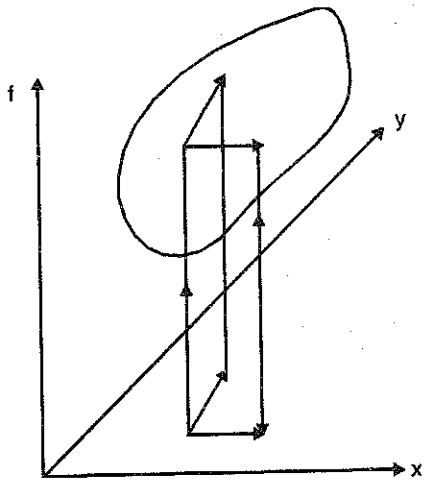


Fig. 7.3.3

Case 3 : Output is a function of three variables (3-dimensional eg. Density at different points in the atmosphere or pressure at different points) (Not useful in image processing)

$$P = f(x, y, z)$$

$\therefore$ the three partial derivatives are

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x + h, y, z) - f(x, y, z)}{h}$$

$$\frac{\partial f}{\partial y} = \lim_{k \to 0} \frac{f(x, y + k, z) - f(x, y, z)}{k}$$

$$\frac{\partial f}{\partial z} = \lim_{l \to 0} \frac{f(x, y, z + l) - f(x, y, z)}{l}$$

$$\therefore \text{ gradient } \nabla F = \hat{i}\frac{\partial f}{\partial x} + \hat{j}\frac{\partial f}{\partial y} + \hat{k}\frac{\partial f}{\partial z} \quad ...(7.3.2)$$

Case 3 will not be used in this book

Now, magnitude of a vector is given by

$$A^2 = A_x^2 + A_y^2$$

$$|A| = \sqrt{A_x^2 + A_y^2}$$

Similarly, magnitude of a 2-D gradient is

$$|\nabla F| = \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2} \quad ...(7.3.3)$$

Here $\frac{\partial f}{\partial x}$ = Rate of change of 'f' w.r.t the x-direction

$\frac{\partial f}{\partial y}$ = Rate of change of 'f' w.r.t the y-direction

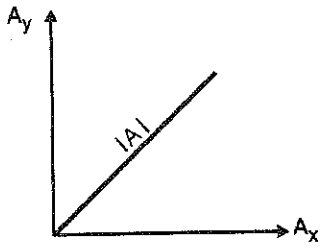


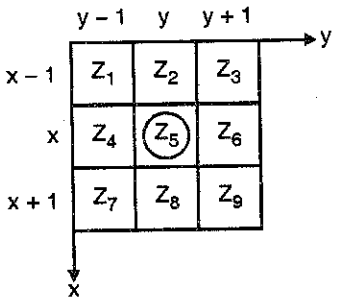
Fig. 7.3.4

7.4 Finding Gradients using Masks

Consider a 3 × 3 neighbourhood with Z_5 as the origin.

We know, $\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x + h, y) - f(x, y)}{h}$

$$\frac{\partial f}{\partial y} = \lim_{k \to 0} \frac{f(x, y + k) - f(x, y)}{k}$$



In the discrete domain, h = k = 1

$$\therefore \frac{\partial f}{\partial x} = f(x + 1, y) - f(x, y)$$

and $$\frac{\partial f}{\partial y} = f(x, y + 1) - f(x, y)$$

From the 3 × 3 neighbourhood we have,

$$\frac{\partial f}{\partial x} = Z_8 - Z_5 \quad \text{and} \quad \frac{\partial f}{\partial y} = Z_6 - Z_5$$

Since, $$|\nabla F| = \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2}$$

$$\therefore |\nabla F| = [(Z_8 - Z_5)^2 + (Z_6 - Z_5)^2]^{1/2}$$

So as to make it easier for implementation,

$$|\nabla F| = |Z_8 - Z_5| + |Z_6 - Z_5|$$

i.e. $$|\nabla F| = |Z_5 - Z_8| + |Z_5 - Z_6| \quad ...(7.4.1)$$

This is the first order difference gradient. This can be implemented using two masks.

$|Z_5 - Z_8| =$

1	0
-1	0

→ mask 1

and $|Z_5 - Z_6| =$

1	-1
0	0

→ mask 2

This is known as the Ordinary operator.

Steps to compute the gradient of an image are as follows

(1) Convolve the original image with mask 1. This gives us the gradient along the x-direction.

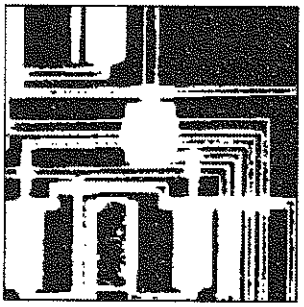
(2) Convolve the original image with mask 2. This gives us the gradient along the y-direction.

(3) Add the results of 1 and 2.

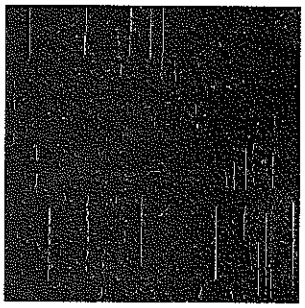
The advantage of this prolonged method is that, we can see the effect of each mask separately.

MATLAB code for ordinary operator

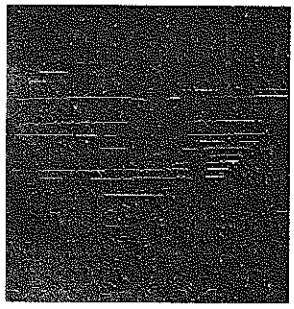
```
%% Ordinary operator %%
clear all
clc
aa = imread('test.jpg');
a = double(aa);
[row col] = size(a);
w1=[1 0;-1 0];
w2=[1 -1; 0 0];
for x=2:1:row-1;
for y=2:1:col-1;
a1(x,y)=w1(1)*a(x,y)+w1(2)*a(x,y+1)+w1(3)*a(x+1,y)
+w1(4)*a(x+1,y+1);
a2(x,y)=w2(1)*a(x,y)+w2(2)*a(x,y+1)+w2(3)*a(x+1,y)
+w2(4)*a(x+1,y+1);
    end
end
a3=a1+a2;          %% To find the resultant gradient image
figure(1)
imshow(uint8(a1))
    %% The x-gradient image, you might need normalization
figure(2)
imshow(uint8(a2))
    %% The y-gradient image, you might need normalization
figure(3)
imshow(uint8(a3)) %% Final image
```



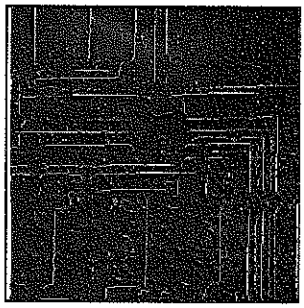
(a) Original image



(b) X-gradient image



(c) Y-gradient image



(d) Resultant gradient image

Fig. 7.4.1

There is another direct method of implementing the ordinary operator practically. It gives results that are almost similar.

