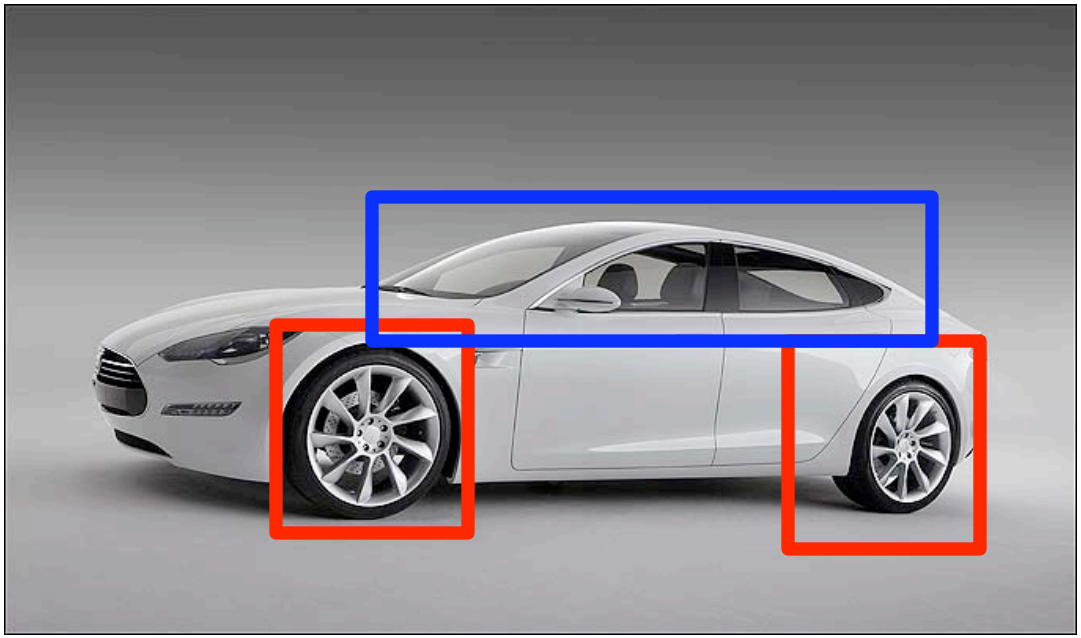
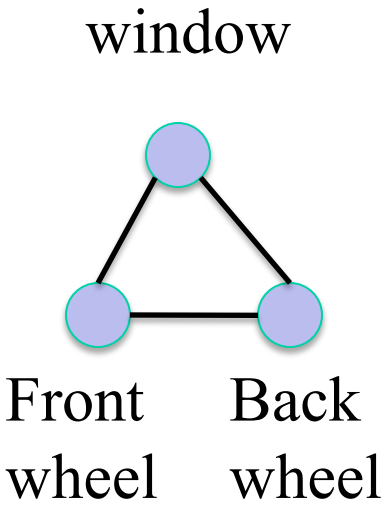
Model Fitting

Hao Jiang
Computer Science Department
Oct 4, 2011

Model Fitting

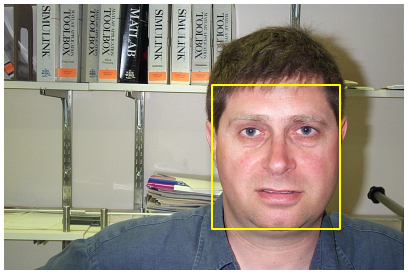
- An object model constrains the kind of object we wish to find in images.



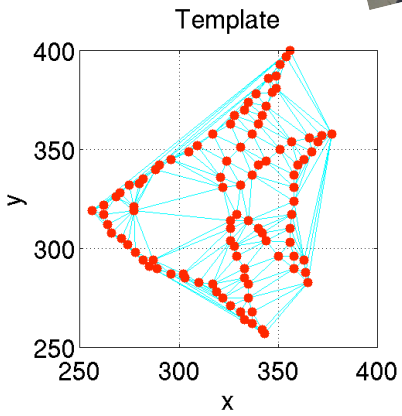


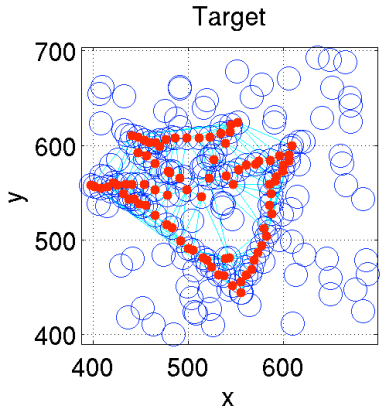
Type of 2D Models

- Rigid models

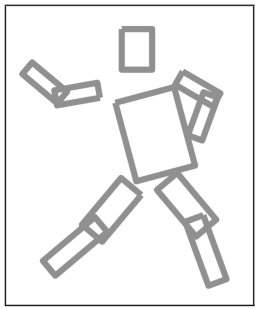


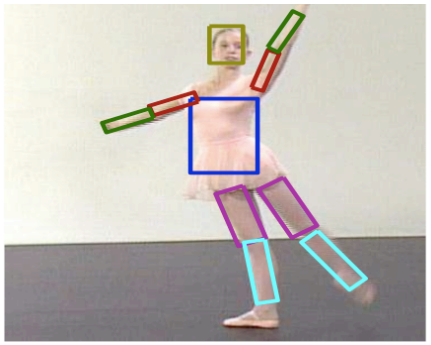
- Deformable models

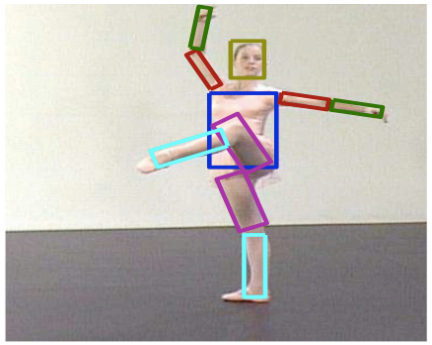


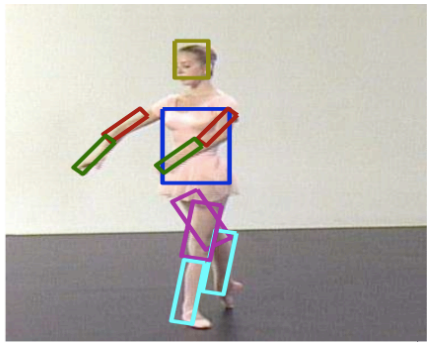


- Articulated models









Template Matching

Template

Target image

Template Matching

Template

Target image

Template Matching

Template

Target image

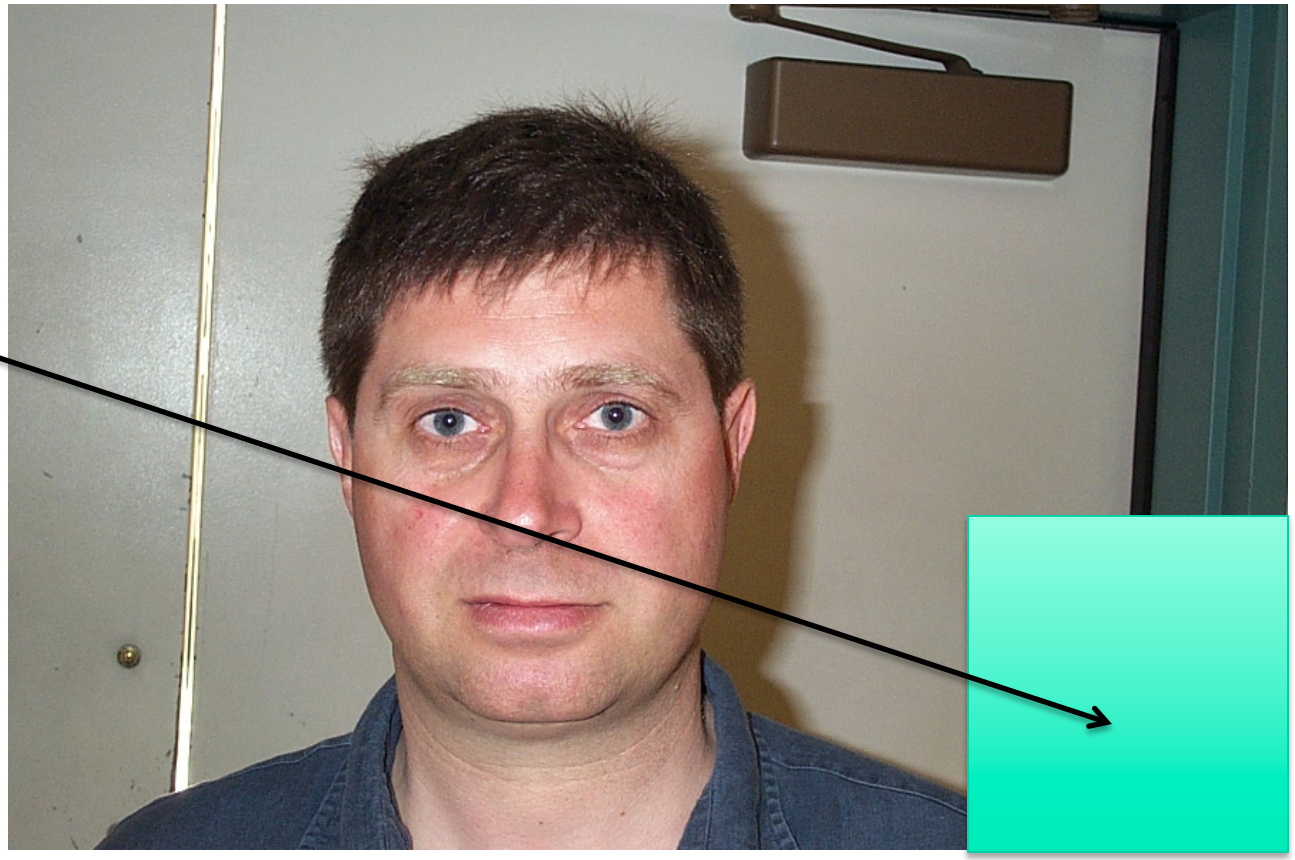
Template Matching

Template

Target image

Template Matching

Template



Target image

Matching Criteria

At each location (x,y) the matching cost is:

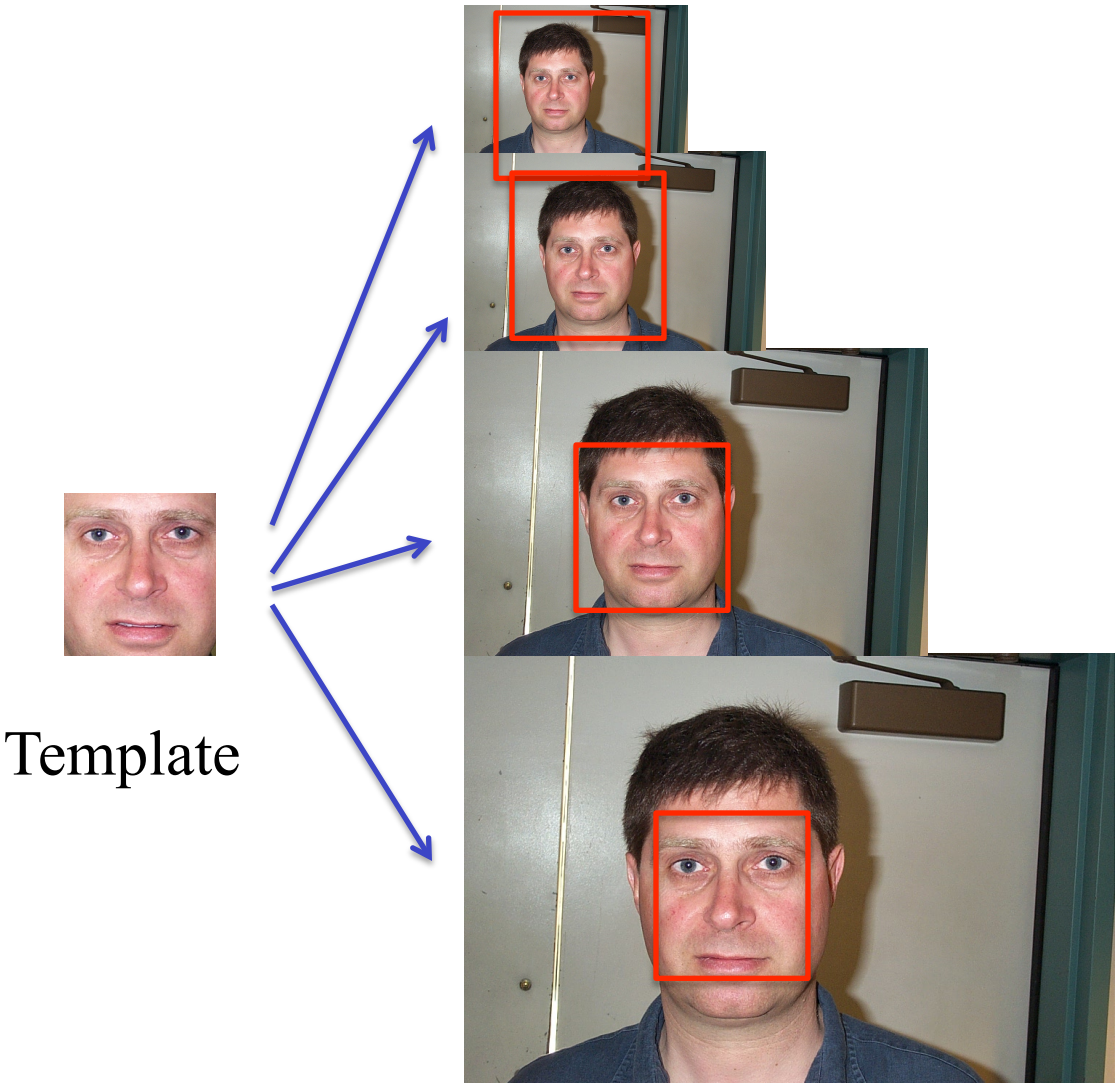
$$\sum_{(\delta_x,\delta_y)\in W} d(T(\delta_x,\delta_y), I(x+\delta_x, y+\delta_y))$$

Dealing with Rotations

Template

Target image

Dealing with Scale Changes

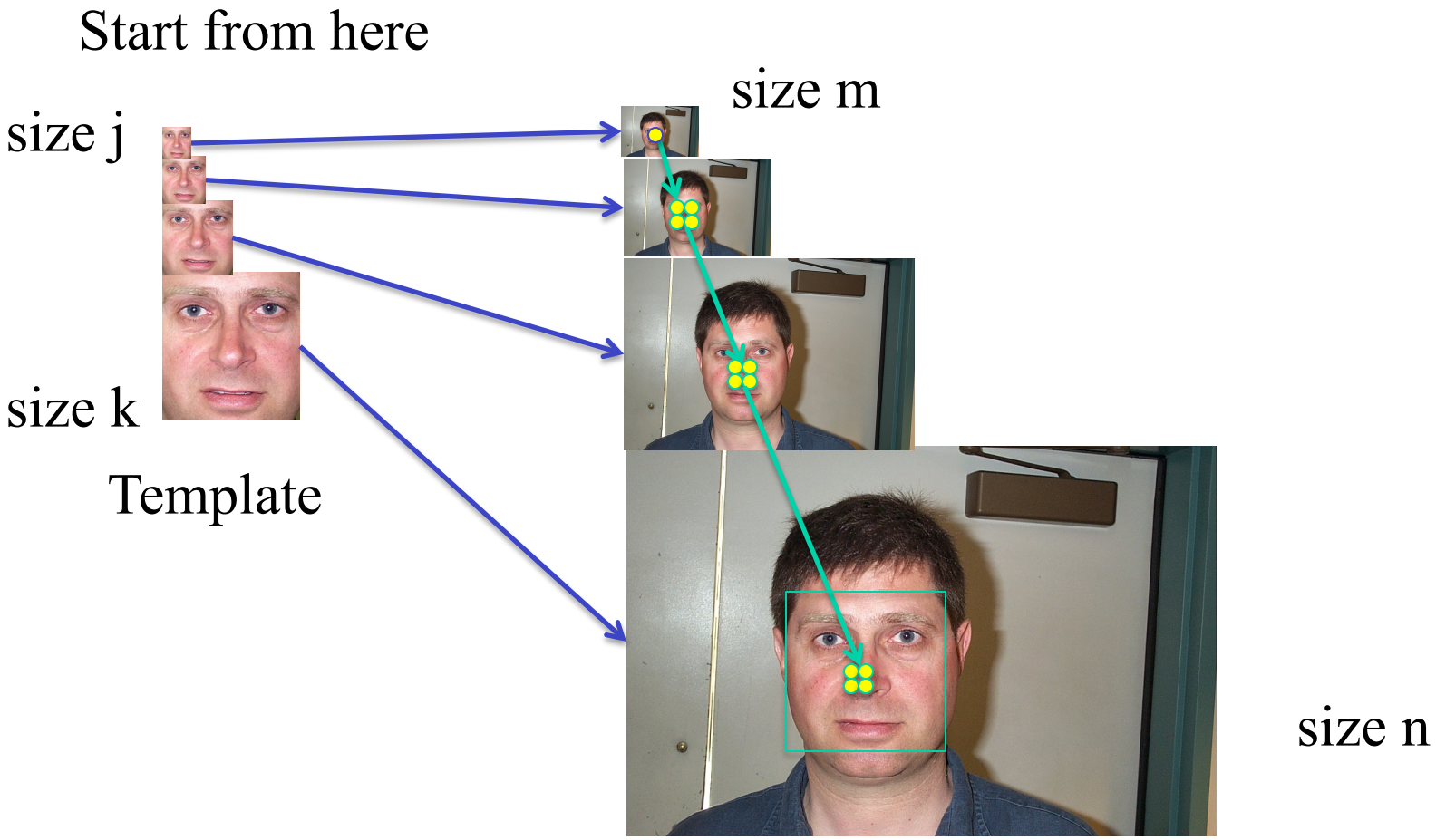


Template

Sweep window and find the best match

Target image in multiple scales

Multi-scale Refinement



Complexity reduced from O(n^2 k^2) to O(log(n/m) + m^2 j^2)

Multi-scale Refinement

1. Generate the image pyramids

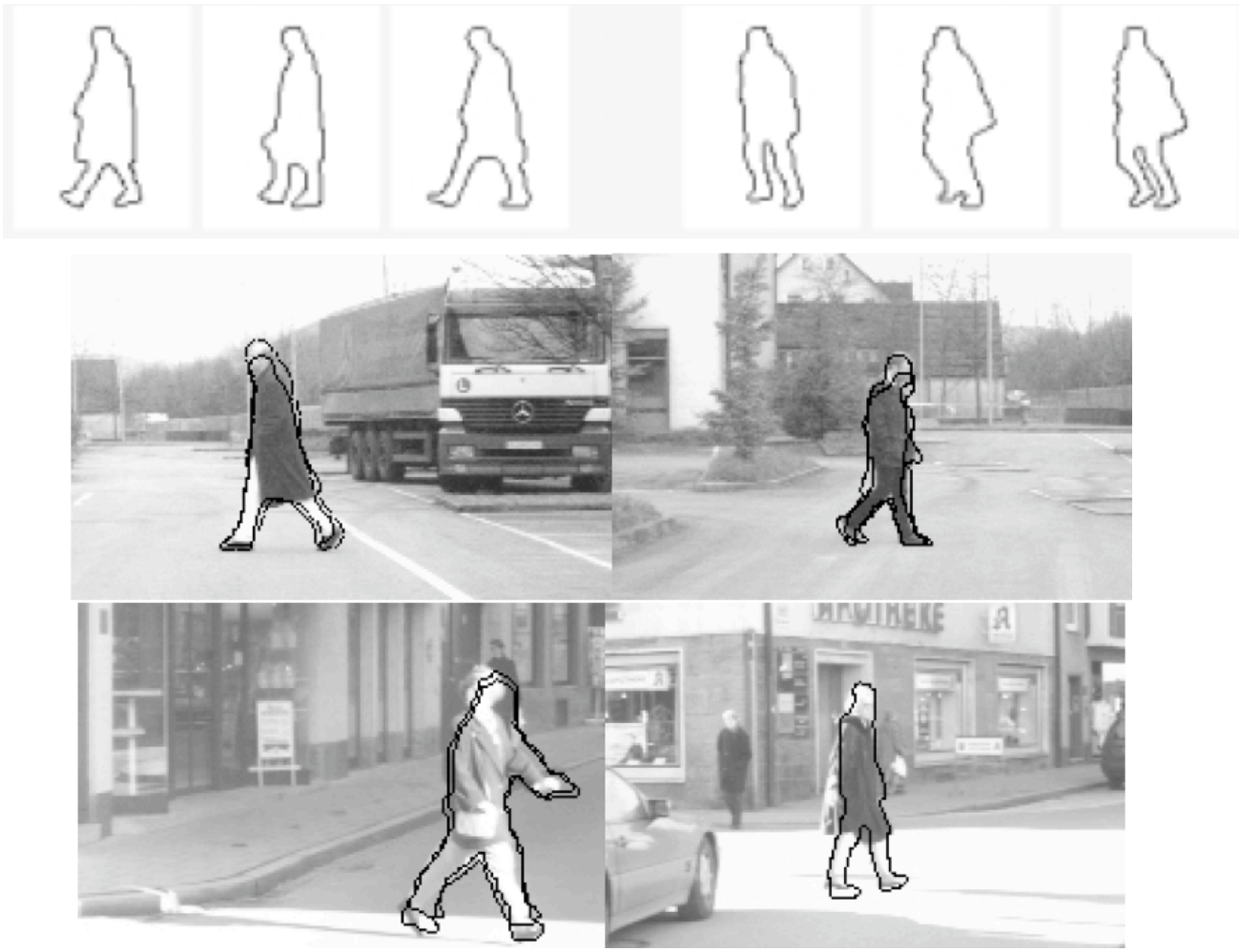
2. Exhaustive search the smallest image and get location (x,y)

3. Go to the next level of the pyramid and check the locations (x, y), (x+1,y), (x,y+1), (x+1,y+1), pick up the best and update the estimate (x,y)

4. If at the bottom of the pyramid, stop; otherwise go to 3.

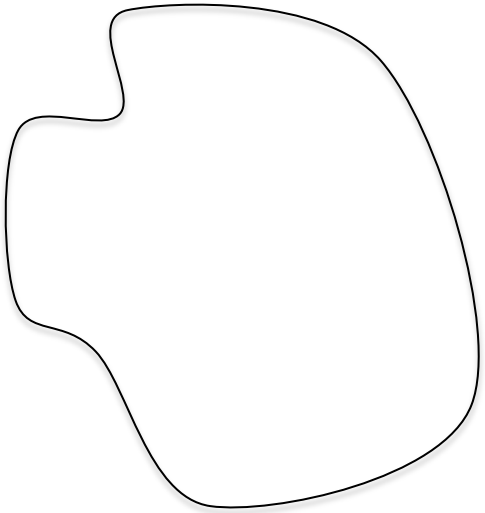
5. Output the estimate (x,y)

Binary Template Matching

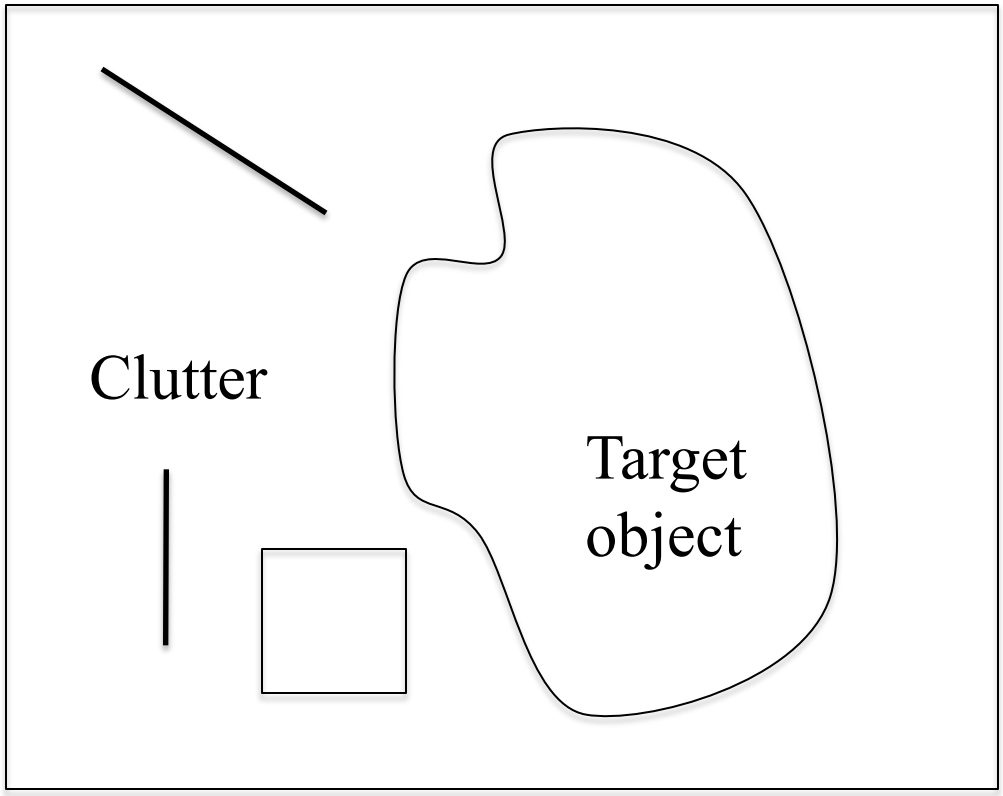


G.M. Gavrila, “Pedestrian detection from a moving vechicle”, ECCV 2000

Matching Binary Template

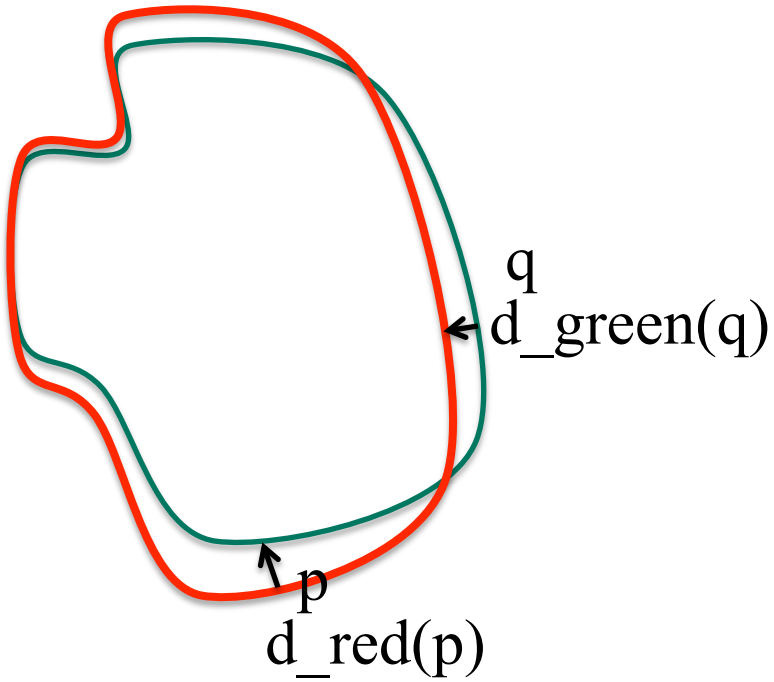


Template



Binary target image

Distance of Curves



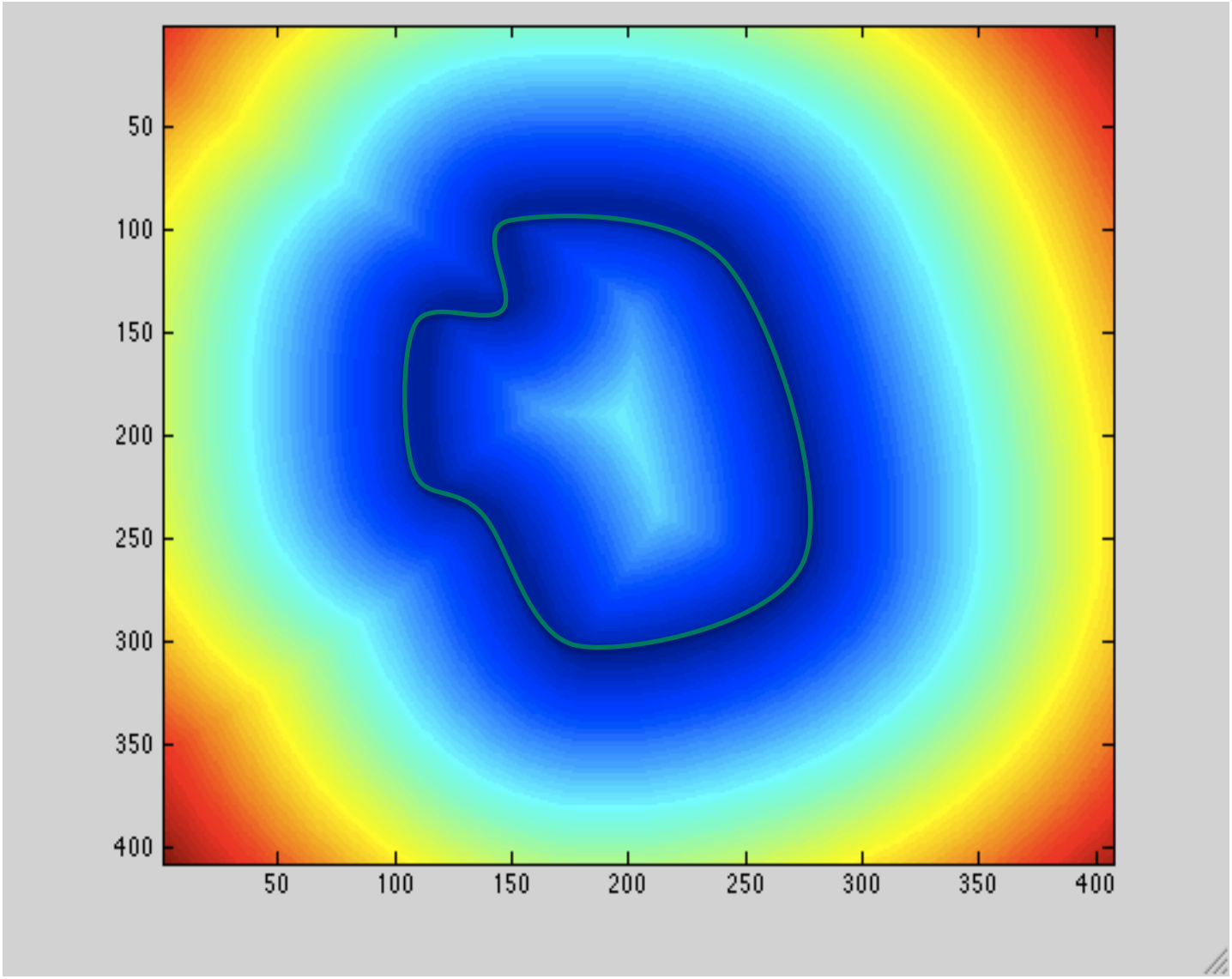
For each point p on the red curve,
find the closed point q on the green curve.
Denote $d(p,q) = d_{red}(p)$

