# Fitting

Thursday, Sept 24

# Last time

- Grouping / segmentation to identify coherent image regions
  - – Clustering algorithms
    - K-means
    - Graph cuts; normalized cuts criterion
  - – Feature spaces
    - Color, intensity, texture, position…

# Review

- What is the objective function for k-means; that is, what does the algorithm try to minimize?

$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} ||p - c_i||^2$$

- The best assignment of points to clusters would minimize the total distances between points and their assigned (nearest) centers.

# Review

- Assuming we use a fully connected graph, what is the time complexity of computing the affinities for a graph cuts-based segmentation?

# Fitting

- Want to associate a model with observed features

[Fig from Marszalek & Schmid, 2007]

For example, the model could be a line, a circle, or an arbitrary shape.

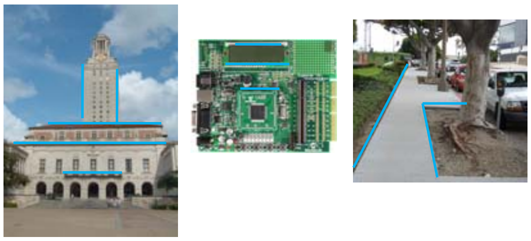
## Fitting

- 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

Source: L. Lazebnik

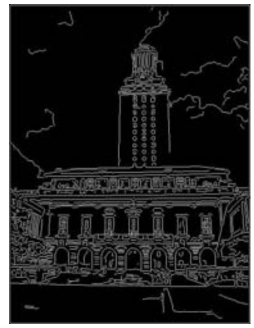
# Example: Line fitting

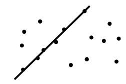
- Why fit lines?
  Many objects characterized by presence of straight lines



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

# Difficulty of line fitting





- **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?

# 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.
- Ok if some features not observed, as model can span multiple fragments.

# Fitting lines

- 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

  Main idea:
  1. Record all possible lines on which each edge point lies.
  2. Look for lines that get many votes.

# Finding lines in an image: Hough space

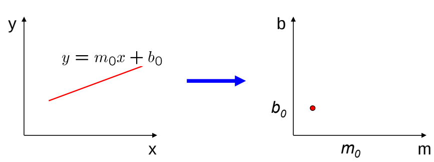



image space Hough (parameter) space

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

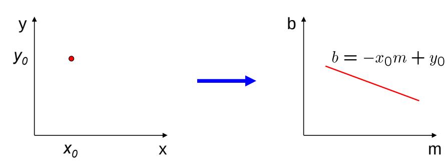



image space Hough (parameter) space

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 = $-x_0 m + y_0$
  - this is a line in Hough space

Slide credit: Steve Seitz

## Finding lines in an image: Hough space

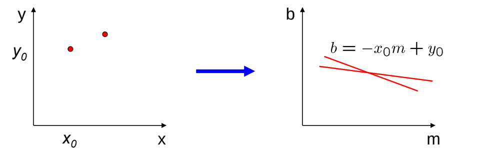



image space

Hough (parameter) 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 $b = -x_0 m + y_0$
  - this is a line in Hough space

## Finding lines in an image: Hough space

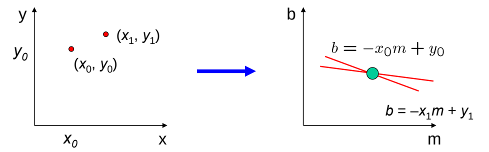



image space

Hough (parameter) 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 $b = -x_0 m + y_0$ and $b = -x_1 m + y_1$

## Finding lines in an image: Hough algorithm

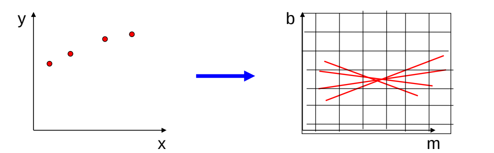



image space

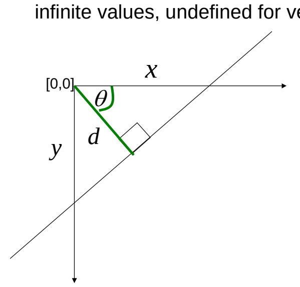
Hough (parameter) space

**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

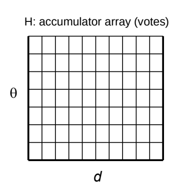
$$x\cos\theta - y\sin\theta = d$$

Point in image space → 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, θ]=0
2. for each edge point I[x,y] in the image
   for θ = 0 to 180 // some quantization
   $d = x\cos\theta - y\sin\theta$
   H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by $d = x\cos\theta - y\sin\theta$

[Hough line demo]

Time complexity (in terms of number of votes)?

