Lecture 5
Image Features

Christopher Brunner
AMME4710 – Computer Vision and Image Processing
Semester 2, 2013

Resources

• Readings
  – Szeliski, Computer Vision, Ch. 4
    http://szeliski.org/Book/

• Additional material (optional)

• Slides credits
  – A. Torralba, S. Seitz, T. Darrell, K. Grauman, R. Fergus
Outline

• Introduction
• Corners (Harris detector)
• Edges (Canny edge detector)
• Lines (RANSAC, Hough transform)
• Invariant Descriptors (SIFT)
• Matching (ICP)
LINES
RANSAC, HOUGH TRANSFORM
Fitting

• We’ve learned how to detect edges and corners. Now what?
• We would like to form a higher-level, more compact representation of the features in the image by grouping multiple features according to a simple model
Fitting

• Choose a parametric model to represent a set of features

  simple model: lines
  simple model: circles
  complicated model: car

Source: K. Grauman
Example: Line fitting

- Why fit lines?
  Many objects characterized by presence of straight lines

- Wait, why isn’t edge detection from earlier sufficient?
Line Fitting: Issues

- **Extra** edge points (clutter), multiple models:
  - which points go with which line, if any?

- Only some parts of each line detected, and some parts are **missing**:
  - how to find a line that bridges missing evidence?

- **Noise** in measured edge points, orientations:
  - how to detect true underlying parameters?
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

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

\[
Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \quad B = \begin{bmatrix} m \\ b \end{bmatrix} 
\]

\[ E = \|Y - XB\|^2 = (Y - XB)^T (Y - XB) = Y^T Y - 2(XB)^T Y + (XB)^T (XB) \]

\[
\frac{dE}{dB} = 2X^T XB - 2X^T Y = 0 
\]

Normal equations: least squares solution to \(XB=Y\)
Problem with “vertical” least squares

• Not rotation-invariant
• Fails completely for vertical lines
Least squares as likelihood maximization

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

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

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

\(ax + by = d\)
Least squares as likelihood maximization

- **Generative model**: line points are sampled independently and 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: \[
L(x_1, y_1, \ldots, x_n, y_n \mid a, b, d) = -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (ax_i + by_i - d)^2
\]
Robust estimators

- General approach: find model parameters $\theta$ that 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 $u$.
RANSAC

• Robust fitting can deal with a few outliers – what if we have very many?
• Random sample consensus (RANSAC): Very general framework for model fitting in the presence of outliers

• Algorithm
  1. Choose a small subset of points uniformly at random
  2. Fit a model to that subset
  3. Find all remaining points that are “close” to the model and reject the rest as outliers
  4. Do this many times and choose the best model

RANSAC for line fitting example

Source: R. Raguram
RANSAC for line fitting example

Least-squares fit

Source: R. Raguram
RANSAC for line fitting example

1. Randomly select minimal subset of points

Source: R. Raguram
RANSAC for line fitting example

1. Randomly select minimal subset of points
2. Hypothesize a model

Source: R. Raguram
RANSAC for line fitting example

1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

Source: R. Raguram
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

Source: R. Raguram
RANSAC for line fitting example

1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat hypothesize-and-verify loop

Source: R. Raguram
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat hypothesize-and-verify loop

Source: R. Raguram
RANSAC for line fitting example

1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat hypothesize-and-verify loop

Source: R. Raguram
RANSAC for line fitting example

1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram
RANSAC for line fitting

Repeat $N$ times:

1. Draw $s$ points uniformly at random
2. Fit line to these $s$ points
3. Find inliers to this line among the remaining points (i.e., points whose distance from the line is less than $t$)
4. If there are $d$ or more inliers, accept the line and refit using all inliers
RANSAC pros and cons

• Pros
  • Simple and general
  • Applicable to many different problems
  • Often works well in practice

• Cons
  • Lots of parameters to tune
  • Doesn’t work well for low inlier ratios (too many iterations, or can fail completely)
  • Can’t always get a good initialization of the model based on the minimum number of samples
Voting

• It’s not feasible to check all combinations of features by fitting a model to each possible subset.

• **Voting** is a general technique where we let the features vote for all models that are compatible with it.
  
  – Cycle through features, cast votes for model parameters.
  – Look for model parameters that receive a lot of votes.

• Noise & clutter features will cast votes too, *but* typically their votes should be inconsistent with the majority of “good” features.
Fitting lines: Hough transform

- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?
- **Hough Transform** is a voting technique that can be used to answer all of these questions.