The distance from red curve to the green curve = $mean_{p \in \text{red}} d_{red}(p)$

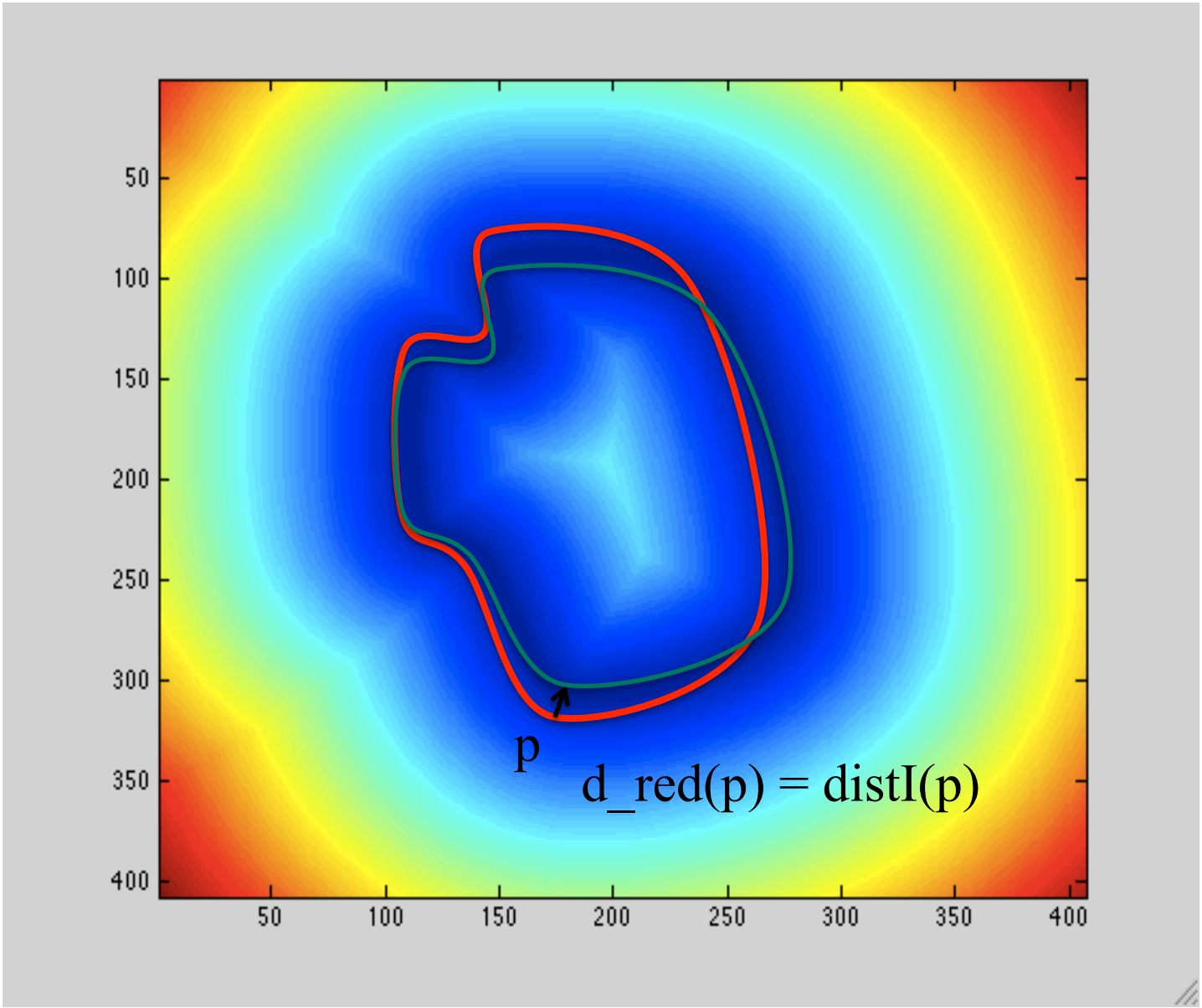
Similarly, the distance from the green curve to the red curve is
$mean_{q \in \text{green}} d_{green}(q)$

Distance Transform

- Transform binary image into an image whose pixel values equal the closest distance to the binary foreground.



Chamfer Matching



Chamfer Matching in Matlab

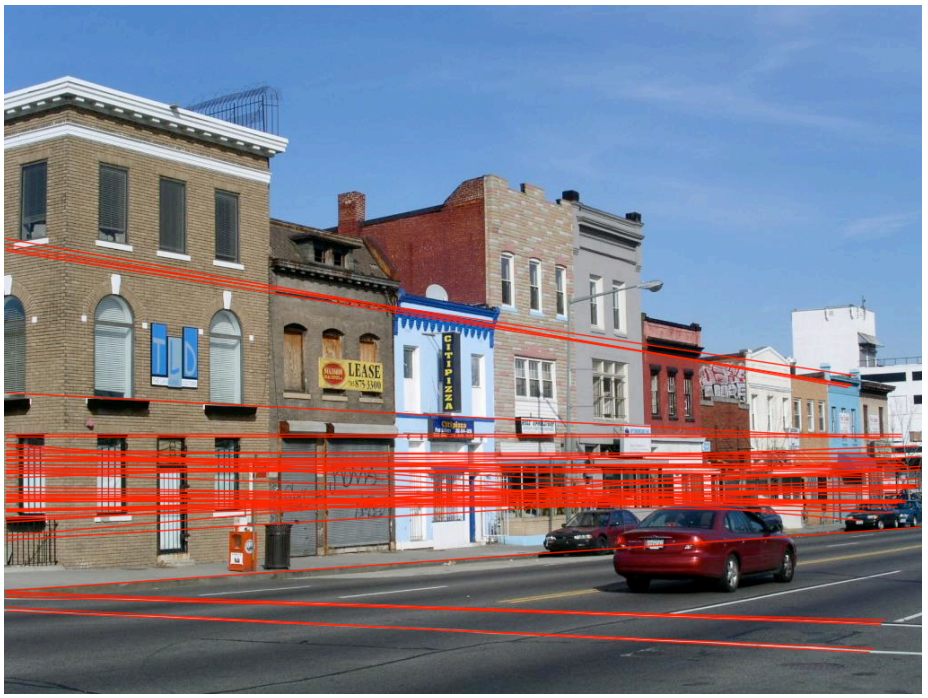
```
% distance transform and matching
dist = bwdist(imTarget, 'euclidean');
map = imfilter(dist, imTemplate, 'corr', 'same');

%look for the local minima in map to locate objects
smap = ordfilt2(map, 25, true(5));
[r, c] = find((map == smap) & (map < threshold));
```

Hough Transform

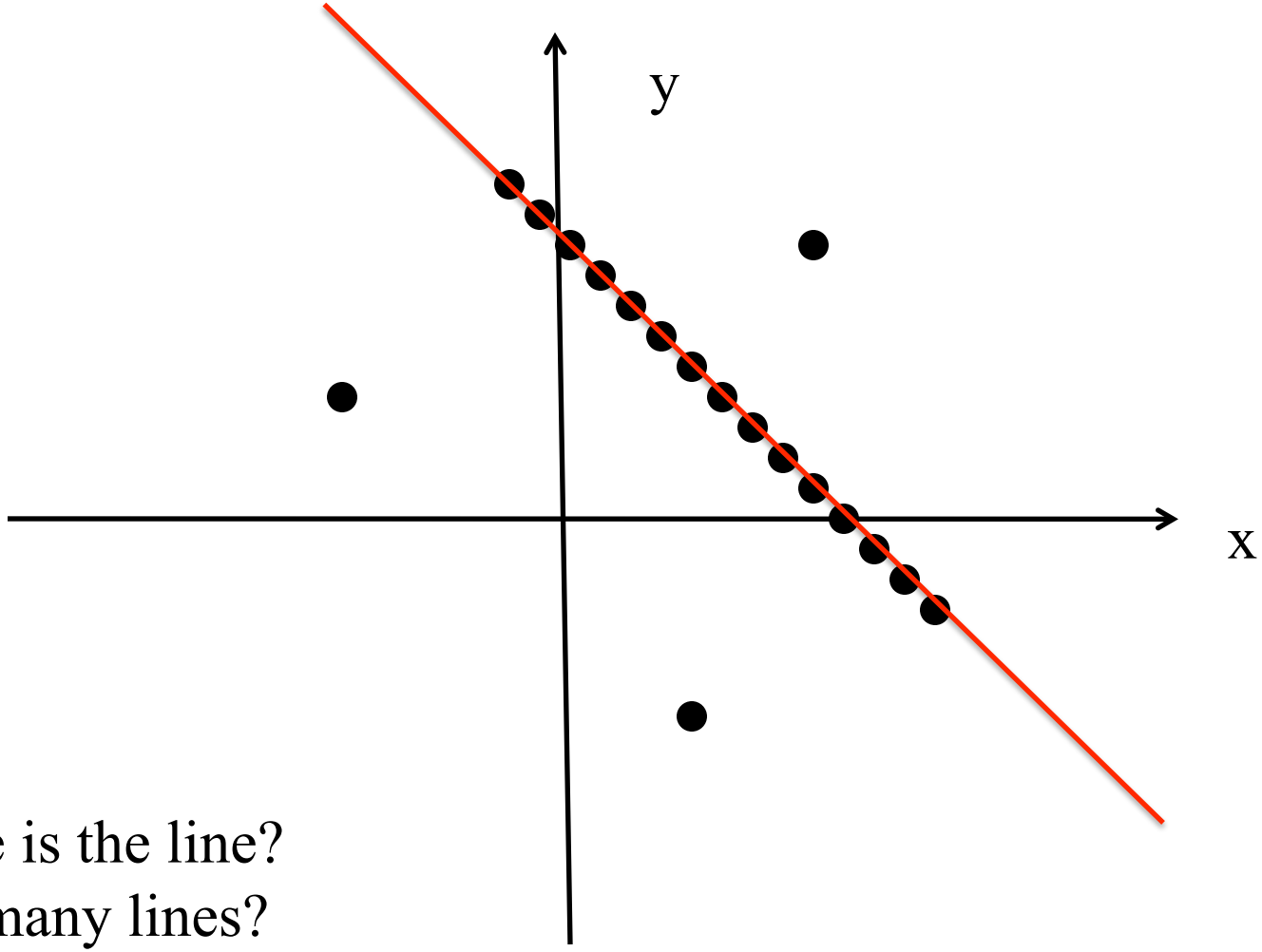
- Template matching may become very complex if transformations such as rotation and scale is allowed.
- A voting based method, Hough Transform, can be used to solve the problem.
- Instead of trying to match the target at each location, we collect votes in a parameter space using features in the original image.

Line Detection



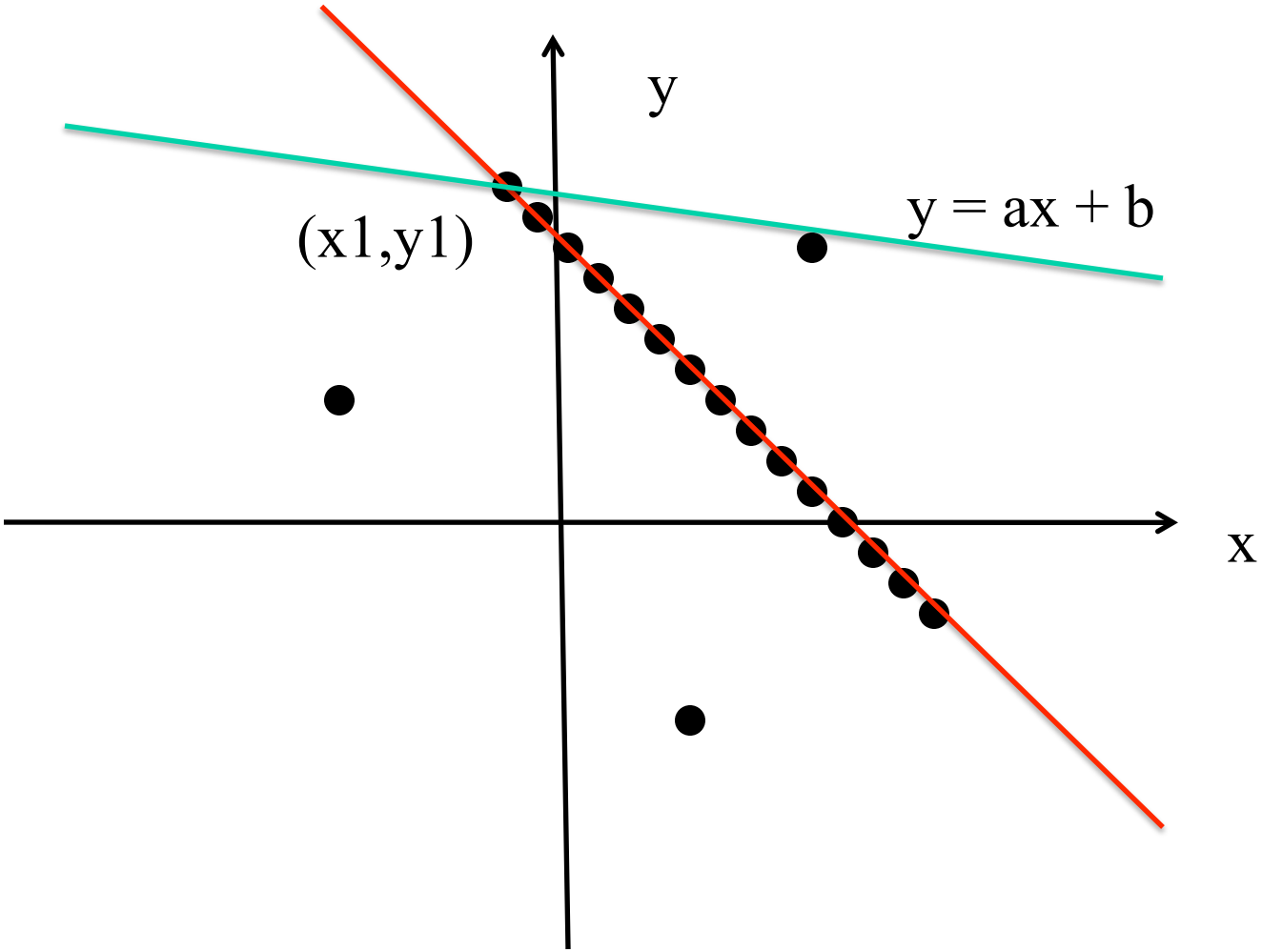
Line Detection

Line Detection



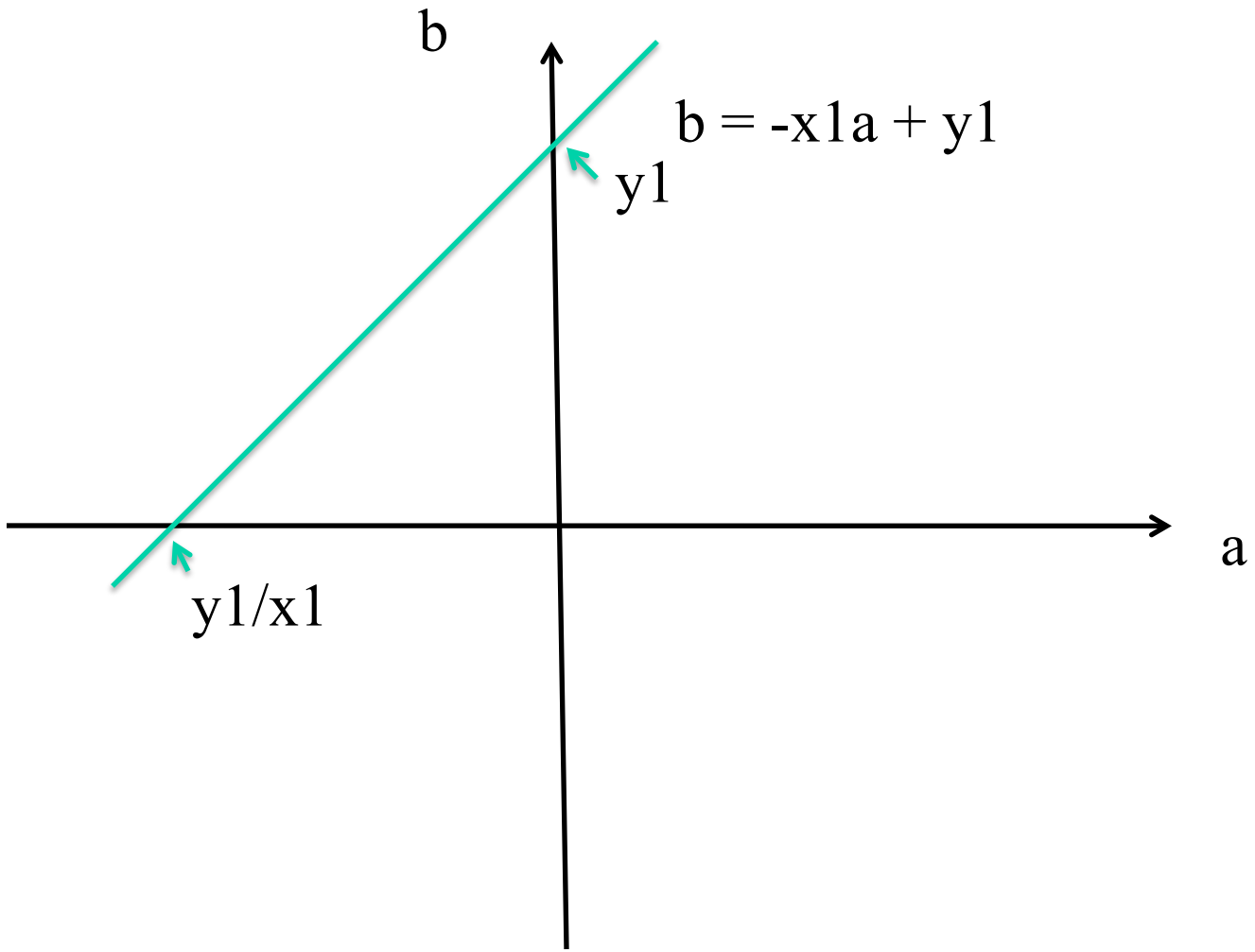
(1) Where is the line?
(2) How many lines?
(3) Which point belongs to which line?

Line Detection



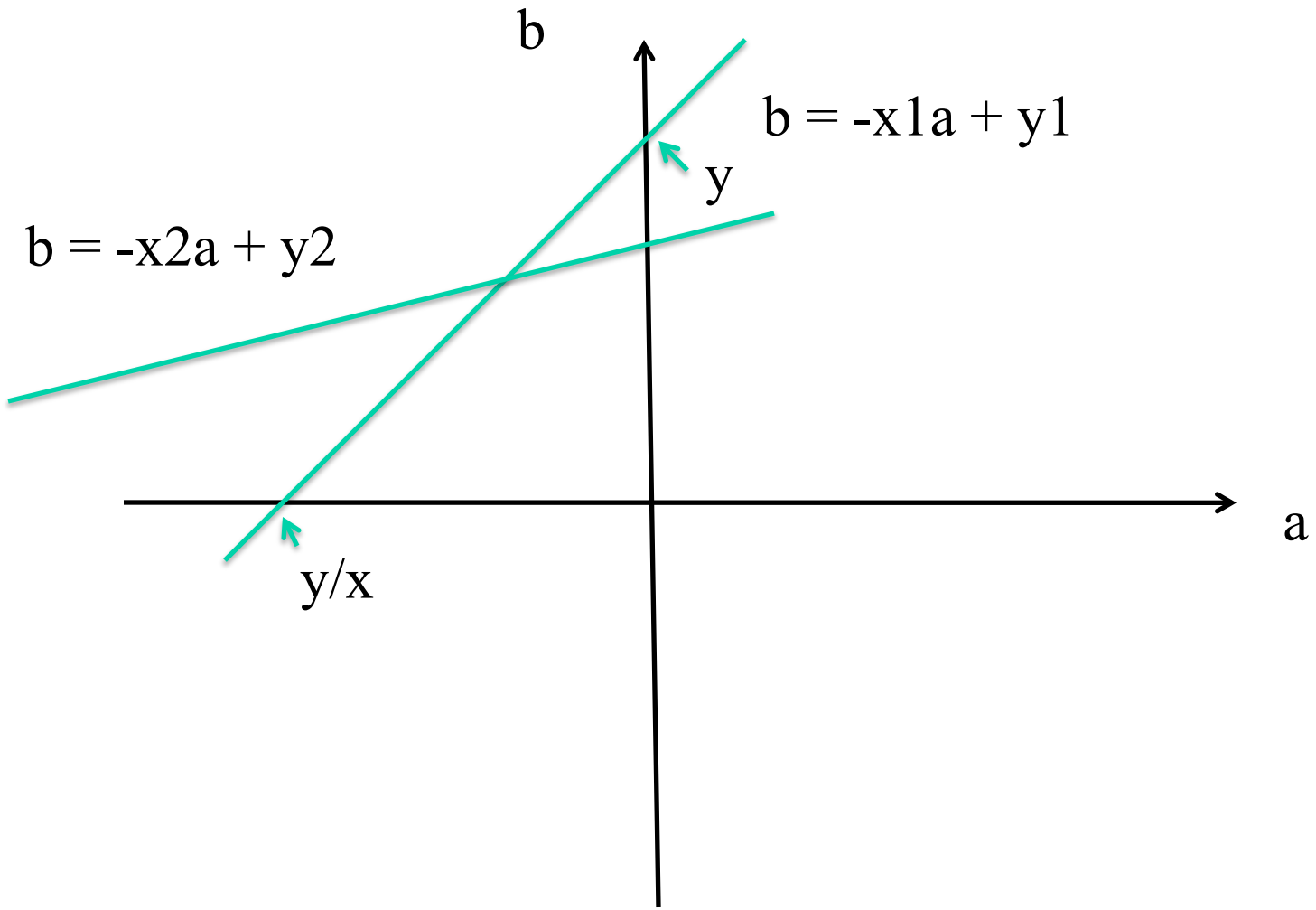
For each point, there are many lines that pass it.

Line Detection



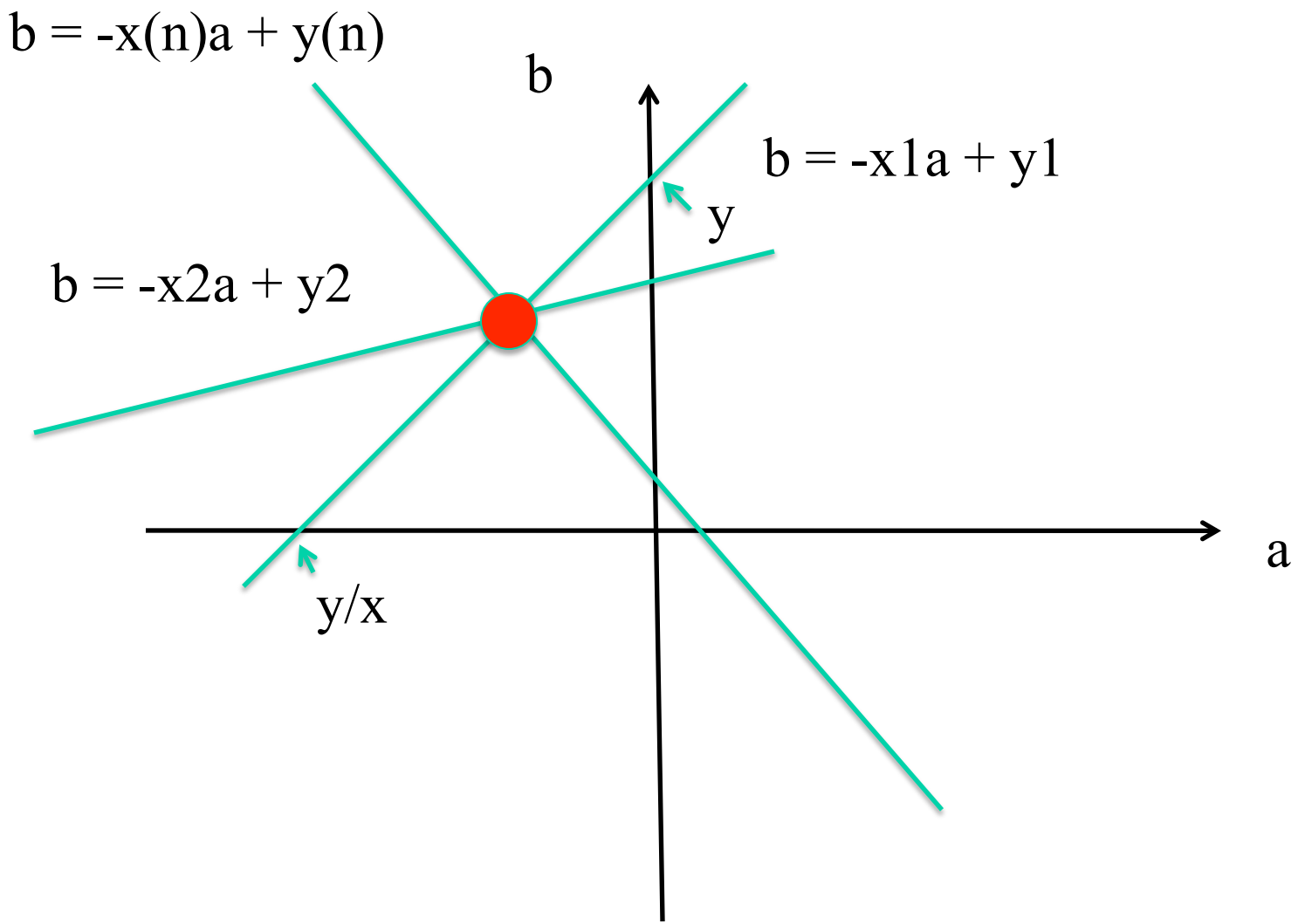
All the lines passing (x1,y1) can be represented as another line in the *parameter* space.

