Fitting and Alignment

Computer Vision
Jia-Bin Huang, Virginia Tech

Many slides from S. Lazebnik and D. Hoiem
Administrative Stuffs

• Homework grading policy
  • Graduate students: graded out of 600 points
  • Undergrad students: graded out of 525 points
  • For example, grad students need to complete on average 25 points extra credits.

• HW 1
  • Extra credits due 11:59 PM Friday 9/23
  • Competition: Edge Detection
    • Submission link
    • Leaderboard

• Anonymous feedback
  • Lectures are too fast, too many slides
Where are we?

• Interest points
  • Find *distinct* and *repeatable* points in images
  • Harris-> corners, DoG -> blobs
  • SIFT -> feature descriptor

• Feature tracking and optical flow
  • Find motion of a keypoint/pixel over time
  • Lucas-Kanade:
    • brightness consistency, small motion, spatial coherence
  • Handle large motion:
    • iterative update + pyramid search

• Fitting and alignment (this class)
  • find the transformation parameters that best align matched points

• Object instance recognition (next class)
  • Keypoint-based object instance recognition and search
Review: Harris Corner Detector

• Second moment matrix

\[
\mu(\sigma_I, \sigma_D) = g(\sigma_I) \ast \begin{bmatrix}
I^2_x(\sigma_D) & I_x I_y(\sigma_D) \\
I_x I_y(\sigma_D) & I^2_y(\sigma_D)
\end{bmatrix}
\]

1. Image derivatives (optionally, blur first)

\[
\det M = \lambda_1 \lambda_2 \\
\text{trace } M = \lambda_1 + \lambda_2
\]

2. Square of derivatives

3. Gaussian filter \(g(\sigma_I)\)

4. Cornerness function – both eigenvalues are strong

\[
har = \det[\mu(\sigma_I, \sigma_D)] - \alpha[\text{trace}(\mu(\sigma_I, \sigma_D))^2] = g(I^2_x)g(I^2_y) - [g(I_x I_y)]^2 - \alpha[g(I^2_x) + g(I^2_y)]^2
\]

5. Non-maxima suppression
Review: Find local maxima in position-scale space of Difference-of-Gaussian

\[ L_{xx}(\sigma) + L_{yy}(\sigma) \Rightarrow \text{List of } (x, y, s) \]
Review: SIFT Descriptor

Histogram of oriented gradients

- Captures important texture information
- Robust to small translations / affine deformations

[Lowe, ICCV 1999]
Review: Lucas-Kanade Tracker

\begin{align*}
\begin{bmatrix}
(x, y) \\
\text{displacement} = (u, v)
\end{bmatrix} & 
\begin{bmatrix}
(x + u, y + v)
\end{bmatrix}

I(x, y, t) & = I(x + u, y + v, t + 1) \\
I(x + u, y + v, t + 1) & \approx I(x, y, t) + I_x \cdot u + I_y \cdot v + I_t \\
I_x \cdot u + I_y \cdot v + I_t & \approx 0
\end{align*}

Small motion

Spatial coherence
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\(^{nd}\) moment matrix for feature patch in first image

3. Shift window by \((u, v)\):
\[x' = x' + u; \quad y' = y' + v;\]

4. Recalculate \(I_t\)

5. Repeat steps 2-4 until small change
   • Use interpolation for subpixel values
Dealing with larger movements: coarse-to-fine registration

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

5. Repeat steps 3-4 for finer levels of the pyramid.
Fitting ['fidiNG]:
find the parameters of a model that best fit the data

Alignment [əˈlīnmənt]:
find the parameters of the transformation that best align matched points
Fitting and alignment

- Choose a *parametric model* to represent a set of features

- simple model: lines
- simple model: circles
- complicated model: car
- complicated model: face shape
Fitting and Alignment - Design challenges

• Design a suitable *goodness of fit* measure
  • Similarity should reflect application goals
  • Encode robustness to outliers and noise

• Design an *optimization* method
  • Avoid local optima
  • Find best parameters quickly
Fitting and Alignment: Methods

- Global optimization / Search for parameters
  - Least squares fit
  - Robust least squares
  - Iterative closest point (ICP)

- Hypothesize and test
  - Generalized Hough transform
  - RANSAC
Simple example: Fitting a line
Least squares line fitting

- Data: \((x_1, y_1), \ldots, (x_n, y_n)\)
- Line equation: \(y_i = mx_i + b\)
- Find \((m, b)\) to minimize

\[
E = \sum_{i=1}^{n} (y_i - mx_i - b)^2
\]

\[
E = \sum_{i=1}^{n} \left( \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2
= \left\| \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2
= \|Ap - y\|^2
\]

\[
y = mx + b
\]

\[
\begin{align*}
dE \\
dp
&= 2A^T Ap - 2A^T y = 0
\end{align*}
\]