Source: Steve Seitz

## Example: Hough transform for straight lines

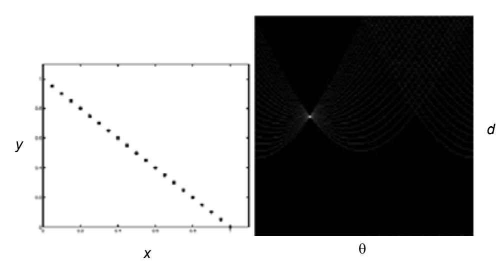



**Image space edge coordinates**

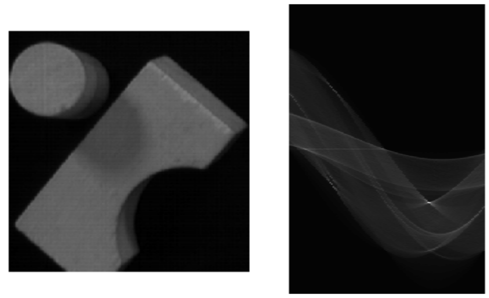
**Votes**

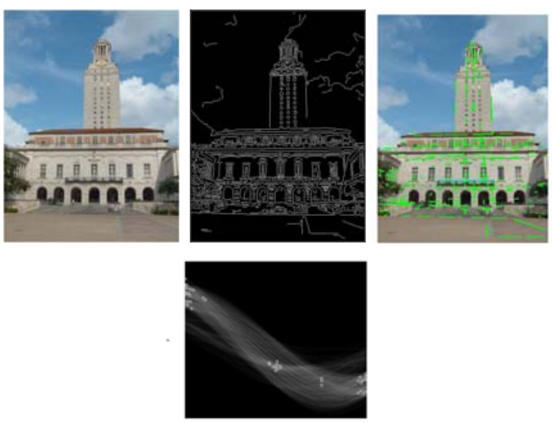
Bright value = high vote count
Black = no votes

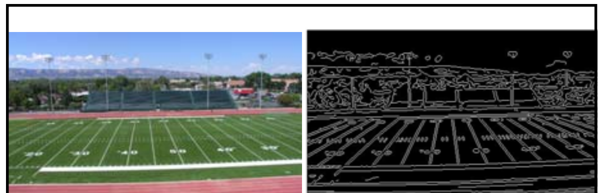
## Example: Hough transform for straight lines

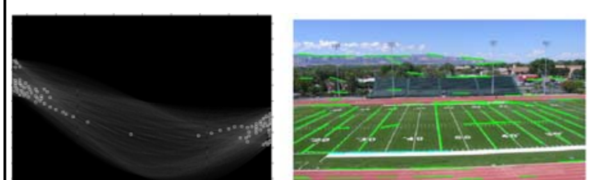
Square :

## Example: Hough transform for straight lines









Showing longest segments found

## Impact of noise on Hough

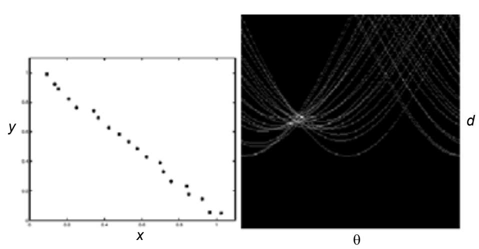



**Image space edge coordinates**

**Votes**

What difficulty does this present for an implementation?

## Impact of noise on Hough

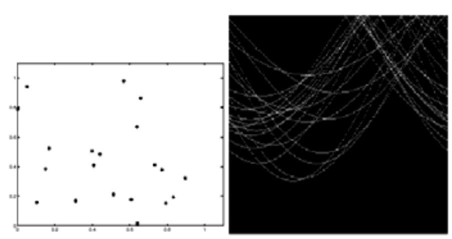


**Image space edge coordinates**

**Votes**

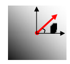
Here, everything appears to be “noise”, or random edge points, but we still see peaks in the vote space.

