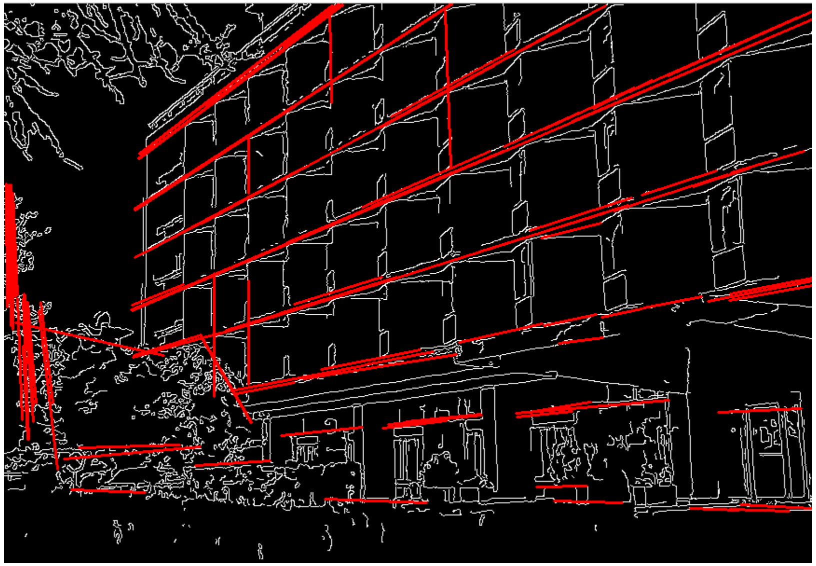
# Hough Transform



# Outline

- Hough transform
- Homography

Some slides from Lazebnik

# Voting schemes

- Let each feature vote for all the models that are compatible with it
- Hopefully the noise features will not vote consistently for any single model
- Missing data doesn’t matter as long as there are enough features remaining to agree on a good model

# The Hough Transform

- A powerful method for detecting curves from boundary information.
- Exploits the duality between points on a curve and parameters of the curve.
- Can detect analytic as well as non-analytic curves

# Hough transform

- An early type of voting scheme
- General outline:
  - Discretize parameter space **into bins**
  - For each **feature point in the image**, put a **vote** in every **bin in the parameter space** that could have generated this point
  - Find bins that have the **most votes**

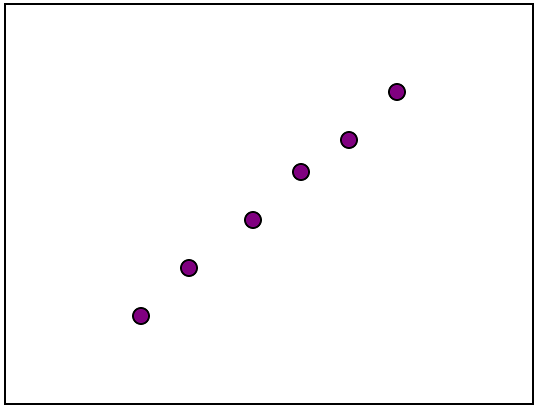
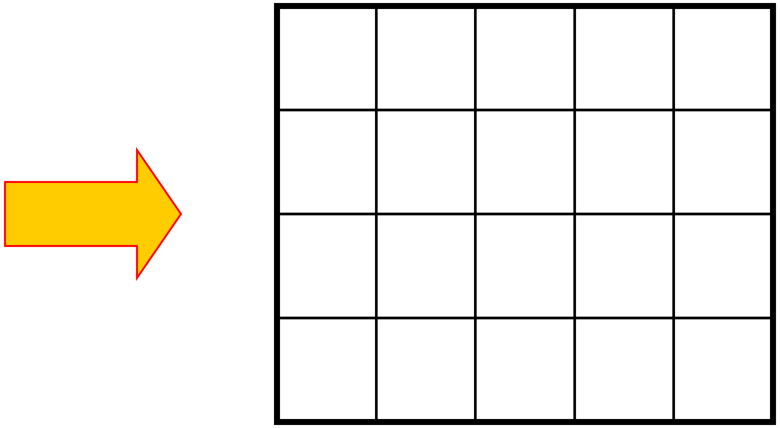


Image space

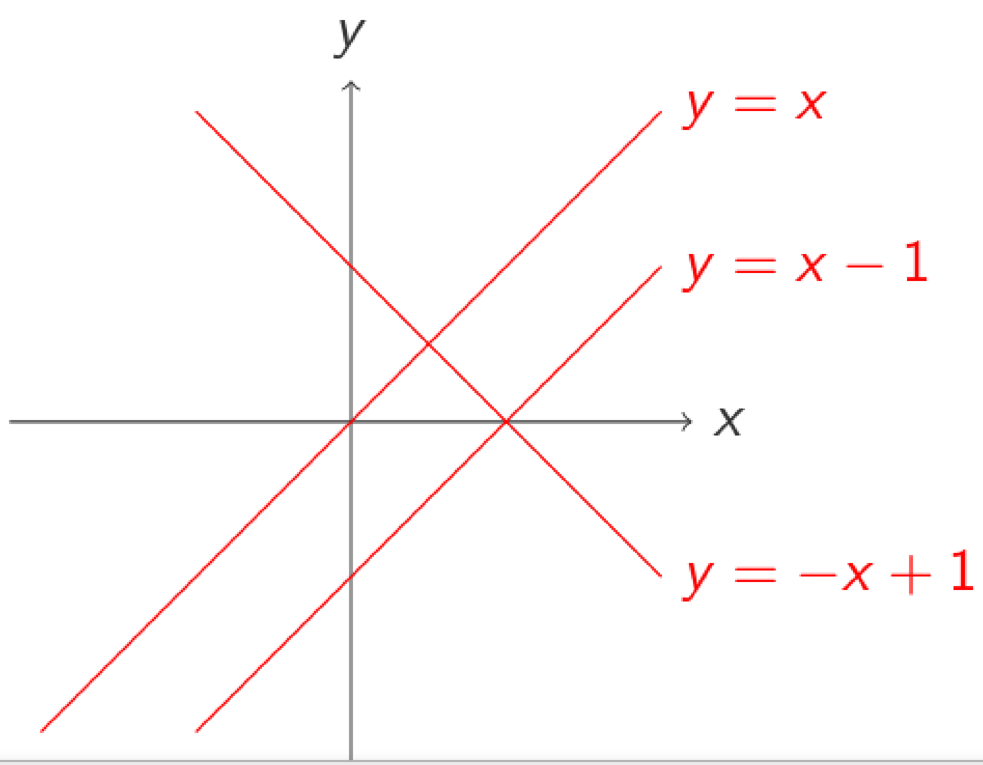


Hough parameter space

P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

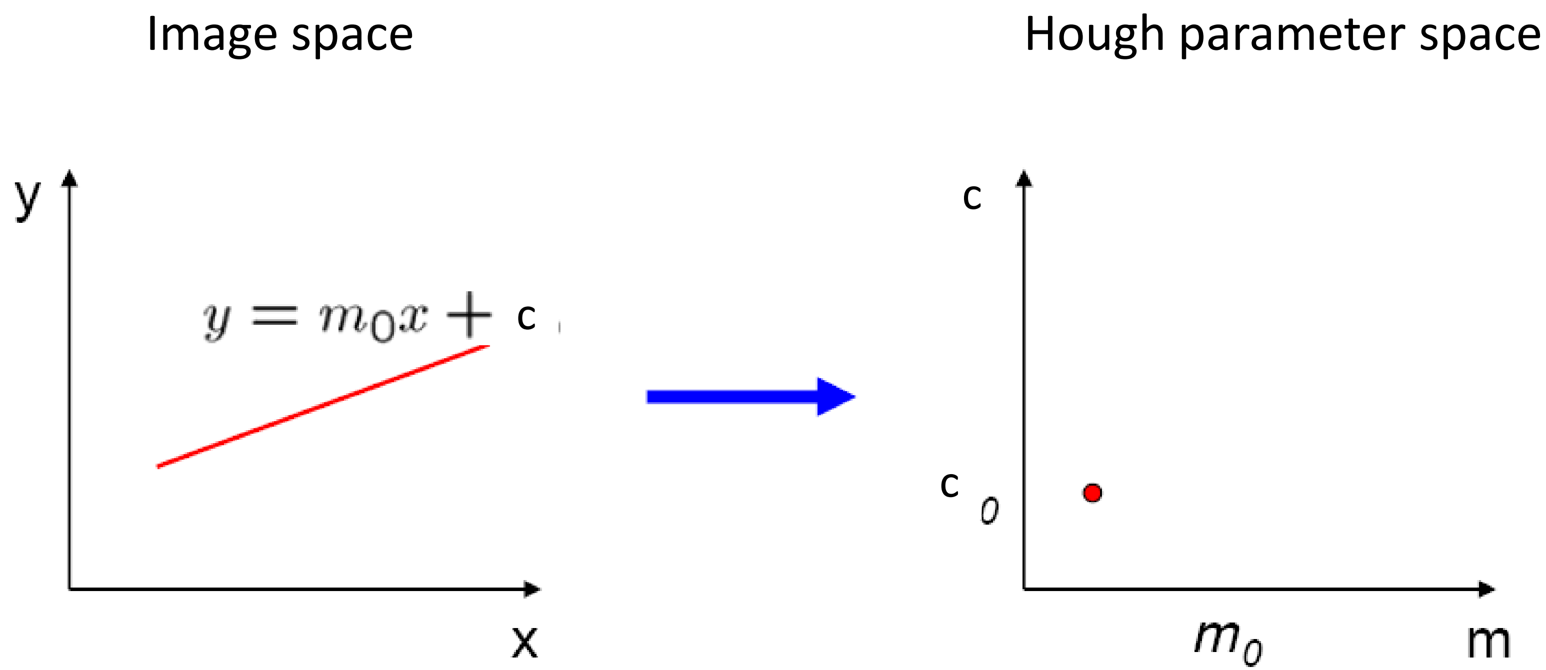
# Analytic representation of a line

- In the analytic representation of a line $y = mx + c$, every choice of parameters $(m, c)$ represents a different line.
- This is known as the *slope-intercept* parameter space.
- Weakness: vertical lines have $m = \infty$.



