Lecture 21
Image Segmentation with MATLAB

BioE594: Imaging Informatics

Instructors: Robert Langlois and Hui Lu
UIC Bioinformatics

Recordings: Robert Langlois
UIC Bioinformatics

Lecture Overview

- Will review some previously discussed segmentation methods and then learn how they are implemented in MATLAB.

- Topics to be Covered

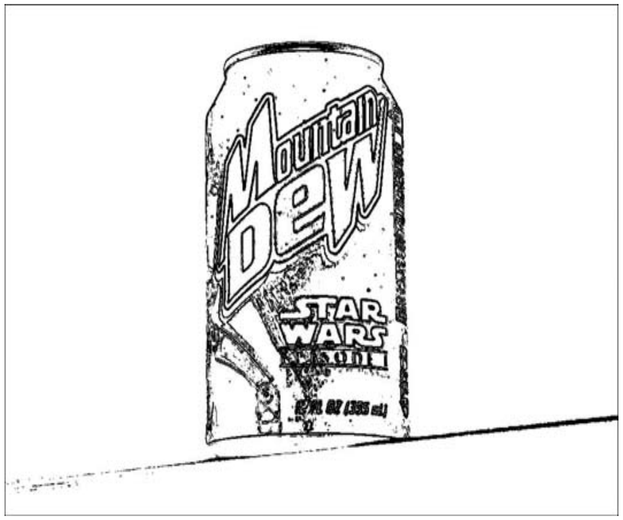
  - Edge Detection
    - Sobel Edge Detector
    - Canny Edge Detector

  - Line Detection
    - Hough Transform

  - Thresholding
    - Global Thresholding

Edge Detection Review

- Most common approaches for detecting meaningful discontinuities in intensity values.
- Such discontinuities are detected by using first- and second-order derivatives (i.e., typically the gradient and the Laplacian, respectively).
- By edge detecting an image, one can significantly reduce the amount of data by filtering out useless information, while preserving the important structural properties in the image.
- Within this lecture, we will be reviewing the Sobel and Canny edge detector methods.



Review of Gradient

- The gradient of a 2D function, f(x,y), is defined as the vector

$$\nabla f \equiv grad(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}$$

whose magnitude is

$$M(x,y) = mag(\nabla f) = \sqrt{g_x^2 + g_y^2}$$

and whose direction is given by the angle

$$\alpha(x,y) = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

- α is measured with respect to the x-axis and is an image of the same size as the original created by the array division of image g_y by image g_x

- The direction of an edge at an arbitrary point (x, y) is orthogonal to the direction, α(x, y), of the gradient vector at the point.

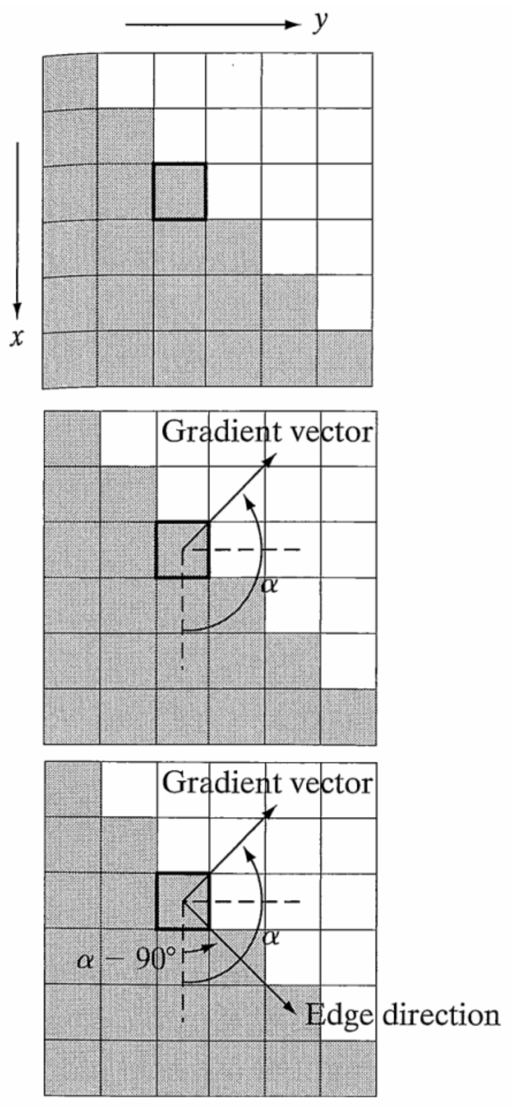


Figure 10.12.
Gonzalez & Woods (2008), p 707.

Edge Detection Using Function edge in MATLAB

- General syntax

  [g, t] = edge (f, 'method', parameters)

  where

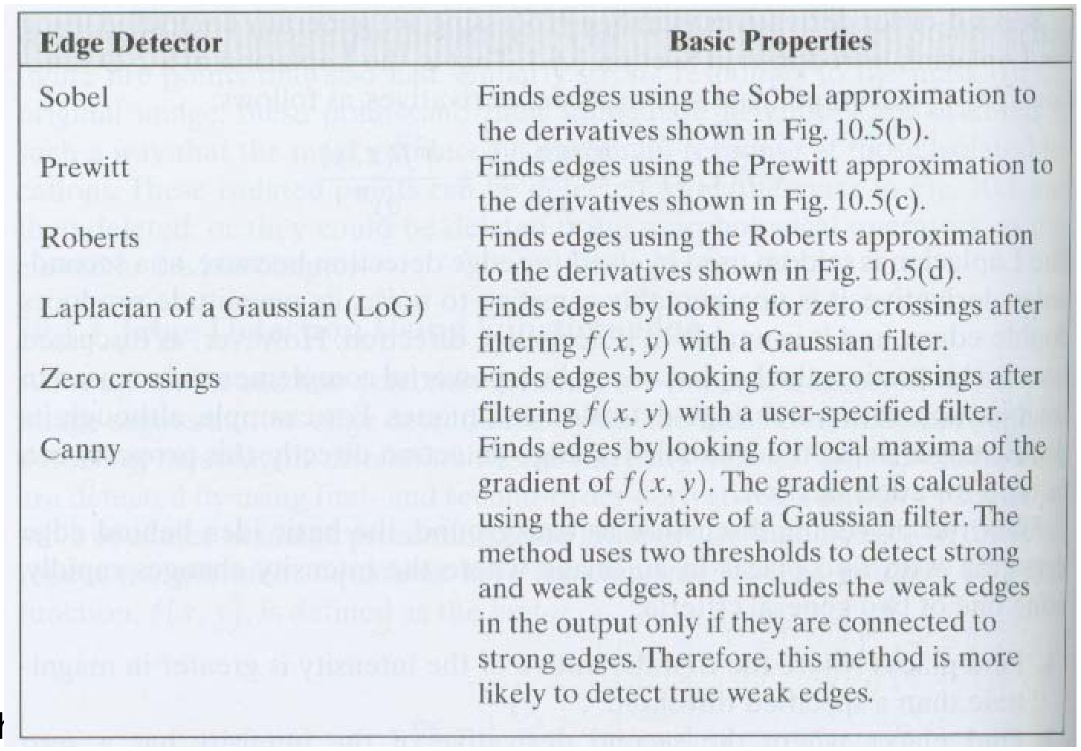
  f = input image

  method = one of the available edge detectors shown to the right.

