PITT | SWANSON ENGINEERING
ELECTRICAL & COMPUTER

ECE 1390/2390
Image Processing and Computer Vision – Fall 2021

Lecture 5 and 6: Hough transform

Ahmed Dallal
Assistant Professor of ECE
University of Pittsburgh

Reading

- FP 10.1

Introduction: model fitting

Now some "real" vision…

- So far, we applied operators/masks/kernels to images to produce new image

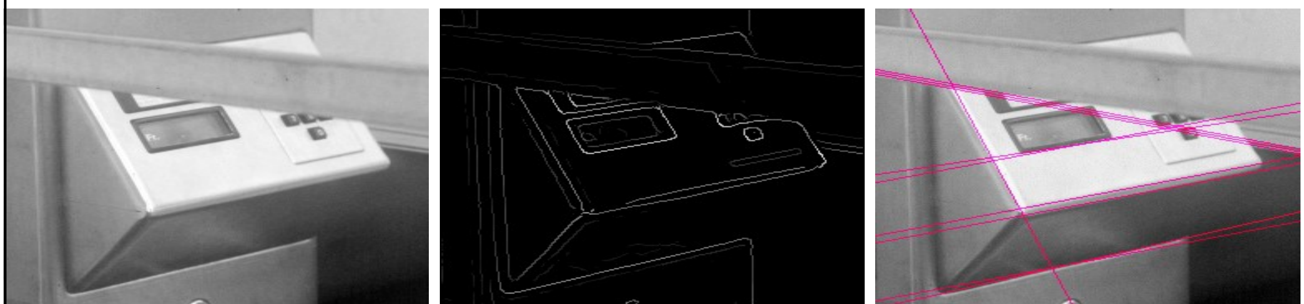
Image processing: $F : I(x, y) \longrightarrow I'(x, y)$

- Now real vision:

$$F : I(x, y) \longrightarrow \text{good stuff}$$

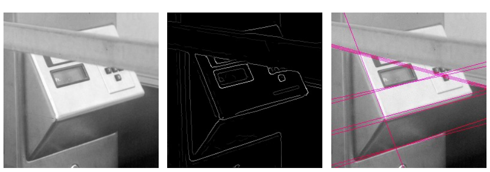
Fitting a model

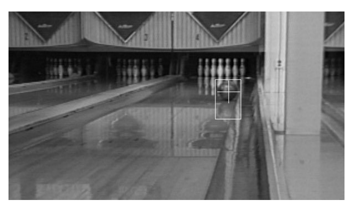
- Want to associate a model with observed features

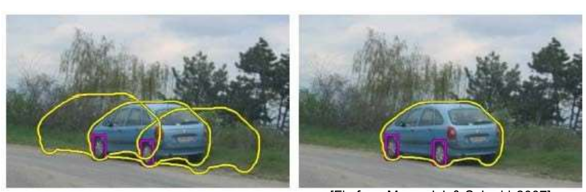


Fitting a model

- 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.

Parametric model

- A *parametric* model can represent a class of instances where each is defined by a value of the parameters.
- Examples include lines, or circles, or even a parameterized template.

Fitting a parametric model

- Choose a parametric model to represent a set of features
- Membership criterion is not local:

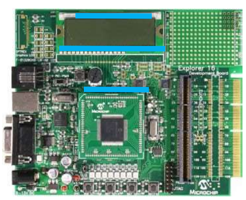
  Can't tell whether a point in the image belongs to a given model just by looking at that point
- Computational complexity is important

  Not feasible to examine possible parameter setting

Example: Line fitting

- Why fit lines?

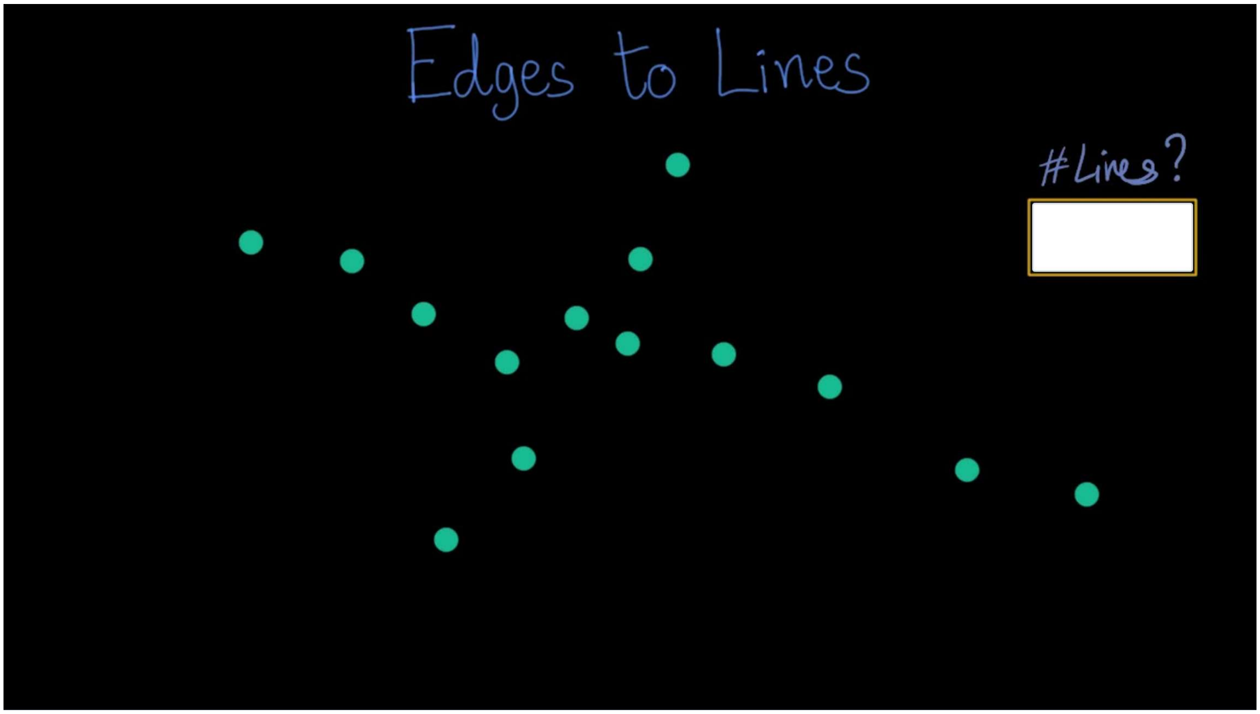
Many objects characterized by presence of straight lines



- 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 possible models or all combinations of features (e.g. edge pixels) 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.

1. Cycle through features, each casting votes for model parameters.
2. Look for model parameters that receive a lot of votes.

