Chapter 9:

Image Segmentation

9.1 Introduction

- **Segmentation** is the operation of partitioning an image into component parts or into separate objects.

- In this chapter, we will investigate two very important topics: **thresholding** and **edge detection**
  - Single & double thresholding
  - How to determine the threshold value
  - Adaptive thresholding
  - Edge detection: 1st and 2nd derivatives
  - Canny edge detector
  - Hough Transform

Single Thresholding

- grayscale image -> binary image

$$\text{A pixel becomes } \begin{cases} \text{white if its gray level is} > T, \\ \text{black if its gray level is} \leq T. \end{cases}$$

```
>> r=imread('rice.tif');
>> imshow(r),figure,imshow(r>110)
```

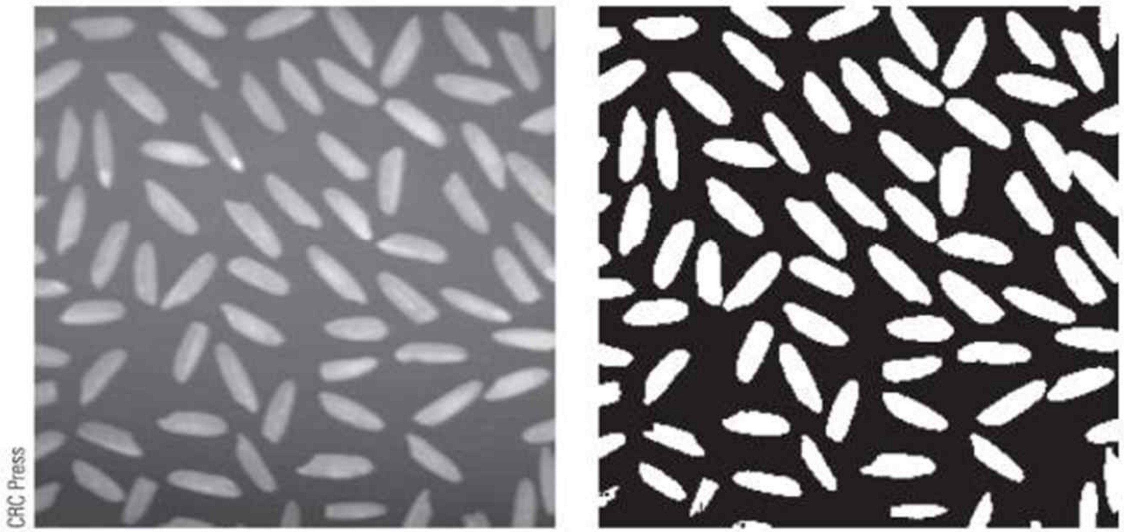


FIGURE 9.1 *Thresholded image of rice grains.*

Single Thresholding

```
>> b=imread('bacteria.tif');
>> imshow(b),figure,imshow(b>100)
```

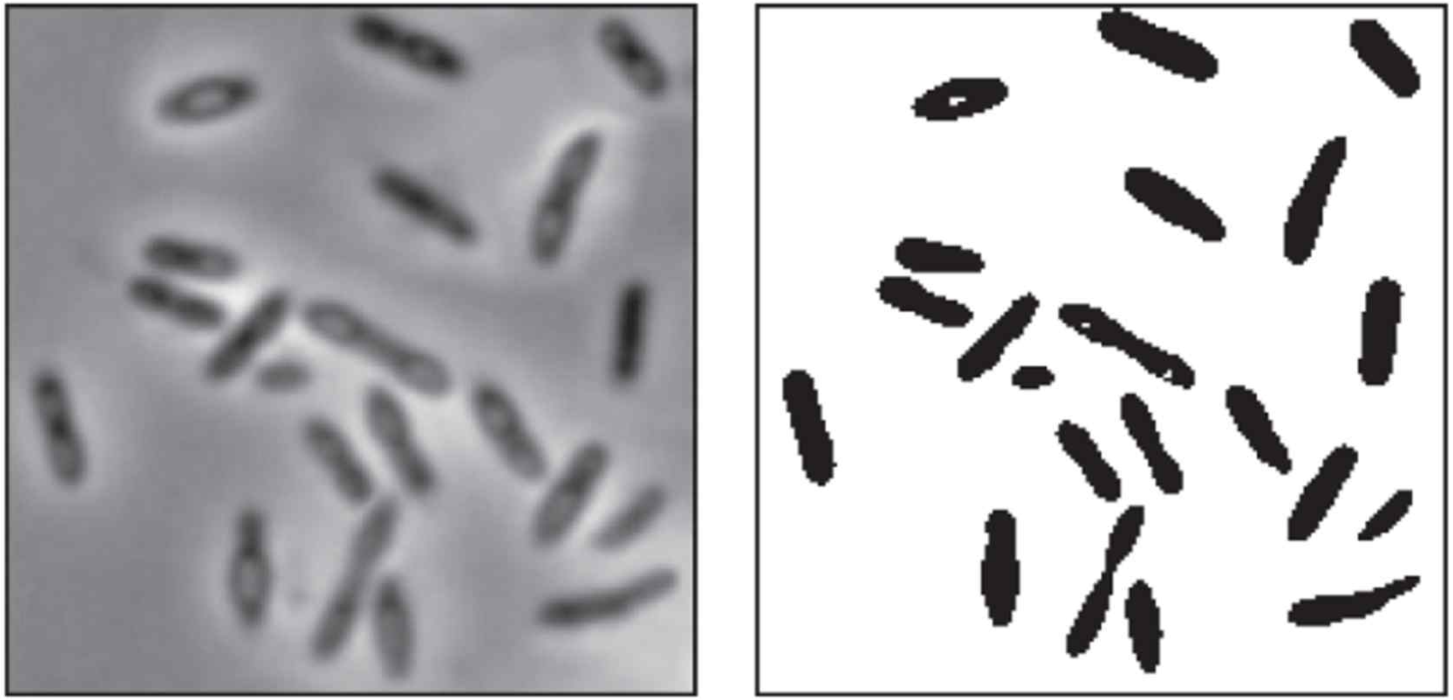


FIGURE 9.2 *Thresholded image of bacteria.*

Single Thresholding in Matlab: im2bw()

- MATLAB has the **im2bw** function, which thresholds an image, using the general syntax.
- It works on grayscale, colored, and indexed images of data type uint8, uint16, or double.

```
im2bw(image,level)
```

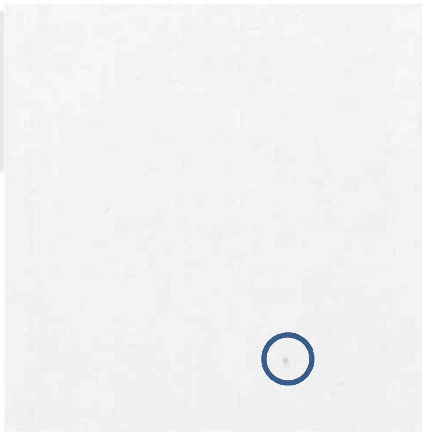
e.g.

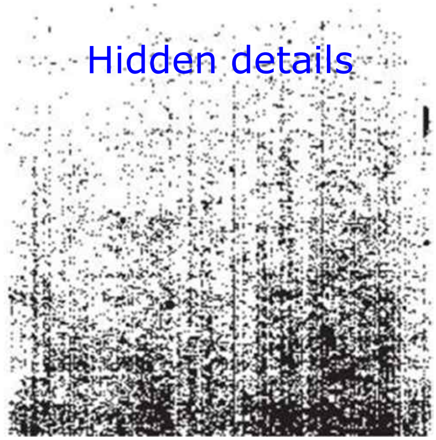
```
>> im2bw(r,0.43);
>> im2bw(b,0.39);
```

- choosing an appropriate value of T is important.

```
>> p=imread('paper1.tif');
>> imshow(p),figure,imshow(p>241)
```

Nearly all gray values are very high.





Double Thresholding

- grayscale image with two thresholds -> binary image

$$\text{a pixel becomes } \begin{cases} \text{white if its gray level is between } T_1 \text{ and } T_2, \\ \text{black if its gray level is otherwise.} \end{cases}$$

```
>> [x,map]=imread('spine.tif');
>> s=ind2gray(x,map);
>> imshow(s),figure,imshow(s>115 & s<125)
```

