Image and Video Procesing

Model Fitting – Hough Transform

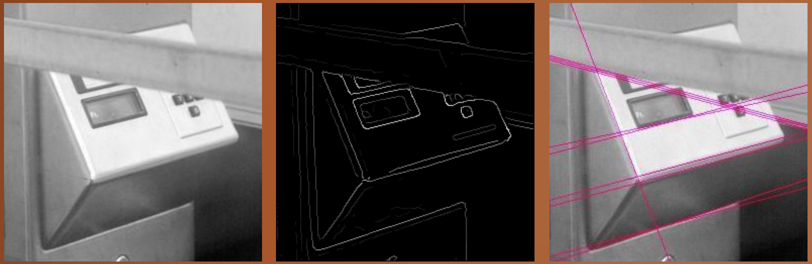
Now some computer vision…

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

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

Fitting a model

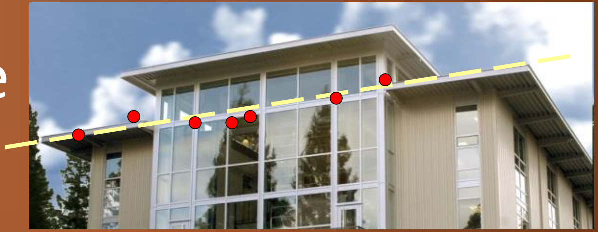
Fitting observed data/set of features into a parametric model that we are assuming holds

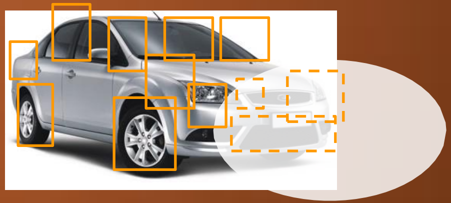


Parametric model

- A ***parametric*** model can represent a class of instances. Examples include lines, or circles, or even a parameterized template.

- Data measured in real images is inaccurate
  - Occlusions
  - Noise
  - Ambiguity
  - Wrong feature extraction





Fitting a parametric model

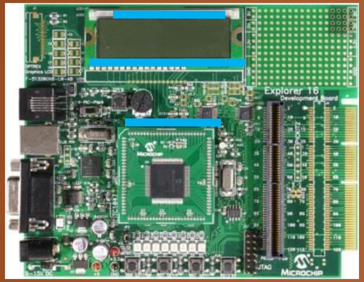
- Choose a parametric model to represent a set of features
- Membership criterion is not local:

  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 all possible parameter setting

Source: S. Lazebnik

Example: Line fitting



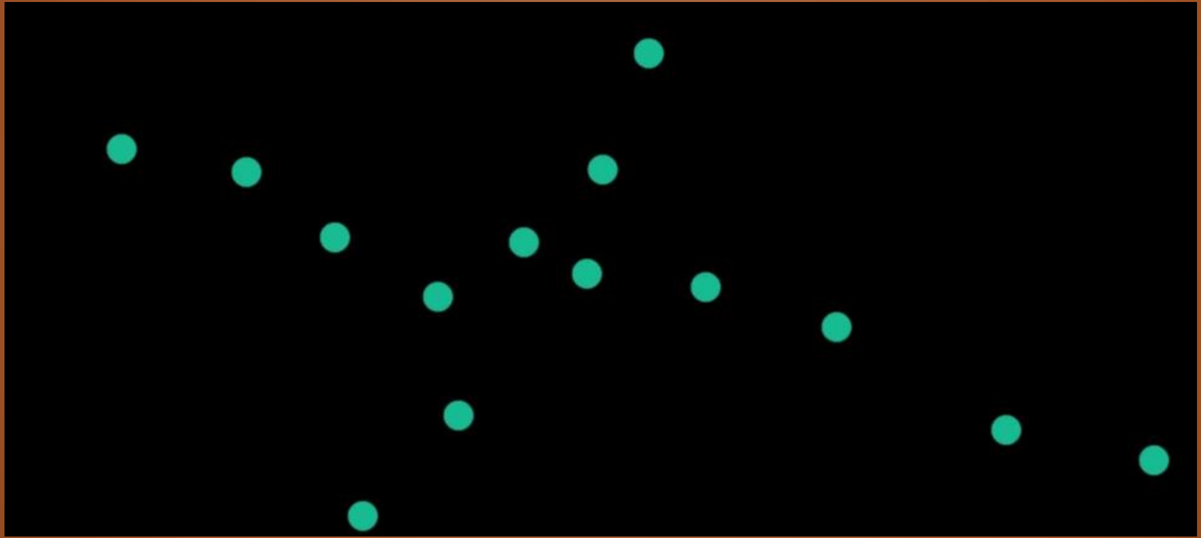
Source: K. Grauman

Difficulty of line fitting

- Extra edge points (clutter), multiple models.
- Only some parts of each line detected, and some parts are missing.
- Noise in measured edge points, orientations.

Quiz: Edges to lines

How many lines can you identify here?



Fitting methods/techniques

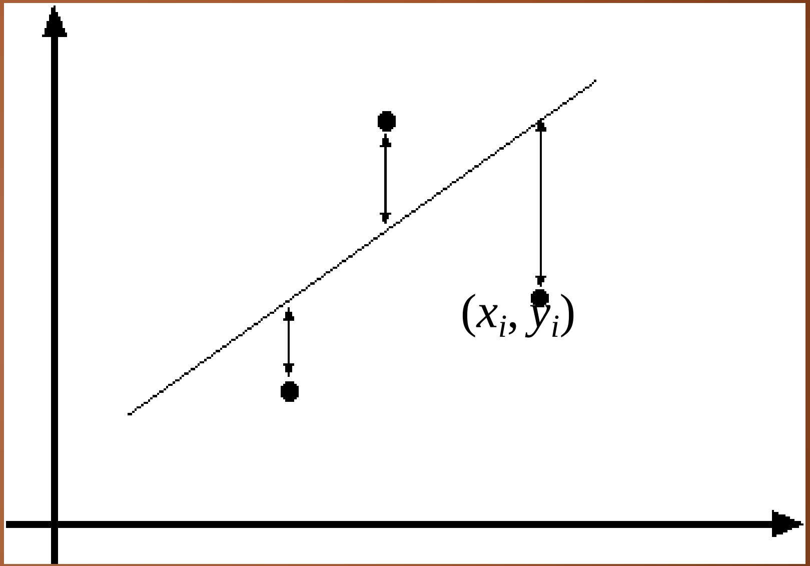
- Global optimization / Search for parameters
  - Least squares fit
  - Robust least squares

- Voting
  - Hough transform
  - RANSAC

Typical least squares line fitting

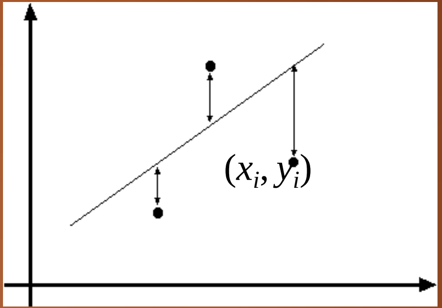
- Data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Line equation: $y_i = m\, x_i + b$
- Find (m, b) to minimize:

$$E = \sum_{i=1}^{n} (y_i - m x_i - b)^2$$



Typical least squares line fitting

$$E=\sum_{i=1}^{n}(y_i - m x_i - b)^2$$



$$E=\sum_{i=1}^{n}\left(y_i - \begin{bmatrix}x_i & 1\end{bmatrix}\begin{bmatrix}m\\ b\end{bmatrix}\right)^2 = \left\|\begin{bmatrix}y_1\\ \vdots\\ y_n\end{bmatrix} - \begin{bmatrix}x_1 & 1\\ \vdots & \vdots\\ x_n & 1\end{bmatrix}\underbrace{\begin{bmatrix}m\\ b\end{bmatrix}}_{\mathbf{h}}\right\|^2 = \|\mathbf{y}-\mathbf{X}\mathbf{h}\|^2$$

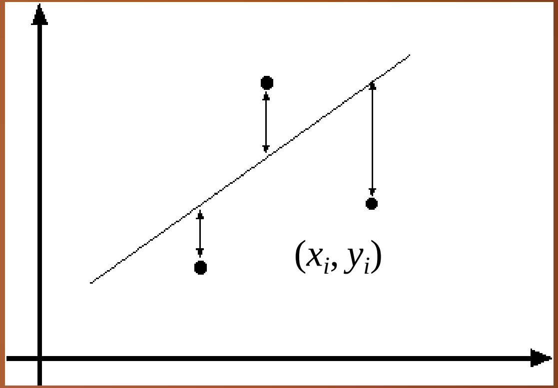
$$E=(\mathbf{y}-\mathbf{X}\mathbf{h})^T(\mathbf{y}-\mathbf{X}\mathbf{h}) = \mathbf{y}^T\mathbf{y} - 2(\mathbf{X}\mathbf{h})^T\mathbf{y} + (\mathbf{X}\mathbf{h})^T(\mathbf{X}\mathbf{h})$$

Typical least squares line fitting

$$E = (\mathbf{y} - \mathbf{X}\mathbf{h})^T(\mathbf{y} - \mathbf{X}\mathbf{h}) = \mathbf{y}^T\mathbf{y} - 2(\mathbf{X}\mathbf{h})^T\mathbf{y} + (\mathbf{X}\mathbf{h})^T(\mathbf{X}\mathbf{h})$$

$$\Rightarrow \frac{dE}{d\mathbf{h}} = 2\mathbf{X}^T\mathbf{X}\mathbf{h} - 2\mathbf{X}^T\mathbf{y} = 0$$

$$\mathbf{X}^T\mathbf{X}\mathbf{h} = \mathbf{X}^T\mathbf{y} \Rightarrow \mathbf{h} = \underbrace{(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T}_{\textit{pseudoinverse}}\mathbf{y}$$



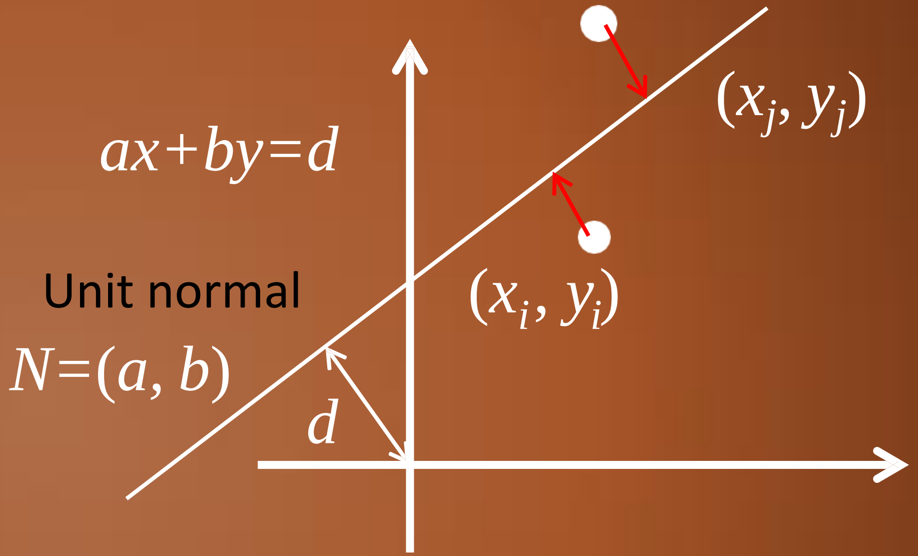
Standard over-constrained least squares solution