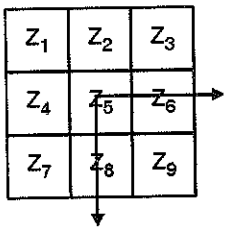
(1) Add masks 1 and 2

1	0	+	1	-1	=	2	-1
-1	0		0	0		-1	0

(2) Convolve the original image with this resultant mask to get the gradient image.

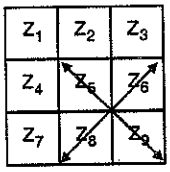
7.4.1 Roberts Mask

Robert L.G in 1965 published a paper in which he stated that better results could be obtained if cross differences were taken instead of the straight difference.



Ordinary masks $|\nabla F| = |Z_5 - Z_8| + |Z_5 - Z_6|$

Roberts masks $|\nabla F| = |Z_5 - Z_9| + |Z_6 - Z_8|$ ...(7.4.2)



∴ Roberts masks are

1	0
0	−1

0	1
−1	0

The results of the Roberts mask are shown along with the program.

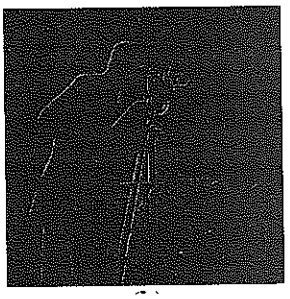
The sum of the two Roberts masks is

1	1
−1	−1

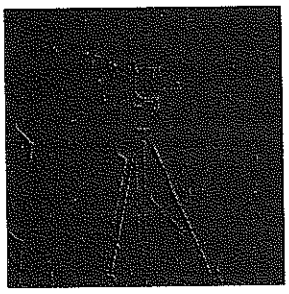
MATLAB code for Roberts operator

```
% Roberts operator %%
clear all
clc
aa=imread('cameraman.jpg');
a=double(aa);
[row col]=size(a);
w1= [1 0; 0 -1];
w2= [0 1; -1, 0];
for x=2:1:row-1;
for y=2:1:col-1;
a1(x,y)=w1(1)*a(x,y)+w1(2)*a(x,y+1)+w1(3)*a(x+1,y)
+w1(4)*a(x+1,y+1);
a2(x,y)=w2(1)*a(x,y)+w2(2)*a(x,y+1)+w2(3)*a(x+1,y)
+w2(4)*a(x+1,y+1);
    end
end
a3=a1+a2;          %% Final Gradient Image %%
figure(1)
imshow(uint8(a1))
    %% The x-gradient image, you might need normalization
figure(2)
imshow(uint8(a2))
    %% The y-gradient image, you might need normalization
figure(3)
imshow(uint8(a3)) %% Final image %%
```

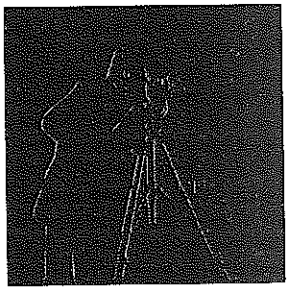
(a) Original image



(b) X-gradient image



(c) Y-gradient image



(d) Resultant image

Fig. 7.4.2

7.4.2 Prewitts and Sobel Operators

Roberts mask, as is evident, is an even sized mask (2 × 2). Masks of even sizes are awkward to implement. Note that the differential along the diagonals of a 2 × 2 mask is used and the edge value after the convolution corresponds to the central point $\left(r-\frac{1}{2}, c-\frac{1}{2}\right)$.

This problem can be avoided using 3 × 3 masks.

Prewitts Operator : Prewitt J.M.S in his paper "Object Enhancement and Extraction" in 1970 came up with a 3 × 3 mask. The Prewitt operator, as is now called, while approximating the first derivative, assigns similar weights to all the neighbours of the candidate pixel whose edge strength is being calculated.

$$\nabla F \approx |(Z_7 + Z_8 + Z_9) - (Z_1 + Z_2 + Z_3)| + |(Z_3 + Z_6 + Z_9) - (Z_1 + Z_4 + Z_7)| \quad ...(7.4.3)$$

x-gradient y-gradient

From this equation, the masks that we obtain are

-1	-1	-1
0	⓪	0
1	1	1

F_x

-1	0	1
-1	⓪	1
-1	0	1

F_y

These masks are known as the Prewitts masks.

Sobel Operator : Duda R.O and Hart P.E in 1973 published a paper "Pattern Classification and Scene Analysis" in which they used a new operator known as the Sobel Operator.

In the Sobel Operator higher weights are assigned to the pixels close to the candidate pixels.

$$\nabla F \approx |(Z_7 + 2Z_8 + Z_9) - (Z_1 + 2Z_2 + Z_3)| + |(Z_3 + 2Z_6 + Z_9) - (Z_1 + 2Z_4 + Z_7)| \quad ...(7.4.4)$$

x-gradient y-gradient

From this equation, the masks that we obtain are

-1	-2	-1
0	⓪	0
1	2	1

F_x

-1	0	1
-2	0	2
-1	0	1

F_y

Implementation of Prewitt and Sobel Operators

(1) Convolve original image with the Fx mask to get the x-gradient image.

(2) Convolve original image with Fy mask to get the y-gradient image.

(3) Add the results of step (1) and (2)

As mentioned earlier, the advantage of using three steps is that we can see the results of the Fx mask and the Fy mask separately.

A shorter way of doing this which would give almost similar results is

(1) Add the Fx and Fy masks first.

(2) Convolve the new mask with the original image.