Line Detection



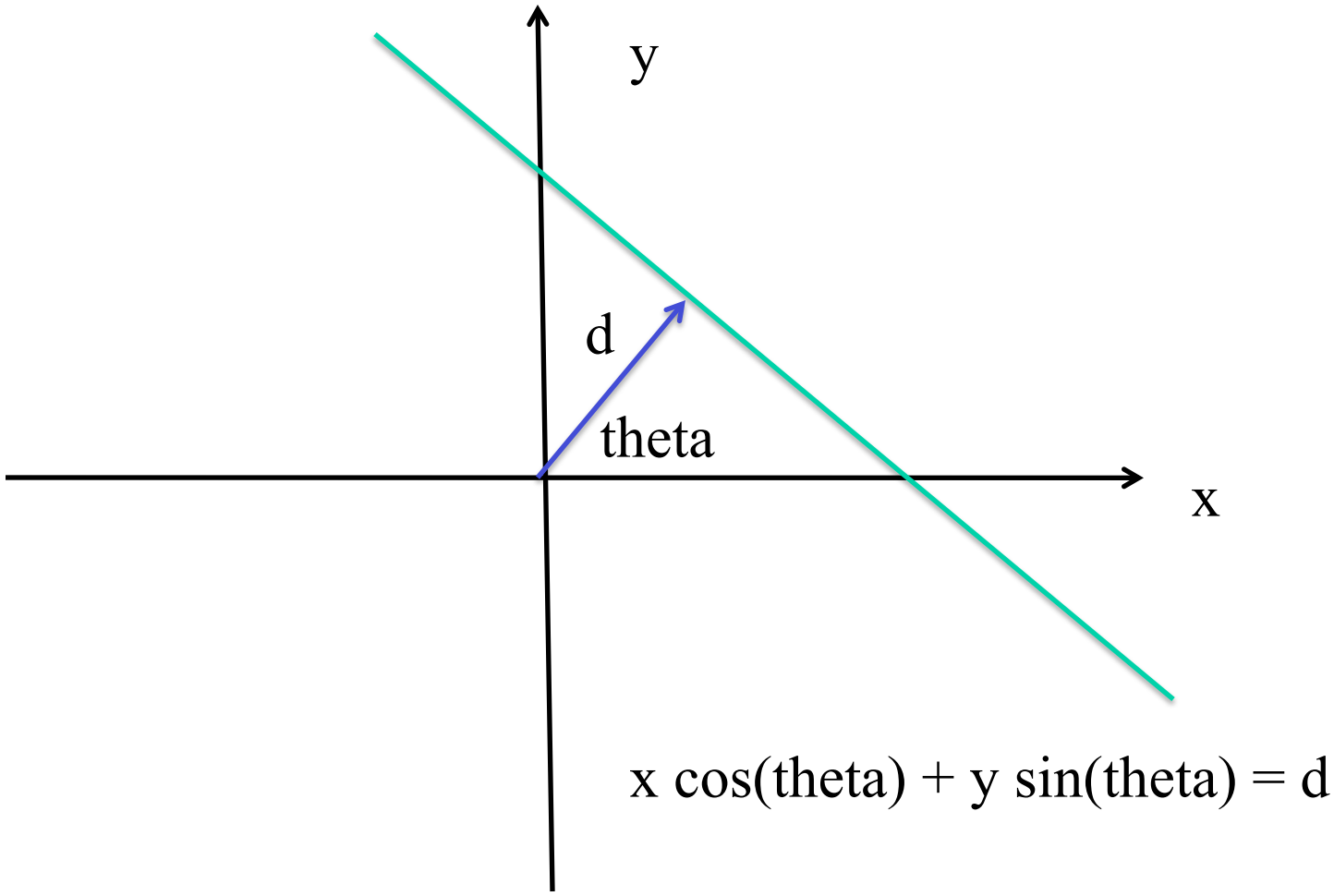
The co-linear points vote for the same point in the a-b space.

Line Detection



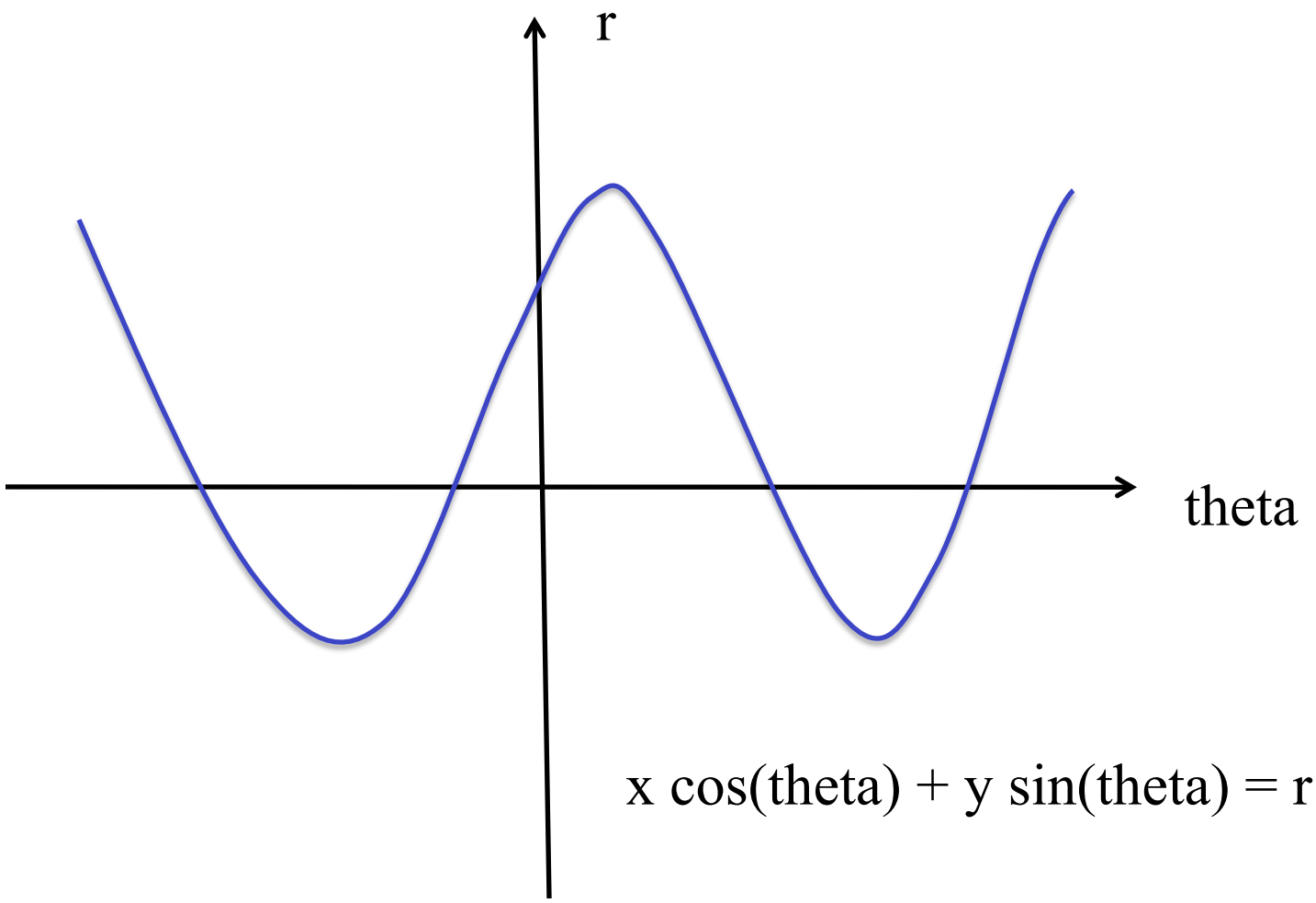
The co-linear points vote for the same point in the a-b space. The point that receives the most votes corresponds to the line.

Line Parameterization



$$x \cos(\text{theta}) + y \sin(\text{theta}) = d$$

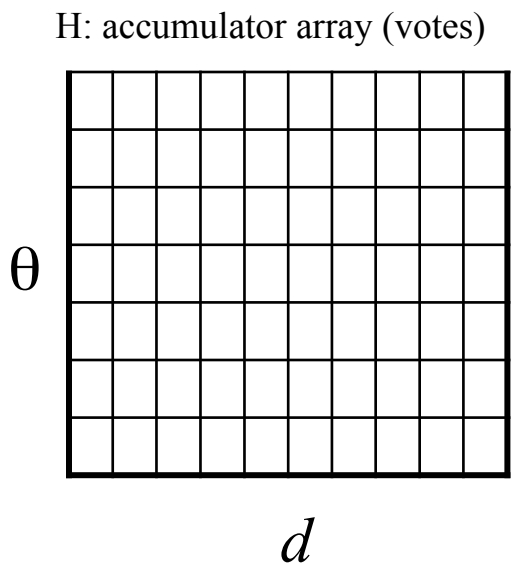
Parameter Space



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

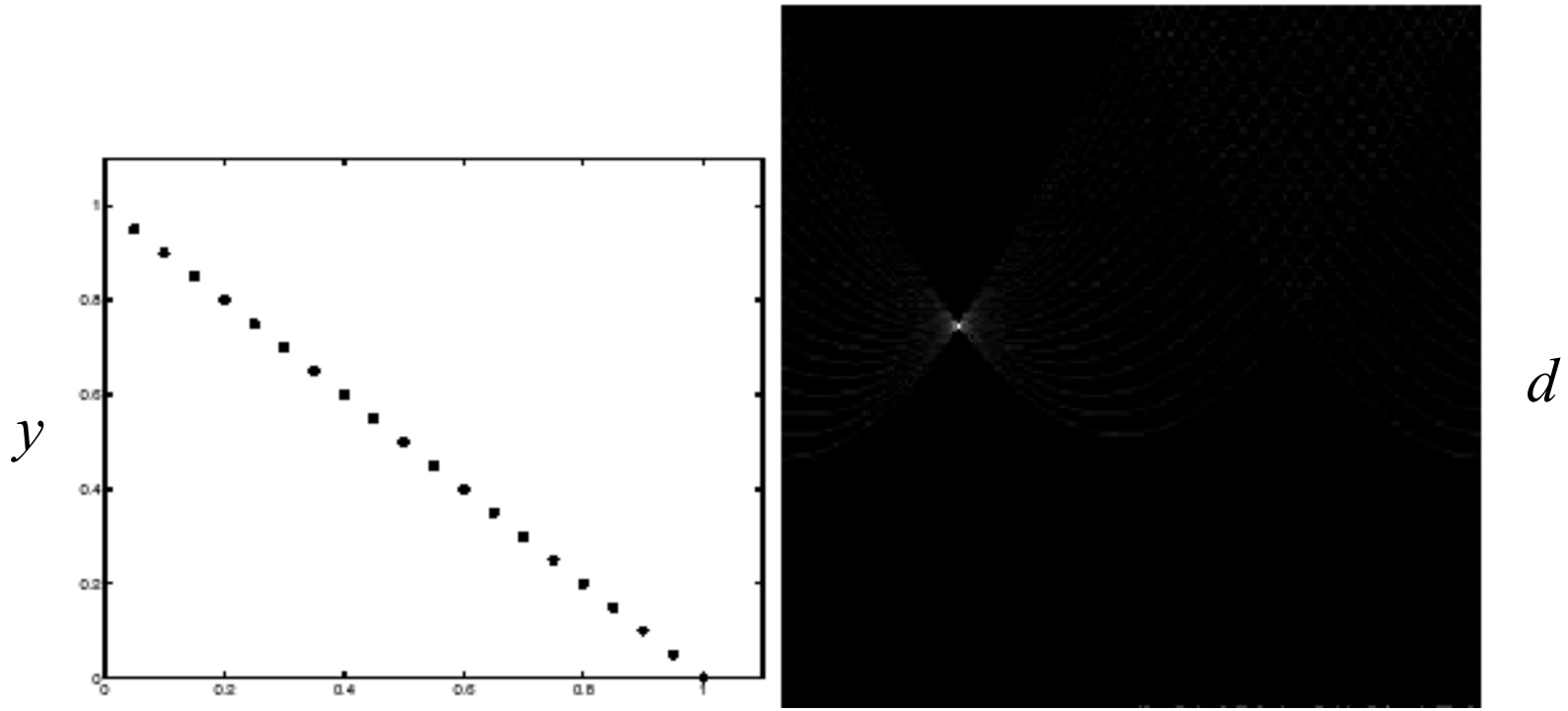


Image space
edge coordinates

Votes

Bright value = high vote count
Black = no votes

Source: K. Grauman

Impact of noise on Hough

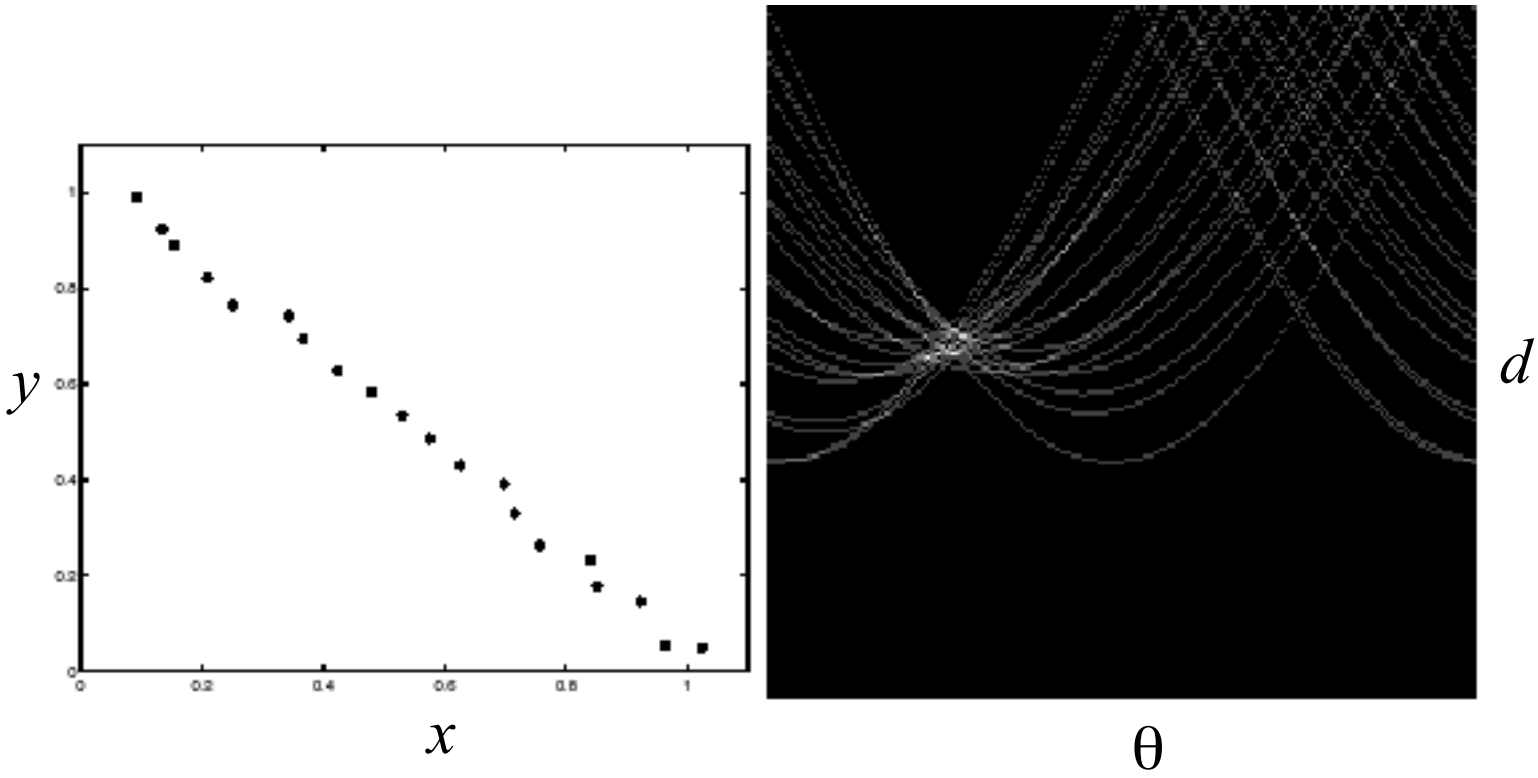


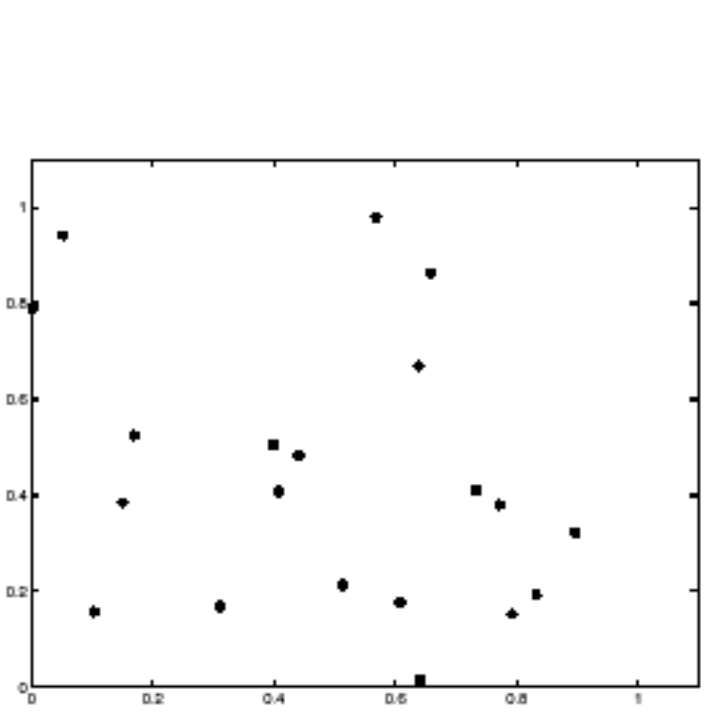
Image space
edge coordinates

Votes

What difficulty does this present for an implementation?

Source: K. Grauman

Impact of noise on Hough



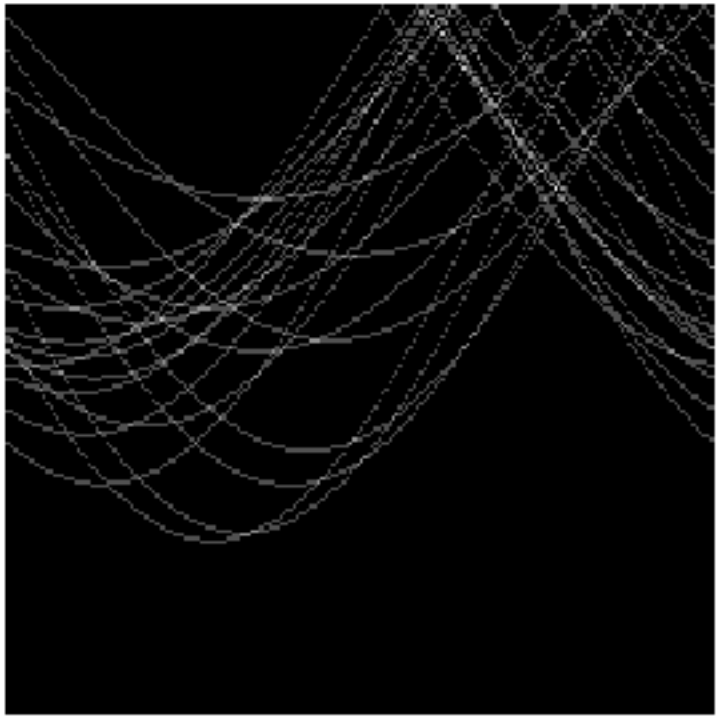


Image space
edge coordinates

Votes

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

Source: K. Grauman

Extensions

Extension 1: Use the image gradient

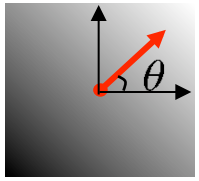
1. same
2. for each edge point I[x,y] in the image

   θ = gradient at (x,y)

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

   H[d, θ] += 1
3. same
4. same

(Reduces degrees of freedom)



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

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

Source: K. Grauman

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…

Source: K. Grauman

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

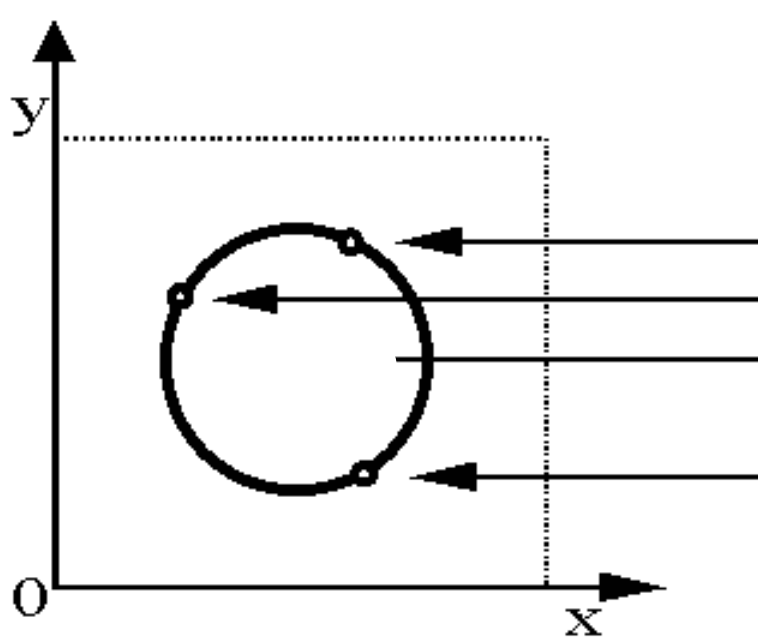
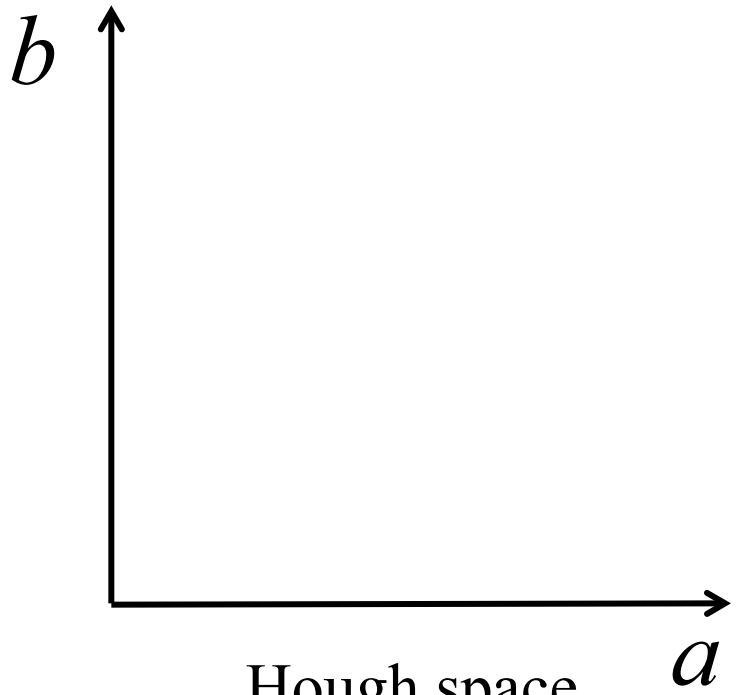


Image space



Hough space

Source: K. Grauman

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

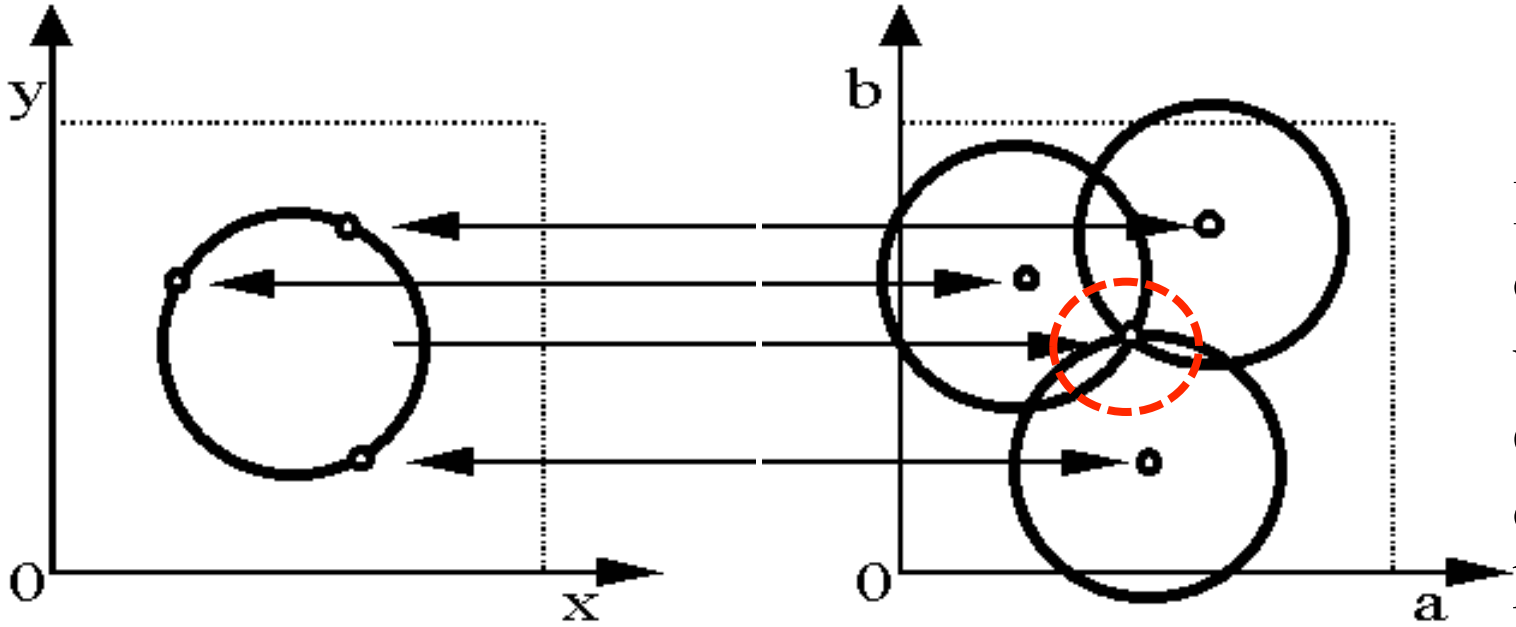


Image space

Hough space

Intersection: most votes for center occur here.