**Main idea:**
1. Record vote for each possible line on which each edge point lies.
2. Look for lines that get many votes.
Finding lines in an image: Hough space

Connection between image (x,y) and Hough (m,b) spaces

– A line in the image corresponds to a point in Hough space
– To go from image space to Hough space:
  • given a set of points (x,y), find all (m,b) such that \( y = mx + b \)
Finding lines in an image: Hough space

Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that \( y = mx + b \)
- What does a point \((x_0, y_0)\) in the image space map to?
  - Answer: the solutions of \( b = -x_0m + y_0 \)
  - this is a line in Hough space

Slide credit: Steve Seitz
Finding lines in an image: Hough space

What are the line parameters for the line that contains both \((x_0, y_0)\) and \((x_1, y_1)\)?

- It is the intersection of the lines \(b = -x_0m + y_0\) and \(b = -x_1m + y_1\)
Finding lines in an image: Hough algorithm

How can we use this to find the most likely parameters \((m,b)\) for the most prominent line in the image space?

- Let each edge point in image space *vote* for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.
Parameter space representation

- Problems with the \((m,b)\) space:
  - Unbounded parameter domain
  - Vertical lines require infinite \(m\)

- Alternative: polar representation

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

Each point will add a sinusoid in the \((\theta,\rho)\) parameter space
Hough transform algorithm

Using the polar parameterization:

\[ x \cos \theta - y \sin \theta = d \]

Basic Hough transform algorithm

1. Initialize \( H[d, \theta] = 0 \)
2. for each edge point \( I[x, y] \) in the image
   for \( \theta = [\theta_{\text{min}} \text{ to } \theta_{\text{max}}] \) // some quantization
   \[ d = x \cos \theta - y \sin \theta \]
   \( H[d, \theta] += 1 \)
3. Find the value(s) of \( (d, \theta) \) where \( H[d, \theta] \) is maximum
4. The detected line in the image is given by \( d = x \cos \theta - y \sin \theta \)

Time complexity (in terms of number of votes per pt)?

Source: Steve Seitz
Hough transform: pros and cons

Pros
• All points are processed independently, so can cope with occlusion, gaps
• Some robustness to noise: noise points unlikely to contribute consistently to any single bin
• Can detect multiple instances of a model in a single pass

Cons
• Complexity of search time increases exponentially with the number of model parameters
• Non-target shapes can produce spurious peaks in parameter space
• Quantization: can be tricky to pick a good grid size
Generalized Hough Transform

• What if we want to detect arbitrary shapes?

Intuition:

Model image

Novel image

Vote space

Kristen Grauman
Fitting: Summary

• If we know which points belong to the line, how do we find the “optimal” line parameters?
  – Least squares

• What if there are outliers?
  – Robust fitting, RANSAC

• What if there are many lines?
  – Voting methods: Hough transform, RANSAC

• What if we’re not even sure it’s a line?
  – Model selection
DESCRIPTORS
SIFT
Invariance

Suppose we are comparing two images $I_1$ and $I_2$

- $I_2$ may be a transformed version of $I_1$
- What kinds of transformations are we likely to encounter in practice?

We’d like to find the same features regardless of the transformation

- This is called transformational *invariance*
- Most feature methods are designed to be invariant to
  - Translation, 2D rotation, scale
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transformations (some are fully affine invariant)
  - Limited illumination/contrast changes
How to achieve invariance

Need both of the following:

1. Make sure your detector is invariant
   • Harris is invariant to translation and rotation
   • Scale is trickier
     – SIFT uses automatic scale selection
     – simpler approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS) and add them all to database

2. Design an invariant feature descriptor
   • A descriptor captures the information in a region around the detected feature point
   • The simplest descriptor: a square window of pixels
     – What’s this invariant to?
   • Let’s look at some better approaches…
Rotation Invariant Descriptors

- Find local orientation
  Dominant direction of gradient for the image patch

- Rotate patch according to this angle
  This puts the patches into a canonical orientation.
Rotation Invariant Descriptors

Image from Matthew Brown
Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images
Scale Invariant Detection

• The problem: how do we choose corresponding circles *independently* in each image?
Scale Invariant Detection

- Solution:
  - Design a function on the region (circle), which is “scale invariant” (the same for corresponding regions, even if they are at different scales)
    - Example: average intensity. For corresponding regions (even of different sizes) it will be the same.
  - For a point in one image, we can consider it as a function of region size (circle radius)
Scale Invariant Detection

• Common approach:
  
  Take a local maximum of this function

  Observation: region size, for which the maximum is achieved, should be \textit{invariant} to image scale.