Prewitt mask

-1	-1	-1
0	0	0
1	1	1

+

-1	0	1
-1	0	1
-1	0	1

=

-2	-1	0
-1	0	1
0	1	2

F_x F_y $F_x + F_y$

Sobel mask

-1	-2	-1
0	0	0
1	2	1

+

-1	0	1
-2	0	2
-1	0	1

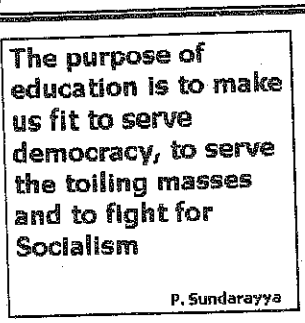
=

-2	-2	0
-2	0	2
0	2	2

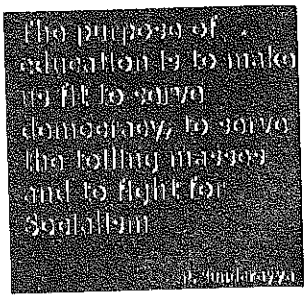
F_x F_y $F_x + F_y$

MATLAB code for Prewitts operator

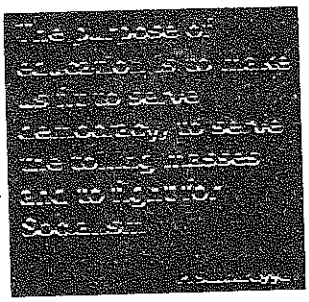
```
%% Prewitts operator %%

clear all

clc

aa=imread('test.tif');

a=double(aa);

[row col]=size(a);

w2=[-1  0  1;-1  0  1;-1  0  1];

w1=[-1  -1  -1; 0  0  0; 1  1  1];


for x=2:1:row-1;

for y=2:1:col-1;

a1(x,y)=w1(1)*a(x-1,y-1)+w1(2)*a(x-1,y)+
        w1(3)*a(x-1,y+1)+w1(4)* ...
        a(x,y-1)+w1(5)*a(x,y)+w1(6)*a(x,y+1)+
        w1(7)*a(x+1,y-1)+w1(8)* ...
a(x+1,y)+w1(9)*a(x+1,y+1);

a2(x,y)=w2(1)*a(x-1,y-1)+w2(2)*a(x-1,y)+
        w2(3)*a(x-1,y+1)+w2(4)* ...
        a(x,y-1)+w2(5)*a(x,y)+w2(6)*a(x,y+1)+
        w2(7)*a(x+1,y-1)+w2(8)* ...
a(x+1,y)+w2(9)*a(x+1,y+1);

 end

end

a3=a1+a2;       %% The final gradient value %%

figure(1)

imshow(uint8(a1))

   %% The x-gradient image, Normalisation might be required

figure(2)

imshow(uint8(a2))              %% The y-gradient image %%

figure(3)

imshow(uint8(a3))

    %% Final gradient image, Normalisation might be required
```



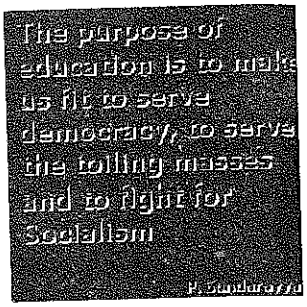
(a) Original image



(b) X-gradient image



(c) Y-gradient image



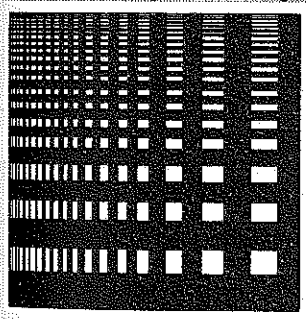
(d) Resultant gradient image

Fig. 7.4.3

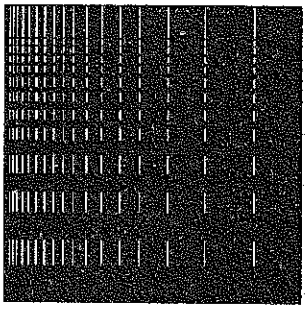
MATLAB code for Sobel operator

```
%% Sobel operator %%

clear all

clc

aa=imread('warne.tif');

a=double(aa);

[row col]=size(a);

w1=[-1   -2   -1;    0   0   0;   1   2   1  ];

w2=[-1    0    1;   -2   0   2;  -1   0   1  ];

for x=2:1:row-1;

for y=2:1:col-1;

a1(x,y)=w1(1)*a(x-1,y-1)+w1(2)*a(x-1,y)+
        w1(3)*a(x-1,y+1)+w1(4)* ...
a(x,y-1)+w1(5)*a(x,y)+w1(6)*a(x,y+1)+
w1(7)*a(x+1,y-1)+w1(8)* ...
a(x+1,y)+w1(9)*a(x+1,y+1);

a2(x,y)=w2(1)*a(x-1,y-1)+w2(2)*a(x-1,y)
        +w2(3)}*a(x-1,y+1)+w2(4)* ...
```

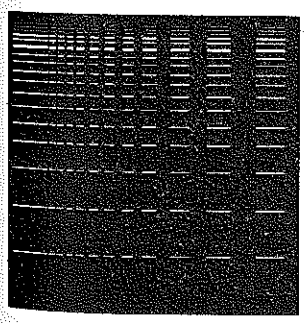
```
a(x,y-1)+w2(5)*a(x,y)+w2(6)*a(x,y+1)+
w2(7)*a(x+1,y-1)+w2(8)* ...
a(x+1,y)+w2(9)*a(x+1,y+1);
    end
end
a3=a1+a2;          %% Final gradient values %%
figure(1)
imshow(uint8(a1))
  %% The x-gradient image, Normalisation might be required
figure(2)
imshow(uint8(a2))
  %% The y-gradient image, Normalisation might be required
figure(3)
imshow(uint8(a3))
  %% Final gradient image, Normalisation might be required
```



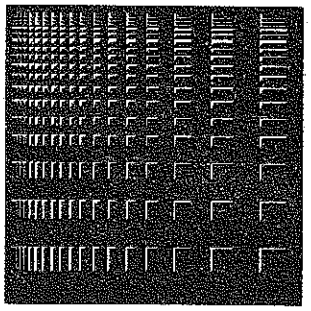
(a) Original image



(b) X-gradient image



(c) Y-gradient image



(d) Resultant gradient image

Fig. 7.4.4

Both results can be obtained using absolute values.

Have you noticed one thing that is common to all the edge detection masks that we have studied so far ?

Let us draw all the masks together.

1	0
−1	0

1	−1
0	0

Ordinary operator

1	0
0	−1

0	1
−1	0

Robert's operator

−1	−1	−1
0	0	0
1	1	1

−1	0	1
−1	0	1
−1	0	1

Prewitt mask

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Sobel mask

Sum of the coefficients of each of these masks is zero !!

This is a very important property to note. Edges are abrupt discontinuities in the gray levels and hence are high frequency regions. Since the sum of the coefficients of all these masks is zero, they eliminate all the low frequency components of the image i.e., when these masks are placed on low frequency regions, the output is zero. Hence these masks give edges without any low frequency regions in the final output image.

Derivative filters (Gradient filters), as the name suggests, calculate the gradient of the image.

Since noise is also high frequency, the derivative of the noise terms will always be large values and hence derivative filters are very sensitive to noise.

There is a huge advantage in using Prewitt and Sobel masks for edge detection. Both these operators provide a smoothing effect along with providing differentiation.

Hence Prewitt and Sobel operators perform well even when the image is noisy because they smoothen the noise !!

How do they achieve this ?

Let us consider the Prewitt operator F_x and F_y

-1	-1	-1
0	0	0
1	1	1

and

-1	0	1
-1	0	1
-1	0	1

These operators can be factored into the successive application of two simpler operators.

i.e.

