CS 4495 Computer Vision
**RAN**dom **SA**mple **C**onsensus

Aaron Bobick
School of Interactive Computing
Administrivia

• PS 3:
  • Check Piazza - good conversations. In fact some pretty explicit solutions...  
  • The F matrix: the actual numbers may vary quite a bit. But check the epipolar lines you get.
  
  • Normalization: read extra credit part. At least try removing the centroid. Since we’re using homogenous coordinates (2D homogenous have 3 elements) it’s easy to have a transformation matrix that subtracts off an offset.

• Go back an recheck slides: A 3 vector in these projective geometry is both a point and a line.
Matching with Features

• Want to compute transformation from one image to the other

• Overall strategy:
  • Compute features
  • Match matching features (duh?)
  • Compute best transformation (translation, affine, homography) from matches
An introductory example:

**Harris corner detector**

Harris Detector: Mathematics

\[ M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \]

Measure of corner response:

\[ R = \det M - \alpha (\text{trace } M)^2 \]

\[ \det M = \lambda_1 \lambda_2 \]

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

(\(\alpha\)– empirical constant, typically 0.04 - 0.06)
Harris corner response function

\[ R = \text{det}(M) - \alpha \text{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2 \]

- \( R \) depends only on eigenvalues of \( M \), but don’t compute them (no sqrt, so really fast!)
- \( R \) is large for a **corner**
- \( R \) is negative with large magnitude for an **edge**
- \(|R|\) is small for a **flat** region
Key point localization

- General idea: find robust extremum (maximum or minimum) both in space and in scale.

- SIFT specific suggestion: use DoG pyramid to find maximum values (remember edge detection?) – then eliminate “edges” and pick only corners.

- More recent: use Harris detector to find maximums in space and then look at the Laplacian pyramid (we’ll do this later) for maximum in scale.

Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below.
Point Descriptors

• We know how to detect points
• How to match them? Two parts:
  • Compute a descriptor for each and make the descriptor both as invariant and as distinctive as possible. (Competing goals) SIFT one example.
Idea of SIFT

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters.
SIFT vector formation

- 4x4 array of gradient orientation histograms over 4x4 pixels
  - not really histogram, weighted by magnitude
- 8 orientations x 4x4 array = 128 dimensions
- Motivation: some sensitivity to spatial layout, but not too much.
Point Descriptors

- We know how to detect points
- How to match them? Two parts:
  - Compute a descriptor for each and make the descriptor both as invariant and as distinctive as possible. (Competing goals)
  - SIFT one example

- Need to figure out which point matches which..
Feature-based alignment outline

- Extract features
Feature-based alignment outline

- Extract features
- Compute *putative matches* – e.g. “closest descriptor”
Feature-based alignment outline

- Extract features
- Compute *putative matches* – e.g. “closest descriptor”
- Loop:
  - *Hypothesize* transformation $T$ from some matches
Feature-based alignment outline

- Extract features
- Compute **putative matches**—e.g. “closest descriptor”
- Loop:
  - Hypothesize transformation $T$ from some matches
  - Verify transformation (search for other matches consistent with $T$)
Feature-based alignment outline

- Extract features
- Compute *putative matches*—e.g. “closest descriptor”
- Loop:
  - *Hypothesize* transformation $T$ from some matches
  - *Verify* transformation (search for other matches consistent with $T$)
- Apply transformation
How to get “putative” matches?
Feature matching

- Exhaustive search
  - for each feature in one image, look at all the other features in the other image(s) – pick best one

- Hashing
  - compute a short descriptor from each feature vector, or hash longer descriptors (randomly)

- Nearest neighbor techniques
  - k-trees and their variants
Feature-space outlier rejection

- Let’s not match all features, but only these that have “similar enough” matches?
- How can we do it?
  - SSD(patch1,patch2) < threshold
  - How to set threshold?
Feature-space outlier rejection

• 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 much better than the next?
Feature matching

- Exhaustive search
  - for each feature in one image, look at all the other features in the other image(s) – pick best one

- Hashing
  - compute a short descriptor from each feature vector, or hash longer descriptors (randomly)

- Nearest neighbor techniques
  - $k$-trees and their variants

- But...

- Remember: distinctive vs invariant competition? Means:

  - Problem: Even when pick best match, still lots (and lots) of wrong matches – “outliers”
Another way to remove mistakes

• Why are we doing matching?
  • To compute a model of the relation between entities

• So this is really “model fitting”
Fitting

• Choose a parametric model to represent a set of features – *remember this???

simple model: lines  simple model: circles

complicated model: car

Source: K. Grauman
Fitting: Issues

Case study: Line detection

- **Noise** in the measured feature locations
- **Extraneous data**: clutter (outliers), multiple lines
- **Missing data**: occlusions
Total least squares

- Distance between point \((x_i, y_i)\) and line \(ax + by = d\) \((a^2 + b^2 = 1)\):

\[
|ax_i + by_i - d|
\]
Total least squares

- Distance between point \((x_i, y_i)\) and line \(ax + by = d\) \((a^2 + b^2 = 1)\):
  \[
  |ax_i + by_i - d|
  \]