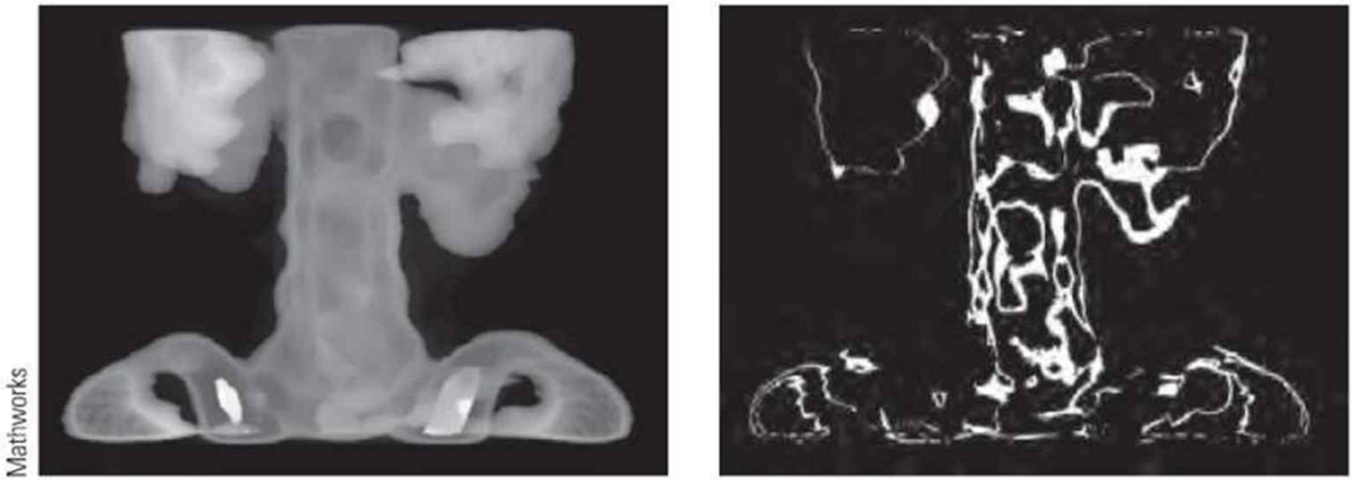


FIGURE 9.4 *The image* `spine.tif` *as the result after double thresholding.*

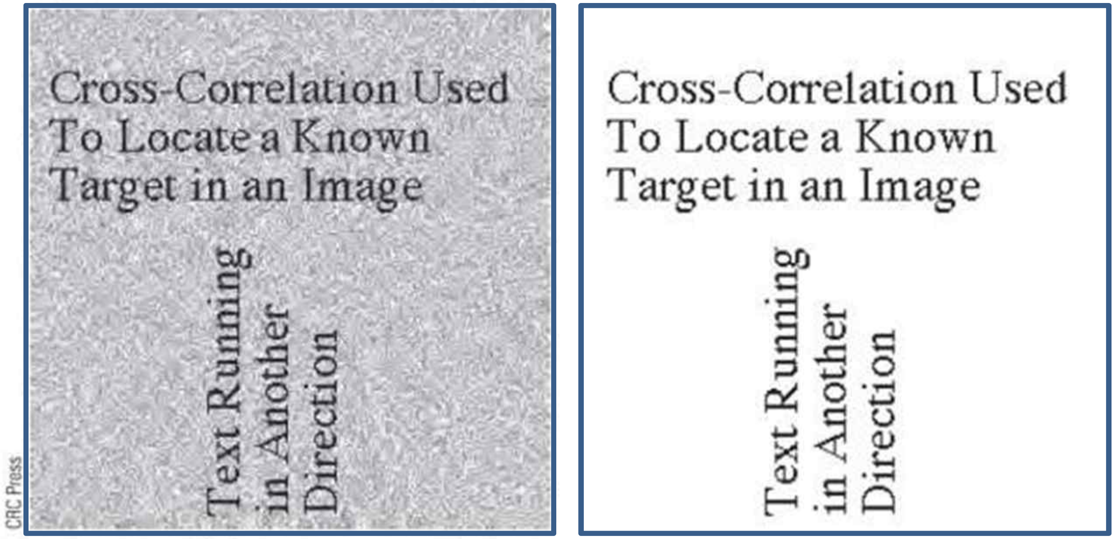
Applications of Thresholdng

1. Remove unnecessary detail: e.g. rice and bacteria images
2. bring out hidden detail : e.g. paper and spine images
3. remove a varying background from text or a drawing

```
>> r=rand(256)*128+127;
>> t=imread('text.tif');
>> tr=uint8(r.*double(not(t)));
>> imshow(tr)
```

```
>> imshow(tr>100)
```

text in noisy background

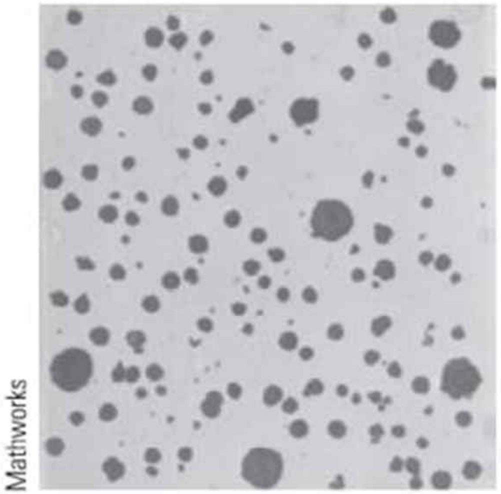


segmentation by single thresholding

FIGURE 9.5 *Text on a varying background and thresholding.*

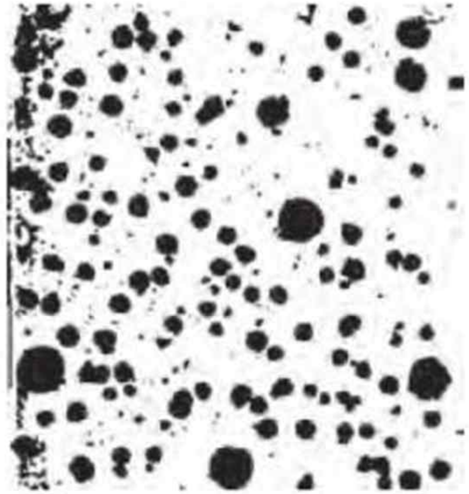
Choosing an Appropriate Thresholding Value

```
>> n=imread('nodules1.tif');
>> imshow(n);
>> n1=im2bw(n,0.35);
>> n2=im2bw(n,0.75);
>> figure,imshow(n1),figure,imshow(n2)
```



n: Original image

n1: Threshold too low



n2: Threshold too high

FIGURE 9.6 *Attempts at thresholding.*

Histogram Analysis

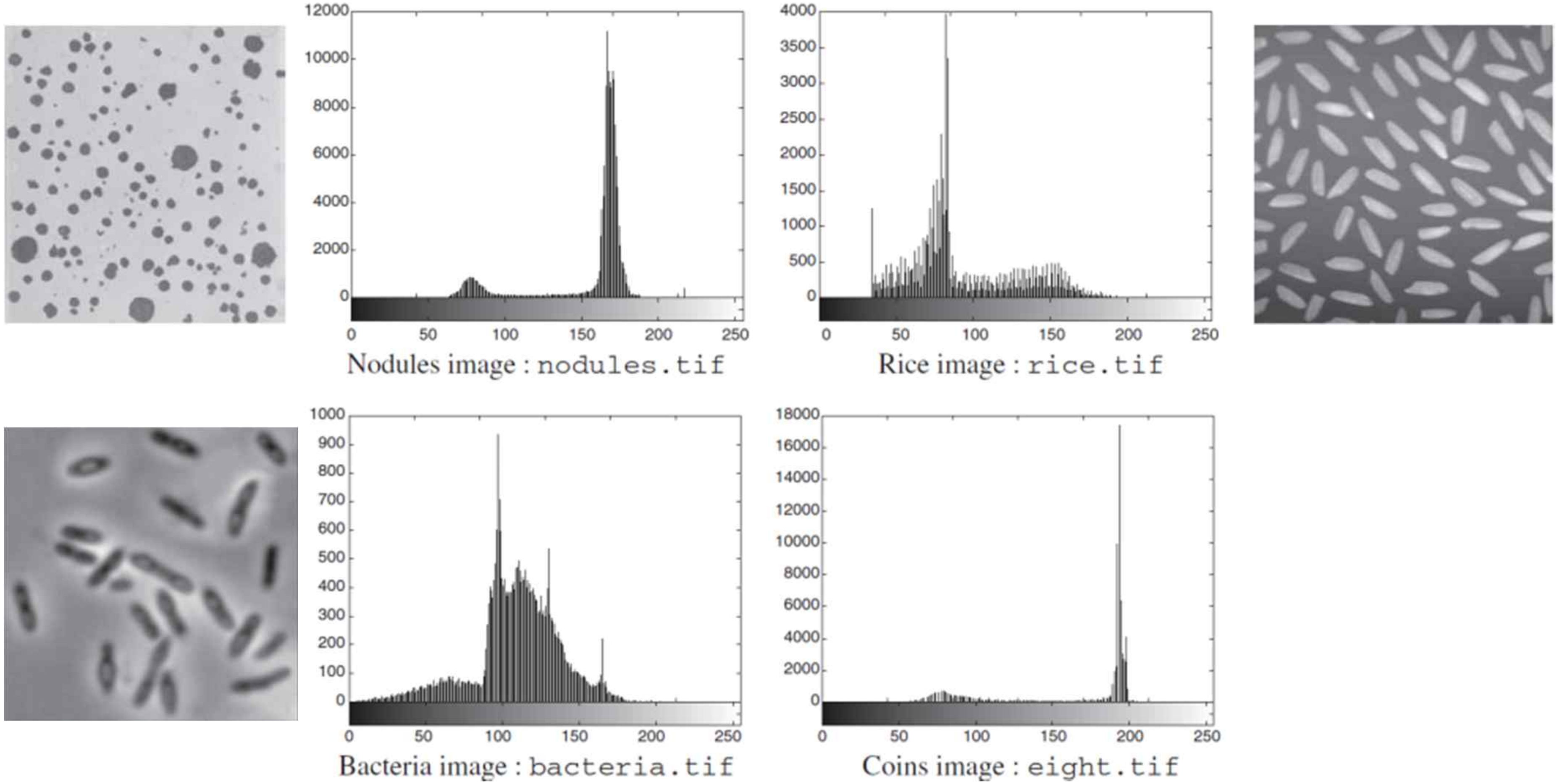


FIGURE 9.7 *Histograms.*

Histogram Analysis

- Choosing a threshold value is easy only if we have a prior knowledge of individual histograms of objects and background.
- However, in general, histogram of object and background is overlaped so that we cannot easily determine the appropriate threshold values.

-> needs more intelligent way to choose an optimal threshold.