**Important:** this scale invariant region size is found in each image \textit{independently}!
Scale Invariant Detection

• A “good” function for scale detection: has one stable sharp peak

• For usual images: a good function would be a one which responds to contrast (sharp local intensity change)
Scale Invariant Detection

- Functions for determining scale

Kernels:

\[ L = \sigma^2 \left( G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right) \]

(Laplacian)

\[ DoG = G(x, y, k\sigma) - G(x, y, \sigma) \]

(Difference of Gaussians)

where Gaussian

\[ G(x, y, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}} \]

Note: both kernels are invariant to scale and rotation

\[ f = \text{Kernel} \ast \text{Image} \]
\[ \text{det } M = \lambda_1 \lambda_2 \]
\[ \text{trace } M = \lambda_1 + \lambda_2 \]

From Lindeberg 1998

blob detection; Marr 1982; Voorhees and Poggio 1987; Blostein and Ahuja 1989; …
**Scale Invariant Feature Transform**

Basic idea:

- Take a 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

Adapted from slide by David Lowe
SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

Adapted from slide by David Lowe
Properties of SIFT

Extraordinarily robust matching technique

• Can handle changes in viewpoint
  – Up to about 60 degree out of plane rotation
• Can handle significant changes in illumination
  – Sometimes even day vs. night (below)
• Fast and efficient: can run in real time!
• Lots of code available
Working with SIFT descriptors

• One image yields:
  – $n$ 128-dimensional descriptors: each one is a histogram of the gradient orientations within a patch
    • $[n \times 128 \text{ matrix}]$
  – $n$ scale parameters specifying the size of each patch
    • $[n \times 1 \text{ vector}]$
  – $n$ orientation parameters specifying the angle of the patch
    • $[n \times 1 \text{ vector}]$
  – $n$ 2d points giving positions of the patches
    • $[n \times 2 \text{ matrix}]$
Scale Invariant Detectors

- **Harris-Laplacian**
  - Find local maximum of:
    - Harris corner detector in space (image coordinates)
    - Laplacian in scale

- **SIFT (Lowe)**
  - Find local maximum of:
    - Difference of Gaussians in space and scale

---

Scale Invariant Detectors

• Experimental evaluation of detectors w.r.t. scale change

Repeatability rate:
\[ \frac{\# \text{ correspondences}}{\# \text{ possible correspondences}} \]

Scale Invariant Detection: Summary

- **Given:** two images of the same scene with a large *scale difference* between them
- **Goal:** find *the same* interest points *independently* in each image
- **Solution:** search for *maxima* of suitable functions in *scale* and in *space* (over the image)

**Methods:**

1. **Harris-Laplacian** [Mikolajczyk & Schmid, ‘01]: maximize Laplacian over scale, Harris’ measure of corner response over the image
2. **SIFT** [Lowe, ‘04]: maximize Difference of Gaussians over scale and space
MATCHING
ICP
Feature descriptors

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

Lots of possibilities (this is a popular research area)

- Simple option: match square windows around the point
- State of the art approach: SIFT
Feature matching

Given a feature in $I_1$, how to 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 one with min distance
Feature distance

How to define the difference between two features \( f_1, f_2 \)?

- Simple approach is SSD\((f_1, f_2)\)
  - sum of square differences between entries of the two descriptors
  - can give good scores to very ambiguous (bad) matches
Feature distance

How to define the difference between two features $f_1$, $f_2$?

- Better approach: ratio distance $= \frac{\text{SSD}(f_1, f_2)}{\text{SSD}(f_1, f_2')}$
  - $f_2$ is best SSD match to $f_1$ in $I_2$
  - $f_2'$ is 2nd best SSD match to $f_1$ in $I_2$
  - gives small values for ambiguous matches
Eliminating bad matches

Throw out features with distance > threshold
  • How to choose the threshold?
True/false positives

The distance threshold affects performance

- True positives = # of detected matches that are correct
  - Suppose we want to maximize these—how to choose threshold?
- False positives = # of detected matches that are incorrect
  - Suppose we want to minimize these—how to choose threshold?
Evaluating the results

How can we measure the performance of a feature matcher?

![Diagram]

- The matcher correctly found a match
  - \# true positives matched\n  - \# true positives

- Features that really do have a match

- Features that really don’t have a match

- False positive rate

- True positive rate

- The matcher said yes when the right answer was no

# true positives matched
\# true positives

# false positives matched
\# true negatives

0.7
Evaluating the results

How can we measure the performance of a feature matcher?

ROC Curves
- Generated by counting # current/incorrect matches, for different thresholds
- Want to maximize area under the curve (AUC)
- Useful for comparing different feature matching methods
Iterated closest point

- **Iterative Closest Point (ICP)** is an algorithm employed to match two clouds of points. This matching is used to reconstruct 3D surfaces from different scans, to localize robots, to match bone models with measures in real-time, etc.

- The algorithm is very simple and is commonly used in *real-time*. It iteratively estimates the transformation (translation, rotation) between two raw scans.

- Inputs: two raw scans, initial estimation of the transformation, criteria for stopping the iteration.
- Output: refined transformation.

- Essentially the algorithm steps are:
  1. Associate points by the nearest neighbor criteria.
  2. Estimate the parameters using a mean square cost function.
  3. Transform the points using the estimated parameters.
  4. Iterate (re-associate the points and so on).
Matching Example