

BSB663

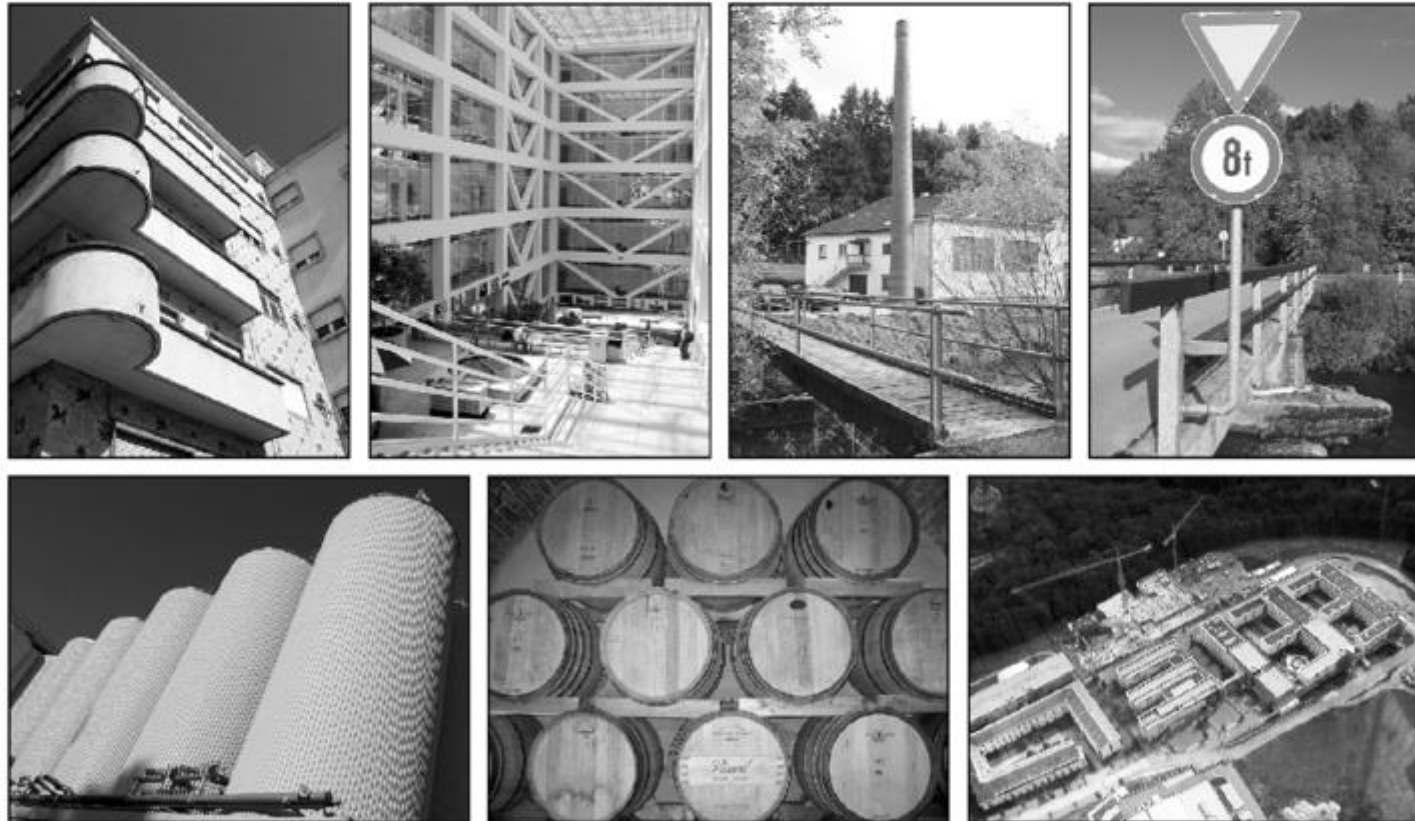
Image Processing

Pinar Duygulu

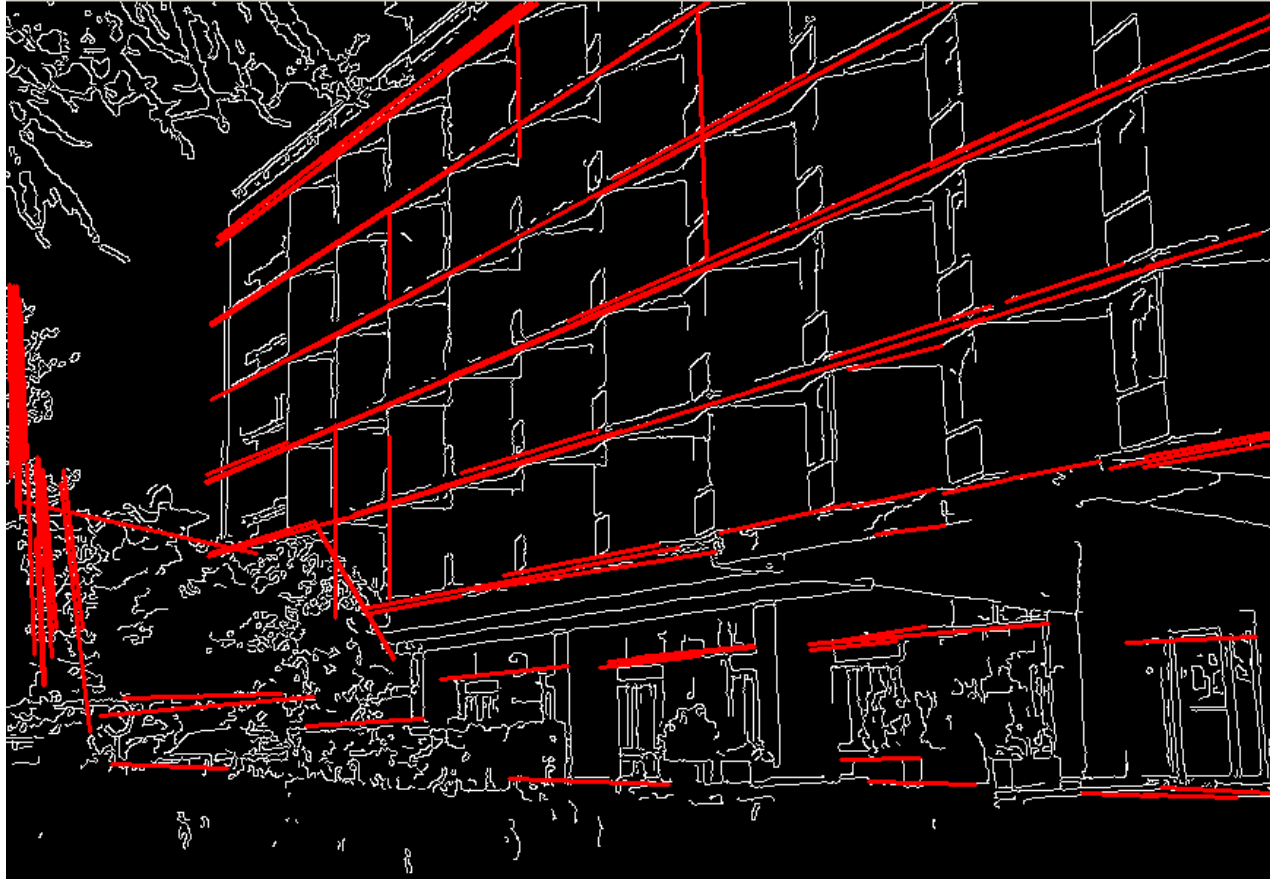
Slides are adapted from
Selim Aksoy

Detecting Lines and Simple Curves

- Many man-made objects exhibits simple geometric forms: lines, circles, ellipses



Fitting



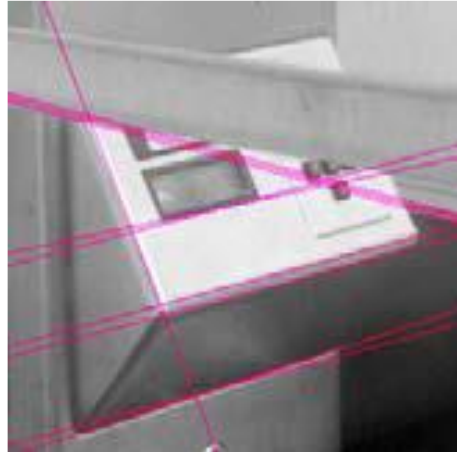
Fitting

- We've learned how to detect edges, corners, blobs. Now what?
- We would like to form a higher-level, more compact representation of the features in the image by grouping multiple features according to a simple model



Fitting

- Choose a parametric model to represent a set of features



simple model: lines



simple model: circles



complicated model: car

Fitting: Issues

Case study: Line detection



- **Noise** in the measured feature locations
- **Extraneous data:** clutter (outliers), multiple lines
- **Missing data:** occlusions

Canny edge detector

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \quad I_y = G_\sigma^y * I$$

2. Compute magnitude of gradient at every pixel

$$M(x, y) = |\nabla I| = \sqrt{I_x^2 + I_y^2}$$

3. Eliminate those pixels that are not local maxima of the magnitude in the direction of the gradient

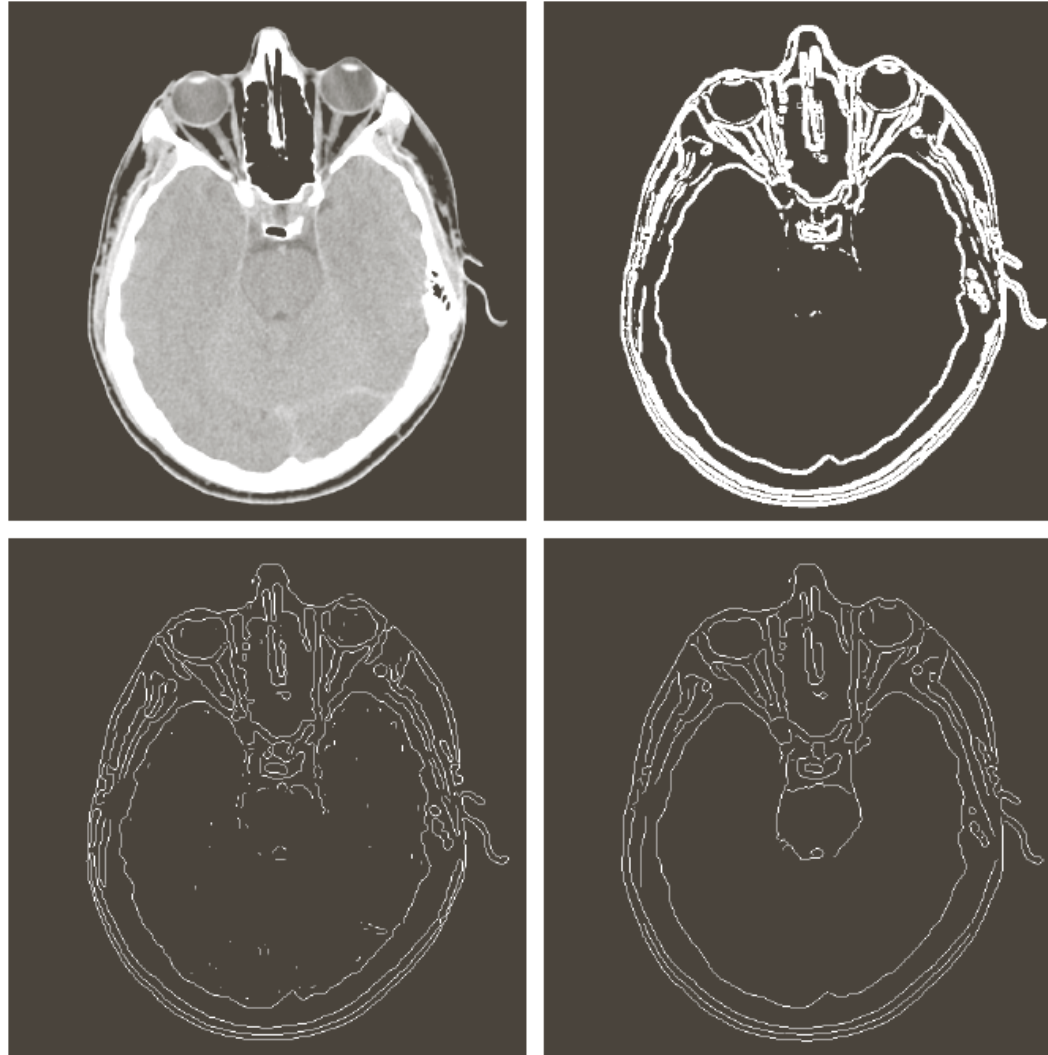
4. Hysteresis Thresholding

- Select the pixels such that $M > T_h$ (high threshold)

Adapted from Martial Hebert, CMU

- Collect the pixels such that $M > T_l$ (low threshold) that are neighbors of already collected edge points

Canny edge detector



a	b
c	d

FIGURE 10.26

(a) Original head CT image of size 512×512 pixels, with intensity values scaled to the range $[0, 1]$.

(b) Thresholded gradient of smoothed image.

(c) Image obtained using the Marr-Hildreth algorithm.

(d) Image obtained using the Canny algorithm.

(Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)

Canny edge detector

- The Canny operator gives single-pixel-wide images with good continuation between adjacent pixels.
- It is the most widely used edge operator today; no one has done better since it came out in the late 80s. Many implementations are available.
- It is very sensitive to its parameters, which need to be adjusted for different application domains.