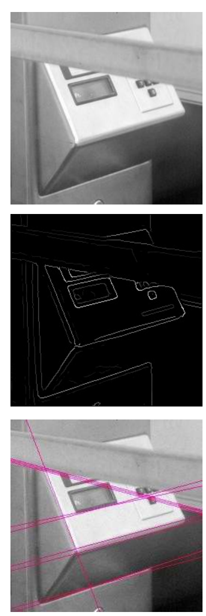
Voting – why it works

- 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

To fit lines we need to answer a few questions:

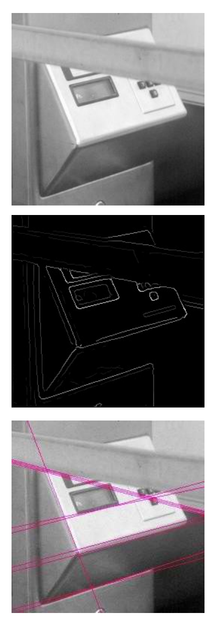
- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?



Hough transform: Lines

Fitting lines

Hough Transform is a voting technique that can be used to answer all of these



Main idea

1. Each edge point votes for compatible lines.
   - 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

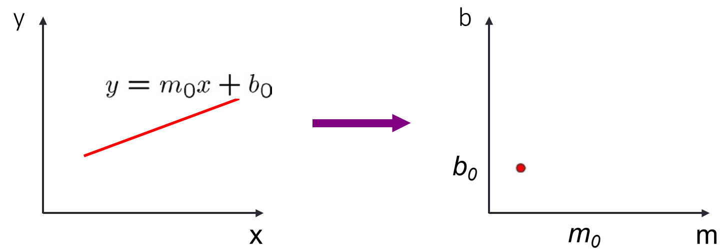


image space | Hough (parameter) space

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

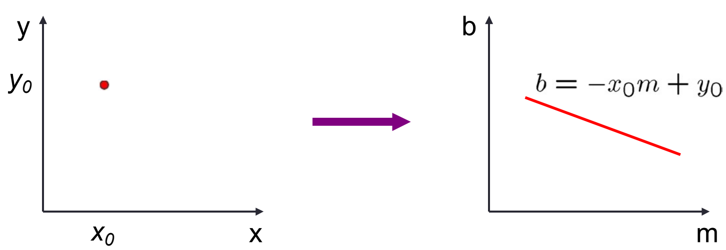


image space | Hough (parameter) space

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

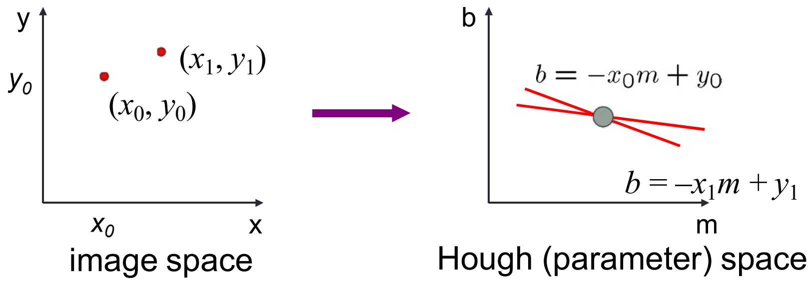
What does a point (x_0, y_0) in the image space map to?

Answer: the solutions of $b = -x_0 m + y_0$

this is a line in Hough space

$$y_0 = m x_0 + b \longrightarrow b = -x_0 m + y_0$$

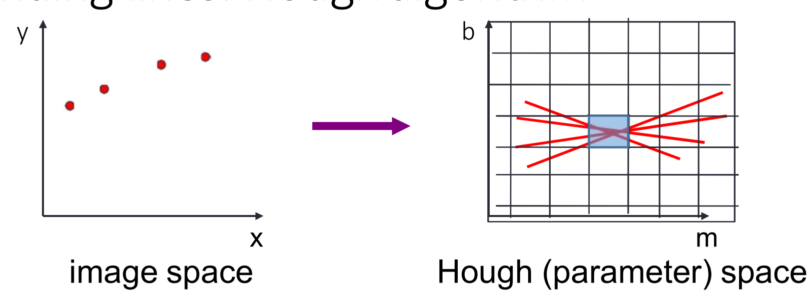
Finding lines in an image: Hough transform



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: Hough algorithm



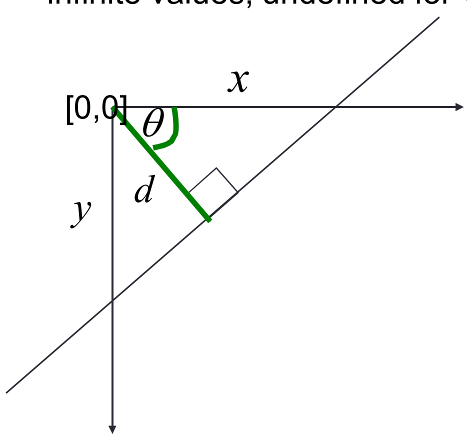
- How can we use this to find the most likely parameters (m,b) for the most prominent line in the image space?
- Let each edge point in image space vote for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

Line representation issues

- Before we implement this we need to rethink our representations of lines.
- As you may remember, there are issues with the $y = mx + b$ representation of line.
- In particular, undefined for vertical lines with m being infinity. So we use a more robust polar representation of lines.

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

θ : angle the perpendicular makes with the x-axis

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

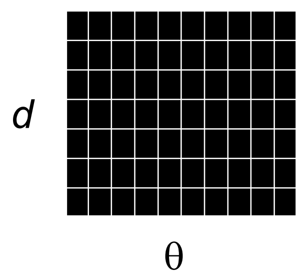
Point in image space → sinusoid segment in Hough space

Hough transform algorithm

Using the polar parameterization:

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

And a *Hough Accumulator Array (keeps the votes)*



Source: Steve Seitz

Basic Hough transform algorithm

1. Initialize H[d, θ]=0
2. For each ***edge*** point (x_0, y_0) in the image
   - for θ =-90 to +90 // some quantization;

     $$d = x_0\cos\theta - y_0\sin\theta$$

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

Source: Steve Seitz

Complexity of the Hough transform

Space complexity? k^n (n dimensions, k bins each)

Time complexity (in terms of number of voting elements)? Voting proportional to number of edge pixels

