RANSAC

Source: N. Snavely
A similar geometric problem: triangulation

Given projection $p_i$ of unknown 3D point $X$ in two or more images (with known cameras $P_i$), find $X$
Triangulation

Given projection $p_i$ of unknown 3D point $X$ in two or more images (with known cameras $P_i$), find $X$.

Why is the calibration here important?

Source: D. Fouhey
Triangulation

Rays in principle should intersect, but in practice usually don’t exactly due to noise, numerical errors.

Source: D. Fouhey
Triangulation – Geometry

Find shortest segment between viewing rays, set $X$ to be the midpoint of the segment.

Source: D. Fouhey
Triangulation – Non-linear Optim.

Find $X$ minimizing $d(p_1, P_1X)^2 + d(p_2, P_2X)^2$

where $d$ is distance in image space

Source: D. Fouhey
Triangulation – Linear Optimization

Source: D. Fouhey
First: A better way to handle homogeneous coordinates in linear optimization

Recall: projection in homogeneous coordinates.

\[ p_i \equiv PX_i \]

Remember: this implies \( PX_i \) & \( p_i \) are proportional/scaled copies of each other

\[ p_i = \lambda PX_i, \ \lambda \neq 0 \]

This implies their cross product is 0, since

\[ a \times b = ||a|| \ ||b|| \ \sin(\theta). \]

\[ p_i \times PX_i = 0 \]

Handles the “divide by 0” issue we saw before.

Source: D. Fouhey
Triangulation – Linear Optimization

\[ p_1 \equiv P_1X \quad \Rightarrow \quad p_1 \times P_1X = 0 \quad \Rightarrow \quad [p_{1x}]P_1X = 0 \]

\[ p_2 \equiv P_2X \quad \Rightarrow \quad p_2 \times P_2X = 0 \quad \Rightarrow \quad [p_{2x}]P_2X = 0 \]

Cross Prod. as matrix

\[ a \times b = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [a_x]b \]

\[ [p_{1x}]P_1X = 0 \]

\[ [p_{2x}]P_2X = 0 \]

Two eqns per camera for 3 unknowns in X

Source: D. Fouhey
Next time: Estimating 3D structure

• Given many images, how can we…
  1. Figure out where they were all taken from?
  2. Build a 3D model of the scene?

This is the **structure from motion** problem

Source: N. Snavely
Structure from motion

- Input: images with pixels in correspondence \( p_{i,j} = (u_{i,j}, v_{i,j}) \)

- Output
  - **Structure**: 3D location \( x_i \) for each point \( p_i \)
  - **Motion**: camera parameters \( R_j, t_j \) possibly \( K_j \)

- Objective function: minimize reprojection error

Source: N. Snavely
Camera calibration & triangulation

• Suppose we know 3D points
  – And have matches between these points and an image
  – Computing camera parameters similar to homography estimation

• Suppose we have know camera parameters, each of which observes a point
  – We can solve for the 3D location

• Seems like a chicken-and-egg problem, but in SfM we can solve both at once

Source: N. Snavely
Next class: more 3D