  parameters = additional parameters

  g = an outputted logical array with 1s at th locations where edge points were detected in f and 0s elsewhere.

  t = optional parameter that gives the threshold used by edge to determine which gradient values are strong enough to be called edge points.



Edge Detector	Basic Properties
Sobel	Finds edges using the Sobel approximation to the derivatives shown in Fig. 10.5(b).
Prewitt	Finds edges using the Prewitt approximation to the derivatives shown in Fig. 10.5(c).
Roberts	Finds edges using the Roberts approximation to the derivatives shown in Fig. 10.5(d).
Laplacian of a Gaussian (LoG)	Finds edges by looking for zero crossings after filtering $f(x, y)$ with a Gaussian filter.
Zero crossings	Finds edges by looking for zero crossings after filtering $f(x, y)$ with a user-specified filter.
Canny	Finds edges by looking for local maxima of the gradient of $f(x, y)$. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. Therefore, this method is more likely to detect true weak edges.

Table 10.1: Edge detectors available in function edge*. Gonzalez, Woods, & Eddins (2004), p 386.

***Fig 10.5 mentioned in above table is shown in next slide.**

Performing Sobel Edge Detection in MATLAB

- By using the Sobel masks shown to the right to approximate the first derivatives G_x and G_y, the Sobel detector can compute the gradient at the center point in the neighborhood as $g = [G^2_x + G^2_y]^{1/2}$.

- Can then define a pixel at location (x,y) as an edge pixel if $g \geq T$ at that location. Here T is a specified threshold.

- Syntax

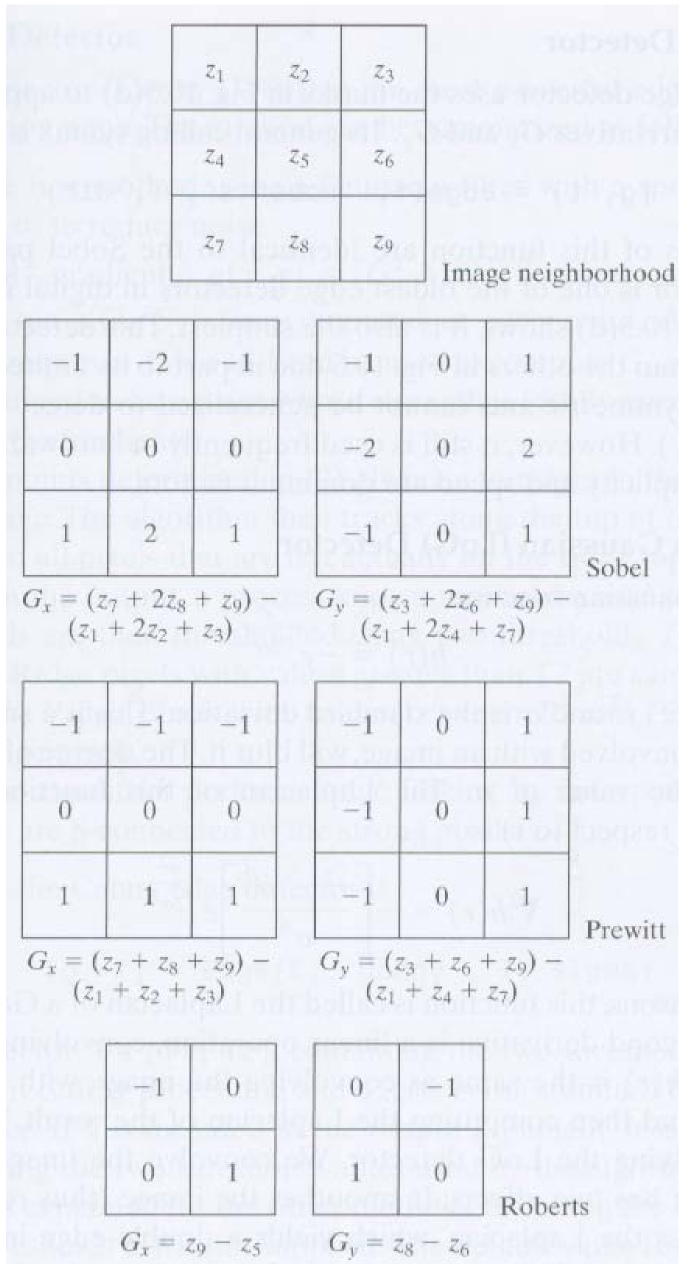
  [g, t] = edge (f, 'sobel', T, dir) where

  f = input image

  T = specified threshold

  dir = preferred direction of the edges detected (i.e., ' horizontal', ' vertical', or 'both' (the default)).