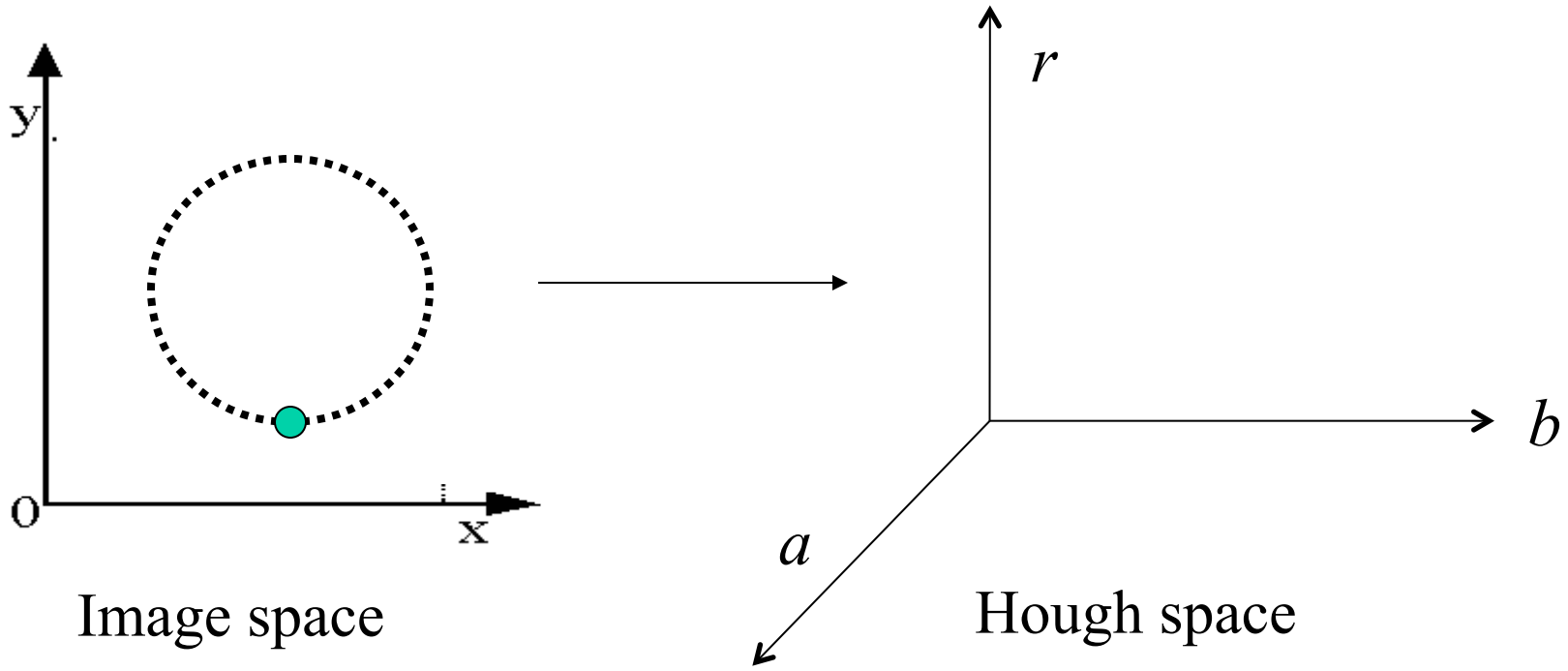
Source: K. Grauman

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



Source: K. Grauman

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

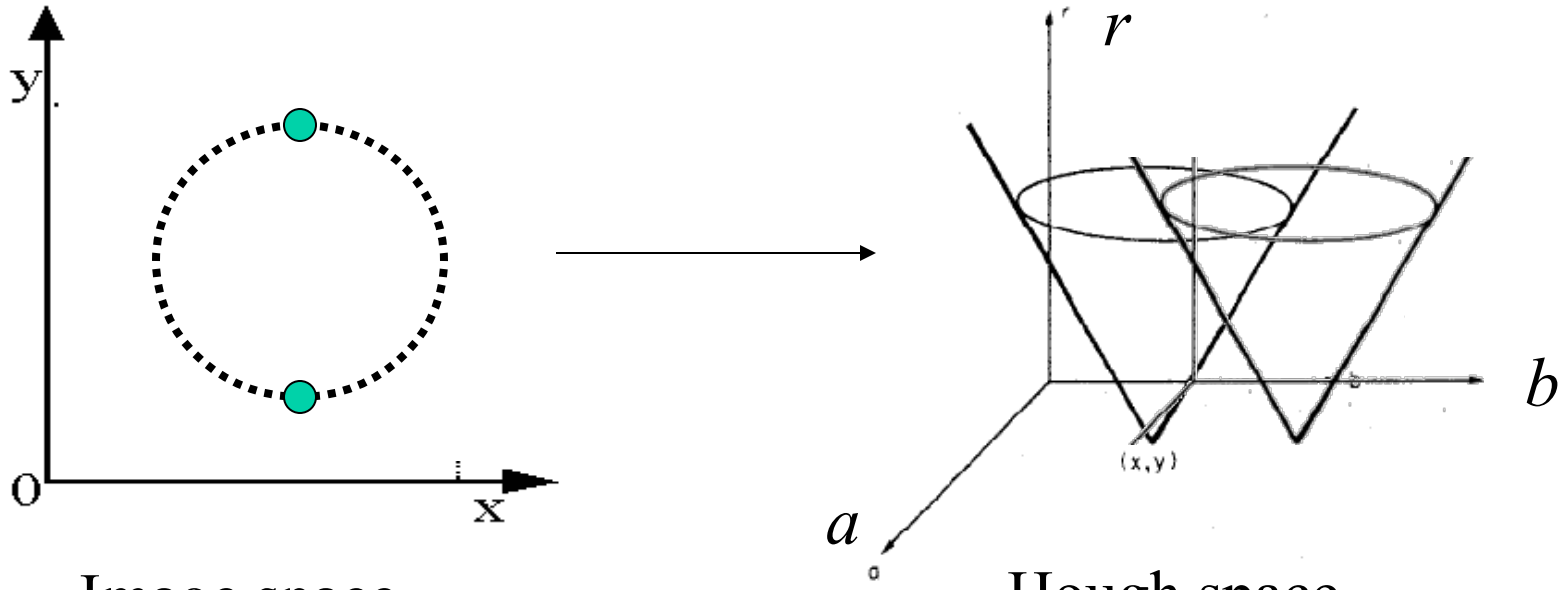


Image space

Hough space

Source: K. Grauman

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

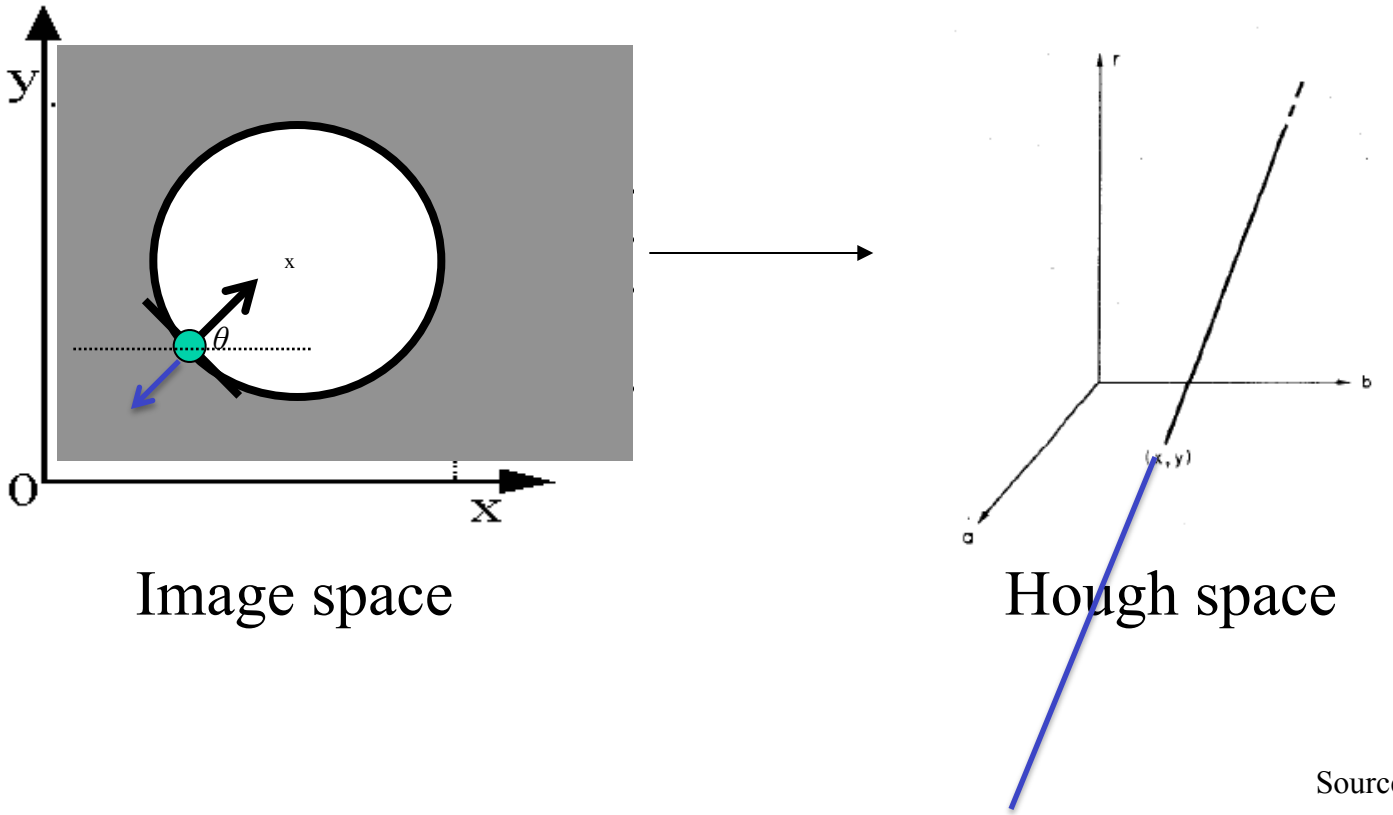


Image space

Hough space

Source: K. Grauman

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*

    $a = x + r\cos(\theta)$

    $b = y + r\sin(\theta)$

    H[a,b,r] += 1

  end

end

Source: K. Grauman

Example: detecting circles with Hough

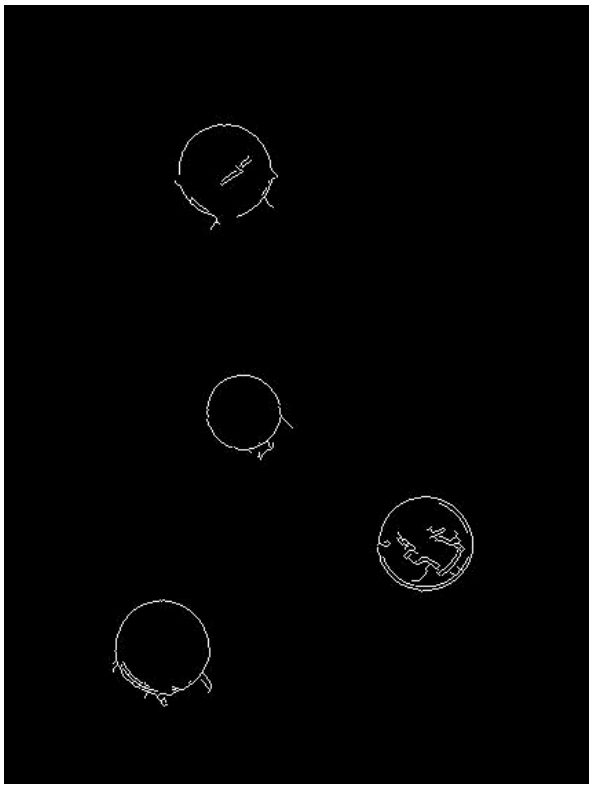
Crosshair indicates results of Hough transform, bounding box found via motion differencing.

Source: K. Grauman

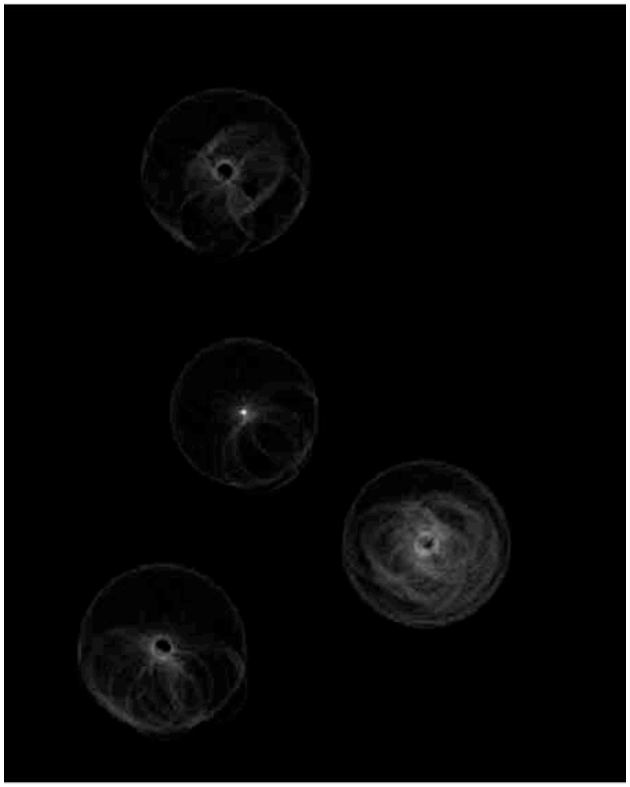
Example: detecting circles with Hough

Original

Edges



Votes: Penny

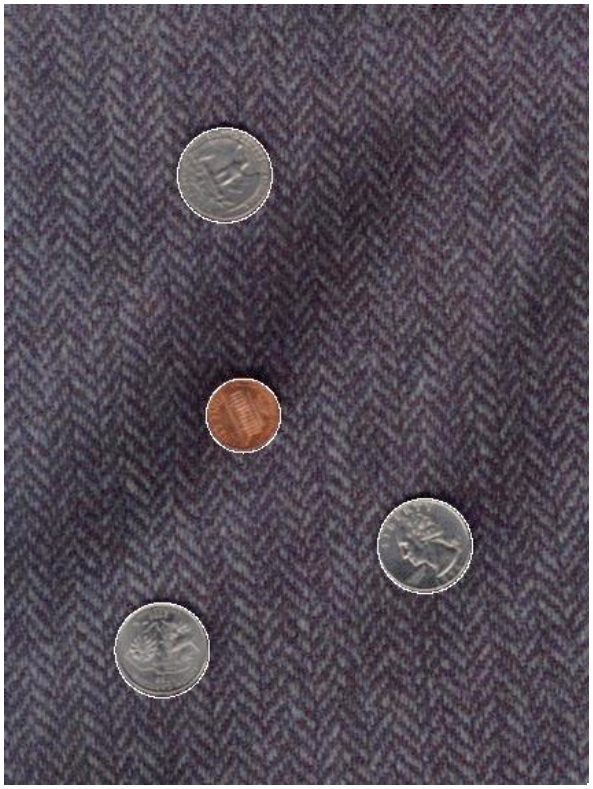


Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

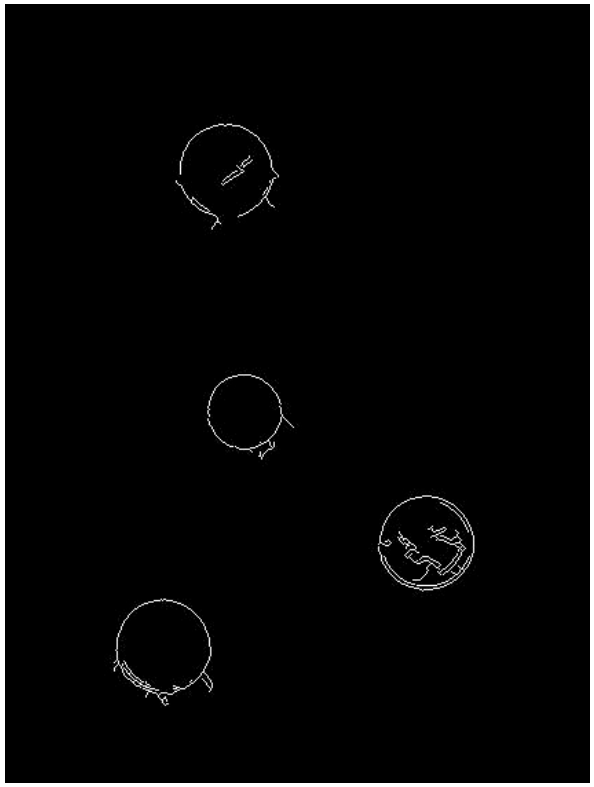
Source: K. Grauman

Example: detecting circles with Hough

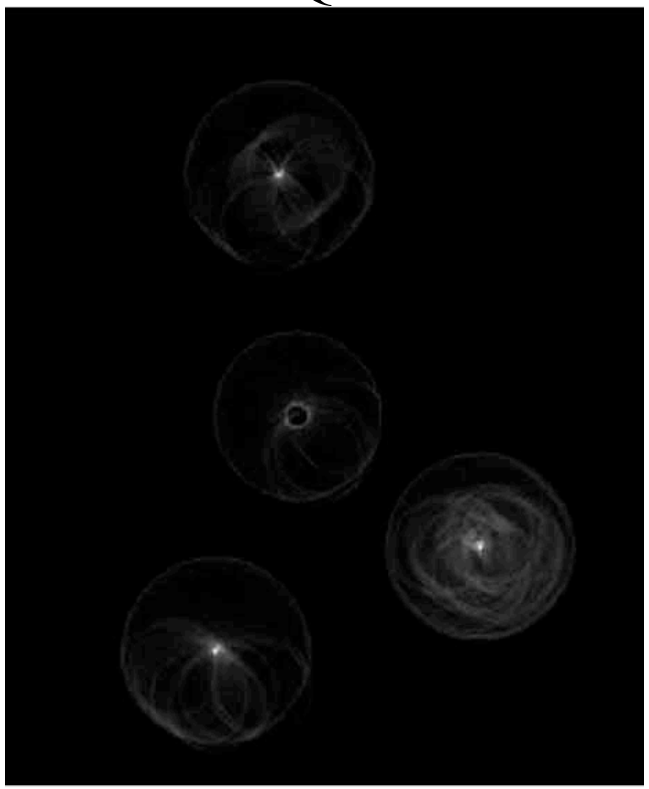
Original



Edges



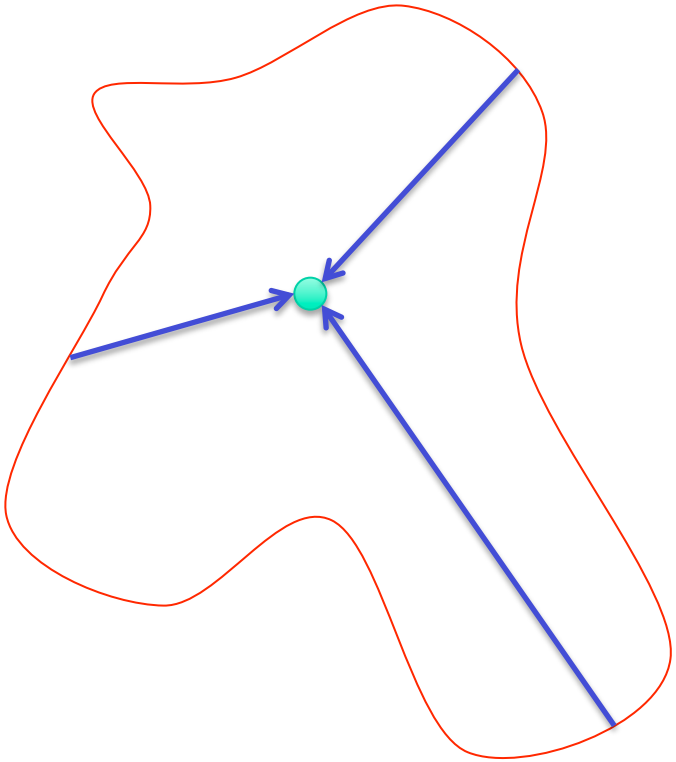
Votes: Quarter



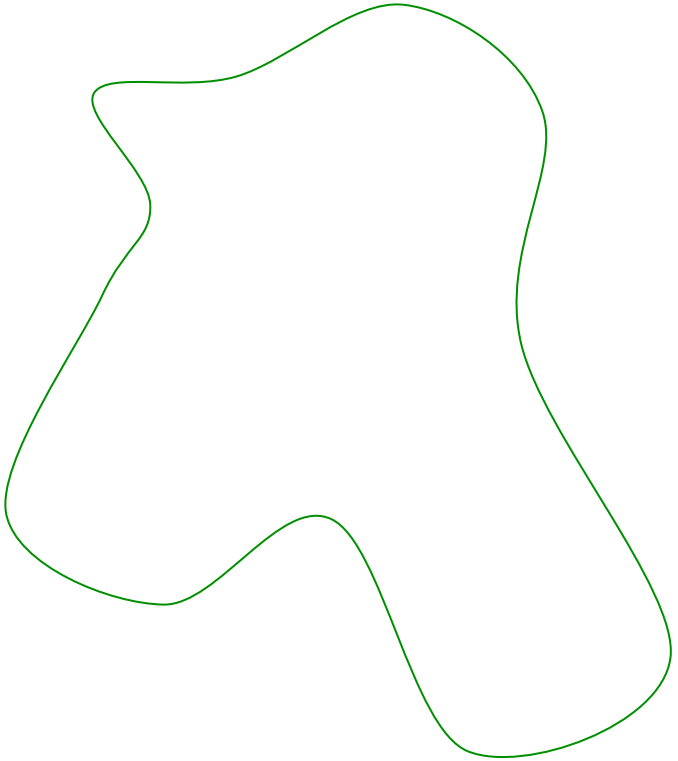
Source: K. Grauman
Coin finding sample images from: Vivek Kwatra

General Hough Transform

- Hough transform can also be used to detection arbitrary shaped object.



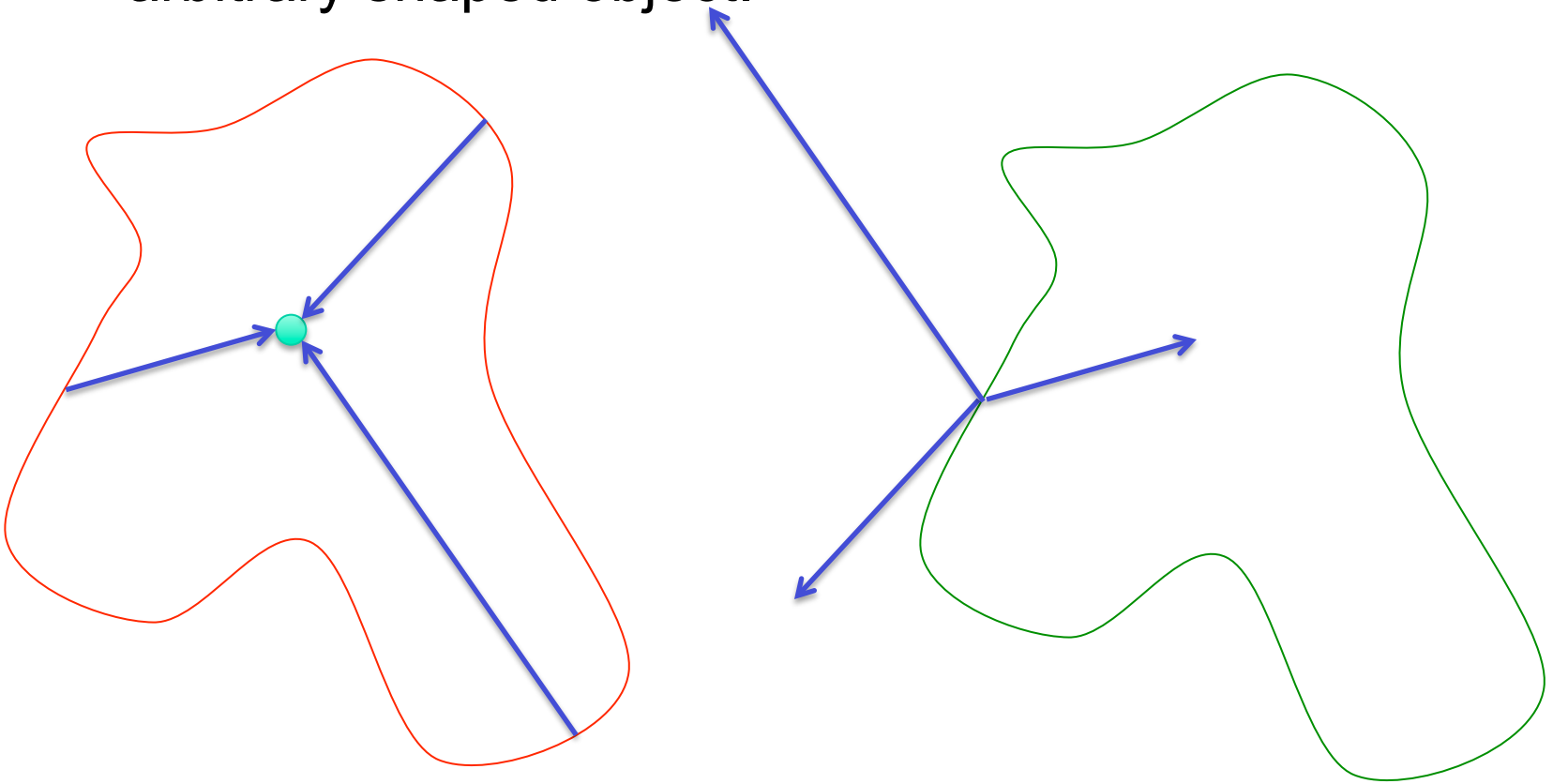
Template



Target

General Hough Transform

- Hough transform can also be used to detection arbitrary shaped object.