Problem with “vertical” leastsquares

- Not rotation-invariant
- Fails completely for vertical lines

Total least squares

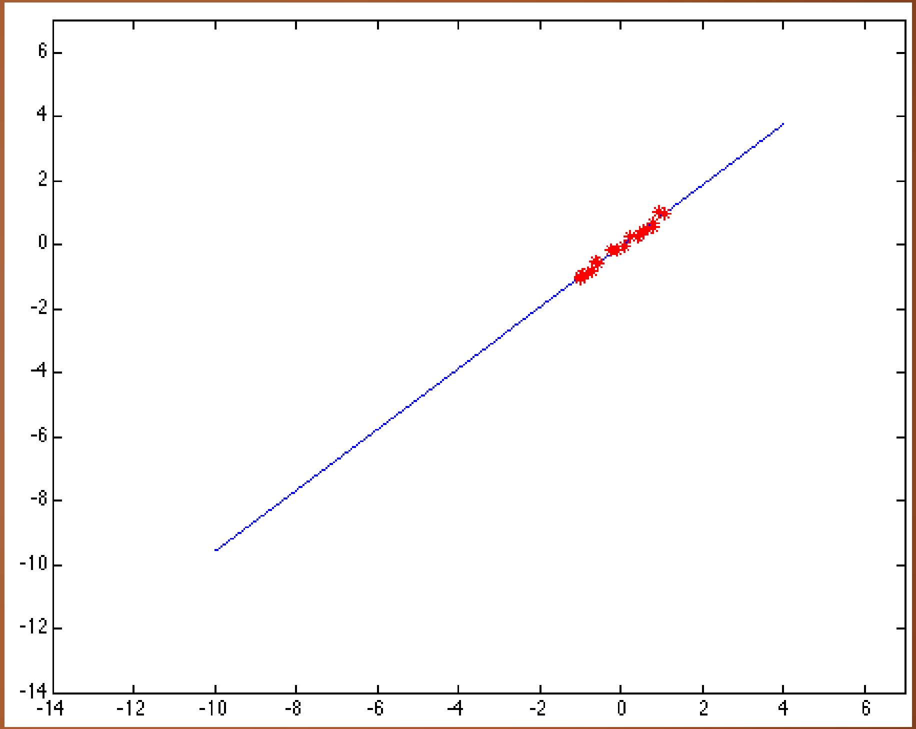
- Distance between point (x_i, y_i) and line $ax + by = d$
- Find (a, b, d) to minimize the sum of squared *perpendicular* distances



$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$

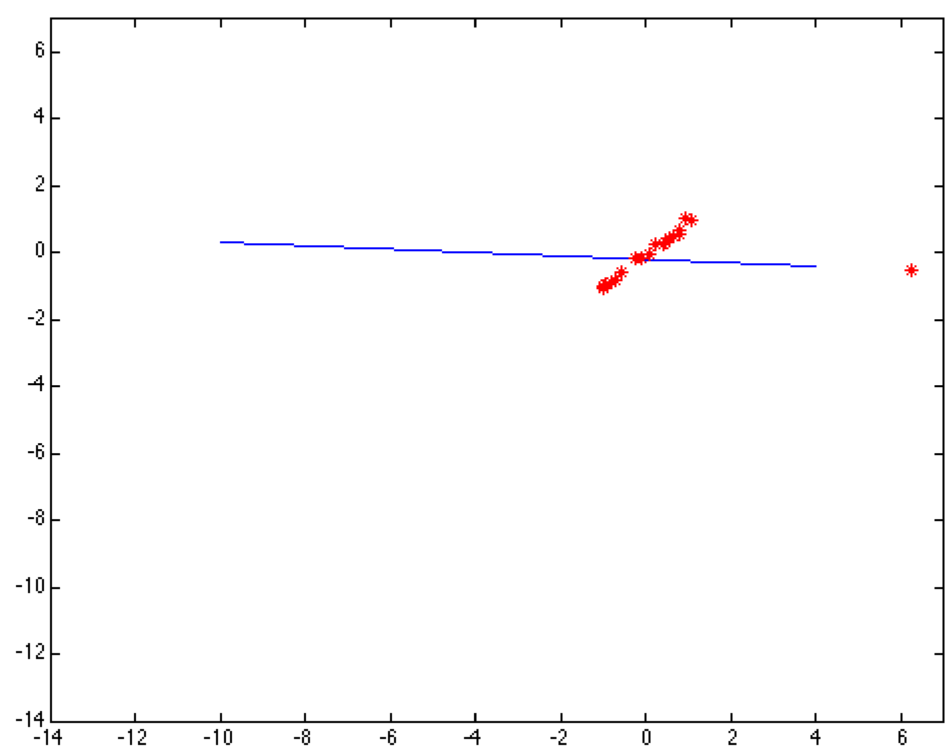
Least squares: good for Gaussian noise

- Least squares fit to the red points:



Least squares: Non-robustness to (very) non-Gaussian noise

- Least squares fit with an outlier…

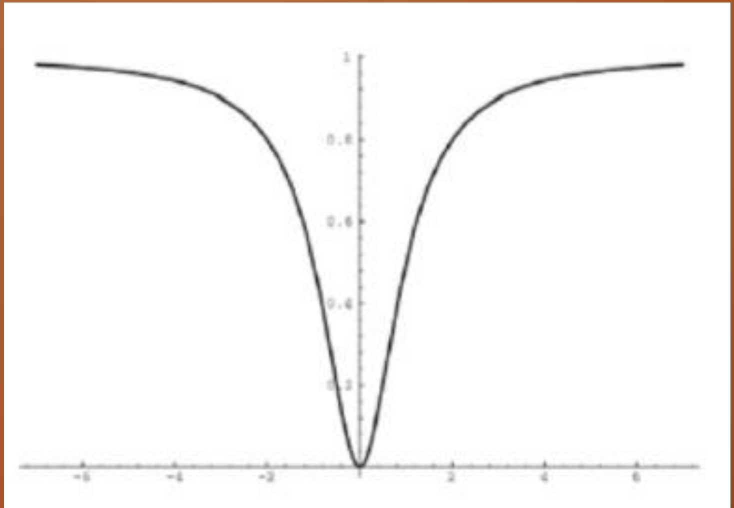


Problem: *squared error heavily penalizes outliers*

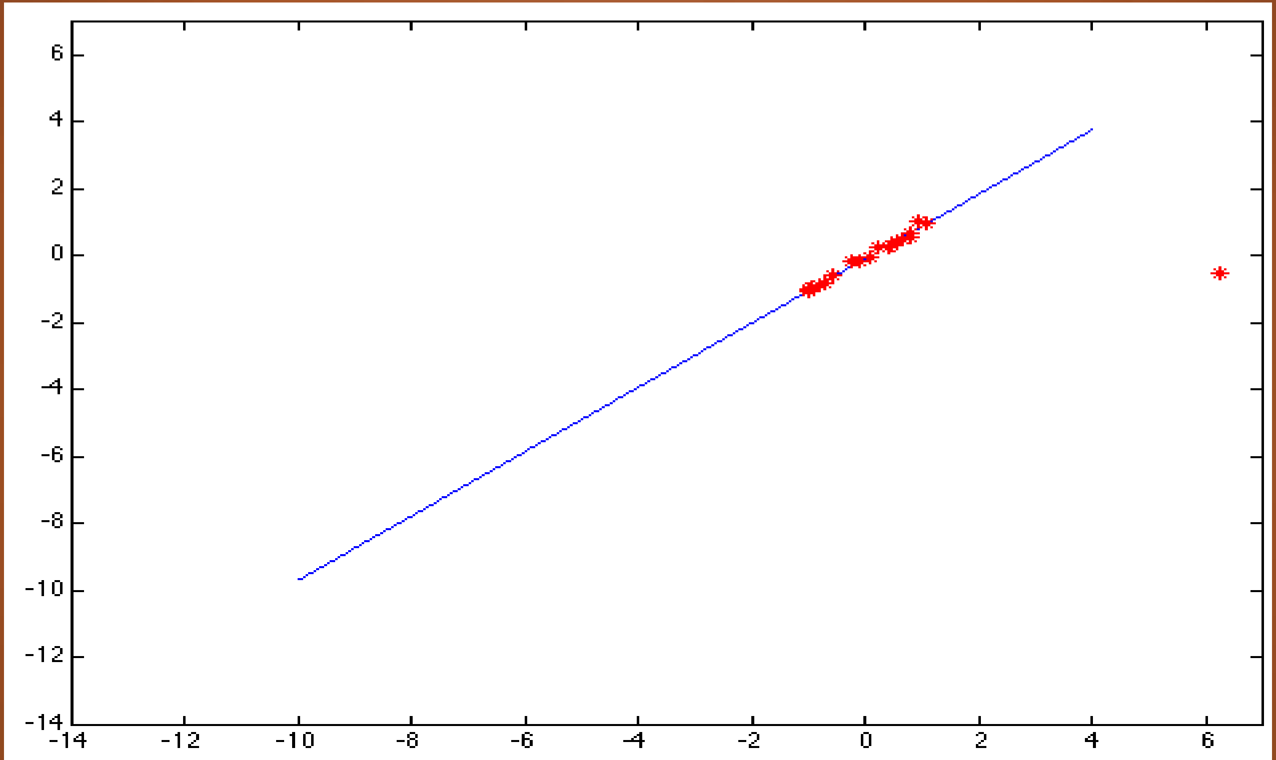
Robust estimators

- Squared error was problematic
- Take absolute value – hard to minimize
- Robust norm: The robust function ρ behaves like squared distance for small values of the residual *u* but saturates for larger values of *u*