Edge linking

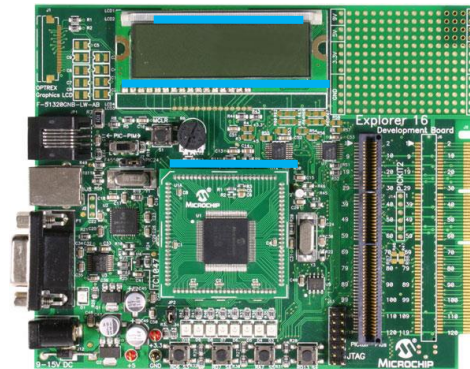
- Hough transform
 - Finding line segments
 - Finding circles
- Model fitting
 - Fitting line segments
 - Fitting ellipses
- Edge tracking

Fitting: main idea

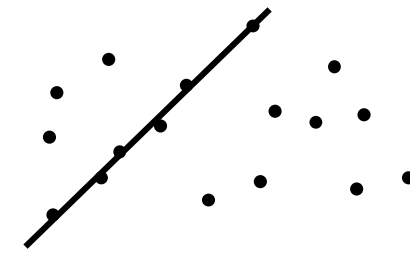
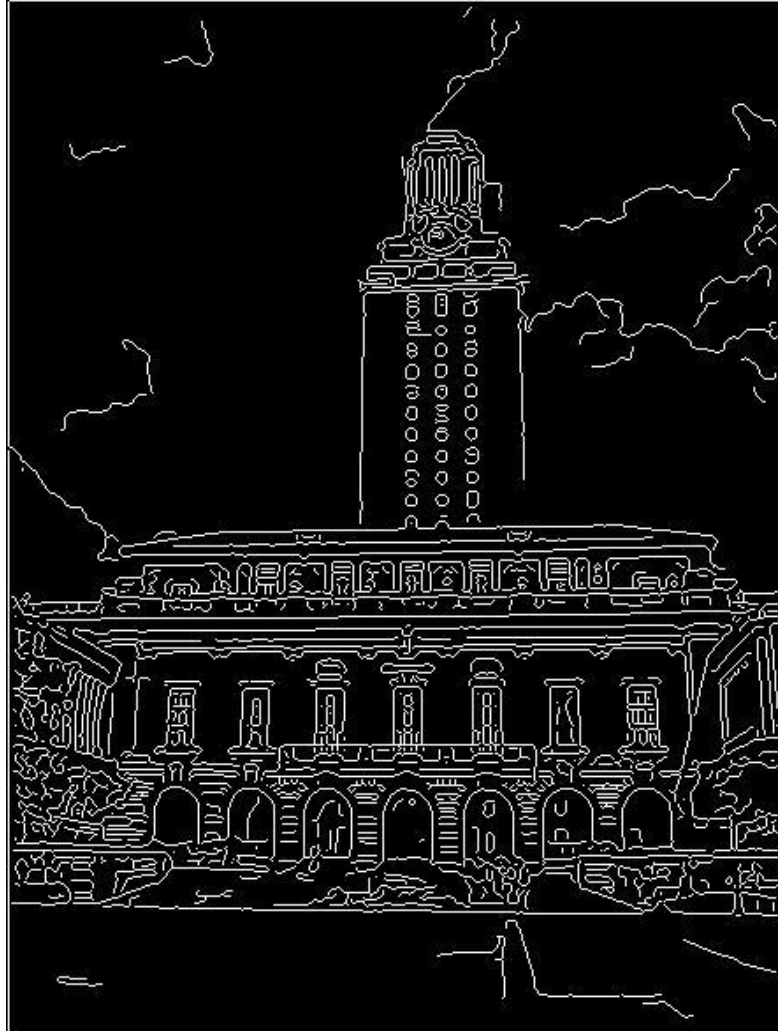
- Choose a parametric model to represent a set of features
- Membership criterion is not local
 - Cannot tell whether a point belongs to a given model just by looking at that point
- Three main questions:
 - What model represents this set of features best?
 - Which of several model instances gets which feature?
 - How many model instances are there?
- Computational complexity is important
 - It is infeasible to examine every possible set of parameters and every possible combination of features

Example: line fitting

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



Difficulty of line fitting



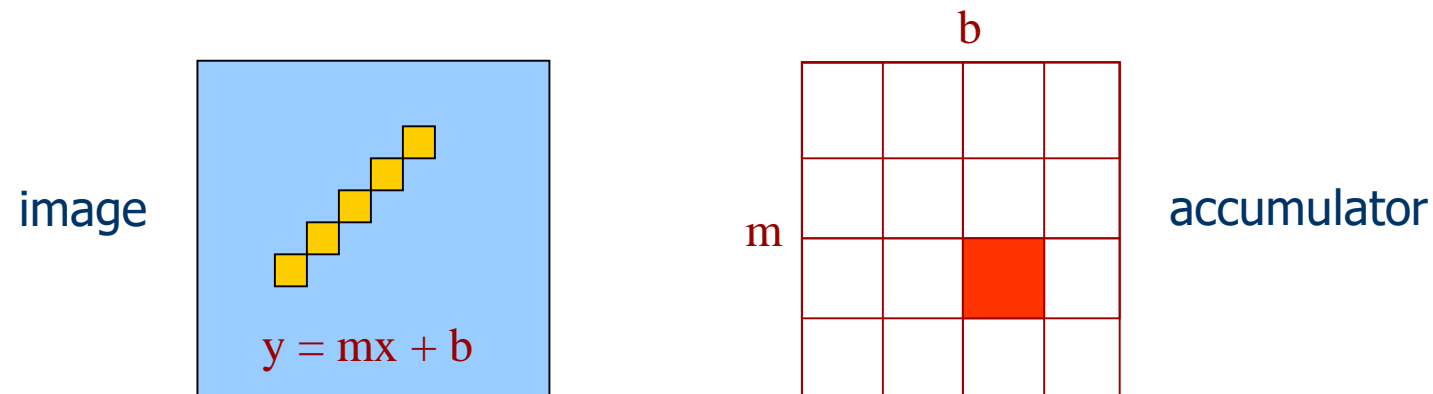
- **Extra** edge points (clutter), multiple models:
 - which points go with which line, if any?
- Only some parts of each line detected, and some parts are **missing**:
 - how to find a line that bridges missing evidence?
- **Noise** in measured edge points, orientations:
 - how to detect true underlying parameters?

Voting

- It is not feasible to check all combinations of features by fitting a model to each possible subset.
- Voting is a general technique where we let each feature vote for all models that are compatible with it.
 - Cycle through features, cast votes for model parameters.
 - Look for model parameters that receive a lot of votes.
- Noise and clutter features will cast votes too, but typically their votes should be inconsistent with the majority of “good” features.

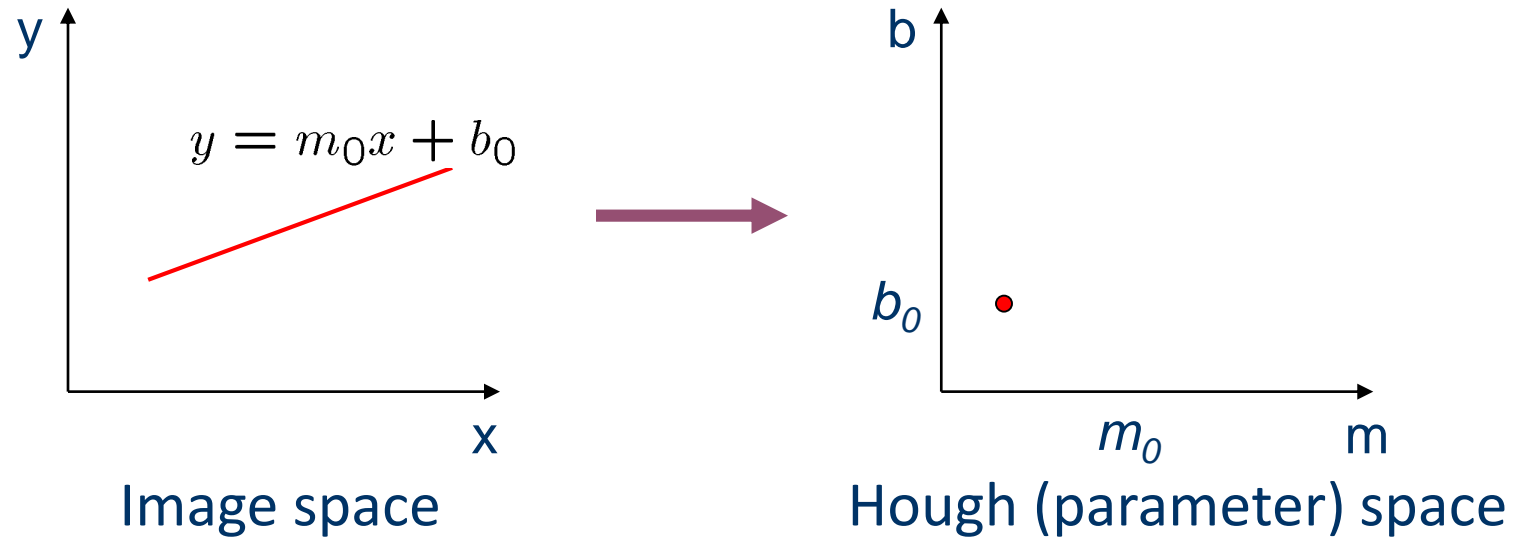
Hough transform

- The Hough transform is a method for detecting lines or curves specified by a parametric function.
- If the parameters are p_1, p_2, \dots, p_n , then the Hough procedure uses an n -dimensional accumulator array in which it accumulates votes for the correct parameters of the lines or curves found on the image.



Adapted from Linda Shapiro, U of Washington

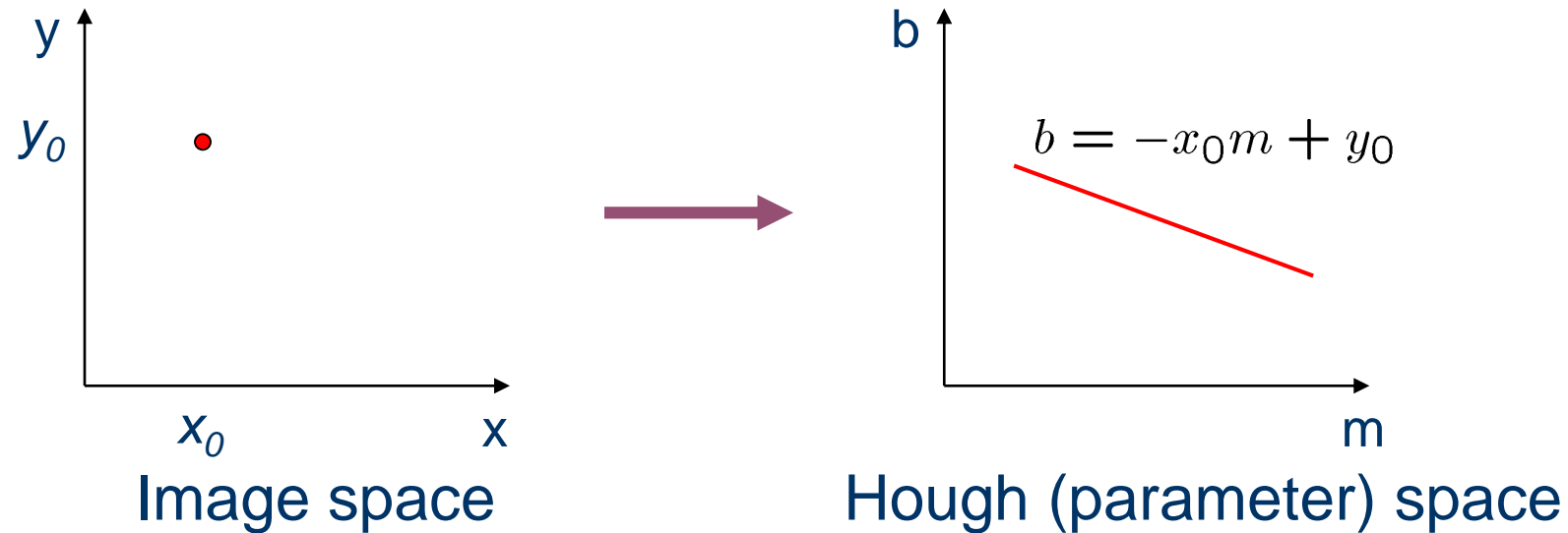
Hough transform: line segments



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 $y = mx + b$

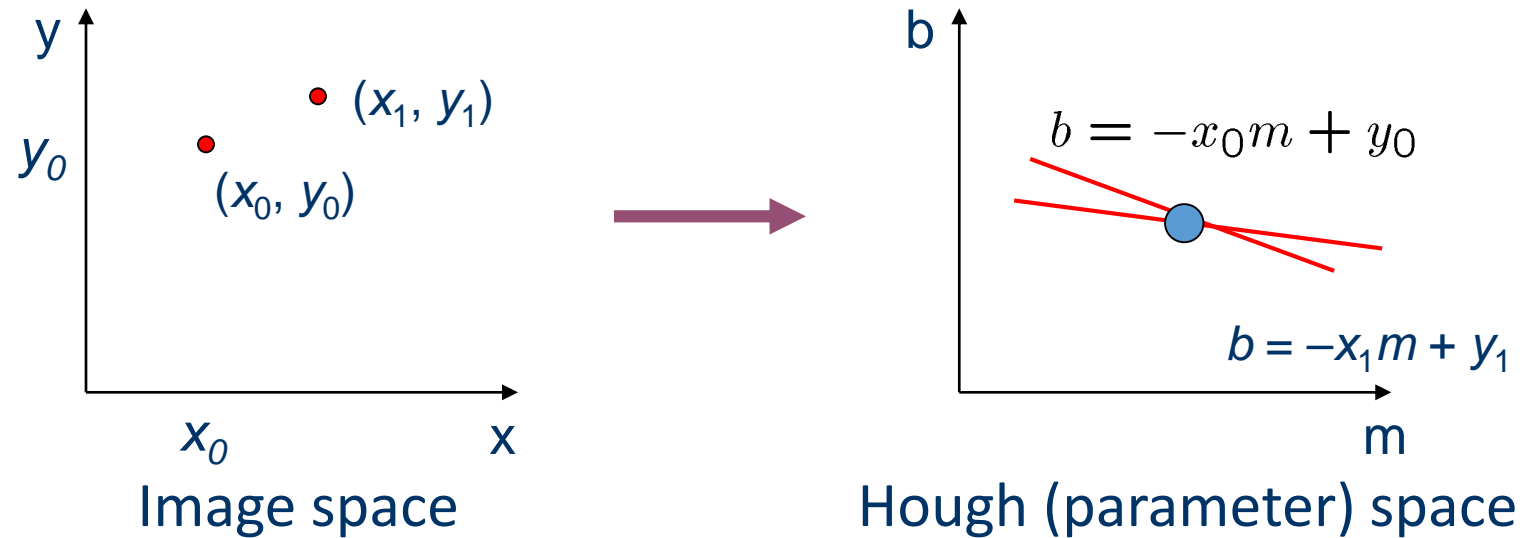
Hough transform: line segments



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

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
 - given a set of points (x,y) , find all (m,b) such that $y = mx + b$
- What does a point (x_0, y_0) in the image space map to?
 - Answer: the solutions of $b = -x_0m + y_0$
 - This is a line in Hough space

