

[J 7.4, 9.1-8, 9.13]

[GW 3.5-8, 4.1-4, 10.1-2]

### Structural Operations

- Image enhancement using point operations is rather limited in scope
- Can do much more by taking into account the local image environment
  - define neighbourhood using the concept of mask (or window, template, spatial filters)
  - a mask is a small 2D array of coefficients
  - the coefficients in a mask represent the weights of the neighbours
- Place centre of mask over each and every pixel of image, apply transformation on the given pixel which depends on the weighted gray levels of the neighbours of the given pixel
- This operation is equivalent to carrying out correlation (See Chapter 2)

### example

(0,0)

 $\Pi_j$ 

↓

	a(-1,-1)	a(-1,0)	a(-1,1)		
	a(0,-1)	a(0,0)	a(0,1)		
	a(1,-1)	a(1,0)	a(1,1)		

Let  $u(i, j)$  be original image  
 $v(i, j)$  be spatially filtered image  
 $a(x, y)$  be spatial filter

Then

$$v(i, j) =$$

i.e. this example implemented neighbourhood averaging.

Every correlation can be re-written as a convolution and vice versa. All it takes is a re-arrangement of coefficients of the mask by reflection

e.g.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \circ f = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} \otimes f$$

where the mask has been reflected about the vertical axis, and then the horizontal axis

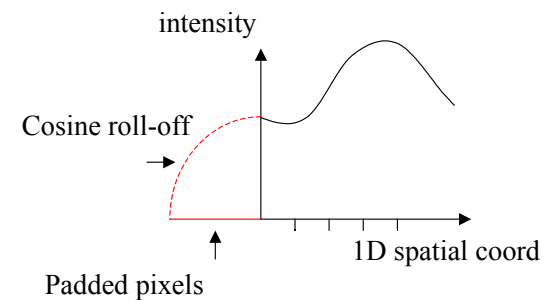
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \xrightarrow[\text{Reflection}]{\text{Horizontal}} \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{bmatrix} \xrightarrow[\text{Reflection}]{\text{Horizontal}} \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

### Remarks

- (1) mask size is usually  $n \times n$ ,  $n$  odd
- (2) different mask coefficients can lead to very different effects (and hence applications)
- (3) edge effects: when the mask is placed near the borders of the image, some of the mask coefficients do not cover

image pixels. Thus the concept of spatial filtering requires modification at the borders, leading to undesirable edge effects (e.g. note the vertical bands across the Wiener Filter restorations in the picture book on Chapter 2: Image Restoration)

- ❖ Zero padding at borders (add pixels of zero intensity around the border)
- ❖ Gaussian rolloff: the intensity of the padded pixels are not zero, but are equal to the intensity of the adjacent border pixels multiplied by a decreasing (roll-off) function. In this case, use Gaussian function as roll-off
- ❖ Cosine rolloff: use the Cosine function as roll-off



- ❖ Wraparound: identify opposite edges (mathematically, the image is now treated as a torus: the digital image world is round!)

e.g.  $(-1, 9) \rightarrow (6, 2)$  for  $7 \times 7$  image

$(x, y) \rightarrow (x \% M, y \% N)$ , using C-syntax

- Adopting the principle of wraparound, many spatial filters can be implemented easily & efficiently using DFT

e.g. neighbourhood averaging

$$v(m, n) = \sum_k \sum_l u(m+k, n+l) a(k, l)$$

$$= a \circ u \quad (\text{spatial correlation})$$

$\therefore$  using correlation theorem, we have♦

$$V(i, j) = A(-i, -j) U(i, j)$$

$$\Rightarrow v = \underline{DFT^{-1} \{ A(-i, -j) U(i, j) \}}$$

• Recall:  $a$  real  $\Rightarrow A^*(i, j) = A(-i, -j)$

Special case:  $a$  symmetric ( i.e.  $a(x, y) = a(-x, -y)$  )

$a$  symmetric  $\Rightarrow A$  symmetric

$\therefore$   $a$  real and symmetric  $\Rightarrow A^* = A$  (real and symmetric)

In this case, correlation is *equivalent* to convolution!