  If T is specified, then t = T. Otherwise, edge sets t to an automatically determined threshold.



z_1 z_2 z_3
z_4 z_5 z_6
z_7 z_8 z_9
Image neighborhood
$\begin{array}{ccc} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{array}$
$\begin{array}{ccc} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{array}$
Sobel
$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$
$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$
$\begin{array}{ccc} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{array}$
$\begin{array}{ccc} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{array}$
Prewitt
$G_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$
$G_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$
$\begin{array}{cc} -1 & 0 \\ 0 & 1 \end{array}$
$\begin{array}{cc} 0 & -1 \\ 1 & 0 \end{array}$
Roberts
$G_x = z_9 - z_5$
$G_y = z_8 - z_6$

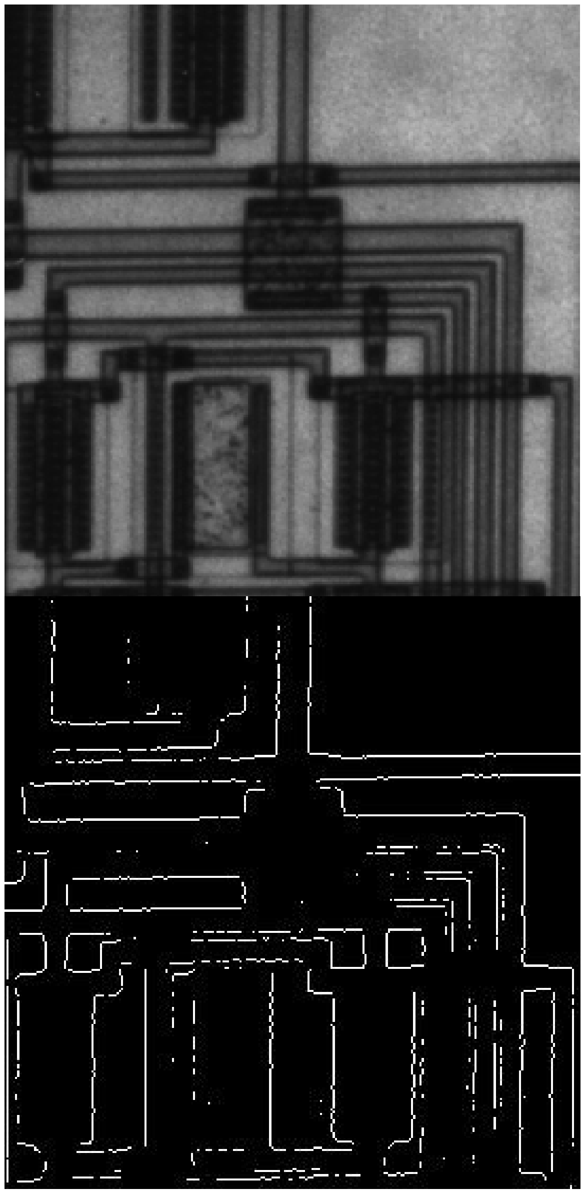
Figure 10.5: Some edge detector masks and the first order derivatives they implement. Gonzalez, Woods, & Eddins (2004), p 387.

Performing Sobel Edge Detection in MATLAB

- For example, consider the top figure to the right that shows the image of a circuit. To obtain the edges of this image using the Sobel method with default arguments, one would use the code shown below.

```
>> I = imread('circuit.tif');
>> BW = edge(I,‘sobel');
>> figure, imshow(BW)
```

The bottom right figure shows the resulting image.



Figures showing a circuit (top) and the result of applying the Sobel method to detect edges (below).

Review of Canny Edge Detection

- Also known as an optimal edge detector

- Steps of the Canny Algorithm
  - Filter out any noise in the original image with a Gaussian filter.

    $g(m,n) = G_\sigma(m,n) * f(m,n)$ where $G_\sigma = \frac{1}{\sqrt{2\pi\sigma^2}} exp[-\frac{m^2+n^2}{2\sigma^2}]$

  - Using a gradient operator (eg, Sobel, Roberts, etc), take the gradient of the image to find the edge strength.

    $M(n,n) = \sqrt{g_m^2(m,n) + g_n^2(m,n)}$ and $\theta(m,n) = tan^{-1}[g_n(m,n)/g_m(m,n)]$ (the gradient direction)

  - Suppress non-maxima pixels in the edges to thin edge ridges that may have been broadened in the first step. If M_t(m,n) is greater than its two neighbors along the gradient direction θ (m,n), M_t(m,n) remains unchanged. Otherwise, it is set to 0.

  - Threshold M. Choose T to keep all edge elements while suppressing most of the noise.

    $$M_T(m,n) = \begin{cases} M(m,n) & \text{if } M(m,n) > T \\ 0 & \text{otherwise} \end{cases}$$

  - The gradient is reduced further by hysteresis. In this process, the current result is thresholded by two different thresholds τ_1 and τ_2 (where $\tau_1 < \tau_2$) to obtain two binary images T_1 and T_2. T_2 has less noise and fewer false edges than T_1, but larger gaps between edge elements.

  - By comparing edge segments between T_1 and T_2, edge segments in T_2 can be linked to form continuous edges.

Performing Canny Edge Detection in MATLAB

- Syntax

  [g, t] = edge (f, ‘canny’, T, sigma) where

  f = input image

  T = a vector, T=[T1, T2], containing the two thresholds discussed in the algorithm. T is automatically computed if not supplied.

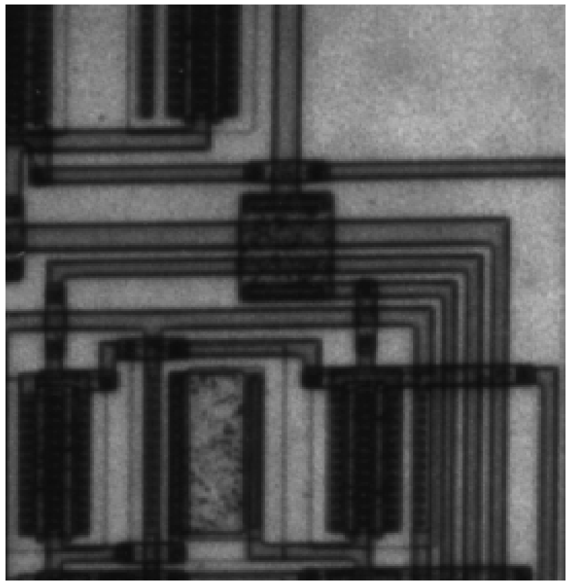
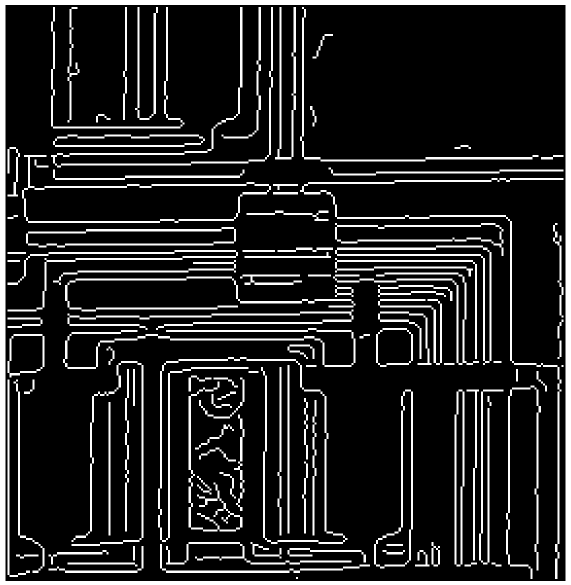
  sigma = the standard deviation of the smoothing filter (default = 1).