Hough transform: line segments

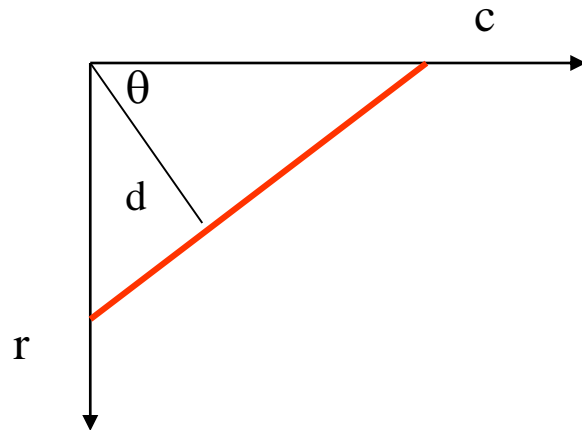


What are the line parameters for the line that contains both (x_0, y_0) and (x_1, y_1) ?

- It is the intersection of the lines $b = -x_0 m + y_0$ and $b = -x_1 m + y_1$

Hough transform: line segments

- $y = mx + b$ is not suitable (why?)
- The equation generally used is:
$$d = r \sin(\theta) + c \cos(\theta).$$



d is the distance from the line to origin.

θ is the angle the perpendicular makes with the column axis.

Hough transform: line segments

Accumulate the straight line segments in gray-tone image S to accumulator A .

$S[R, C]$ is the input gray-tone image.

NLINES is the number of rows in the image.

NPIXELS is the number of pixels per row.

$A[DQ, THETAQ]$ is the accumulator array.

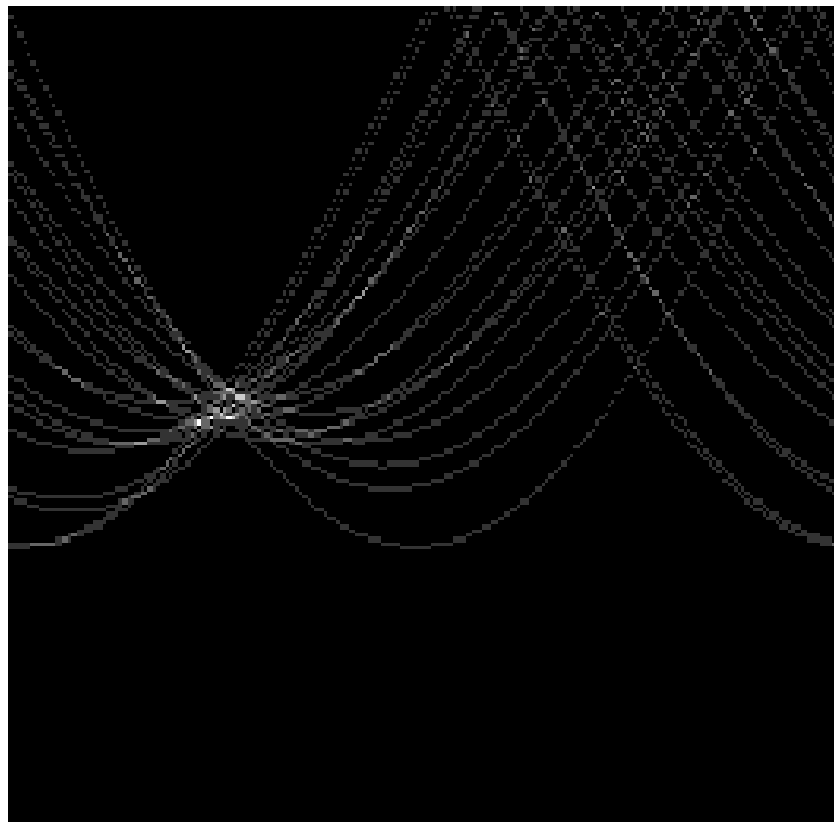
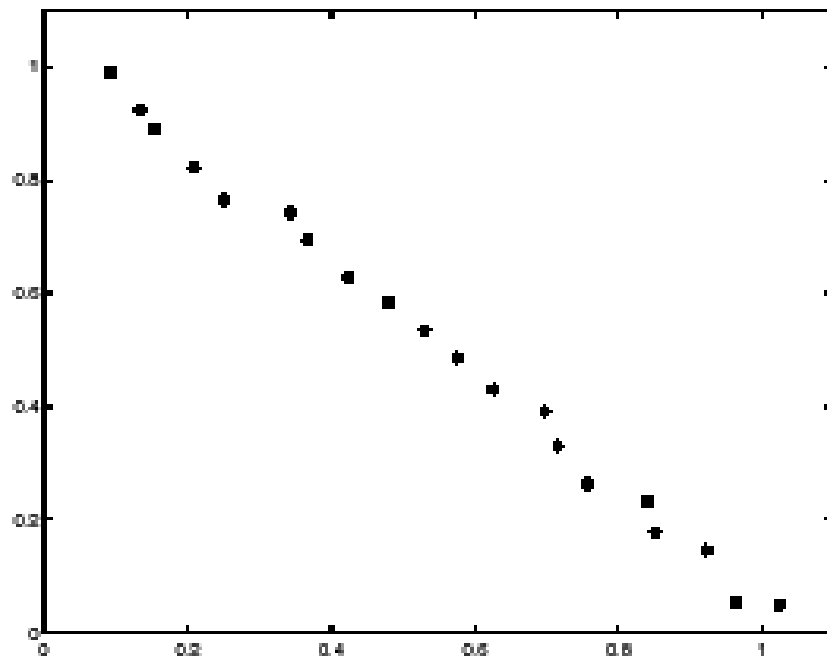
DQ is the quantized distance from a line to the origin.

THETAQ is the quantized angle of the normal to the line.

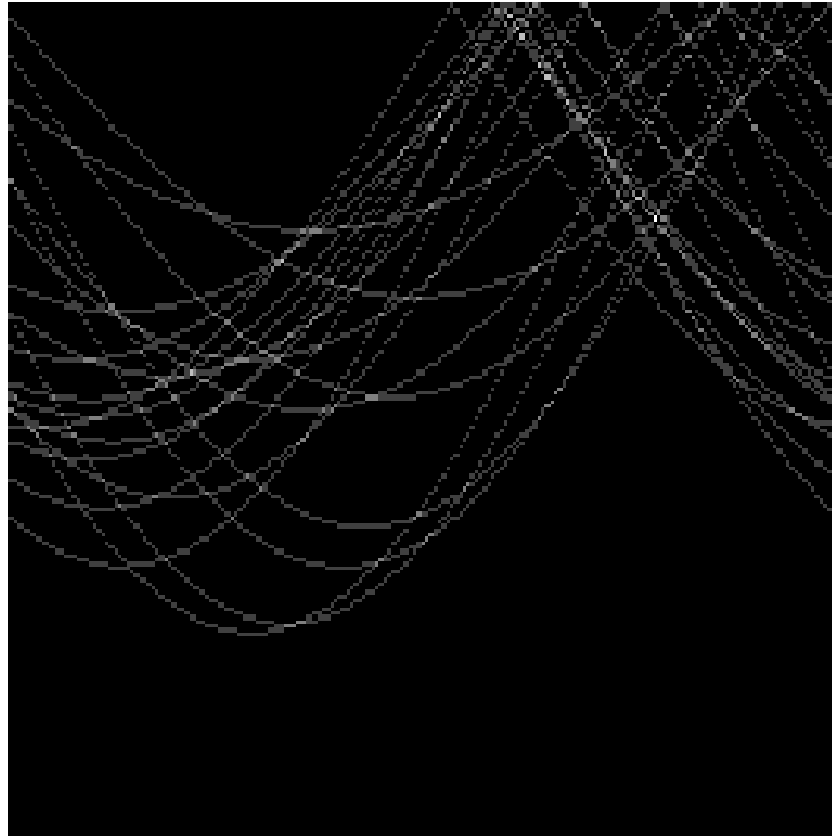
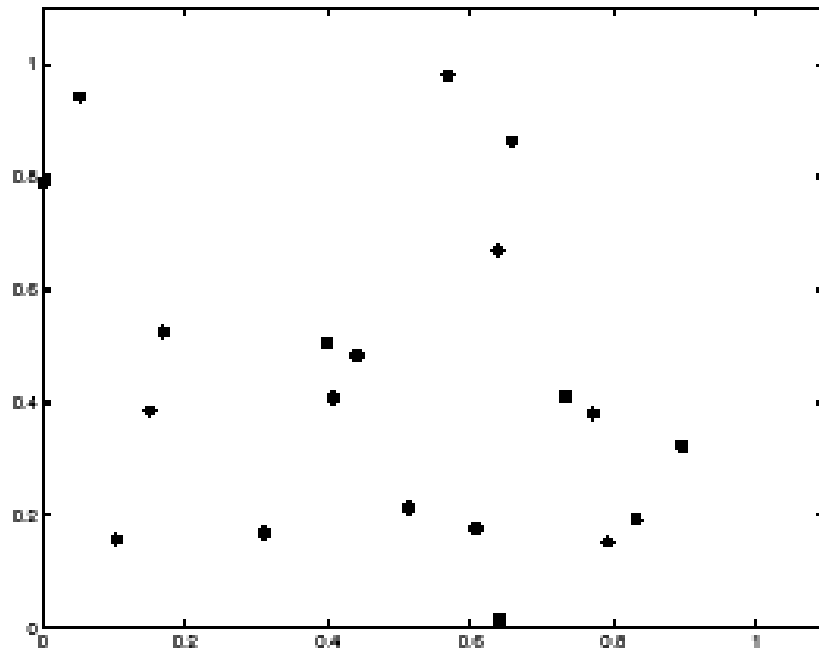
```
procedure accumulate_lines(S,A);
{
  A := 0;
  PTLIST := NIL;
  for R := 1 to NLINES
    for C := 1 to NPIXELS
      {
        DR := row_gradient(S,R,C);
        DC := col_gradient(S,R,C);
        GMAG := gradient(DR,DC);
        if GMAG > gradient_threshold
          {
            THETA := atan2(DR,DC);
            THETAQ := quantize_angle(THETA);
            D := abs(C*cos(THETAQ) - R*sin(THETAQ));
            DQ := quantize_distance(D);
            A[DQ,THETAQ] := A[DQ,THETAQ]+GMAG;
            PTLIST(DQ,THETAQ) := append(PTLIST(DQ,THETAQ),[R,C])
          }
      }
}
```