## Extensions

Extension 1: Use the image gradient

1. same
2. for each edge point I[x,y] in the image
   θ = gradient at (x,y)
   $d = x\cos\theta - y\sin\theta$
   H[d, θ] += 1
3. same
4. same

(Reduces degrees of freedom)



$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right)$

## Extensions

Extension 1: Use the image gradient

1. same
2. for each edge point I[x,y] in the image
   compute unique (d, θ) based on image gradient at (x,y)
   H[d, θ] += 1
3. same
4. same

(Reduces degrees of freedom)

Extension 2

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

Extension 3

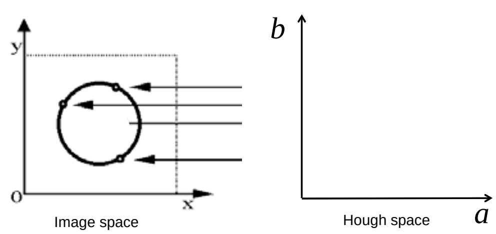
- change the sampling of (d, θ) to give more/less resolution

Extension 4

- The same procedure can be used with circles, squares, or any other shape…

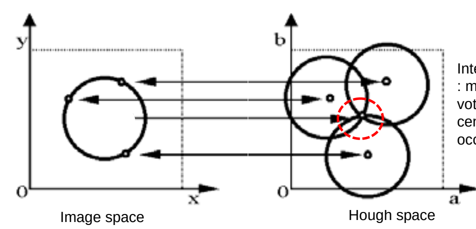
# Hough transform for circles

- Circle: center (a,b) and radius r
  $$(x_i - a)^2 + (y_i - b)^2 = r^2$$
- For a fixed radius r, unknown gradient direction




# Hough transform for circles

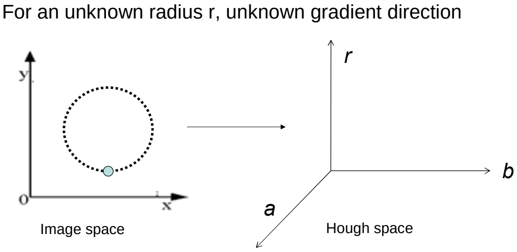
- Circle: center (a,b) and radius r
  $$(x_i - a)^2 + (y_i - b)^2 = r^2$$
- For a fixed radius r, unknown gradient direction




Intersection: most votes for center occur here.

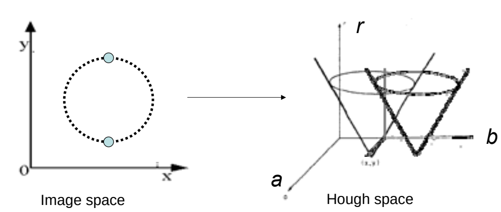
# Hough transform for circles

- 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




# 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



# 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

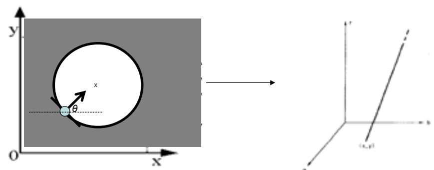


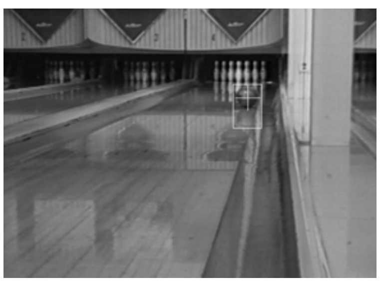

Image space

Hough space

# Hough transform for circles

For every edge pixel (*x,y*) :
  For each possible radius value *r*:
    For each possible gradient direction *θ: // or use estimated gradient*
    *a* = *x* – *r* cos(*θ*)
    *b* = *y* + *r* sin(*θ*)
    H[*a,b,r*] += 1
  end
end

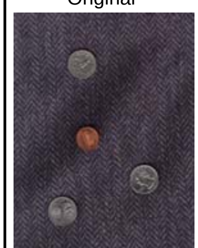
# Example: detecting circles with Hough

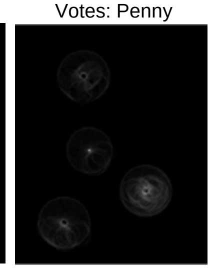


Crosshair indicates results of Hough transform, bounding box found via motion differencing.