  T = if included, it is a 2 element vector containing the two threshold values used by the algorithm.

- Consider again the image of a circuit (shown top right). To obtain the edges of this image using the Canny method with default arguments, one would use the code shown below.

```
>> I = imread('circuit.tif');
>> BW = edge(I,'canny');
>> figure, imshow(BW)
```

  The bottom right figure shows the resulting image.

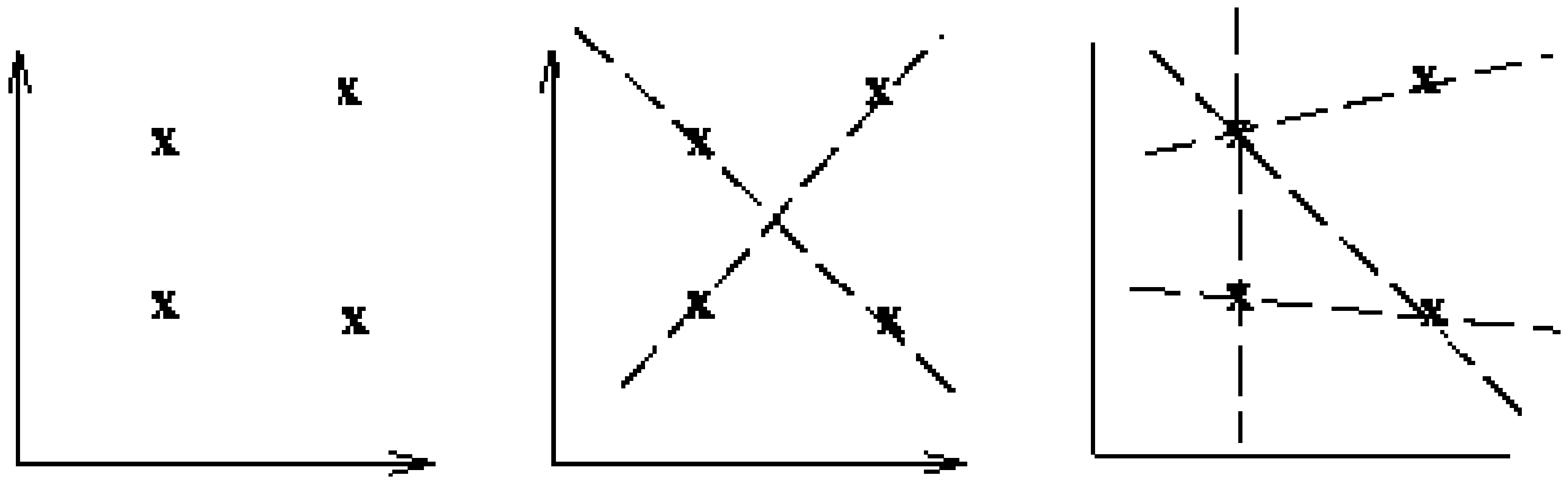
Figures showing a circuit (top) and the result of applying the Canny method (below).

Line Detection and the Hough Transform

- Ideally, the previous edge detector methods should yield pixels lying only on edges.
- However, due to spurious intensity discontinuities resulting from noise, nonuniform illumination, and other effects, these methods are more likely to produce pixels which seldom characterize an edge completely.
- To deal with this problem, the edge detector methods are usually followed by linking procedures to assemble edge pixels into meaningful edges. The Hough transform is one such approach that can be used to find and link line segments in an image.
- How Hough Works
  - The Hough technique useful for
    - Computing a global description of a feature(s)
    - Number of solution classes need not be known *a priori*
    - Given (possibly noisy) local measurements
  - The motivating idea for line detection
    - Each input measurement (*e.g.* coordinate point)
    - Indicates its contribution to a globally consistent solution
    - *e.g.* the physical line which gave rise to that image point

Hough - Motivating Example

- Common problem: fitting a set of line segments to a set of discrete image points
- *e.g.* Pixel locations output from an edge detector
- Figure below shows some possible solutions to this problem
- Lack of *a priori* knowledge: number of desired line segments
- Ambiguity about what constitutes a line segment
- Render this problem under-constrained



Hough - Parametric Form

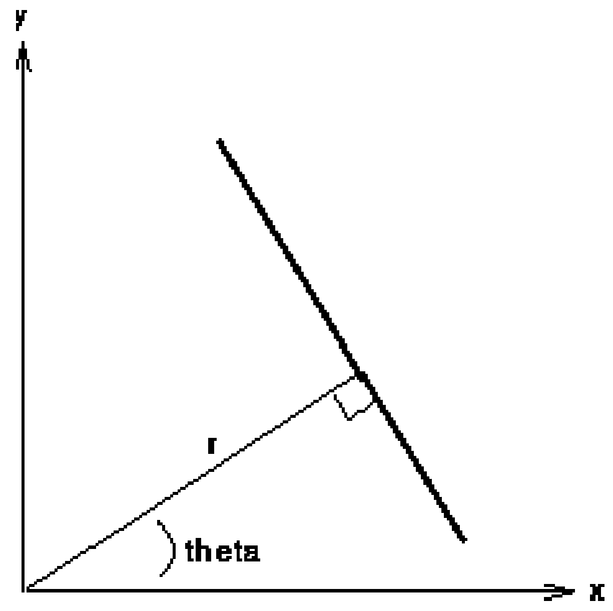
- Many forms to describe line segments
- Convenient form (Parametric)

$$y = mx + b$$

$$\downarrow$$

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

- *r* is length of normal from origin
- θ is orientation of *r* wrt. x-axis
- For any point (*x*,*y*), *r* and θ are <u>constant</u>



Hough - Interpretation

- Known: points (x_i,y_i)
  - Serve as constraints for parametric equation
- Unknown: r and θ

- Point-curve Transformation
  - Cartesian → Polar (Hough) Space
  - (x_i, y_i) → (r, θ)

- In Hough Space
  - Collinear points → intersecting curves
  - At common (r, θ)

Hough - Method

- Quantize the Hough parameter space into
  - finite intervals or
  - *accumulator cells*