Adapted from Shapiro and Stockman

Hough transform: line segments



Hough transform: line segments



Hough transform: line segments

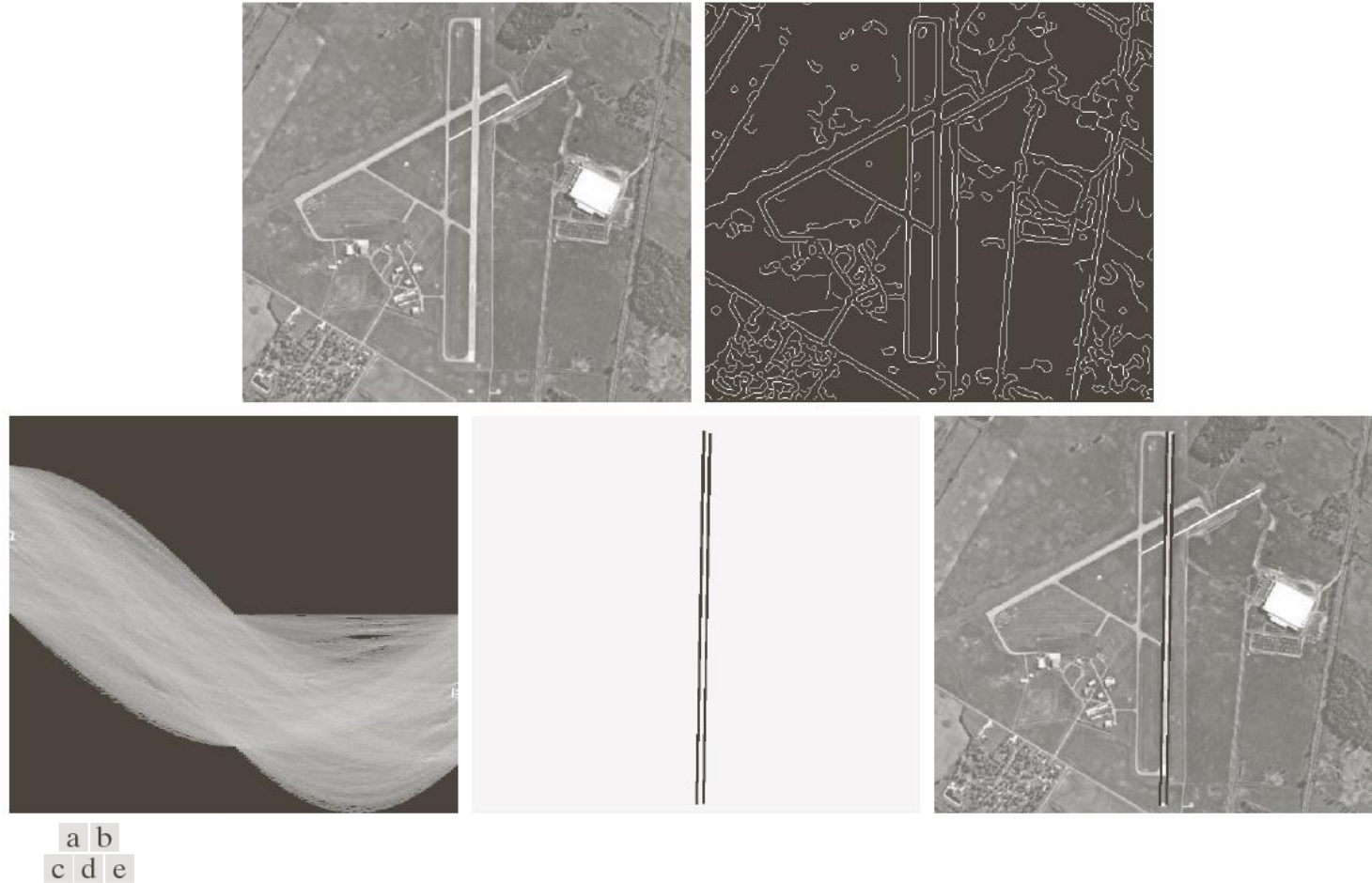
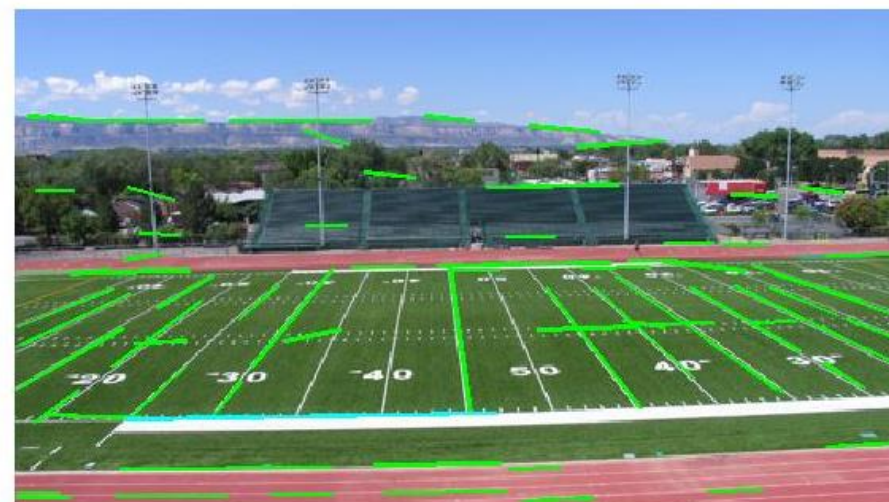
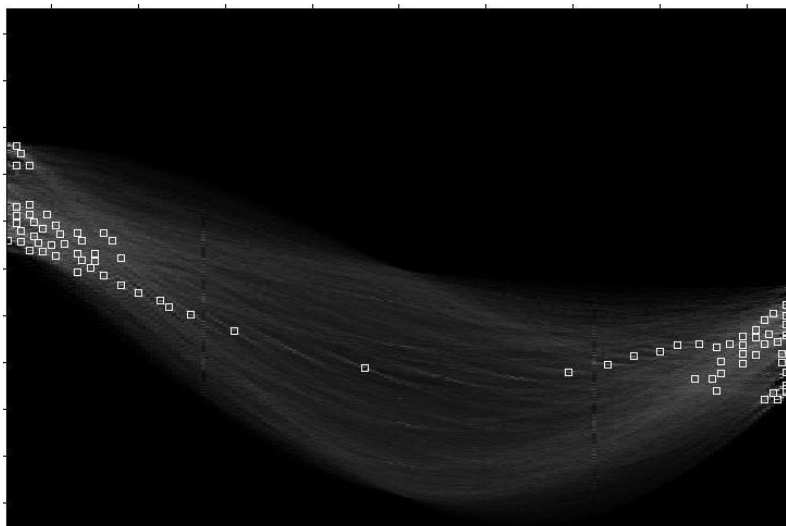


FIGURE 10.34 (a) A 502×564 aerial image of an airport. (b) Edge image obtained using Canny's algorithm. (c) Hough parameter space (the boxes highlight the points associated with long vertical lines). (d) Lines in the image plane corresponding to the points highlighted by the boxes. (e) Lines superimposed on the original image.

Hough transform: line segments

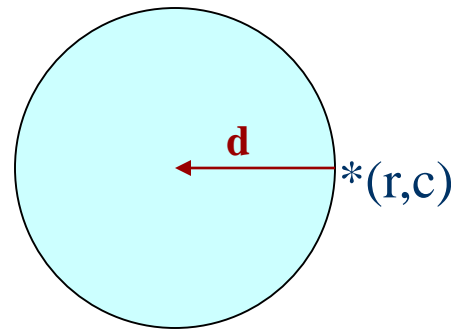


Adapted from Kristen Grauman

Hough transform: circles

- Main idea: The gradient vector at an edge pixel points the center of the circle.
- Circle equations:
 - $r = r_0 + d \sin(\theta)$
 - $c = c_0 + d \cos(\theta)$

r_0, c_0, d are parameters



Hough transform: circles

Accumulate the circles in gray-tone image S to accumulator A .

$S[R, C]$ is the input gray-tone image.

NLINES is the number of rows in the image.

NPIXELS is the number of pixels per row.

$A[R, C, RAD]$ is the accumulator array.

R is the row index of the circle center.

C is the column index of the circle center.

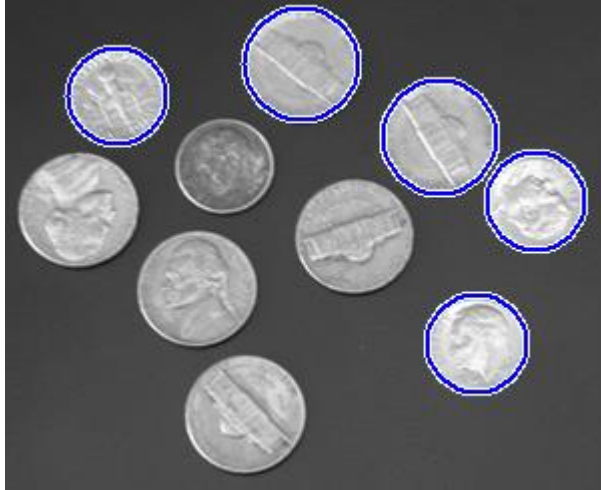
RAD is the radius of the circle.

```
procedure accumulate_circles(S,A);
{
  A := 0;
  PTLIST := 0;
  for R := 1 to NLINES
    for C := 1 to NPIXELS
      for each possible value RAD of radius
        {
          THETA := compute_theta(S,R,C,RAD);
          R0 := R - RAD*cos(THETA);
          C0 := C - RAD*sin(THETA);
          A[R0,C0,RAD] := A[R0,C0,RAD]+1;
          PTLIST(R0,C0,RAD) := append(PTLIST(R0,C0,RAD),[R,C])
        }
  }
```

Adapted from Shapiro and Stockman

Algorithm 9: Hough Transform for Finding Circles

Hough transform: circles



<https://www.mathworks.com/help/images/ref/imfindcircles.html>

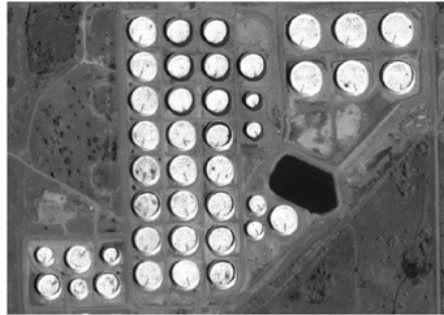


http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html

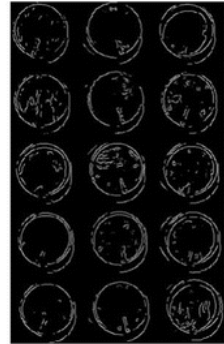


<http://shreshai.blogspot.com.tr/2015/01/matlab-tutorial-finding-center-pivot.html>

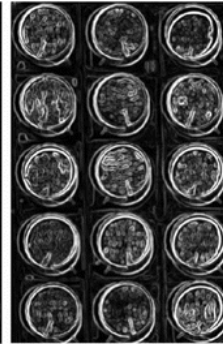
Hough transform: circles



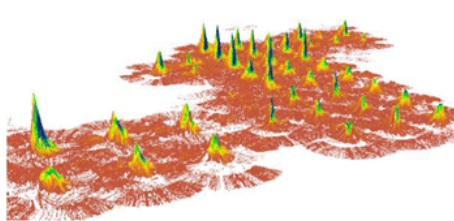
a The original satellite image



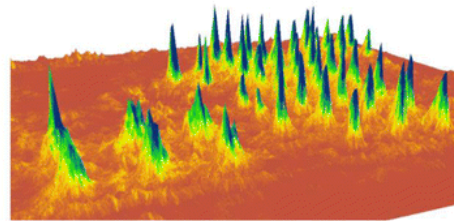
b The Canny edge map



c The gradient image



d The edge based voting result



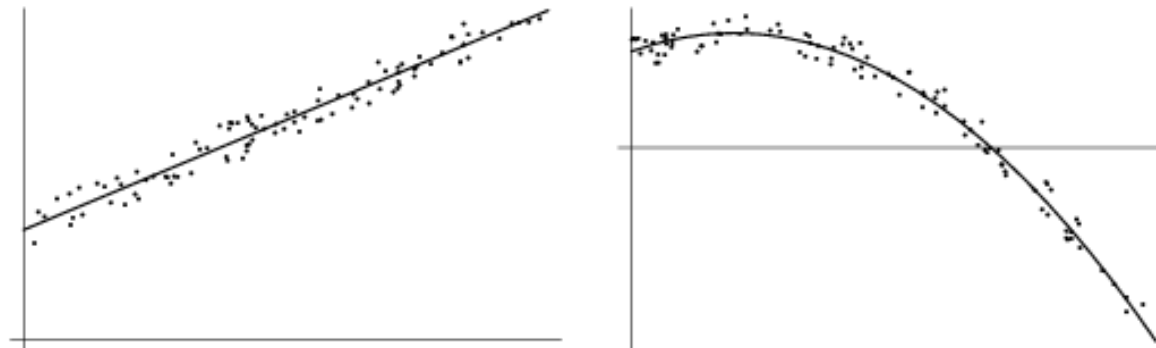
e The gradient based voting result



Zhao et al., "Oil Tanks Extraction from High Resolution Imagery Using a Directional and Weighted Hough Voting Method", Journal of the Indian Society of Remote Sensing, September 2015

Model fitting

- Mathematical models that fit data not only reveal important structure in the data, but also can provide efficient representations for further analysis.
- Mathematical models exist for lines, circles, cylinders, and many other shapes.
- We can use the [method of least squares](#) for determining the parameters of the best mathematical model fitting the observed data.



Model fitting: line segments



Model fitting: line segments

- Given a set of observed points $\{(x_i, y_i), i = 1, \dots, n\}$.
- A straight line can be modeled as a function with two parameters:

$$y = ax + b.$$

- To measure how well a model fits a set of n observations can be computed using the *least-squares error criteria*:

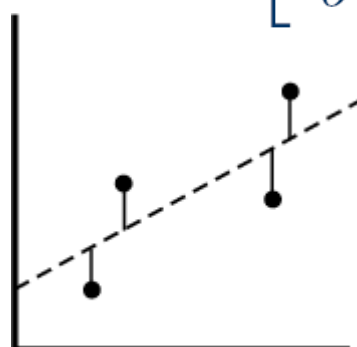
$$LSE = \sum_{i=1}^n (ax_i + b - y_i)^2$$

where $ax_i + b - y_i$ is the algebraic distance.

- The best model is the model with the parameters minimizing this criteria.