### Applications of spatial filtering: Smoothing (removing noise)

- neighbourhood averaging

☞ use  $n \times n$  masks with coefficients

$$a(i, j) = \frac{1}{\text{\# of pixels in mask}}$$

e.g.  $n = 5$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

× 1/25

☞ bigger windows can cause severe blurring

- the method can be generalized to allow different weights in the mask, leading to *weighted average* of image intensities of pixels within the pre-specified neighbourhood, e.g. a  $3 \times 3$  window centred at the  $(x, y)$  pixel

- Gaussian filter

- in the special case when the neighbourhood is the entire image, and the weights are given by the 2D Gaussian function:

$$h(x, y) = \exp \frac{-x^2 + y^2}{2\sigma^2}$$

then we have the Gaussian filtering

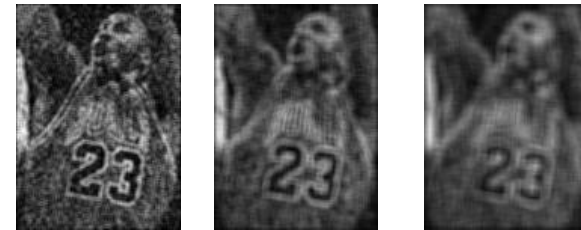
- Median filter

- instead of calculating the *mean* value, we calculate the *median* of all the neighbouring pixels in the window
- as before, bigger windows can cause severe blurring
- very effective for salt 'n' pepper (or binary) noise, but poor when SNR is low
- cannot* be implemented using DFT

Example: Effects of window size using averaging filter to remove zero-mean Gaussian noise of variance 0.05



Top row: Original and noisy image. Bottom row: Averaging filter results with window size of 3, 5 and 7.



Example: Averaging filter ( $3 \times 3$ ) to remove 5% binary noise



Noisy Image



Averaging filter result

Example: Effect of window size using Median filter to remove 5% binary noise. Result of 3-by-3 averaging filter is shown for comparison.



Original

Noisy

Averaging Filter



Median Filter 3x3

5x5

7x7

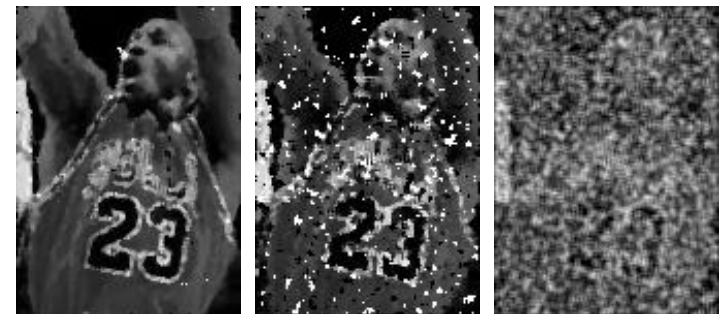
Example: Median filter to remove 20% and 50% binary noise (every pixel has 50% chance of bit reversal!). Result of 3-by-3 averaging filter is shown for comparison.



Original

20% noise

50% noise

Median Filter  
(20% noise case)Median Filter  
(50% noise case)Averaging Filter  
(50% noise case)

### Theory of Successive Filtering

Let  $h_1$  and  $h_2$  be two spatial filters, then by the correlation theorem

$$h_2 \circ (h_1 \circ u) \Leftrightarrow H_2^* (H_1^* U)$$

But since

$$(h_2 \otimes h_1) \circ u \Leftrightarrow$$

$$\therefore h_2 \circ (h_1 \circ u) =$$

■

In words, this result means that successive filtering with  $h_1$  followed by  $h_2$  is equivalent to filtering with  $(h_2 \otimes h_1)$ .

- This result allows us to implement some 2D filters by successive 1D-filters (obviously more convenient)

- ❖ Example: 1D-implementation of Neighbourhood Averaging filters

$$h_1 = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, h_2 = \frac{1}{3} [1 \ 1 \ 1] \quad \left. \vphantom{\begin{matrix} h_1 \\ h_2 \end{matrix}} \right\} \text{1D-filters}$$

$$\Rightarrow h_2 \otimes h_1 = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Thus this 2D neighbourhood averaging filter  $(h_2 \otimes h_1)$  can be implemented using two successive 1D filters  $h_1$  and  $h_2$ . In this case, these two 1D filters correspond to neighbourhood averaging filters in vertical and horizontal windows, respectively.