Matlab: \(p = A \ \backslash \ y\); 

\[
A^T Ap = A^T y \Rightarrow p = (A^T A)^{-1} A^T y
\]

Modified from S. Lazebnik
Problem with “vertical” least squares

• Not rotation-invariant
• Fails completely for vertical lines
Total least squares

If \((a^2+b^2=1)\) then

Distance between point \((x_i, y_i)\) and line \(ax+by+c=0\) is \(|ax_i + by_i + c|\)

proof: [http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html](http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html)
Total least squares

If \((a^2+b^2=1)\) then

Distance between point \((x_i, y_i)\) and line \(ax+by+c=0\) is \(|ax_i + by_i + c|\)

Find \((a, b, c)\) to minimize the sum of squared perpendicular distances

\[
E = \sum_{i=1}^{n} (ax_i + by_i + c)^2
\]
Total least squares

Find \((a, b, c)\) to minimize the sum of squared perpendicular distances

\[
E = \sum_{i=1}^{n} (ax_i + by_i + c)^2
\]

\[
\frac{\partial E}{\partial c} = \sum_{i=1}^{n} 2(ax_i + by_i + c) = 0
\]

\[
E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \sum_{i=1}^{n} (y_i - b \bar{x} - a \bar{y})^2 = \mathbf{p}^T \mathbf{A}^T \mathbf{A} \mathbf{p}
\]

minimize \(\mathbf{p}^T \mathbf{A}^T \mathbf{A} \mathbf{p}\) s.t. \(\mathbf{p}^T \mathbf{p} = 1\) \(\Rightarrow\) minimize \(\frac{\mathbf{p}^T \mathbf{A}^T \mathbf{A} \mathbf{p}}{\mathbf{p}^T \mathbf{p}}\)

Solution is eigenvector corresponding to smallest eigenvalue of \(\mathbf{A}^T \mathbf{A}\)

# Recap: Two Common Optimization Problems

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimize $|Ax - b|^2$</td>
<td>$x = (A^T A)^{-1} A^T b$</td>
</tr>
<tr>
<td>least squares solution to $Ax = b$</td>
<td>$x = A \backslash b$ (matlab)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem statement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimize $x^T A^T A x$ s.t. $x^T x = 1$</td>
<td>$[v, \lambda] = \text{eig}(A^T A)$</td>
</tr>
<tr>
<td>minimize $\frac{x^T A^T A x}{x^T x}$</td>
<td>$\lambda_1 &lt; \lambda_{2..n} : x = v_1$</td>
</tr>
<tr>
<td>non-trivial lsq solution to $Ax = 0$</td>
<td></td>
</tr>
</tbody>
</table>
Least squares (global) optimization

Good
• Clearly specified objective
• Optimization is easy

Bad
• May not be what you want to optimize
• Sensitive to outliers
  • Bad matches, extra points
• Doesn’t allow you to get multiple good fits
  • Detecting multiple objects, lines, etc.
Robust least squares (to deal with outliers)

General approach:

\[
\text{minimize} \quad \sum_i \rho(u_i(x_i, \theta); \sigma) \quad u^2 = \sum_{i=1}^n (y_i - mx_i - b)^2
\]

- \( u_i(x_i, \theta) \) – residual of i\textsuperscript{th} point w.r.t. model parameters \( \theta \)
- \( \rho \) – robust function with scale parameter \( \sigma \)

The robust function \( \rho \)

- Favors a configuration with small residuals
- Constant penalty for large residuals
Robust Estimator

\[ \sigma = 1.5 \cdot \text{median}(\text{error}) \]

1. Initialize: e.g., choose \( \theta \) by least squares fit and

\[ \sum_i \frac{\text{error}(\theta, \text{data}_i)^2}{\sigma^2 + \text{error}(\theta, \text{data}_i)^2} \]

2. Choose params to minimize:
   - E.g., numerical optimization
   \[ \sigma = 1.5 \cdot \text{median}(\text{error}) \]

3. Compute new

4. Repeat (2) and (3) until convergence
Other ways to search for parameters (for when no closed form solution exists)

• Line search
  1. For each parameter, step through values and choose value that gives best fit
  2. Repeat (1) until no parameter changes

• Grid search
  1. Propose several sets of parameters, evenly sampled in the joint set
  2. Choose best (or top few) and sample joint parameters around the current best; repeat

• Gradient descent
  1. Provide initial position (e.g., random)
  2. Locally search for better parameters by following gradient
Hypothesize and test

1. Propose parameters
   - Try all possible
   - Each point votes for all consistent parameters
   - Repeatedly sample enough points to solve for parameters

2. Score the given parameters
   - Number of consistent points, possibly weighted by distance

3. Choose from among the set of parameters
   - Global or local maximum of scores

4. Possibly refine parameters using inliers
Hough Transform: Outline

1. Create a grid of parameter values

2. Each point votes for a set of parameters, incrementing those values in grid

3. Find maximum or local maxima in grid
Hough transform


Given a set of points, find the curve or line that explains the data points best

\[ y = m x + b \]
Hough transform

Slide from S. Savarese
Hough transform


Issue: parameter space \([m,b]\) is unbounded...