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

\[
E = \sum_{i=1}^{n} (ax_i + by_i - d)^2
\]
Least squares as likelihood maximization

- **Generative model**: line points are corrupted by Gaussian noise in the direction perpendicular to the line

\[
\begin{pmatrix}
  x \\
  y
\end{pmatrix} = \begin{pmatrix}
  u \\
  v
\end{pmatrix} + \mathcal{E} \begin{pmatrix}
  a \\
  b
\end{pmatrix}
\]

- point on the line
- noise: sampled from zero-mean Gaussian with std. dev. \( \sigma \)
- normal direction
Least squares as likelihood maximization

- **Generative model**: line points are corrupted by Gaussian noise in the direction perpendicular to the line

\[
\begin{pmatrix}
    x \\
    y
\end{pmatrix} = \begin{pmatrix}
    u \\
    v
\end{pmatrix} + \varepsilon \begin{pmatrix}
    a \\
    b
\end{pmatrix}
\]

**Likelihood** of points given line parameters \((a, b, d)\):

\[
P(x_1, y_1, \ldots, x_n, y_n \mid a, b, d) = \prod_{i=1}^{n} P(x_i, y_i \mid a, b, d) \propto \prod_{i=1}^{n} \exp \left( -\frac{(ax_i + by_i - d)^2}{2\sigma^2} \right)
\]

Log-likelihood:

\[
L(x_1, y_1, \ldots, x_n, y_n \mid a, b, d) = -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (ax_i + by_i - d)^2
\]
Least squares: Lack of robustness to (very) non-Gaussian noise

- Least squares fit to the red points:
Least squares: Lack of robustness to (very) non-Gaussian noise

- Least squares fit with an outlier:

*Problem*: squared error heavily penalizes outliers
Robust estimators

- General approach: minimize $\sum_i \rho(r_i(x_i, \theta); \sigma)$

$r_i(x_i, \theta)$ – residual of ith point w.r.t. model parameters $\theta$

$\rho$ – robust function with scale parameter $\sigma$

The robust function $\rho$ behaves like squared distance for small values of the residual $u$ but saturates for larger values of $u$
Choosing the scale: Just right

The effect of the outlier is minimized
Choosing the scale: Too small

The error value is almost the same for every point and the fit is very poor.
Choosing the scale: Too large

Behaves much the same as least squares
“Find consistent matches”???

- Some points (many points) are static in the world
- Some are not
- Need to find the right ones so can compute pose.
- Well tried approach:
  - *Random Sample Consensus (RANSAC)*
Simpler Example

• Fitting a straight line
Discard Outliers

- Assume few real points with distance $d > \theta$

- **RANSAC:**
  - RANdom SAmple Consensus
  - Fischler & Bolles 1981
  - Copes with a large proportion of outliers

RANSAC
(RANdom SAmple Consensus)

Algorithm:
1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
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
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (\#=2)
2. **Solve** for model parameters using sample
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 sample
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

Line fitting example

Algorithm:
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using the sample
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 sample
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
The fundamental RANSAC assumption:

*More support implies better fit.*
RANSAC for general model

- A given model has a *minimal set* – the smallest number of samples from which the model can be computed.
  - Line: 2 points

- Image transformations are models. Minimal set of $s$ of point pairs/matches:
  - *Translation*: pick one point pair
  - *Homography* (for plane) – pick 4 point pairs
  - *Fundamental matrix* – pick 8 point pairs (really 7 but lets not go there)

- Algorithm
  - Randomly select $s$ points (or point pairs) to form a sample
  - Instantiate a model
  - Get consensus set $C_i$
  - If $|C_i| > T$, terminate and return model
  - Repeat for N trials, return model with max $|C_i|$
Distance Threshold

- Let’s assume **location** noise is Gaussian with $\sigma^2$.
- Then the **distance** $d$ has **Chi** distribution with $k$ degrees of freedoms where $k$ is the dimension of the Gaussian.
- If one dimension, e.g. distance off a line, then 1DOF:

$$f(d) = \frac{\sqrt{2}e^{-\frac{d^2}{2\sigma^2}}}{\sqrt{\pi}\sigma}, \quad d \geq 0$$
Distance Threshold

For 95% cumulative threshold \( t \) when Gaussian with \( \sigma^2 \)

\[ t^2 = 3.84\sigma^2 \]

That is: if \( t^2 = 3.84\sigma^2 \) then 95% probability that \( d < t \) when point is inlier

But...

Usually set by “empirically”...
How many samples should we try?