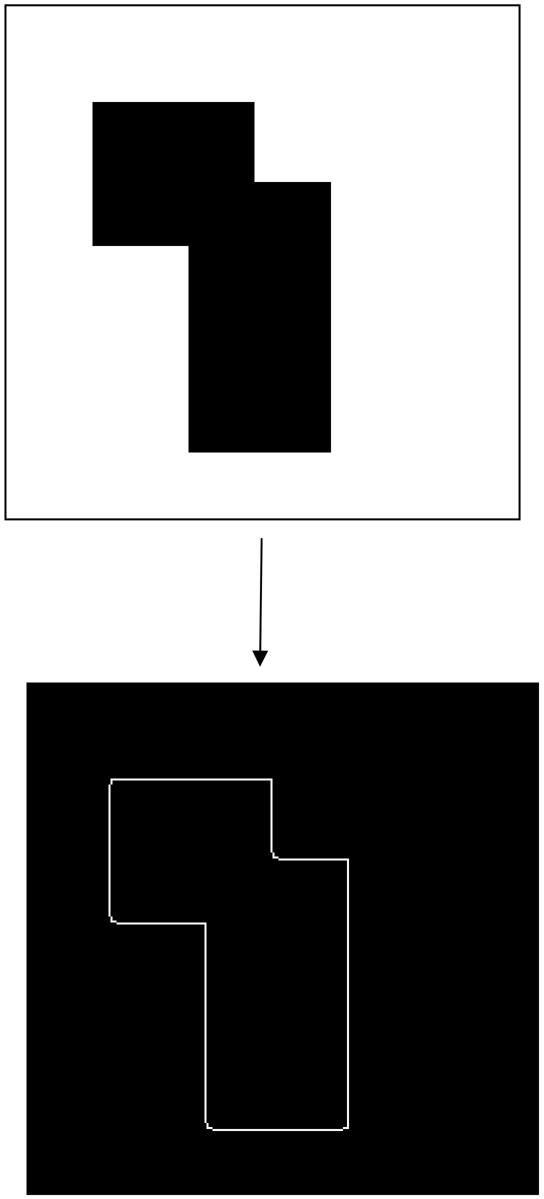
- Each (x_i, y_i) transformed into a discretized (r, θ) curve
  - the accumulator cells lying along this curve are incremented

- Resulting peaks in accumulator array
  - strong evidence that a corresponding straight line exists in the image

		r			
		1-2	2-3	3-4	4-5
θ	1-10	0	1	3	0
	10-20	0	0	2	0

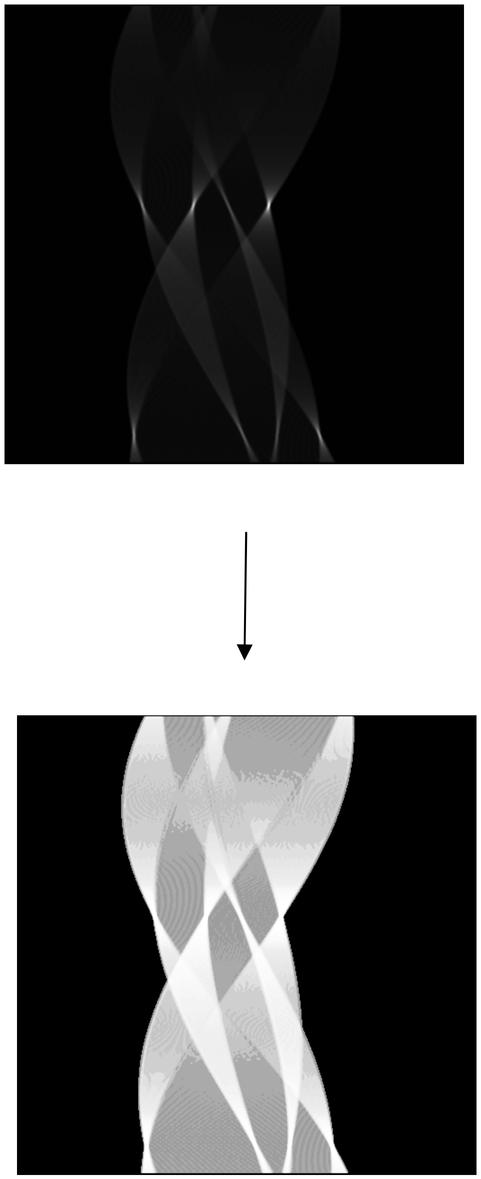
Hough - Example

- Begin simple image
  - Two rectangles (Top)
- Canny Edge Detector
  - Boundary (Bottom)
- Need features within boundary!
- Hough Transform (line detect)
  - Detect 8 line segments
  - True geometric structure



Hough - Example

- Hough Transform
  - Input: Edge/Boundary Points
  - Output: Curve in polar space
- Accumulator Array (Top)
  - Viewed as intensity image
  - Plotted on rectangular (not Polar)
- Histogram Equalizing (Bottom)
  - Patterns in low intensity pixels
  - Accumulator space wraps around vertical edge of image
  - Only 8 real peaks



Hough - Intersection Points

- Collinear points → Intersect in peaks
  - Collinear in Cartesian (original) space
  - Intersection points in Hough space

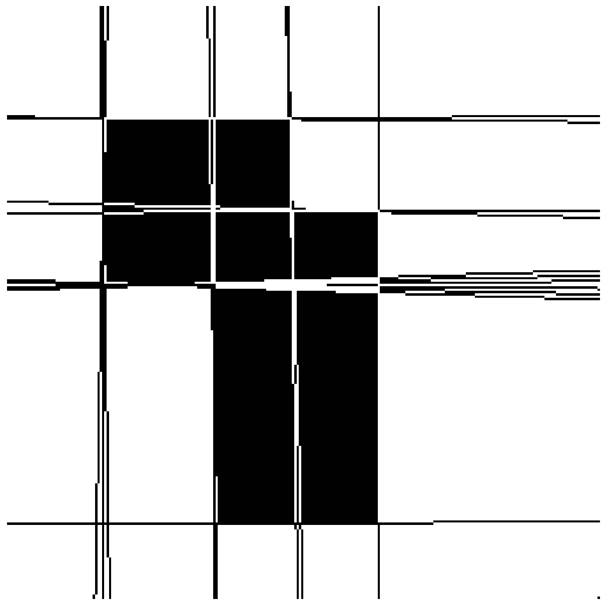
- Intersection points characterize straight line segments

- Extract bright points (peaks or local maxima)
  - One method: thresholding then thinning
  - Isolate clusters of bright spots
  - Relative threshold: fixed percentage of global max

Hough - Dehoughing

- Mapping back from Hough transform space
  - Yields a set of line descriptions
  - Overlay images (right)
  - Found 8 true sides
  - Two rectangles

- Accuracy of alignment not perfect
  - Determined by accumulator quantization
  - Many image edges have several detected lines
  - Arise from several nearby peaks
  - Hough lines infinitely long



Performing Hough Transform-Based Line Detection in MATLAB

- 3 Steps
  - First, compute the Hough transform of an image using the hough () function.