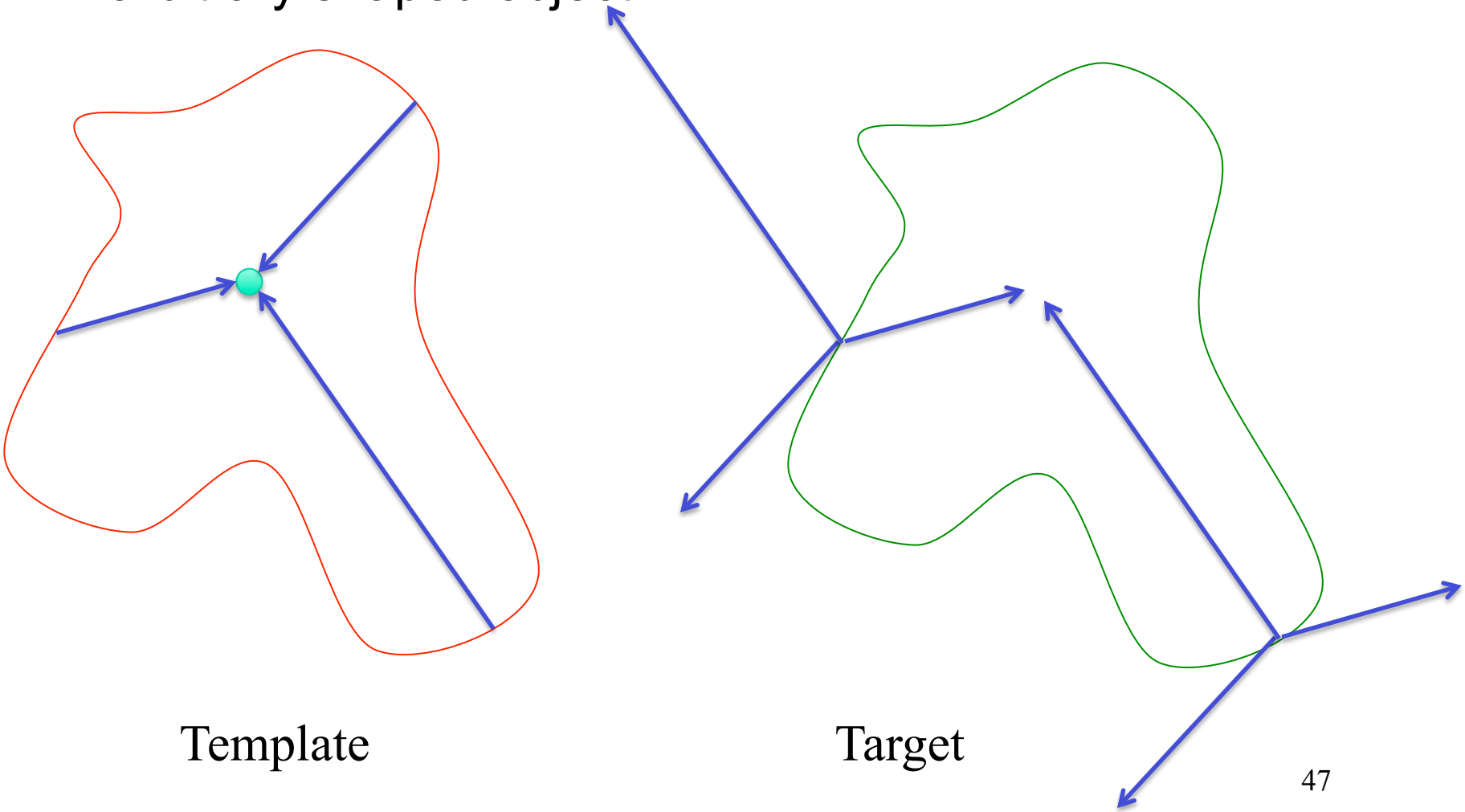


Template

Target

General Hough Transform

- Hough transform can also be used to detection arbitrary shaped object.

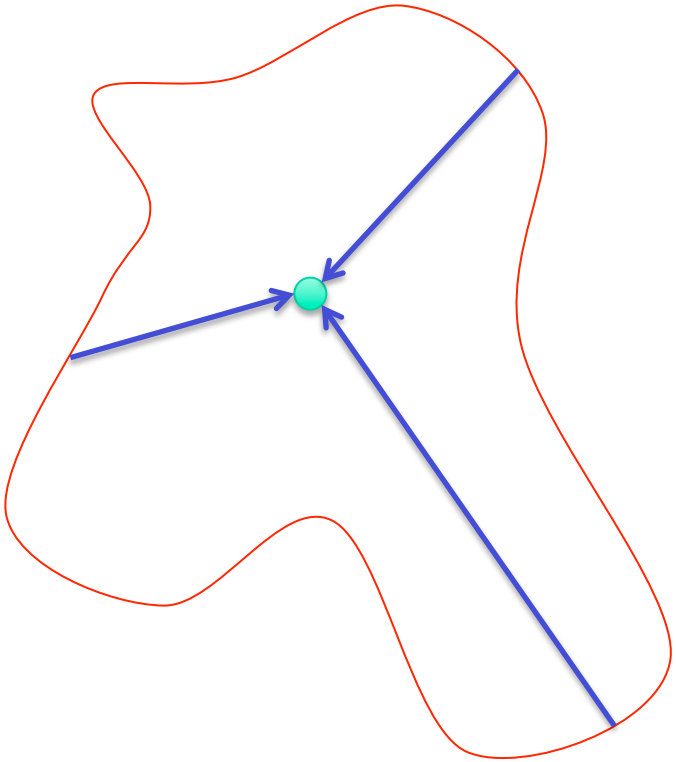


Template

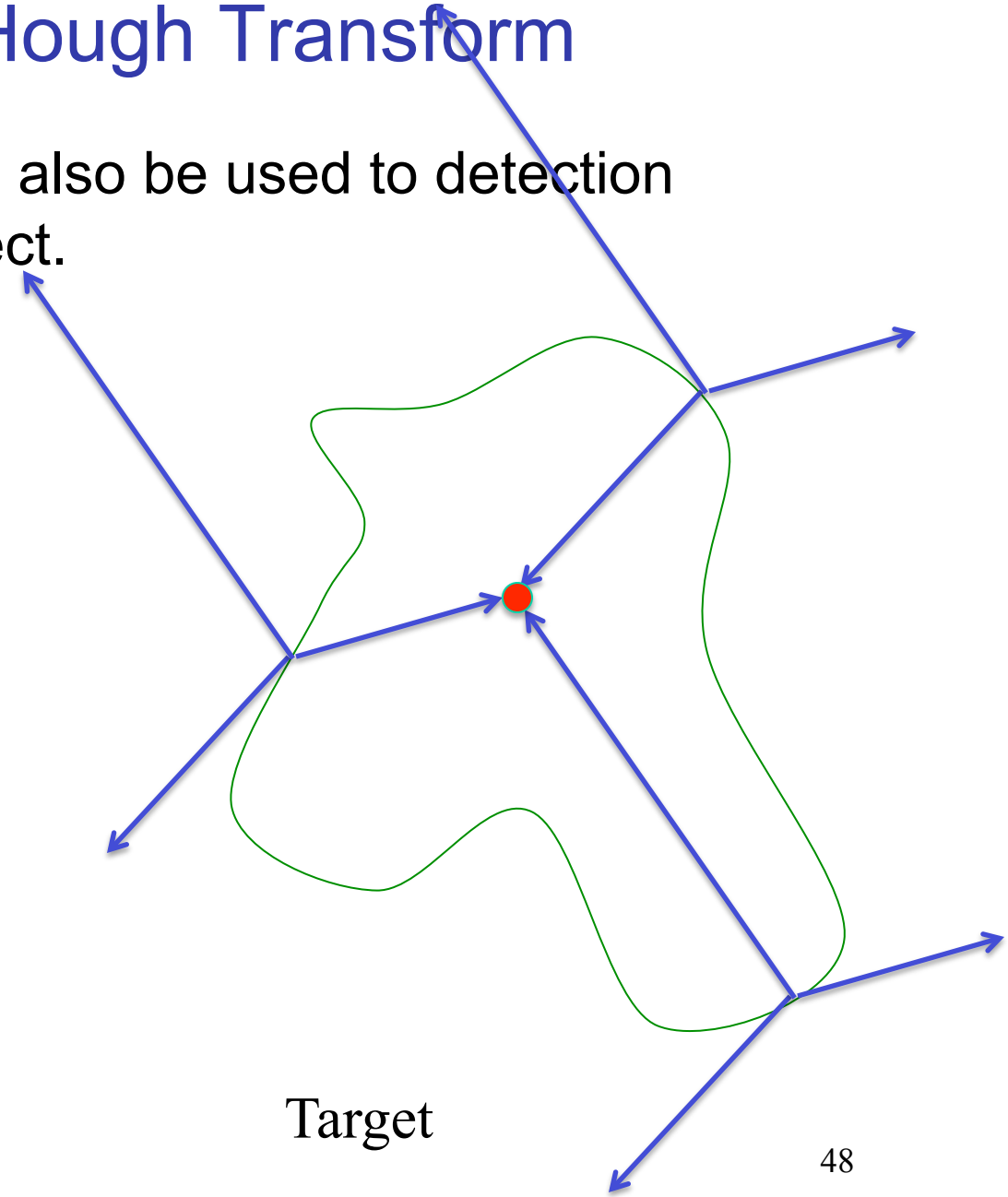
Target

General Hough Transform

- Hough transform can also be used to detection arbitrary shaped object.



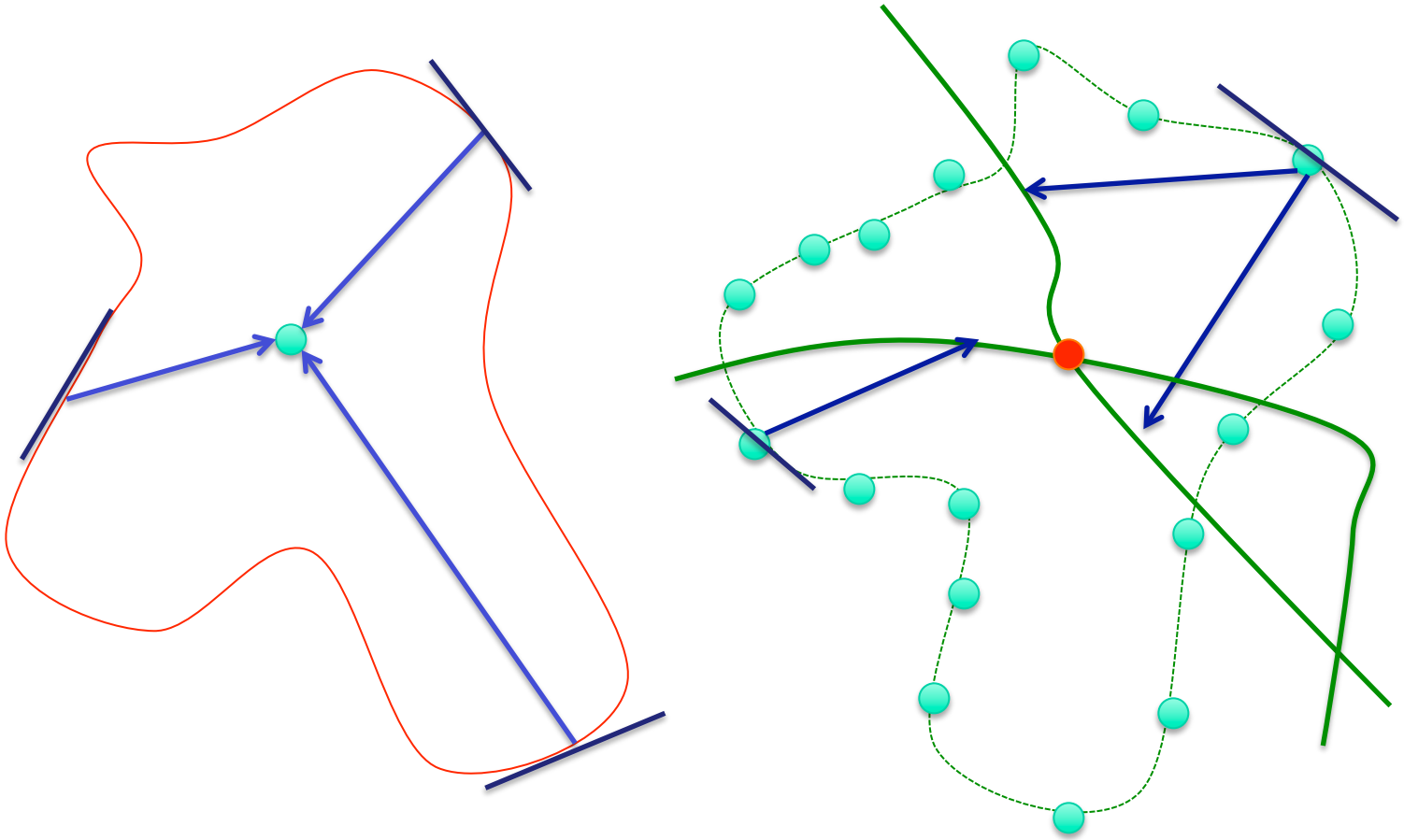
Template



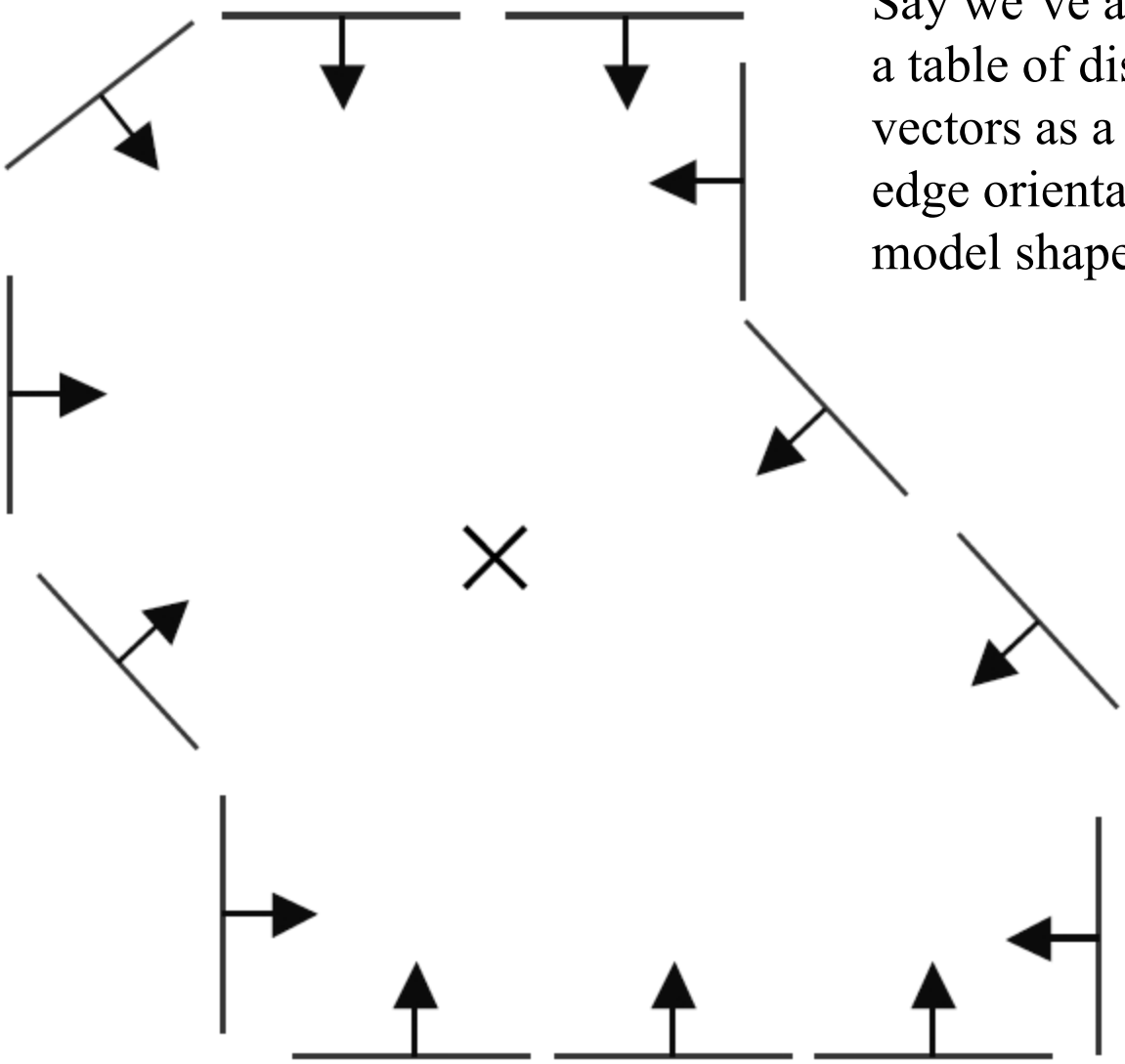
Target

General Hough Transform

- Rotation Invariance



Example

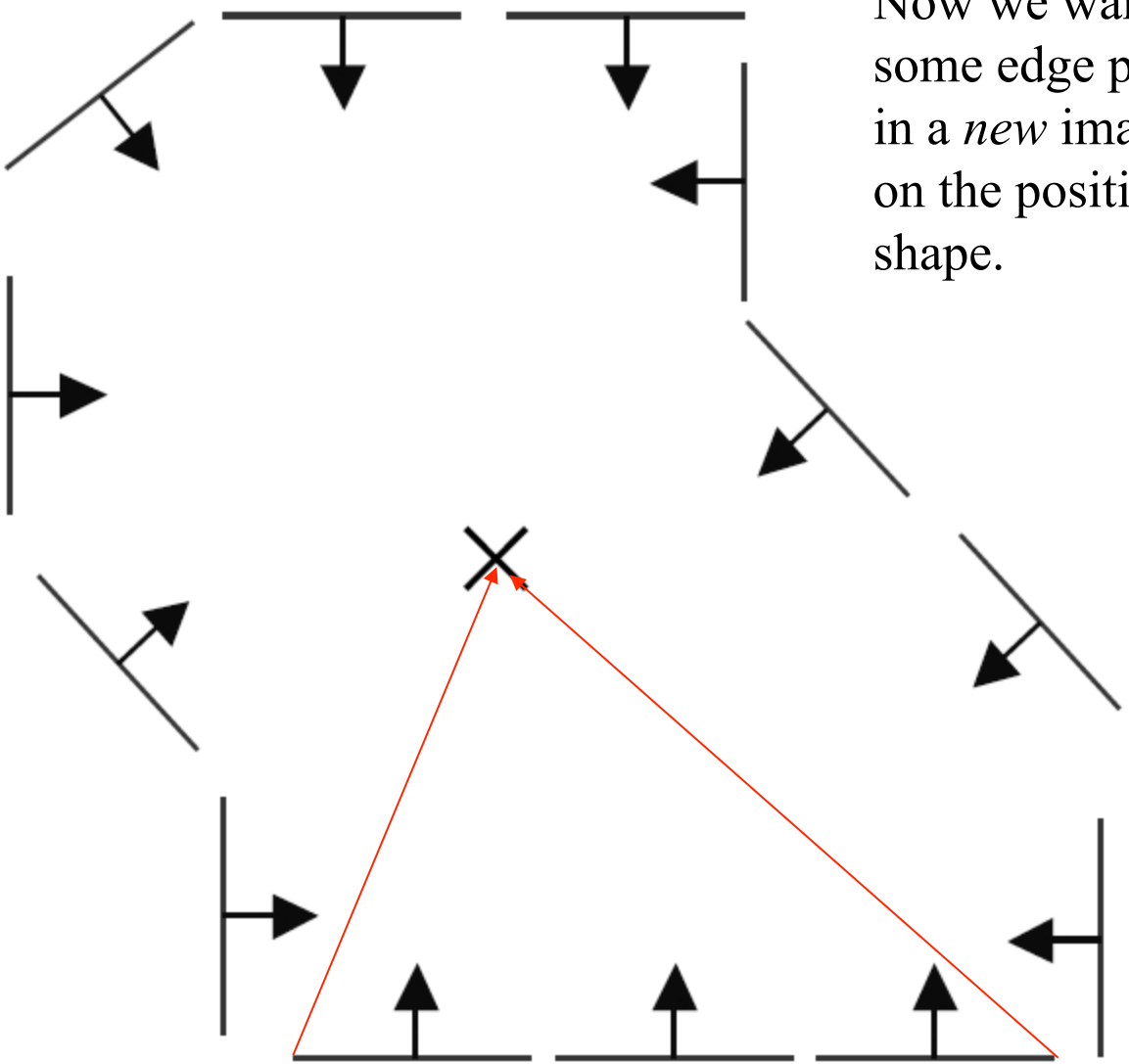


model shape

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

Source: L. Lazebnik

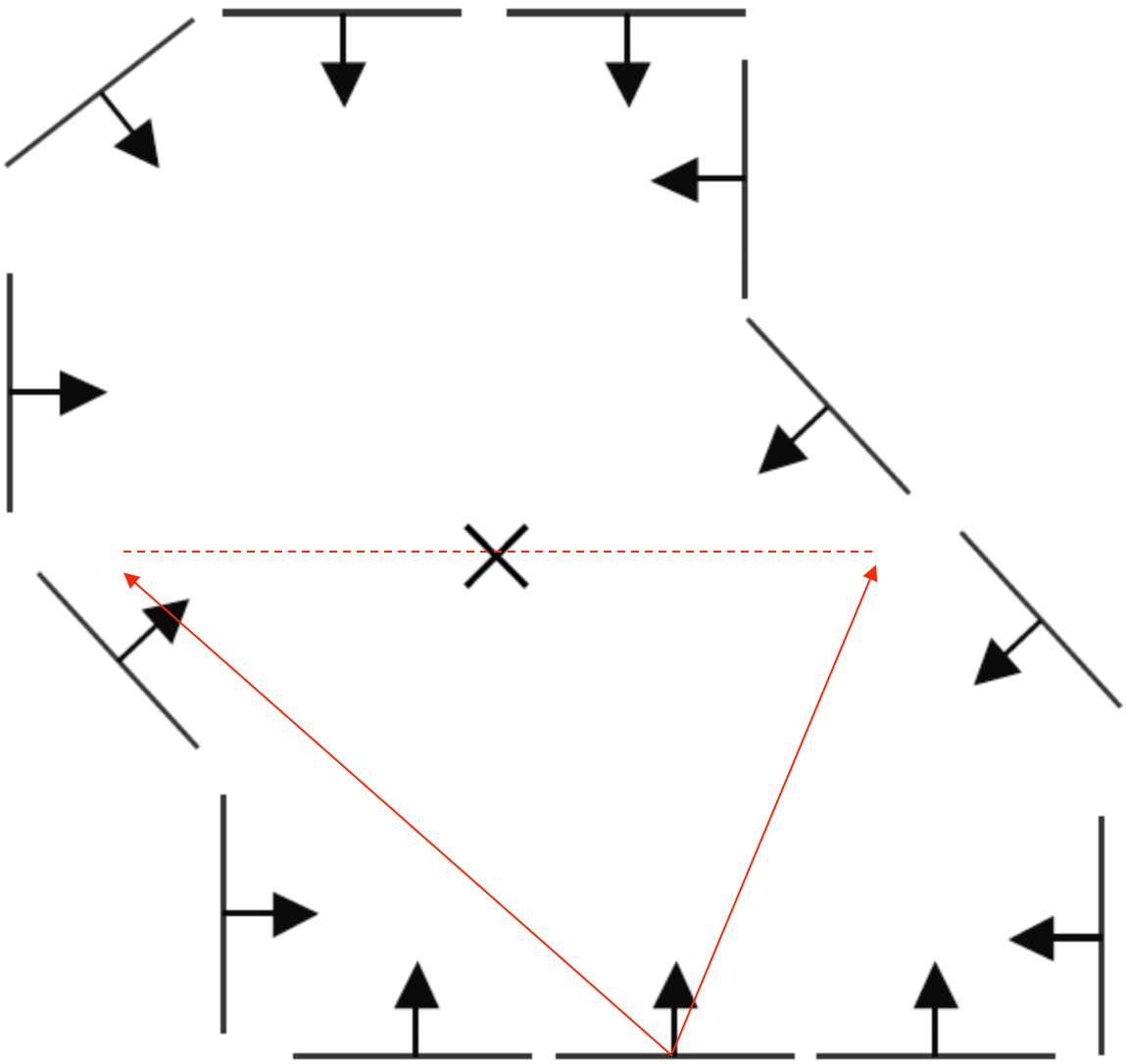
Example



displacement vectors for model points

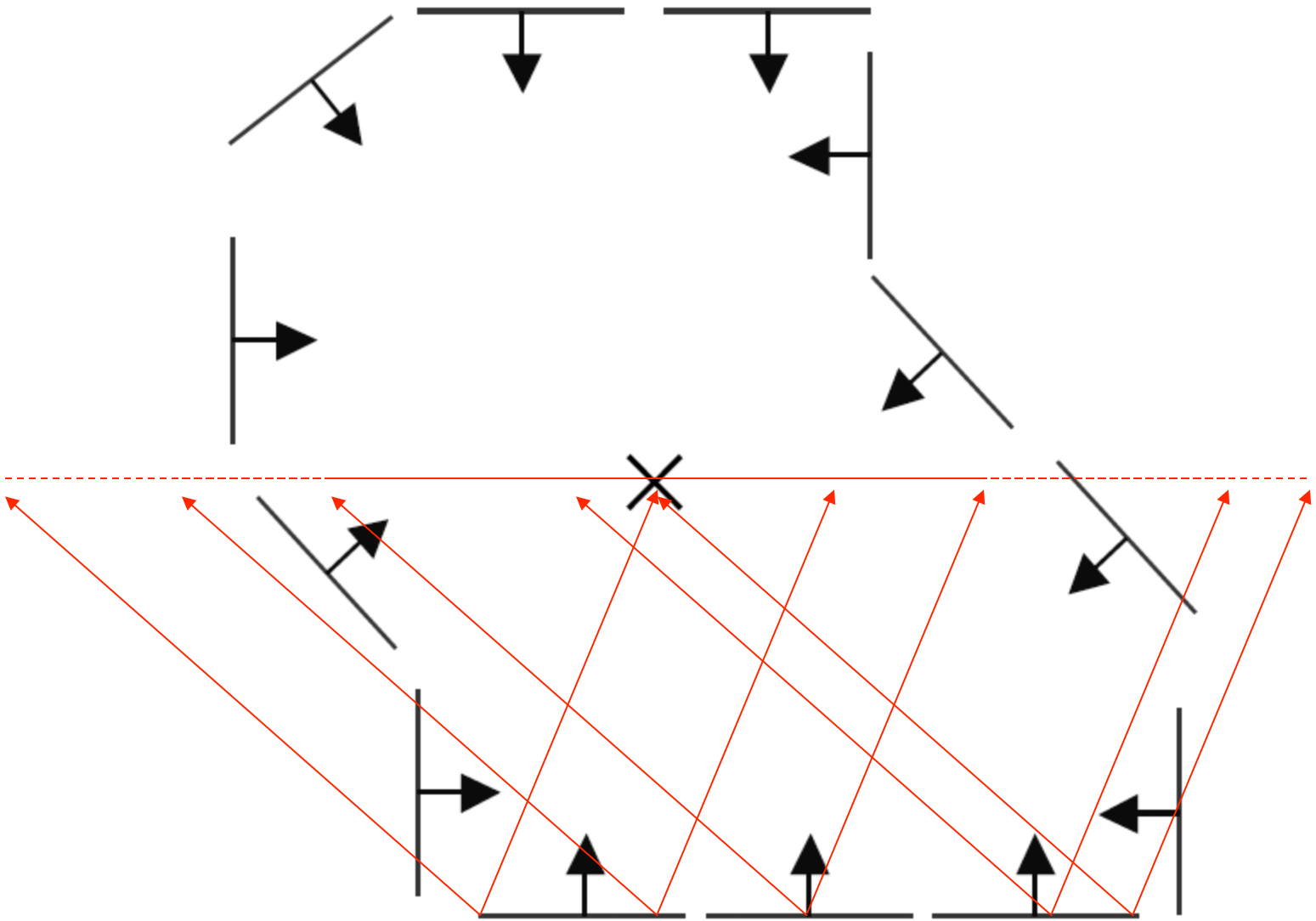
Now we want to look at some edge points detected in a *new* image, and vote on the position of that shape.