# Example: detecting circles with Hough

Original | Edges | Votes: Penny

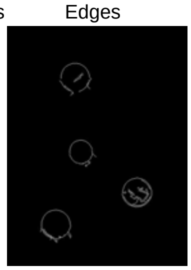


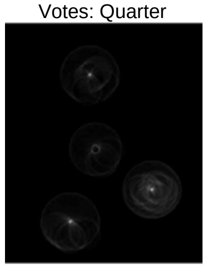


Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

# Example: detecting circles with Hough

Combined detections / Original | Edges | Votes: Quarter





Coin finding sample images from: Vivek Kwatra

# Voting: practical tips

- Minimize irrelevant tokens first (take edge points with significant gradient magnitude)
- 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
- Vote for neighbors, also (smoothing in accumulator array)
- Utilize direction of edge to reduce free 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
- 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: hard to pick a good grid size

# Generalized Hough transform

- What if want to detect arbitrary shapes defined by boundary points and a reference point?

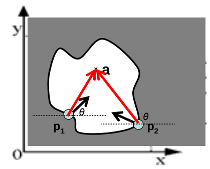



Image space

At each boundary point, compute displacement vector: $\mathbf{r} = \mathbf{a} - \mathbf{p}_i$.

For a given model shape: store these vectors in a table indexed by gradient orientation $\theta$.

[Dana H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, 1980]

# Generalized Hough transform

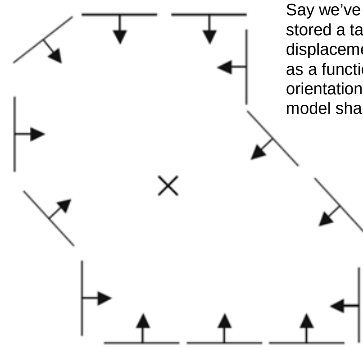
To *detect* the model shape in a new image:

- For each edge point
  - Index into table with its gradient orientation $\theta$
  - Use retrieved **r** vectors to vote for position of reference point
- Peak in this Hough space is reference point with most supporting edges

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

## Example

Say we've already stored a table of displacement vectors as a function of edge orientation for this model shape.

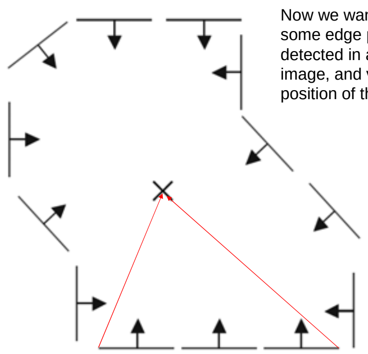


model shape

Source: L. Lazebnik

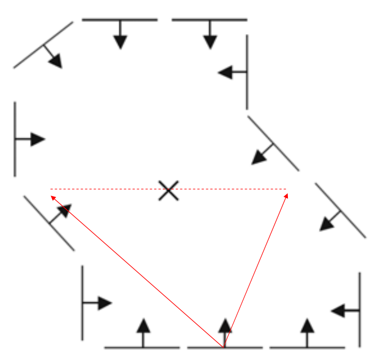
## Example

Now we want to look at some edge points detected in a *new* image, and vote on the position of that shape.



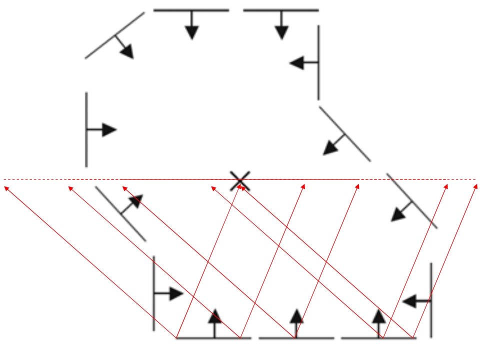
displacement vectors for model points

## Example



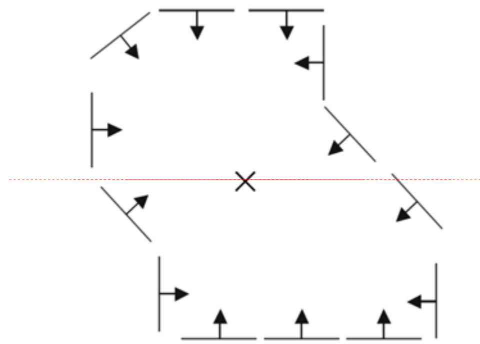
range of voting locations for test point