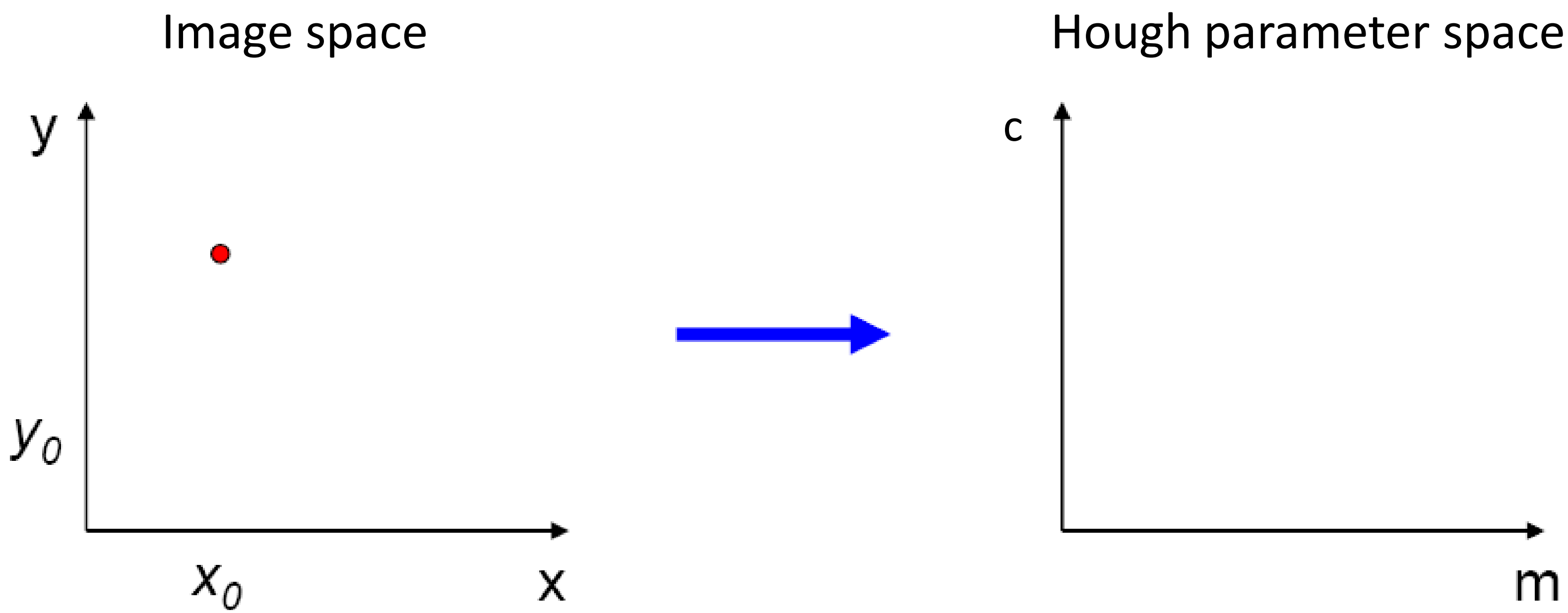
# Parameter space representation

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



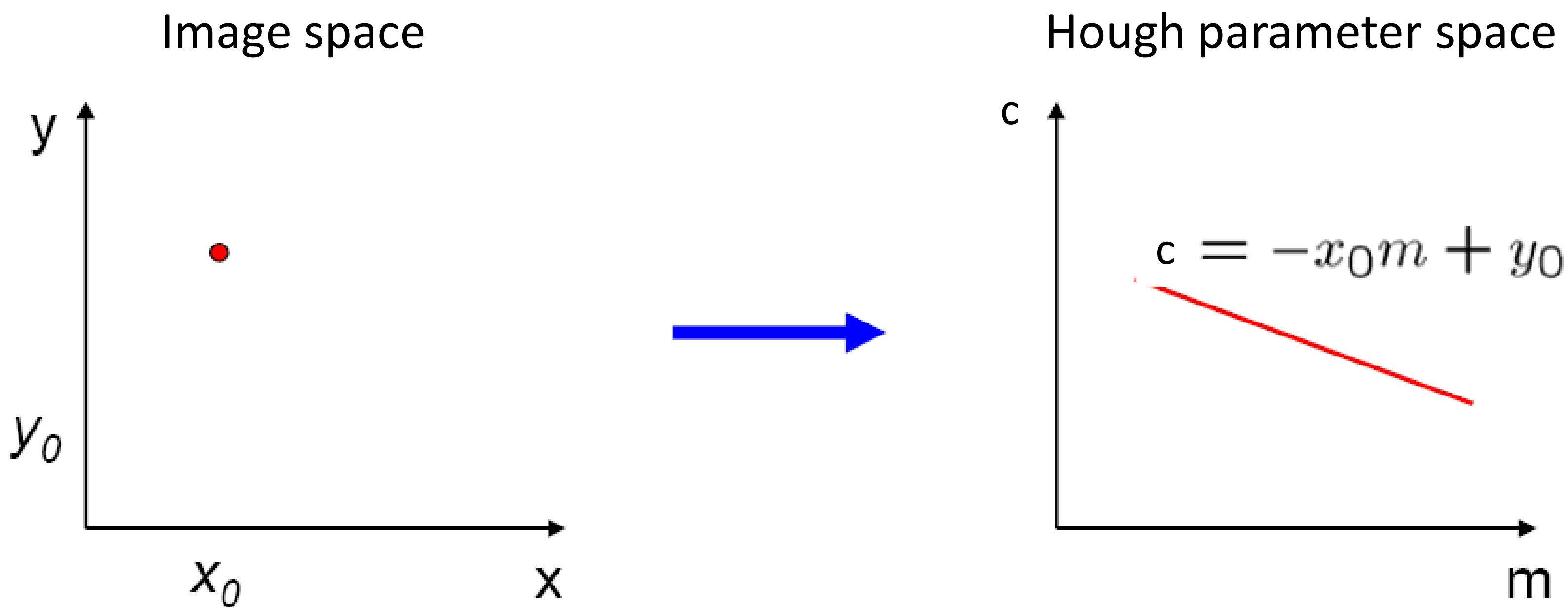
# Parameter space representation

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



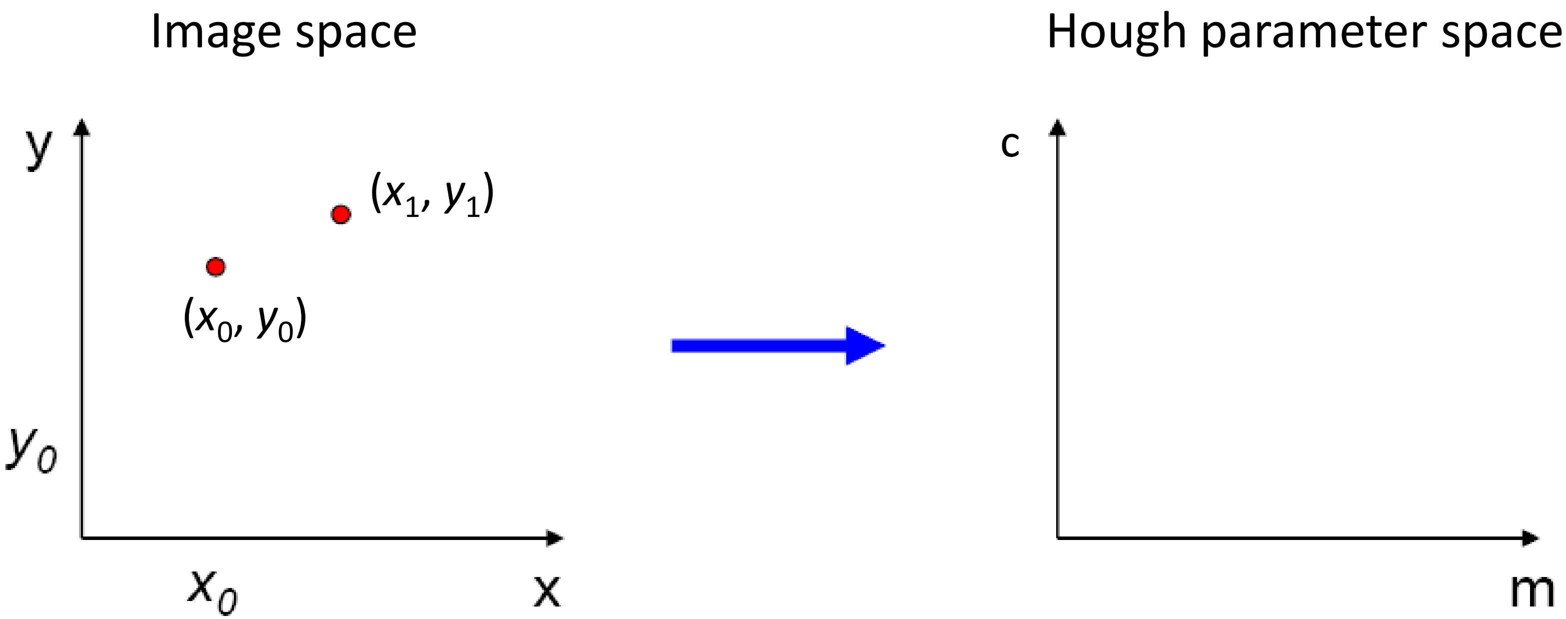
# Parameter space representation

- What does a point $(x_0, y_0)$ in the image space map to in the Hough space?
  - Answer: the solutions of $c = -x_0 m + y_0$
  - This is a line in Hough space



# Parameter space representation

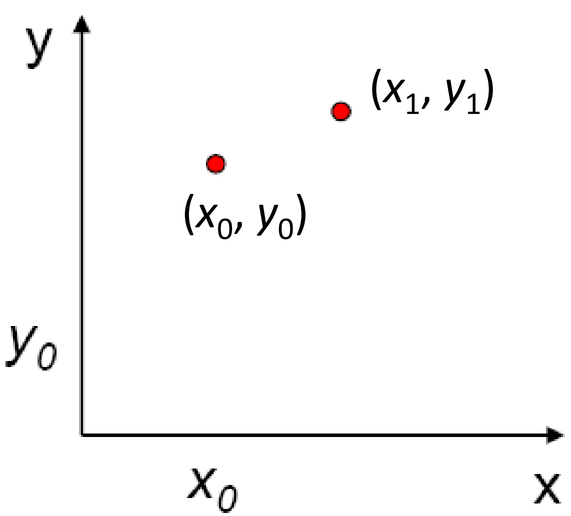
- Where is the line that contains both $(x_0, y_0)$ and $(x_1, y_1)$?



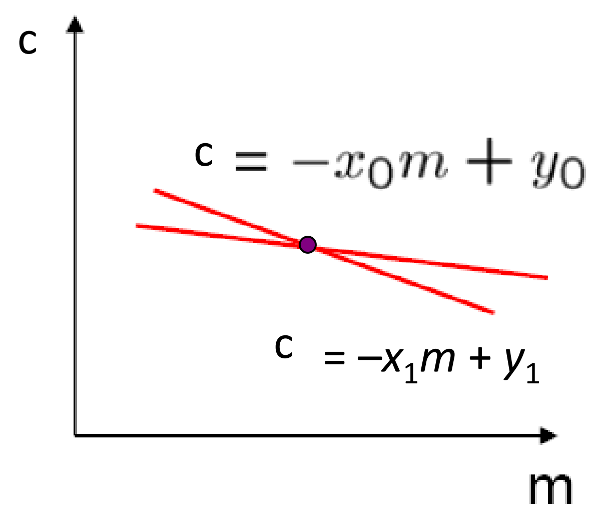
# Parameter space representation

- Where is the line that contains both $(x_0, y_0)$ and $(x_1, y_1)$?
  - It is the intersection of the lines $c = -x_0m + y_0$ and $c = -x_1m + y_1$

Image space

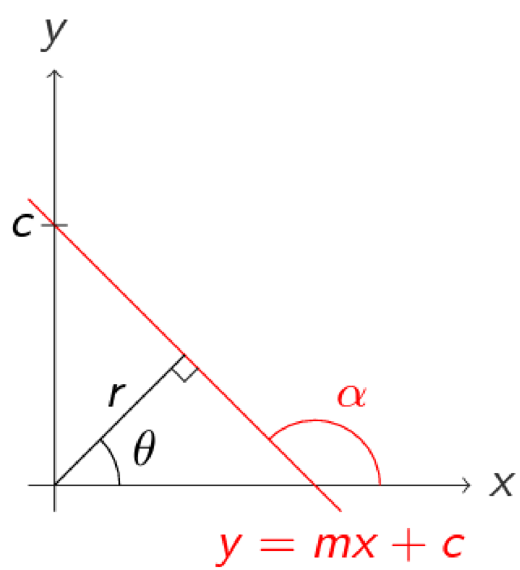



Hough parameter space



# Parameter space representation

- Problems with the (m,c) space:
  - Unbounded parameter domain
  - Vertical lines require infinite m
- Alternative: polar representation




$$r = x\cos(\theta) + y\sin(\theta)$$

Each point will add a sinusoid in the $(\theta,\rho)$ parameter space

# Polar representation of a line

- Solution: Polar representation $(r, \theta)$ where
  - $r =$ perpendicular distance of line from origin
  - $\theta =$ angle of vector orthogonal to the line
- Every $(r, \theta)$ pair represents a 2D line.

**Additional formulae**

$\sin(A + B) = \sin A \cos B + \sin B \cos A$

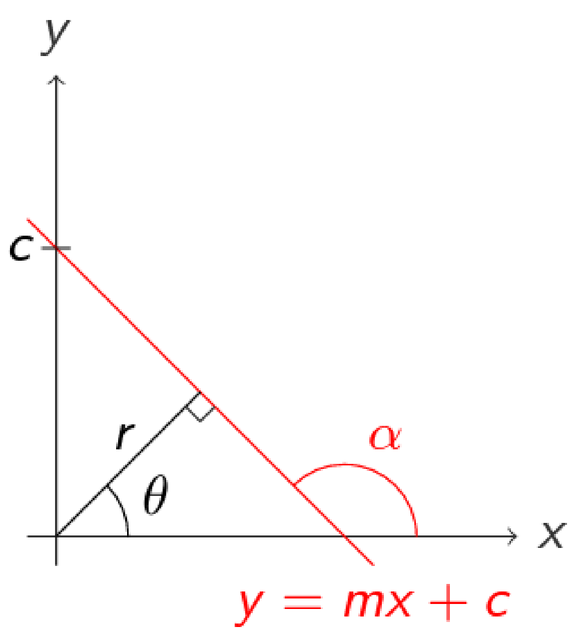
$\sin(A - B) = \sin A \cos B - \sin B \cos A$

$\cos(A + B) = \cos A \cos B - \sin A \sin B$

$\cos(A - B) = \cos A \cos B + \sin A \sin B$

$$\tan(A+B) = \frac{\tan A + \tan B}{1 - \tan A \tan B}$$


$$\tan(A+B) = \frac{\tan A + \tan B}{1 - \tan A \tan B}$$


$$y = mx + c$$

$$m = \tan(\alpha) = \tan(\theta + \frac{\pi}{2})$$

$$= \frac{\sin(\theta + \frac{\pi}{2})}{\cos(\theta + \frac{\pi}{2})} = \frac{\cos(\theta)}{-\sin(\theta)}$$

$$c = \frac{r}{\sin(\theta)}$$

$$y = -\frac{\cos(\theta)}{\sin(\theta)}x + \frac{r}{\sin(\theta)}$$

$$r = x\cos(\theta) + y\sin(\theta)$$

# Hough Transform for Line Detection

- An algorithm for finding lines given some edge points.
- Given point $(x, y)$, line passing through it with angle $\theta$ must have perpendicular $r = x\cos(\theta) + y\sin(\theta)$.
- Given any edge pixel $(x, y)$, potentially 180 lines could pass through it assuming angular resolution of $1^\circ$.
- Looping through the angles gives $(r, \theta)$ pairs for all lines through $(x, y)$.
- So pixel $(x, y)$ should *vote for* all those lines.

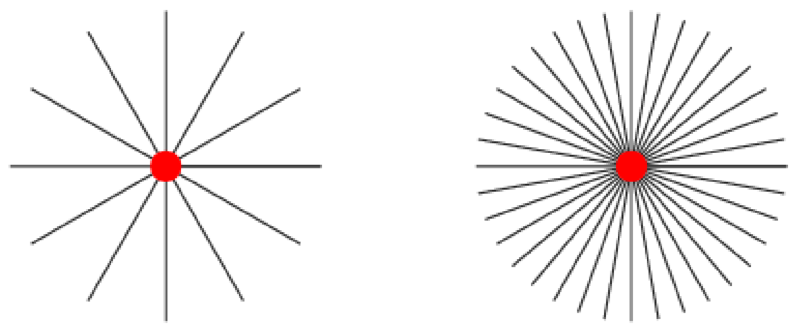


**Figure:** Lines passing through a point. **Left**: Angular resolution of $30^\circ$. **Right**: Angular resolution of $10^\circ$. Author: N. Khan (2021)

# Hough Transform for Line Detection

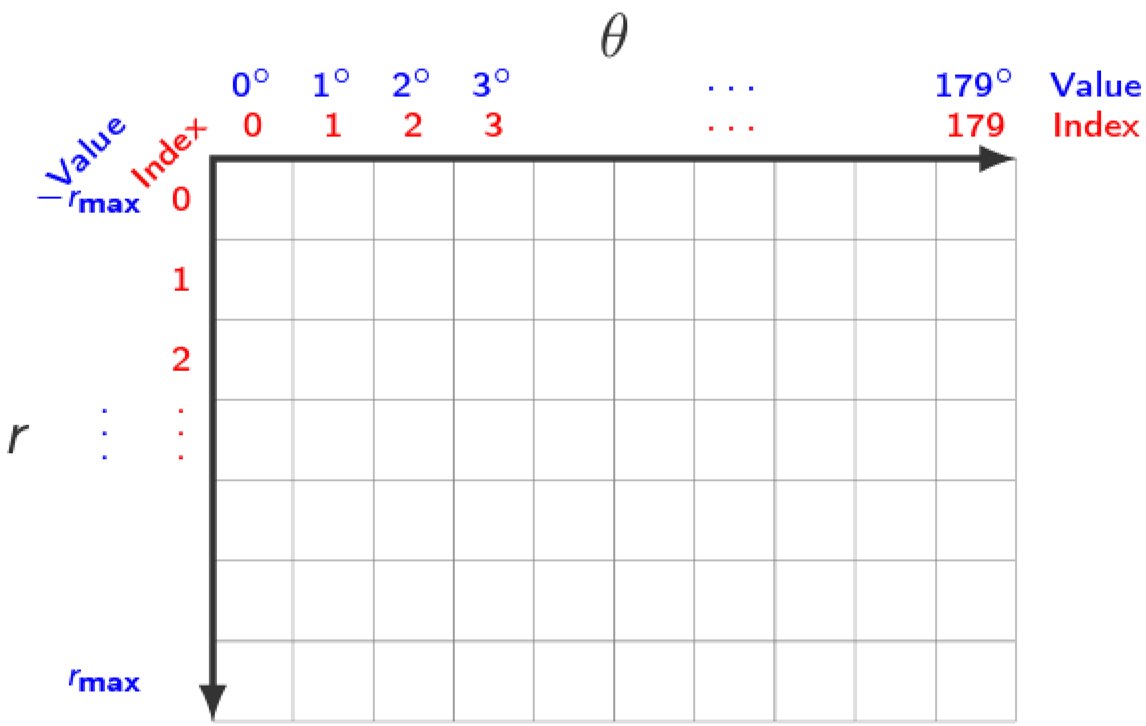



**Figure:** The accumulator array used to gather votes for each line. Each $(r, \theta)$ p
needs to be quantized into bin-indices before casting a vote. Author: N. Khan (2

## Hough Transform for Line Detection

- By repeating this process for all edge pixels, actual lines will get a high number of votes.