Example



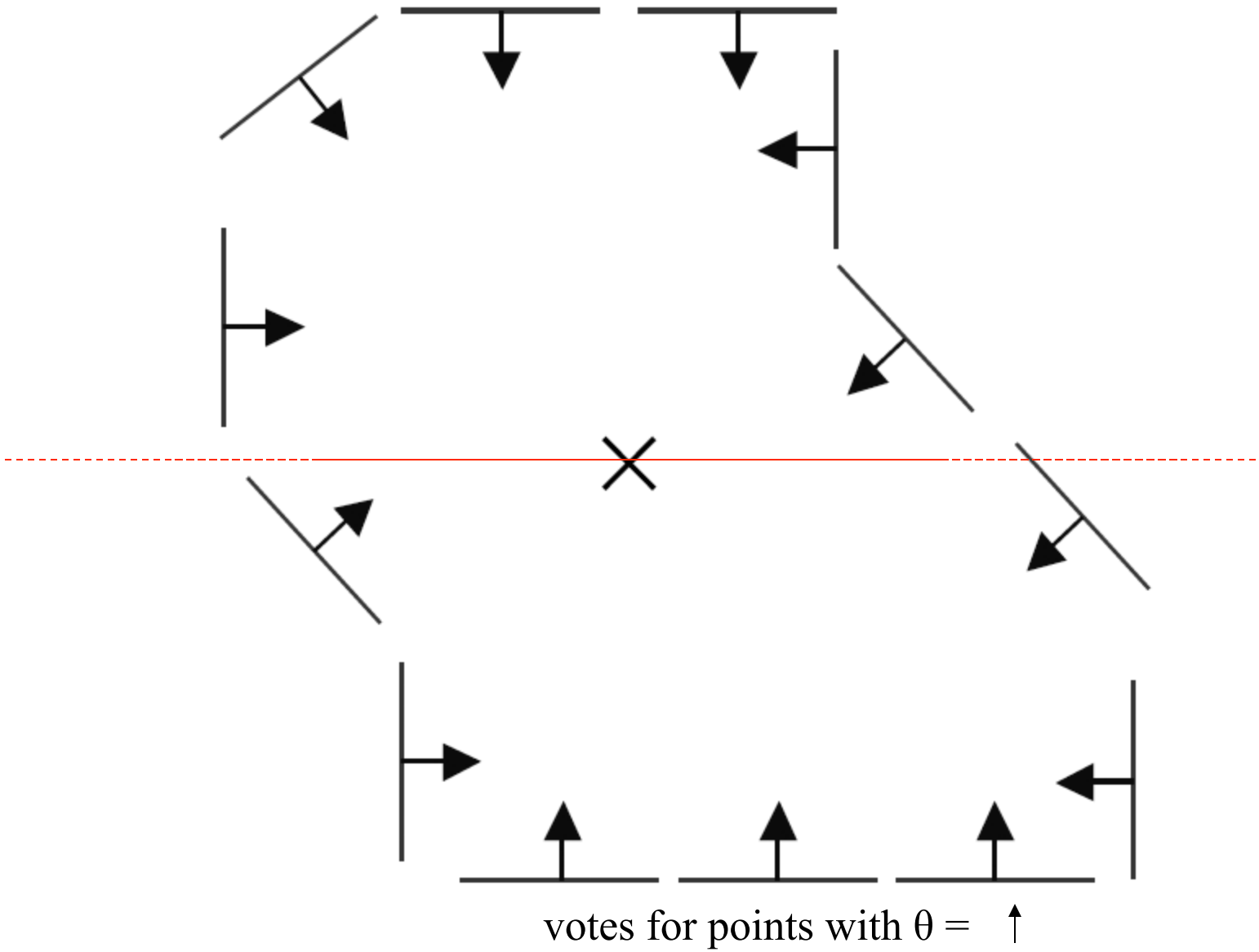
range of voting locations for test point

Example

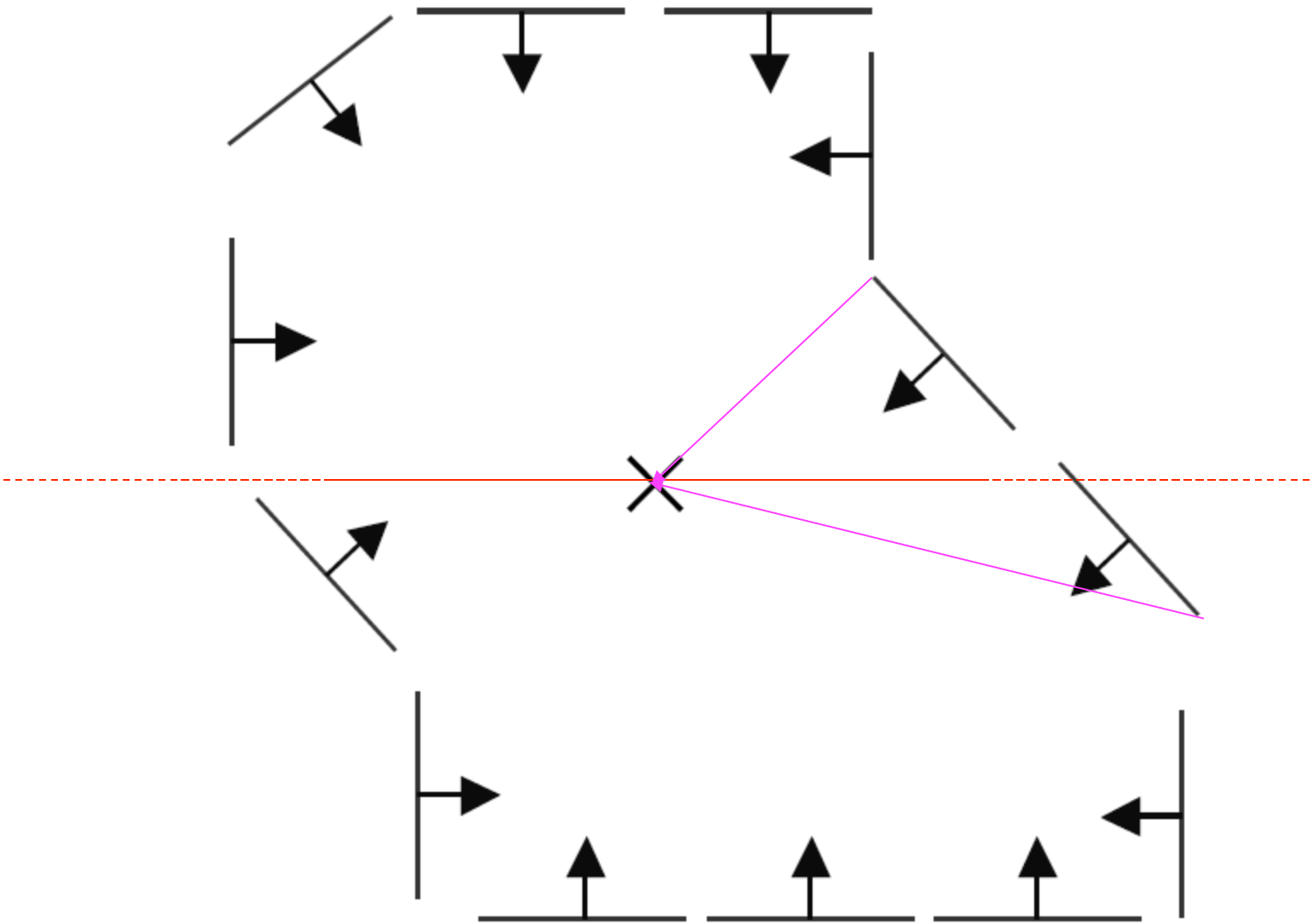


range of voting locations for test point

Example

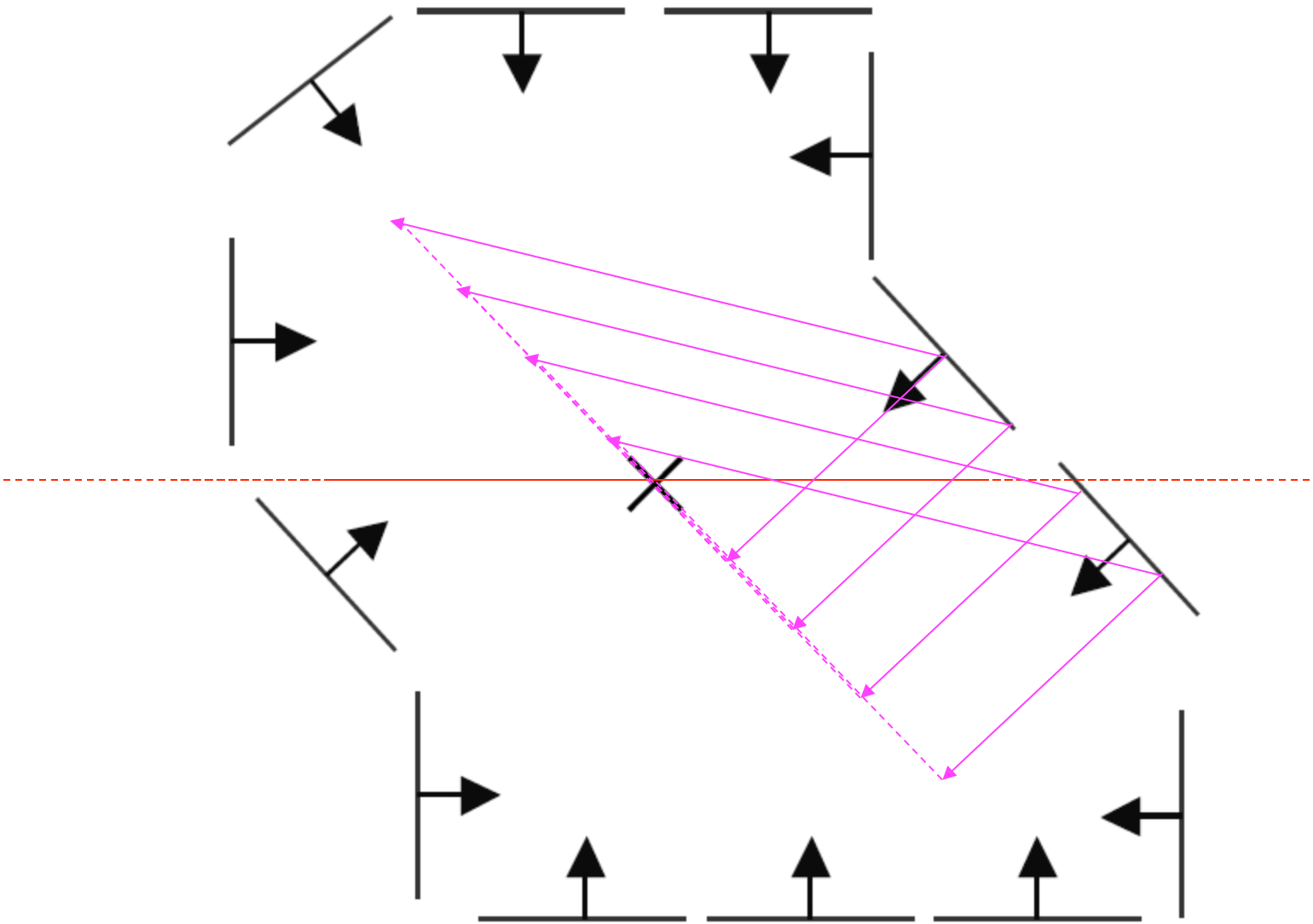


Example



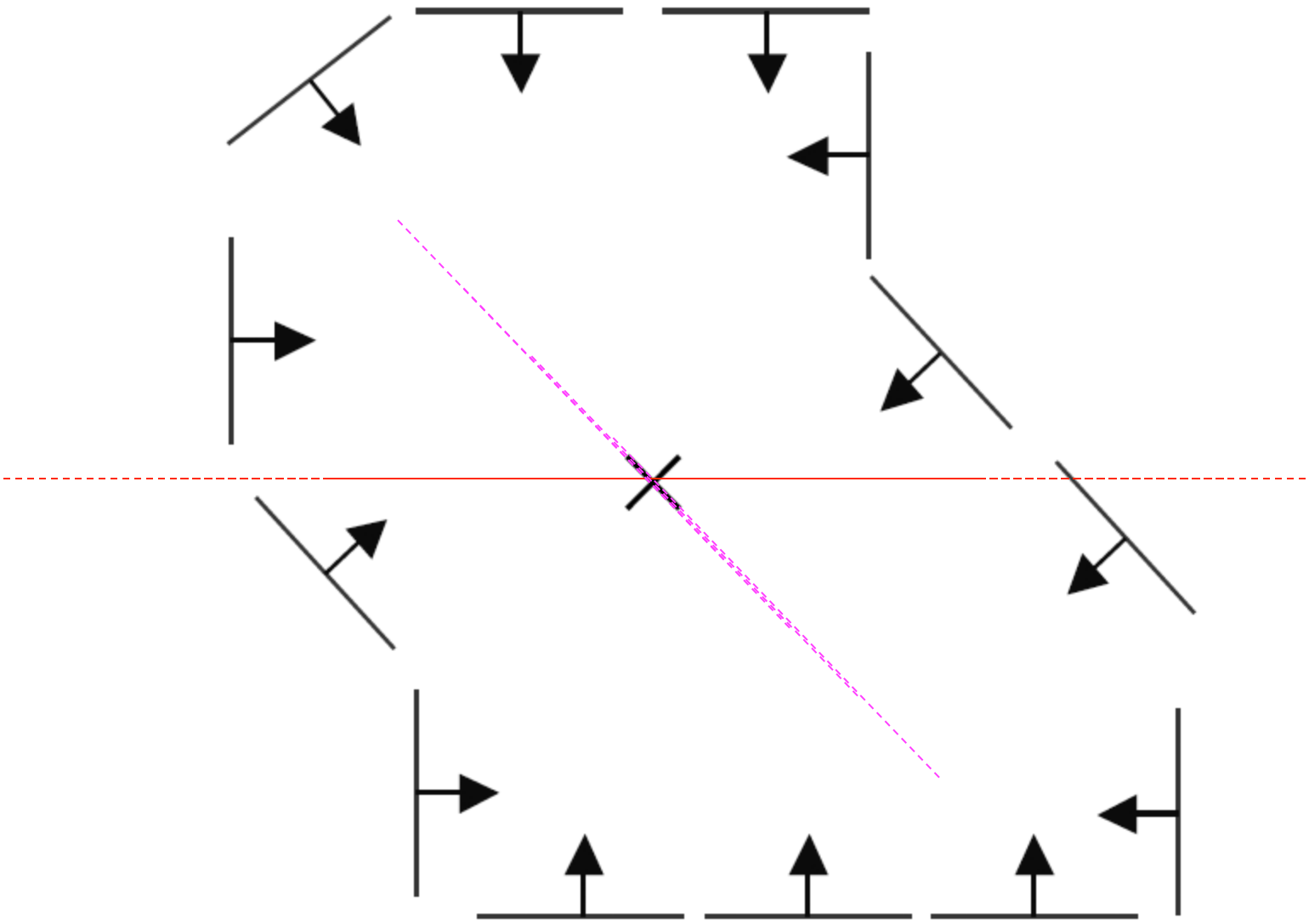
displacement vectors for model points

Example



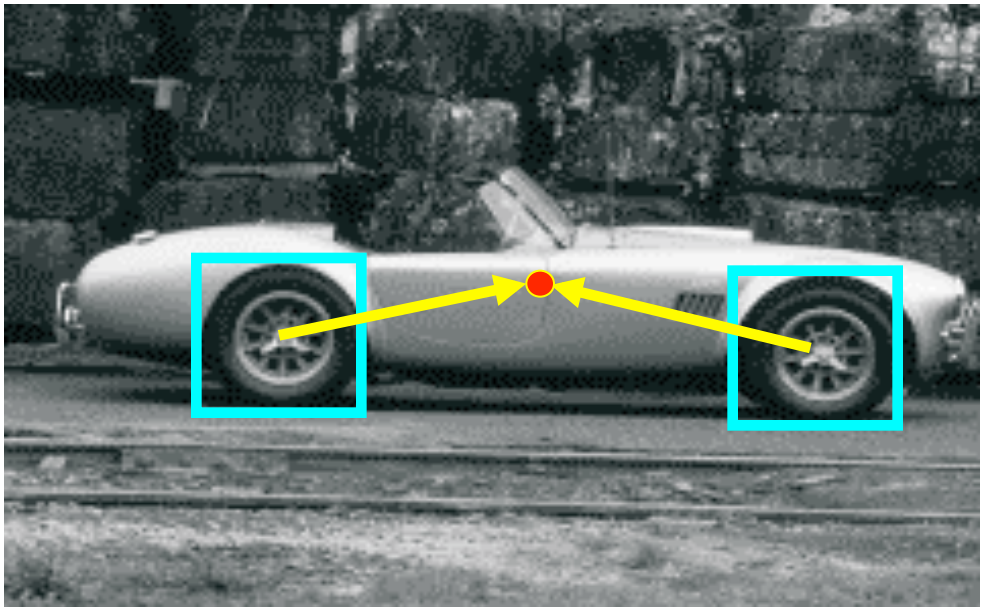
range of voting locations for test point

Example



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

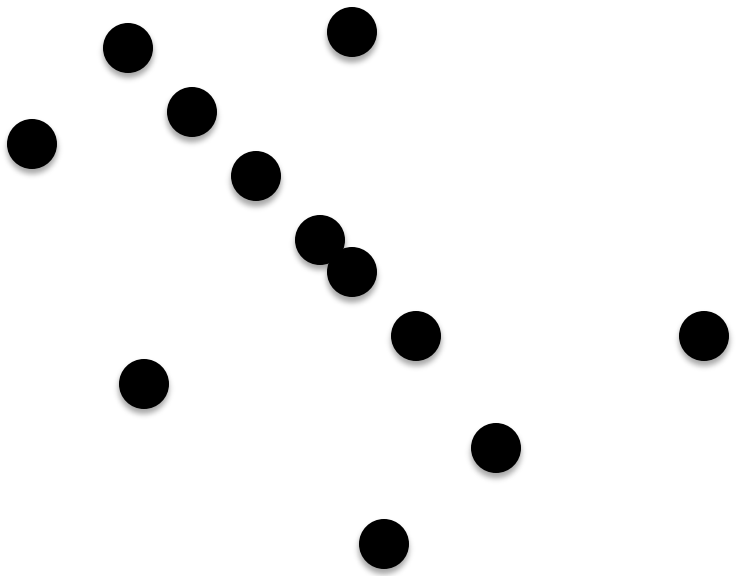
Source: L. Lazebnik

RANSAC

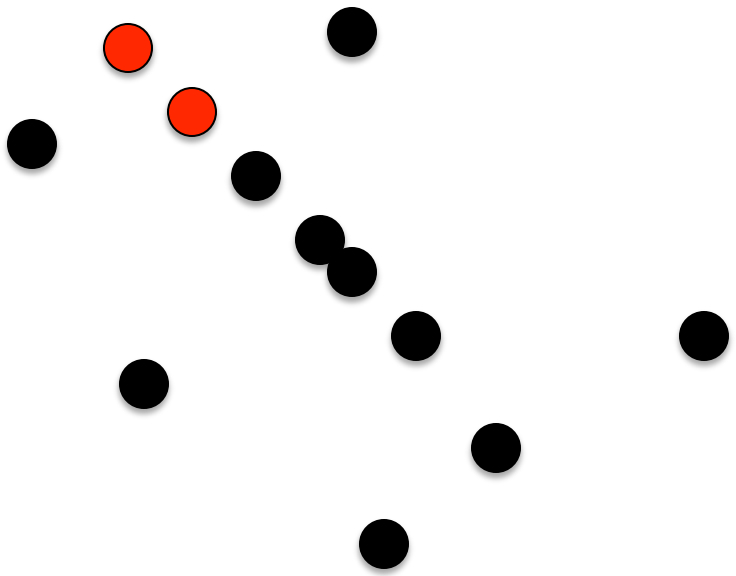
- RANSAC – Random Sampling Consensus

- Procedure:
  - Randomly generate a hypothesis
  - Test the hypothesis
  - Repeat for large enough times and choose the best one

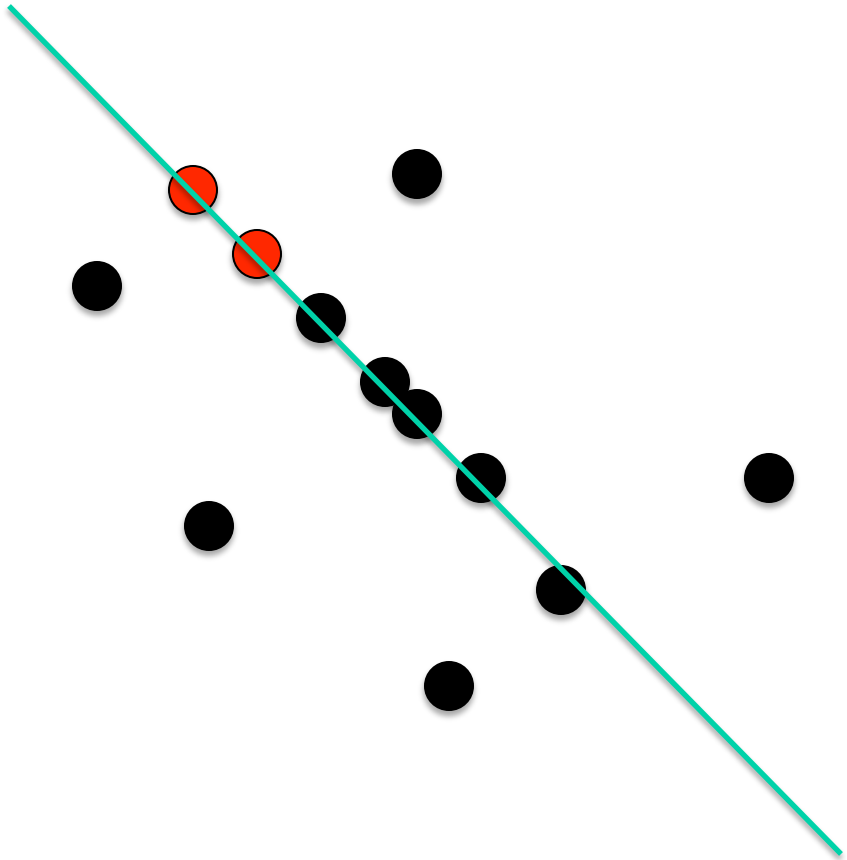
Line Detection



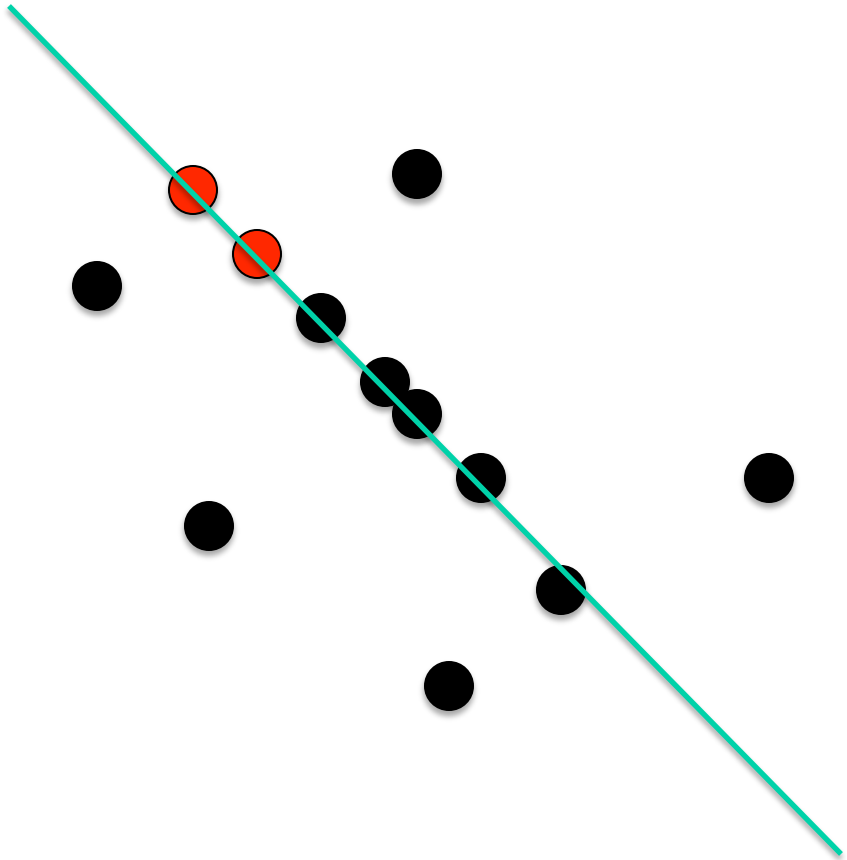
Line Detection



Line Detection

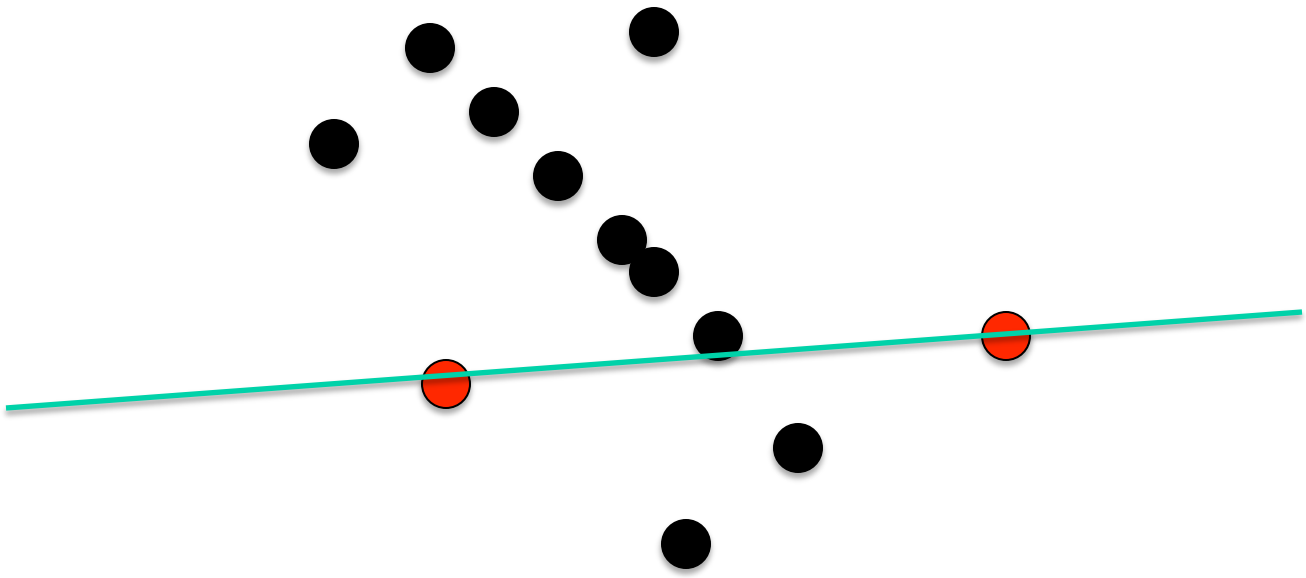


Line Detection



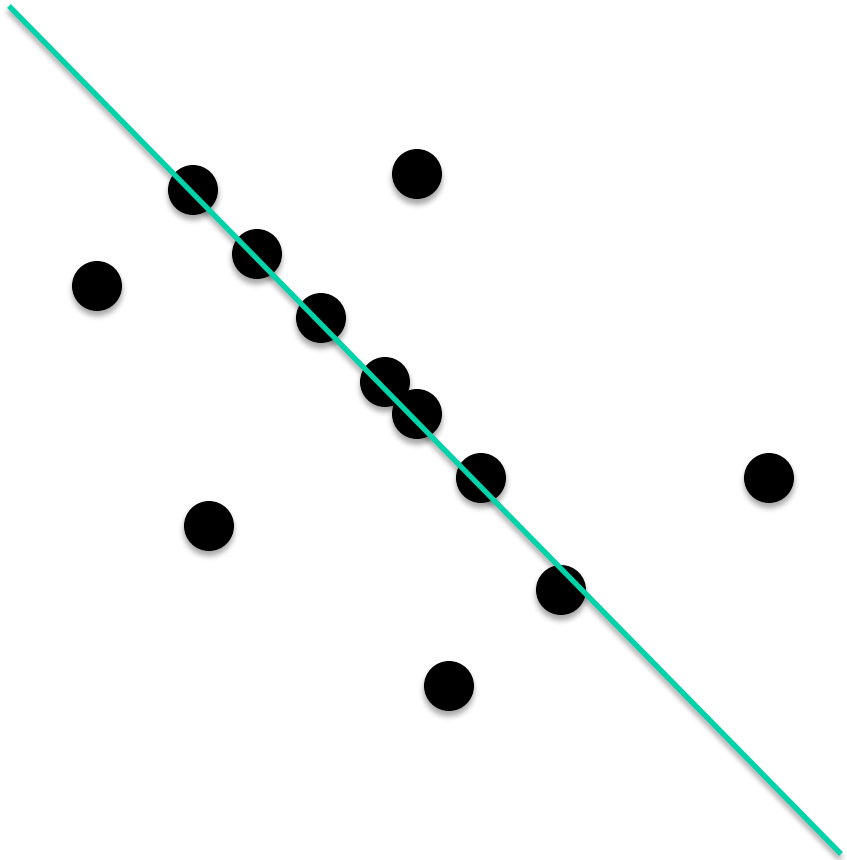
Count “inliers”: 7

Line Detection



Inlier number: 3

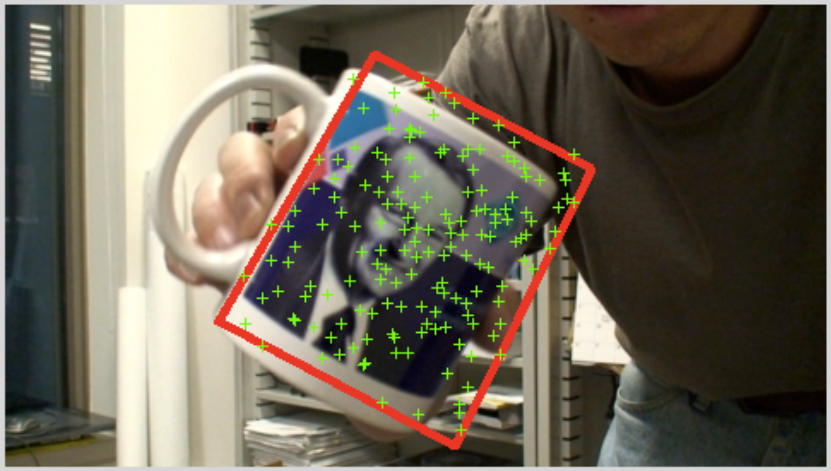
Line Detection



After many trails, we decide this is the best line.