- More importantly, the theory of successive filtering enables us to *design* new filters by combining simpler filters with known properties
- ❖ Example: Sobel filters (see later)

### Applications of spatial filtering: Edge Detection

#### First Order Methods

- An edge is defined by sharp changes in the image intensity (i.e. large gradient)

- Let  $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$ , where  $f(x, y)$  denotes the

intensity of the image at the  $(x, y)$  pixel

- A pixel in the image is labelled an edge pixel if its intensity gradient,  $\|\nabla f\|$ , exceeds a certain threshold (usually application-dependent)
- For digital images, the gradient vector is approximated by finite difference:

$$\nabla f \approx \begin{bmatrix} f(x+1, y) - f(x-1, y) \\ f(x, y+1) - f(x, y-1) \end{bmatrix}$$

- In terms of mask filtering, the horizontal gradient can be implemented using the  $h = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$  filter.

- Similarly, the vertical gradient can be implemented

using the  $v = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$  filter.

- However, in practice, all images contain noise that would be amplified by the gradient filter. It is therefore important to reduce the effect of noise by some kind of smoothing operation before gradients are computed (see section on smoothing above)
- Note that while these filters can be quite effective at reducing noise, they can also *blur edges*. This conflicts with our original aim of reducing the effect of noise in edge detection. A compromise is the *Sobel filter* which neatly combines noise reduction with gradient estimation in an elegant way.
- Sobel filter for Vertical Edge Detection

A point on a vertical edge is characterized by sharp changes in the horizontal gradient, which can be estimated using the  $h = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$  filter. To remove noise without corrupting the *vertical* edge under study,

we apply neighbourhood averaging in a *vertical*  $3 \times 1$  neighbourhood: specifically, we use the scheme

$$f(x, y) \rightarrow \frac{f(x, y-1) + 2f(x, y) + f(x, y+1)}{4}$$

which can be implemented by the  $nav = \frac{1}{4} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$  filter.

Note that the factor of  $\frac{1}{4}$  is usually omitted as it can be absorbed by a threshold.

The Sobel filter is then the successive operations of the noise reduction filter followed by the gradient estimator. From the Theory of Successive Filtering, the Sobel filter for vertical edge detection is given by:

$$h \otimes nav = (-1 \ 0 \ 1) \otimes \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} =$$

- ▶ thus Sobel is robust to noise because of the effect of neighbourhood averaging *along* the direction of

the edge (which, in general, will *not* cause blurring to the edge)

- ▶ if we wish to superimpose the above gradient map onto the original image, we only need to add the identity function to obtain an *edge-enhancement* filter

$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \otimes [1 \ 2 \ 1] + [1]\alpha =$$

- ▶ the parameter  $\alpha$  controls the amount of enhancement
  - ▶ thus the Theory of Successive Filtering can help us design edge-enhancement filters or other special effect filters by combining simpler filters in succession
  - Sobel filter for Horizontal Edge Detection
- Similarly, the Sobel filter for Horizontal Edge Detection can be implemented using:

$$v \otimes nah = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \otimes (1 \ 2 \ 1) =$$

- Other first order edge detection filters include:

◊ Roberts filters

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

◊ Prewitt filters

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

- Let  $G_x$  and  $G_y$  be the output of a horizontal and vertical first order edge detection filters. These components may be combined to give *total gradient information* by e.g. taking their t2-norm or 1-norm:

$$G = \sqrt{G_x^2 + G_y^2} \quad \text{or} \quad |G_x| + |G_y|$$

- The direction of the edge,  $\theta$ , may be obtained by treating the two components as a vector:

$$\tan \theta = G_y / G_x$$

- However, such angular information tends to be poor.

#### ▣ Second Order Methods

- An edge is defined by second order derivatives in the image intensity
- Let  $\nabla^2 f = \left( \frac{\partial^2 f}{\partial x^2} \right) + \left( \frac{\partial^2 f}{\partial y^2} \right)$ , the Laplacian of  $f(x, y)$ , where  $f(x, y)$  denotes the intensity of the image at the  $(x, y)$  pixel.
- A finite difference scheme for (minus) the Laplacian is given by:

$$-\nabla^2 f(x, y) \rightarrow 4f(x, y) - f(x, y+1) - f(x, y-1) - f(x+1, y) - f(x-1, y)$$

which can be implemented by the following filter:

$$lap = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

- Some engineers prefer the more balanced filter

$$lap' = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

- In frequency domain, the Laplacian is given by

$$\nabla^2 f \Leftrightarrow -(2\pi)^2 (u^2 + v^2) F(u, v)$$

- As in first order methods, one expect noise amplification to occur and therefore some kind of noise reduction is required
- Since we are dealing with a 2-dimensional filter (as opposed to 1-D filters as in the Sobel case), we shall apply low-pass filtering using the 2-D Gaussian filter (note that in principle the neighbourhood is now infinite in extent, but in practice the weights become negligible a few  $\sigma$  away)
- The effect of Gaussian filtering followed by Laplacian filtering can be analyzed in the frequency domain:

$$-\nabla^2 (h \otimes f) \Leftrightarrow (2\pi)^2 (u^2 + v^2) (H(u, v) F(u, v))$$

But since

$$-\nabla^2 h \Leftrightarrow (2\pi)^2 (u^2 + v^2) H(u, v)$$

therefore, we have the result

$$-\nabla^2 (h \otimes f) = (-\nabla^2 h) \otimes f$$

Thus the combined effect can be implemented using the Laplacian of Gaussian (LOG) filter:

$$\nabla^2 h = \frac{1}{r} \left( \frac{\partial}{\partial r} \left( r \frac{\partial}{\partial r} h \right) \right) + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2} h$$

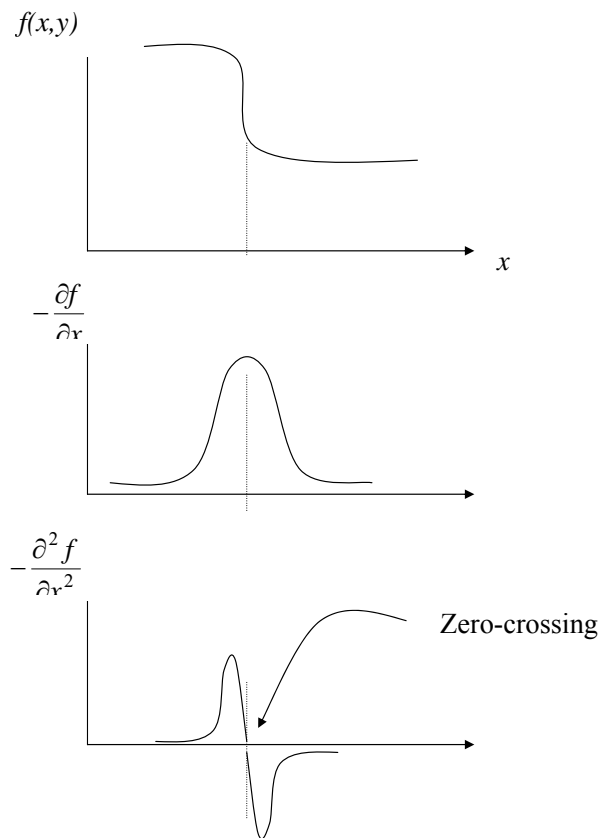
$$\text{but } h(r, \theta) = \frac{1}{2\pi} \sigma^2 e^{-r^2/2\sigma^2}$$

$$\Rightarrow \frac{\partial^2}{\partial \theta^2} h = 0 \text{ since } h \text{ is independent of } \theta$$

$$\Rightarrow -\nabla^2 h = \frac{1}{\pi\sigma^4} \left( 1 - \frac{r^2}{2\sigma^2} \right) e^{-r^2/2\sigma^2}$$

- We could apply thresholding to the LOG filter output and thus define an edge pixel to be one in which the second order derivatives is large compared to the threshold.

- A more popular alternative is to define edge pixels to be points of zero-crossing of the LOG filter. I.e. these are points which correspond to the locations where the first order derivatives are maximum.