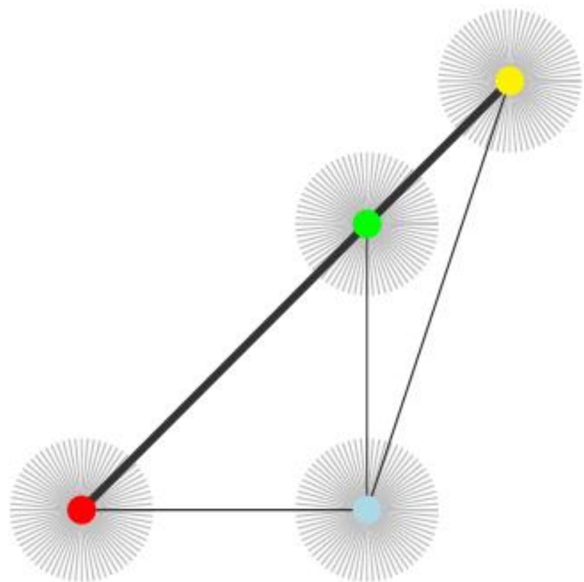


**Figure:** Each point votes for every line that passes through it. Genuine lines will get more votes. Author: N. Khan (2021)

# Hough Transform for Line Detection

- Pseudocode

initialize 2D (vote) accumulator array $A$ to all zeros.
for every edge point $(x, y)$
    for $\theta = 0$ to $\pi$
        compute $r = x\cos(\theta) + y\sin(\theta)$
        compute indices $(r_{\text{ind}}, \theta_{\text{ind}})$ corresponding to $(r, \theta)$
        increment $A(r_{\text{ind}}, \theta_{\text{ind}})$ by 1 $\longleftarrow$ vote of point $(x, y)$ for line $(r, \theta)$
valid lines are where $A >$ threshold

# Hough Transform for Line Detection (detailed Pseudocode)

1. $\theta_{\text{range}} = 180^\circ$
2. $\theta_{\text{binsize}} = 1^\circ$ (for example)
3. $\theta_{\text{size}} = \left\lceil \frac{\theta_{\textbf{range}}}{\theta_{\textbf{binsize}}} \right\rceil$
4. $r_{\text{max}} =$ length of image diagonal
5. $r_{\text{range}} = 2r_{\text{max}}$
6. $r_{\text{binsize}} = 1$ pixel (for example)
7. $r_{\text{size}} = \left\lceil \frac{r_{\textbf{range}}}{r_{\textbf{binsize}}} \right\rceil$
8. initialize 2D (vote) accumulator array $A$ of size $(r_{\text{size}}, \theta_{\text{size}})$ to all zeros.
9. for every edge point $(x, y)$
10. for $\theta = 0$ to $\theta_{\text{range}}$
11. compute $r = x\cos(\theta) + y\sin(\theta)$
12. $r_{\text{ind}} = \text{round}\left(\frac{r + r_{\textbf{max}}}{r_{\textbf{binsize}}}\right)$
13. $\theta_{\text{ind}} = \text{round}\left(\frac{\theta \bmod 180}{\theta_{\textbf{binsize}}}\right)$
14. increment $A(r_{\text{ind}}, \theta_{\text{ind}})$ by 1 $\longleftarrow$ vote of point $(x, y)$ for line $(r, \theta)$

# Hough Transform for Line Detection (detailed Pseudocode)

15. smooth votes via Gaussian convolution with kernel $G_{\sigma_h}$ to account for uncertainties in the gradient direction
16. perform non-maxima suppression in $k \times k$ neighborhoods to remove fake lines around real ones
17. valid lines[1] are where $A > \tau$ which can be computed as a percentile

# Improvement

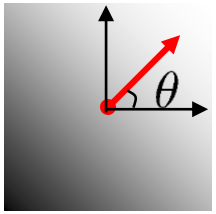
After edge detection, we already know the gradient direction at $(x, y)$.

- So there is no need to iterate over all possible $\theta$.
- Use the correct $\theta$ from the gradient direction.
- This removes the loop at step 10.
- Pixel $(x, y)$ only votes for the line that was actually passing through it.

This speeds-up the algorithm.

# Extension: 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!




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

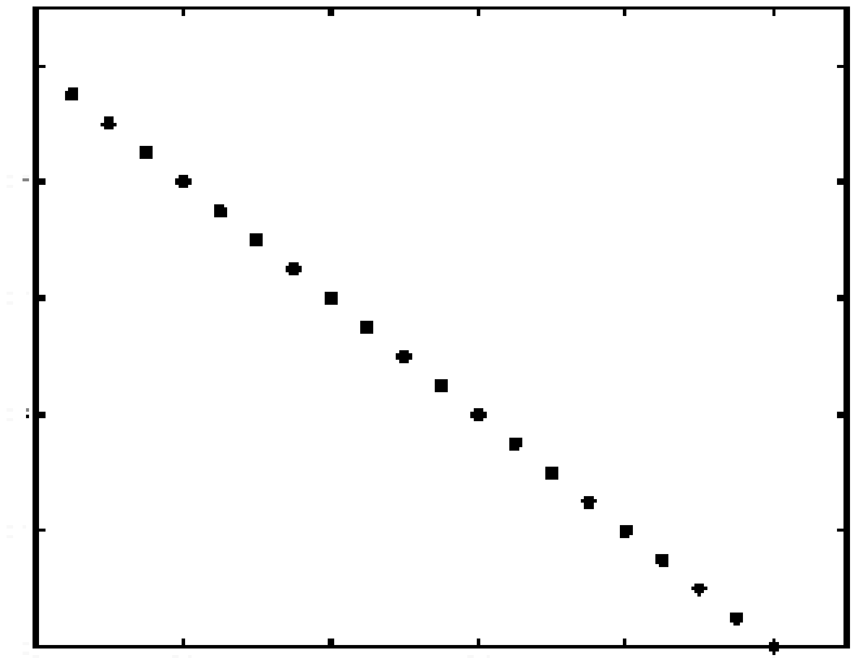
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- Modified Hough transform:

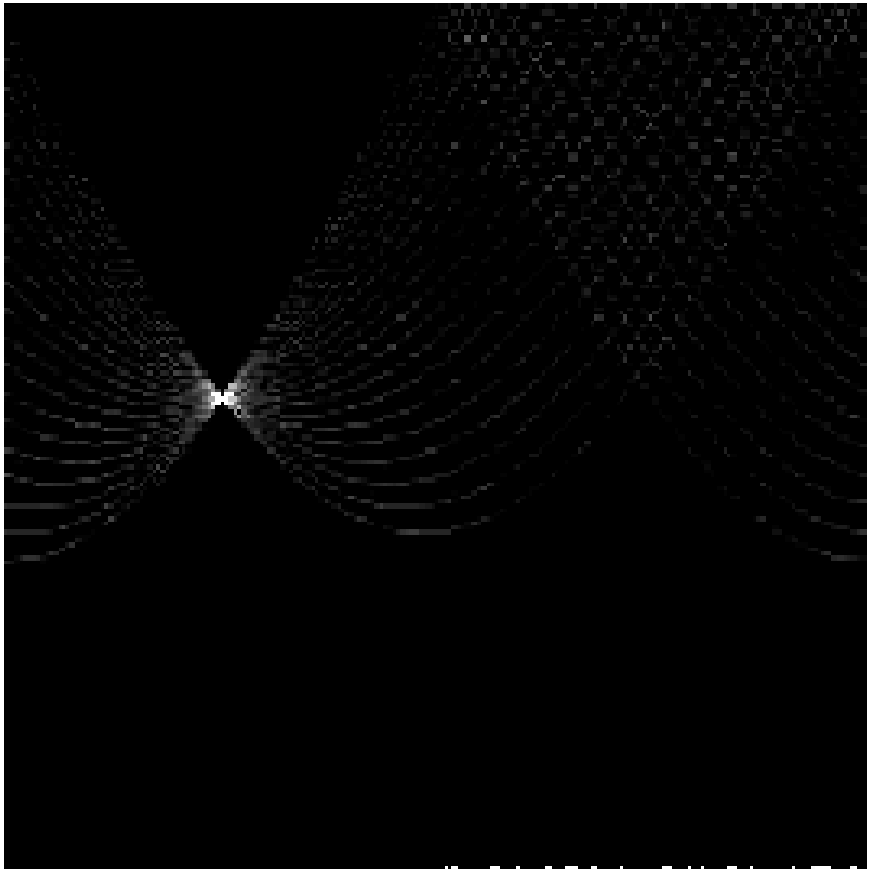
- For each edge point (x,y)
  - θ = gradient orientation at (x,y)
  - r = x cos θ + y sin θ
  - H(θ, r) = H(θ, r) + 1