Object Detection Using RANSAC

Template

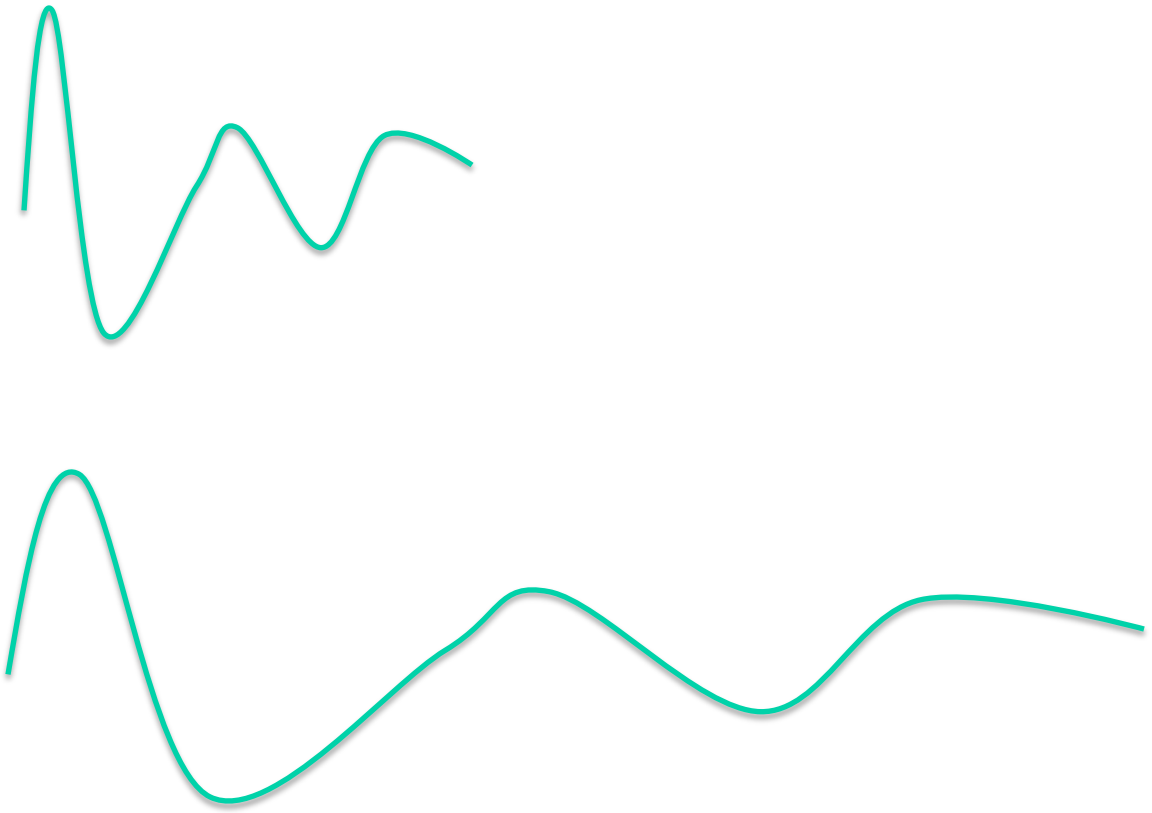


Target

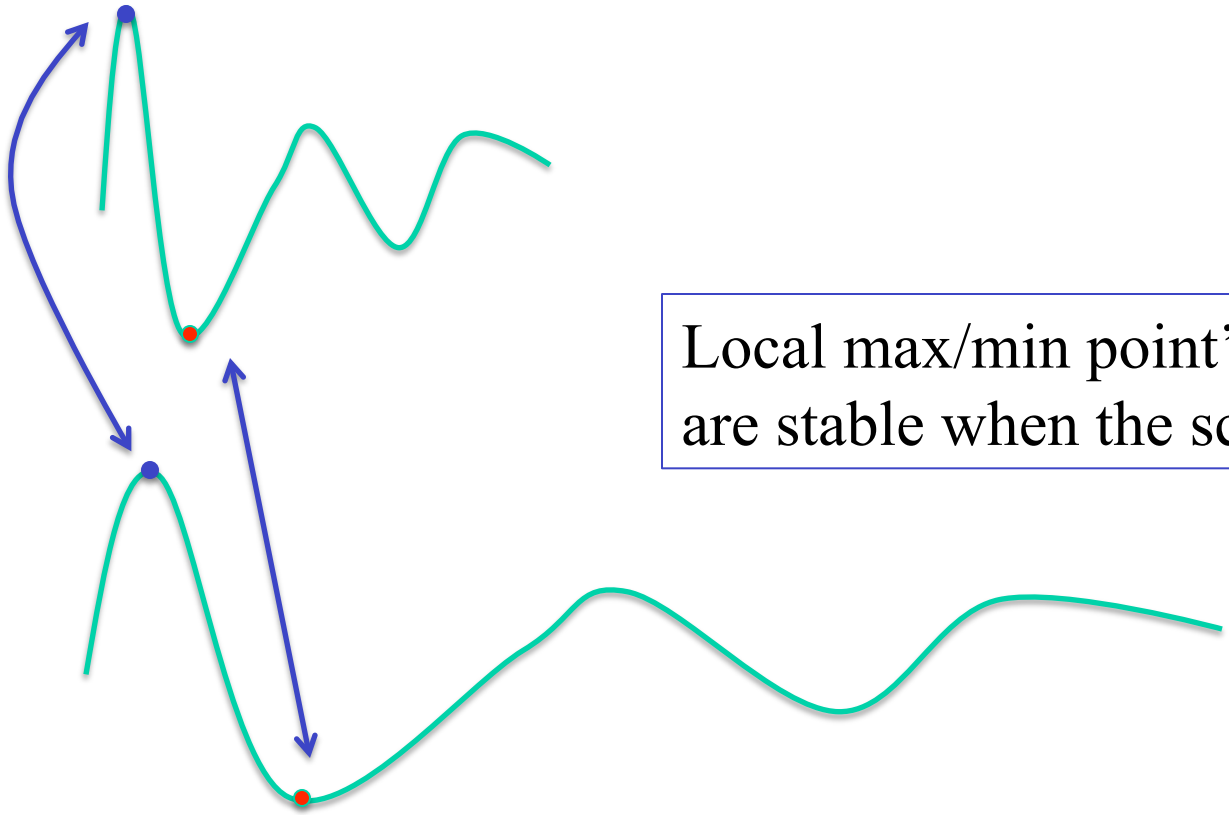
Scale and Rotation Invariant Feature

- SIFT (D. Lowe)

Stable Feature

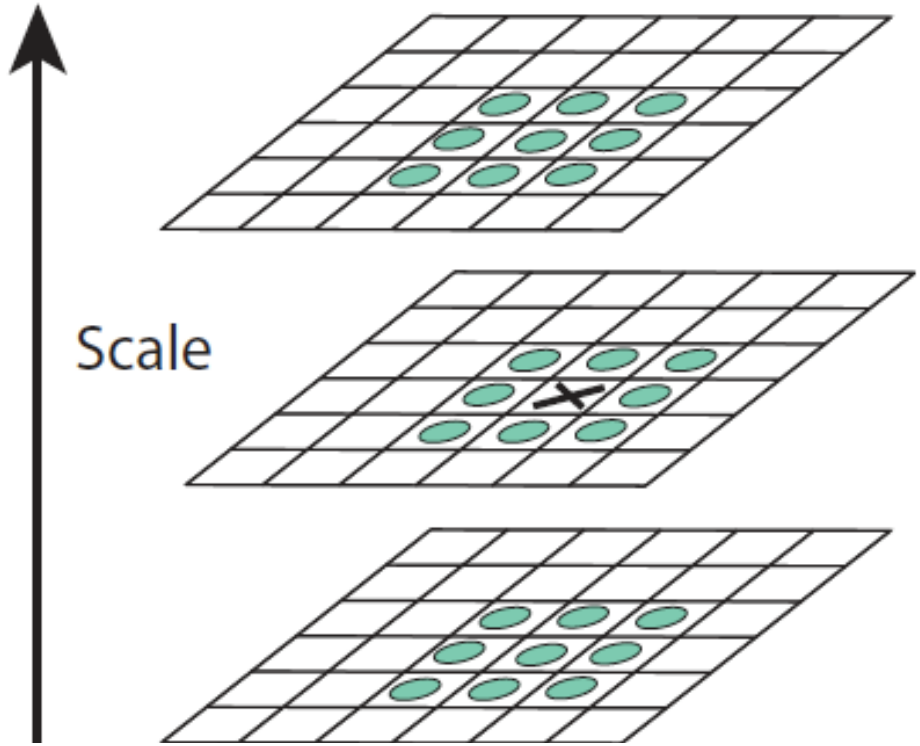


Stable Feature



Local max/min point's values are stable when the scale changes

SIFT



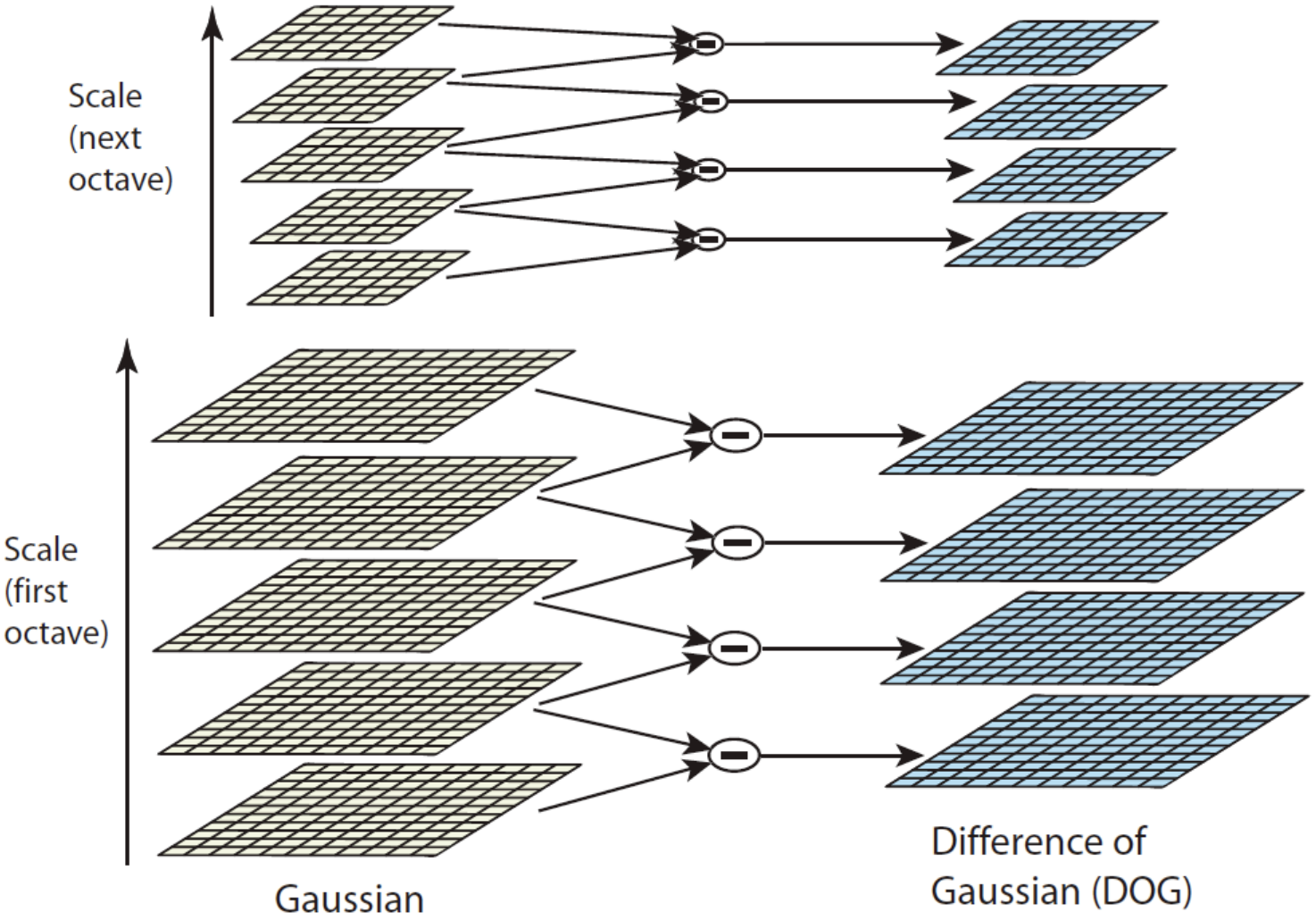
Filtering the image using filters at different scales. (for example using Gaussian filter)

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

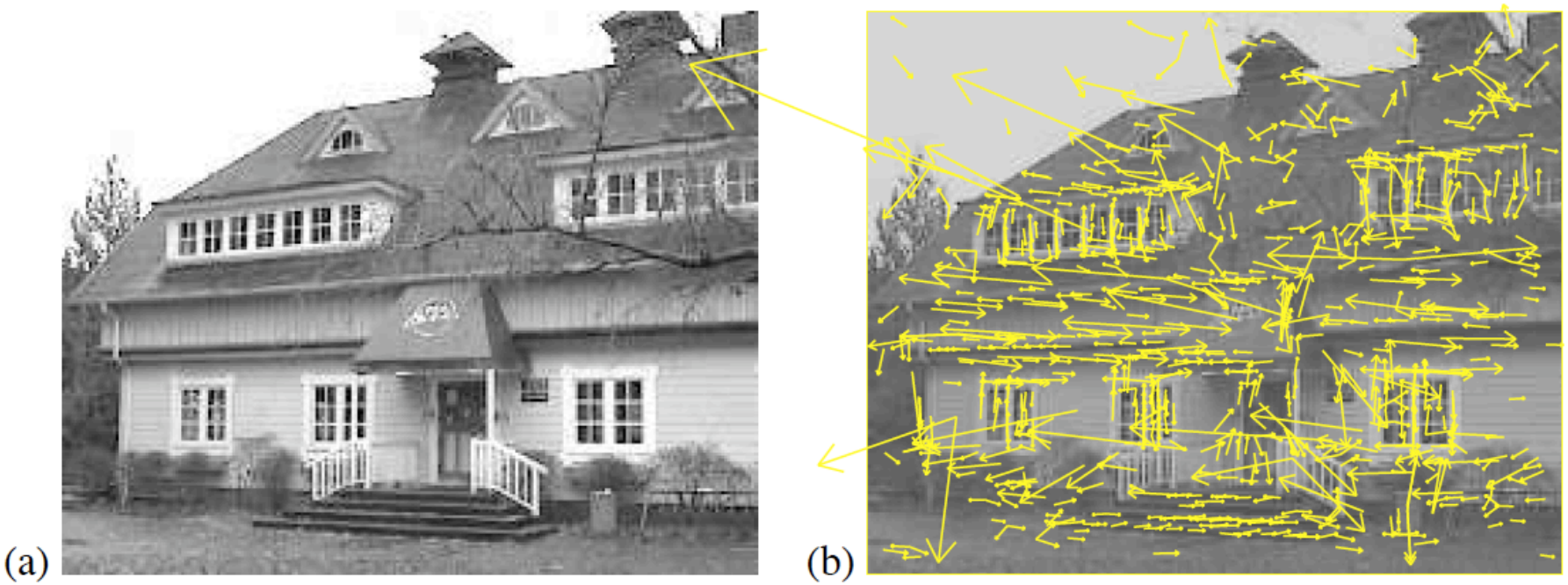
where $*$ is the convolution operation in x and y, and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}.$$

Difference of Gaussian



SIFT Feature Points



(b) Shows the points at the local max/min of DOG scale space for the image in (a).

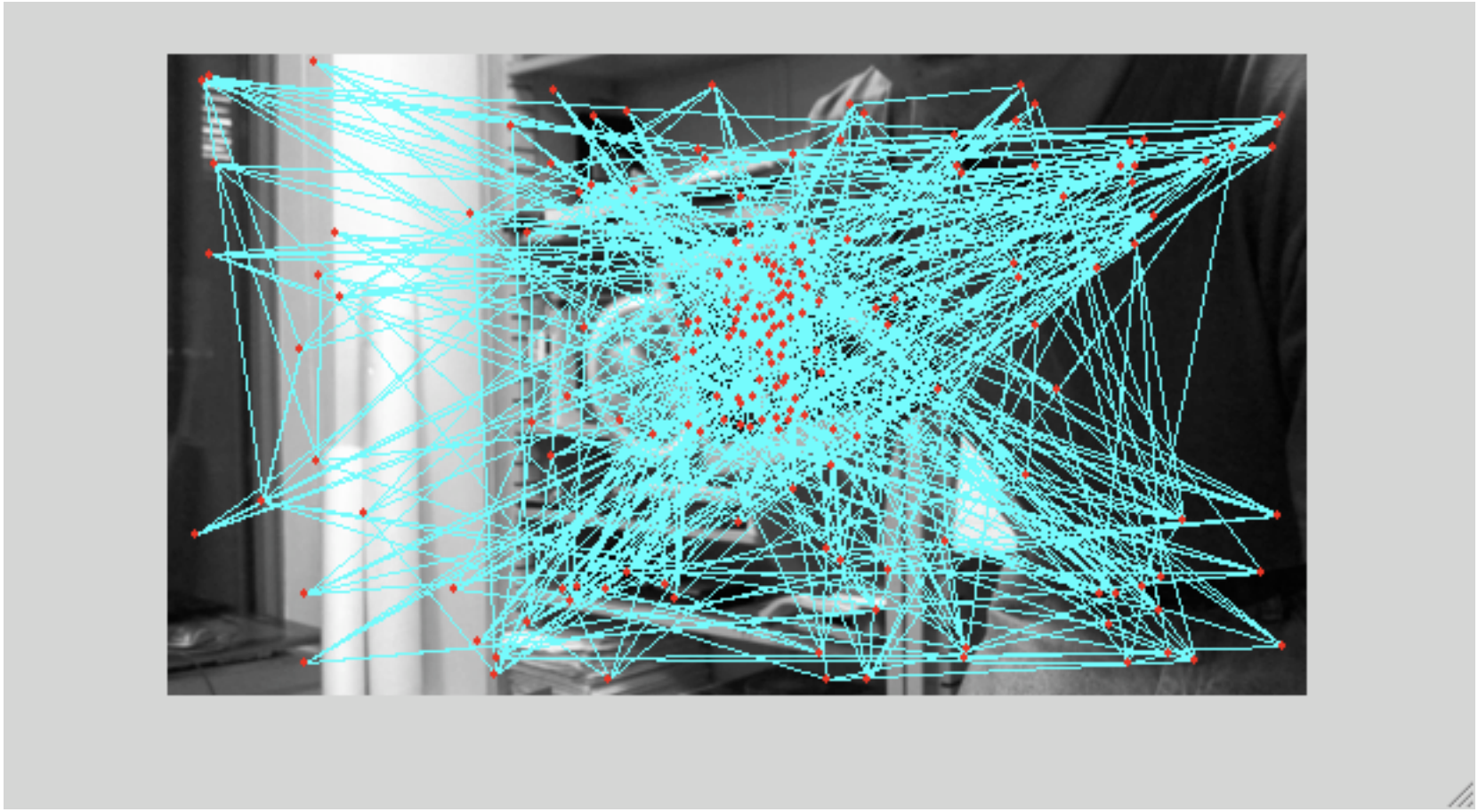
Matching SIFT

Template

Matching SIFT

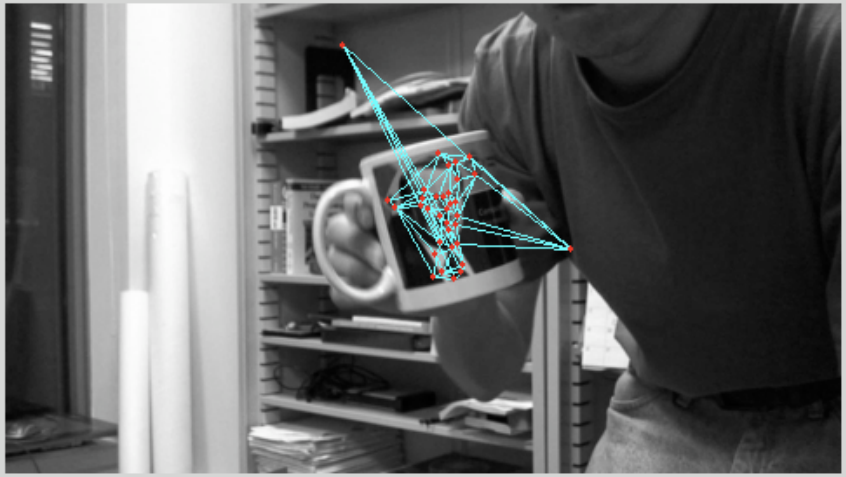
Target image

Matching SIFT Using Nearest Neighbor



Matching result

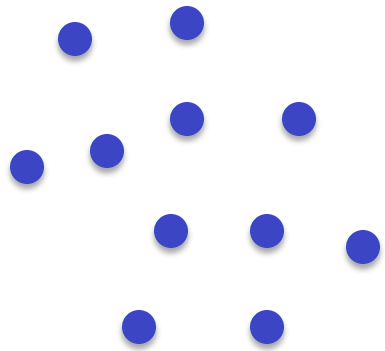
Matching SIFT Using Nearest Neighbor



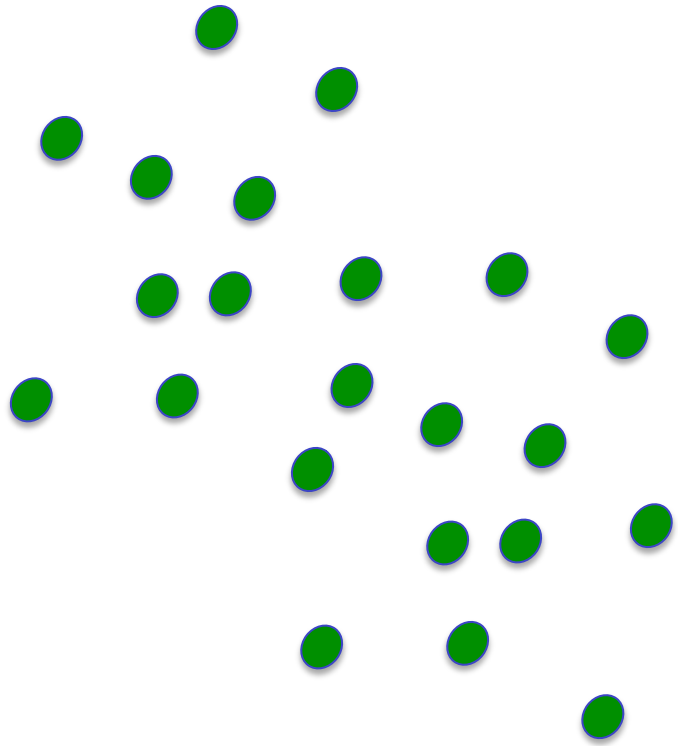
Matching points whose ratio
(best_match_cost / second_best_match_cost) < 0.7