-1	-1	-1
0	0	0
1	1	1

$=$

-1
0
1

$\times$ | 1 | 1 | 1 |

and

-1	0	1
-1	0	0
-1	0	1

$=$

1
1
1

$\times$ | -1 | 0 | 1 |

In this | 1 | 1 | 1 | is a low pass or a smoothing operator, while | -1 | 0 | 1 | is a high pass operator.

Hence the Prewitt operator performs uniform smoothing in one direction with edge detection in the perpendicular direction.

In a similar manner, we can split up the Sobel operator into smaller and simpler operators.

i.e.

-1	-2	-1
0	0	0
1	2	1

$=$

-1
0
1

$\times$ | 1 | 2 | 1 |

LP

HP

and

-1	0	-1
-2	0	2
1	0	1

$=$

1
2
1

$\times$ | -1 | 0 | 1 |

HP

LP

One can verify this by comparing the results of a Prewitt / Sobel filter with that of standard highpass filter on an image corrupted by Gaussian noise. (Use **imnoise** command to corrupt the image).

7.4.3 Compass Operators

It is seen that edges in the horizontal as well as in the vertical direction are enhanced when Prewitt's or Sobel's operator is used. There are applications, in which we need edges in all the directions. A simple method would be to rotate the Prewitt's or Sobel's mask in all the possible directions.

Considering a Prewitt's operator

-1	-1	-1
0	0	0
1	1	1

We move this mask in the anticlockwise direction to get all other masks.

-1	-1	-1
0	0	0
1	1	1

-1	-1	0
-1	0	1
0	1	1

-1	0	1
-1	0	1
-1	0	1

0	1	1
-1	0	1
-1	-1	0

1	1	1
0	0	0
-1	-1	-1

1	1	0
1	0	-1
0	-1	-1

1	0	-1
1	0	-1
1	0	-1

0	-1	-1
1	0	-1
1	1	0

-1	-1	-1
0	0	0
1	1	1

This operator is known as the compass operator and is very useful for detecting weak edges.

Compass operators can also be implemented using the Sobel operator.

```
%% Program of compass operator %%

clear all, clc

aa=imread('lily.tif');

I=double(aa);

s=size(I);

a=[-1 -1 -1; 0 0 0; 1 1 1]

b=[a(1,2) a(1,3) a(2,3); a(1,1) a(2,2) a(3,3); a(2,1) a(3,1) a(3,2)]

c=[b(1,2) b(1,3) b(2,3); b(1,1) b(2,2) b(3,3); b(2,1) b(3,1) b(3,2)]

d=[c(1,2) c(1,3) c(2,3); c(1,1) c(2,2) c(3,3); c(2,1) c(3,1) c(3,2)]


e=[d(1,2) d(1,3) d(2,3); d(1,1) d(2,2) d(3,3); d(2,1) d(3,1) d(3,2)]

f=[e(1,2) e(1,3) e(2,3); e(1,1) e(2,2) e(3,3); e(2,1) e(3,1) e(3,2)]

g=[f(1,2) f(1,3) f(2,3); f(1,1) f(2,2) f(3,3); f(2,1) f(3,1) f(3,2)]

h=[g(1,2) g(1,3) g(2,3); g(1,1) g(2,2) g(3,3); g(2,1) g(3,1) g(3,2)]

for x=2:1:s(1)-1;

for y=2:1:s(2)-1;

A(x,y)=[a(1)*I(x-1,y-1)+a(2)*I(x-1,y)+
        a(3)*I(x-1,y+1)+a(4)*I(x,y-1)+ ...
        a(5)*I(x,y)+a(6)*I(x,y+1)+
        a(7)*I(x+1,y-1)+a(8)*I(x+1,y)+a(9)*I(x+1,y+1)];

B(x,y)=[b(1)*I(x-1,y-1)+b(2)*I(x-1,y)+
        b(3)*I(x-1,y+1)+b(4)*I(x,y-1) ...
        +b(5)*I(x,y)+b(6)*I(x,y+1)+b(7)*I(x+1,y-1)
        + b(8)*I(x+1,y)+b(9)*I(x+1,y+1)];

C(x,y)=[c(1)*I(x-1,y-1)+c(2)*I(x-1,y)
        +c(3)*I(x-1,y+1)+c(4)*I(x,y-1) ...
        +c(5)*I(x,y)+c(6)*I(x,y+1)+
        c(7)*I(x+1,y-1)+c(8)*I(x+1,y)+c(9)*I(x+1,y+1)];

D(x,y)=[d(1)*I(x-1,y-1)+d(2)*I(x-1,y)+
        d(3)*I(x-1,y+1)+d(4)*I(x,y-1)+ ...
        d(5)*I(x,y)+d(6)*I(x,y+1)+
        d(7)*I(x+1,y-1)+d(8)*I(x+1,y)+d(9)*I(x+1,y+1)];

E(x,y)=[e(1)*I(x-1,y-1)+e(2)*I(x-1,y)
        +e(3)*I(x-1,y+1)+e(4)*I(x,y-1) ...
        +e(5)*I(x,y)+e(6)*I(x,y+1)+
        e(7)*I(x+1,y-1)+e(8)*I(x+1, y)+e(9)*I(x+1,y+1)];

F(x,y)=[f(1)*I(x-1,y-1)+f(2)*I(x-1,y)+
        f(3)*I(x-1,y+1)+f(4)*I(x,y-1) ...
        +f(5)*I(x,y)+f(6)*I(x,y+1)+
        f(7)*I(x+1,y-1)+f(8)*I(x+1,y)+f(9)*I(x+1,y+1)];

G(x,y)=[g(1)*I(x-1,y-1)+g(2)*I(x-1,y)+
        g(3)*I(x-1,y+1)+g(4)*I(x,y-1) ...
        +g(5)*I(x,y)+g(6)*I(x,y+1)+
        g(7)*I(x+1,y-1)+g(8)*I(x+1,y)+g(9)*I(x+1, y+1)];

H(x,y)=[h(1)*I(x-1,y-1)+h(2)*I(x-1,y)+
        h(3)*I(x-1,y+1)+h(4)*I(x,y-1) ...
        +h(5)*I(x,y)+h(6)*I(x,y+1)+
        h(7)*I(x+1,y-1)+h(8)*I(x+1,y)+h(9)*I(x+1, y+1)];

    end

end

W=max(max(max(max(max(max(A,B),C),D),E),F),G),H);
%% To calculate maximum

%% Plotting %%

figure(1),imshow(uint8(A)),figure(2),imshow(uint8(B))

figure(3),imshow(uint8(C)),figure(4),imshow(uint8(D))

figure(5),imshow(uint8(E)),figure(6),imshow(uint8(F))

figure(7),imshow(uint8(G)),figure(8),imshow(uint8(H))

figure(9),imshow(uint8(W)),figure(10),imshow(uint8(I))

%% Normalisation may be required %%
```