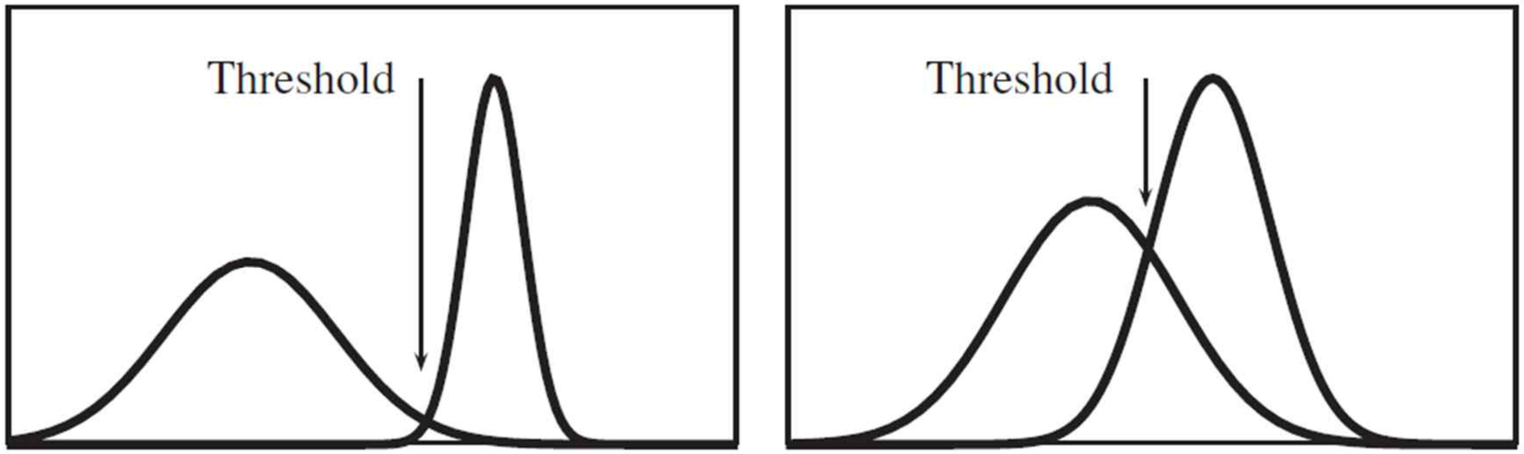


FIGURE 9.8 *Splitting up a histogram for thresholding.*

Otsu's Method: Choosing an Appropriate Thresholding Value

- Good classifier should be able to divide two classes so that each class has a small variance.
- method 1: Find threshold value t that makes sum of variances of two classes becomes minimum.

$$t = \arg\min_k \left[\omega_1(k)\,\mathrm{var}_1(k) + \omega_2(k)\,\mathrm{var}_2(k)\right]$$

$$\omega_1(k) = \sum_{i=0}^{k} p_i, \quad \omega_2(k) = \sum_{i=k+1}^{255} p_i$$

probability of pixel value i

minimizes within-class variance

- method 2(faster): Find threshold value t that makes difference of weighted average of two classes becomes maximum.

$$t = \arg\max_k \left[\omega_1(k)\omega_2(k)(\mu_1(k) - \mu_2(k))^2\right]$$

maximizes between-class variance

- Matlab command: `graythresh()`
- http://en.wikipedia.org/wiki/Otsu's_method

Example of Otsu's Method

```
>> tn=graythresh(n)

tn =

    0.5804

>> r=imread('rice.tif');
>> tr=graythresh(r)

tr =

    0.4902

>> b=imread('bacteria.tif');
>> tb=graythresh(b)

tb =

    0.3765

>> e=imread('eight.tif');
>> te=graythresh(e)

te =

    0.6490
```

```
>> imshow(im2bw(n,tn))
>> figure,imshow(im2bw(r,tr))
>> figure,imshow(im2bw(b,tb))
>> figure,imshow(im2bw(e,te))
```

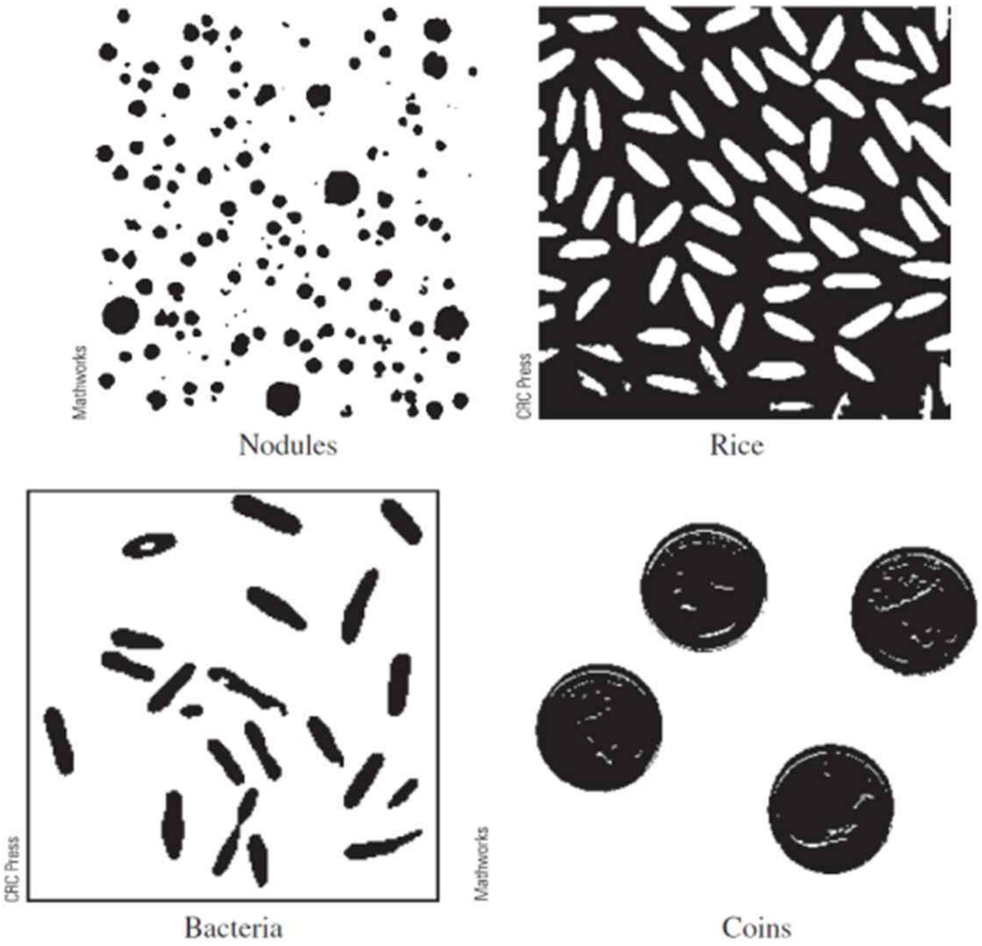


FIGURE 9.9 *Thresholding with values from* `graythresh`.

Otsu’s Method: Assumptions

- Histogram (and the image) are *bimodal.*
- No use of *spatial coherence,* nor any other notion of object structure.
- Assumes uniform illumination (implicitly), so the bimodal brightness behavior arises from object appearance differences only.

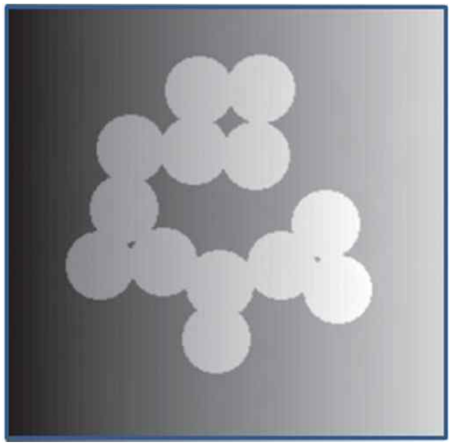
구간을 나눠서 어떠한 작업을 하는걸 adaptive(적응형) 라고 한다

Adaptive Thresholding

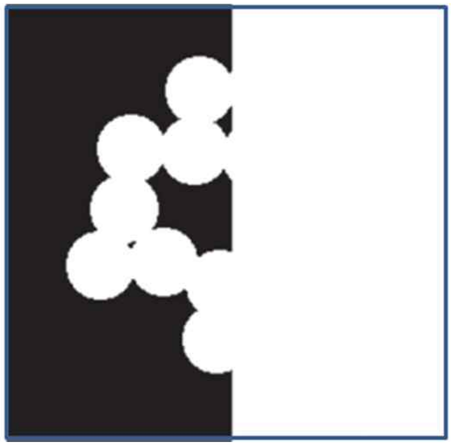
- Sometimes a single global threshold values (even if it was extracted by Otsu's method) does not work.
- Needs to determine the threshold values in each image block depending on the characteristic of each block.

  -> adaptively thresholding

```
>> c=imread('circles.tif');
>> x=ones(256,1)*[1:256];
>> c2=double(c).*(x/2+50)+(1-double(c)).*x/2;
>> c3=uint8(255*mat2gray(c2));
```



c3 from circles.tif



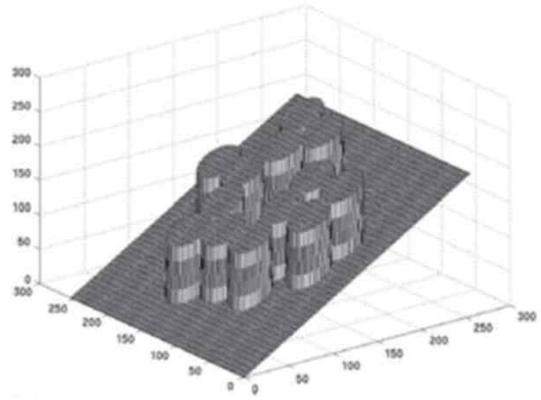
global thresholding

```
>> t=graythresh(c3)

t =

    0.4196

>> ct=im2bw(c3,t);
```

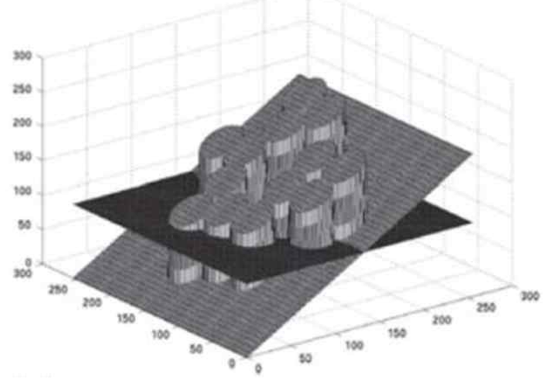
Adaptive Thresholding

Problem determination by 3D surface plots

surface plot of c3



global threshold (black plane) onto c3



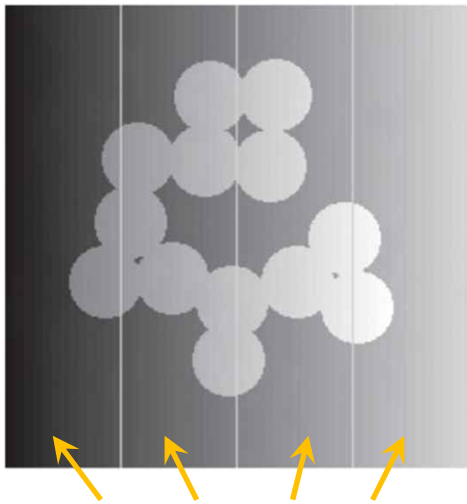
solution: apply blockwise thresholding adaptively