$$\rho(u;\sigma) = \frac{u^2}{\sigma^2 + u^2}$$

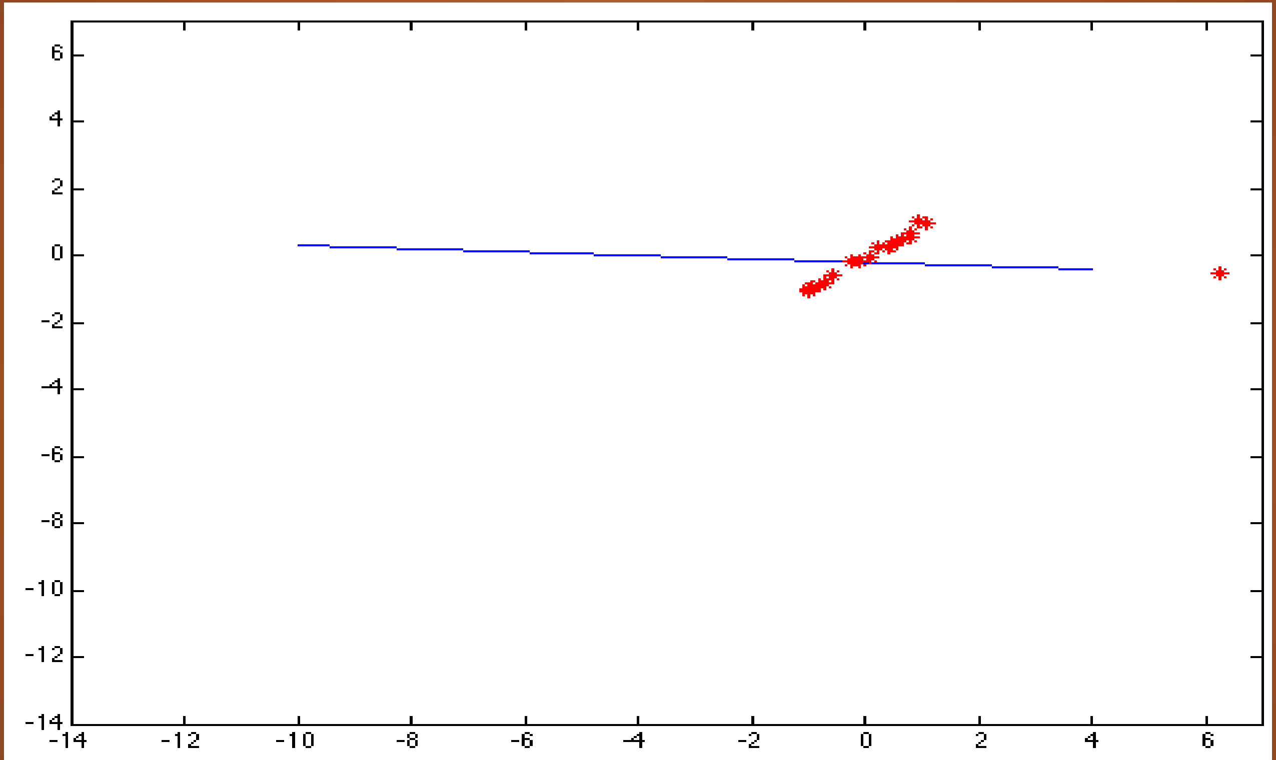


Choosing the scale: Just right



The effect of the outlier is minimized

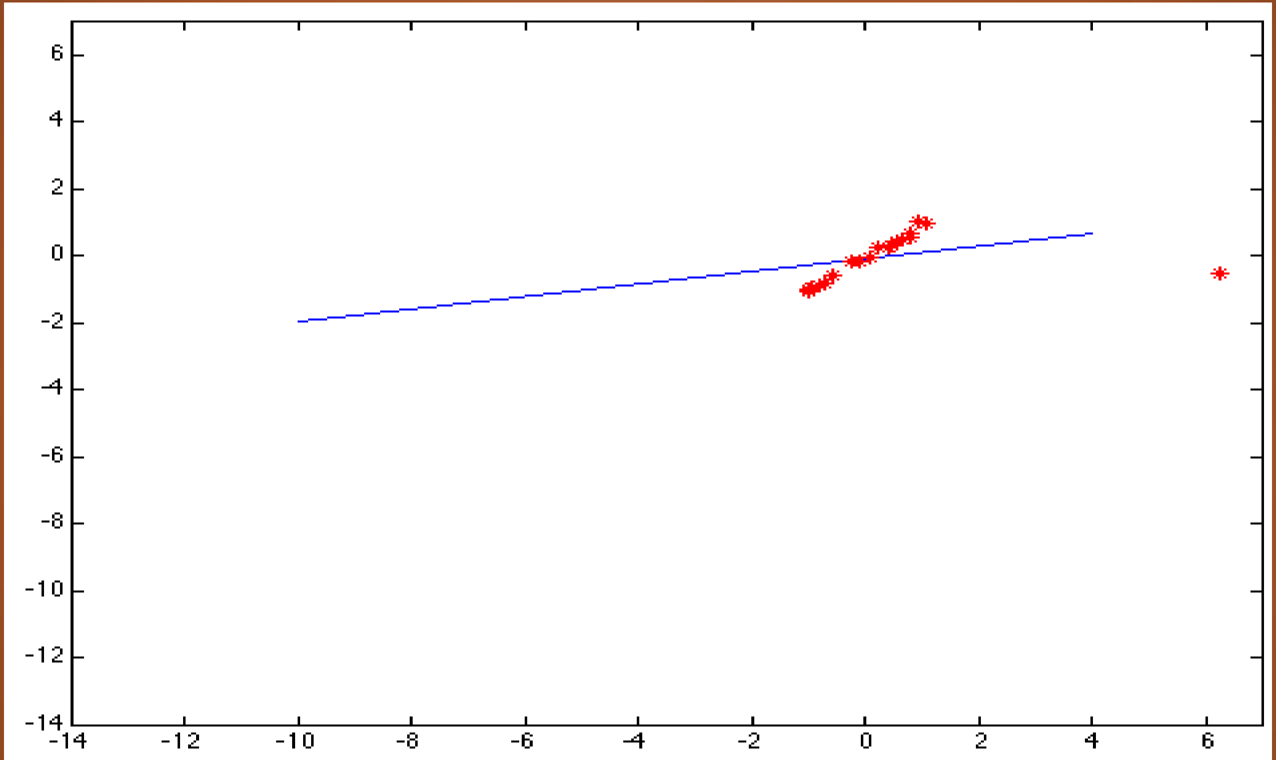
Choosing the scale: Too small



The error value is almost the same for every point and the fit is very poor

Slide: S. Lazebnik

Choosing the scale: Too large



Behaves much the same as least squares

Fitting methods/techniques

- Global optimization / Search for parameters
  - Least squares fit
  - Robust least squares
- Voting
  - Hough transform
  - RANSAC

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.

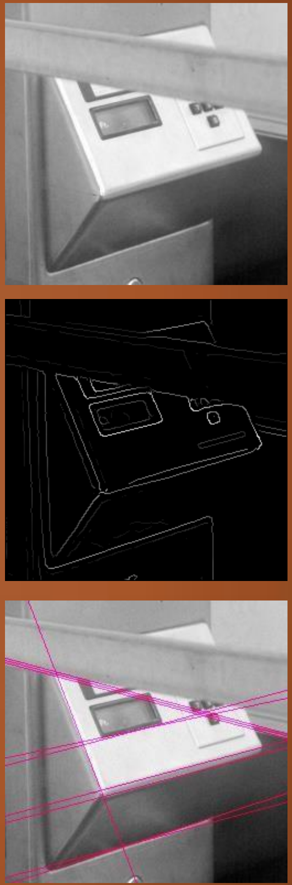
Voting – why it works

- 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?

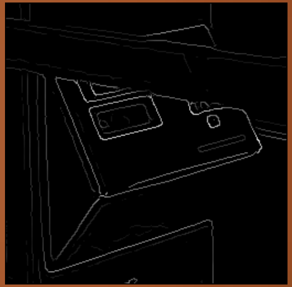


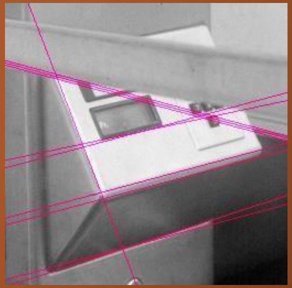
Fitting lines

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.
2. Look for lines that get many votes.





Hough space

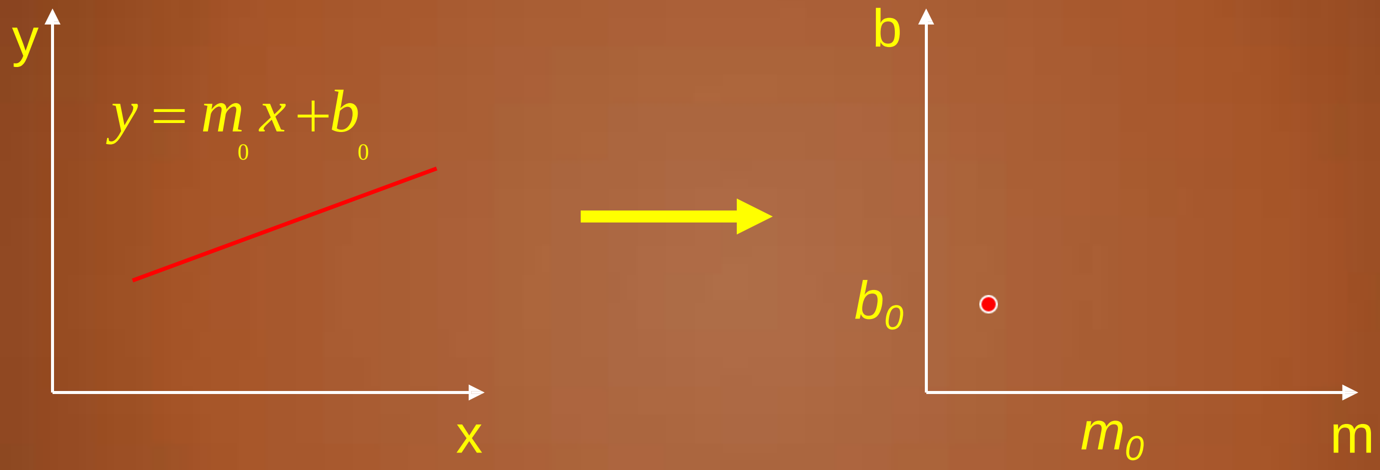


image space

Hough (parameter) space

A line in the image corresponds to a point in Hough space

Slide credit: Steve Seitz

Hough space

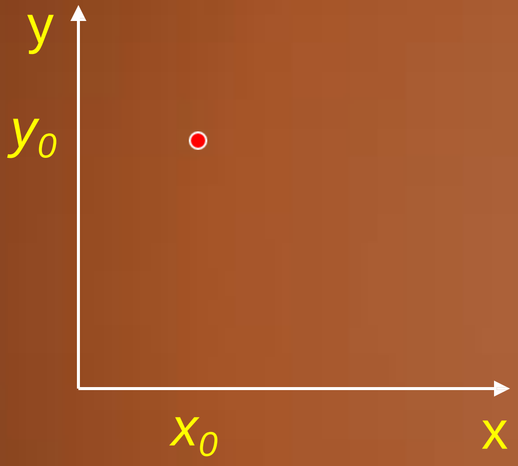
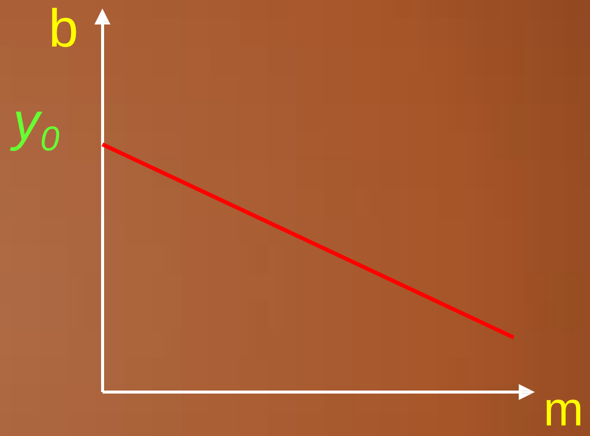


image space

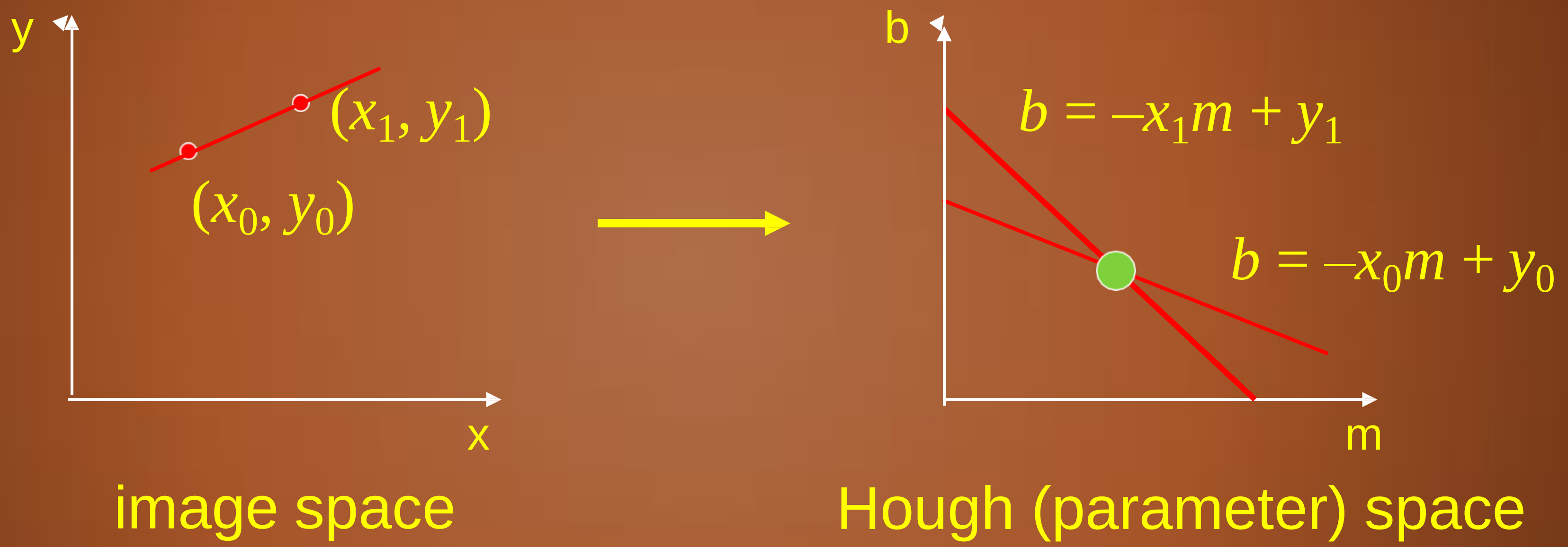


Hough (parameter) space

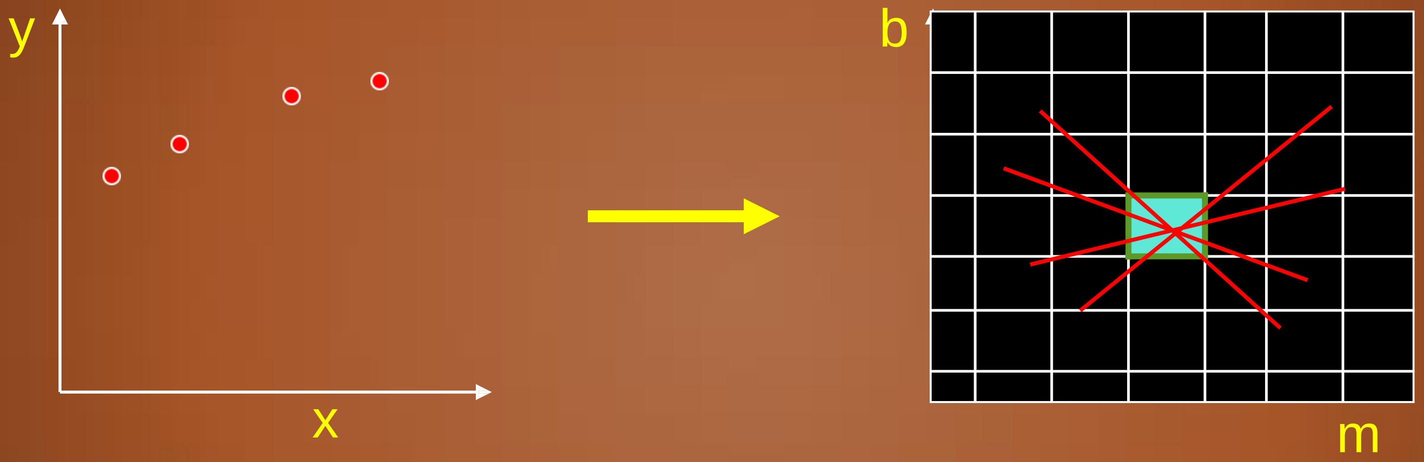
$$y_0 = mx_0 + b$$

$$b = -x_0 m + y_0$$

Hough space


y
(x_1, y_1)
(x_0, y_0)
x
image space
b
$b = -x_1 m + y_1$
$b = -x_0 m + y_0$
m
Hough (parameter) space