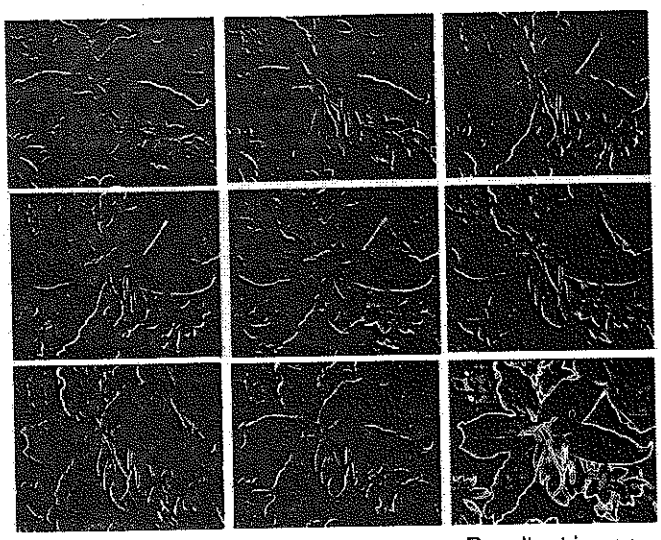


Fig. 7.4.5 : Compass operated images

7.5 Image Segmentation using the Second Derivative-The Laplacian

We have already seen the effects of using the first derivative of the image. It was shown that the first derivative does enhance the edges of the image. We now try to find out the effects of computing the second derivative on the edges.

We know,

$$\nabla F = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}$$

$$\therefore \quad \frac{\partial f}{\partial x} = f(x+1, y) - f(x, y)$$

$$\text{and} \quad \frac{\partial f}{\partial y} = f(x, y+1) - f(x, y)$$

The second derivative is given by

$$\nabla^2 F = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Where

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad \text{and}$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

$$\nabla^2 F = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\therefore \quad |\nabla^2 F| = [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)] \quad ...(7.5.1)$$

Considering the 3×3 neighbourhood

	y − 1	y	y + 1
x − 1	Z_1	Z_2	Z_3
x	Z_4	Z_5	Z_6
x + 1	Z_7	Z_8	Z_9

This equation in the discrete form reduces to

$$|\nabla^2 F| = [Z_8 + Z_2 + Z_6 + Z_4 - 4Z_5]$$

This equation can be implemented using a mask shown.

0	1	0
1	(−4)	1
0	1	0

This is known as the Laplacian operator. Some books reverse the signs of the coefficients of the mask i.e.,

0	−1	0
−1	(4)	−1
0	−1	0

There might be cases when, we would need to add higher weights at the center pixel (>4). While doing this we must make sure that the sum of the coefficients of the mask is zero since edges are high frequency components. Hence if we increase the weights at the centre, we would also have to change the values of the coefficients at the borders.

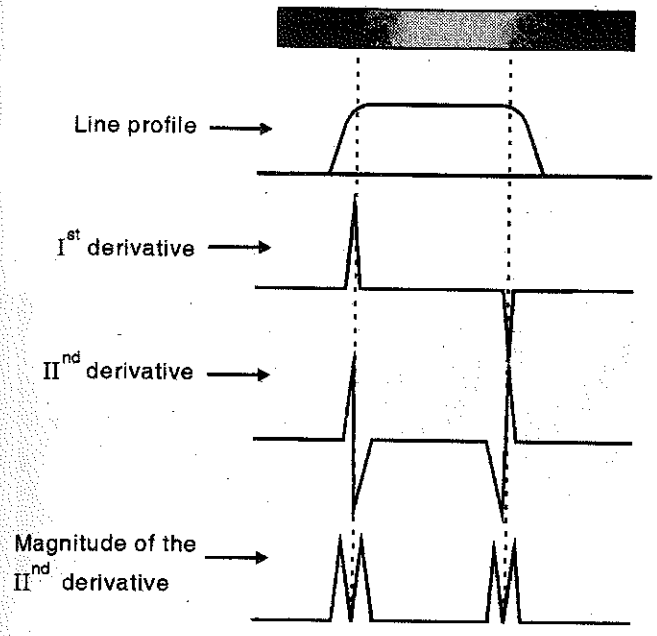


Fig. 7.5.1

One important thing to note about the Laplacian mask is that unlike the Sobel and Prewitt operator, they are isotropic filters. i.e., their response is independent of the direction of the discontinuties in the image. In other words, isotropic filters are rotation invarient. i.e., rotating the image and then applying the mask gives the same result as applying the mask to the image first and then rotating the result.

Now that we have the second derivative Laplacian mask with us, can we use it directly on the image ? the answer is No. The Laplacian is not generally used in its original form as an edge detector for mainly two reasons.

(1) Since the Laplacian is a second derivative filter, it is very sensitive to noise (much more than the first derivative). Hence in an image, if there is any noise present, the Laplacian gives very large values and ruins the entire image.

(2) The magnitude of the Laplacian produces double edges, which is an undesireable effect. Let us explain what we mean by double edges. Consider a strip of image shown in Fig. 7.5.1. It is seen that the magnitude of the 2nd derivative produces two peaks for a single edge.

The zero crossing property of the Laplacian is used to detect edges.

As stated earlier, the Laplacian mask evokes very strong response to stray noise pixels. Hence if some kind of noise cleaning is done prior to the application of the Laplacian operator, better results could be obtained. In practice the Laplacian is preceded by a smoothing operation (Gaussian smoothing). The resultant algorithm is commonly known as a Laplacian of Gaussian (LoG) or Marr-Hildreth operator. (Marr D.C. and Hildreth presented a paper "Theory of Edge Detection" in 1980 which explains in this concept)

Let us explain the mathematics of this.

What we just said was that, we first smoothen the image using a Gaussian function (Remember low pass Gaussian filter?) and then take the Laplacian (second derivative of the image).

A simple method to implement this practically, is to combine the two i.e. combine the action of a Gaussian function and a second derivative function.

Consider the Gaussian function

$$h(x, y) = e^{-\left(\frac{x^2 + y^2}{2\sigma^2}\right)} \qquad \ldots(7.5.2)$$

where σ is the standard deviation and x and y are the two variables (spatial coordinates). It is σ that determines the degree of blurring.

Let $x^2 + y^2 = r^2$

$$\therefore \quad h(r) = e^{-\left(\frac{r^2}{2\sigma^2}\right)}$$

The Laplacian of h i.e., the second derivative of h with respect to r is now calculated.