`blkproc` and `im2bw`: pp233

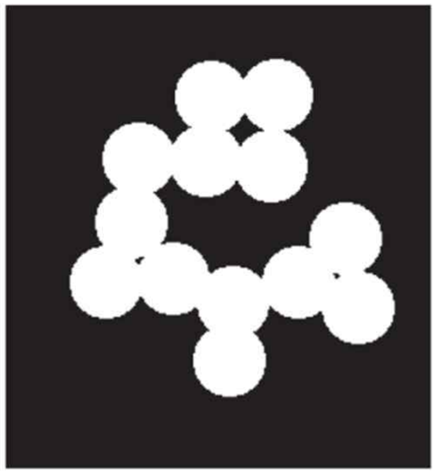
```
>> p1=c3(:,1:64);
>> p2=c3(:,65:128);
>> p3=c3(:,129:192);
>> p4=c3(:,193:256);
```

```
>> g1=im2bw(p1,graythresh(p1));
>> g2=im2bw(p2,graythresh(p2));
>> g3=im2bw(p3,graythresh(p3));
>> g4=im2bw(p4,graythresh(p4));
```

```
>> imshow([g1 g2 g3 g4])
```



4 sub-blocks



Edge Detection in Matlab

- The general MATLAB command for finding edges is

`edge(image,'method',parameters. . .)`

51	52	53	59
54	52	53	62
50	52	53	68
55	52	53	55

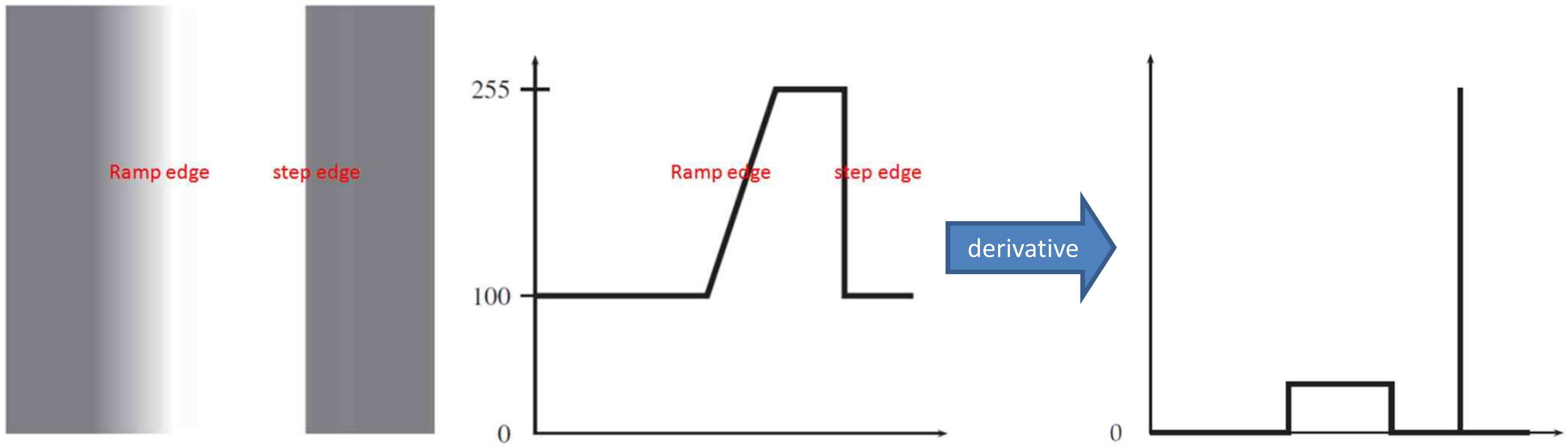
(a)

50	53	155	160
51	53	160	170
52	53	167	190
51	53	162	155

(b)

FIGURE 9.13 *Blocks of pixels.*

Derivatives and Edges



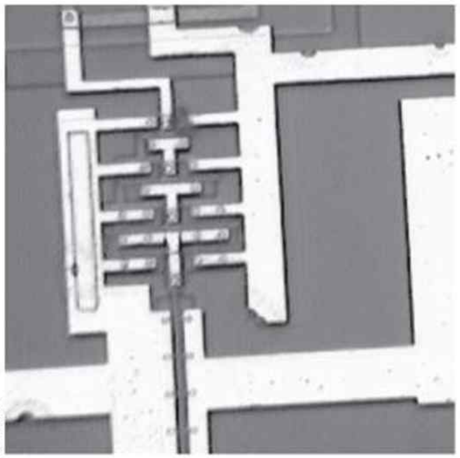
- derivative f'(x)=df/dx
- discrete version: f(x+1)-f(x), f(x)-f(x-1), (f(x+1)-f(x-1))/2
- Expansion into 2D image -> gradient

$$\begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix} \xrightarrow{\text{magnitude}} \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

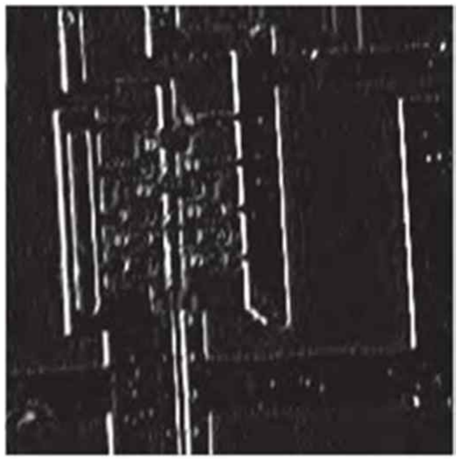
Edge Detection Filters: Prewitt

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

```
>> ic=imread('ic.tif');
```

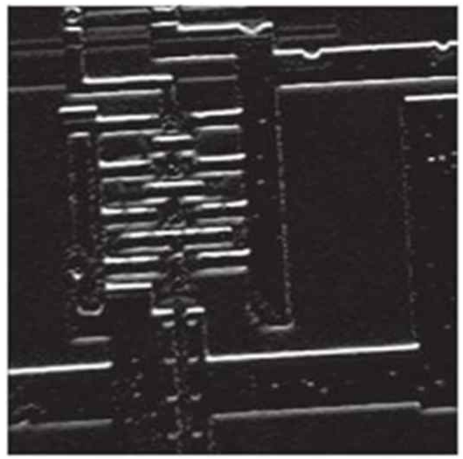



```
>> px=[-1 0 1;-1 0 1;-1 0 1];
>> icx=filter2(px,ic);
>> figure,imshow(icx/255)
```



Prewitt filter: Px

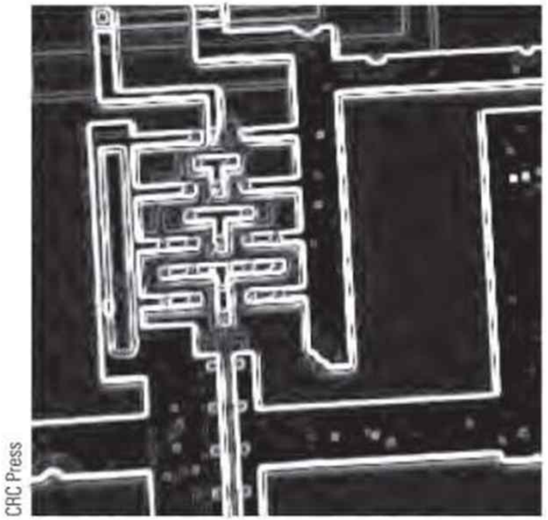
```
>> py=px';
>> icy=filter2(py,ic);
>> figure,imshow(icy/255)
```



Prewitt filter: Py

Edge Detection by Prewitt Filter

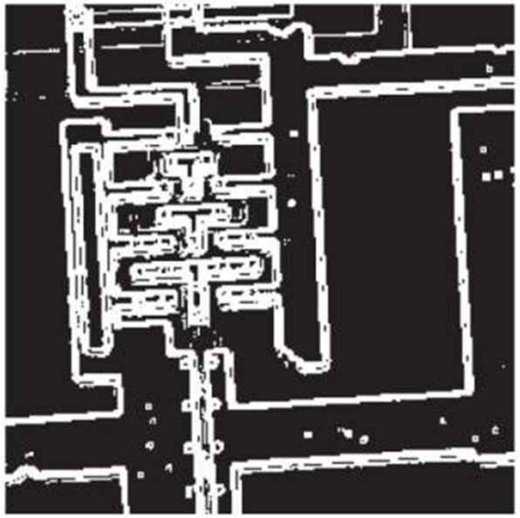
magnitude of gradient images (icx and icy) -> gray scale



(a)

```
>> edge_p=sqrt(icx.^2+icy.^2);
>> figure,imshow(edge_p/255)
```

Thresholding magnitude of gradient images -> binary



(b)

```
>> edge_t=im2bw(edge_p/255,0.3);
```

edge() in matlab: Prewitt magnitude + thresholding + some extra processing

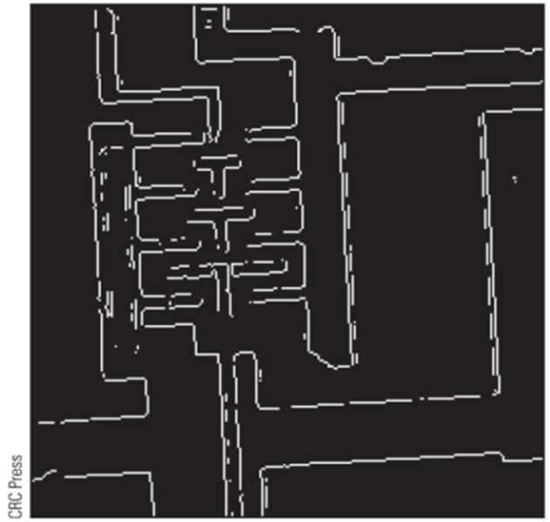


FIGURE 9.19 *The* `prewitt` *option of* `edge`.