Model fitting: line segments

- For the model $y = ax + b$, the parameters that minimize *LSE* can be found by taking partial derivatives and solving for the unknowns.
- The parameters of the best line are:


$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n 1 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n y_i \end{bmatrix}.$$

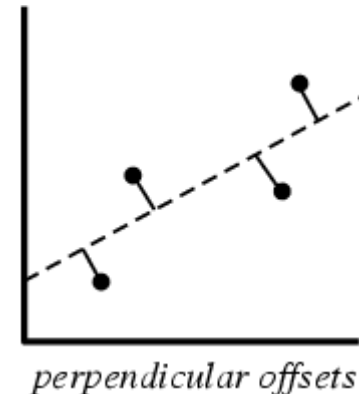
vertical offsets

Model fitting: line segments

- If we use the geometric distance where $ax + by + c = 0$ and $a^2 + b^2 = 1$, the solution for $[a \ b]^T$ is the eigenvector corresponding to the smallest eigenvalue of

$$\begin{bmatrix} \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i\right)^2 & \sum_{i=1}^n x_i y_i - \left(\sum_{i=1}^n x_i\right) \left(\sum_{i=1}^n y_i\right) \\ \sum_{i=1}^n x_i y_i - \left(\sum_{i=1}^n x_i\right) \left(\sum_{i=1}^n y_i\right) & \sum_{i=1}^n y_i^2 - \left(\sum_{i=1}^n y_i\right)^2 \end{bmatrix}$$

and $c = -a \sum_{i=1}^n x_i - b \sum_{i=1}^n y_i$.



Model fitting: line segments

- Problems in fitting:
 - Outliers
 - Error definition (algebraic vs. geometric distance)
 - Statistical interpretation of the error (hypothesis testing)
 - Nonlinear optimization
 - High dimensionality (of the data and/or the number of model parameters)
 - Additional fit constraints

Model fitting: ellipses

- Fitting a general conic represented by a second-order polynomial

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

can be approached by minimizing the sum of squared algebraic distances.

- See Fitzgibbon *et al.* (PAMI 1999) for an algorithm that constrains the parameters so that the conic representation is forced to be an ellipse.

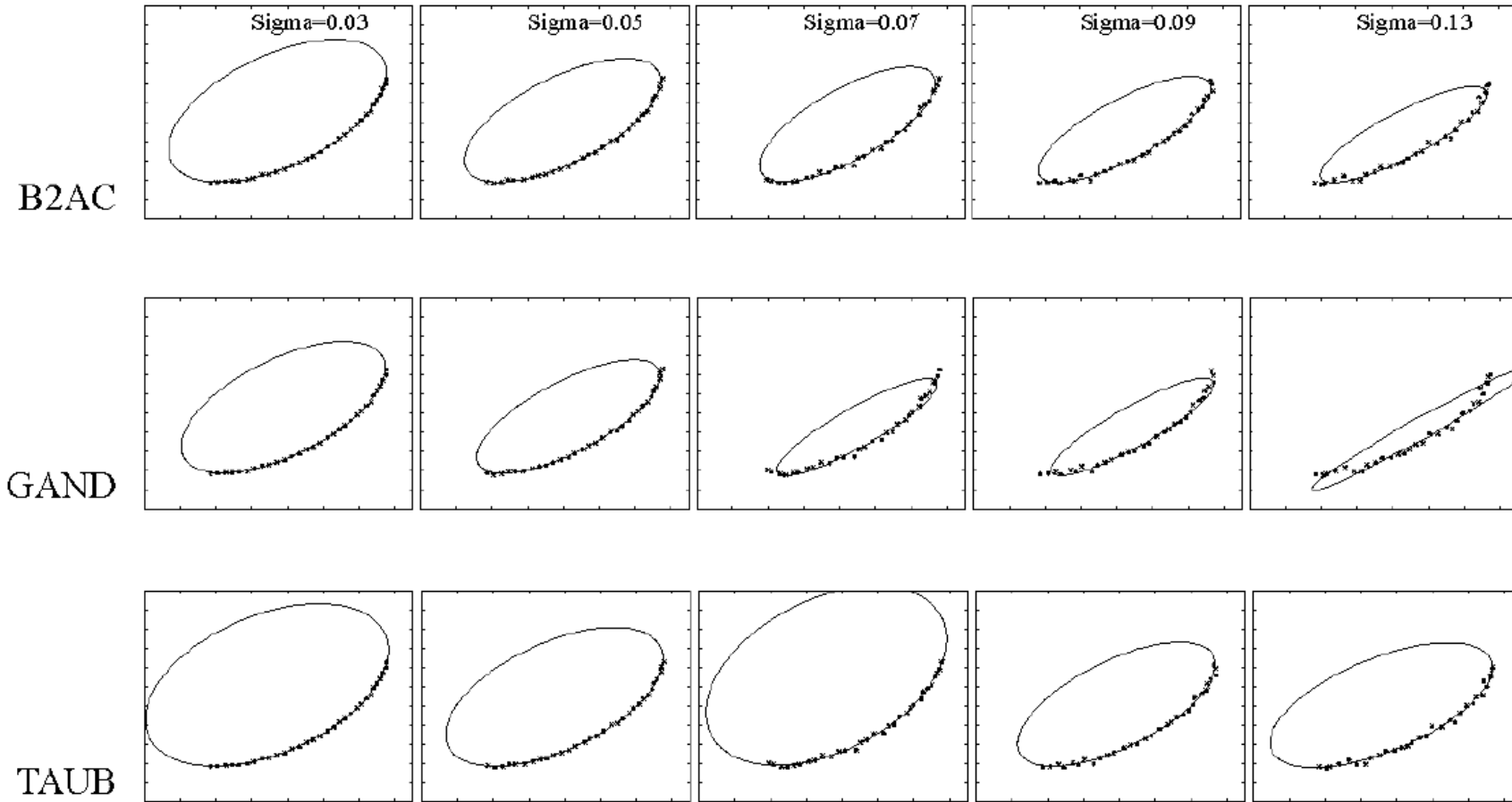
Model fitting: ellipses

```
% x,y are vectors of coordinates
function a=fit_ellipse(x,y)
% Build design matrix
  D = [ x.*x x.*y y.*y x y ones(size(x)) ];
% Build scatter matrix
  S = D'*D;
% Build 6x6 constraint matrix
  C(6,6)=0; C(1,3)=-2; C(2,2)=1; C(3,1)=-2;
% Solve generalised eigensystem
  [gevec, geval] = eig(S,C);
% Find the only negative eigenvalue
  [NegR, NegC] = find(geval<0 & ~isinf(geval));
% Get fitted parameters
  a = gevec(:,NegC);
```

Simple six-line Matlab implementation of the ellipse fitting method.

Adapted from Andrew Fitzgibbon, PAMI 1999

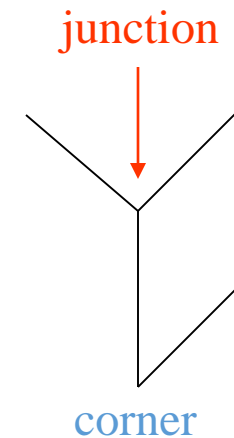
Model fitting: ellipses



Fits to arc of ellipse with increasing noise level.

Edge tracking

- Mask-based approach uses masks to identify the following events:
 - start of a new segment,
 - interior point continuing a segment,
 - end of a segment,
 - junction between multiple segments,
 - corner that breaks a segment into two.

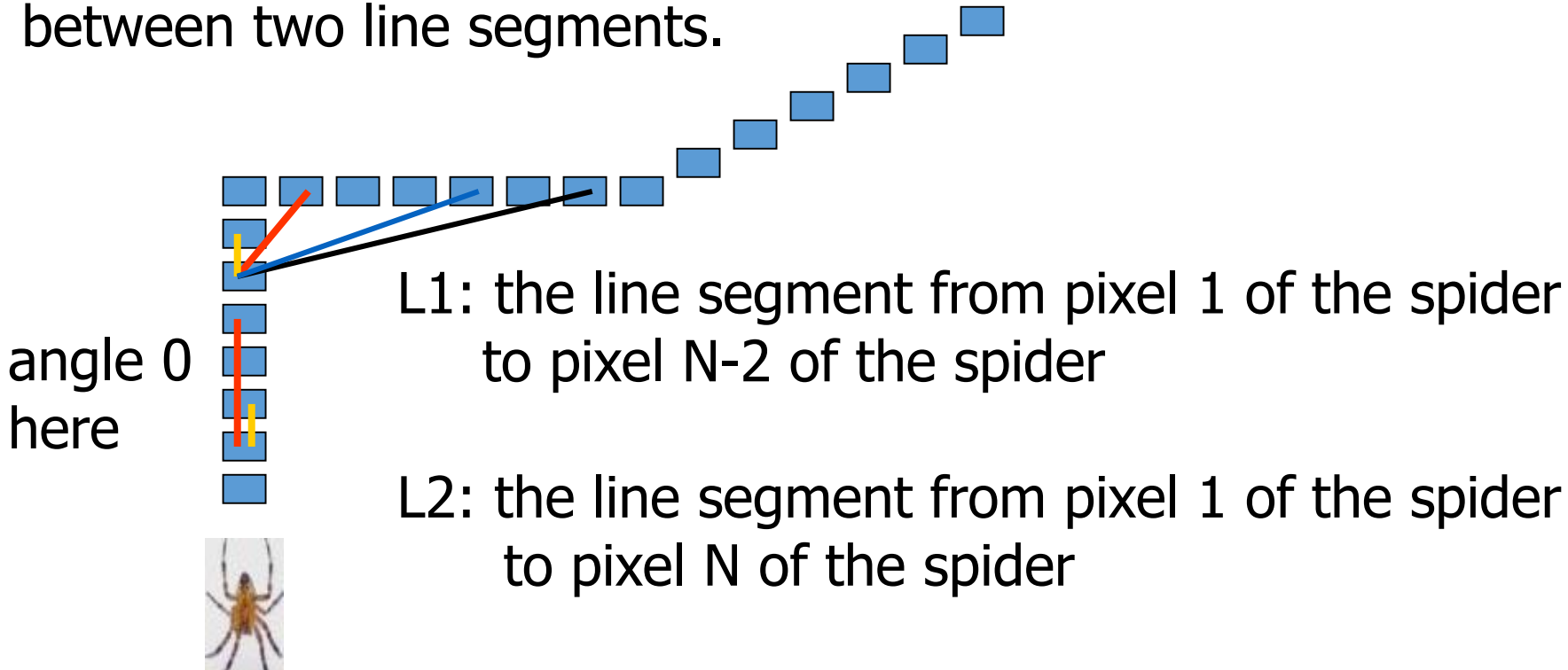


Edge tracking: ORT Toolkit

- The algorithm is called Strider and is like a spider moving along pixel chains of an image, looking for junctions and corners.
- It identifies them by a measure of local asymmetry.
 - When it is moving along a straight or curved segment with no interruptions, its legs are symmetric about its body.
 - When it encounters an obstacle (i.e., a corner or junction) its legs are no longer symmetric.
 - If the obstacle is small (compared to the spider), it soon becomes symmetrical.
 - If the obstacle is large, it will take longer.
- The accuracy depends on the length of the spider and the size of its stride.
 - The larger they are, the less sensitive it becomes.

Edge tracking: ORT Toolkit

The measure of asymmetry is the angle between two line segments.



The angle must be $\leq \arctan(2/\text{length}(L2))$

Longer spiders allow less of an angle.

Adapted from Linda Shapiro, U of Washington

Edge tracking: ORT Toolkit

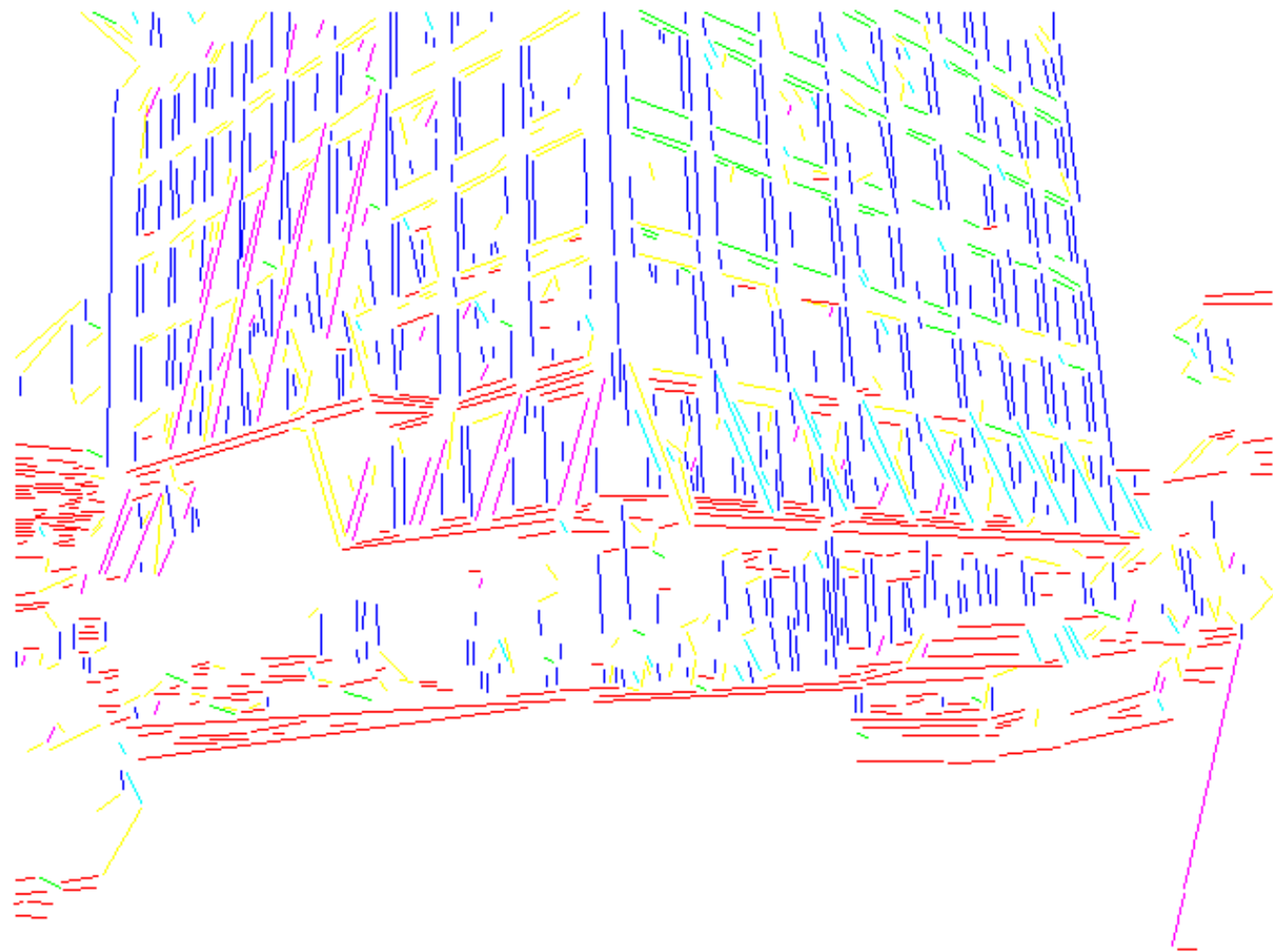
- The parameters are the length of the spider and the number of pixels per step.
- These parameters can be changed to allow for less sensitivity, so that we get longer line segments.
- The algorithm has a final phase in which adjacent segments whose angle differs by less than a given threshold are joined.
- Advantages:
 - Works on pixel chains of arbitrary complexity.
 - Can be implemented in parallel.
 - No assumptions and parameters are well understood.