We first calculate the first derivative w.r.t. r

$$\frac{\partial}{\partial r} h(r) = \frac{\partial}{\partial r} e^{-r^2/2\sigma^2} = e^{-r^2/2\sigma^2} \cdot \frac{\partial}{\partial r}\left(\frac{-r^2}{2\sigma^2}\right)$$

$$\frac{\partial}{\partial r} h(r) = e^{-r^2/2\sigma^2} \cdot \left(\frac{-2r}{2\sigma^2}\right) = e^{-r^2/2\sigma^2} \cdot \left(\frac{-r}{\sigma^2}\right)$$

Now taking the second derivative, we get

$$\frac{\partial^2}{\partial r^2} h(r) = \frac{\partial}{\partial r}\left(\frac{-r}{\sigma^2} e^{-r^2/2\sigma^2}\right) = \frac{-1}{\sigma^2}\frac{\partial}{\partial r}\left(re^{-r^2/2\sigma^2}\right)$$

$$= \frac{-1}{\sigma^2}\left(re^{-r^2/2\sigma^2}\left(\frac{-r}{\sigma^2}\right) + e^{-r^2/2\sigma^2}\right)$$

$$= \frac{1}{\sigma^2}\left(\frac{r^2 e^{-r^2/2\sigma^2}}{\sigma^2} - e^{-r^2/2\sigma^2}\right)$$

$$= \frac{1}{\sigma^2} e^{-r^2/2\sigma^2}\left(\frac{r^2}{\sigma^2} - 1\right)$$

$$\frac{\partial^2}{\partial r^2} h(r) = \frac{r^2 - \sigma^2}{\sigma^4} e^{-r^2/2\sigma^2}$$

$$\therefore \nabla^2 h = \left(\frac{r^2 - \sigma^2}{\sigma^4}\right) e^{-r^2/2\sigma^2} \quad ...(7.5.3)$$

This is the Laplacian of a Gaussian (LoG). $\nabla^2 h$ can be easily plotted using MATLAB.

Due to the shape of the LoG, it commonly referred to as the Mexican Hat function.

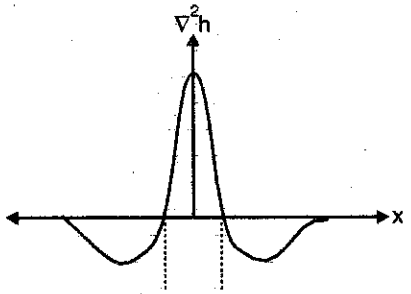


Fig. 7.5.2

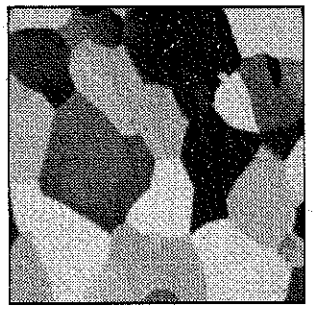
It can be seen that $\nabla^2 h$ has a large positive value at the centre and negative values at the peripheries.

We can approximate this by using a 5 × 5 mask as shown.

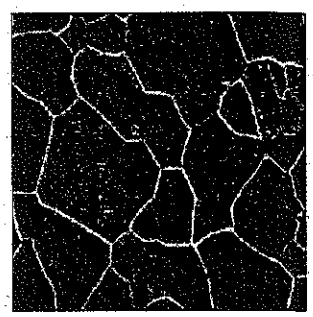
Remember, this mask is not unique. We could use any values that approximate the shape of the LoG.

One thing that needs to be kept in mind is that since our intention is to enhance the edges, the sum of coefficients of the mask should be zero.

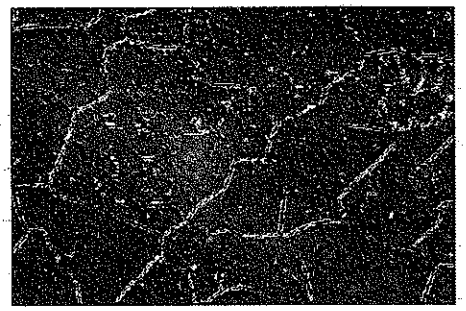
0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0



(a) Original image



(b) Image using LoG operator



(c) Zero crossing image

Fig. 7.5.3

7.6 Edge Linking

It was seen that gradient operators like Roberts, Sobel, Prewitts as well as the Compass operators enhance the edges. When we implement these filters practically, there are usually breaks in lines. Due to noise, there are spurious intensity discontinuties because of which the output of a gradient filtered image would have pixels that lie on the edge as well as pixels that represent noise.

Due to this fact, edge detection algorithms are generally followed by edge linking procedures which form edges (lines) from edge pixels. This is done by joining the edge pixels.

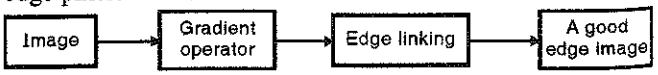


Fig. 7.6.1

We shall discuss two major techniques for edge detection.

7.6.1 Local Processing

A simple approach for edge linking is to analyse pixels in a neighbourhood (3×3 or 5×5) of the image that have undergone edge detection. All points that share some common properties are linked together.

The two common properties that we consider to link pixels together are :

(1) Strength of the response of the gradient operator.

(2) The direction of the gradient.

(1) **Strength of the Gradient :** We have already studied gradient operators like Prewitt and Soble operators.

$$|\nabla F| = [F_x^2 + F_y^2]^{1/2}$$

Where $F_x \rightarrow$ gradient in the x-direction

$F_y \rightarrow$ gradient image in the y-direction

A pixel in the neighbourhood of pixel $\nabla F(x, y)$ is linked to the pixel (x, y) if

$$|\nabla F(x, y) - \nabla F(x', y')| \leq T \quad \ldots(7.6.1)$$

Where T is a non-negative threshold.

By ∇F (x, y) we simply mean the image obtained after using the Sobels or Prewitts mask.

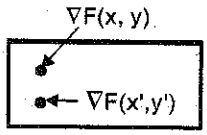


Fig. 7.6.2

(2) **Direction of the Gradient :** The direction gradient of vector ∇F is given by

$$\alpha(x, y) = \tan^{-1}\left(\frac{F_x}{F_y}\right) \quad \ldots (7.6.2)$$

Where $F_x \rightarrow$ x gradient image

$F_y \rightarrow$ y gradient image

A pixel in the neighbourhood of pixel $\alpha(x, y)$ is linked to the pixel (x, y) if they have a similar angle given by

$$|\alpha(x, y) - \alpha(x', y')| < A$$

Only if both these condition are satisfied, are the pixels linked together.

Hence for every pixel, all its 8 neighbours are checked for these two conditions.

7.6.2 Hough Transform

Hough transform has emerged over the past decade as a method of choice for pixel linking and curve detection. Hough presented this transform in the year 1962.

Given a set of discrete pixels, the Hough transform checks if these points lie on a straight line and if yes, it draws a line joining all these points.

Hough transform is best explained by an example. Consider a point (x_1, y_1). A line passing through this point can be written in the slope-intercept form as $y_1 = ax_1 + b$.