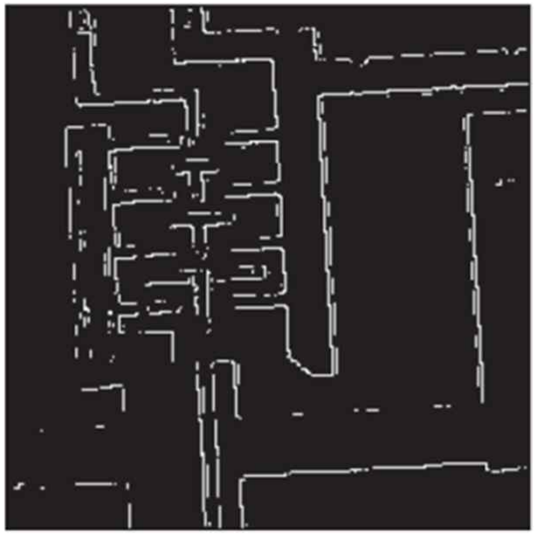
```
>> edge_p=edge(ic,'prewitt');
```

More Edge Detection Filters in Matlab

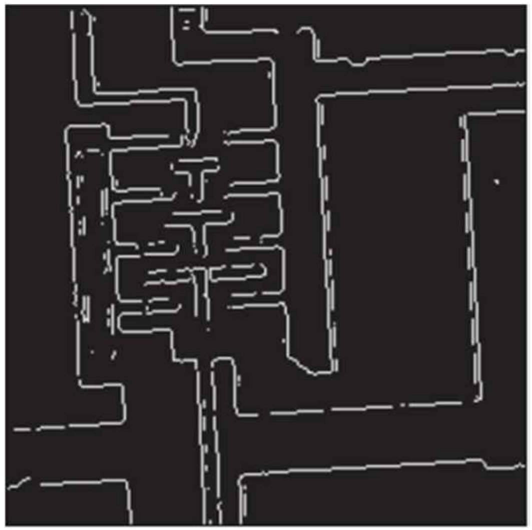
- Roberts cross-gradient filters: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

- Sobel filters: $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

```
>> edge_r=edge(ic,'roberts');
>> figure,imshow(edge_r)
```



```
>> edge_s=edge(ic,'sobel');
>> figure,imshow(edge_s)
```

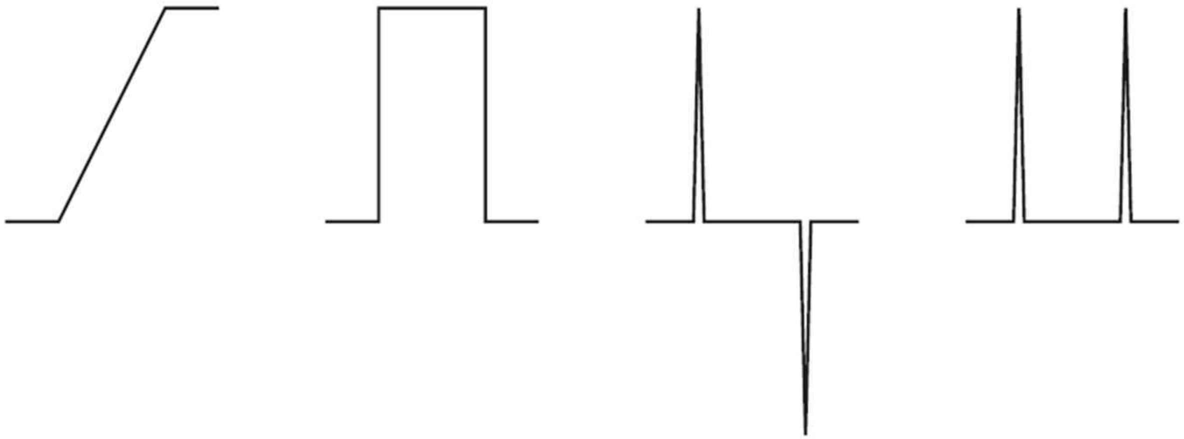


Second Derivatives for Edge Detection

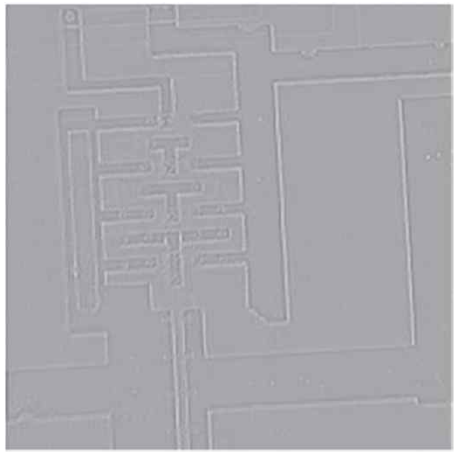
- The Laplacian: $\nabla^2 f = \dfrac{\partial^2 f}{\partial x^2} + \dfrac{\partial^2 f}{\partial y^2} \xrightarrow{\text{discrete version}} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

  ✓ Pros: isotropic(= rotation invariant) filter

  ✓ Cons: very sensitive to noise



The edge First derivative Second derivative Absolute values



```
>> l=fspecial('laplacian',0);
>> ic_l=filter2(l,ic);
>> figure,imshow(mat2gray(ic_l))
```

Edge Detection by Zero Crossing

- The position where the result of the filter **changes sign**.
- E.g., the simple image given below and the result after filtering with a Laplacian mask 이런 방식들은 노이즈에 너무 취약함
- What if we simply take all zero crossing points as edges ?

50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50
50	50	200	200	200	200	200	200	50	50
50	50	200	200	200	200	200	200	50	50
50	50	200	200	200	200	200	200	50	50
50	50	200	200	200	200	200	200	50	50
50	50	50	50	200	200	200	200	50	50
50	50	50	50	200	200	200	200	50	50
50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50

(a)

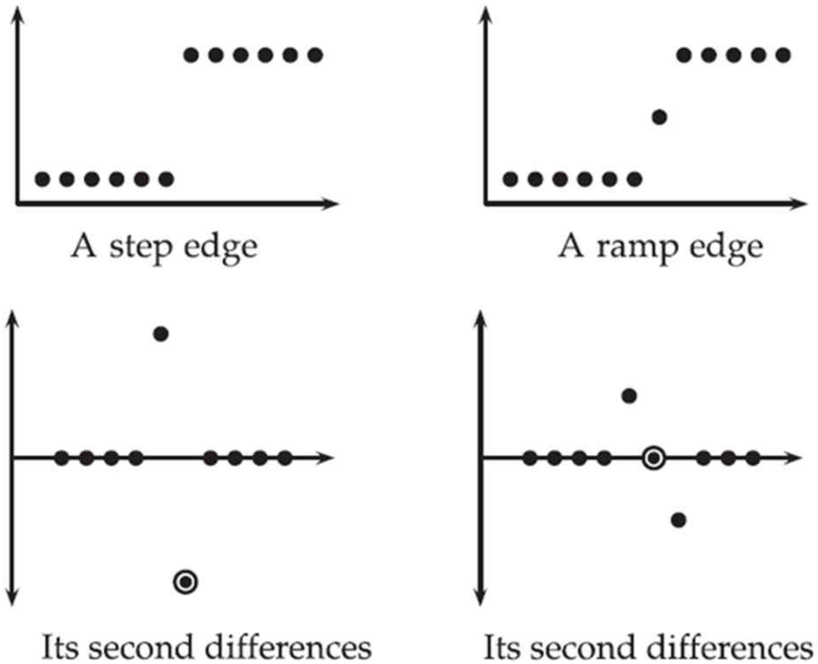
−100	−50	−50	−50	−50	−50	−50	−50	−50	−100
−50	0	150	150	150	150	150	150	0	−50
−50	150	−300	−150	−150	−150	−150	−300	150	−50
−50	150	−150	0	0	0	0	−150	150	−50
−50	150	−150	0	0	0	0	−150	150	−50
−50	150	−300	−150	0	0	0	−150	150	−50
−50	0	150	300	−150	0	0	−150	150	−50
−50	0	0	150	−300	−150	−150	−300	150	−50
−50	0	0	0	150	150	150	150	0	−50
−100	−50	−50	−50	−50	−50	−50	−50	−50	−100

(b)

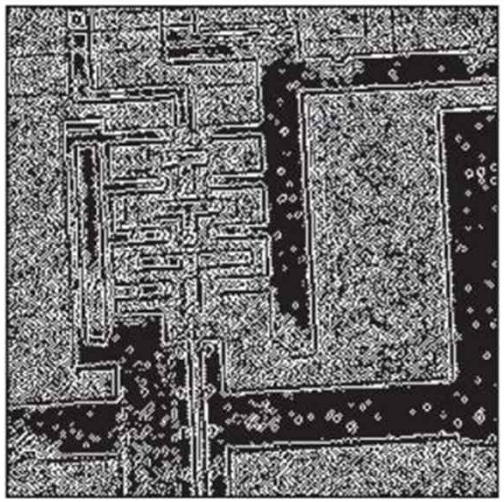
FIGURE 9.23 *Locating zero crossings in an image. (a) A simple image. (b) After laplace filtering.*

Edge Detection by Zero Crossing

- Two cases of zero crossing
  - ✓ They have a negative gray value and are orthogonally adjacent to a pixel whose gray value is positive.
  - ✓ They have a value of zero and are between negative- and positive-valued pixels




```
>> l=fspecial('laplace',0);
>> icz=edge(ic,'zerocross',l);
>> imshow(icz)
```



Too many edges are detected.
How can we improve the results?
-> Marr-Hildreth Method

Edge Detection by Zero Crossing

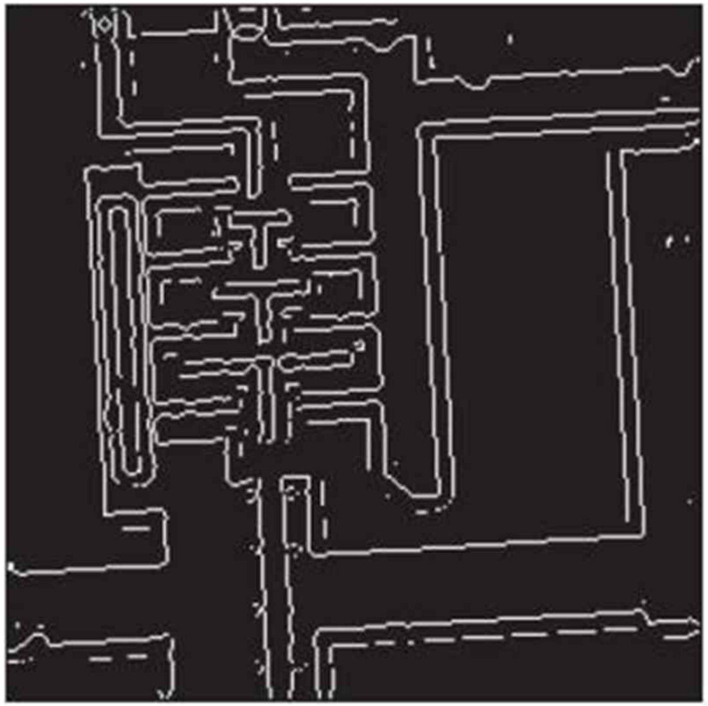
- **Marr-Hildreth** method
  - ✓ Smooth the image with a Gaussian filter.
  - ✓ Convolve the result with a Laplacian filter.
  - ✓ Find the zero crossings.