Hough algorithm

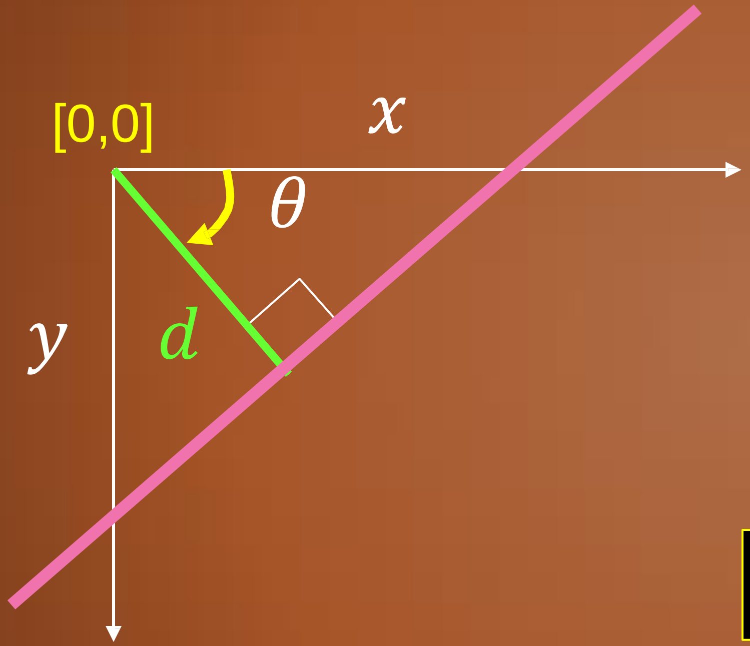


- Let each edge point in image space vote for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

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

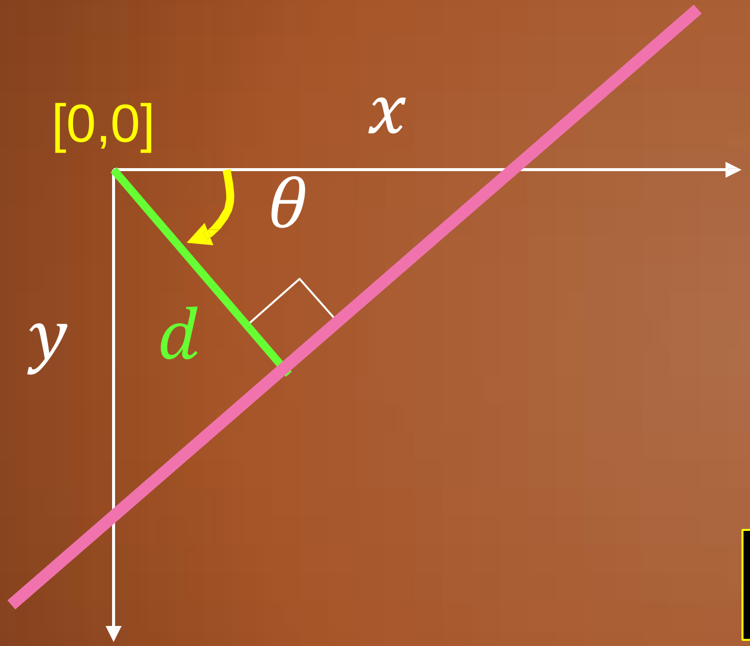


$\boldsymbol{d}$: perpendicular distance from line to origin

$\boldsymbol{\theta}$: angle the perpendicular makes with the x-axis

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

Polar representation for lines



$\boldsymbol{d}$: perpendicular distance from line to origin

$\boldsymbol{\theta}$: angle the perpendicular makes with the x-axis

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

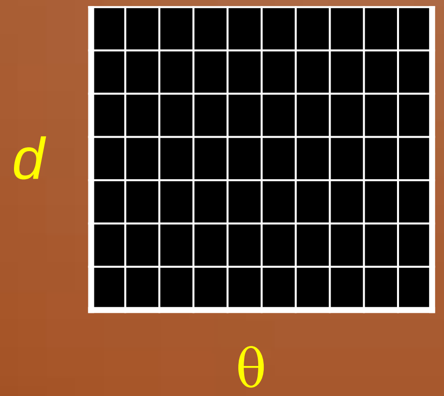
Point in image space is now sinusoid segment in Hough space

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 in $E(x, y)$ in the image

   for θ = -90 to +90 // some quantization

   $d = x\cos\theta + y\sin\theta$ // maybe negative

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

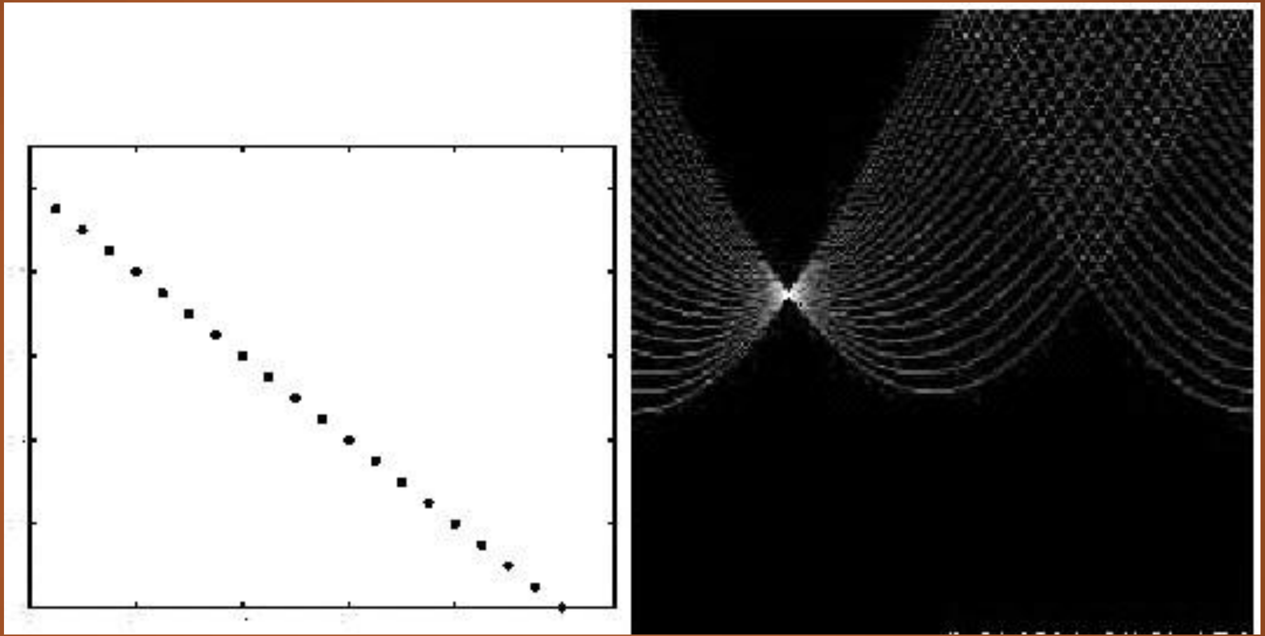
Source: Steve Seitz

Complexity of the Hough transform

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

Time complexity (in terms of number of voting elements)?

Hough example



y

x

d

θ

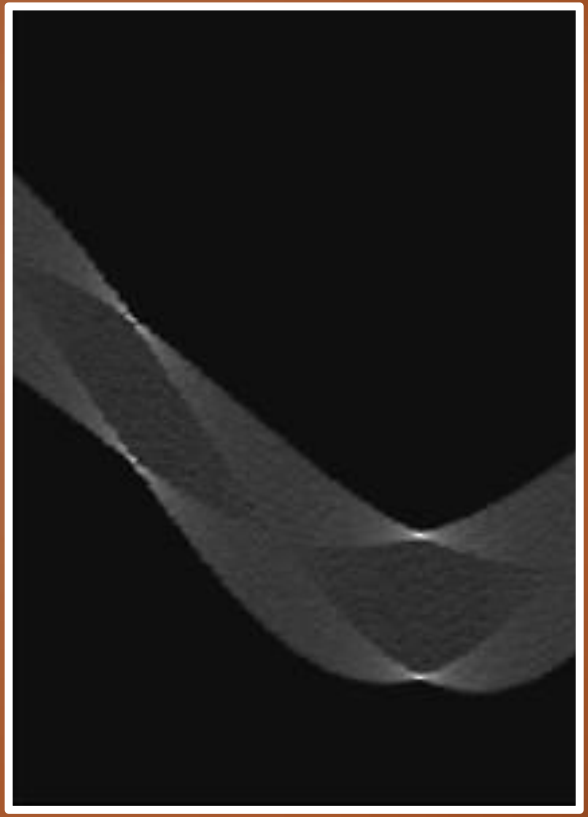
Image space
edge coordinates

Votes

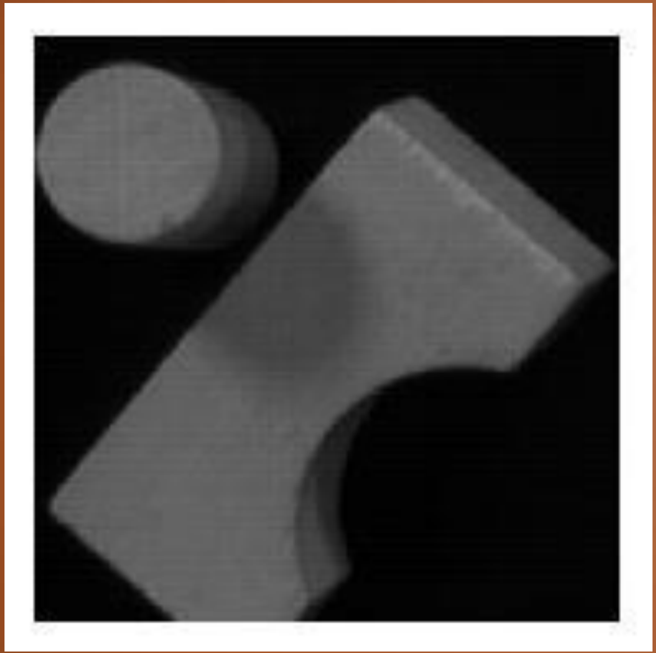
Bright value = high vote count
Black = no votes

Example: Hough transform of a square

Square :