Examples: edge detection in the presence of Gaussian noise



Original

0.01 Gaussian

0.05 Gaussian



Sobel edge detector results



Laplacian edge detector results

### Applications of spatial filtering: Line Detection

#### ◆ Horizontal and vertical line detection

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

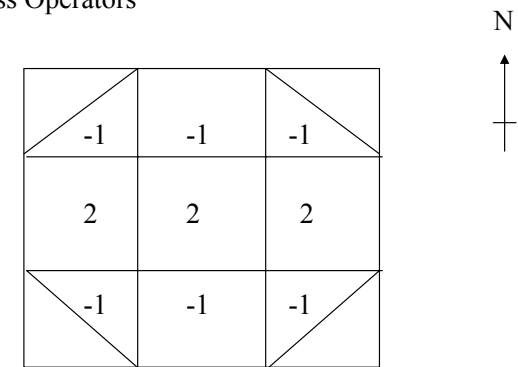
↓                      ↓

detects horizontal lines    detects vertical lines

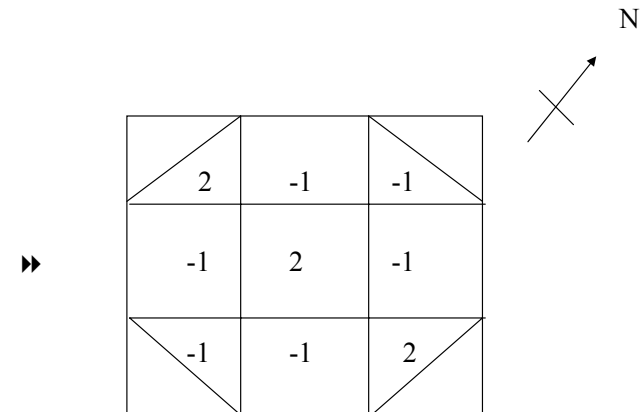
#### Remarks

- ▶▶ can generalise to detection of lines (and similarly for edges) at multiples of  $45^\circ$  to the horizontal by rotating the masks (see compass operators below)
- ▶▶ noise in the image can be amplified in the filtered output  $\Rightarrow$  need for good noise reduction and/or optimal thresholds
- ▶▶ some 2D filters can be implemented using 1D-filters (see prove by using FT theorems below)

#### ◆ Compass Operators



Create a new mask for detecting lines at  $45^\circ$  by rotating the hexagon containing the coefficients (see demo in class) clockwise. Similarly, all other directions can be obtained by additional rotations.



### Frequency-domain filters

Recall: spatial filters  $a(m,n)$  can be implemented using DFT

$$v = \text{DFT}^{-1}\{A^*U\}$$

- thus many spatial filters can be implemented in the *frequency-domain* by the filter  $A(i, j)$
- alternatively, can achieve elaborate spatial filtering effects by simple *designs* in the frequency-domain:

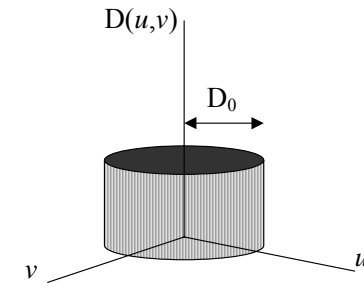
$$a(m, n) \Leftrightarrow A(i, j)$$

### Low-pass, High-pass, Band-pass filters

- Ideal low-pass

$$H_{ILP}(u, v) = \begin{cases} 1, & \text{if } D(u, v) \leq D_0 \\ 0, & \text{otherwise} \end{cases}$$

where  $D(u, v) = \sqrt{u^2 + v^2}$ , and  $D_0$  is a non-negative cutoff frequency.



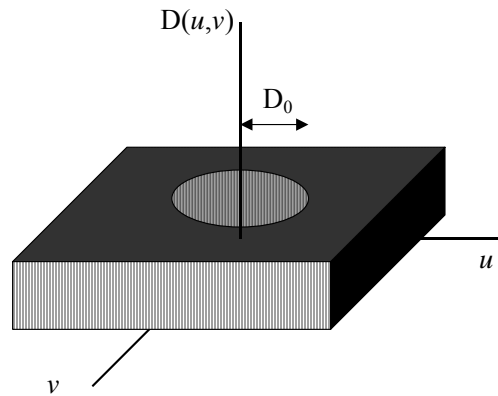
This is a low-pass filter because only the low-frequency components are retained, any components with frequency above the cutoff are discarded. Since the cutoff is *sharp* in this definition, we say this is an *ideal* low-pass filter. In practice, an ideal low-pass filter cannot be realized.

Once the low-pass filter is defined, the high-pass and band-pass filters can be constructed from it.