  end

# Basic illustration



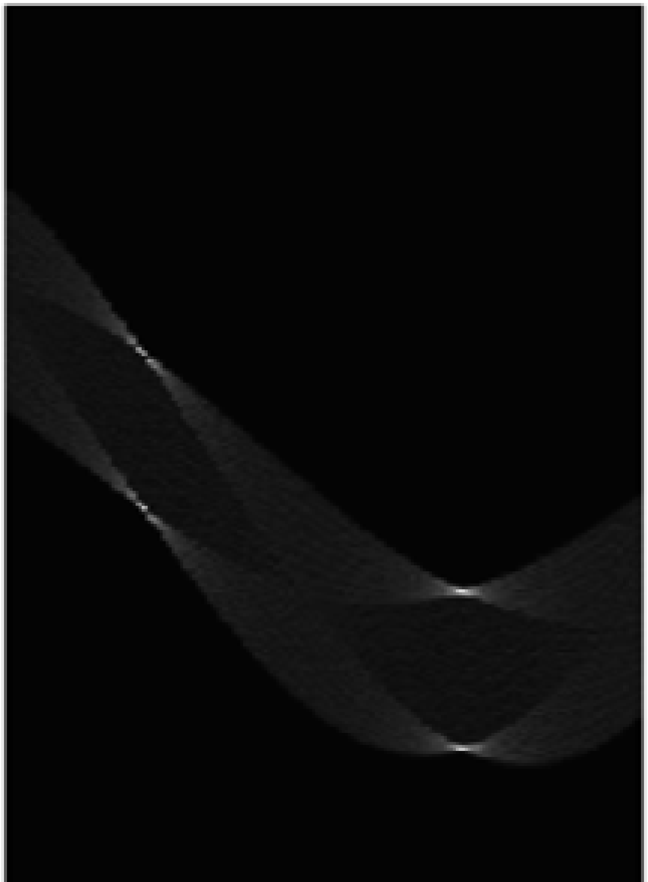
features



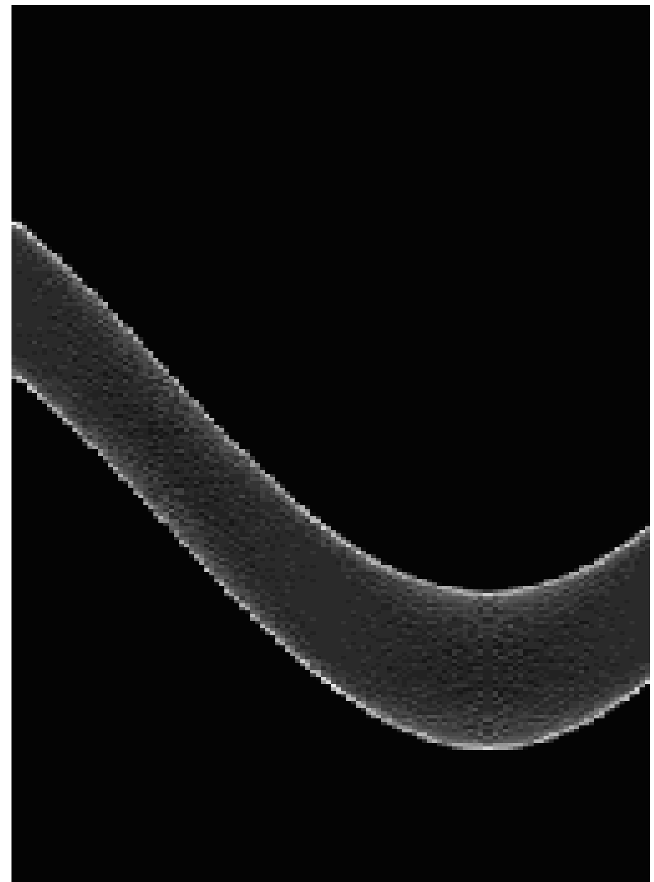
votes

# Other shapes

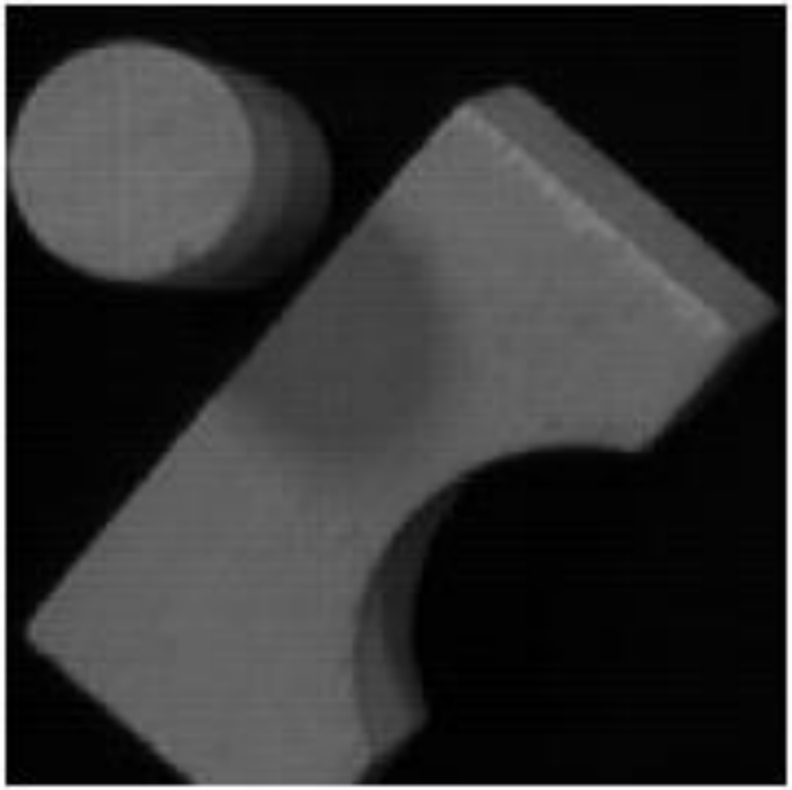
Square

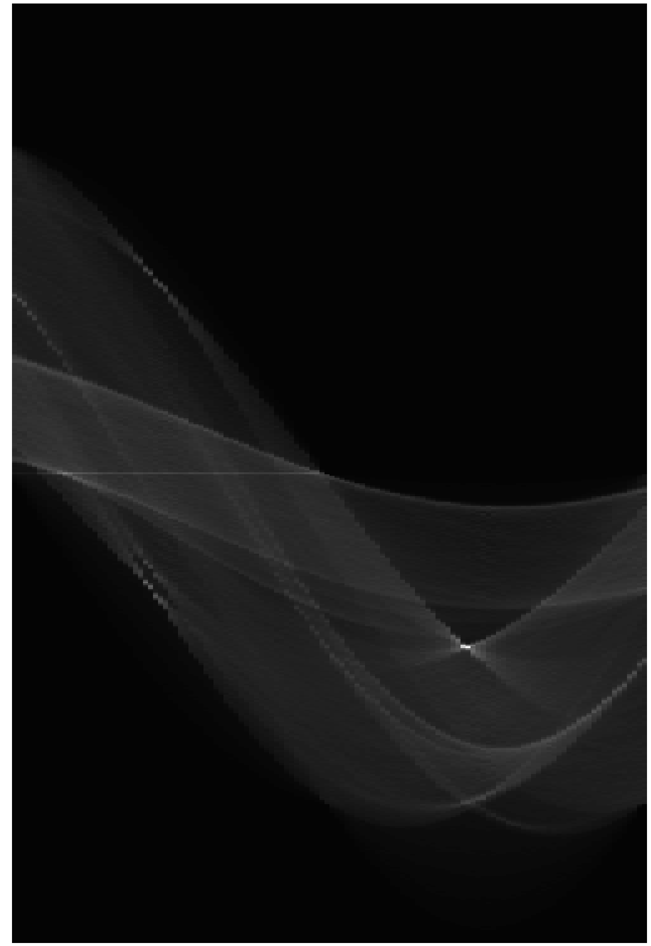


Circle

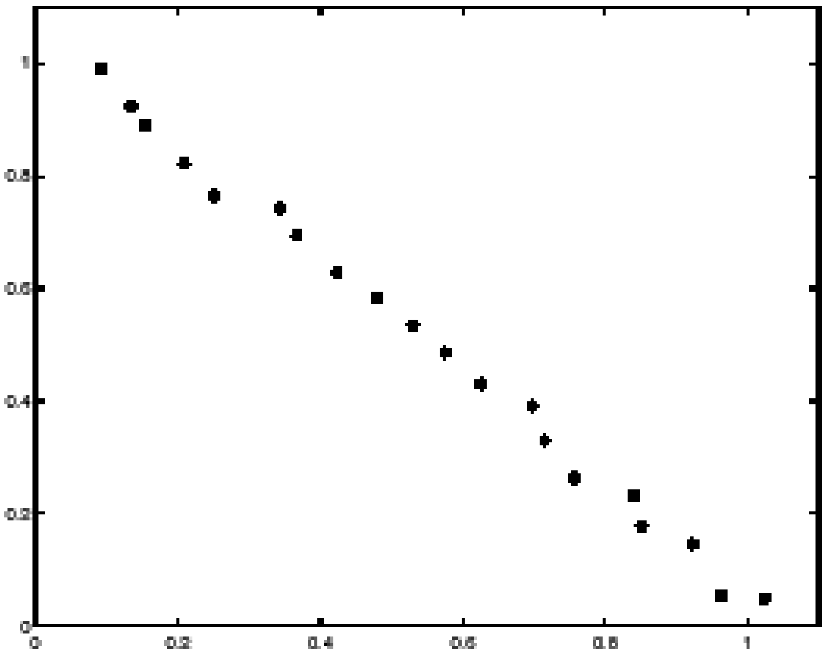


# Several lines



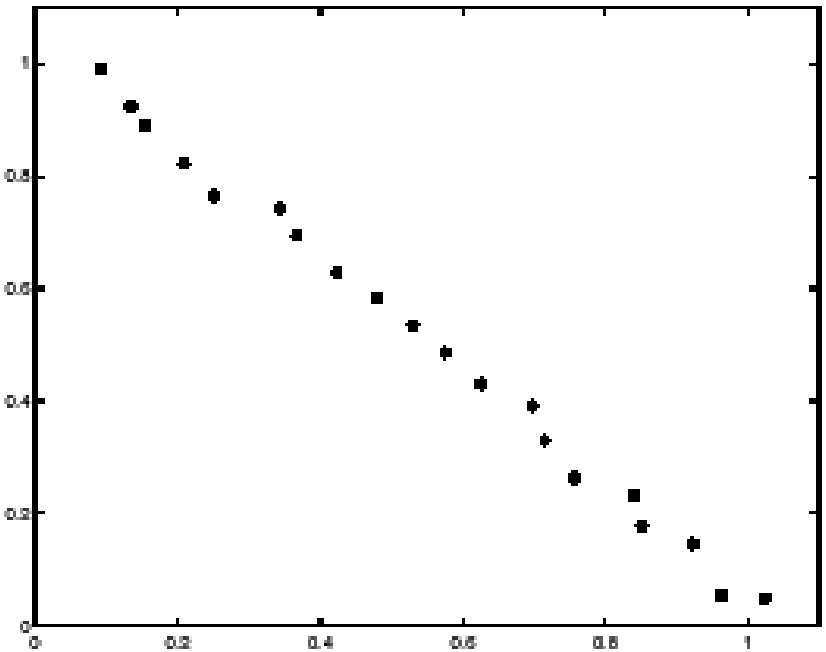


# Effect of noise

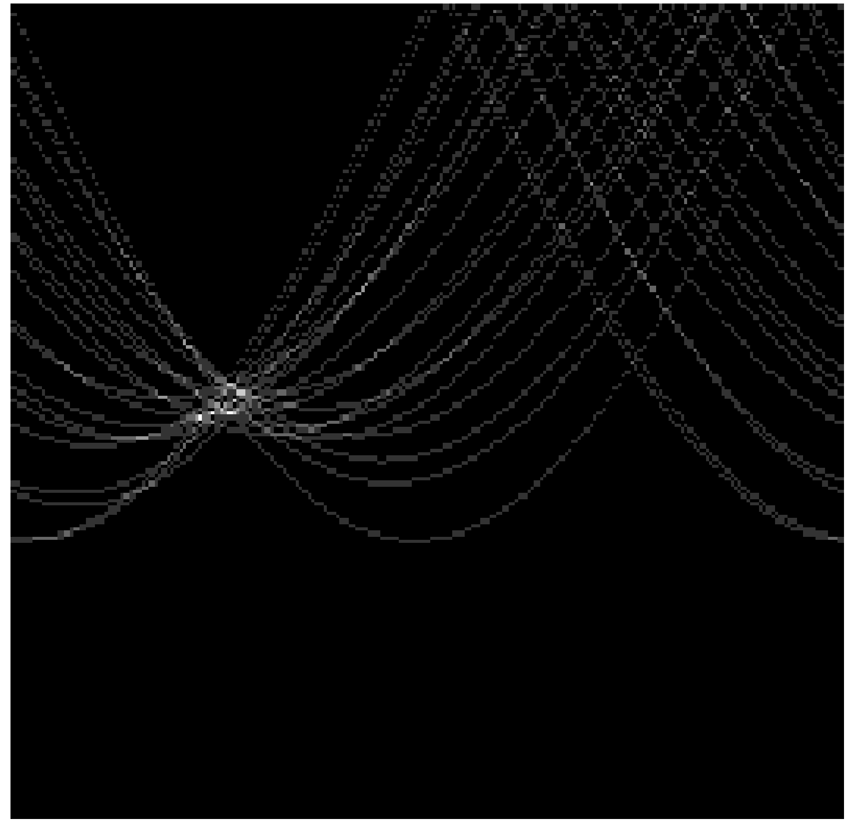



features

# Effect of noise




features

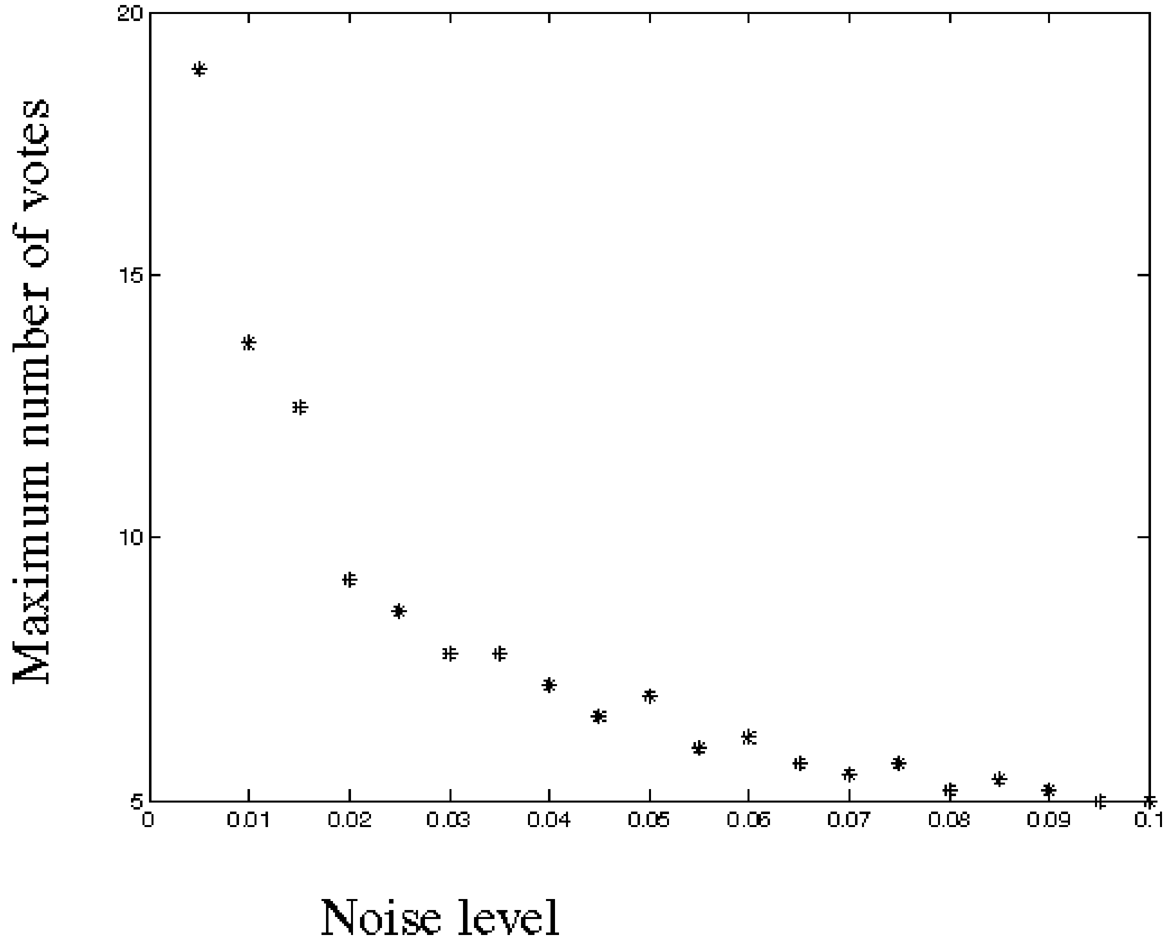


votes

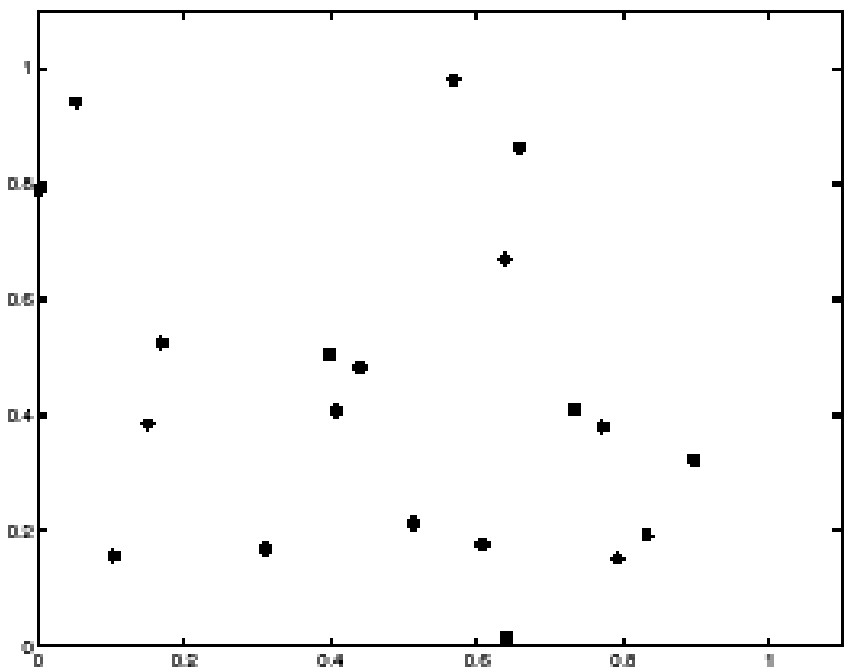
- Peak gets fuzzy and hard to locate

# Effect of noise

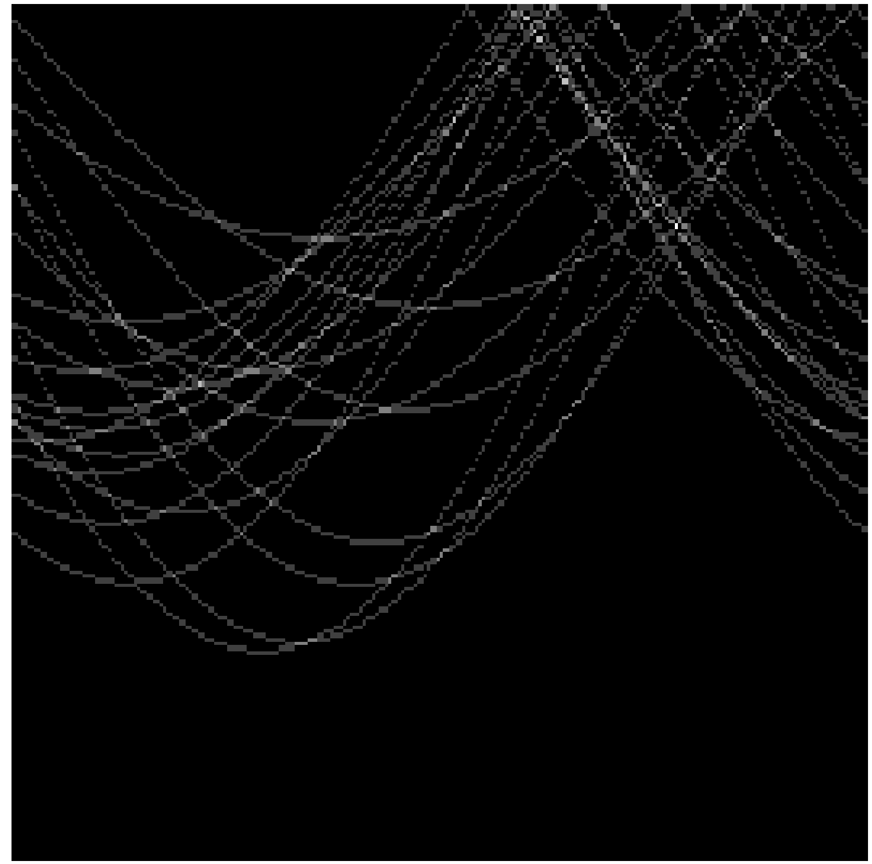
- Number of votes for a line of 20 points with increasing noise:



# Random points




features

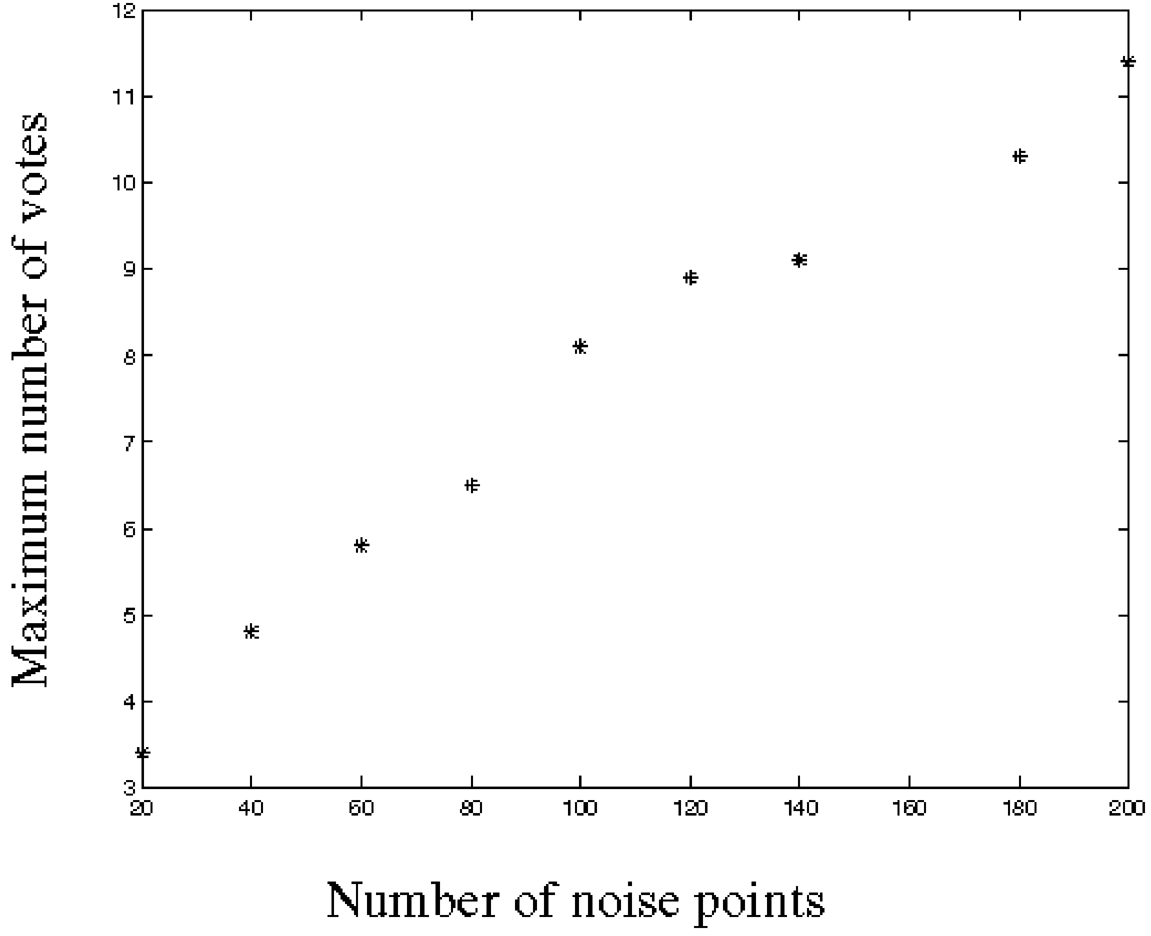


votes

- 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:



# Hough transform: Pros

- Can deal with non-locality and occlusion
- Can detect multiple instances of a model in a single pass
- Some robustness to noise: noise points unlikely to contribute consistently to any single bin

# 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
- It’s hard to pick a good grid size

# Results


Original

Using edge pixels only

$\tau$ = 95-th percentile

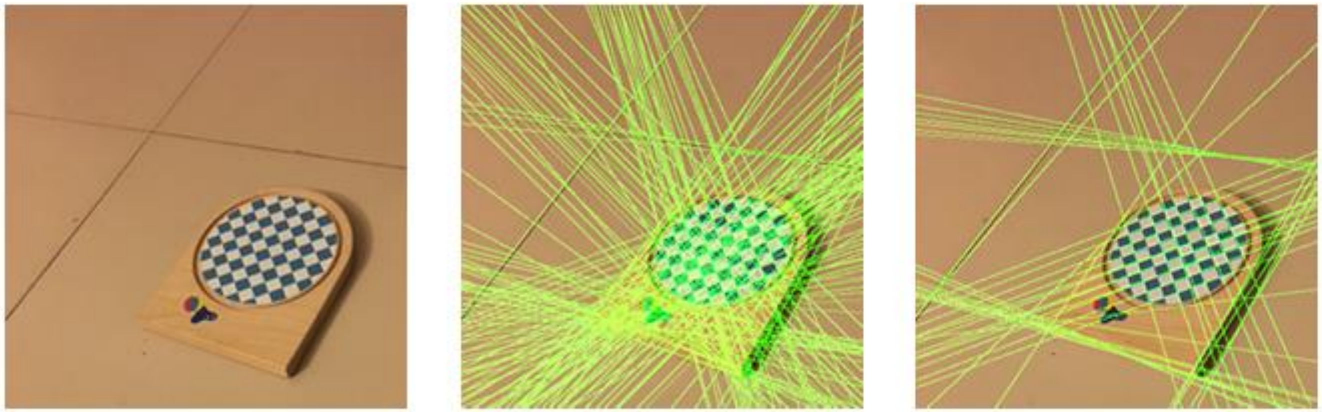
Using edge pixels and

gradient orientations

$\tau$ = 70-th percentile

**Figure:** Line detection via Hough transform. Canny parameters: $\sigma_e = 1$, $t_h = 80$-th percentile, $t_l = 40$-th percentile. Hough parameters: $\sigma_h = \frac{\sigma_e}{5}$, $k = 3$. Author: N. Khan (2021)

# Results



Original

Using edge pixels only

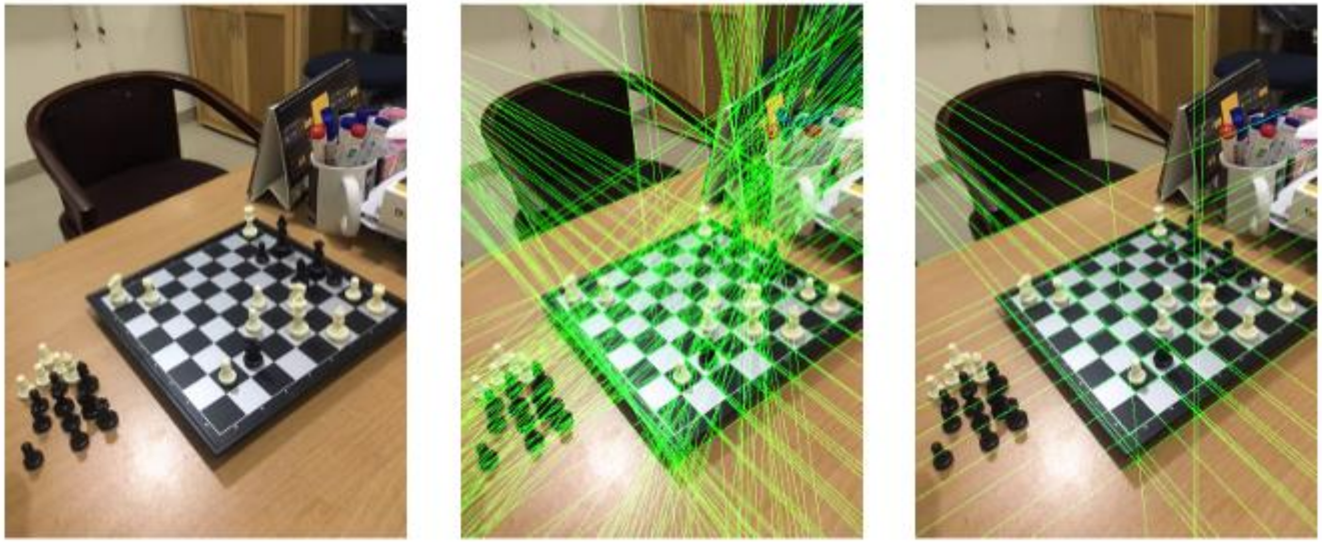
$\tau$ = 95-th percentile

Using edge pixels and gradient orientations

$\tau$ = 70-th percentile

**Figure:** Line detection via Hough transform. Canny parameters: $\sigma_e = 1$, $t_h =$ 80-th percentile, $t_l =$ 40-th percentile. Hough parameters: $\sigma_h = \frac{\sigma_e}{5}$, $k = 3$. Author: N. Khan (2021)

# Results



Original

Using edge pixels only

$\tau$ = 95-th percentile

Using edge pixels and

gradient orientations

$\tau$ = 90-th percentile

**Figure:** Line detection via Hough transform. Canny parameters: $\sigma_e = 1$, $t_h$ = 80-th percentile, $t_l$ = 40-th percentile. Hough parameters: $\sigma_h = \frac{\sigma_e}{5}$, $k = 3$. Author: N. Khan (2021)

# Hough transform for circles

- How many dimensions will the parameter space have?
- Given an oriented edge point, what are all possible bins that it can vote for?

# Hough Transform for Circle Detection

- Analytic representation of circle of radius $r$ centered at $(a, b)$ is $(x-a)^2 + (y-b)^2 - r^2 = 0$.
- Hough space has 3 parameters $(a, b, r)$.

**Pseudocode**

For every boundary point $(x, y)$
    For every $(a, b)$ in image plane
        Compute $r(a, b) = \sqrt{(x-a)^2 + (y-b)^2}$
        Compute $a_{\text{ind}}, b_{\text{ind}}$ and $r_{\text{ind}}$
        Increment $A(a_{\text{ind}}, b_{\text{ind}}, r_{\text{ind}})$ by 1

$\mathrm{NMS}(A * G_{\sigma_h}) > \tau$ represents valid circles.

- If we know the gradient vector $\nabla I(x, y)$ at point $(x, y)$, then we also know that the center $(a, b)$ can only lie along this line.
- Hough space still has 3 parameters $(a, b, r)$ but we search for $r$ over a 1D space instead of a 2D plane.

**Pseudocode**

For every boundary point $(x, y)$

For every $(a, b)$ along gradient vector $\nabla I(x, y)$

Compute $r(a, b) = \sqrt{(x-a)^2 + (y-b)^2}$

Compute $a_{\text{ind}}, b_{\text{ind}}$ and $r_{\text{ind}}$

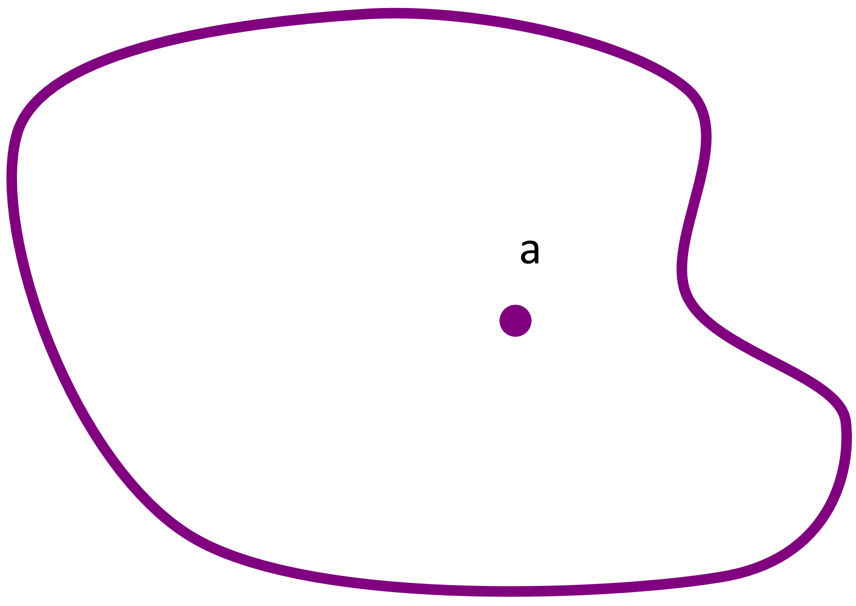
Increment $A(a_{\text{ind}}, b_{\text{ind}}, r_{\text{ind}})$ by 1

$\text{NMS}(A * G_{\sigma_h}) > \tau$ represents valid circles.