Hough transform of blocks scene



Matlab Demo

Impact of noise on Hough

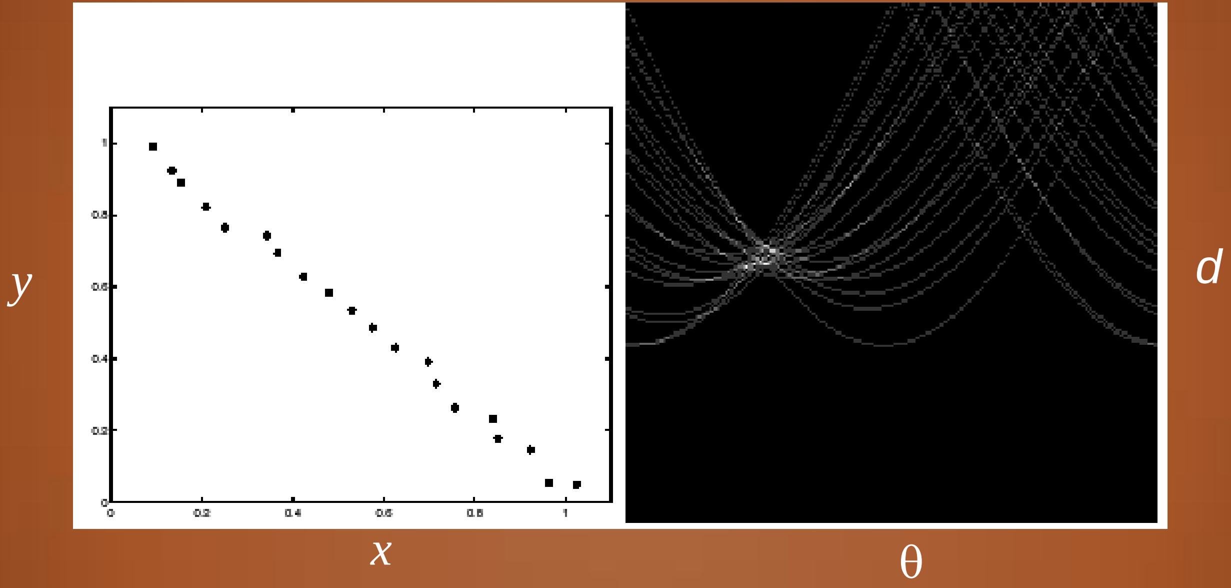


Image space edge coordinates

Votes

Impact of more noise on Hough

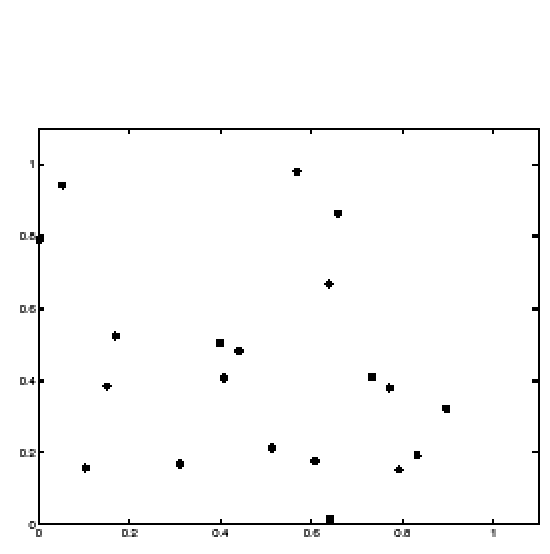
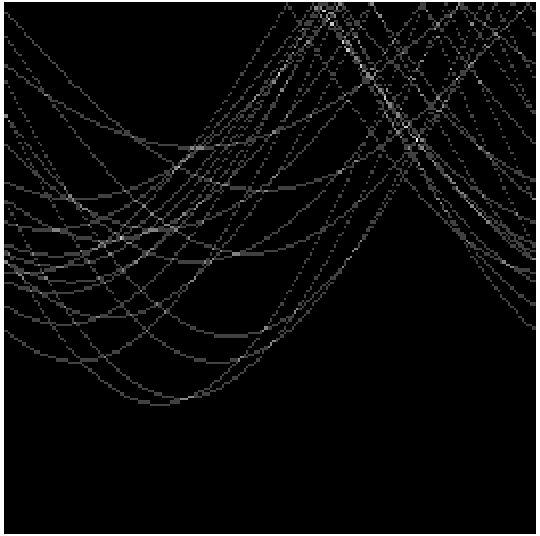


Image space
edge coordinates



Votes

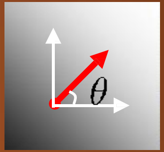
Extensions – using the gradient

1. Initialize H[d, θ]=0
2. For each ***edge*** point in $E(x, y)$ in the image

   θ = gradient at (x,y)

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

   $H[d, \theta] += 1$
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by $d = x\cos\theta + y\sin\theta$

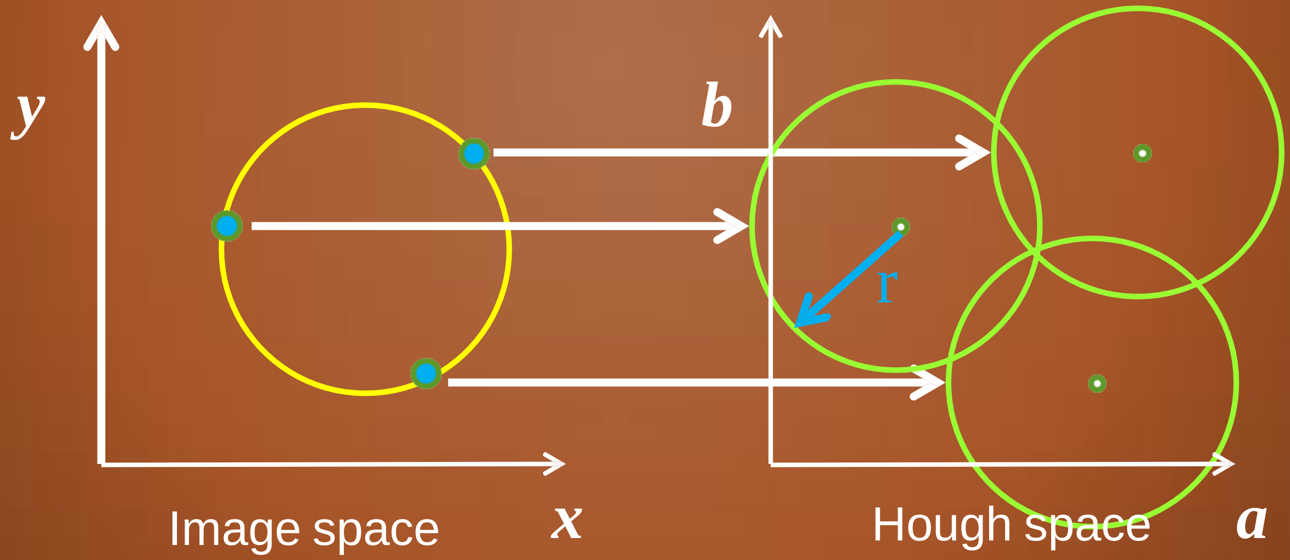


$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

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

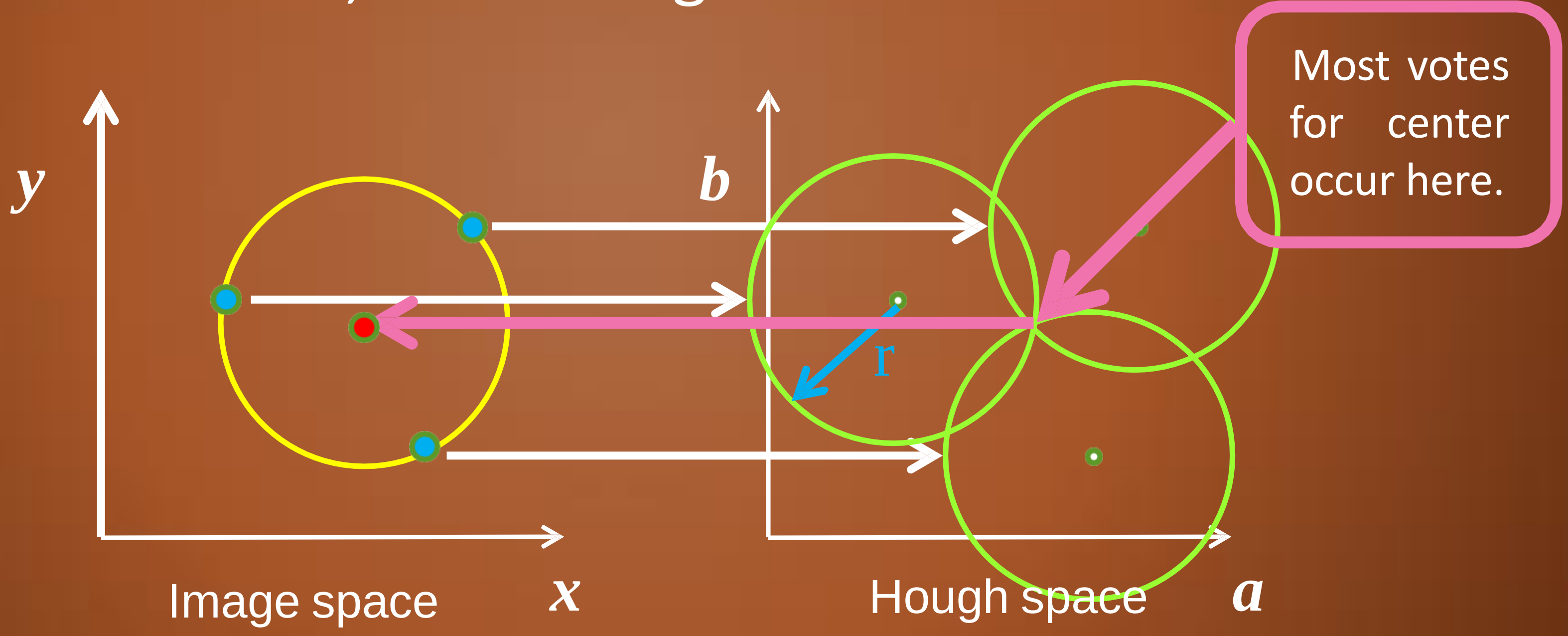
Hough transform for circles

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



Hough transform for circles

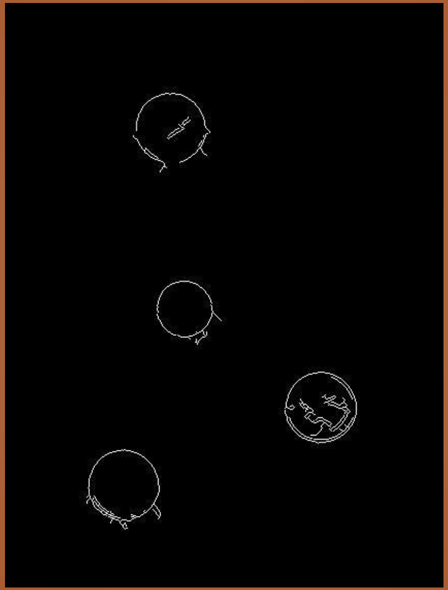
- Circle: center (a,b) and radius r $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For a fixed radius r, unknown gradient direction:



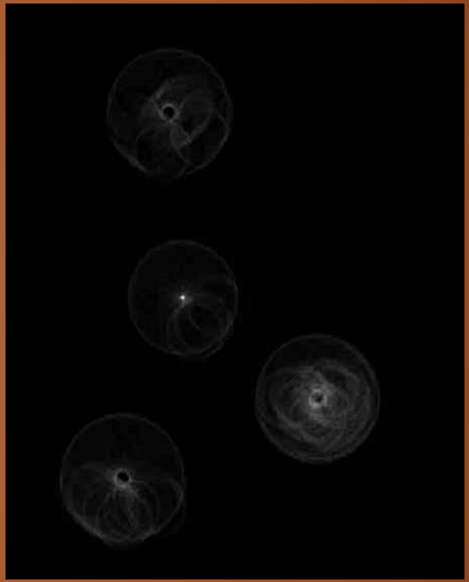
Example: detecting circles with Hough

Original

Edges



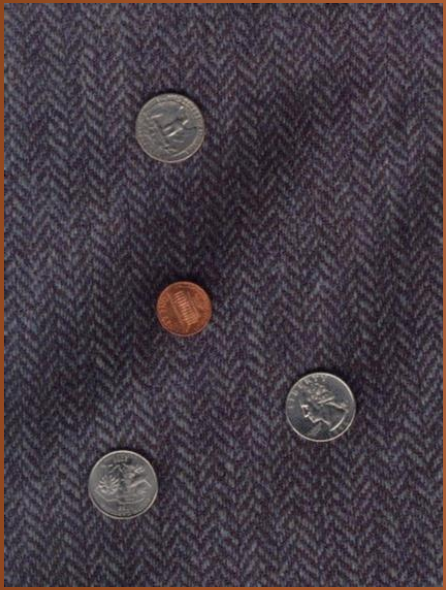
Votes: Penny



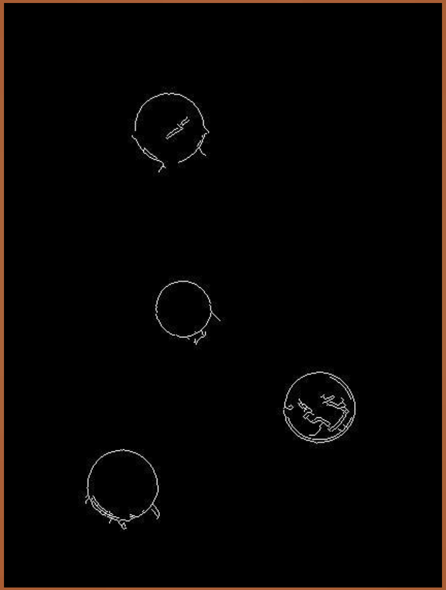
Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Example: detecting circles with Hough

