Fitting: Motivation

- We’ve learned how to detect edges, corners, blobs. 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.
Boundaries of Objects from Edges

But, low strength edges may be very important

Preprocessing Edge Images

Edge Tracking Methods

- Adjusting a priori boundaries:
  - Given: Approximate location of boundary
  - Task: Find accurate location of boundary

- Search for STRONG EDGES along normals to approximate boundary.
- Fit curve (e.g., polynomials) to strong edges.
Edge Tracking Methods

Divide and Conquer:

Given: Boundary lies between points A and B
Task: Find Boundary

- Connect A and B with Line
- Find strongest edge along line bisector
- Use edge point as break point
- Repeat

Fitting – The main idea

- Choose a parametric model to represent a set of features

Fitting lines to edges

- Choose a parametric model to represent a set of features
- Membership criterion is not local
  - Can’t tell whether a point belongs to a given model just by looking at that point
- Three main questions:
  - What model represents this set of features best?
  - Which of several model instances gets which feature?
  - How many model instances are there?
- Computational complexity is important
  - It is infeasible to examine every possible set of parameters and every possible combination of features
Example: Line fitting

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

- Wait, why aren’t we done just by running edge detection?

Slide credit: K. Grauman

Fitting: Issues

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

Case study: Line detection

Slide credit: K. Grauman

Fitting: Issues

- 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: RANSAC, Hough transform

- What if we’re not even sure it’s a line?
  - Model selection

Slide credit: S. Lazebnik

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

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

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

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

\[
\frac{dE}{dB} = 2X^TXB - 2X^TY = 0
\]

\[
X^T XB = X^T Y
\]

Normal equations: Least Squares solution to \(XB = Y\)

Slide credit: S. Lazebnik
Problem with “vertical” least squares

- Not rotation-invariant
- Fails completely for vertical lines

Total Least Squares

Distance between point \((x_i, y_i)\) and line \(ax + by = d\) with \(a^2 + b^2 = 1\): 
\[
E = \sum_{i=1}^{n} (ax_i + by_i - d)^2
\]

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

\[
\frac{dE}{dd} = \sum_{i=1}^{n} -2(a(x_i + by_i - d)) = 0
\]

\[
d = \frac{d}{n} \sum_{i=1}^{n} x_i + \frac{b}{n} \sum_{i=1}^{n} y_i = a \bar{x} + b \bar{y}
\]

\[
E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \left\| \begin{bmatrix} x_1 - \bar{x} \\ y_1 - \bar{y} \\ \vdots \\ x_n - \bar{x} \\ y_n - \bar{y} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right\|^2 = (UN)^T (UN)
\]

Solution to \((U^T U)N = 0\), subject to \(||N||^2 - 1\): eigenvector of \(U^T U\) associated with the smallest eigenvalue (least squares solution to homogeneous linear system \(UN = 0\))
Total Least Squares

\[ U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix}, \quad U^T U = \begin{bmatrix} \sum_{i=1}^{n} (x_i - \bar{x})^2 & \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^{n} (y_i - \bar{y})^2 \end{bmatrix} \]

second moment matrix

Least squares as likelihood maximization

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

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

point on the line

noise: normal direction

sampled from zero-mean Gaussian with std. dev. \( \sigma \)

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: Robustness to noise

• Least squares fit to the red points:

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 $i$th 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!
The error value is almost the same for every point and the fit is very poor.

Choosing the scale: Too small

Choosing the scale: Too large

Behaves much the same as least squares.

Robust estimation: Details

- Robust fitting is a nonlinear optimization problem that must be solved iteratively.
- Least squares solution can be used for initialization.
- Adaptive choice of scale: approx. 1.5 times median residual.

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.

- Outline
  - Choose a small subset of points uniformly at random
  - Fit a model to that subset
  - Find all remaining points that are “close” to the model and reject the rest as outliers
  - Do this many times and choose the best model.
1. Randomly select minimal subset of points

Least-squares fit
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
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

Uncontaminated sample

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

Slide credit: R. Raguram

RANSAC for line fitting

- Repeat \( N \) times:
  - Draw \( s \) points uniformly at random
  - Fit line to these \( s \) points
  - Find inliers to this line among the remaining points (i.e., points whose distance from the line is less than \( t \))
  - If there are \( d \) or more inliers, accept the line and refit using all inliers

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 \( p \) (e.g., 0.95)
  - 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., \( p=0.99 \)) (outlier ratio: \( c \))

Slide credit: M. Pollefeys
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 $p$ (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. $\sigma : \sigma^2 = 3.84 \sigma^2$
- Number of samples $N$
  - Choose $N$ so that, with probability $p$, at least one random sample is free from outliers (e.g. $p=0.99$) (outlier ratio: $e$)

\[
(1 - (1 - e)) \log(\frac{1}{(1 - e)}) = 1 - p
\]

$N = \log(1 - p) / \log(\frac{1}{(1 - e)})$

$e$ : prob. that any selected data point is an inlier
$1 - e$ : prob. of observing an outlier

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:
  - $N=\infty$, $sample\_count = 0$
  - While $N > sample\_count$
    - Choose a sample and count the number of inliers
    - Set $e = 1 - (number\_of\_inliers)/(total\_number\_of\_points)$
    - Recompute $N$ from $e$
      \[
      N = \log(1 - p) / \log(\frac{1}{(1 - e)})
      \]
    - Increment the $sample\_count$ by 1
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