# Questions

# Generalized Hough transform

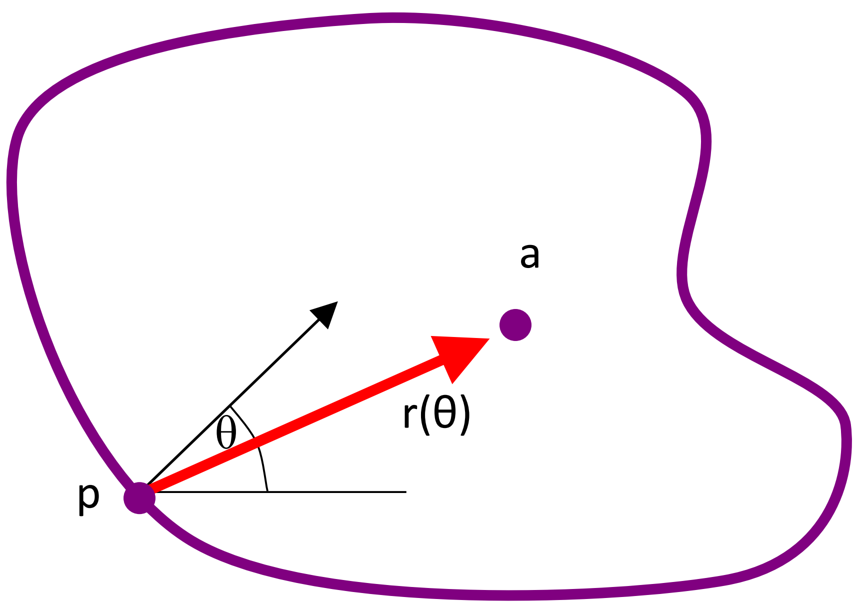
- We want to find a shape defined by its boundary points and a reference point




D. Ballard, [Generalizing the Hough Transform to Detect Arbitrary Shapes](), Pattern Recognition 13(2), 1981, pp. 111-122.

# Generalized Hough transform

- We want to find a shape defined by its boundary points and a reference point
- For every boundary point p, we can compute the displacement vector r = a – p as a function of gradient orientation $\theta$



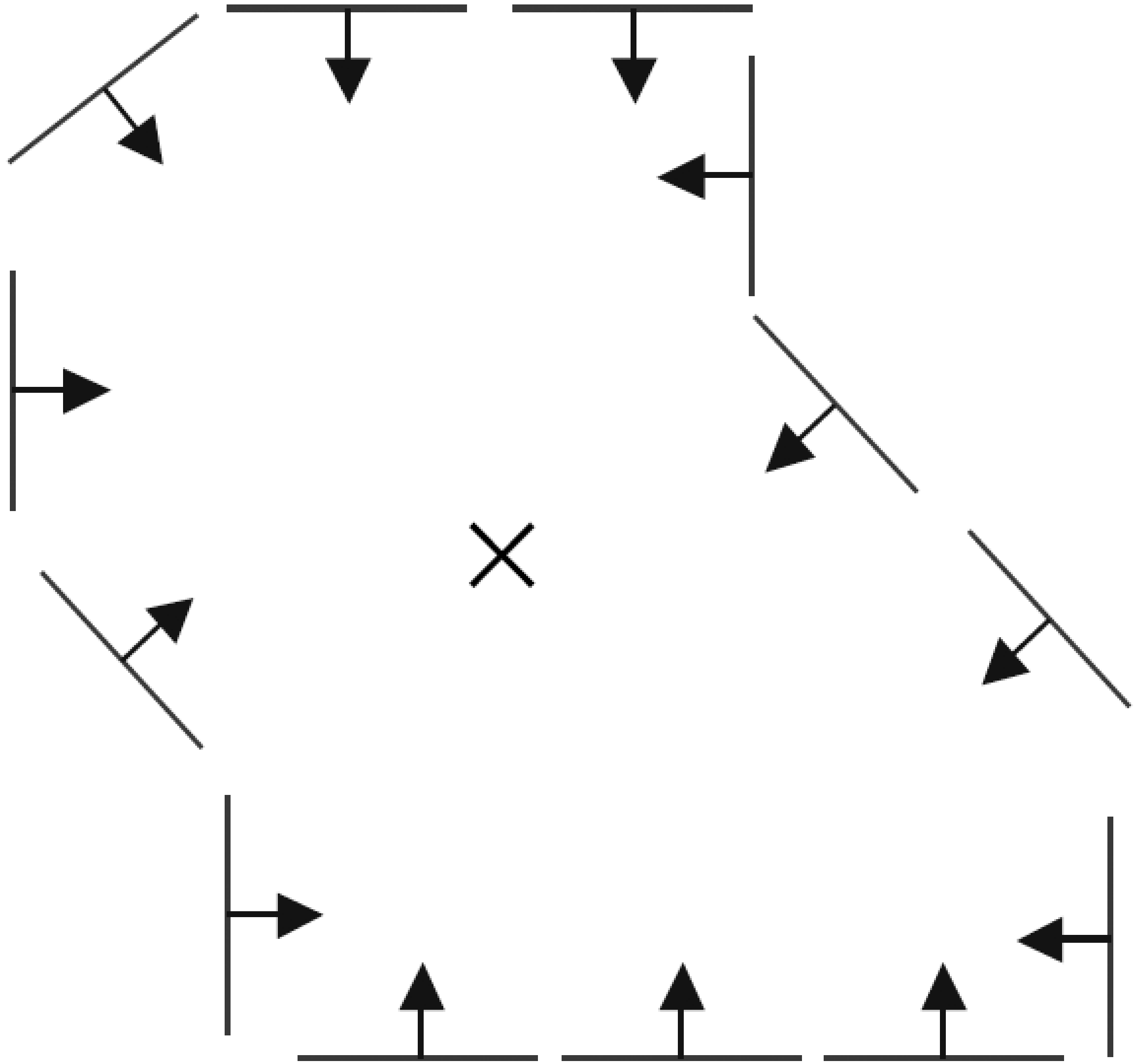

D. Ballard, [Generalizing the Hough Transform to Detect Arbitrary Shapes](), Pattern Recognition 13(2), 1981, pp. 111-122.

# Generalized Hough transform

- For model shape: construct a table indexed by $\vartheta$ storing displacement vectors r as function of gradient direction
- Detection: For each edge point *p* with gradient orientation $\vartheta$:
  - Retrieve all *r* indexed with $\vartheta$
  - For each $r(\vartheta)$, put a vote in the Hough space at $p + r(\vartheta)$
- Peak in this Hough space is reference point with most supporting edges
- Assumption: translation is the only transformation here, i.e., orientation and scale are fixed