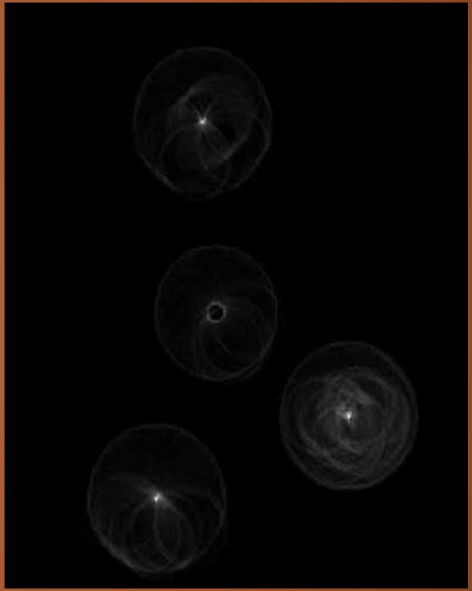
Original



Edges



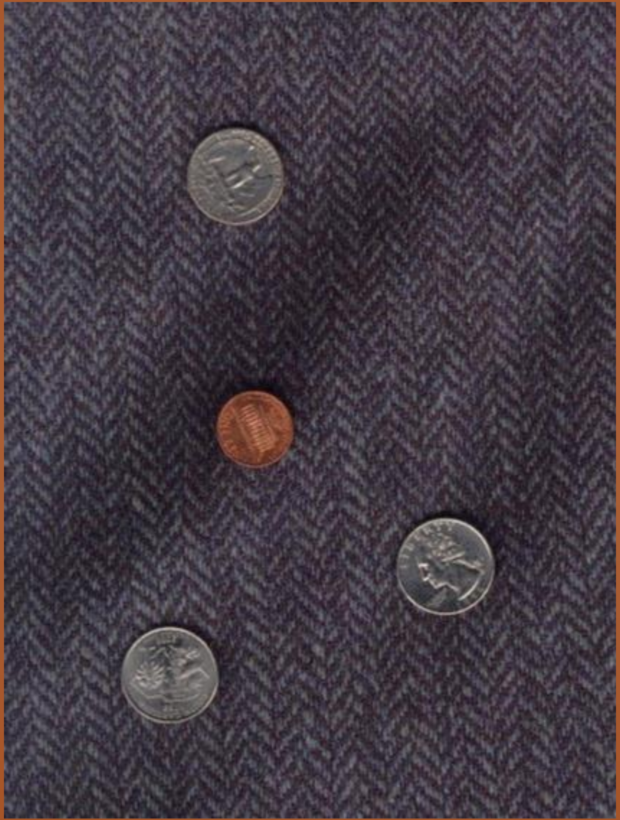
Votes: Quarter



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

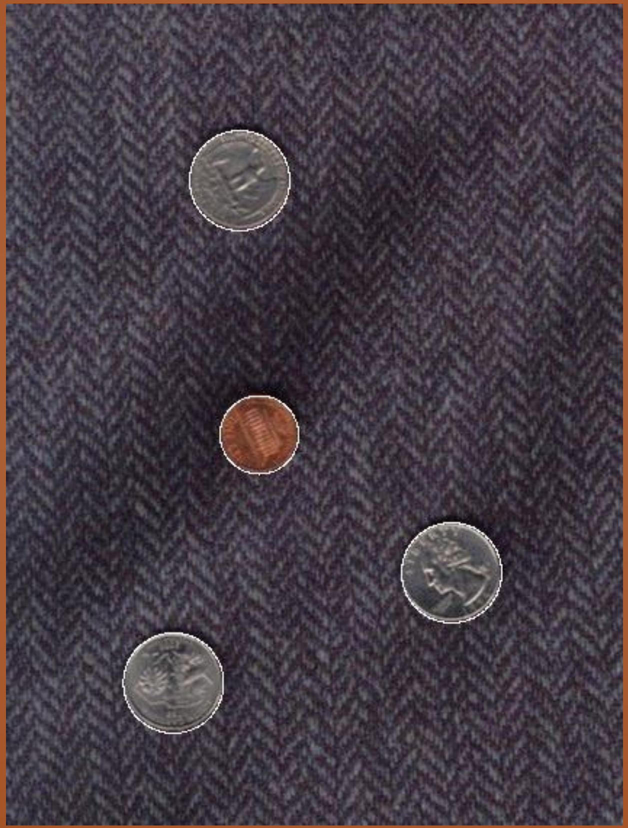
Example: detecting circles with Hough

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:

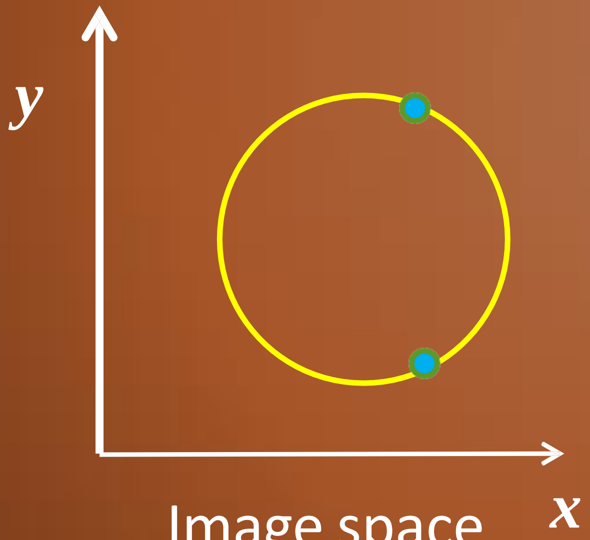
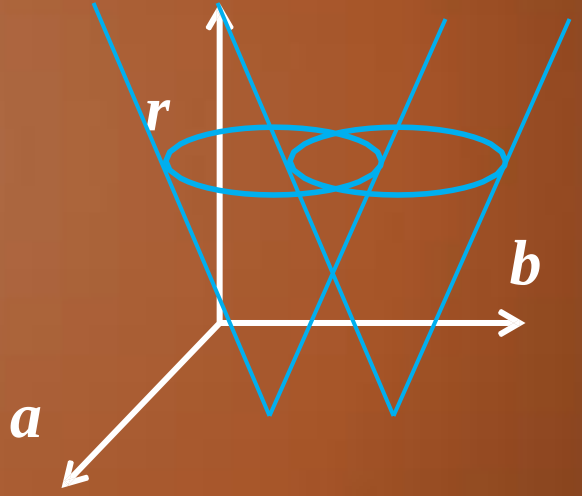


Image space



Hough space

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***:

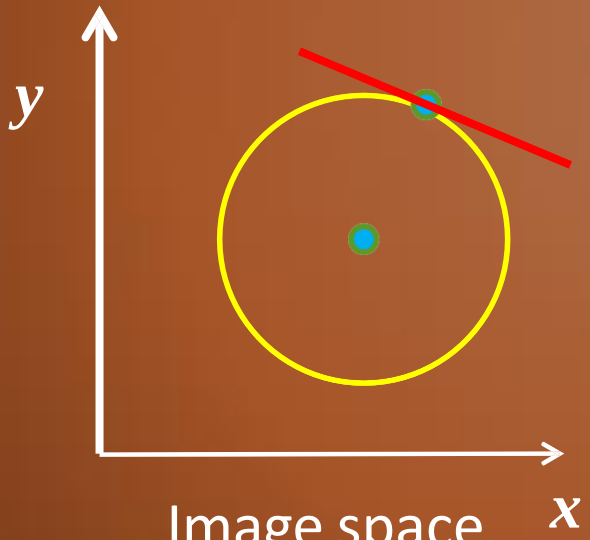
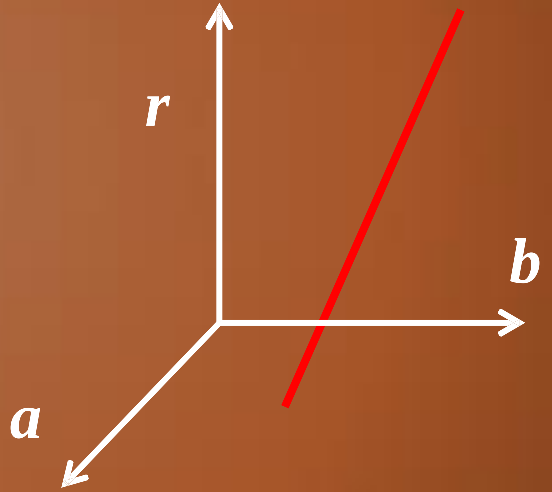


Image space



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
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 bucket
  - Too fine: miss lines because some points that are not exactly collinear cast votes for different buckets

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

- 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

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

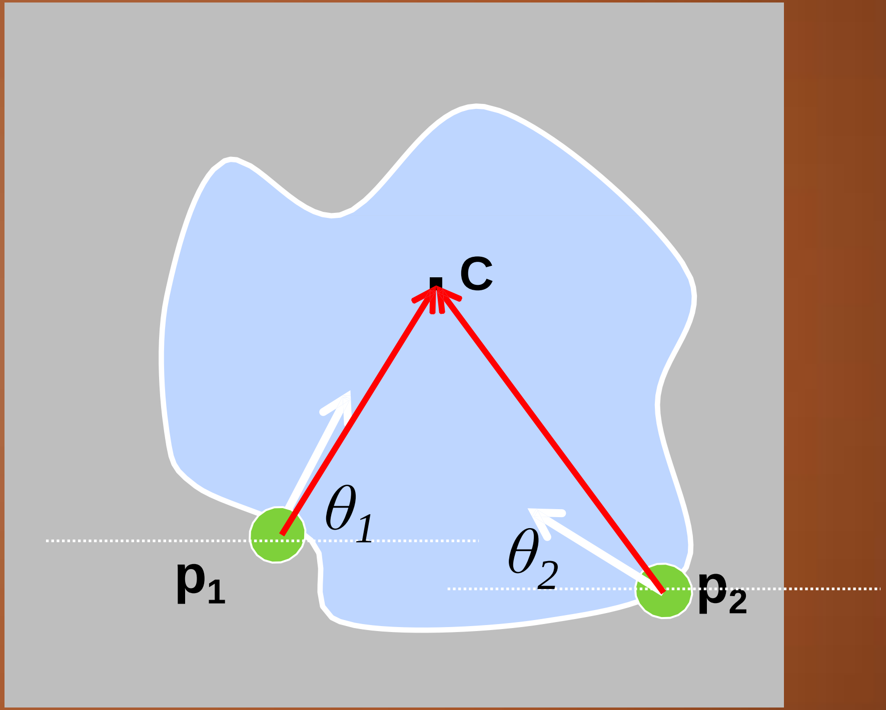
Generalized Hough transform

- **Non-analytic** models
  - *Parameters express variation in pose or scale of fixed but arbitrary shape*
- **Visual code-word** based features
  - *Not edges but detected templates learned from models (more recent approach)*

Generalized Hough transform

Training: build a Hough table

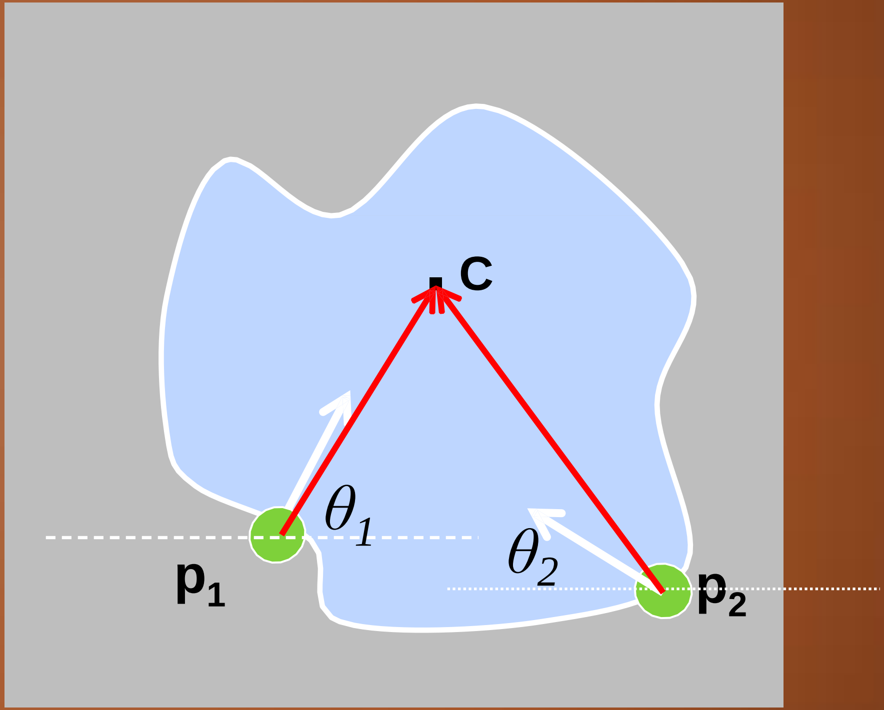
1. At each boundary point, compute displacement vector: **r** = **c** – $\mathbf{p}_i$
2. Measure the gradient angle θ at the boundary point.
3. Store that displacement vector in a table indexed by θ.