Hough example

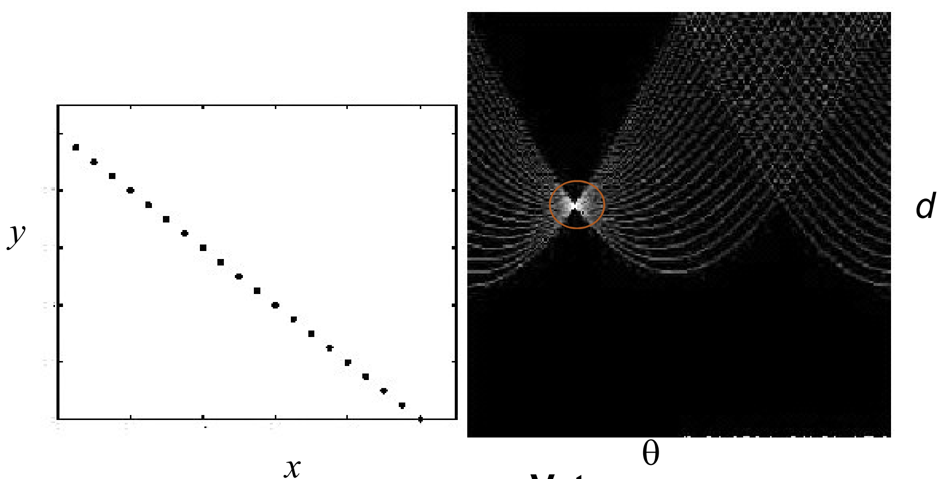
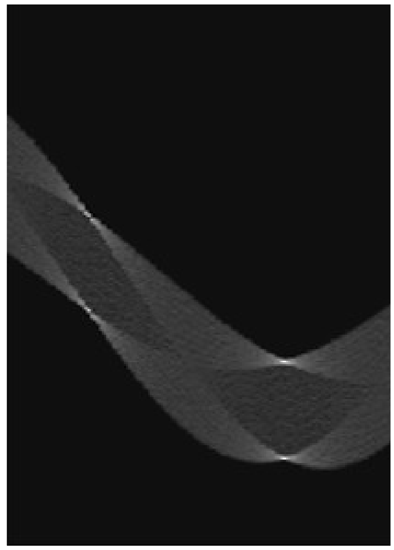


Image space edge coordinates

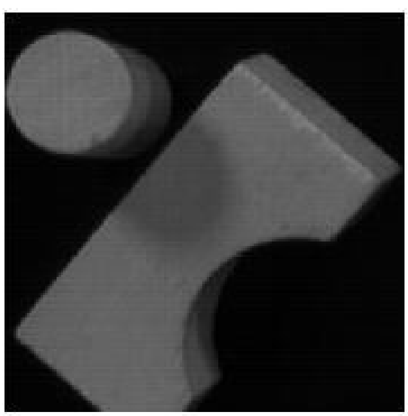
Votes
Bright value = high vote count
Black = no votes

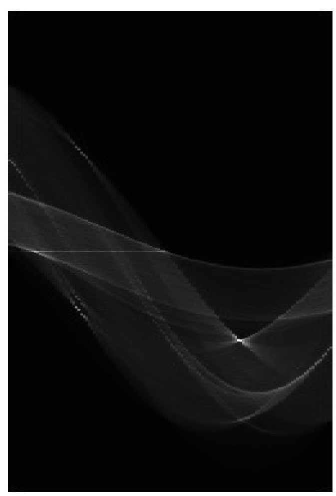
Example: Hough transform of a square

Square :



Hough transform of blocks scene





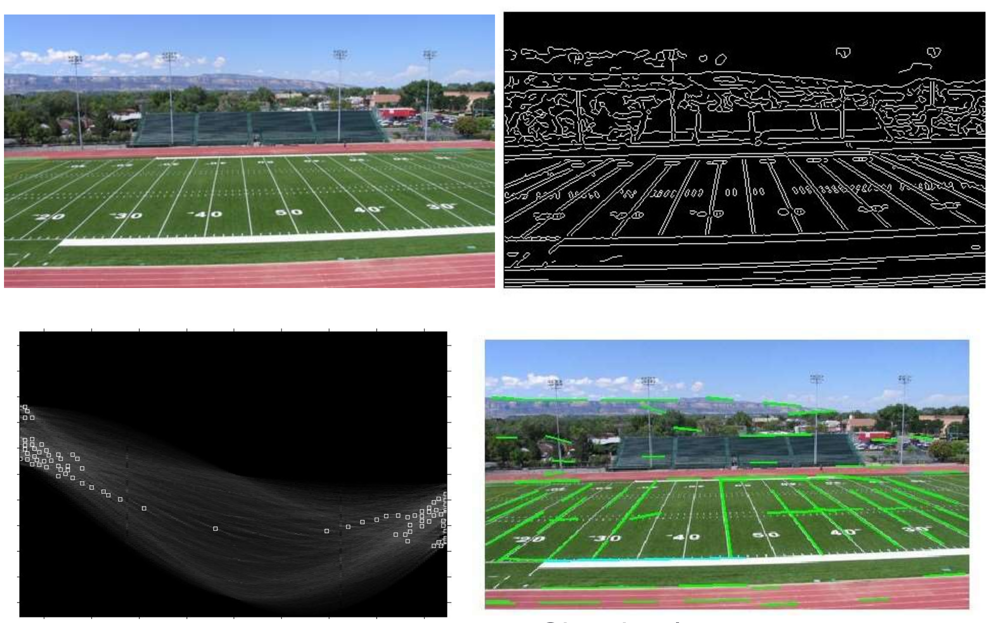
Hough Demo

```
% Hough Demo
pkg load image; % Octave only

%% Load image, convert to grayscale and apply Canny operator to find edge pixels img
= imread('shapes.png');
grays = rgb2gray(img);
edges = edge(grays, 'canny');

%% Apply Hough transform to find candidate lines
[accum theta rho] = hough(edges); % Matlab (use houghtf in Octave)
figure, imagesc(accum, 'XData', theta, 'YData', rho), title('Hough accumulator');

%% Find peaks in the Hough accumulator matrix
peaks = houghpeaks(accum, 100); % Matlab (use immaximas in Octave) hold
on; plot(theta(peaks(:, 2)), rho(peaks(:, 1)), 'rs'); hold off;
```

Hough Demo(contd.)

```
%% Find lines (segments) in the image
line_segs = houghlines(edges, theta, rho, peaks); % Matlab
figure, imshow(img), title('Line segments');
hold on;
for k = 1:length(line_segs)
    endpoints = [line_segs(k).point1; line_segs(k).point2];
    plot(endpoints(:, 1), endpoints(:, 2), 'LineWidth', 2, 'Color', 'green');
end
hold off;

%% Play with parameters to get more precise lines
peaks = houghpeaks(accum, 100, 'Threshold', ceil(0.6 * max(accum(:))), 'NHoodSize', [5 5]);
line_segs = houghlines(edges, theta, rho, peaks, 'FillGap', 50, 'MinLength', 100);
```