Use a polar representation for the parameter space

\[
x \cos \theta + y \sin \theta = \rho
\]

(Hough space)

Slide from S. Savarese
Hough transform - experiments

features

votes

Slide from S. Savarese
Hough transform - experiments

Noisy data

Need to adjust grid size or smooth
Hough transform - experiments

Issue: spurious peaks due to uniform noise
1. Image ➤ Canny
2. Canny $\rightarrow$ Hough votes
3. Hough votes → Edges

Find peaks and post-process
Hough transform example

http://ostatic.com/files/images/ss_hough.jpg
Hough transform for circles

• Circle: center \((a,b)\) and radius \(r\)
  \[
  (x_i - a)^2 + (y_i - b)^2 = r^2
  \]

• For a fixed radius \(r\)

Equation of circle?

Equation of set of circles that all pass through a point?
Hough transform for circles

- Circle: center \((a,b)\) and radius \(r\)
  \[
  (x_i - a)^2 + (y_i - b)^2 = r^2
  \]

- For a fixed radius \(r\)
Hough transform for circles

- Circle: center \((a, b)\) and radius \(r\)
  \[
  (x_i - a)^2 + (y_i - b)^2 = r^2
  \]
- For an unknown radius \(r\)
Hough transform for circles

- Circle: center \((a, b)\) and radius \(r\)

\[
(x_i - a)^2 + (y_i - b)^2 = r^2
\]

- For an unknown radius \(r\)
Hough transform for circles

• Circle: center \((a,b)\) and radius \(r\)

\[
(x_i - a)^2 + (y_i - b)^2 = r^2
\]

• For an unknown radius \(r\), known gradient direction

![Image of Hough transform for circles](image.png)
Example: detecting circles with Hough

Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).
Example: detecting circles with Hough

Combining detections

Edges

Votes: Quarter

Coin finding sample images from: Vivek Kwatra
Generalized Hough for object detection

• Instead of indexing displacements by gradient orientation, index by matched local patterns.

B. Leibe, A. Leonardis, and B. Schiele, Combined Object Categorization and Segmentation with an Implicit Shape Model, ECCV Workshop on Statistical Learning in Computer Vision 2004

“visual codeword” with displacement vectors

Source: L. Lazebnik
Generalized Hough for object detection

• Instead of indexing displacements by gradient orientation, index by “visual codeword”

B. Leibe, A. Leonardis, and B. Schiele, Combined Object Categorization and Segmentation with an Implicit Shape Model, ECCV Workshop on Statistical Learning in Computer Vision 2004

Source: L. Lazebnik
Hough transform conclusions

Good
• Robust to outliers: each point votes separately
• Fairly efficient (much faster than trying all sets of parameters)
• Provides multiple good fits

Bad
• Some sensitivity to noise
• Bin size trades off between noise tolerance, precision, and speed/memory
  • Can be hard to find sweet spot
• Not suitable for more than a few parameters
  • grid size grows exponentially

Common applications
• Line fitting (also circles, ellipses, etc.)
• Object instance recognition (parameters are position/scale/orientation)
• Object category recognition (parameters are position/scale)
RANSAC

(RANdom SAmple Consensus):

Fischler & Bolles in ‘81.

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
RANSAC

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Illustration by Savarese
RANSAC

Algorithm:
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
**Algorithm:**

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence
How to choose parameters?

• Number of samples $N$
  • Choose $N$ so that, with probability $p$, at least one random sample is free from outliers (e.g. $p=0.99$) (outlier ratio: $e$)

• Number of sampled points $s$
  • Minimum number needed to fit the model

• Distance threshold $\delta$
  • Choose $\delta$ so that a good point with noise is likely (e.g., prob=0.95) within threshold
  • Zero-mean Gaussian noise with std. dev. $\sigma$: $t^2=3.84\sigma^2$

\[
N = \frac{\log(1-p)}{\log\left(1 - (1-e)^s\right)}
\]