Generalized Hough transform

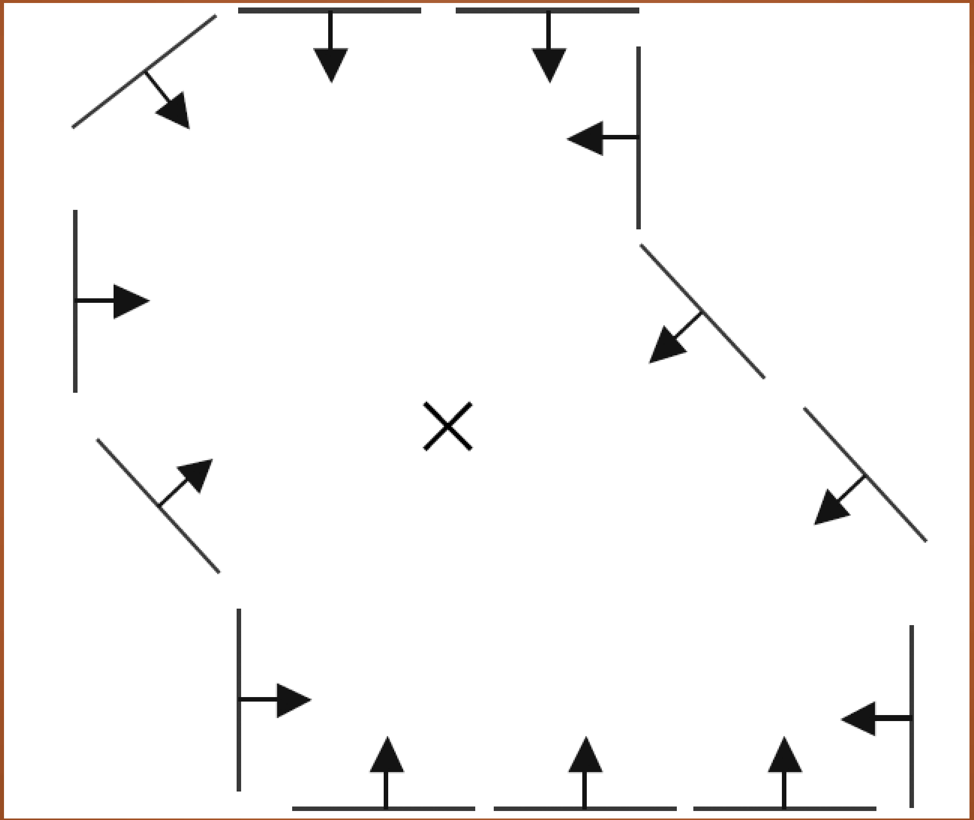
Detection:

1. At each boundary point, measure the gradient angle θ
2. Look up all displacements in θ displacement table.
3. For each r(θ), put a vote in the Hough space at p + r(θ)