- Ideal high-pass

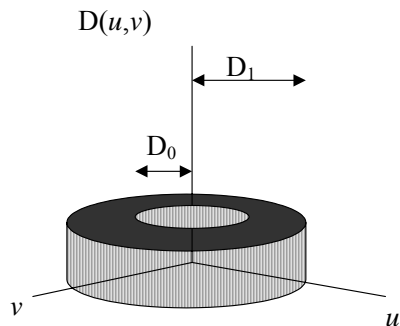
$$H_{IHP}(u, v) = 1 - H_{ILP}(u, v)$$

note: only high-frequency components are retained



- Ideal band-pass (only a specified range of frequencies are retained)

$$H_{IBP}(u, v) = H_{ILP, D_1}(u, v) - H_{ILP, D_0}(u, v)$$



### Butterworth filters

Ideal filters cannot be built, so in practice, use Butterworth filters which can be realized in hardware

- Butterworth low-pass filter of order  $n$

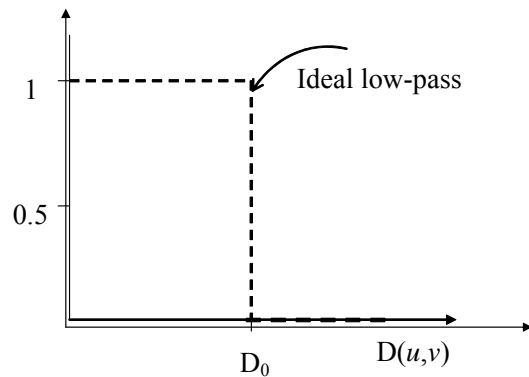
$$H_{BLP}^{(n)}(u, v) = \frac{1}{1 + \left(\frac{D(u, v)}{D_0}\right)^{2n}}$$

⚠ Note that as the order  $n \rightarrow \infty$ , the Butterworth filter converges to the ideal low-pass filter. In practice, even at small order, the Butterworth filter is very close to the ideal case.

⚠ The cutoff frequency,  $D_0$ , is not a sharp cutoff, and represents the point when the filter takes the value of  $1/2$ . Some engineers prefer to set the cutoff at a point when the *square* of the filter takes the value of  $1/2$ , leading to a modified Butterworth filter of the form

$$\frac{1}{1 + (\sqrt{2} - 1) \left(D(u, v) / D_0\right)^{2n}}$$

Since the Butterworth filter is axially symmetric, we shall plot its filter value as a function of the magnitude of the frequency variables. We fix the cutoff frequency and illustrate the effects of increasing the order:



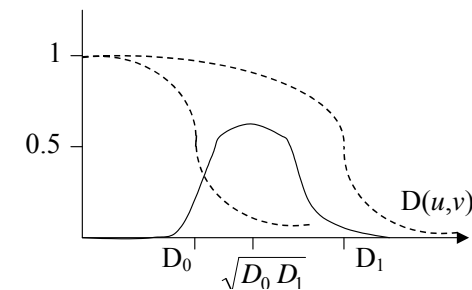
- Butterworth high-pass filter of order  $n$

$$H_{BHP}^{(n)}(u, v) = 1 - H_{BLP}^{(n)} = \frac{1}{1 + \left(\frac{D_0}{D(u, v)}\right)^{2n}}$$

- Butterworth band-pass filter of order  $n$

$$H_{BBP}^{(n)}(u, v) = \frac{1}{1 + \left(\frac{D(u, v)}{D_1}\right)^{2n}} - \frac{1}{1 + \left(\frac{D(u, v)}{D_0}\right)^{2n}}$$

- ▶▶ The width of the band-pass filter is the difference of the two cutoffs:  $D_1 - D_0$
- ▶▶ By differentiation w.r.t. to  $D$  to find the location of the maximum of the filter, the centre of the band can be proved to be the geometric mean of the two cutoffs, namely:  $\sqrt{D_0 D_1}$  (exercise). What is the maximum value of the filter at this centre of the band?



Applications

- ✂ Since noise and edges are essentially high frequency signals, low-pass filtering is useful for removing noise or softening undesirable edges. In fact, neighbourhood averaging filter and the Gaussian filters are examples of low-pass filters and can be implemented in the frequency domain using Fourier Transform.
- ✂ High-pass filtering is useful for extracting edges and sharpening images, but must be prepared to handle problems with noise
- ✂ For problems in which certain ranges of frequencies are known to be important, band-pass filters can be very useful. Examples include design of equalizers and mixers for audio processing, and face recognition (which involve extracting certain facial features such as eyebrows and lips).