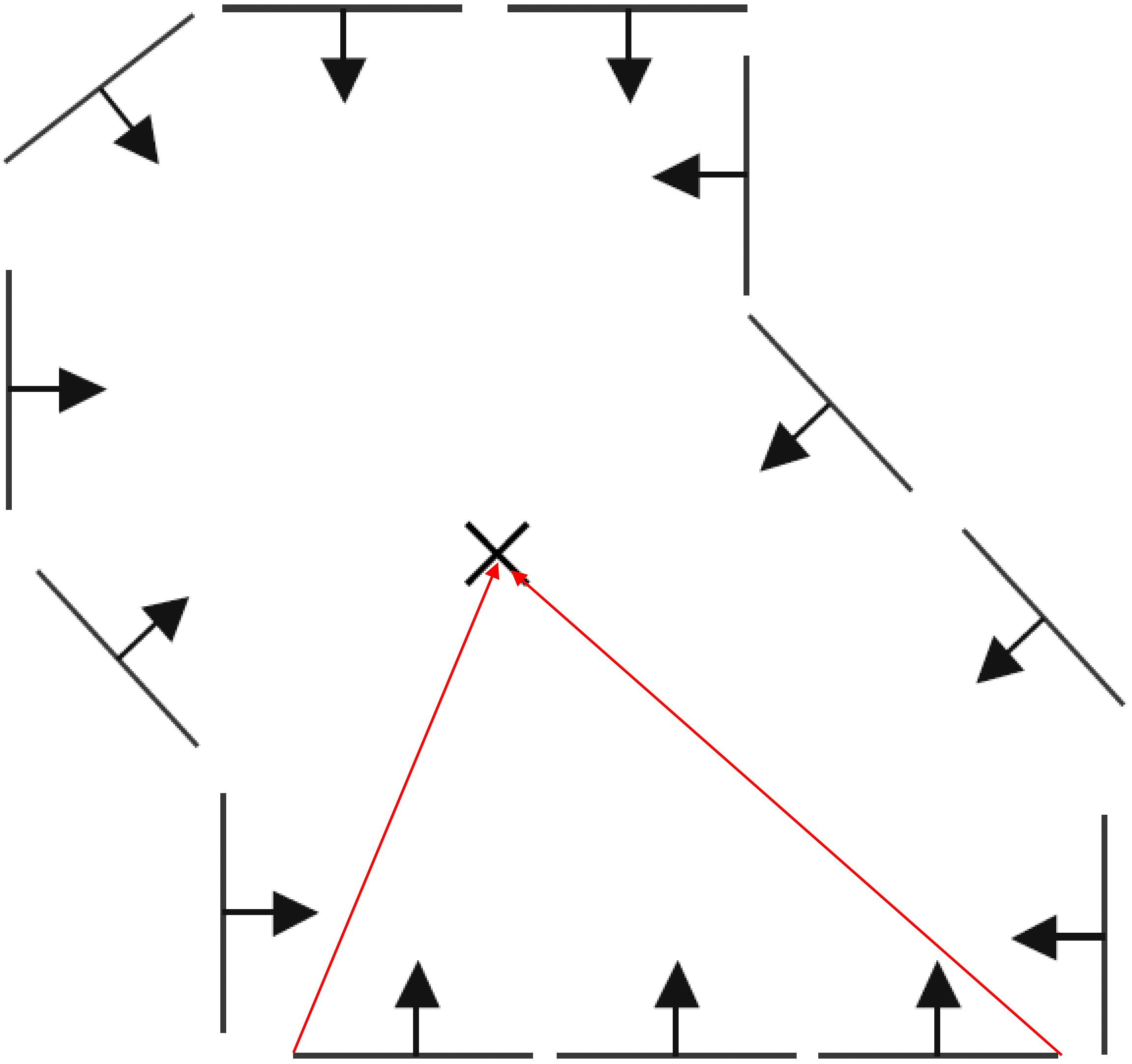
Source: K. Grauman

# Example



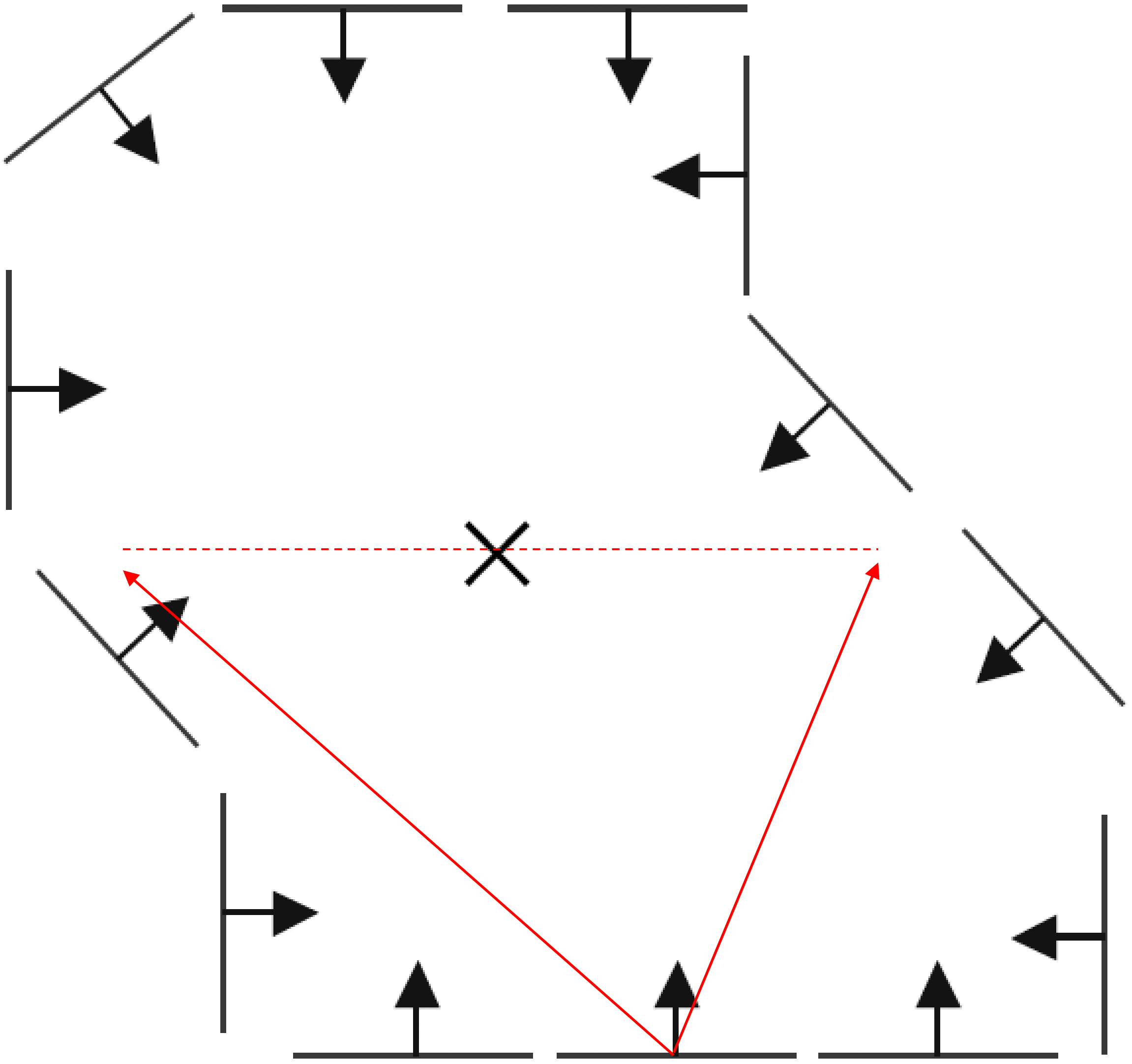
model shape

# Example



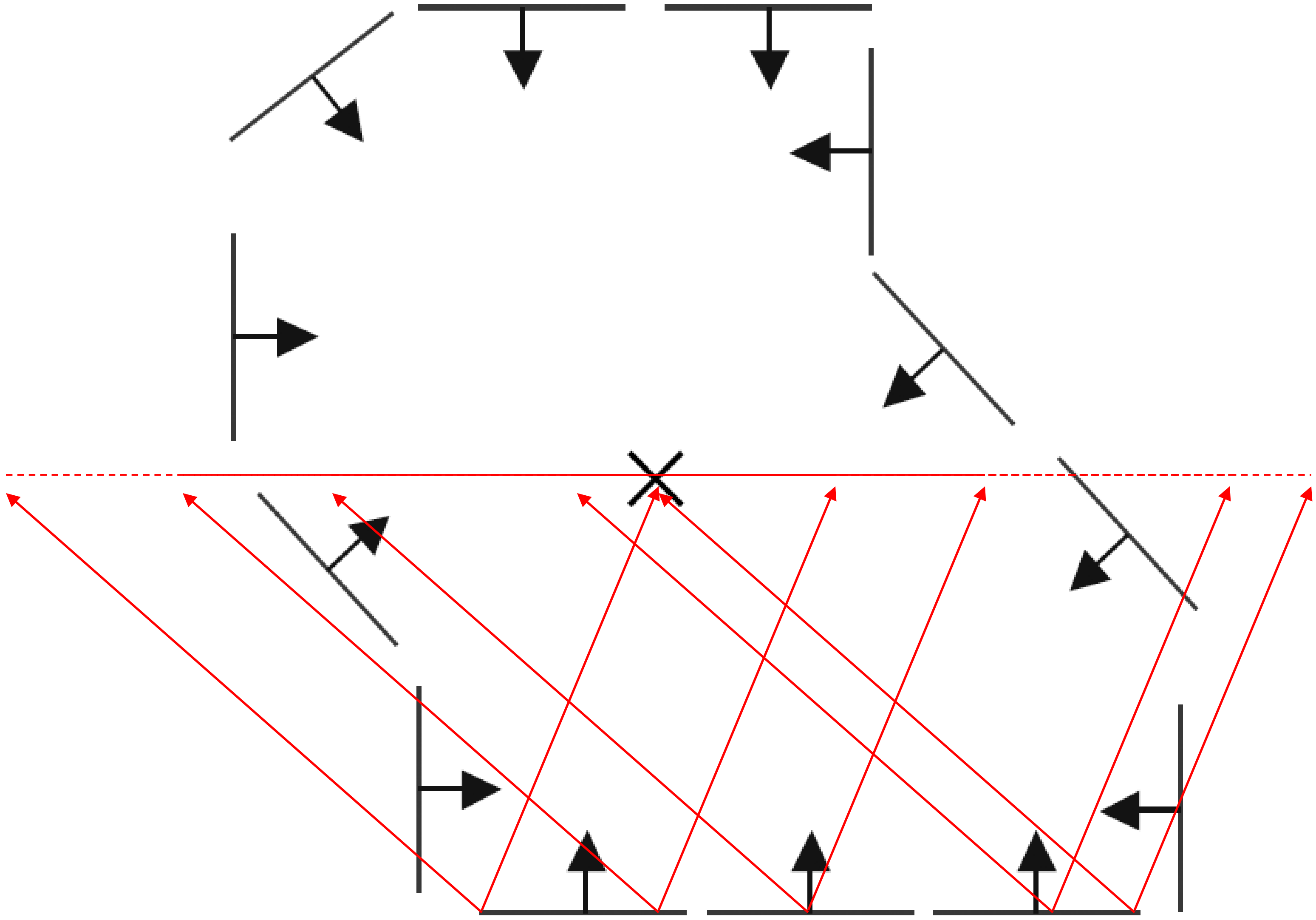
displacement vectors for model points

# Example



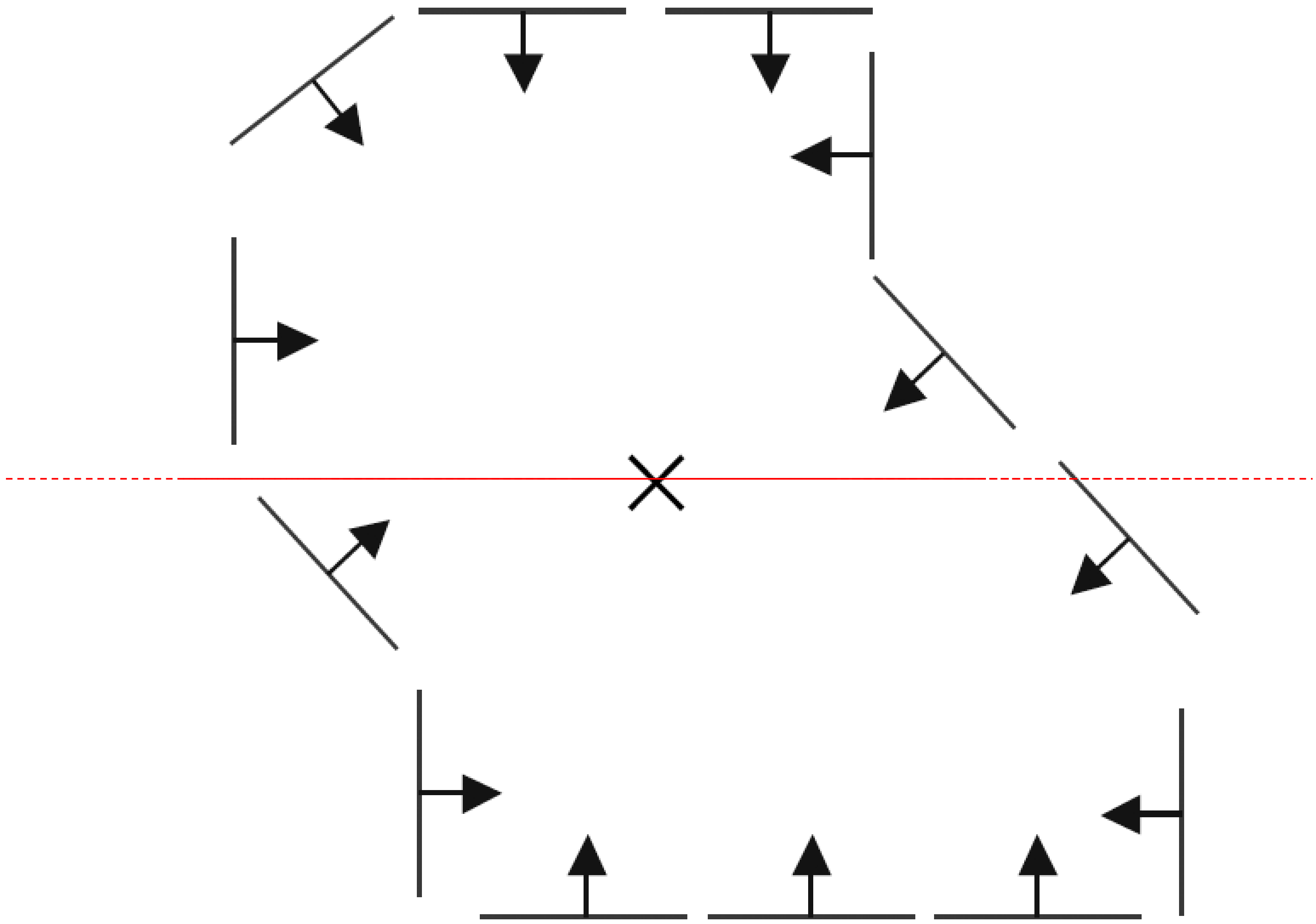
range of voting locations for test point

# Example



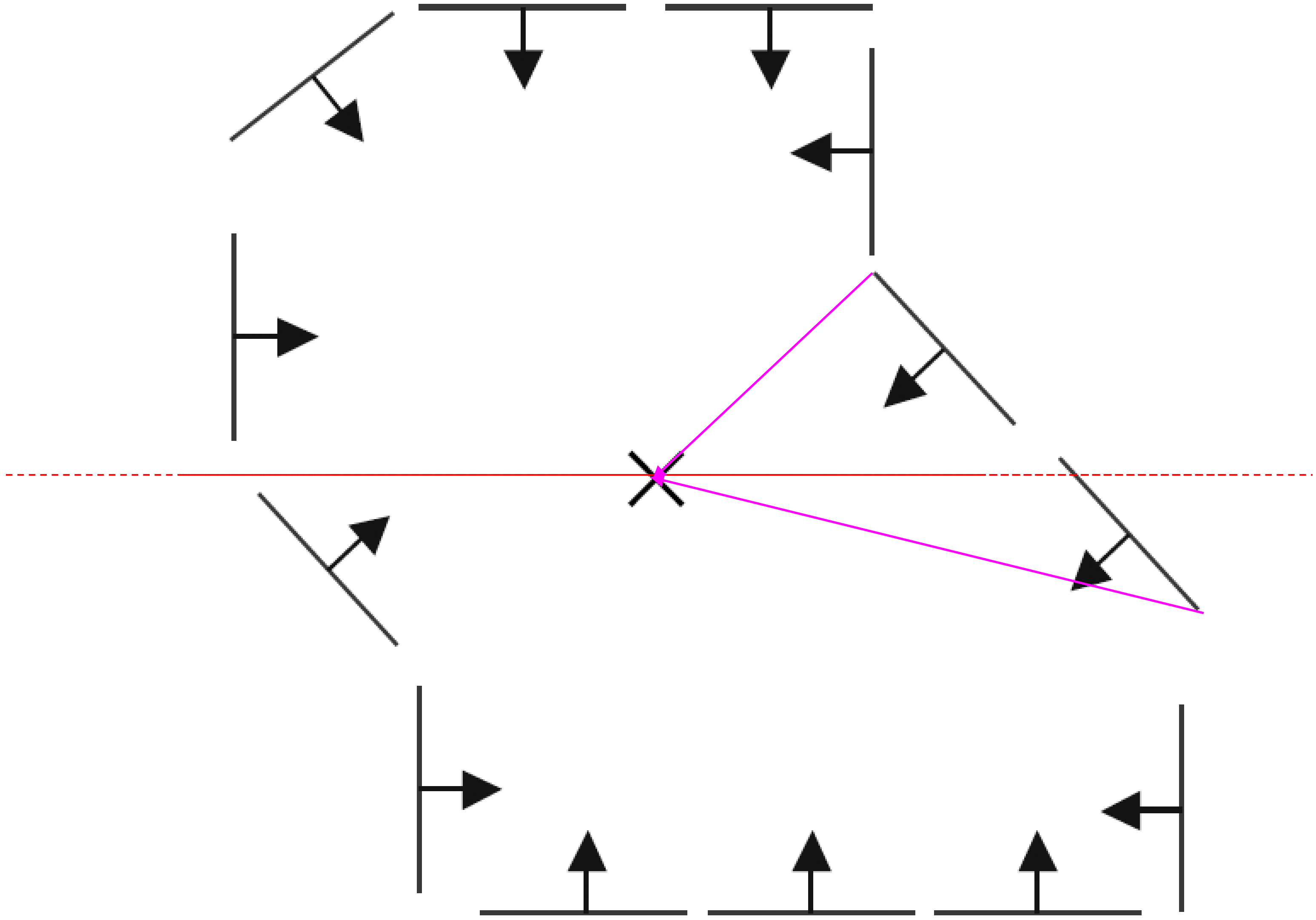
range of voting locations for test point

# Example



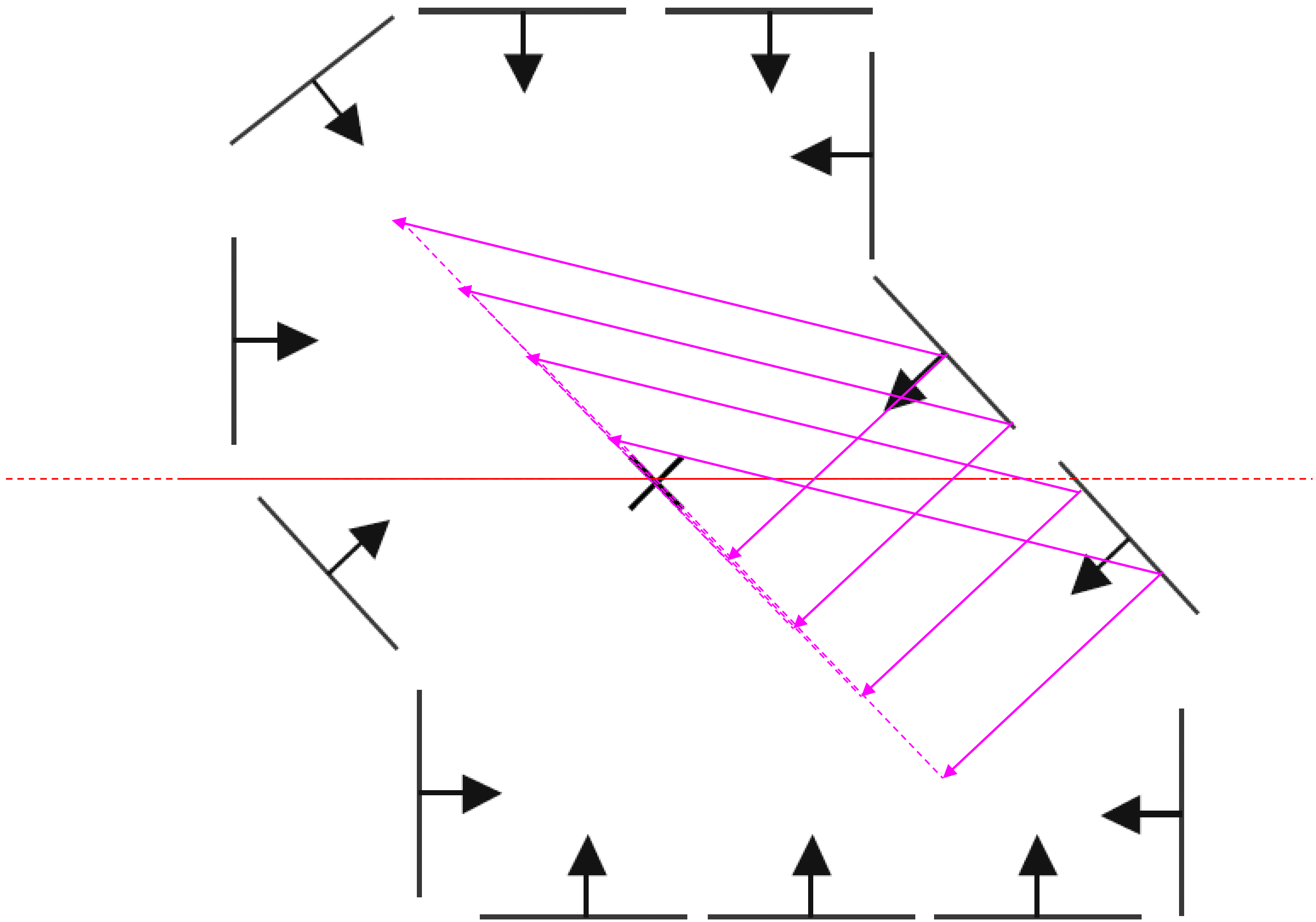
votes for points with $\theta = \uparrow$