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

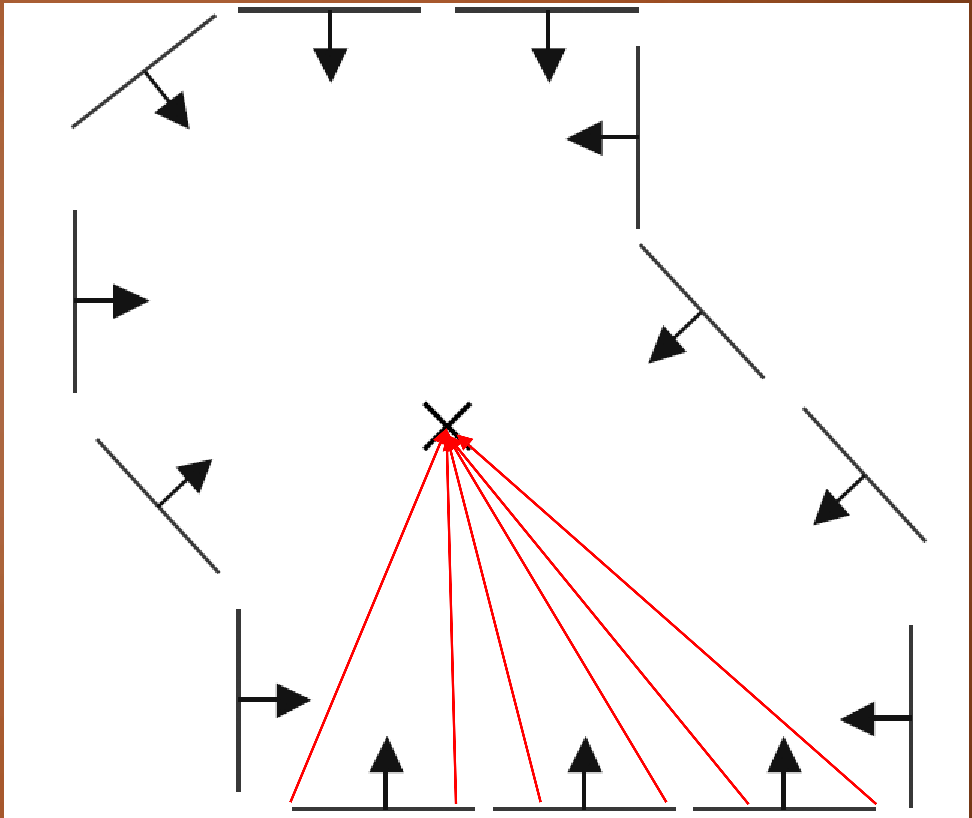
Example



Source: L. Lazebnik

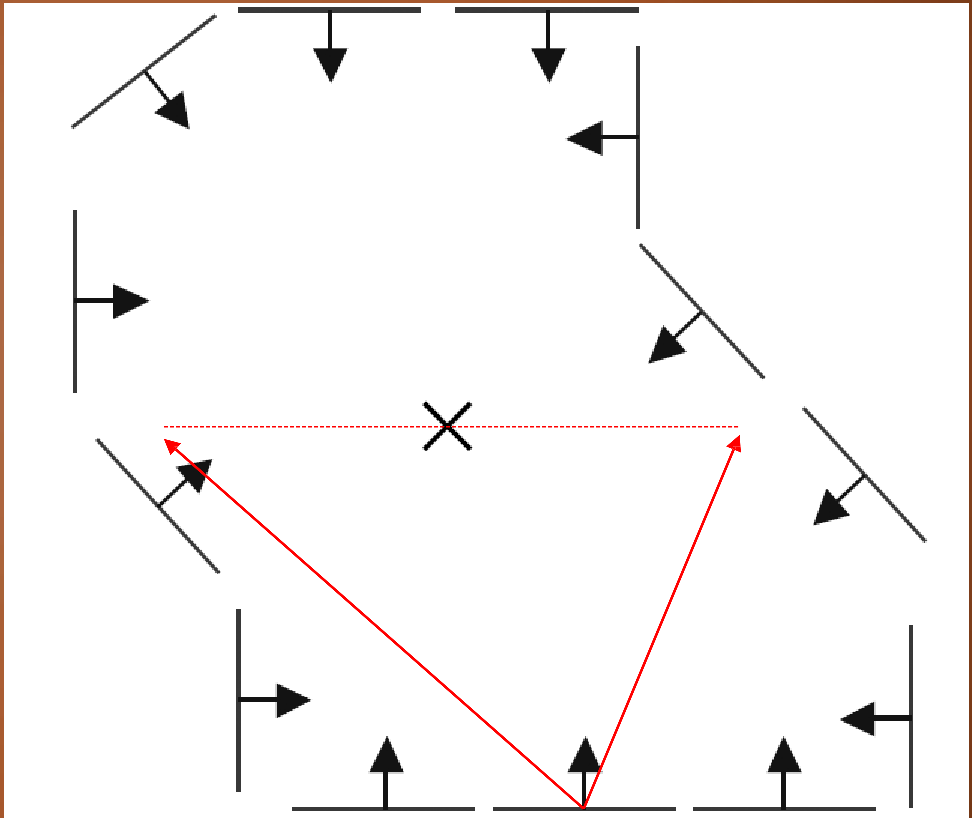
Example

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



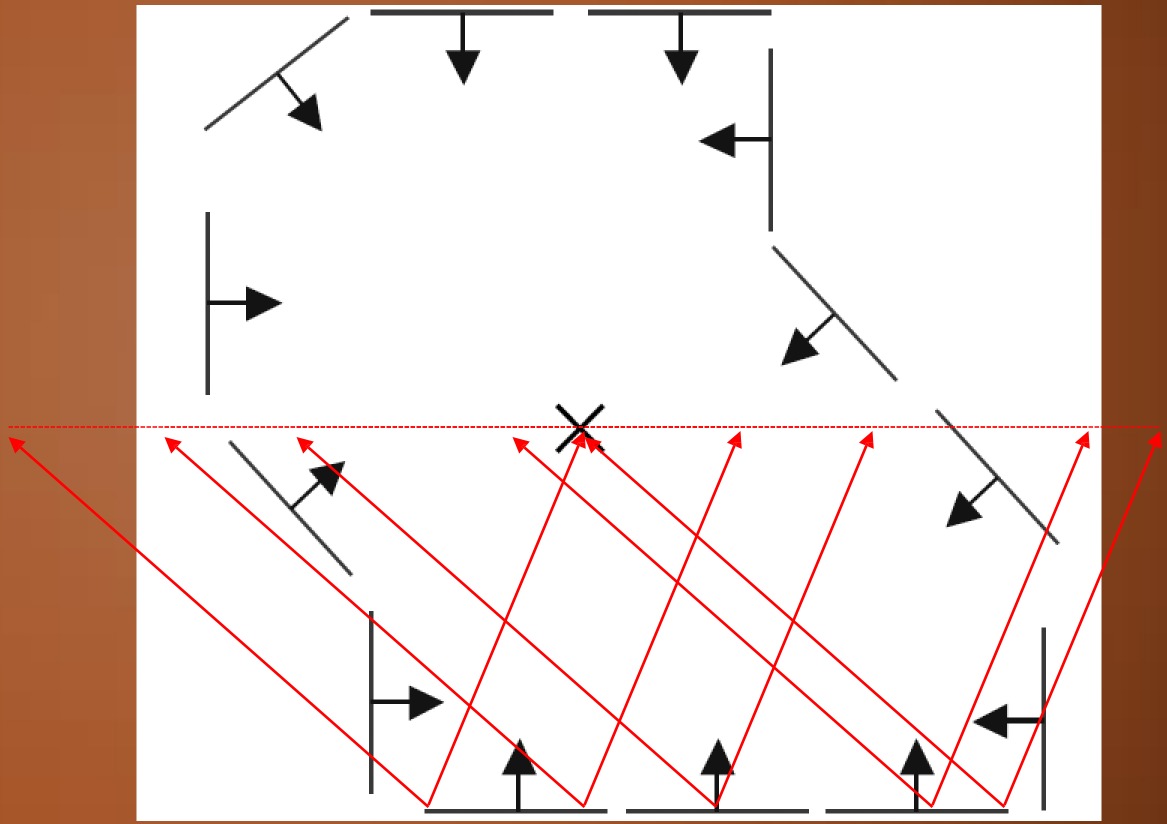
Example

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



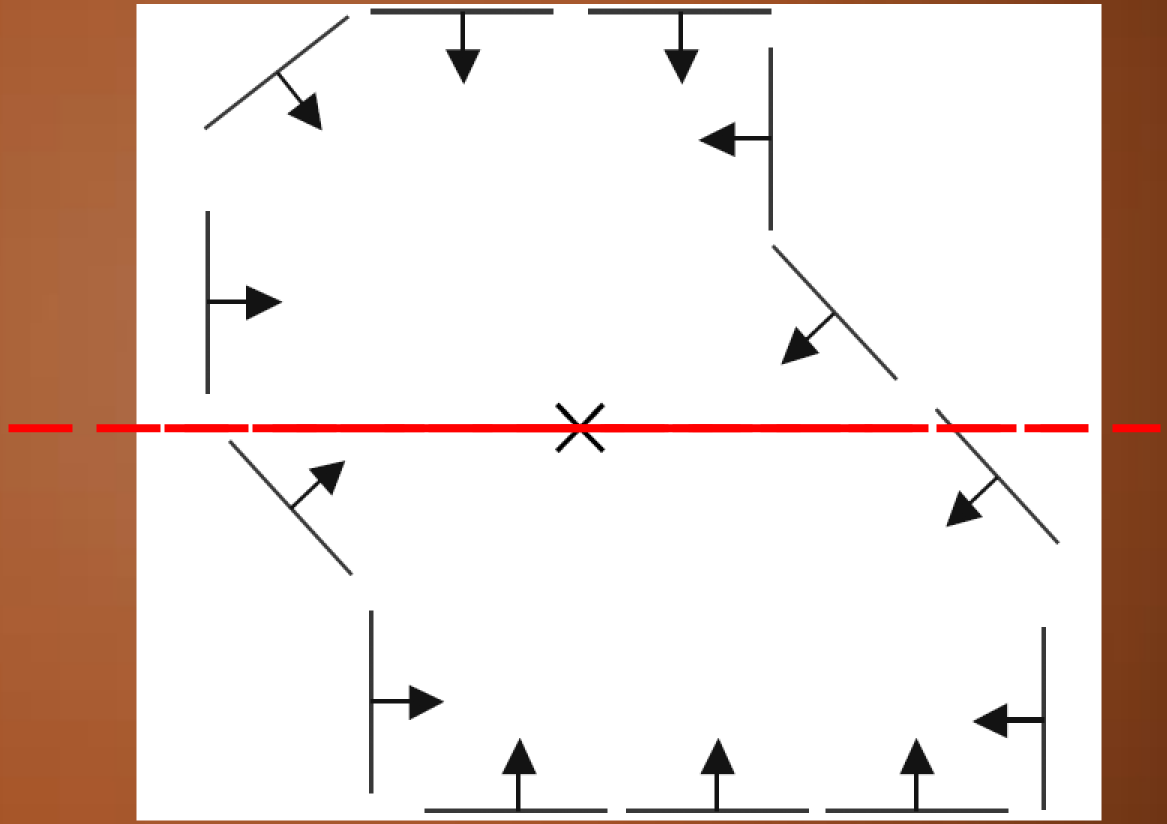
Example

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



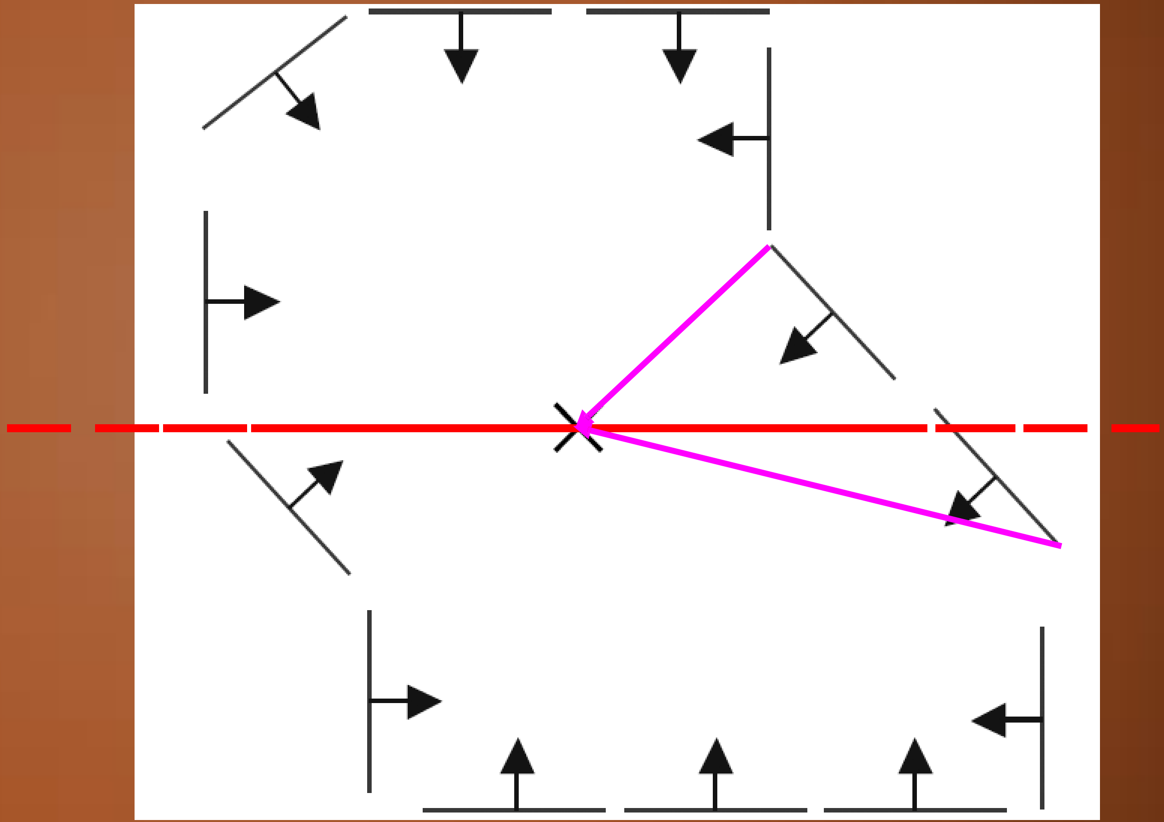
Example

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



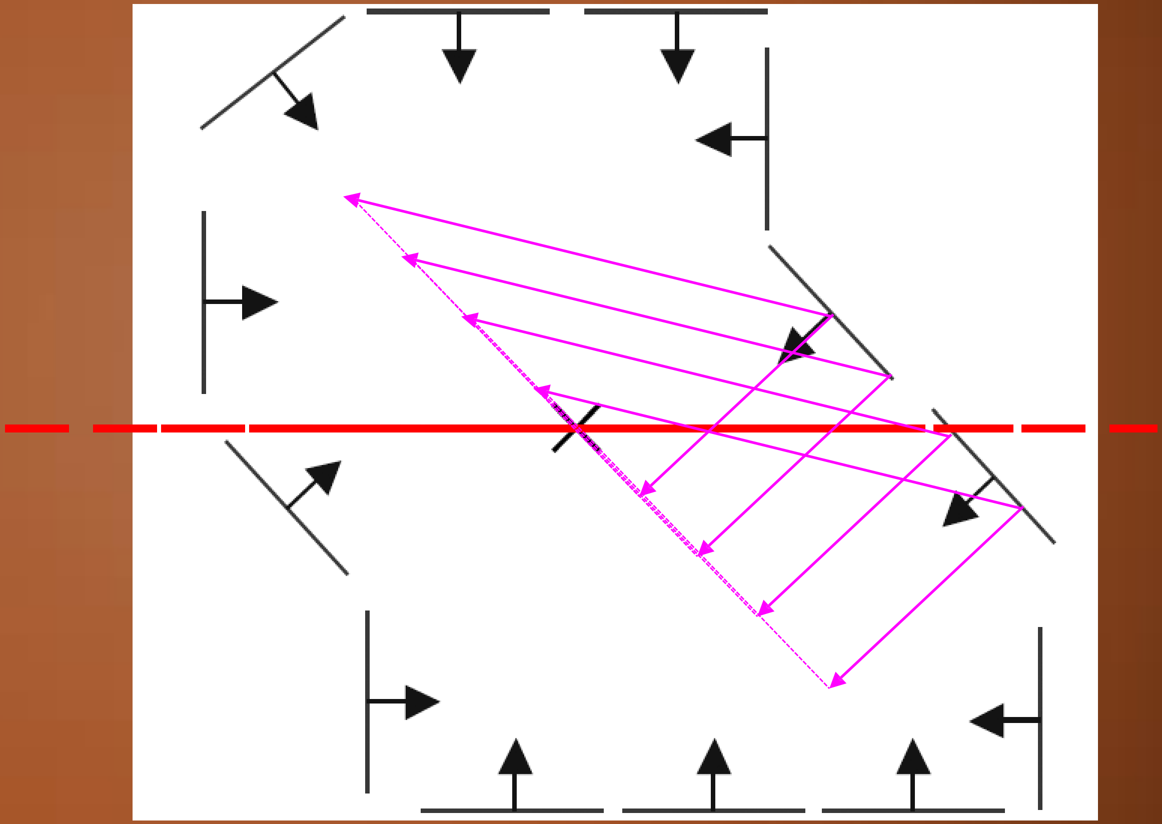
Example

Now do for the leftward pointing diagonals.



Example

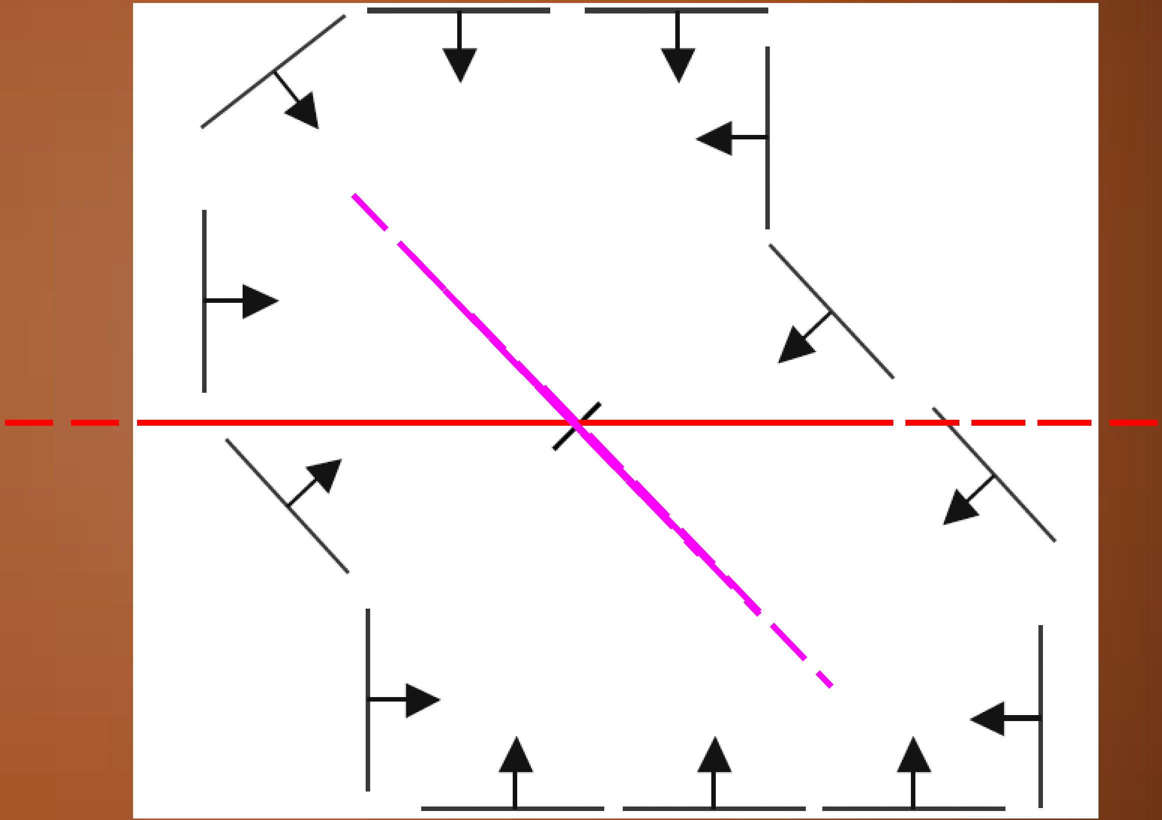
Now do for the leftward pointing diagonals.



Example

Now do for the leftward pointing diagonals.

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

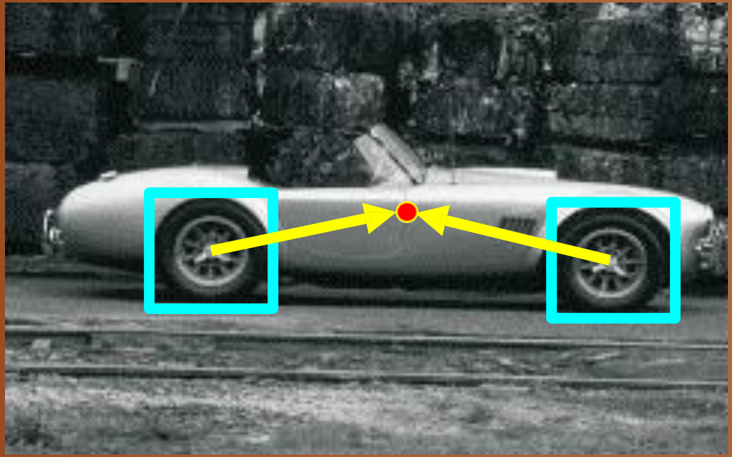
For θ' retrieve displacement vectors r

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

Application in recognition

test image