Example: building detection



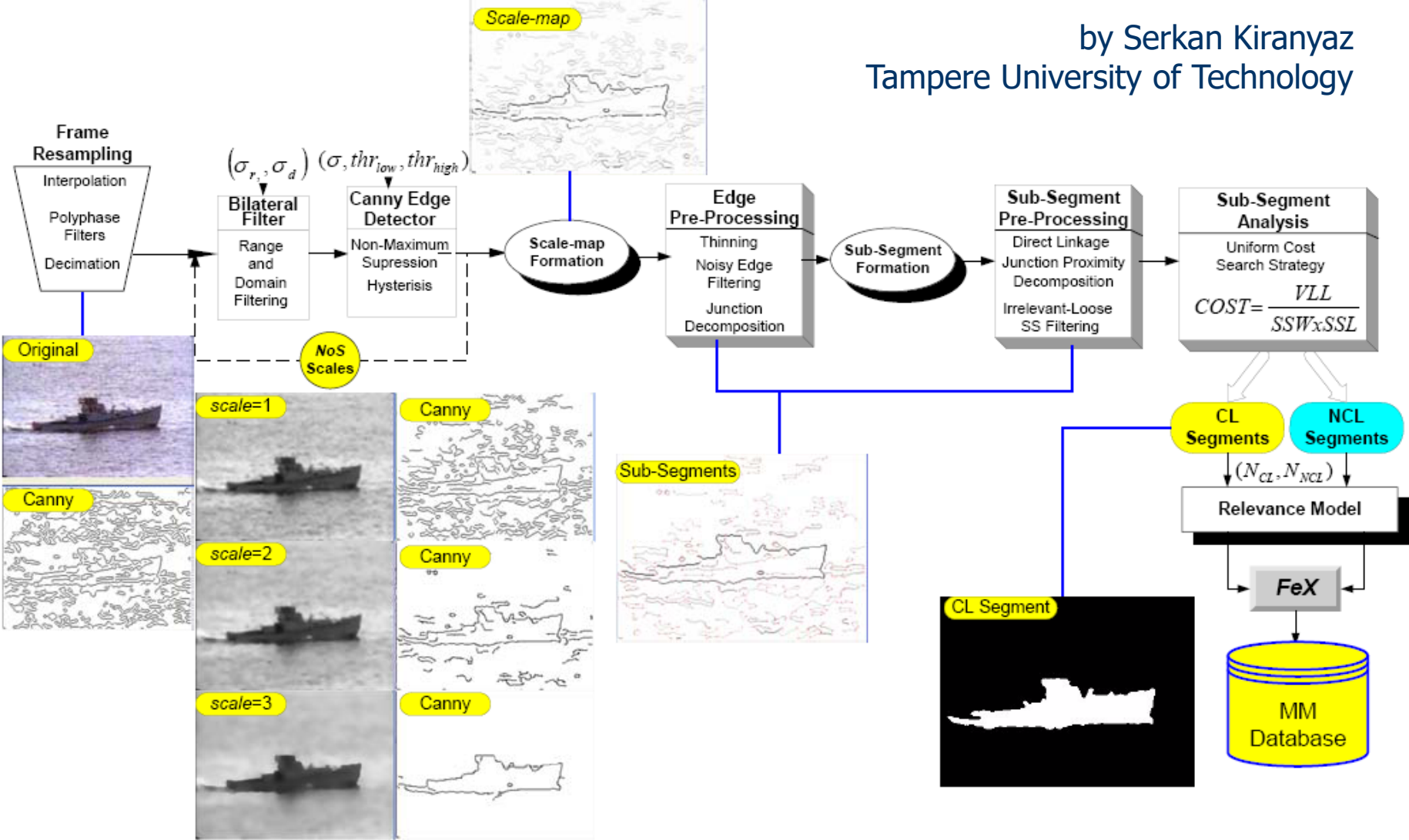
by Yi Li @ University of Washington

Example: building detection



Example: object extraction

by Serkan Kiranyaz
Tampere University of Technology



Example: object extraction

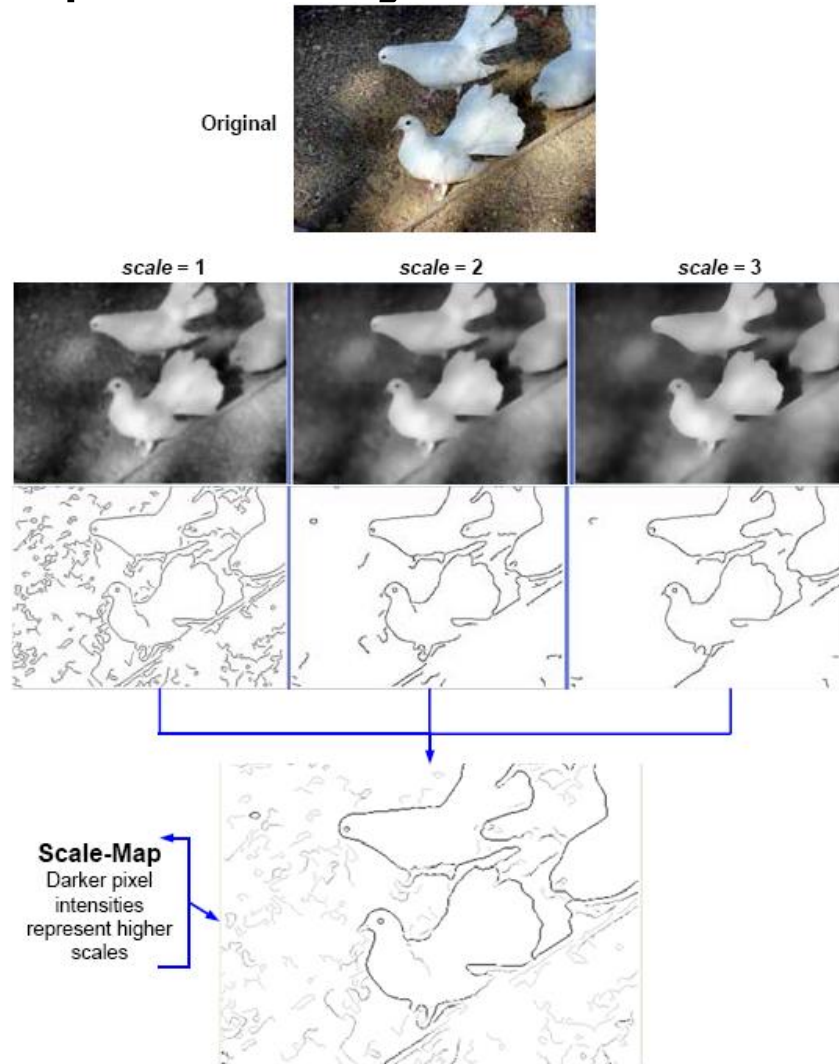


Figure 5: A sample scale-map formation.

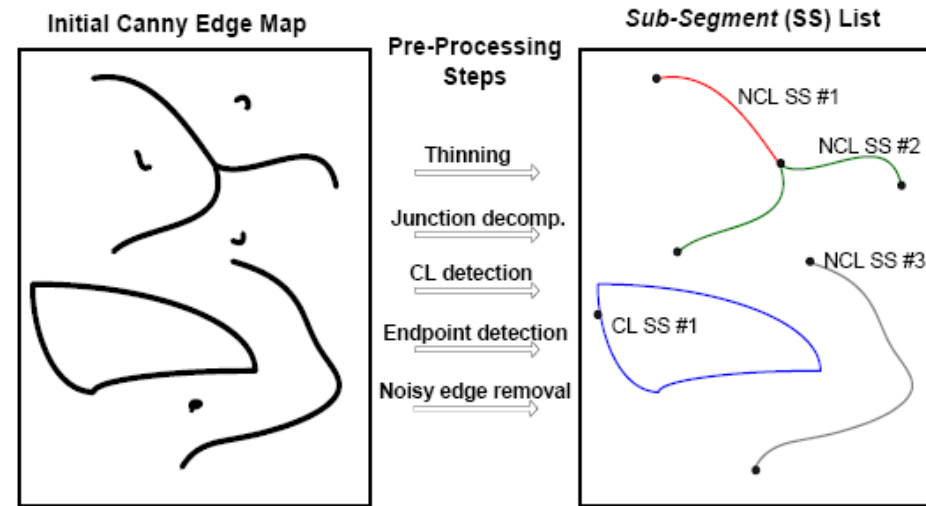


Figure 6: Sub-segment formation from an initial Canny edge field.

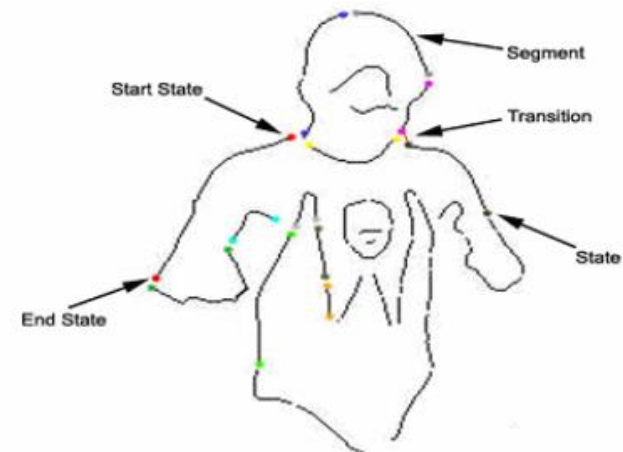


Figure 9: State space for a given sub-segment layout.