High-boost filtering (high frequency emphasis)

- A high-pass filtered image may be computed from a low-passed image :

$$I_{HP} = I - I_{LP}$$

- This image attenuated its low-frequency components  $\Rightarrow$  enhanced its edges, but does not look realistic as the “background” has disappeared
- A high-boost filtered image is a useful compromise by adding a fraction of the low-frequency components back

$$\begin{aligned} I_{\text{high-boost}} &= \alpha I - I_{LP} \\ &= (\alpha - 1)I + I_{HP} \quad , \quad \alpha \geq 1 \end{aligned}$$

- The high-boost filter can be implemented in the frequency domain using Butterworth filters.
- Alternatively, in the spatial domain, we can use the mask

$$\frac{1}{9} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9\alpha - 1 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad \alpha \geq 1$$

$$= \alpha [1] - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= [1] - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} + (\alpha - 1)[1]$$

- This spatial filter first subtract a neighbourhood averaged image (low-passed) from the original to obtain a high-passed image, then add a fraction of the original image to achieve the similar effects of high-boost filtering
- May require post-processing such as normalization or clipping to ensure gray levels are still within range.



Original



Low-passed (Gaussian 2.0)

High-passed Image =  
2\* (Original - Low-passed)High-boost Image  
(add High-passed to  
0.7\*Original)

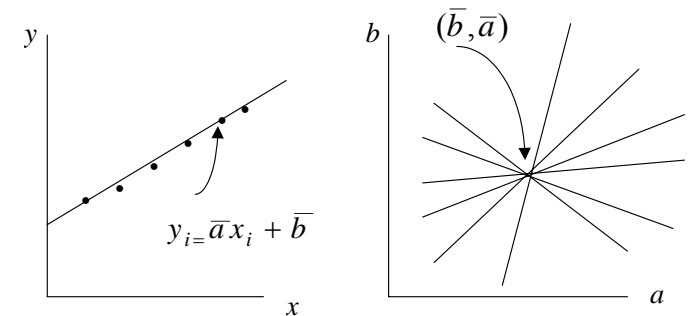
### Hough Transform

- Edge/line detections using spatial masks are local schemes
  - ▶▶ noise and non-uniform lighting can cause spurious intensity discontinuities  $\Rightarrow$  breaks in boundary
  - ▶▶ difficult to detect complicated smooth shapes
- Hough Transform is a global method
  - ▶▶ robust to noise
  - ▶▶ can be generalised to detect all parametrizable curves (useful for pattern recognition)
  - ▶▶ computationally expensive (active research to find efficient implementation)
- Example: line detection using H.T.
  - ▶▶ each edge point  $(x_i, y_i)$  lies on the line  $y = \bar{a}x + \bar{b}$ , which we want to detect (large dots on diagram below)

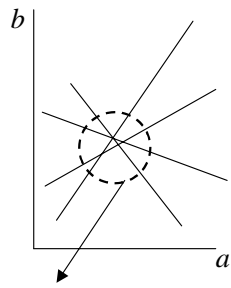
- ▶▶ since we know each point lies on a line, but not the parameters of the line, then in principle, each point  $(x_i, y_i)$  may lie on a line  $y_i = ax_i + b$  for arbitrary line parameters  $(a, b)$
- ▶▶ thus each *point* in the  $(x, y)$  plane corresponds to a *line* in the  $(a, b)$  parameter-plane! Namely,

$$b = y_i - ax_i$$

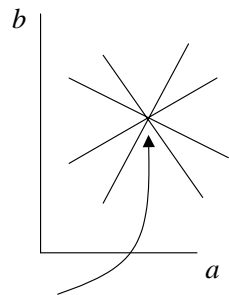
- ▶▶ these parameters represent all the possible lines in the spatial domain that passes through the image point  $(x_i, y_i)$
- ▶▶ the intersection point on the parameter domain,  $(\bar{a}, \bar{b})$ , yields the parameter of the detected line



- ▶▶ H.T. allow simultaneous detection of all lines in the  $(x, y)$  plane (c.f. multiple regression techniques)
- ▶▶ the strength of detection is measured by the nature of intersections in the parameter plane (i.e. how many points in the  $x$ - $y$  plane are consistent with the detected line)
- ▶▶ noise & other effects may, in practice, lead to clustering of weak intersection points rather than one strong intersection point