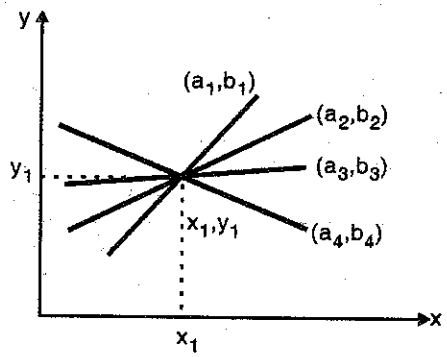


Fig. 7.6.3

Using this equation and varying the values of a and b, infinite number of lines pass through this point (x_1, y_1).

However, if we write this equation as

$$b = -ax_1 + y_1$$

and consider the ab plane instead of the xy plane, we get a single line for a point (x_1, y_1)

This entire line in the ab plane is due to a single point in the xy plane and different values of a and b.

Now consider another point (x_2, y_2) in the xy plane.

The slope intercept equation of this line is

$$y_2 = ax_2 + b$$

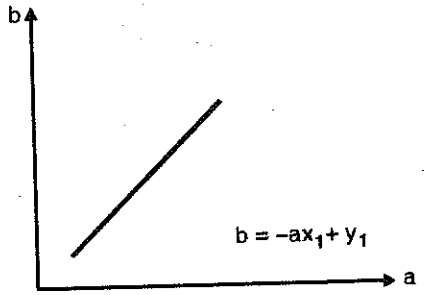


Fig. 7.6.4

Writing this equation in terms of the ab plane we get

$$b = -ax_2 + y_2$$

This is another line in the ab plane. These two line will intersect each other somewhere in the ab plane only if they are a part of a straight line in the xy plane.

The point of intersection in the ab plane is noted (a′, b′).

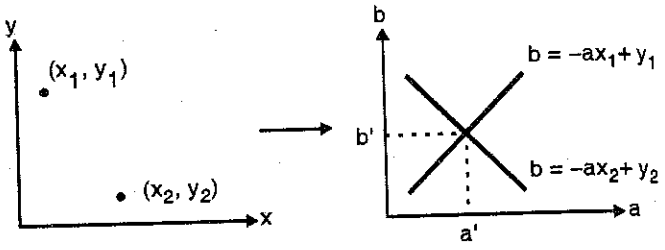


Fig. 7.6.5

Using this (a′, b′) in the standard slope-intercept form i.e., $y = a'x + b'$, we get a line that passes through points (x_1, y_1) and (x_2, y_2) in the xy plane!!!!

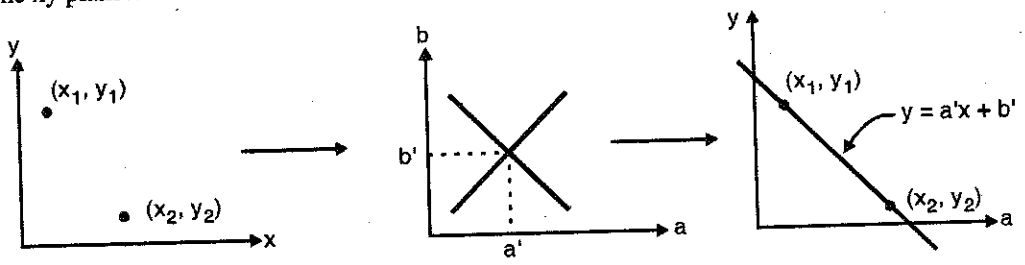


Fig. 7.6.6

In fact if we have n points that lie on a straight line, we get n lines (each line representing a point) in the ab plane and all these lines would intersect at a single point (a′, b′) in the ab plane. Using these values of a′ and b′ in equation $y = a'x + b'$, we would get a line that would pass through all these points in the xy plane.

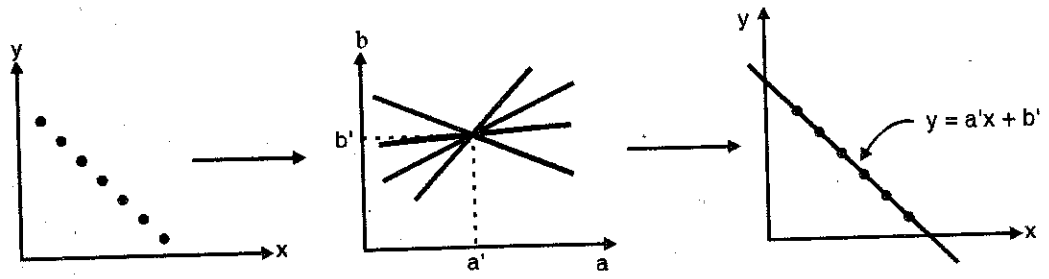


Fig. 7.6.7

The ab plane is called the parameter space.

This parameter space is subdivided into accumulator cells as shown.

Here (a_{max}, a_{min}) and (b_{max}, b_{min}) are the expected ranges of slope and intercept values.

Let us solve a couple of examples to be sure that the things just mentioned are well understood.

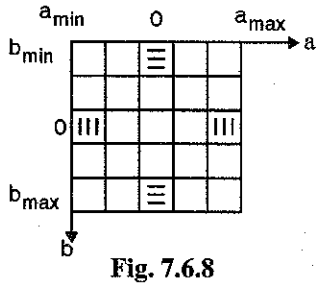


Fig. 7.6.8

Ex. 7.6.1

Given four points in the xy plane with the following coordinates (1, 1), (2, 2), (3, 3), (4, 4) use Hough transform to join these points.

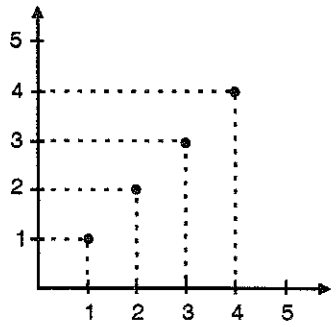


Fig. P. 7.6.1

Soln. :

It is obvious that these four points do form a straight line. But we come to this conclusion only because our visual system is very complex and it helps us to visualise a line joining these points. But the computer needs to be fed with a good algorithm so as to join these lines.

We shall solve this example on a graph paper to get accurate results.

We use the equation

$$y = ax + b$$

Changing the coordinates we get

$$b = -ax + y$$

x	y		
1	1	$\rightarrow b = -a + 1$	...(1)
2	2	$\rightarrow b = -2a + 2$	...(2)
3	3	$\rightarrow b = -3a + 3$	...(3)
4	4	$\rightarrow b = -4a + 4$	...(4)

Using Equations (1), (2), (3), (4) we draw 4 lines in the parameter space ab, each representing a point from the *xy* plane.