# Example



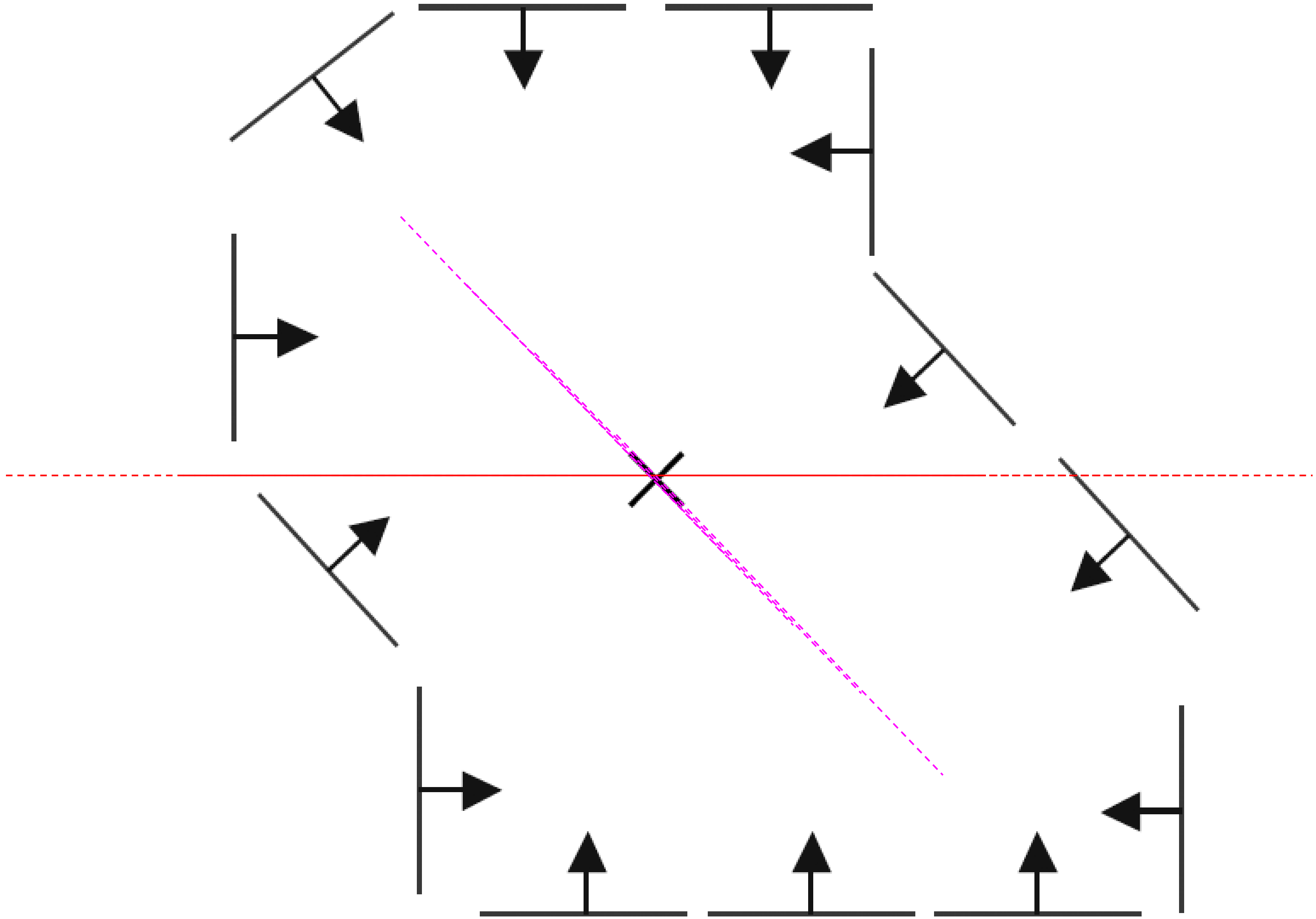
displacement vectors for model points

# Example



range of voting locations for test point

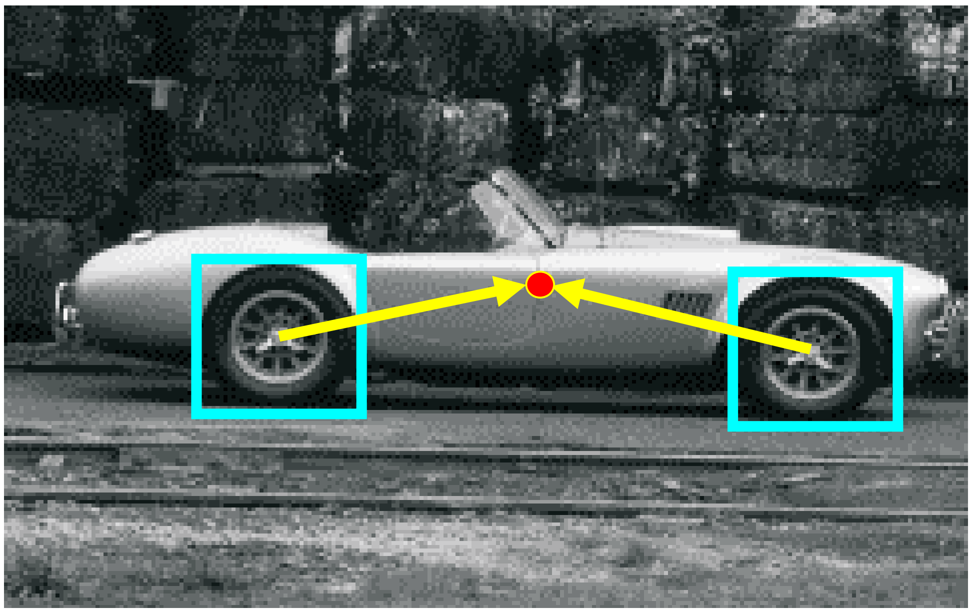
# Example



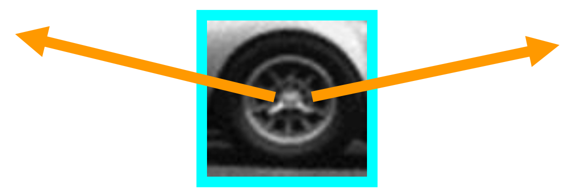
votes for points with $\theta =$ ↙

# 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

# Application in recognition

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

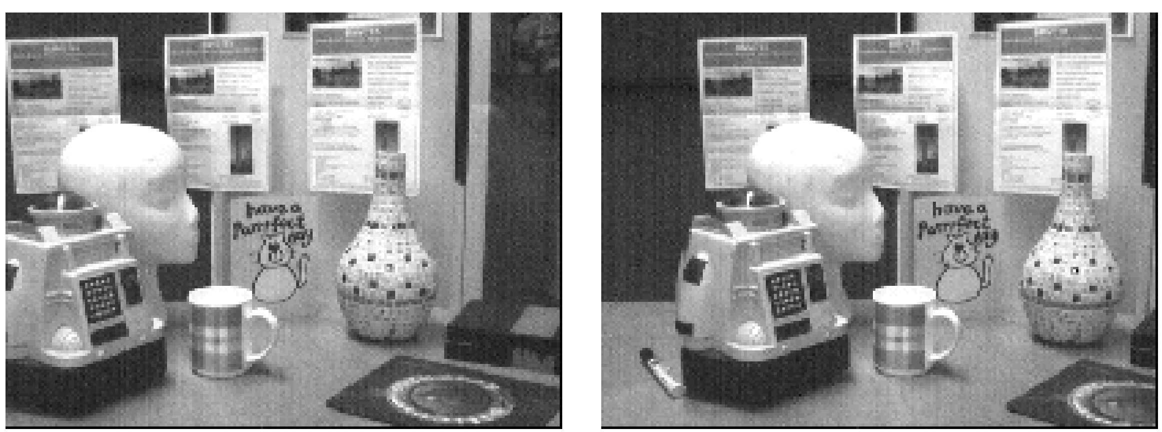
test image

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

# Homography

- The transformation between two views of a planar surface

The transformation between images from two cameras that share the same center



# Fitting a homography

- Recall: homogenenous coordinates

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Converting *to* homogenenous image coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

Converting *from* homogenenous image coordinates

# Fitting a homography

- Recall: homogenenous coordinates

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Converting *to* homogenenous image coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

Converting *from* homogenenous image coordinates

- Equation for homography:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# Fitting a homography

- Equation for homography:

$$\lambda \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\lambda \mathbf{x}_i' = \mathbf{H}\,\mathbf{x}_i = \begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \mathbf{h}_3^T \end{bmatrix} \mathbf{x}_i$$

9 entries, 8 degrees of freedom
(scale is arbitrary)

$$\mathbf{x}_i' \times \mathbf{H}\,\mathbf{x}_i = 0$$

$$\mathbf{x}_i' \times \mathbf{H}\,\mathbf{x}_i = \begin{bmatrix} y_i'\,\mathbf{h}_3^T \mathbf{x}_i - \mathbf{h}_2^T \mathbf{x}_i \\ \mathbf{h}_1^T \mathbf{x}_i - x_i'\,\mathbf{h}_3^T \mathbf{x}_i \\ x_i'\,\mathbf{h}_2^T \mathbf{x}_i - y_i'\,\mathbf{h}_1^T \mathbf{x}_i \end{bmatrix}$$

$$\begin{bmatrix} 0^T & -\mathbf{x}_i^T & y_i'\,\mathbf{x}_i^T \\ \mathbf{x}_i^T & 0^T & -x_i'\,\mathbf{x}_i^T \\ -y_i'\,\mathbf{x}_i^T & x_i'\,\mathbf{x}_i^T & 0^T \end{bmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = 0$$

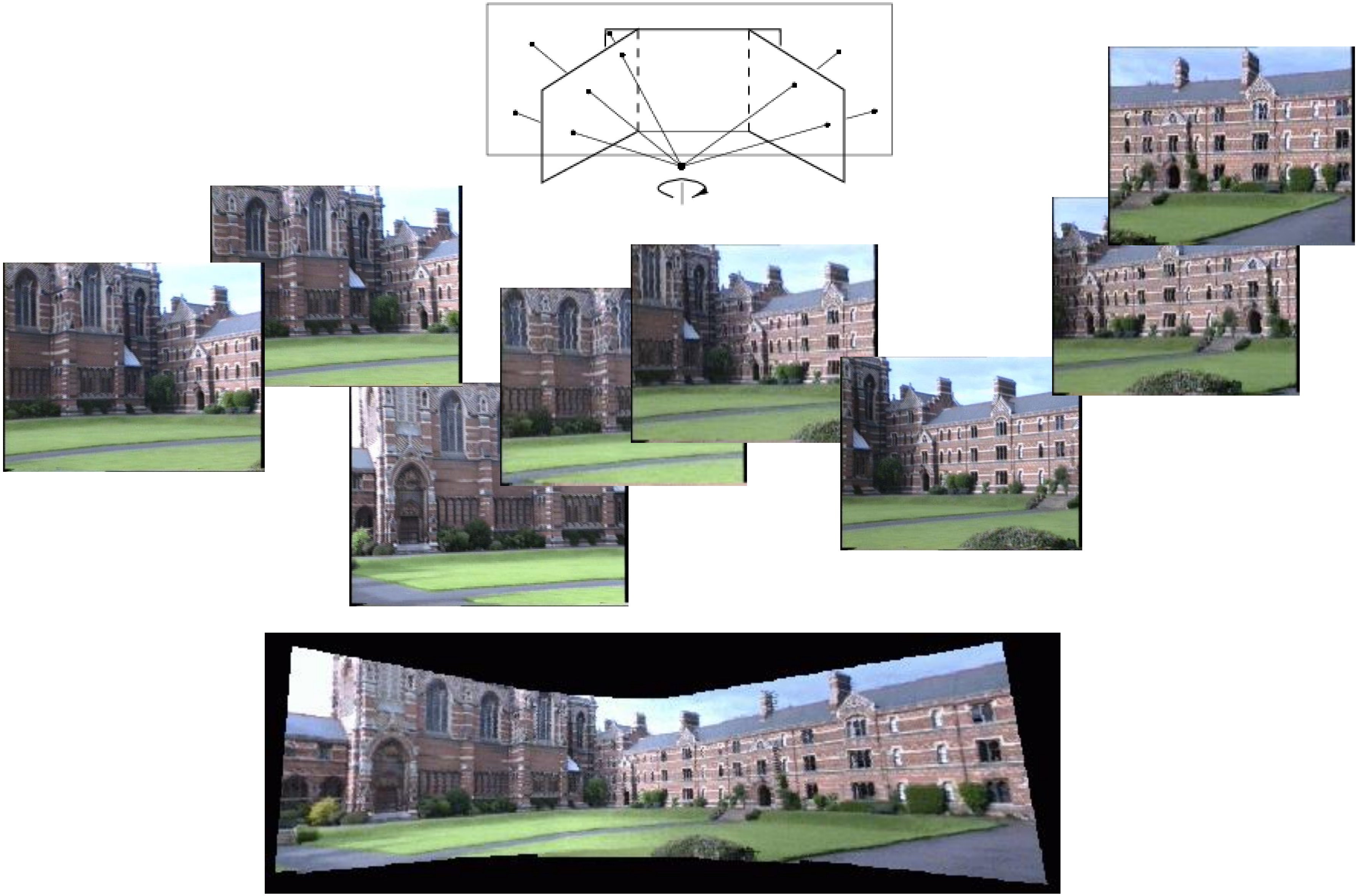
3 equations, only 2 linearly
independent

# Direct linear transform

$$\begin{bmatrix} 0^T & \mathbf{x}_1^T & -y_1' \mathbf{x}_1^T \\ \mathbf{x}_1^T & 0^T & -x_1' \mathbf{x}_1^T \\ \cdots & \cdots & \cdots \\ 0^T & \mathbf{x}_n^T & -y_n' \mathbf{x}_n^T \\ \mathbf{x}_n^T & 0^T & -x_n' \mathbf{x}_n^T \end{bmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = 0 \qquad \mathbf{A}\mathbf{h} = 0$$

- H has 8 degrees of freedom (9 parameters, but scale is arbitrary)
- One match gives us two linearly independent equations
- Four matches needed for a minimal solution (null space of 8x9 matrix)
- More than four: homogeneous least squares

# Application: Panorama stitching



# Recognizing panoramas

- Given contents of a camera memory card, automatically figure out which pictures go together and stitch them together into panoramas

M. Brown and D. Lowe, *"Recognizing Panoramas,"* ICCV 2003.
http://www.cs.ubc.ca/~mbrown/panorama/panorama.html