Laplacian of Gaussian (LoG)

```
>> fspecial('log', 13, 2);
>> edge(ic, 'log');
```

or

```
>> log=fspecial('log',13,2);
>> edge(ic,'zerocross',log);
```



Canny Edge Detector

- Proposed by John Canny in 1986.
- The most popular edge filter ever !
- Stage 1: 1D DoG filters -> result: xe(x,y)
  - ✓ Take our image x
  - ✓ Create a one-dimensional Gaussian filter g
  - ✓ Create a 1D filter dg corresponding to the expression given in

$$\boxed{\frac{d}{dx}e^{-\frac{x^2}{2\sigma^2}} = \left(-\frac{x^2}{\sigma^2}\right)e^{-\frac{x^2}{2\sigma^2}}}$$

- derivative of Gaussian
- to smooth the noise and find possible candidate pixels for edges
- separable (1D filtering twice)

  - ✓ Convolve g with dg to obtain gdg
  - ✓ Apply gdg to x producing $x1$: horizontal operation
  - ✓ Apply gdg' to x producing $x2$: vertical operation
  - ✓ Form an magnitude of edge xe with the equation: gradient

$$\mathrm{xe} = \sqrt{\mathrm{x1}^2 + \mathrm{x2}^2}$$

Canny Edge Detector

- Stage 2: Non-maxima suppression -> result: $g_N(x,y)$
  - Rejects pixels detected as edges in the Stage 1 by detecting the maximum derivative among adjacent pixels.
  - Thresholding alone is not a good way to detection edges in the result *xe*.

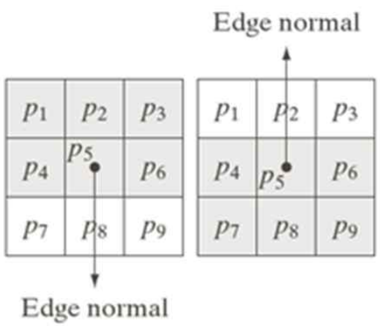
✓Calculate edge normal(gradient vector) *xg* from the 1D gradient images *x1* and *x2*.

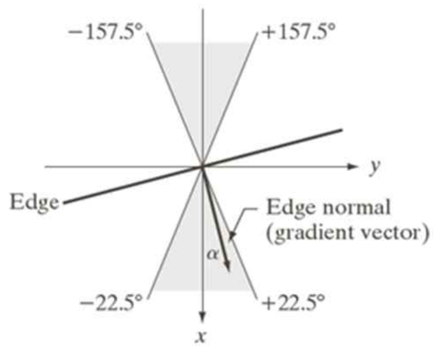
$$\mathrm{xg} = \tan^{-1}\left(\frac{\mathrm{x2}}{\mathrm{x1}}\right)$$

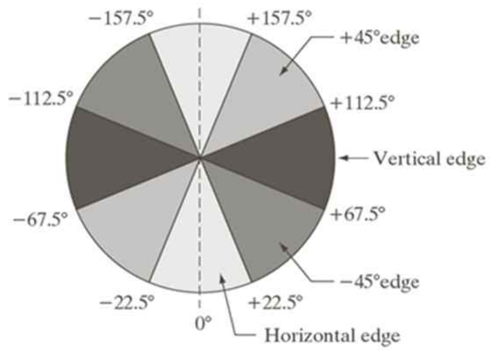
✓Because it is generated using the gradient, xg typically contains wide ridges around local maxima. Thus, we needs to thin those ridges.

Based on xg, quantize edge orientations into the predetermined steps. (e.g. four orientations in 3x3 region: horizontal, vertical, +45°, -45°)

✓If a gradient xe(x,y) is less than at least one of its two neighbors along an edge normal xg_i, let $g_N(x,y) = 0$ (suppression); otherwise, let $g_N(x,y) = xe(x,y)$.







Canny Edge Detector

- Stage 3: Hysteresis thresholding for binary edge image →result: $g_{NH}(x,y)$
  - uses two threshold values: T_L and T_H for lower and higher bounds respectively.
  - ✓ $g_{NH}(x,y) = g_N(x,y) >= T_H$: strong edge pixels
  - ✓ $g_{NL}(x,y) = g_N(x,y) >= T_L$
  - ✓ $g_{NL}(x,y) = g_{NL}(x,y) - g_{NH}(x,y)$: weak edge pixels
  - ✓ Connectivity analysis in $g_{NH}(x,y)$
    1. Locate the next unvisited edge pixel, p, in $g_{NH}(x,y)$.
    2. Mark as valid edge pixels all the weak pixels in $g_{NL}(x,y)$ that are connected to p using 8 connectivity.
    3. If all nonzero pixels in $g_{NH}(x,y)$ have been processed, go to the next step. Otherwise, return to the first step.
    4. Set to 0 all pixels in $g_{NL}(x,y)$ that were not marked as valid edge pixels.
  - ✓ Append all the nonzero pixels from $g_{NL}(x,y)$ to $g_{NH}(x,y)$.

Canny Edge Detector in Matlab

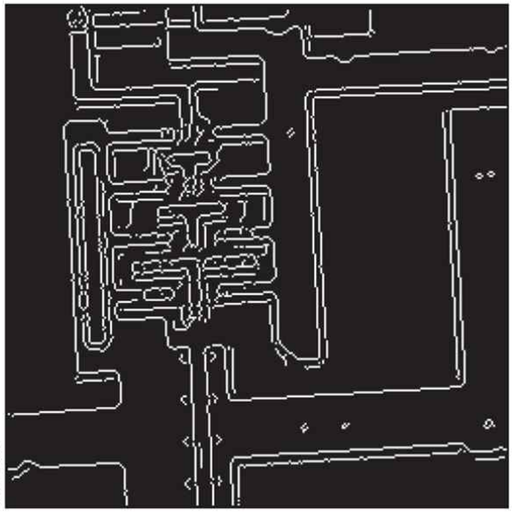
```
>> [icc,t]=edge(ic,'canny');
>> t

t =

    0.0500    0.1250

>> imshow(icc)
```

- BW = EDGE(I,'canny',[low, high],SIGMA)

threshold limits for Stage 3

standard deviation for Stage 1

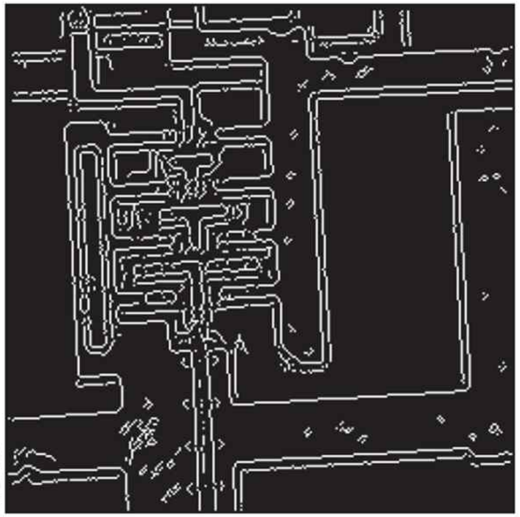
- The higher we make the upper threshold, the fewer edges will be shown.



CRC Press

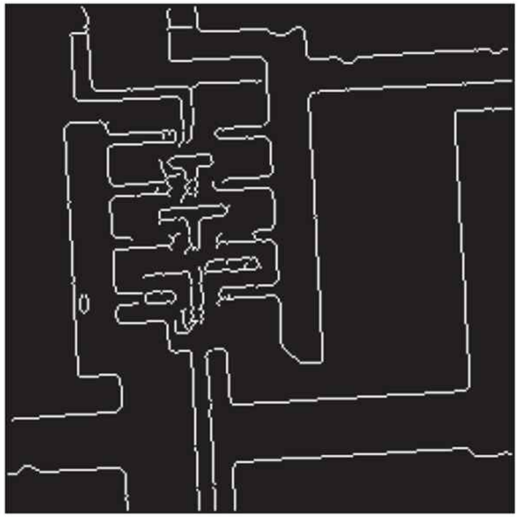
FIGURE 9.29 *Canny edge detection.*

default threshold and standard deviation



CRC Press

edge(ic,'canny',[0,0.05])



edge(ic,'canny',[0.01,0.5])

FIGURE 9.30 *Canny edge detection with different thresholds.*

영상이 매우 선명한 edge들로 구성되어 있을 때 사용

Hough Transform

- If the detected edge points are sparse, we might need to fit a line to those points. However it is a time-consuming and computationally inefficient process.
- Is there any way to find boundary lines to compensate the problem described above ? => Hough Transform
- Basic idea of Hough Transform
  - A coordinate (x,y) in an image can be transformed into a line y = ax + b with all possible pairs of (a,b).
  - Every pixel at (x,y) coordinate in the image can be mapped into a line in the transform array.