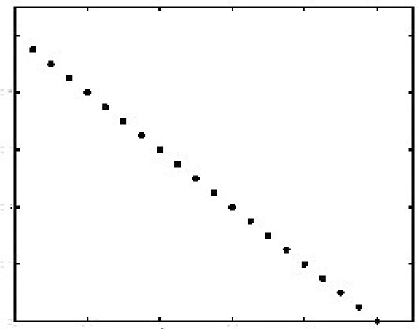
Hough transform: lines - considerations and extensions



Showing longest segments found

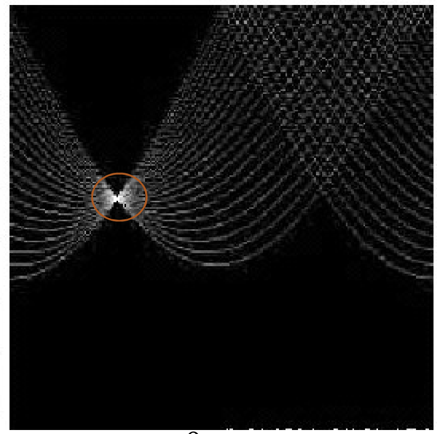
Hough example - revisited

y



x

Image space edge coordinates



d

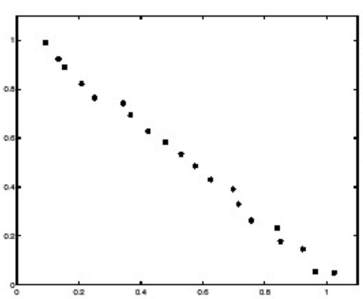
θ

Votes

Bright value = high vote count
Black = no votes

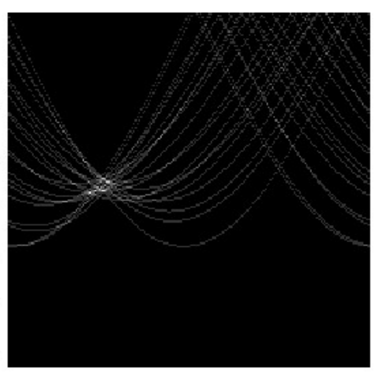
Impact of noise on Hough

y



x

Image space edge coordinates

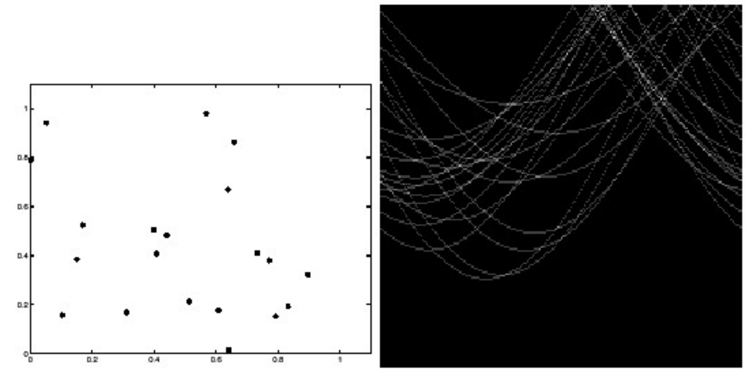


d

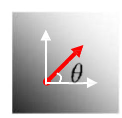
θ

Votes

Impact of more noise on Hough



Extensions – using the gradient



1. Initialize H[d, θ]=0
2. For each ***edge*** point (x_0, y_0) in the image
   θ = gradient at (x_0,y_0)
   $d = x_0 \cos\theta - y_0 \sin\theta$
   H[d, θ] += 1
3. Find the value(s) of (d^*, θ^*) where $H[d^*, \theta^*]$ is maximum
4. The detected line in the image is given by $d^* = x \cos\theta^* - y \sin\theta^*$

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

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

Extensions

Extension 2

Give more votes for stronger edges

Extension 3

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

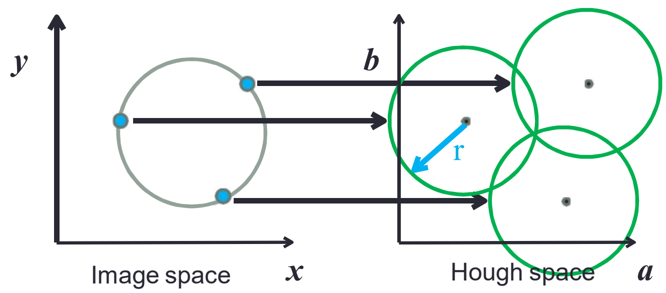
Extension 4

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

Hough transform: Circles

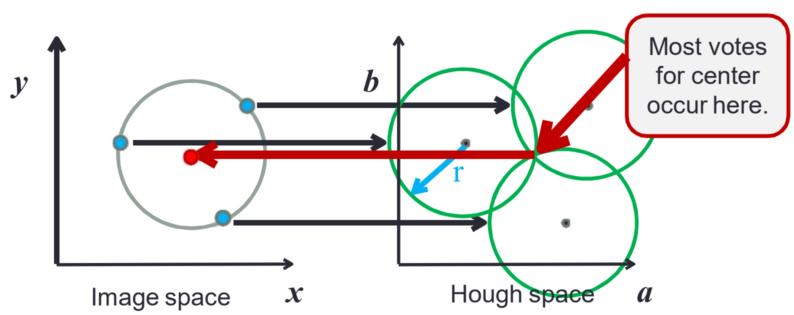
Hough transform for circles

- 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:



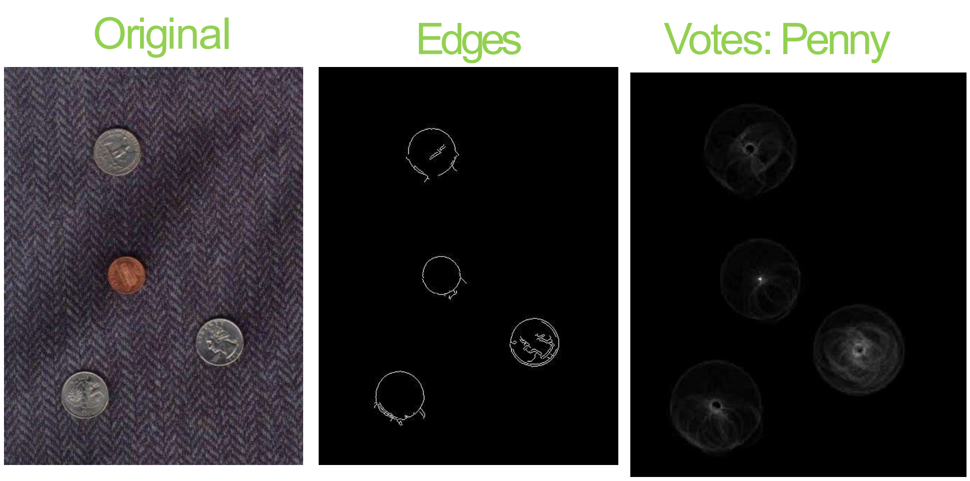
Hough transform for circles

- 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:



Example: detecting circles with Hough

Example: detecting circles with Hough

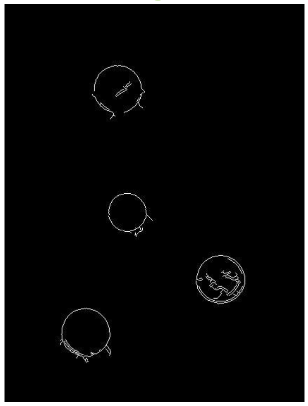


Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

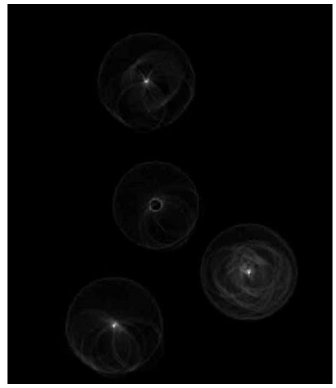
Example: detecting circles with Hough

Original

Edges



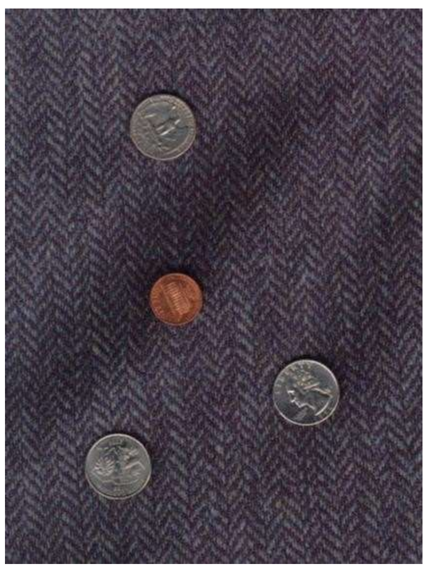
Votes: Quarter



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

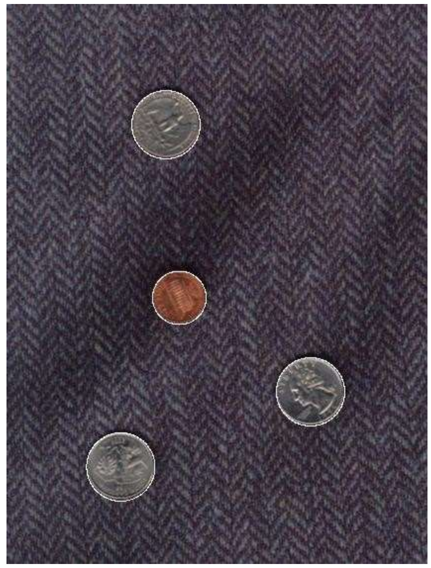
Example: detecting circles with Hough

Original



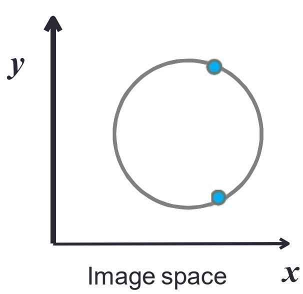
Example: detecting circles with Hough

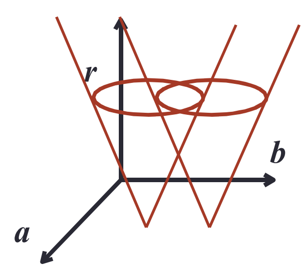
Combined detections



Hough transform for circles

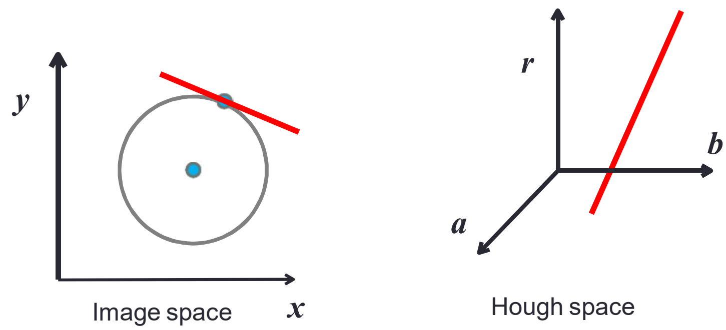
- Circle: center (a,b) and radius r $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For ***unknown*** radius r, no gradient:





Hough transform for circles

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



Hough transform for circles

1. For every edge pixel (x_0,y_0)
2. For each possible radius value r:
3. For each possible gradient direction θ:
   %% or use estimated gradient
4. $a = x_0 - r\cos(\theta)$
5. $b = y_0 + r\sin(\theta)$
6. H[a,b,r] += 1
7. end
8. end
9. end

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 bin
  - Too fine: miss lines because some points that are not exactly collinear (slightly misaligned) cast votes for different bins

Parametrized Hough transform: Practical tips and wrap up

Voting: practical tips

- Vote for neighboring bins (like 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

Parameterized 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

Parameterized Hough transform: pros and cons

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

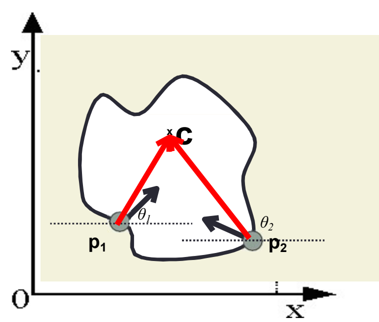
Generalized Hough transform: Then and now

Generalized Hough transform

- Non-analytic models
  - *Parameters express variation in pose or scale of fixed but arbitrary shape (that was then)*
- Visual code-word based features
  - *Not edges but detected templates learned from models (this is "now")*

Generalized Hough transform

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



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

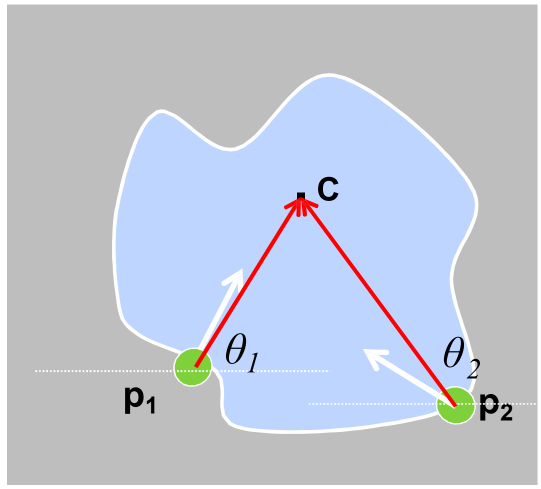
For a given model shape: store these vectors in a table indexed by gradient orientation θ.