A look into the past

• Leningrad during the blockade

Bing streetside images


Image alignment: Applications

- Panorama stitching
- Recognition of object instances

Image alignment: Challenges

- Small degree of overlap
- Intensity changes
- Occlusion, clutter

Image alignment

- Two broad approaches:
  - Direct (pixel-based) alignment
    - Search for alignment where most pixels agree
  - Feature-based alignment
    - Search for alignment where extracted features agree
    - Can be verified using pixel-based alignment

Alignment as fitting

Alignment: fitting a model to a transformation between pairs of features (matches) in two images

Find model $M$ that minimizes

$$\sum_i \text{residual}(x_i, M)$$

Find transformation $T$ that minimizes

$$\sum_i \text{residual}(T(x_i), x'_i)$$
2D transformation models

- Similarity
  (translation, scale, rotation)

- Affine

- Projective
  (homography)

Let’s start with affine transformations

- Simple fitting procedure (linear least squares)
- Approximates viewpoint changes for roughly planar objects and roughly orthographic cameras
- Can be used to initialize fitting for more complex models

Fitting an affine transformation

- Assume we know the correspondences, how do we get the transformation?

\[
\begin{align*}
\begin{bmatrix} x'_1 \\ y'_1 \end{bmatrix} &= \begin{bmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \\
\end{align*}
\]

- Linear system with six unknowns
- Each match gives us two linearly independent equations; need at least three to solve for the transformation parameters
Beyond affine transformations

- **Homography**: plane projective transformation (transformation taking a quad to another arbitrary quad)

Recap: RANSAC

- An example of a “voting”-based fitting scheme
- Each hypothesis gets voted on by each data point, best hypothesis wins
- There are many other types of voting schemes
  - e.g., Hough transforms...

Voting schemes

- 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 \(y = mx + b\)

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.
### Polar representation for lines

Issues with usual $(m,b)$ parameter space: can take on infinite values, undefined for vertical lines.

- $d$: perpendicular distance from line to origin
- $\theta$: angle the perpendicular makes with the x-axis

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

Point in image space $\rightarrow$ sinusoid segment in Hough 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 $\theta = \theta_{\text{min}}$ 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 $d = x \cos \theta - y \sin \theta$
Effect of noise

- Peak gets fuzzy and hard to locate

Random points

- Uniform noise can lead to spurious peaks in the array
Random points

- As the level of uniform noise increases, the maximum number of votes increases too:

![Graph showing the relationship between number of noise points and maximum number of votes.](image)

Dealing with noise

- Choose a good grid / discretization
  - Too coarse: large votes obtained when too many different lines correspond to a single bucket
  - Too fine: miss lines because some points that are not exactly collinear cast votes for different buckets
- Increment neighboring bins (smoothing in accumulator array)
- Try to get rid of irrelevant features
  - Take only edge points with significant gradient magnitude

Incorporating image gradients

- Recall: when we detect an edge point, we also know its gradient direction
- But this means that the line is uniquely determined!

- Modified Hough transform:
  
  For each edge point \((x,y)\)
  
  \[
  \theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)
  \]

Hough transform for circles

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

- For a fixed radius \(r\), unknown gradient direction

![Diagram showing the Hough transform for circles.](image)
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 \(r\), unknown gradient direction

![Image of Hough transform for circles](image)

Slide credit: K. Grauman

81

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 \(r\), unknown gradient direction

![Image of Hough transform for circles](image)

Slide credit: K. Grauman

82

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)

Slide credit: K. Grauman

83

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)

Slide credit: K. Grauman

84
Hough transform for circles

For every edge pixel (x,y):
   For each possible radius value r:
      For each possible gradient direction \( \theta \):
         // or use estimated gradient at (x,y)
         \[ a = x - r \cos(\theta) \] // column
         \[ b = y + r \sin(\theta) \] // row
         \( H[a,b,r] += 1 \)
   end
end

Time complexity per edge?

- Check out online demo: [http://www.markschulze.net/java/hough/](http://www.markschulze.net/java/hough/)

Example: detecting circles with Hough

Original | Edges | Votes: Penny
--- | --- | ---

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

Example: detecting circles with Hough

Original | Edges | Votes: Quarter
--- | --- | ---

Application: iris detection

* Hemerson Pistori and Eduardo Rocha Costa
Application: iris detection


Hough circles vs. Laplacian blobs

Original images
Laplacian circles
Hough-like circles

F. Jurie and C. Schmid, Scale-invariant shape features for recognition of object categories, CVPR 2004

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

- We want to find a template defined by its reference point (center) and several distinct types of landmark points in stable spatial configuration
Generalized Hough transform

• Template representation: for each type of landmark point, store all possible displacement vectors towards the center

Application in recognition

• 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

Application in recognition

• 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
Implicit shape models: Training

1. Build codebook of patches around extracted interest points using clustering (more on this later)

2. Map the patch around each interest point to closest codebook entry

3. For each codebook entry, store all positions it was found, relative to object center

---

Implicit shape models: Testing

1. Given test image, extract patches, match to codebook entry

2. Cast votes for possible positions of object center

3. Search for maxima in voting space

4. Extract weighted segmentation mask based on stored masks for the codebook occurrences
Additional examples


Implicit shape models: Details

- Supervised training
  - Need reference location and segmentation mask for each training car
- Voting space is continuous, not discrete
  - Clustering algorithm needed to find maxima
- How about dealing with scale changes?
  - Option 1: search a range of scales, as in Hough transform for circles
  - Option 2: use scale-covariant interest points
- Verification stage is very important
  - Once we have a location hypothesis, we can overlay a more detailed template over the image and compare pixel-by-pixel, transfer segmentation masks, etc.