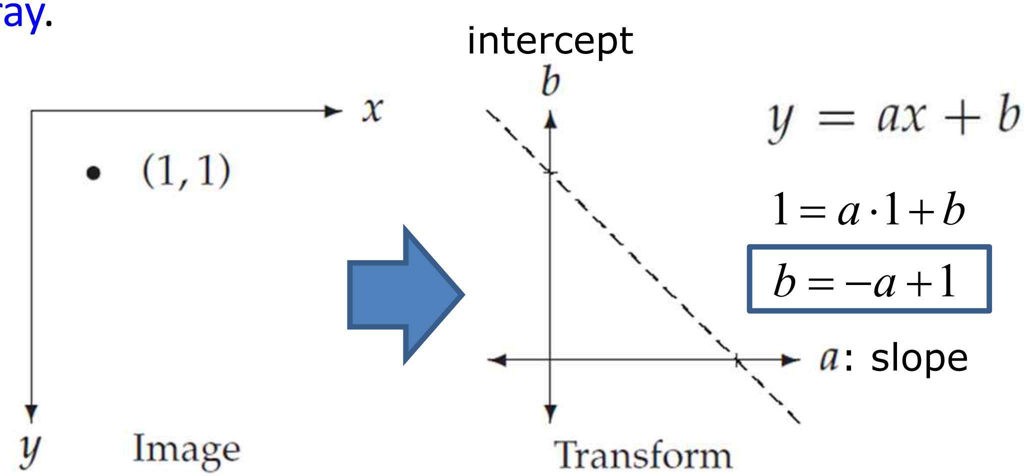


FIGURE 9.31 *A point in an image and its corresponding line in the transform.*

Basic Idea of Hough Transform

5 pixels are transformed into 5 lines

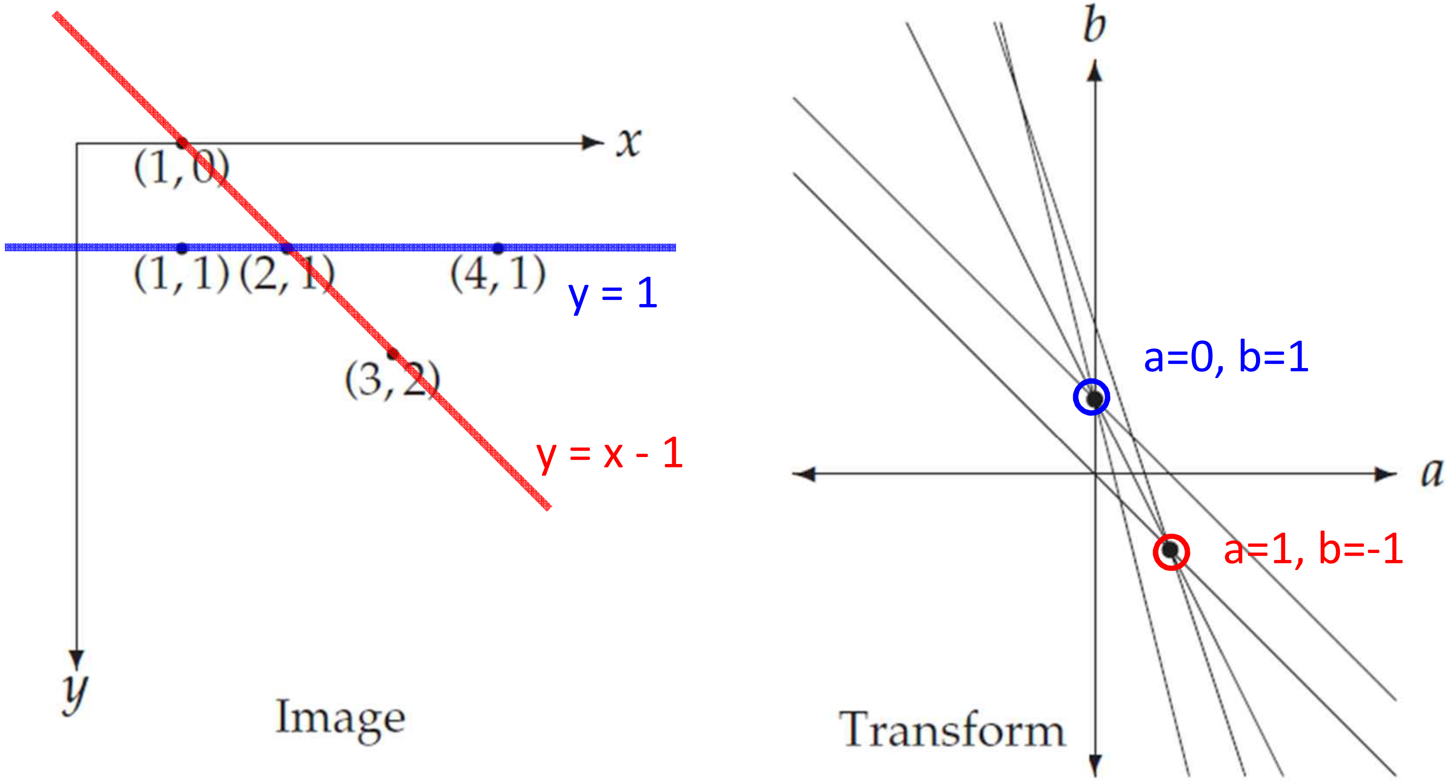
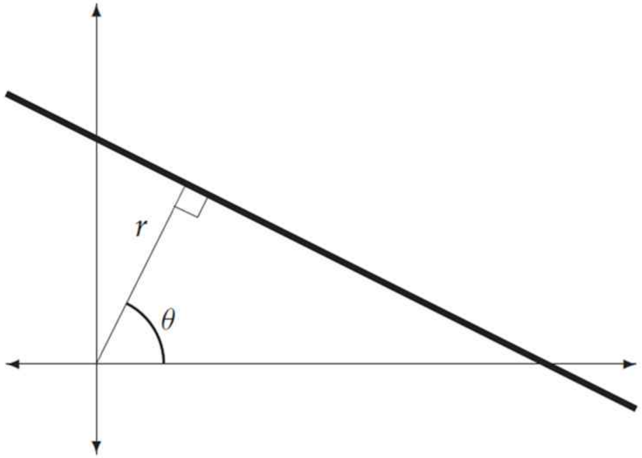


FIGURE 9.32 *An image and its corresponding lines in the transform.*

- Higher value of cross point (a,b) in the transformed array indicates more distinguished lines (more pixels through the line) in the image !

Hough Transform for Generalization

- We cannot express a vertical line in the form $y = mx+c$, because m represents the gradient and a vertical line has infinite gradient.
- Any line can be described in terms of the two parameters r and θ.
  - ✓ r is the perpendicular distance from the line to the origin
  - ✓ θ is the angle of the line's perpendicular to the x axis



$$-90 < \theta \leq 90$$

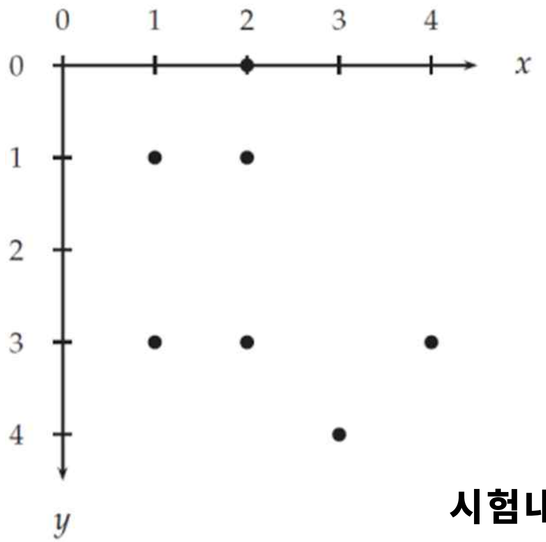
$$\boxed{x\cos\theta + y\sin\theta = r}$$

We can calculate θ and r based on the image coordinate (x,y).
See pp.253 for the derivation.

- Higher value of (r,θ) in the transformed array indicates stronger lines (more pixels through the line) in the image.

Hough Transform Example

- If we discretize θ to use only four values: -45°, 0°, 45°, 90°



시험내기 좋음

(x, y)	$-45°$	$0°$	$45°$	$90°$
$(2,0)$	1.4	2	1.4	0
$(1,1)$	0	1	1.4	1
$(2,1)$	0.7	2	2.1	1
$(1,3)$	-1.4	1	2.8	3
$(2,3)$	-0.7	2	3.5	3
$(4,3)$	0.7	4	4.9	3
$(3,4)$	-0.7	3	4.9	4

θ

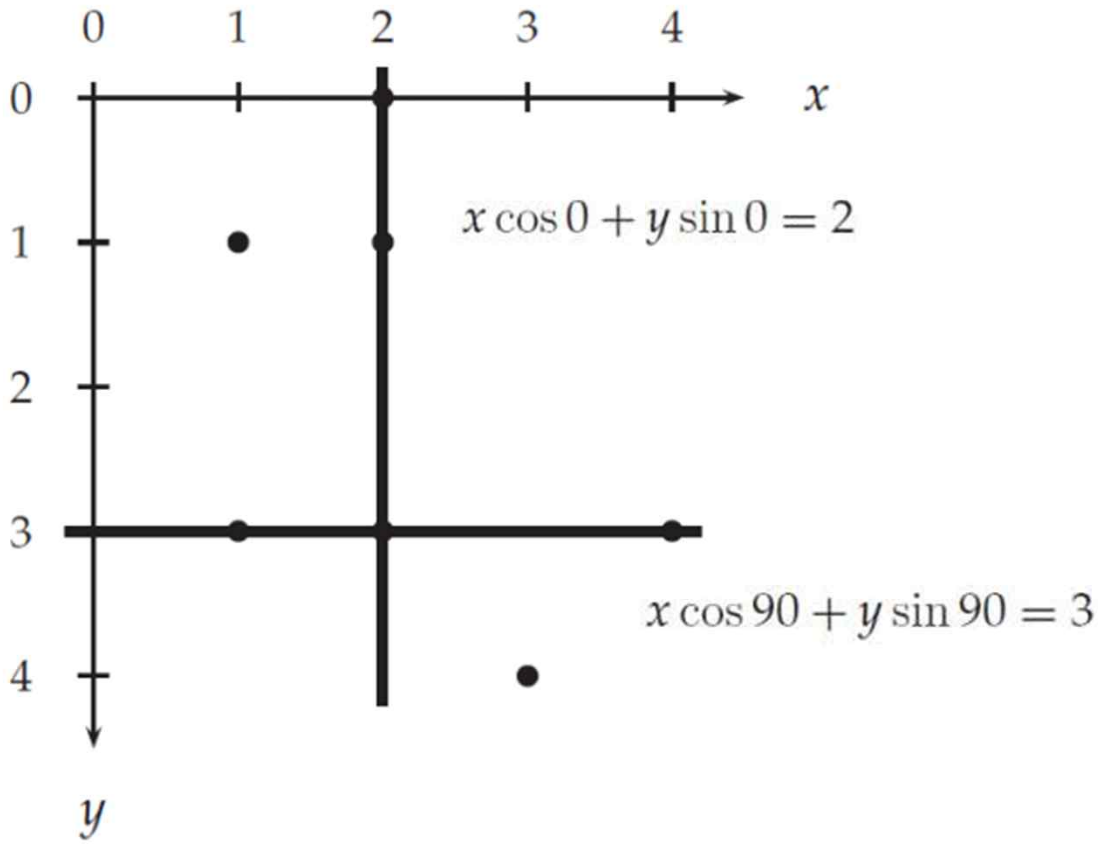
r

$\boxed{x\cos\theta + y\sin\theta = r}$

- A single (x,y) coordinate generates four pairs of (r, θ) in this example.
- The accumulator array contains the number of times each value of (r, θ) appears in the above table.