Example: object extraction

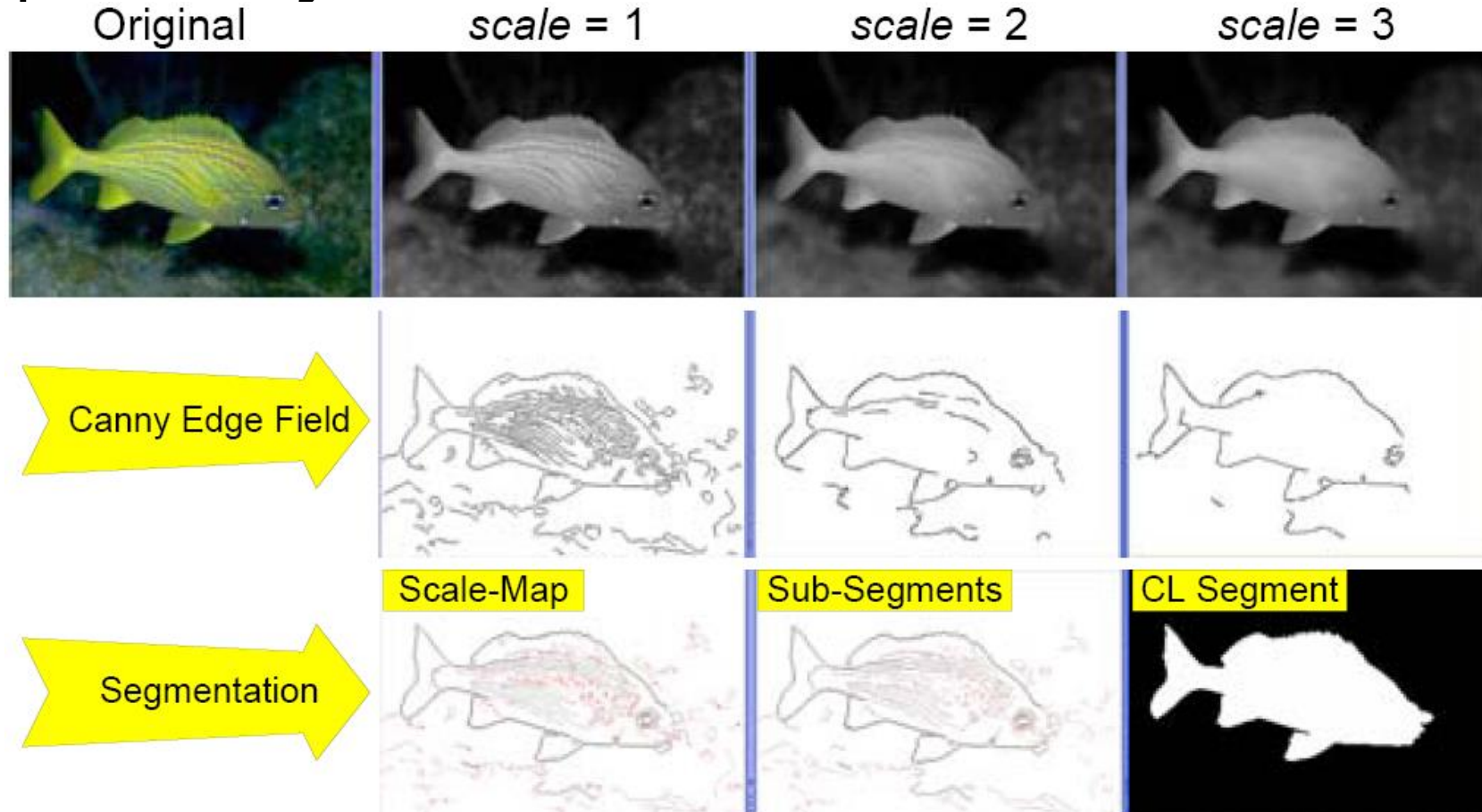
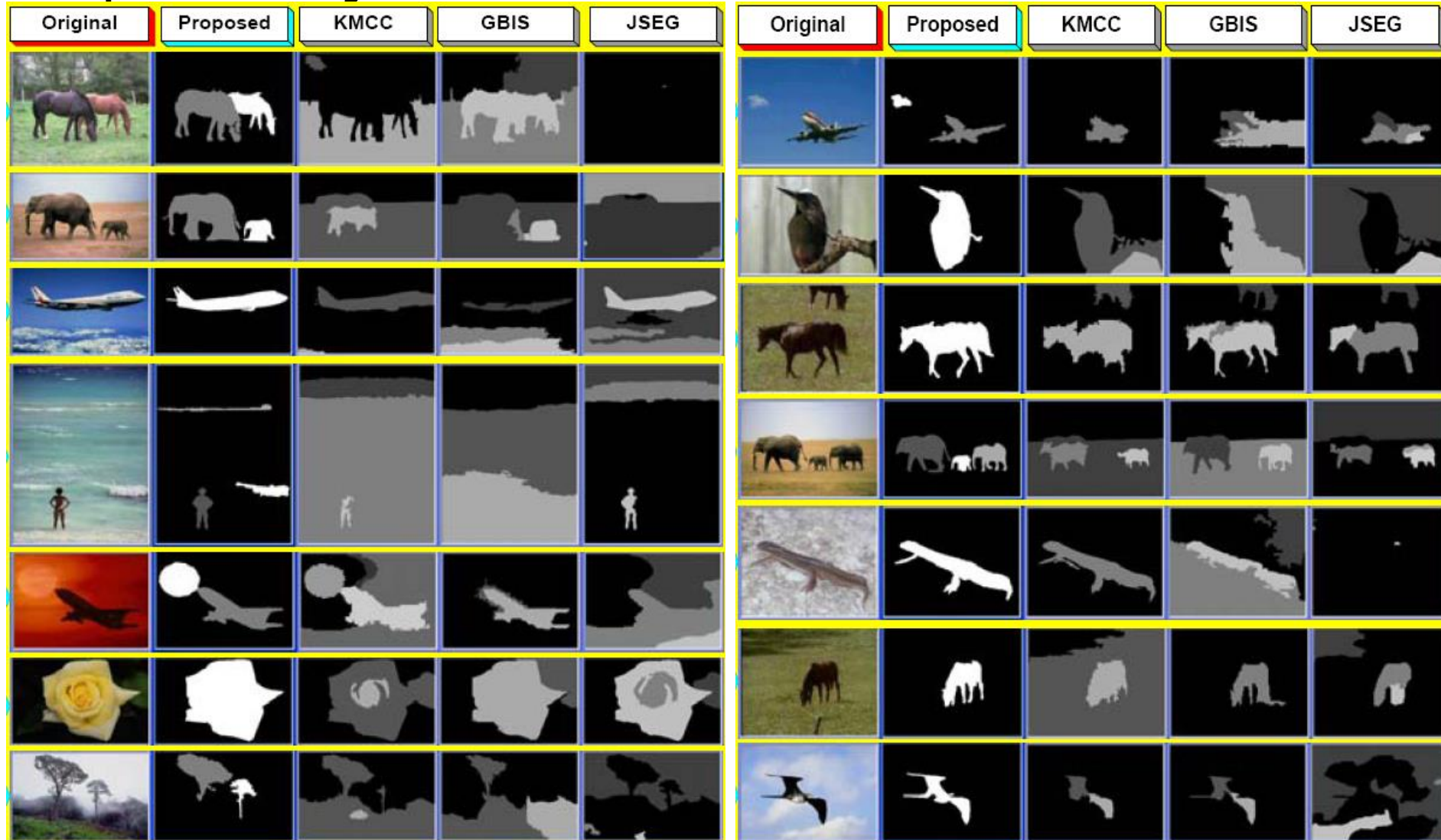


Figure 12: 3-scale simplification process over a natural image and the final CL segment extracted.

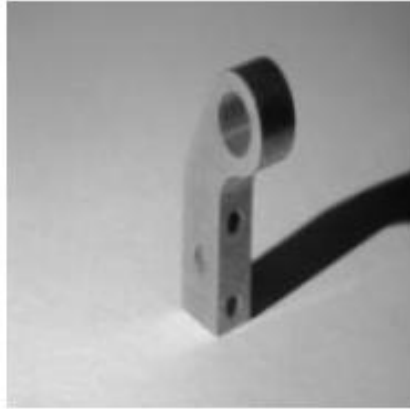
Example: object extraction



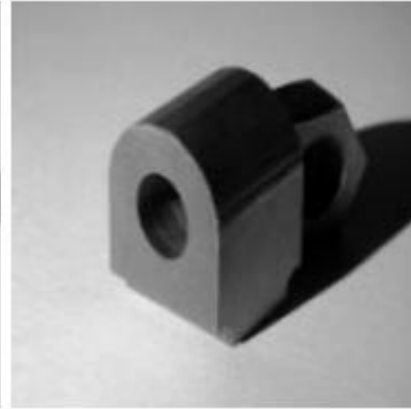
Example: object recognition

- Mauro Costa's dissertation at the University of Washington for recognizing 3D objects having planar, cylindrical, and threaded surfaces:
 - Detects edges from two intensity images.
 - From the edge image, finds a set of high-level features and their relationships.
 - Hypothesizes a 3D model using relational indexing.
 - Estimates the pose of the object using point pairs, line segment pairs, and ellipse/circle pairs.
 - Verifies the model after projecting to 2D.

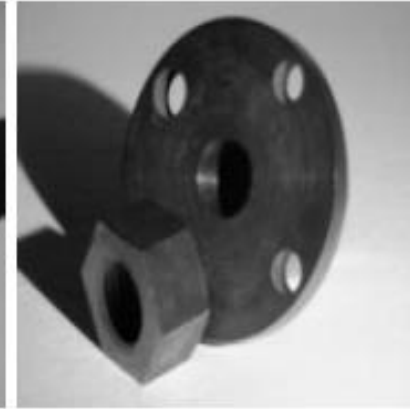
Example: object recognition



(d) Image 4 (left)



(e) Image 5 (left)



(f) Image 6 (right)



(g) Image 7 (left)



(h) Image 8 (right)



(i) Image 9 (right)

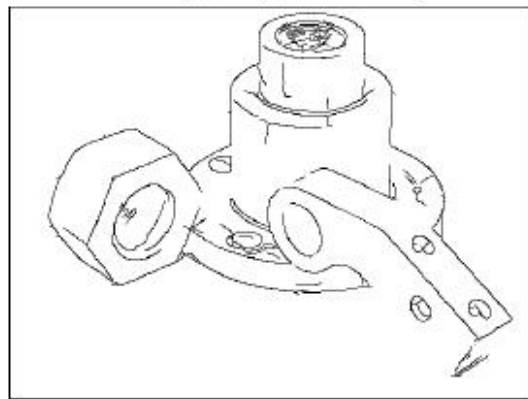
Example scenes used. The labels “left” and “right” indicate the direction of the light source.



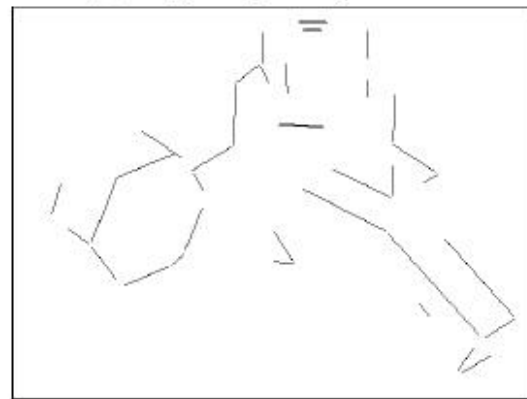
(a) Original left image



(b) Original right image



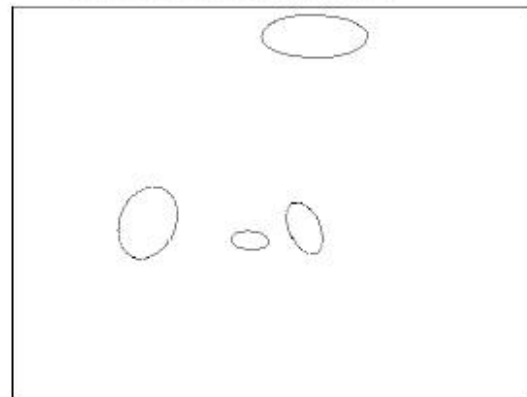
(c) Combined edge image



(d) Linear features detected



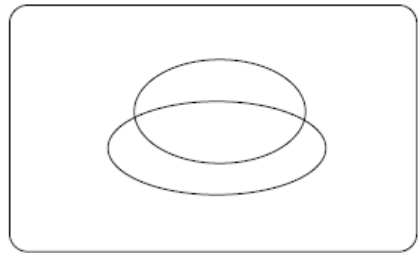
(e) Circular arc features detected



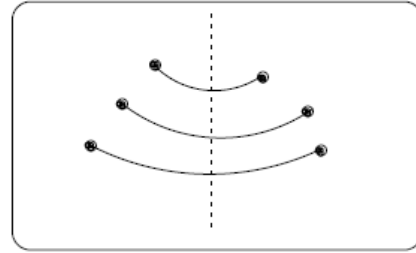
(f) Ellipses detected

Figure 22: Sample run of the system.

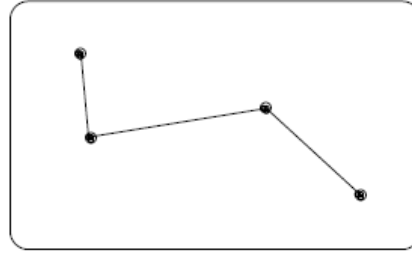
Example: object recognition



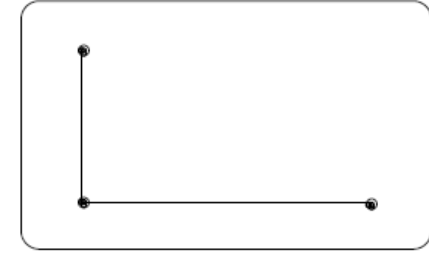
(a) Ellipses



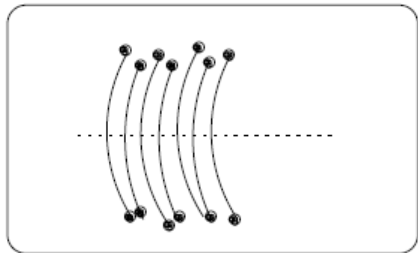
(b) Coaxials-3



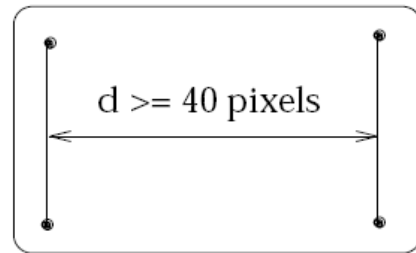
(g) Z-triple



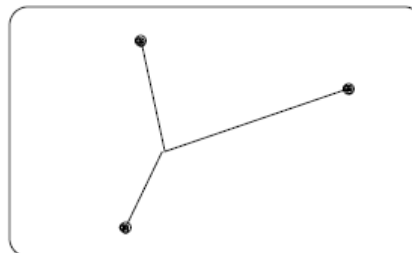
(h) L-junction



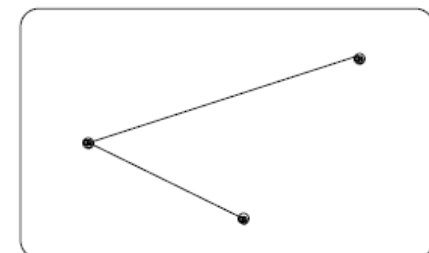
(c) Coaxials-multi



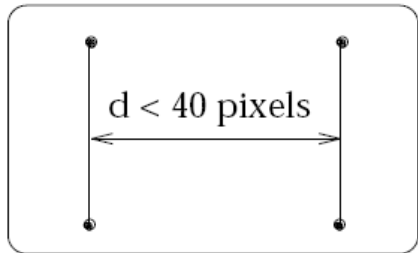
(d) Parallel-far



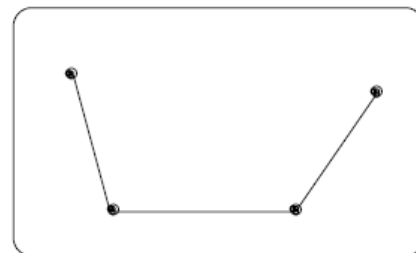
(i) Y-junction



(j) V-junction



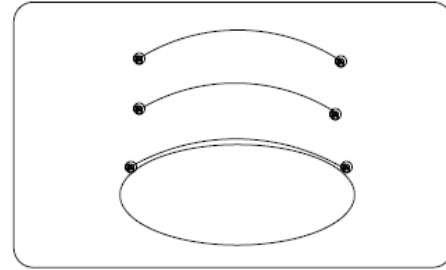
(e) Parallel-close



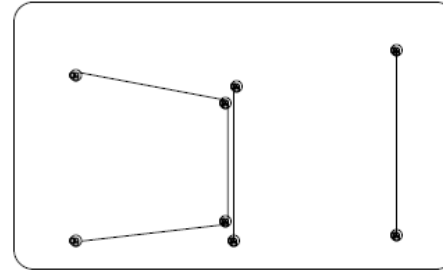
(f) U-triple

Figure 2: Features used in this work.