    [H, theta, rho] = hough(BW, param1, val1, param2, val2) where

    BW = binary image

    param1, val1, param2, val2 = specifies optional parameter/value pairs
    - 'ThetaResolution' -> real scalar value between 0 and 90, exclusive, that specifies the spacing (in degrees) of the Hough transform bins along the theta axis (Default = 1).
    - 'RhoResolution' -> real scalar value between 0 and norm(size(BW)), exclusive, that specifies the spacing of the Hough transform bins along the rho axis (Default =1).

    H= Hough transform matrix

    theta (in degrees) = array of theta values over which the Hough transform matrix was generated

    rho = array of rho values over which the Hough transform matrix was generated

Performing Hough Transform-Based Line Detection in MATLAB

– Next, find a meaningful set of distinct peaks in the Hough transform using the houghpeaks () function.

peaks = houghpeaks(H, numpeaks, param1, val1, param2, val2) where

H = Hough transform matrix generated by the hough function

numpeaks = specifies the maximum number of peaks to identify (If omitted, defaults to 1)

param1, val1, param2, val2 = specifies optional parameter/value pairs

- 'Threshold‘ -> Specifies the threshold at which values of H are considered to be peaks. Can vary from 0 to Inf. Default is 0.5*max(H(:)).
- 'NHoodSize‘ -> Two-element vector of positive odd integers: [M N]. Specifies the size of the suppression neighborhood. This is the neighborhood around each peak that is set to zero after the peak is identified. Default is the smallest odd values greater than or equal to size(H)/50.

peaks = a Q-by-2 matrix, where Q can range from 0 to numpeaks. Q holds the row and column coordinates of the peaks.

Performing Hough Transform-Based Line Detection in MATLAB

– Finally, once a set of candidate peaks has been identified in the Hough transform, it needs to be determined if there are line segments associated with those peaks, as well as where they start and end. This can be accomplished using the using the houghlines () function.

lines = houghlines (BW, theta, rho, peaks, param1, val1, param2, val2) where

BW = binary image

theta, rho = vectors returned by function hough ()

peaks = a matrix returned by the houghpeaks () function that contains the row and column coordinates of the Hough transform bins to use in searching for line segments

param1, val1, param2, val2 = specifies optional parameter/value pairs

- 'FillGap' -> Specifies the distance between two line segments associated with the same Hough transform bin. When the distance between the line segments is less the value specified, the houghlines () function merges the line segments into a single line segment. (Default = 20)
- 'MinLength' -> Specifies whether merged lines should be kept or discarded. Lines shorter than the value specified are discarded. (Default = 40).

lines = a structure array whose length equals the number of merged line segments found. Each element of the structure array has these fields:

- point1 = Two element vector [X Y] specifying the coordinates of the end-point of the line segment
- point2 = Two element vector [X Y] specifying the coordinates of the end-point of the line segment
- theta = Angle in degrees of the Hough transform bin
- rho = rho axis position of the Hough transform bin

Example of Performing Hough Transform-Based Line Detection

- The following example shows how to use these functions to detect lines in an image.
- Read an image into the MATLAB workspace.

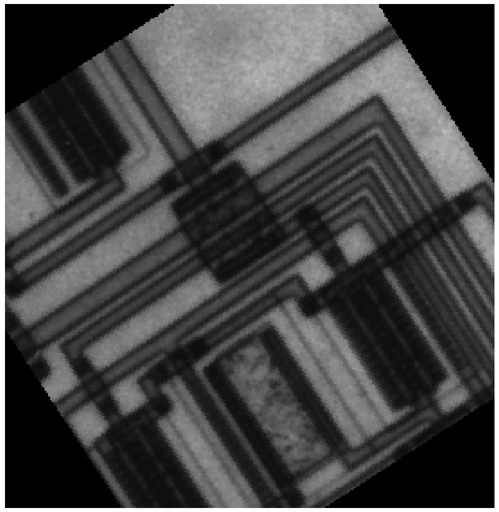
```
>> I = imread('circuit.tif');
```

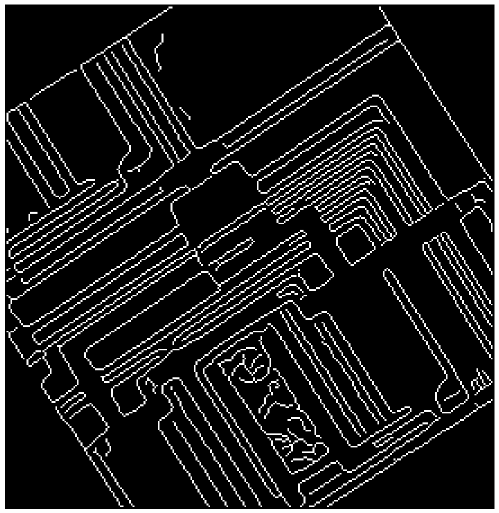
- For this example, rotate and crop the image.

```
>> rotI = imrotate(I,33,'crop');    % Top right figure
```

- Find the edges in the image.

```
>> BW = edge(rotI,'canny');  % Bottom right figure
```





Example of Performing Hough Transform-Based Line Detection

- Compute the Hough transform of the image.

```
>> [H,theta,rho] = hough(BW);
```

- Display the transform.