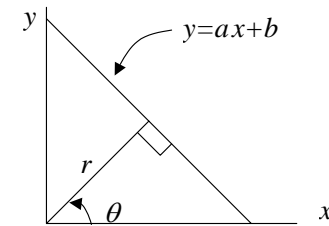


Clustering of Weak intersections



Strong intersection

- Alternative representation for straight line
  - ▶▶ a vertical straight line cannot be represented by the equation  $y = ax + b$  (because the value of “ $a$ ”, the gradient, tends to  $\infty$ )
  - ▶▶ use polar representation for a line,  $(r, \theta)$ , instead



- ▶▶ now any line  $y = ax + b$  (even vertical ones) can be represented by the parameters  $(r, \theta)$ . In fact, the new equation of line is

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

where

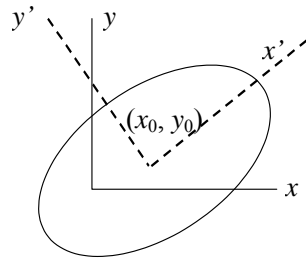
$$a = -\frac{1}{\tan \theta}, \quad b = \frac{r}{\sin \theta}$$

- Detection of ellipses (circles)

- ▶▶ ellipses centred at  $(x_0, y_0)$  with major & minor axes  $(a, b)$  [a total of 4 parameters]

$$\frac{(y - y_0)^2}{b^2} + \frac{(x - x_0)^2}{a^2} = 1$$

- ▶▶ rotated ellipse (5 parameters)



$$\frac{(x')^2}{a^2} + \frac{(y')^2}{b^2} - 1 = 0$$

where

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix}$$

or

$$d(x - x_0)^2 + e(y - y_0)^2 + f(x - x_0)(y - y_0) = 1$$

where

$$d = \frac{\cos^2 \theta}{a^2} + \frac{\sin^2 \theta}{b^2}$$

$$e = \frac{\sin^2 \theta}{a^2} + \frac{\cos^2 \theta}{b^2}$$

$$\begin{aligned} f &= \sin 2\theta \left( \frac{1}{a^2} - \frac{1}{b^2} \right) \\ &= (d - e) \tan 2\theta \end{aligned}$$

$$\left( \because d - e = \left( \frac{1}{a^2} - \frac{1}{b^2} \right) (\cos^2 \theta - \sin^2 \theta) \Rightarrow \frac{1}{a^2} - \frac{1}{b^2} = \frac{(d - e)}{\cos 2\theta} \right)$$

- ▶▶ Now each image point correspond to a curve in the 5-dimensional parameter space

- ▶▶ By finding the intersection(s) of these curves in the parameter space, we may detect ellipse(s) of corresponding parameters In other words, a *point* in the parameter space correspond to an *ellipse* in the spatial domain. Similarly, other parametrized shapes can be detected in the same way.

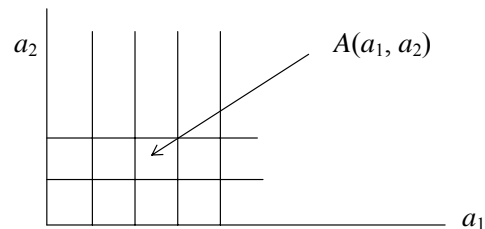
### Hough Transform Algorithm

- Write equation of shape to be detected as

$$f(x, y; a_1, a_2, \dots, a_n) = 0$$

where  $(x, y)$  are the image coordinates, and the  $a_i$  are the shape parameter

- Set up accumulator cells  $A(a_1, a_2, \dots, a_n)$  in the parameter plane e.g.



Thus each cell represents a *set* of parameter values (c.f. quantization).

- Loop through all parameter cells and update  $A(a_1, \dots, a_n)$  by the rule

for each edge point  $(x_i, y_i)$

$$\text{if } |f(x_i, y_i; a_1, \dots, a_n)| \leq \varepsilon$$

$$A(a_1, \dots, a_n) \rightarrow A(a_1, \dots, a_n) + 1$$

endif

end of for loop

- ▶▶ the tolerance level  $\varepsilon$  allows small departure from the true shape, and is useful since all images have noise and geometric distortion due to the digitization process
- ▶▶ accuracy depends on grid/cell size in parameter space (large cells avoid the problem of intersection-clustering, small cells are more accurate but only if the shape to be detected are exact e.g. lines which are not exactly straight may not be detected if the cells are too small)
- Simultaneous shape detection by identifying peaks in the accumulator cells (recall: each count in the cell corresponds to one image point that is consistent with the shape parameters represented by the cell)