RANSAC

- Generate matching hypothesis



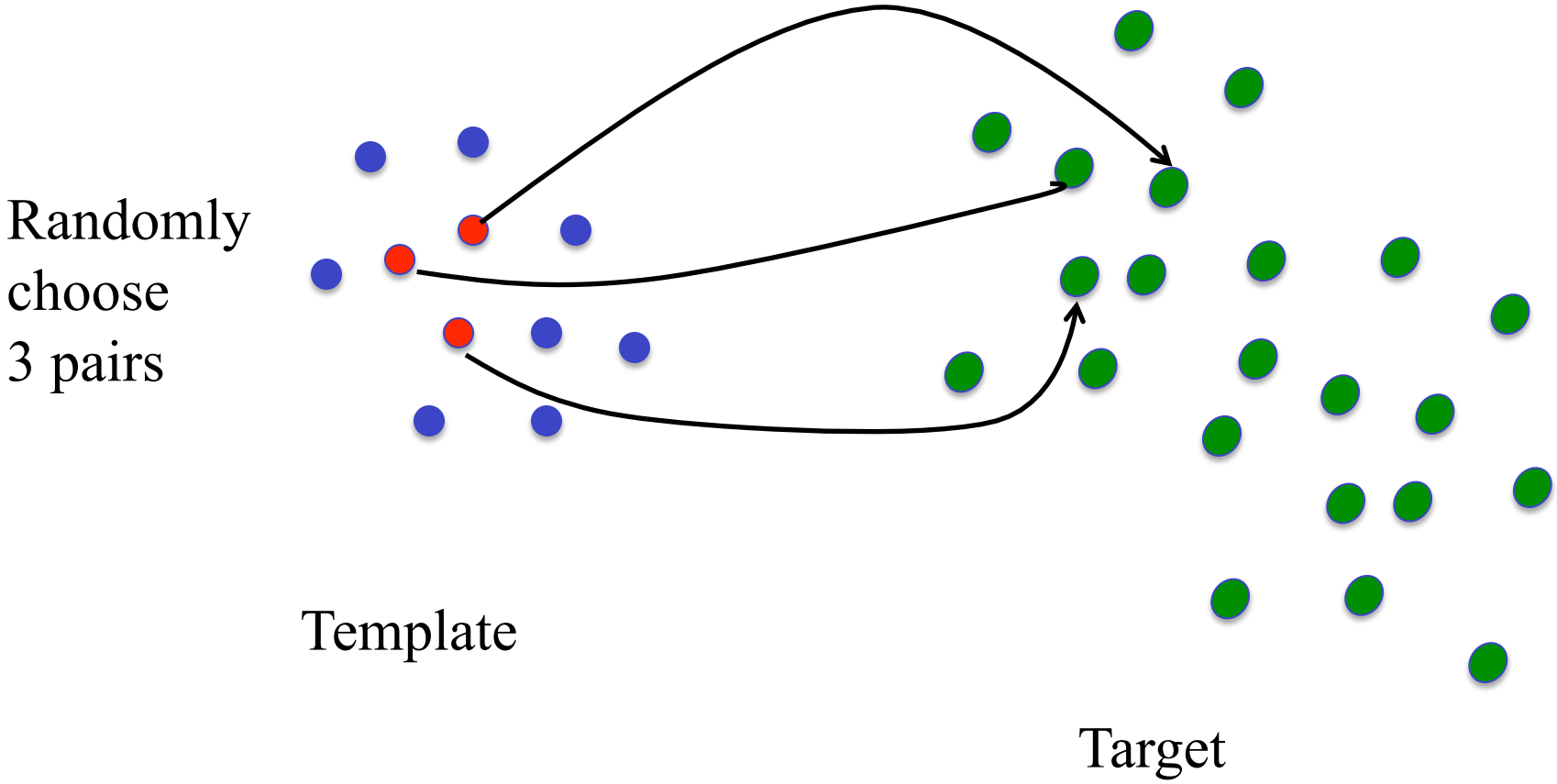
Template



Target

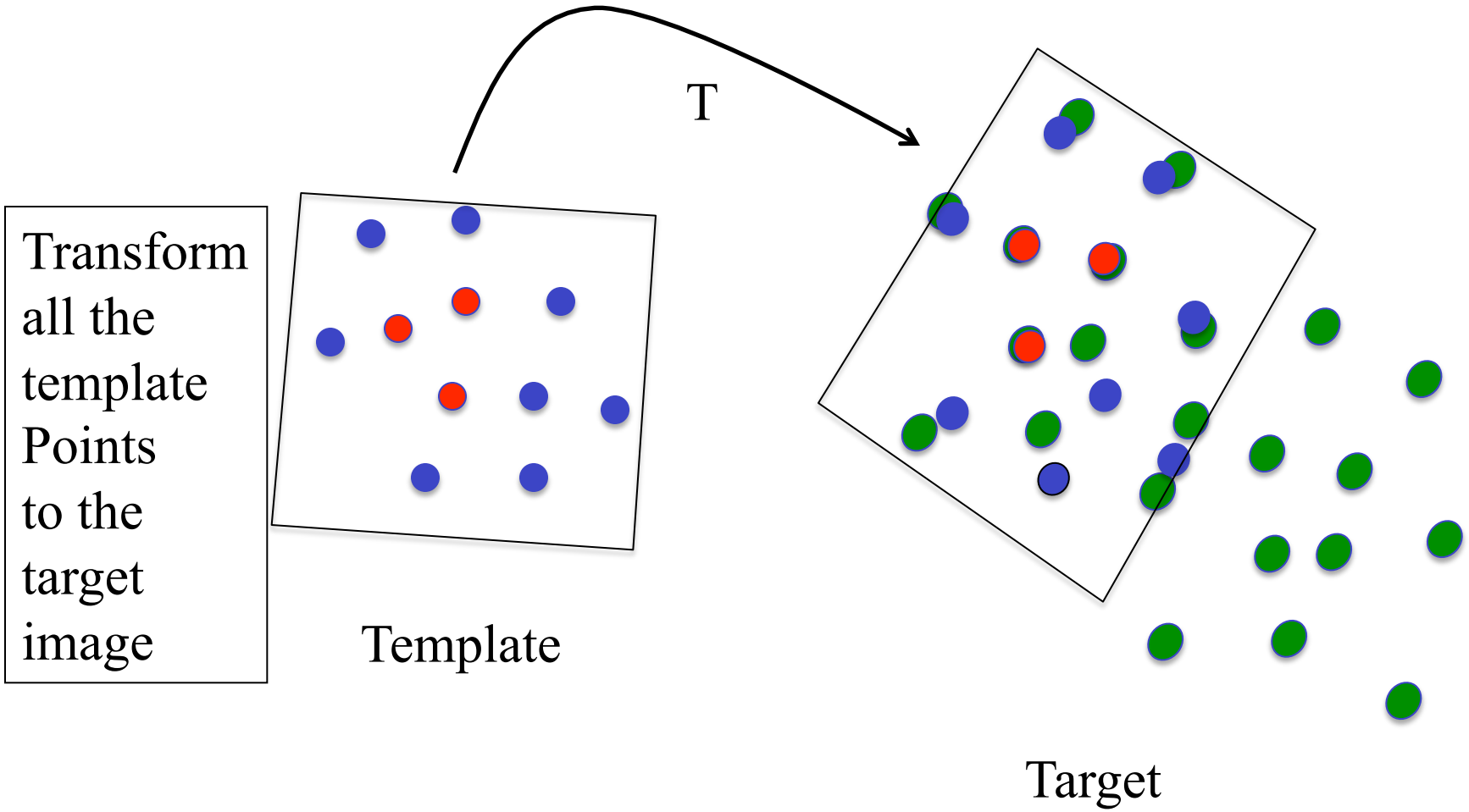
RANSAC

- Generate matching hypothesis



RANSAC

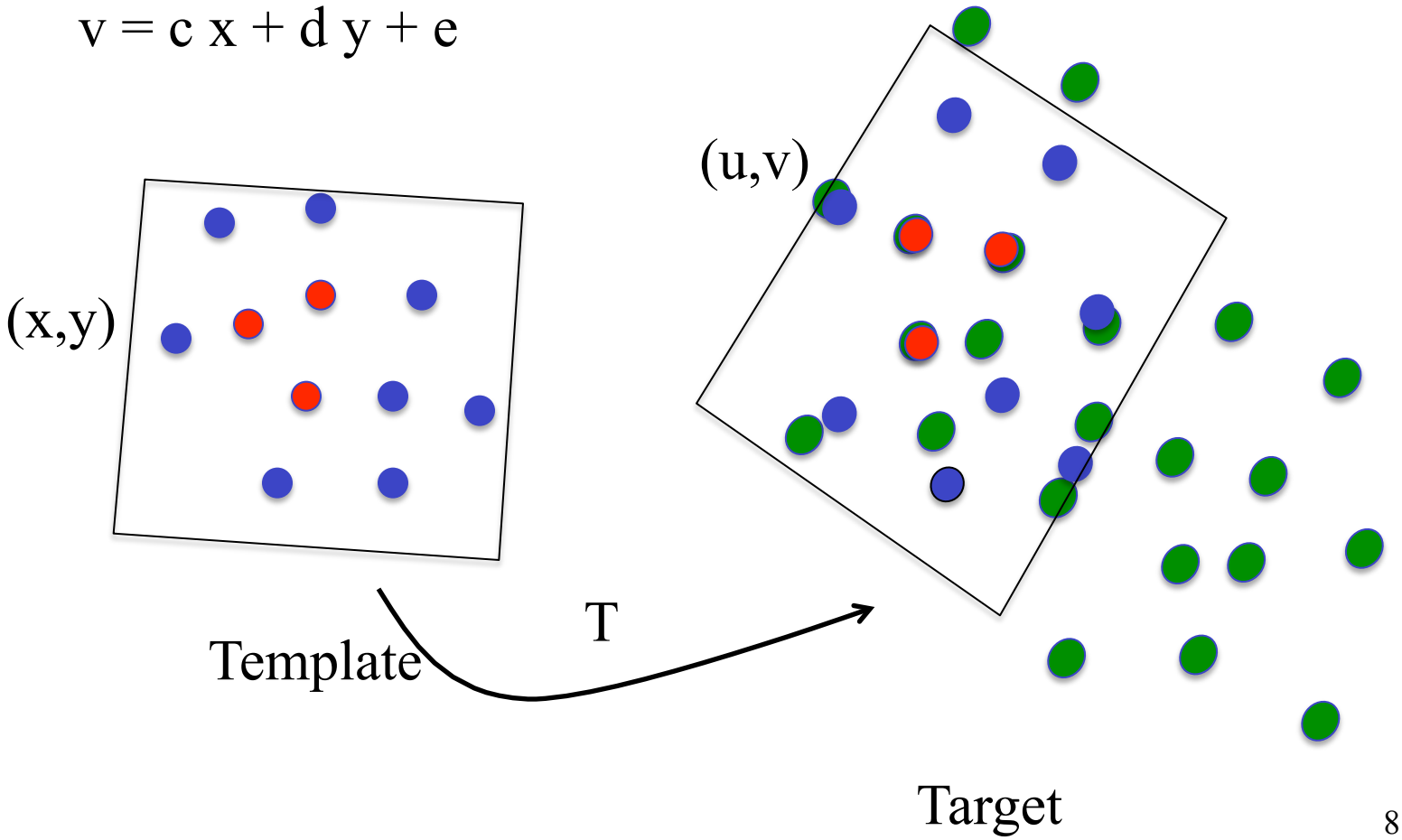
- Generate matching hypothesis



Affine Transformation

Point (x,y) is mapped to (u,v) by the linear function:

$$u = a\,x + b\,y + c$$
$$v = c\,x + d\,y + e$$



Affine Transformation

Point (x,y) is mapped to (u,v) by the linear function:

$$u = a\,x + b\,y + c$$
$$v = c\,x + d\,y + e$$

In matrix format:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ e \end{bmatrix}$$

Affine Transformation

Point (x,y) is mapped to (u,v) by the linear function:

$$u = a\,x + b\,y + c$$
$$v = c\,x + d\,y + e$$

Using homogeneous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ c & d & e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Matlab

```
t = cp2tform(in_points, out_points, 'affine');
```

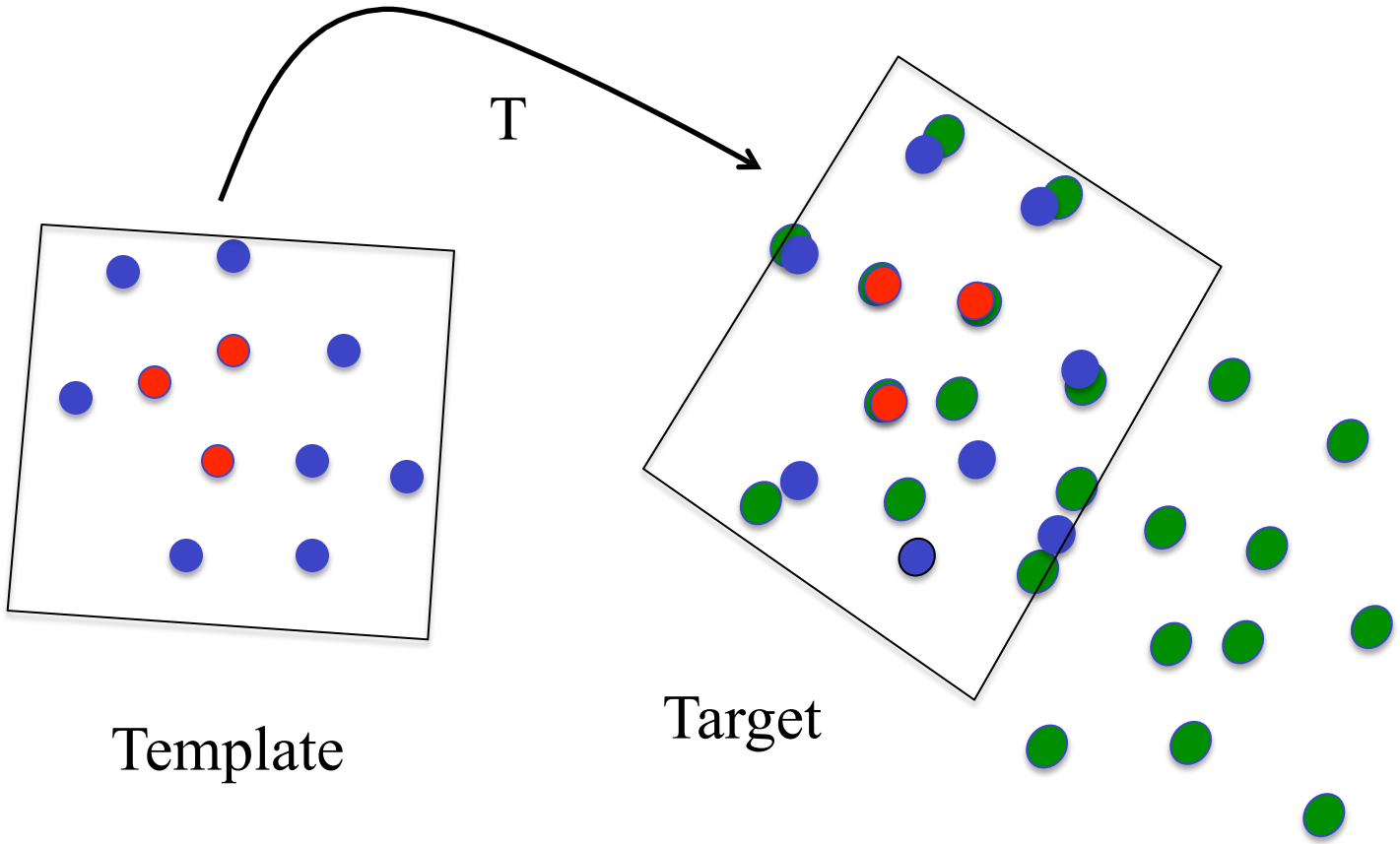
src_points: (x1 y1; x2 y2; x3 y3; …)
target_points: (u1 v1; u2 v2; u3 v3; …)

```
q = [x y 1] * t.tdata.T;  % q = [u ,v, 1]
```

Other transformations: 'similarity', 'projective'

RANSAC

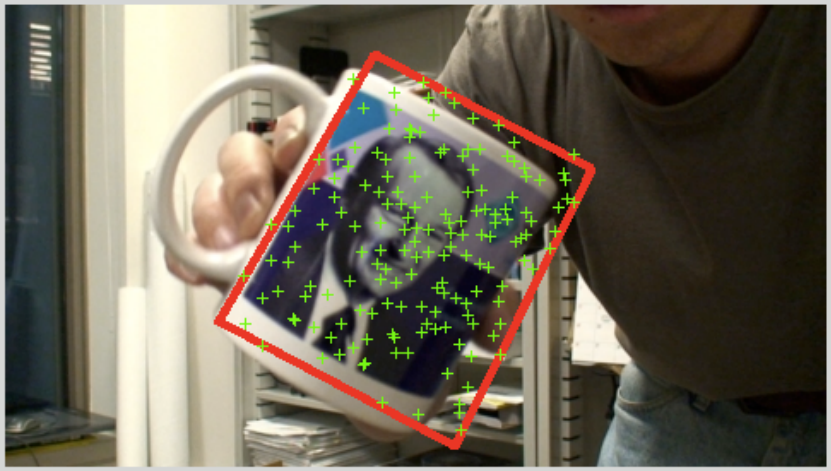
- Validation



Match points to the closes sift target points and compute the overall SIFT feature difference

Demo and Assignment Discussion

Template

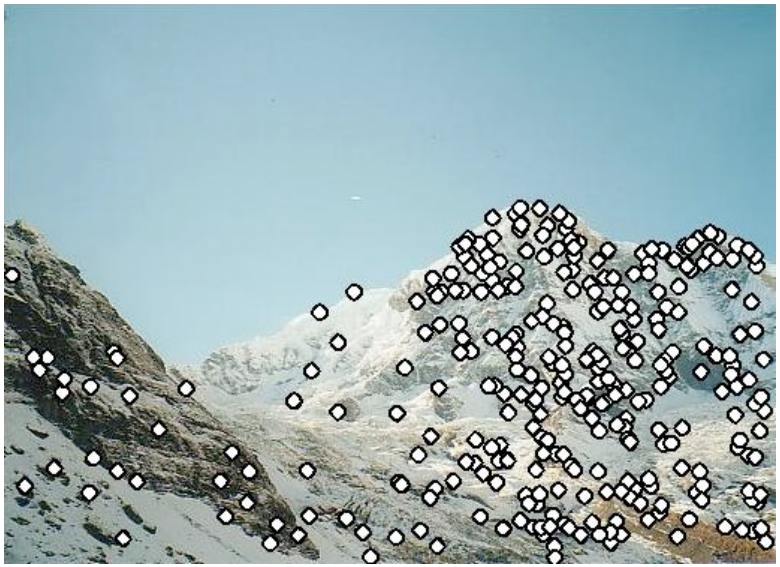


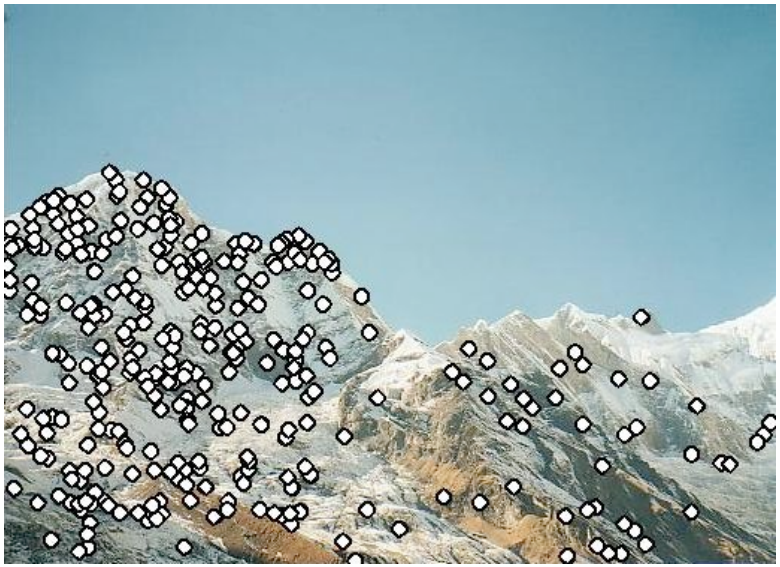
Target

Feature-based alignment outline

Source: L. Lazebnik

Feature-based alignment outline

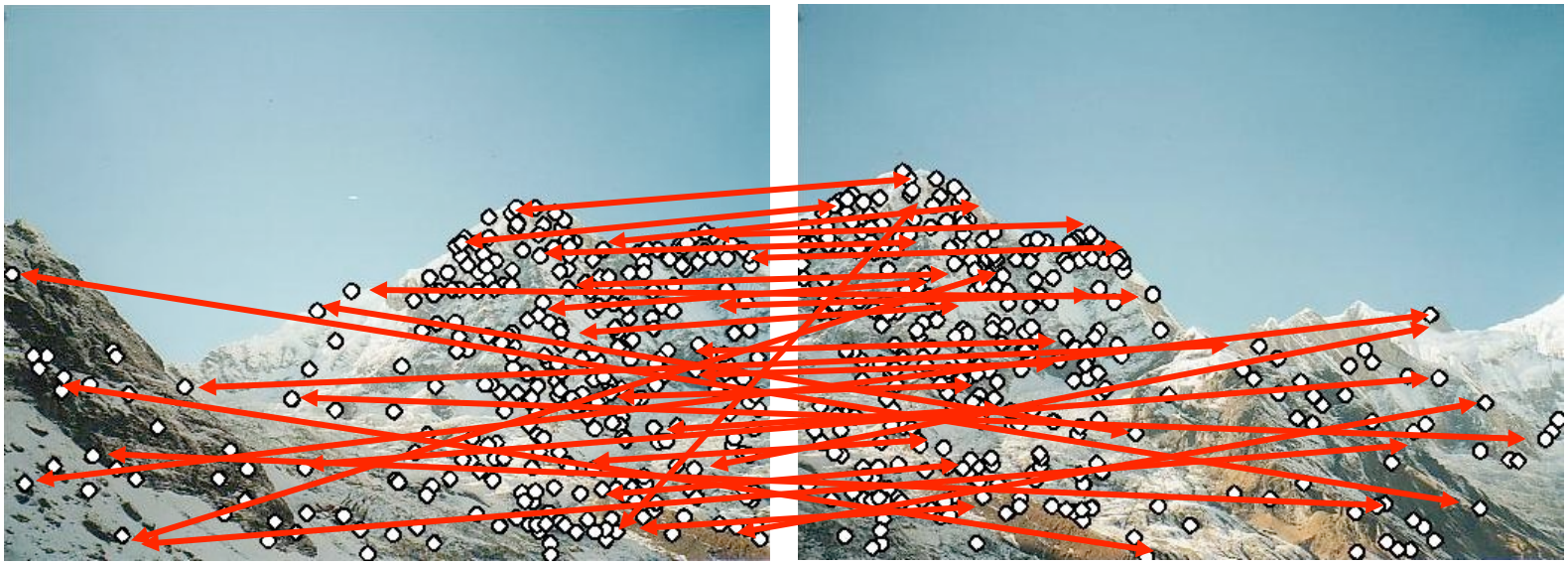




- Extract features

Source: L. Lazebnik

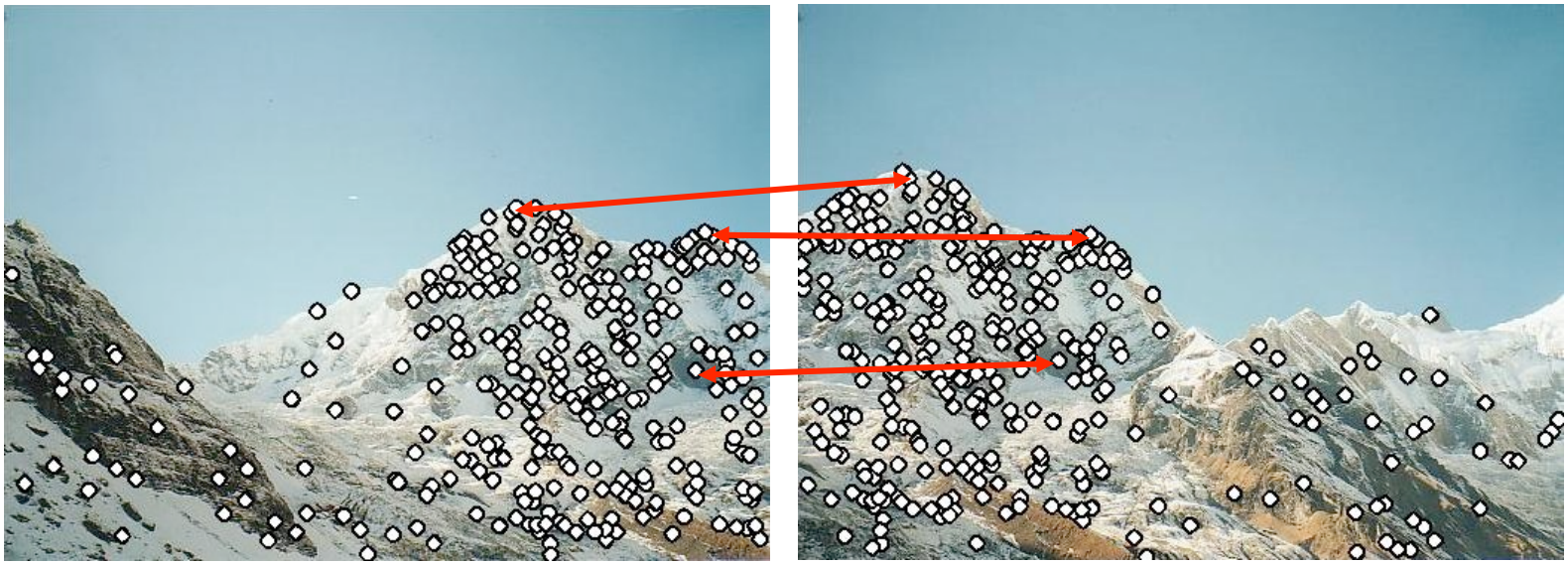
Feature-based alignment outline



- Extract features
- Compute *putative matches*

Source: L. Lazebnik

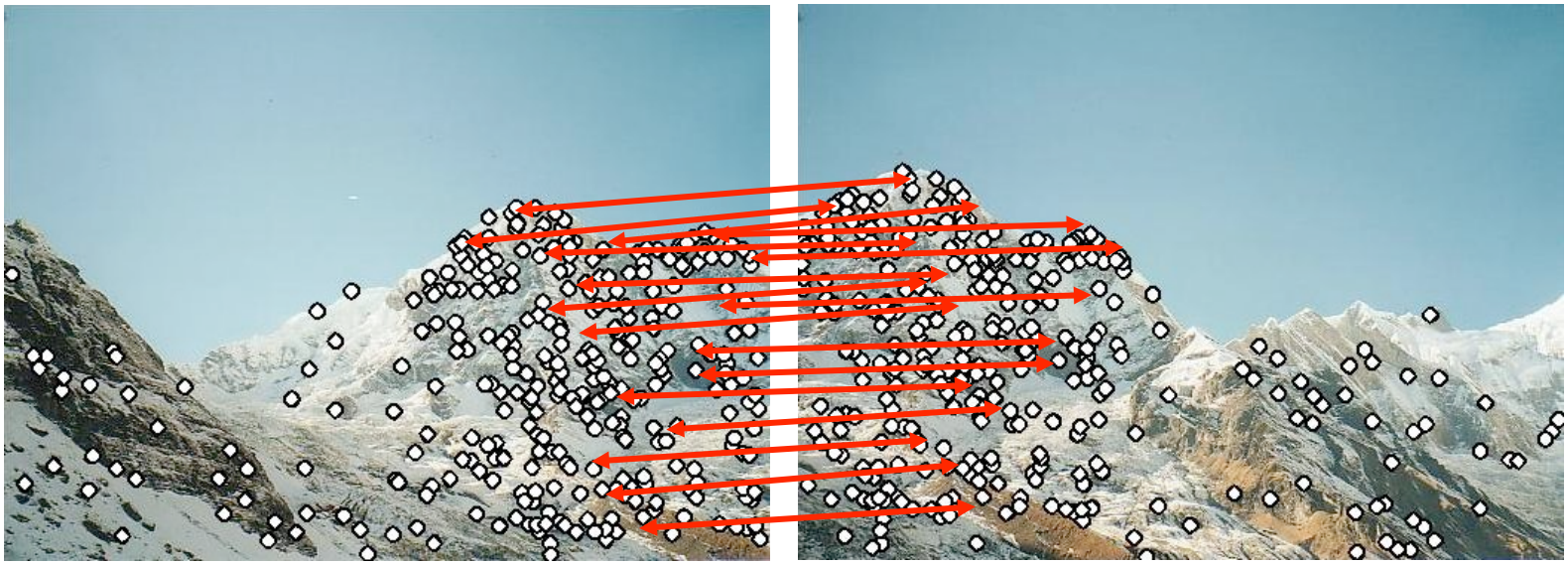
Feature-based alignment outline



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)

Source: L. Lazebnik

Feature-based alignment outline



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

Source: L. Lazebnik

Feature-based alignment outline

- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

Source: L. Lazebnik