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

Generalized Hough transform

Training: build a Hough table

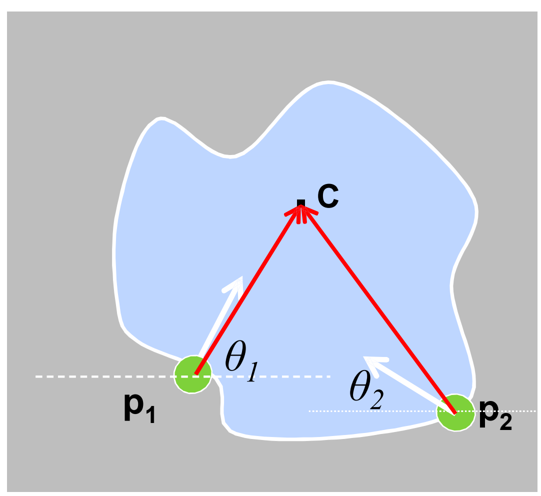
1. At each boundary point, compute displacement vector: $\mathbf{r} = \mathbf{c} - \mathbf{p}_i$
2. Measure the gradient angle θ at the boundary point.
3. Store that displacement in a table indexed by θ.



Generalized Hough transform

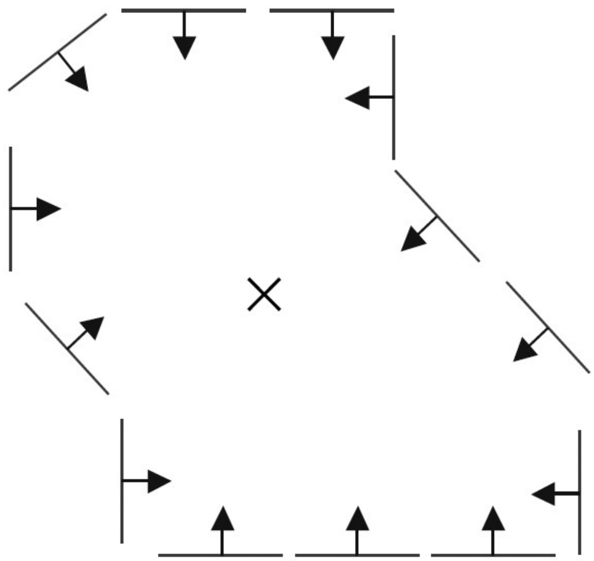
Recognition

1. At each boundary point, measure the gradient angle θ
2. Look up all displacements in θ displacement table.
3. Vote for a center at each displacement.



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

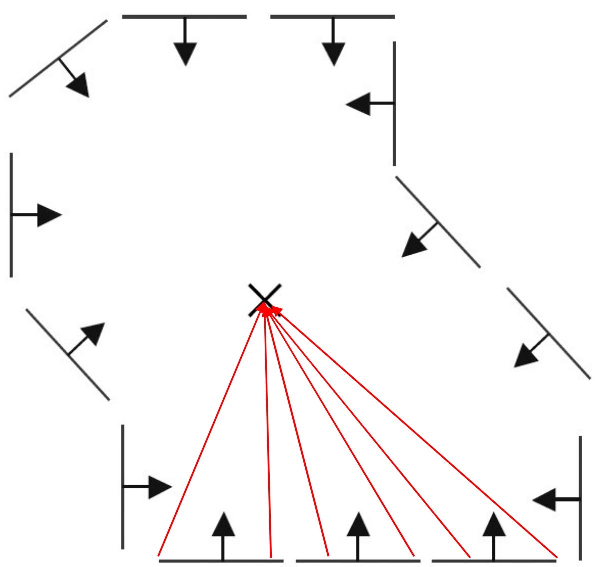
Example (known scale and orientation)



Source: L. Lazebnik

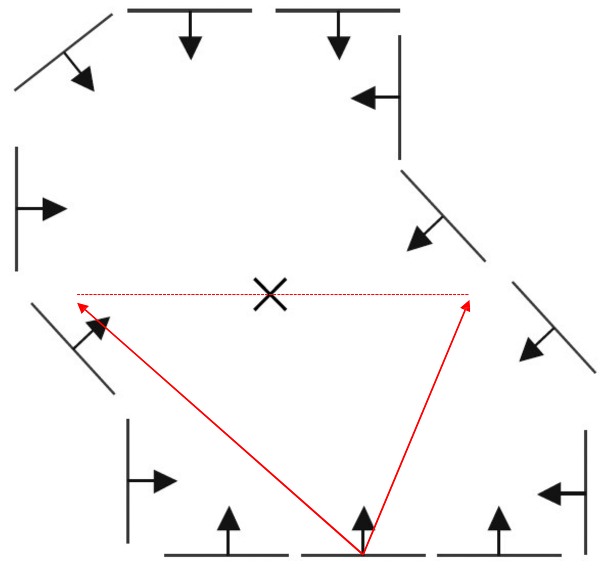
Example

Looking at the bottom horizontal boundary points (all the same θ), the set of displacements ranges over all the red vectors.

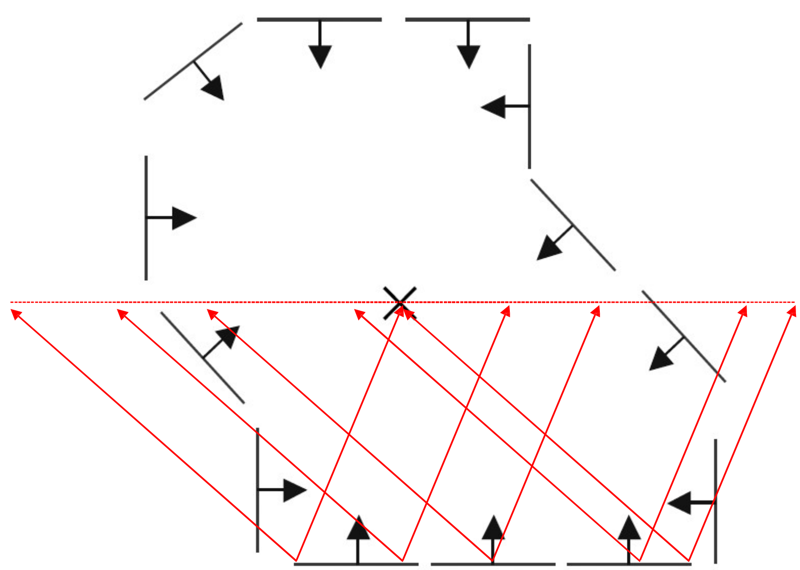


Example

At recognition, each bottom horizontal element votes for all those displacements.