• We want: at least one sample with all inliers

• With random samples we can’t guarantee. But with probability $p$ we can, e.g. $p = 0.99$
Choosing the parameters

• Initial number of points $s$
  • Typically minimum number needed to fit the model

• Distance threshold $t$
  • Choose $t$ so probability for inlier is high (e.g. 0.95)
  • If zero-mean Gaussian noise with std. dev. $\sigma$: $t^2 = 3.84\sigma^2$

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

Source: M. Pollefeys
Calculate $N$

1. $s$ – number of points to compute solution
2. $p$ – probability of success
3. $e$ – proportion outliers, so \( \% \) inliers = \((1 - e)\)
4. \( P(\text{sample set with all inliers}) = (1 - e)^s \)
5. \( P(\text{sample set will have at least one outlier}) = (1 - (1 - e)^s) \)
6. \( P(\text{all } N \text{ samples have outlier}) = (1 - (1 - e)^s)^N \)
7. We want \( P(\text{all } N \text{ samples have outlier}) < (1 - p) \)
8. So: \((1 - (1 - e)^s)^N < (1 - p)\)

\[ N > \log(1 - p) / \log(1 - (1 - e)^s) \]
\( N \) for probability \( p \) of at least one sample with only inliers

\[
N > \log(1 - p) / \log(1 - (1 - e)^s)
\]

- Set \( p=0.99 \) – chance of getting good sample

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\hline
s & 5\% & 10\% & 20\% & 25\% & 30\% & 40\% & 50\% \\
\hline
2 & 2 & 3 & 5 & 6 & 7 & 11 & 17 \\
3 & 3 & 4 & 7 & 9 & 11 & 19 & 35 \\
4 & 3 & 5 & 9 & 13 & 17 & 34 & 72 \\
5 & 4 & 6 & 12 & 17 & 26 & 57 & 146 \\
6 & 4 & 7 & 16 & 24 & 37 & 97 & 293 \\
7 & 4 & 8 & 20 & 33 & 54 & 163 & 588 \\
8 & 5 & 9 & 26 & 44 & 78 & 272 & 1177 \\
\hline
\end{array}
\]

- \( N \) increases steeply with \( s \)
$N$ for probability $p$ of at least one sample with only inliers

$$N > \log(1 - p) / \log(1 - (1 - e)^s)$$

- Set $p=0.99$ – chance of getting good sample

- $s = 2, e = 5\% \quad \Rightarrow \quad N=2$
- $s = 2, e = 50\% \quad \Rightarrow \quad N=17$
- $s = 4, e = 5\% \quad \Rightarrow \quad N=3$
- $s = 4, e = 50\% \quad \Rightarrow \quad N=72$
- $s = 8, e = 5\% \quad \Rightarrow \quad N=5$
- $s = 8, e = 50\% \quad \Rightarrow \quad N=1177$

- $N$ increases steeply with $s$
How big does \( N \) need to be?

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

\[
\text{So } N = f(e, s, p)
\]

\[
\text{What is } N \text{ not a function of?}
\]

\[
N = f(e, s, p), \text{ but not the number of points(matches)!}
\]
Matching features

What do we do about the “bad” matches?
RAndom SAmple Consensus (1)

Select *one* match, count *inliers*
RAndom SAmple Consensus (2)

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

Find “average” translation vector
2D transformation models

2 matches: Similarity (translation, scale, rotation)

3 matches: Affine

4 matches: Projective (homography)

Source: S. Lazebnik
RANSAC for estimating, say, homography

RANSAC loop:
1. Select four feature pairs (at random)
2. Compute homography $H_k$ (exact)
3. Compute inliers where $SSD(p_i', H_k p_i) < \varepsilon$
4. Keep $H_k$, if $C_k$ is the largest set of inliers
5. For a while go to 1
6. Re-compute least-squares $H$ estimate on all of the $C_k$ inliers
Adaptively determining the number of samples

- Inlier ratio $e$ is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers are found, e.g. 80% would yield $e=0.2$

- Adaptive procedure:
  - $N=\infty$, sample_count =0, $e = 1.0$
  - While $N >$sample_count
    - Choose a sample and count the number of inliers
    - Set $e_0 = 1 - \text{(number of inliers)}/\text{(total number of points)}$
    - If $e_0 < e$ Set $e = e_0$ and recompute $N$ from $e$:
      \[
      N = \log(1 - p) / \log(1 - (1 - e)^s)
      \]
    - Increment the sample_count by 1

Source: M. Pollefeys
RANSAC for recognition
RANSAC for fundamental matrix
Putative matches (motion) by cross-correlation (188)
RANSAC for fundamental matrix

Inliers (99)  Outliers (89)
Point cloud planes
Find the plane and object in realtime
2D transformation models

- **Similarity** (translation, scale, rotation)
- **Affine**
- **Projective** (homography)

Source: S. Lazebnik
RANSAC conclusions

*The good...*

- Simple and general
- Applicable to many different problems, often works well in practice
- Robust to large numbers of outliers
- Applicable for larger number of parameters than Hough transform
- Parameters are easier to choose than Hough transform
RANSAC conclusions

The not-so-good...

• Computational time grows quickly with the number of model parameters

• Sometimes problematic for approximate models
RANSAC conclusions

**Common applications**

- Computing a homography (e.g., image stitching) or other image transform
- Estimating fundamental matrix (relating two views)
- *Pretty much every problem in robot vision*