## Example



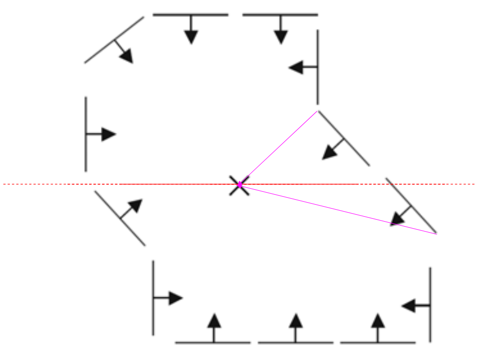
range of voting locations for test point

## Example



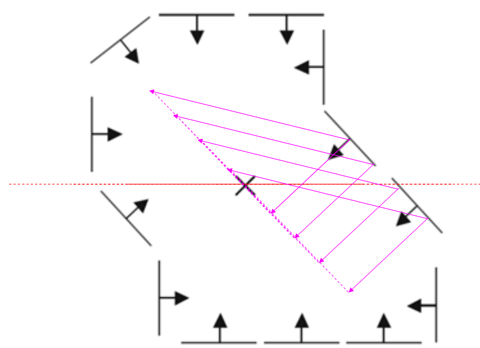
votes for points with $\theta = \uparrow$

## Example



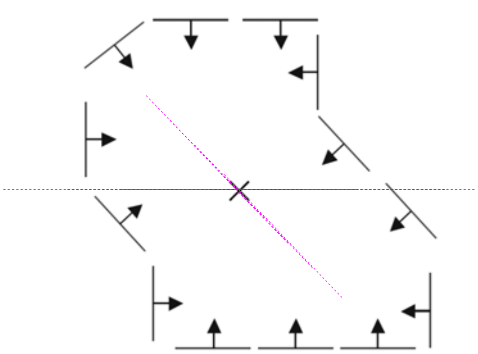
displacement vectors for model points

## Example



range of voting locations for test point

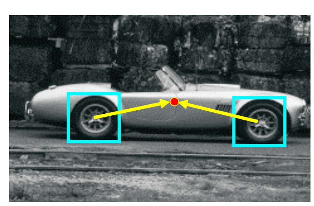
## Example



votes for points with $\theta = \swarrow$

## Application in recognition

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



training image

visual codeword with displacement vectors

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

## Application in recognition

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

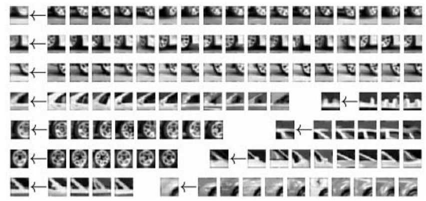
test image

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

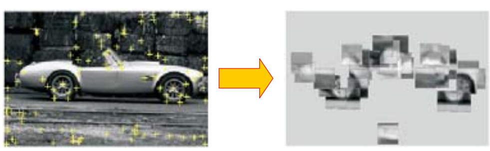
## Implicit shape models: Training

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



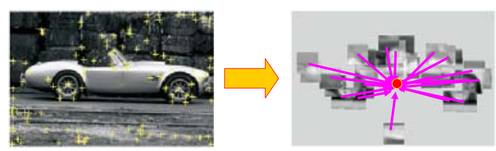
## Implicit shape models: Training

1. Build codebook of patches around extracted interest points using clustering
2. Map the patch around each interest point to closest codebook entry



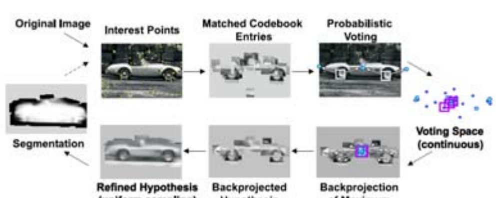
## Implicit shape models: **Training**

1. Build codebook of patches around extracted interest points using clustering
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




## Summary

- **Grouping/segmentation** useful to make a compact representation and merge similar features
  - associate features based on defined similarity measure and clustering objective
- **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**, make it possible to 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.

## Next

- Tuesday 9/29/08
  - Background modeling and Bayesian classification
  - Guest lecture by Professor Aggarwal
- Thursday 10/2/08
  - No class
- Next week: my office hours cancelled
- Harshdeep still available