<table>
<thead>
<tr>
<th>$s$</th>
<th>5%</th>
<th>10%</th>
<th>20%</th>
<th>25%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>19</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>12</td>
<td>17</td>
<td>26</td>
<td>57</td>
<td>146</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>7</td>
<td>16</td>
<td>24</td>
<td>37</td>
<td>97</td>
<td>293</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>8</td>
<td>20</td>
<td>33</td>
<td>54</td>
<td>163</td>
<td>588</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>26</td>
<td>44</td>
<td>78</td>
<td>272</td>
<td>1177</td>
</tr>
</tbody>
</table>
RANSAC conclusions

Good
• Robust to outliers
• Applicable for larger number of objective function parameters than Hough transform
• Optimization parameters are easier to choose than Hough transform

Bad
• Computational time grows quickly with fraction of outliers and number of parameters
• Not as good for getting multiple fits (though one solution is to remove inliers after each fit and repeat)

Common applications
• Computing a homography (e.g., image stitching)
• Estimating fundamental matrix (relating two views)
RANSAC Song
What if you want to align but have no prior matched pairs?

• Hough transform and RANSAC not applicable

• Important applications

Medical imaging: match brain scans or contours

Robotics: match point clouds
Iterative Closest Points (ICP) Algorithm

Goal: estimate transform between two dense sets of points

1. **Initialize** transformation (e.g., compute difference in means and scale)
2. **Assign** each point in \{Set 1\} to its nearest neighbor in \{Set 2\}
3. **Estimate** transformation parameters
   • e.g., least squares or robust least squares
4. **Transform** the points in \{Set 1\} using estimated parameters
5. **Repeat** steps 2-4 until change is very small
Example: solving for translation

Given matched points in \{A\} and \{B\}, estimate the translation of the object

\[
\begin{bmatrix}
  x_i^B \\
  y_i^B
\end{bmatrix} = \begin{bmatrix}
  x_i^A \\
  y_i^A
\end{bmatrix} + \begin{bmatrix}
  t_x \\
  t_y
\end{bmatrix}
\]
Example: solving for translation

Least squares solution

1. Write down objective function
2. Derived solution
   a) Compute derivative
   b) Compute solution
3. Computational solution
   a) Write in form $Ax=b$
   b) Solve using pseudo-inverse or eigenvalue decomposition

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x_1^B - x_1^A \\ y_1^B - y_1^A \\ \vdots \\ x_n^B - x_n^A \\ y_n^B - y_n^A \end{bmatrix}$$

$$\begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_1^B - x_1^A \\ y_1^B - y_1^A \\ \vdots \\ x_n^B - x_n^A \\ y_n^B - y_n^A \end{bmatrix}$$
Example: solving for translation

Problem: outliers

RANSAC solution
1. Sample a set of matching points (1 pair)
2. Solve for transformation parameters
3. Score parameters with number of inliers
4. Repeat steps 1-3 N times

\[
\begin{bmatrix}
    x_i^B \\
    y_i^B
\end{bmatrix} = \begin{bmatrix}
    x_i^A \\
    y_i^A
\end{bmatrix} + \begin{bmatrix}
    t_x \\
    t_y
\end{bmatrix}
\]
Example: solving for translation

Problem: outliers, multiple objects, and/or many-to-one matches

Hough transform solution

1. Initialize a grid of parameter values
2. Each matched pair casts a vote for consistent values
3. Find the parameters with the most votes
4. Solve using least squares with inliers

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$
Example: solving for translation

Problem: no initial guesses for correspondence

ICP solution
1. Find nearest neighbors for each point
2. Compute transform using matches
3. Move points using transform
4. Repeat steps 1-3 until convergence

\[
\begin{bmatrix}
    x_i^B \\
    y_i^B
\end{bmatrix} = \begin{bmatrix}
    x_i^A \\
    y_i^A
\end{bmatrix} + \begin{bmatrix}
    t_x \\
    t_y
\end{bmatrix}
\]
HW 2 – Feature tracker

• Keypoint detection
  • Compute second moment matrix
  • Harris corner criterion
  • Threshold
  • Non-maximum suppression

• Tracking
  • Kanade-Lucas-Tomasi tracking
  • Show keypoint trajectories
  • Show points have moved out of frames
HW 2 – Shape Alignment

- Global transformation (similarity, affine, perspective)
- Iterative closest point algorithm
HW 2 – Local Feature Matching

• Express location of the detected object

• Implement ratio test feature matching algorithm
Things to remember

• Least Squares Fit
  • closed form solution
  • robust to noise
  • not robust to outliers

• Robust Least Squares
  • improves robustness to noise
  • requires iterative optimization

• Hough transform
  • robust to noise and outliers
  • can fit multiple models
  • only works for a few parameters (1-4 typically)

• RANSAC
  • robust to noise and outliers
  • works with a moderate number of parameters (e.g., 1-8)

• Iterative Closest Point (ICP)
  • For local alignment only: does not require initial correspondences
Next week

• Object instance recognition