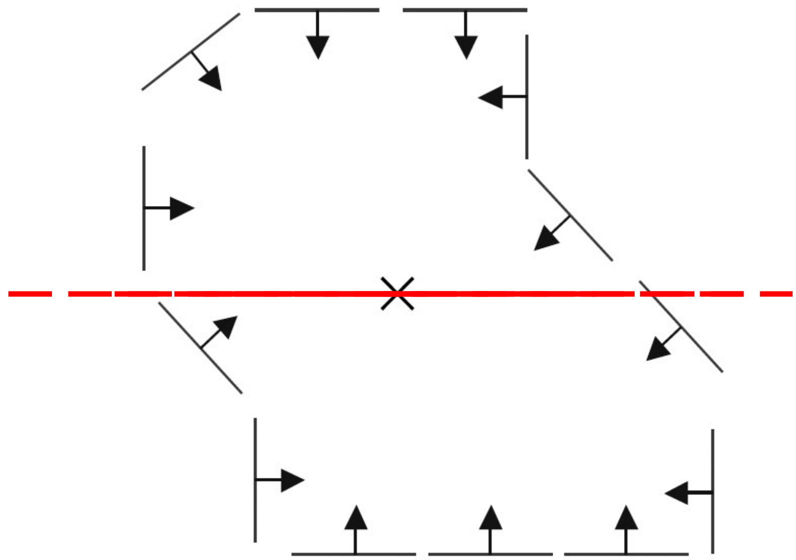


Example



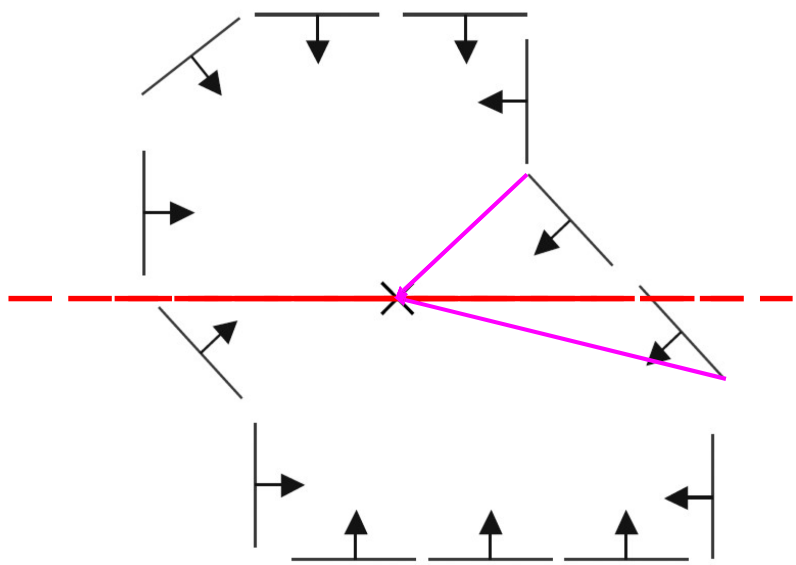
Example

At recognition, each bottom horizontal element votes for all those displacements.



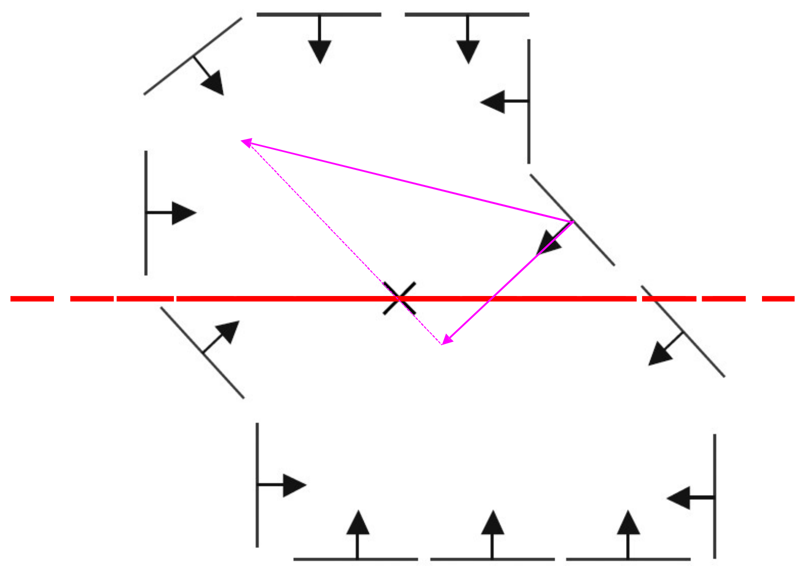
Example

Now do for the leftward pointing diagonals.



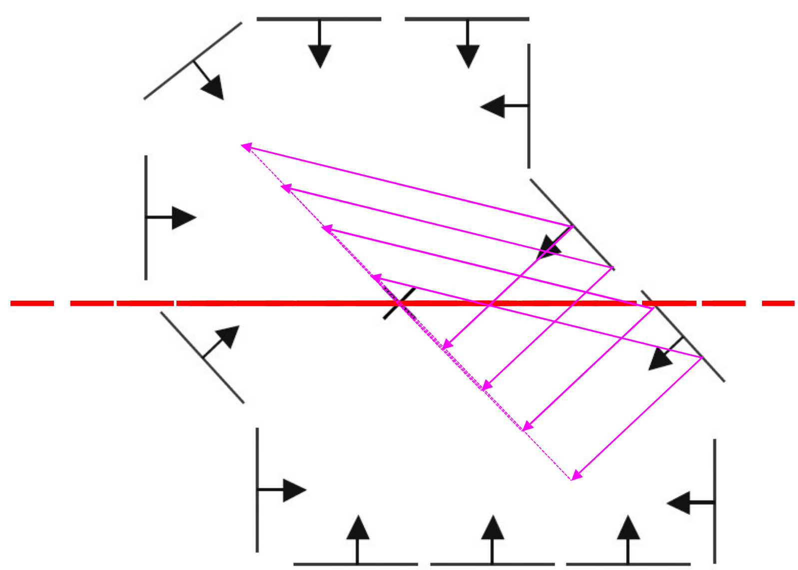
Example

Now do for the leftward pointing diagonals.



