Fitting: Voting and the Hough Transform (part 2)

Last time

- Fitting problems require finding any supporting evidence for a model, even within clutter and missing features.
  - associate features with an explicit model
- Voting approaches, such as the Hough transform, find likely model parameters without searching all combinations of features.
  - Hough transform approach for lines, circles, ..., arbitrary shapes defined by a set of boundary points, recognition from patches.

Outline

- Last time:
  - Fitting: voting and the Hough transform
  - For lines
  - For circles
- Today:
  - Review of Hough circles
  - Generalized Hough algorithm for any shape
  - Background subtraction

Fitting

- Want to associate a model with observed features

Recall -- Fitting: Main idea

- 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

Recall--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

Slide credit: Steve Seitz

Finding lines in an image: Hough space

What are the line parameters for the line that contains both (x₀, y₀) and (x₁, y₁)?
- It is the intersection of the lines b = -x₀m + y₀ and b = -x₁m + y₁

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.

Extensions

Extension 1: Use the image gradient
1. same
2. for each edge point (x,y) in the image
   - θ = gradient at (x,y)
   - d = x cos θ − y sin θ
   - H(d, θ) += 1
3. same
4. same
(Reduces degrees of freedom)

Source: Steve Seitz
**Extensions**

Extension 1: Use the image gradient
1. same
2. for each edge point \([x,y]\) in the image
   compute unique \((d, \theta)\) based on image gradient at \((x,y)\)
   \(H[d, \theta] \leftarrow 1\)
3. same
4. same
(Reduces degrees of freedom)

Extension 2
- give more votes for stronger edges (use magnitude of gradient)

Extension 3
- change the sampling of \((d, \theta)\) to give more/less resolution
- The same procedure can be used with circles, squares, or any other shape...

---

**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

Intersection: most votes for center occur here.

---

**Hough transform for circles**

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

---

**Hough transform for circles**

- Circle: center \((a, b)\) and radius \(r\)
  \[(x - a)^2 + (y - b)^2 = r^2\]
- For an unknown radius \(r\), **known** gradient direction
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 \((x,y)\)
         \[ a = x + r \cos(\theta) \text{ } // \text{column} \]
         \[ b = y - r \sin(\theta) \text{ } // \text{row} \]
         \[ H[a,b,r] += 1 \]
   
\[
\text{Time complexity per edge pixel?}
\]

* Check out online demo: http://www.markschulze.net/java/hough/

Example: detecting circles with Hough

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

Example: iris detection

Voting: practical tips

- Minimize irrelevant tokens first
- Choose a good grid / discretization
- Vote for neighbors, also (smoothing in accumulator array)
- Use direction of edge to reduce parameters by 1
- To read back which points voted for “winning” peaks, keep tags on the votes.
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

Outline

• Last time:
  - Fitting: voting and the Hough transform
    • For lines
    • For circles

• Today:
  - Review of Hough circles
    - Generalized Hough algorithm for any shape
  - Background subtraction

Generalized Hough Transform

• What if we want to detect arbitrary shapes?

Intuition:

Now suppose those colors encode gradient directions...

Generalized Hough Transform

Detection procedure:
- For each edge point:
  - Use its gradient orientation $\theta$ to index into stored table
  - Use retrieved $r$ vectors to vote for reference point

Assuming translation is the only transformation here, i.e., orientation and scale are fixed.

Generalized Hough for object detection

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

Source: L. Lazebnik

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
Generalized Hough for object detection

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

Example: Results on Cows

Original image

Interest points

Matched patches

Votes

1st hypothesis

Source: L. Lazebnik

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

K. Grauman, B. Leibe

Perceptual and Sensory Augmented Computing
Visual Object Recognition Tutorial

Perceptual and Sensory Augmented Computing
Visual Object Recognition Tutorial
Outline

• Last time:
  – Fitting: voting and the Hough transform
    • For lines
    • For circles
• Today:
  – Review of Hough circles
  – Generalized Hough algorithm for any shape
  – Background subtraction

Video as an “Image Stack”

Can look at video data as a spatio-temporal volume
• If camera is stationary, each line through time corresponds to a single ray in space

Input Video

Average Image
Background Subtraction

- Simple techniques can do ok with static camera
- But hard to do perfectly
- Widely used:
  - Traffic monitoring (counting vehicles, detecting & tracking vehicles, pedestrians),
  - Human action recognition (run, walk, jump, squat),
  - Human-computer interaction
  - Object tracking

**Motivation**

- In most cases, objects are of interest, not the scene.
- Makes our life easier: less processing costs, and less room for error.

---

**Simple Approach**

1. Estimate the background for time $t$.
2. Subtract the estimated background from the input frame.
3. Apply a threshold, $Th$, to the absolute difference to get the foreground mask.

**Frame Differencing**

- Background is estimated to be the previous frame.
- Background subtraction equation then becomes:
  
  $$ B(x, y, t) = I(x, y, t - 1) $$

  $$ |I(x, y, t) - I(x, y, t - 1)| > Th $$

- Depending on the object structure, speed, frame rate and global threshold, this approach may or may not be useful (usually not).

**Mean Filter**

- In this case the background is the mean of the previous $n$ frames:
  
  $$ B(x, y, t) = \frac{1}{n} \sum_{i=0}^{n-1} I(x, y, t - i) $$

  $$ |I(x, y, t) - \frac{1}{n} \sum_{i=0}^{n-1} I(x, y, t - i)| > Th $$

- For $n = 10$:
  
  Estimated Background
  
  Foreground Mask
Median Filter

- Assuming that the background is more likely to appear in a scene, we can use the median of the previous \( n \) frames as the background model:
  
  \[
  B(x, y, t) = \text{median}(I(x, y, t - i))
  \]
  
  \[
  |I(x, y, t) - \text{median}(I(x, y, t - i))| > Th \text{ where } i \in \{0, \ldots, n - 1\}.
  \]

- For \( n = 10 \):

  ![Estimated Background](image1)
  ![Foreground Mask](image2)

### Average/Median Image

![Image](image3)

### Background Subtraction

![Image](image4)

### Pros and cons

**Advantages:**
- Extremely easy to implement and use!
- All pretty fast.
- Corresponding background models need not be constant, they change over time.

**Disadvantages:**
- Accuracy of frame differencing depends on object speed and frame rate
- Median background model: relatively high memory requirements.
- Setting global threshold \( Th \)...

*When will this basic approach fail?*

### Background mixture models

**Idea:** model each background pixel with a mixture of Gaussians; update its parameters over time.

![Image](image5)

### Coming up

- Deformable contours