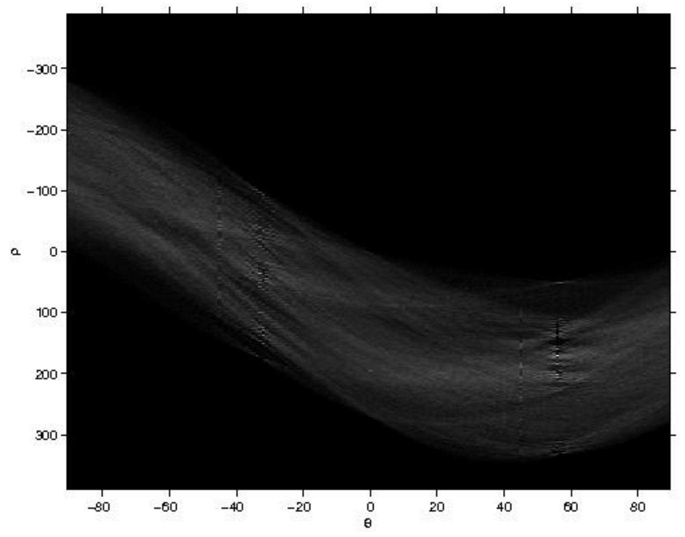
```
>> imshow (H, [], 'XData', theta, 'YData', rho,
        'InitialMagnification', 'fit');    % Top right figure
>> xlabel('\theta'), ylabel('\rho');
>> axis on, axis normal, hold on;
```

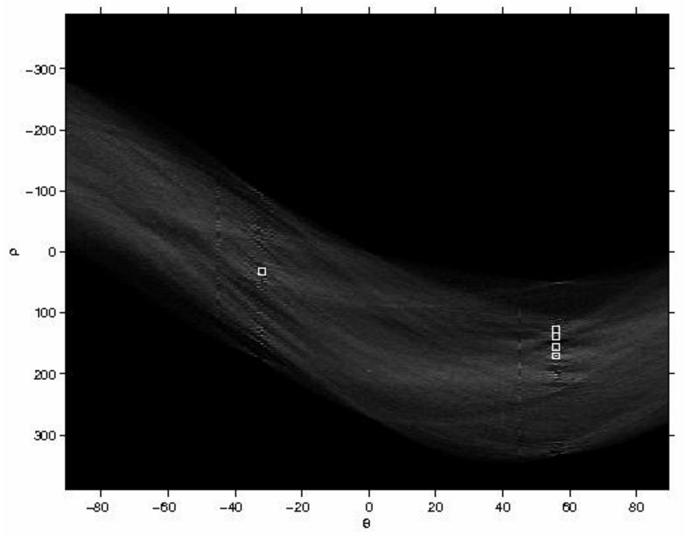
- Find the peaks in the Hough transform matrix, H.

```
>> P = houghpeaks (H, 5, 'threshold',
   ceil(0.3*max(H(:))));
```

- Plot the peaks.

```
>>  x = theta(P(:,2));
>>  y = rho(P(:,1));
>>  plot(x,y,'s','color','white');   % Bottom right figure
```





Example of Performing Hough Transform-Based Line Detection

- Find lines in the image.

```
>>  lines = houghlines (BW, theta, rho, P, 'FillGap', 5,
     'MinLength', 7);
```

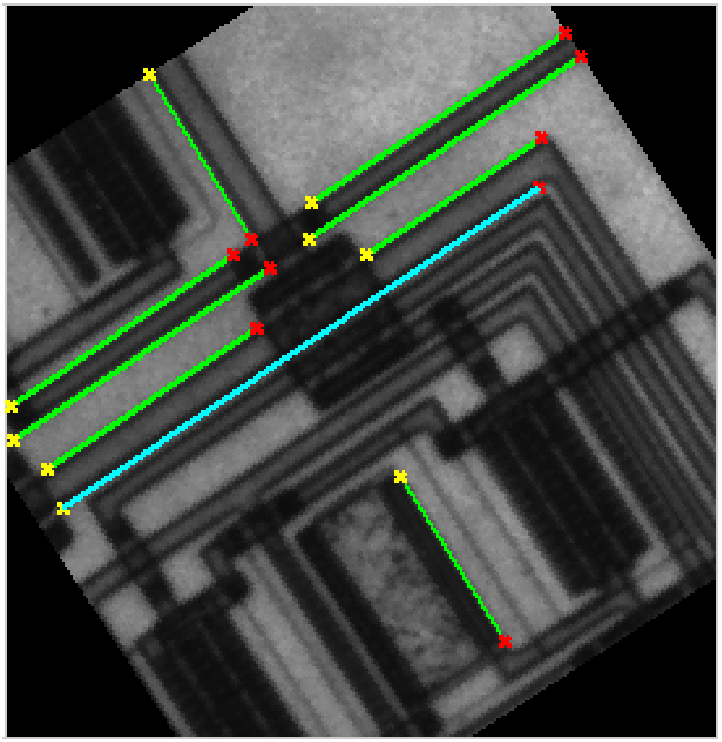
- Create a plot that superimposes the lines on the original image.

```
>> figure, imshow(rotI), hold on
>> max_len = 0;
>> for k = 1:length(lines)
        xy = [lines(k).point1; lines(k).point2];
        plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

        % Plot beginnings and ends of lines
        plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
        plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

        % Determine the endpoints of the longest line segment
        len = norm(lines(k).point1 - lines(k).point2);
        if ( len > max_len)
           max_len = len;
           xy_long = xy;
        end
   end

 % highlight the longest line segment
 plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','cyan');
```



Thresholding

- Thresholding
  - Simplest method of segmentation
  - Pixels are grouped into “Object” and “Background”
  - Input: Gray scale image
  - Output: binary image

- Typically
  - Object pixels given value of 1
  - Background pixels given value of 0
  - Then the thresholded image g(x, y) is defined as

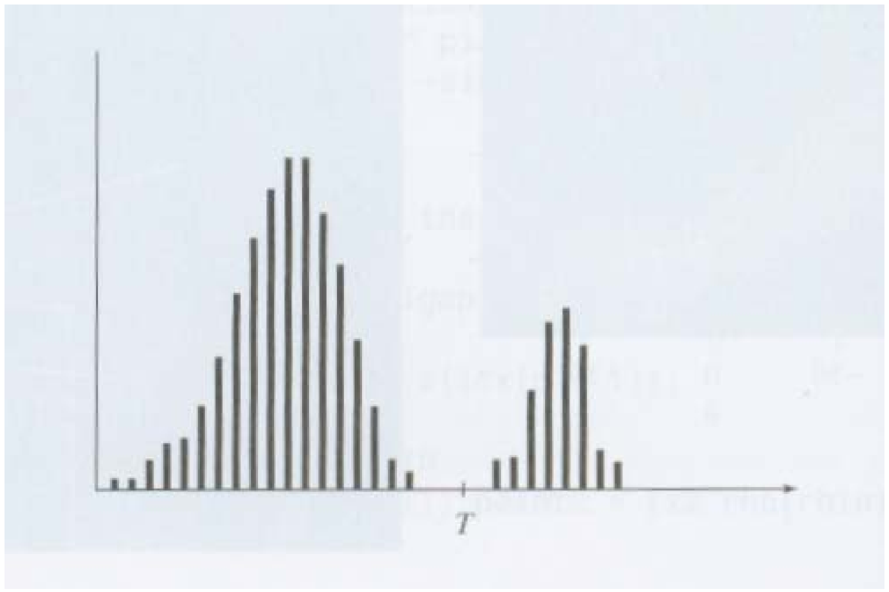


Figure 10.12: Selecting a threshold by visually analyzing a Bimodal histogram.
Gonzalez, Woods, & Eddins (2004), p 404.

$$g(x, y) = \begin{cases} 1 \quad if \;\; f(x, y) \geq T \rightarrow object \\ 0 \quad if \;\; f(x, y) < T \rightarrow background \end{cases}$$

  - When T is a constant, this approach is called global thresholding.