Example

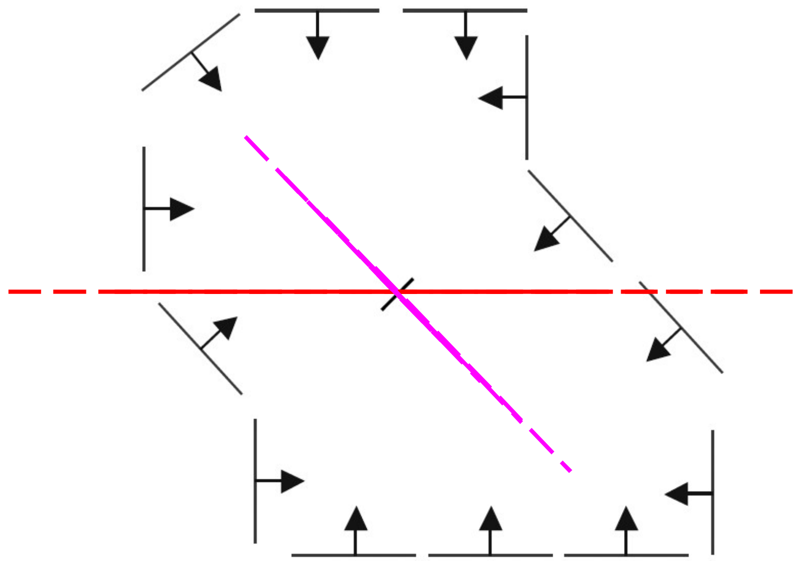
Now do for the leftward pointing diagonals.



Example

Now do for the leftward pointing diagonals.

And the center is found.



Generalized Hough transform

If orientation is known:

1. For each edge point
   - Compute gradient direction θ
   - Retrieve displacement vectors r to vote for reference point.
2. Peak in this Hough space (X,Y) is reference point with most supporting edges

Generalized Hough transform

If orientation is unknown:
For each edge point
For each possible master θ*
Compute gradient direction θ
New θ' = θ - θ*
For θ' retrieve displacement vectors r to vote for reference point.

Peak in this Hough space (now X,Y,θ*) is reference point with most supporting edges

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

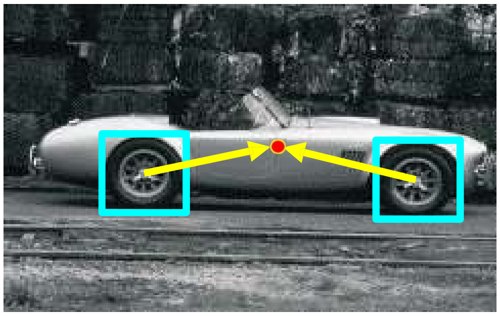
Generalized Hough transform

If scale S is unknown:
For each edge point
For each possible master scale S:
Compute gradient direction θ
For θ' retrieve displacement vectors r
Vote r scaled by S for reference point.

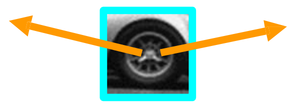
Peak in this Hough space (now X,Y,S) is reference point with most supporting edges

Application in recognition

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



training image



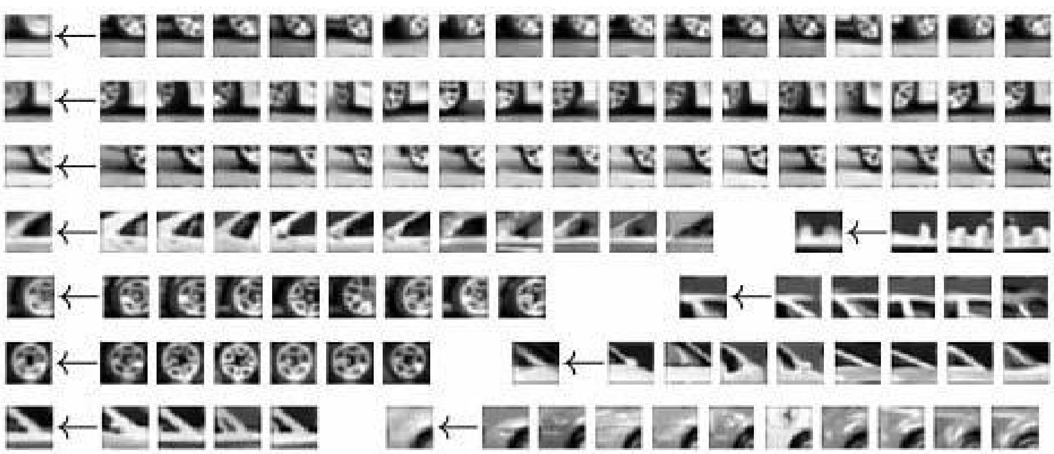
visual codeword with displacement vectors

B. Leibe, A. Leonardis, and B. Schiele, [Combined Object Categorization and Segmentation with an Implicit Shape Model], ECCV Workshop 2004

Source: S. Lazebnik

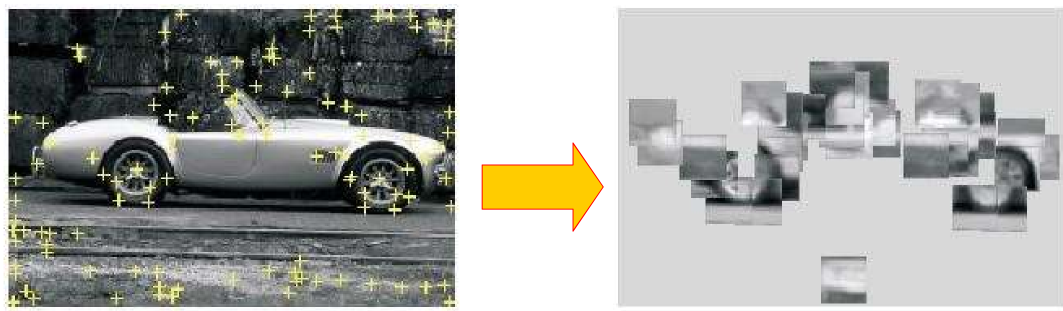
Training: Visual code-words

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



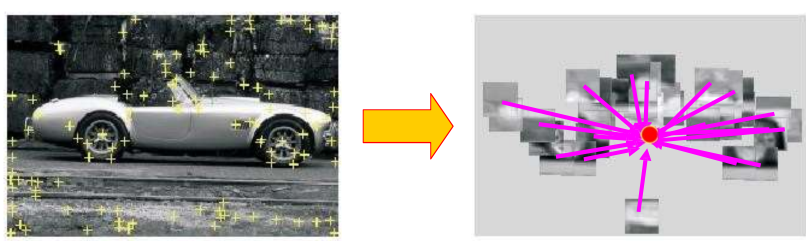
Training: Interest points

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



Training: Displacements

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 displacements relative to object center



Application in recognition

test image

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