	-1.4	-0.7	0	0.7	1	1.4	2	2.1	2.8	3	3.5	4	4.9
$-45°$	1	2	1	2		1							
$0°$					2		3			1		1	
$45°$						2		1	1		1		2
$90°$			1		2					3		1	

Hough Transform Example



Detected lines

Implementing Hough Transform in MATLAB

1. Decide on a discrete set of values of θ and r to use.
2. At each foreground pixel (x, y) in the image, calculate, the values $r = xcos\theta + ysin\theta$ for all θ.
3. Create an accumulator array whose sizes are the numbers of angles θ and values r in the chosen discretizations from Step 1, and
4. Step through all of our r values, updating the accumulator array as we go.

Hough Transform in MATLAB

- Discretizing r and θ
  - Let θ with the range of 0 <= θ < 2π.
  - Restrict r to non-negative value.

```
>> angles = [-90:180]*pi/180;
```

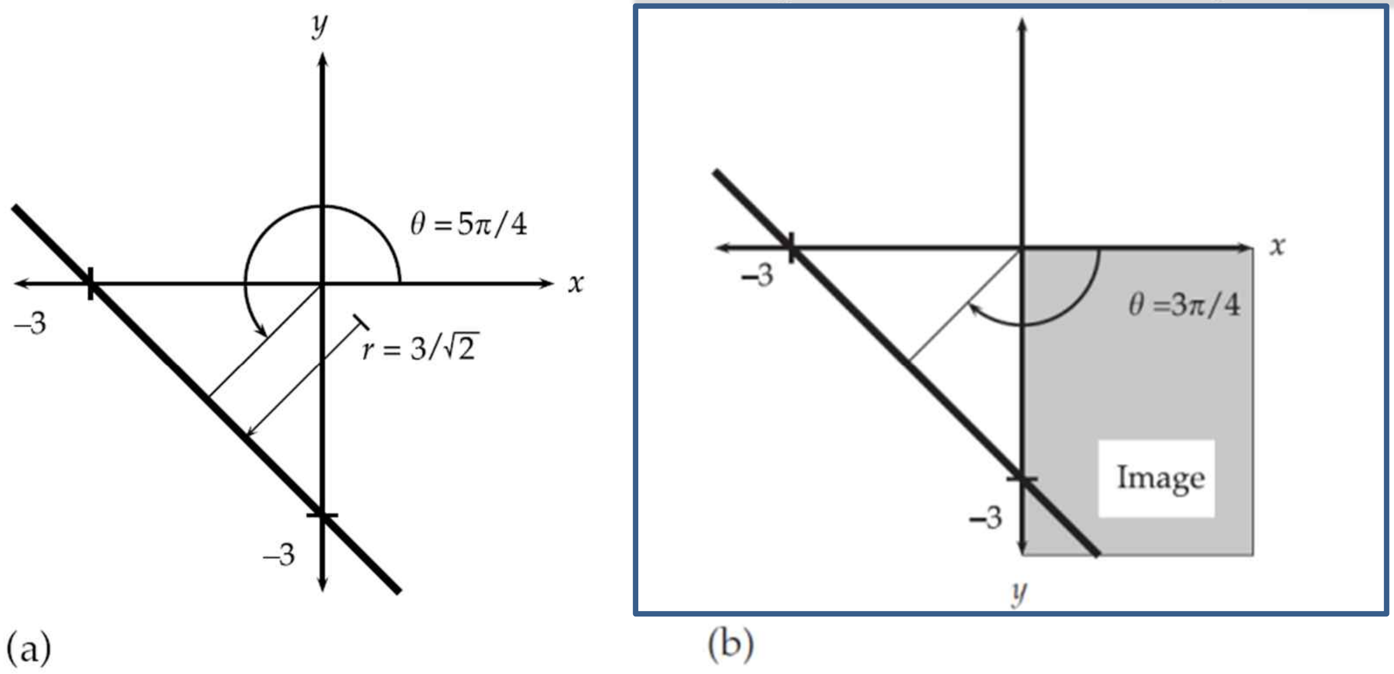


FIGURE 9.37 *A line parameterized with r and θ. (a) Using ordinary Cartesian axes. (b) Using matrix axes.*

Example of Hough Transform in MATLAB

- Calculating the r values
  - ✓ If im is a **binary image**

```
>> [x,y]=find(im);
```

We can create a binary edge image by use of the `edge` function beforehand.

```
>> r=floor(x*cos(angles)+y*sin(angles));
```

- Forming the Accumulator Array

```
>> rmax=max(r(find(r>0)));
>> acc=zeros(rmax+1,270);
```

- Updating the Accumulator Array

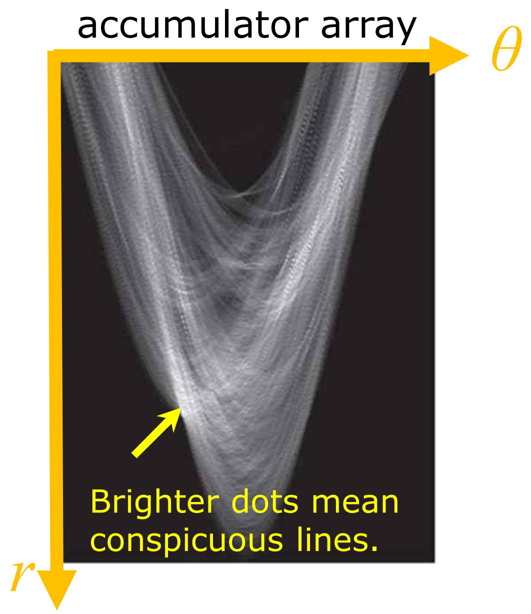
```
function res=hough2(image)

%
% HOUGH2(IMAGE) creates the Hough transform corresponding to the image IMAGE
%

if ~isbw(image)
  edges=edge(image,'canny');
else
  edges=image;
end;
[x,y]=find(edges);
angles=[-90:180]*pi/180;
r=floor(x*cos(angles)+y*sin(angles));
rmax=max(r(find(r>0)));
acc=zeros(rmax+1,270);
for i=1:length(x),
  for j=1:270,
    if r(i,j)>=0
      acc(r(i,j)+1,j)=acc(r(i,j)+1,j)+1;
    end;
  end;
end;
res=acc;
```

FIGURE 9.38 *A simple MATLAB function for implementing the Hough transform.*

Example of Hough Transform in MATLAB

```
>> c=imread('cameraman.tif');
>> hc=hough2(c);
```

```
>> imshow(mat2gray(hc)*1.5)
```




```
>> max(hc(:))

ans =

    91
```

```
>> [r,theta]=find(hc==91)

r =

   138


theta =

   181
```

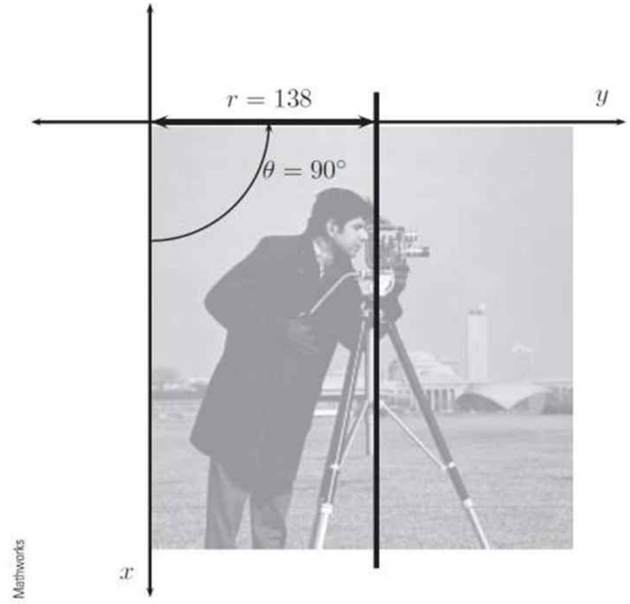


FIGURE 9.40 *A line from the Hough Transform.*

How to Display the Detected Line in Matlab

```
function houghline(image,r,theta)
%
% Draws a line at perpendicular distance R from the upper left corner of the
% current figure, with perpendicular angle THETA to the left vertical axis.
% THETA is assumed to be in degrees.
%
[x,y]=size(image);
angle=pi*(181-theta)/180;
X=[1:x];
if sin(angle)==0
  line([r r],[0,y],'Color','black')
else
  line([0,y],[r/sin(angle),(r-y*cos(angle))/sin(angle)],'Color','black')
end;
```

(a) (b)

See pp. 261 for more detailed information to use houghline().

Summary

- **Chap 9. Image Segmentation**
  - Single & double thresholding
  - How to determine the threshold value
  - Adaptive thresholding
  - Edge detection: 1st and 2nd derivatives
  - Canny edge detector
  - Hough Transform
- **Chap 10. Mathematical Morphology**
  - Erosion & dilation -> boundary detection
  - Opening & closing -> noise removal
  - Hit-or-miss transform -> shape detection
  - Binary applications: region filling, connected components, skeletons
  - Grayscale morphology -> edge detection, noise removal