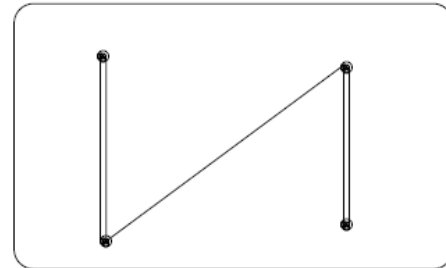
Example: object recognition



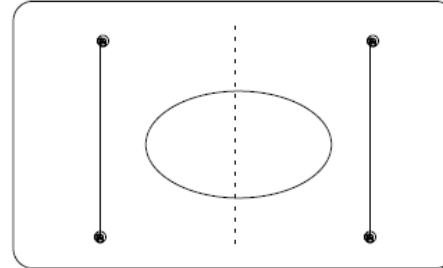
(a) Share one arc



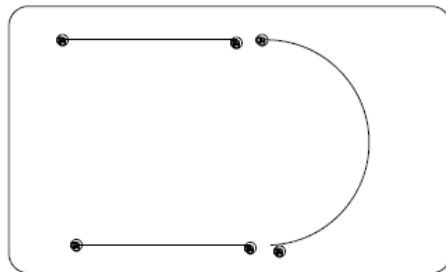
(b) Share one line



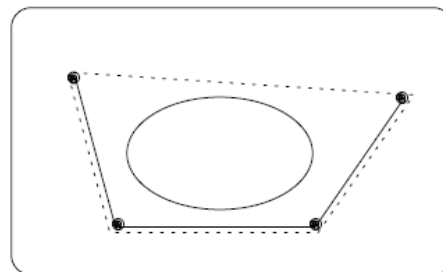
(c) Share two lines



(d) Coaxial



(e) Close at extremal points

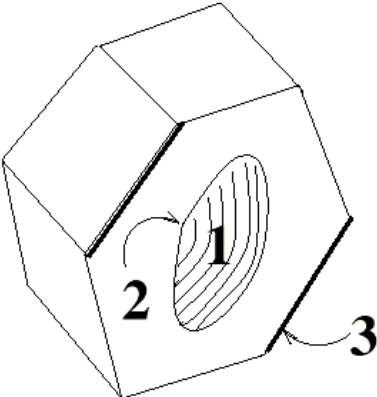


(f) Bounding box encloses/
is enclosed by bounding box

Figure 3: Relations between sample pairs of features.

Example: object recognition

MODEL-VIEW

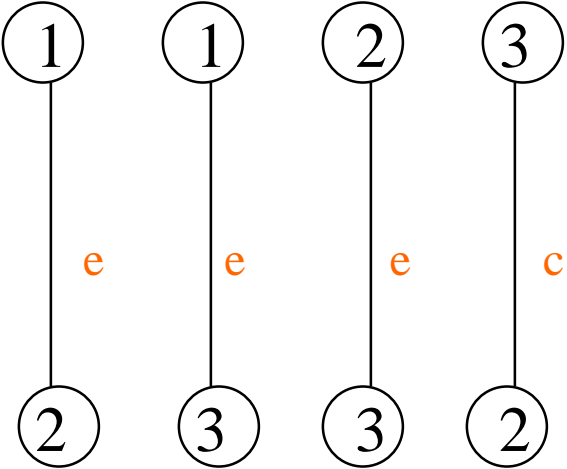
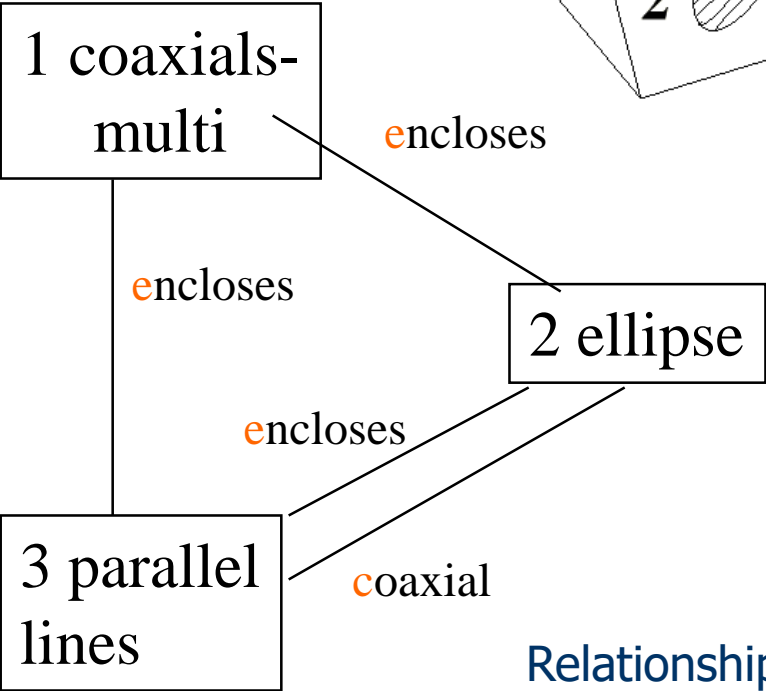


RELATIONS:

- a: encloses
- b: coaxial

FEATURES:

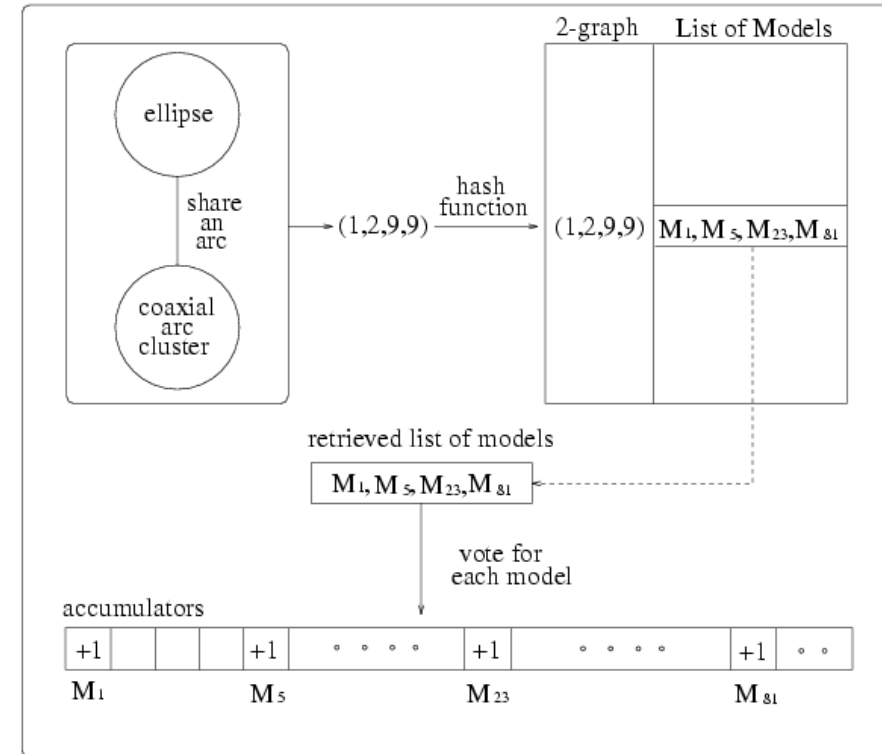
- 1: coaxials-multi
- 2: ellipse
- 3: parallel lines



Relationship graph and the corresponding 2-graphs.

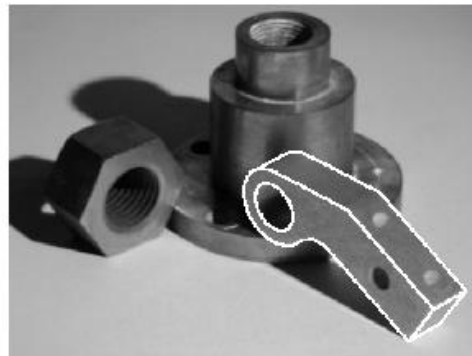
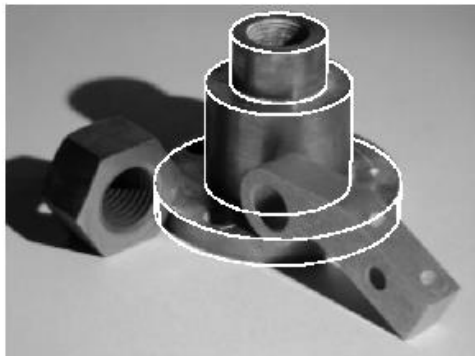
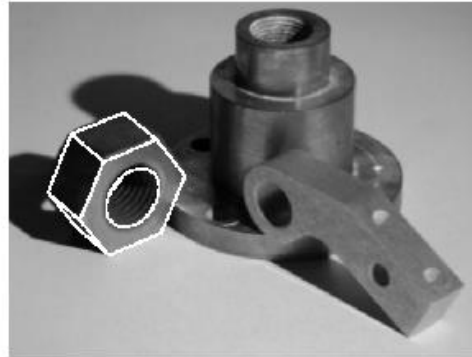
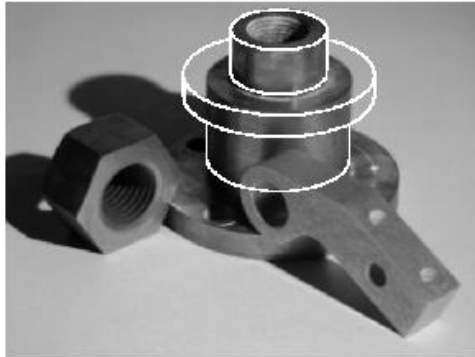
Example: object recognition

- Learning phase: relational indexing by encoding each 2-graph and storing in a hash table.
- Matching phase: voting by each 2-graph observed in the image.



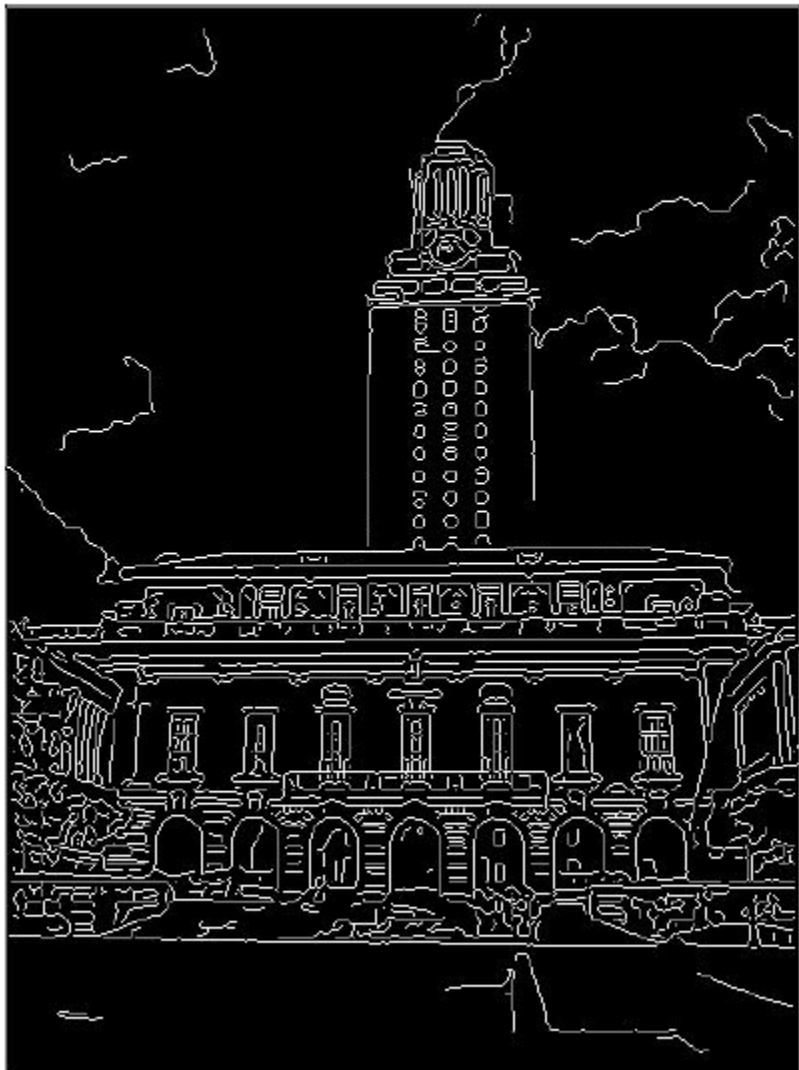
Example: object recognition

Incorrect hypothesis



1. The matched features of the hypothesized object are used to determine its **pose**.
2. The **3D mesh** of the object is used to project all its features onto the image.
3. A **verification procedure** checks how well the object features line up with edges on the image.

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?