Global Thresholding

- Global thresholding applicable if:
  - Distinct intensity distributions for both background and object

- Iterative algorithm
  1. Select initial estimate for threshold *T*
  2. Segment image using *T*
     - Produces two pixel groups:
     - G_1: all pixel values > T
     - G_2: all pixel values < T
  3. Compute average intensity values:
     - M_1 for G_1
     - M_2 for G_2
  4. Compute new threshold $T = \frac{1}{2}(m_1 + m_2)$
  5. Repeat steps 2 through 4 until difference between, $\Delta T < S`$
     - Where S` is some predetermined value

- MATLAB Implementation

```
>> T = 0.5*( double(min(f(:))) + double (max(f(:))) );
>> done = false;
>> while –done
      g = f >= T;
      Tnext = 0.5 * (mean(f(g)) + mean (f(~g)));
      done = abs (T – Tnext) < 0.5;
      T = Tnext;
end
```

Otsu's Method: Optimum Global Thresholding

- Define the normalized histogram of an image as

$$p_r(r_q) = \frac{n_q}{n} \quad q = 0, 1, 2, \ldots, L-1$$

where n = total number of pixels in the image
n_q = number of pixels that have intensity level r_q
L = total number of possible intensity levels in the image

- Choose a threshold k such that C_0 is the set of pixels with levels [0, 1, …, k-1] and C_1 is the set of pixels with levels [k, k+1, …, L-1].

- Otsu's method chooses the threshold value k that maximizes the between-class variance, σ^2_B

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$

where

$$\omega_0 = \sum_{q=0}^{k-1} p_q(r_q) \qquad \omega_1 = \sum_{q=k}^{L-1} p_q(r_q)$$

$$\mu_0 = \sum_{q=0}^{k-1} q p_q(r_q)/\omega_0 \qquad \mu_1 = \sum_{q=k}^{L-1} q p_q(r_q)/\omega_1 \qquad \mu_T = \sum_{q=0}^{L-1} q p_q(r_q)$$

Performing Otsu’s Method in MATLAB

- MATLAB function, graythresh (), computes a threshold using Otsu’s method.

- Syntax

  T = graythresh (f)   where

  f = input image

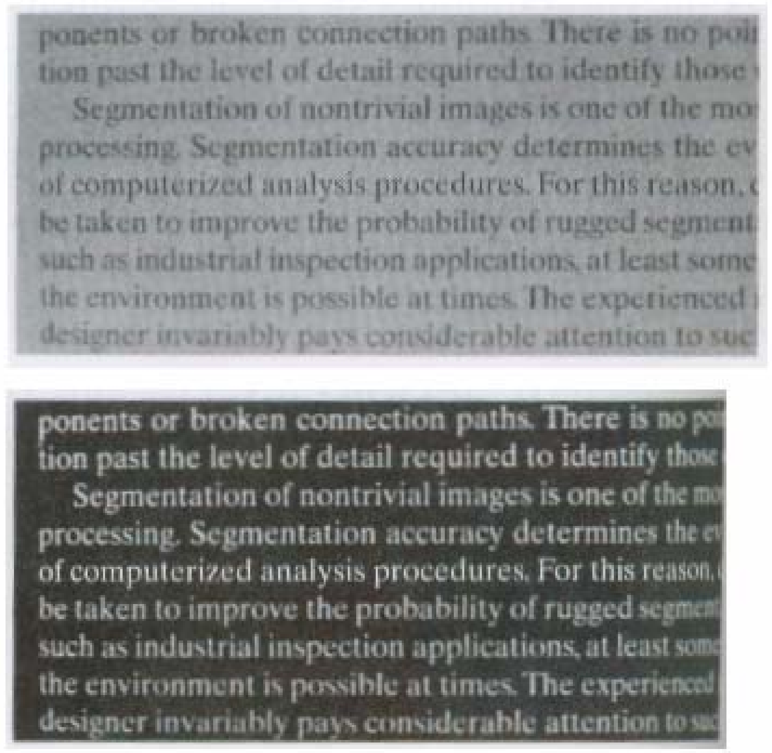
  T = the resulting threshold. It is a normalized value between 0.0 and 1.0.

- To segment the image, we use T in function im2bw (). Because the threshold is normalized to the range [0, 1], it must be scaled to the proper range before it is used.

- For example, consider the scanned text image to the right. The following code produces the thresholded image at the bottom.

```
>> T = graythresh (f)
>> T2 = T * 255
>> g = im2bw (f, T2)
```

Note: 255 was chosen above, since f is of class uint8.



ponents or broken connection paths. There is no poi
tion past the level of detail required to identify those
Segmentation of nontrivial images is one of the mo
processing. Segmentation accuracy determines the ev
of computerized analysis procedures. For this reason, c
be taken to improve the probability of rugged segment
such as industrial inspection applications, at least some
the environment is possible at times. The experienced
designer invariably pays considerable attention to suc

Figure 10.13: Top, scanned text. Bottom, thresholded text obtained using function graythresh.
Gonzalez, Woods, & Eddins (2004), p 406.

We acknowledge the following persons for helping to prepare this lecture:

- James Douglas Vasquez (UIC, Bioengineering)
- Gonzalez, R. C., Woods, R. E., and Eddins, S. L. [2004]. *Digital Image Processing Using MATLAB*, Prentice Hall, Upper Saddle River, NJ.
- Gonzalez, R. C. and Woods, R. E. [2008]. *Digital Image Processing*, *3rd ed*., Prentice Hall, Upper Saddle River, NJ.
- http://www.mathworks.com/support/functions/alpha_lis.html