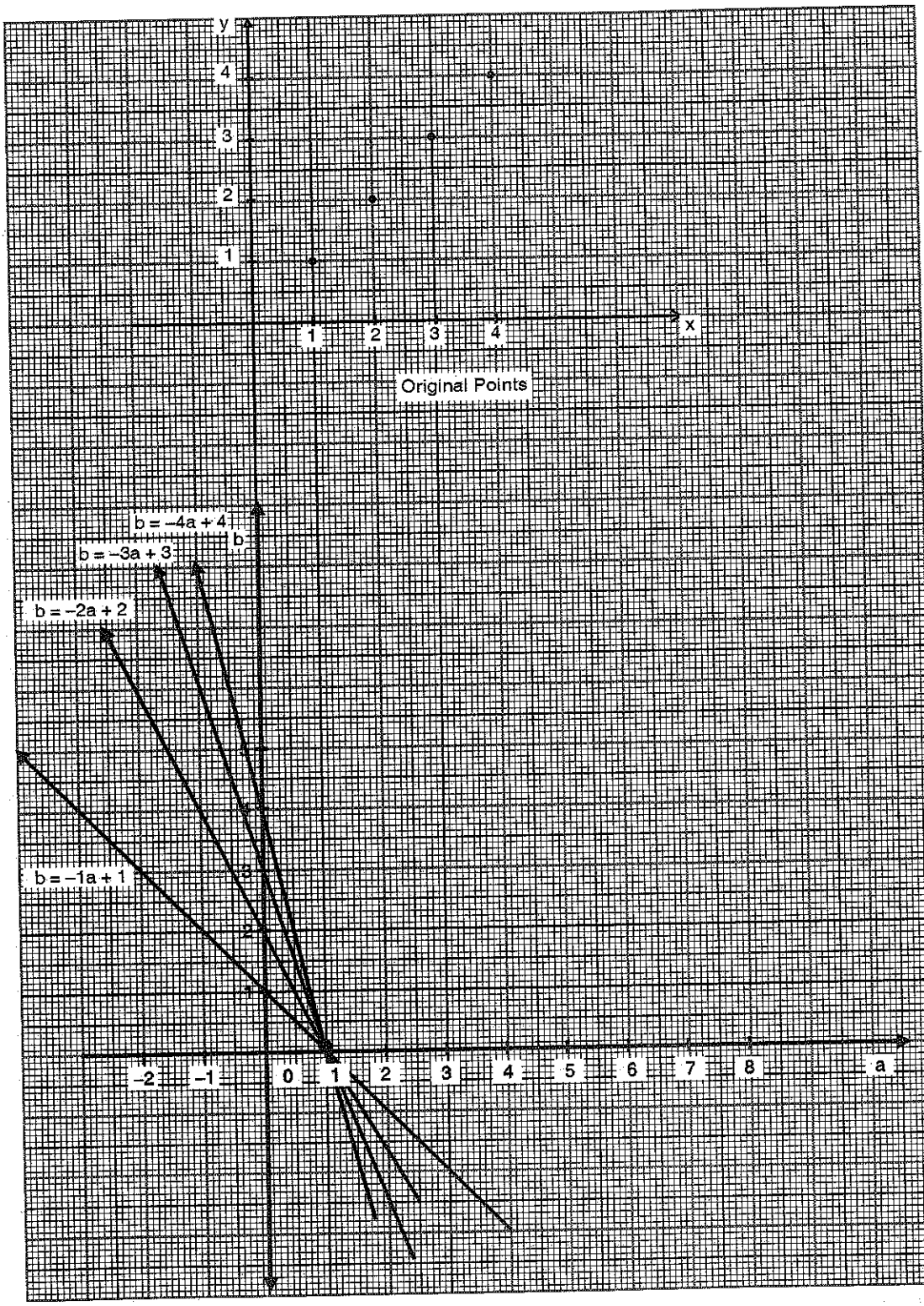


Fig. P. 7.6.1(a)

From the graph, we get a single intersection point which has coordinates

(1, 0) i.e. a = 1, b = 0. Taking these values of a and b, we draw a line using the equation.

$$y = ax + b$$

i.e., $y = 1 \cdot x + b$

i.e., $y = x$

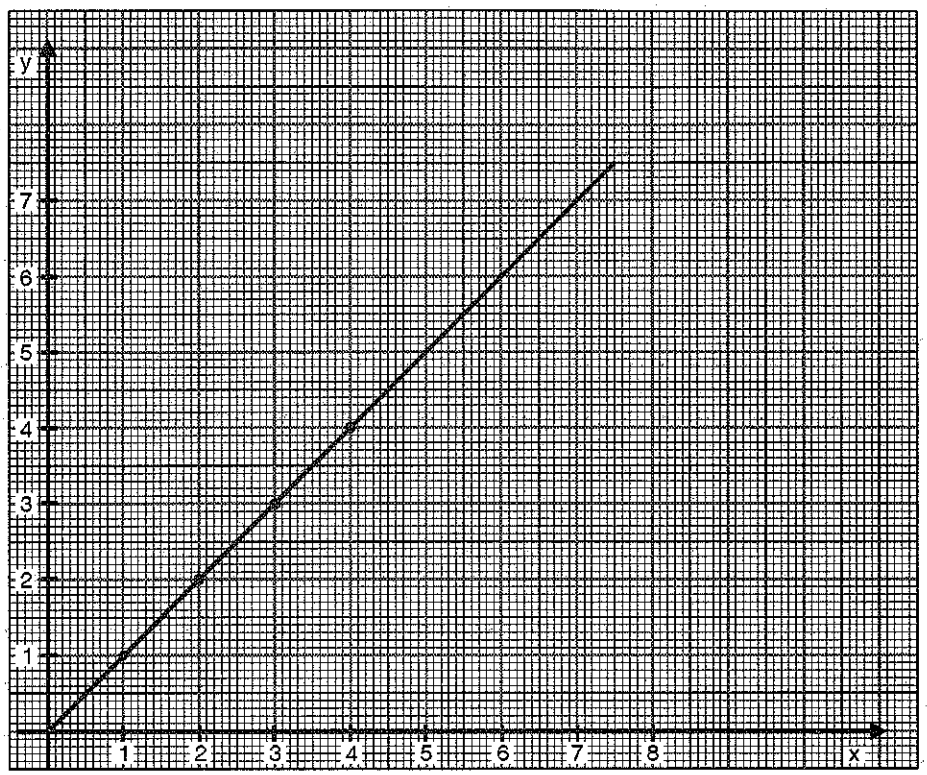


Fig. P. 7.6.1(b)

This line, as can be seen, passes through points (1, 1), (2, 2), (3, 3), (4, 4).

Ex. 7.6.2

Given 5 points, use Hough transform to draw a line joining these points.

(1, 4), (2, 3), (3, 1), (4, 1), (5, 0)

Soln.:

The first step is to draw the given points as they are

We now use the standard slope-intercept equation of a straight line y = ax + b.

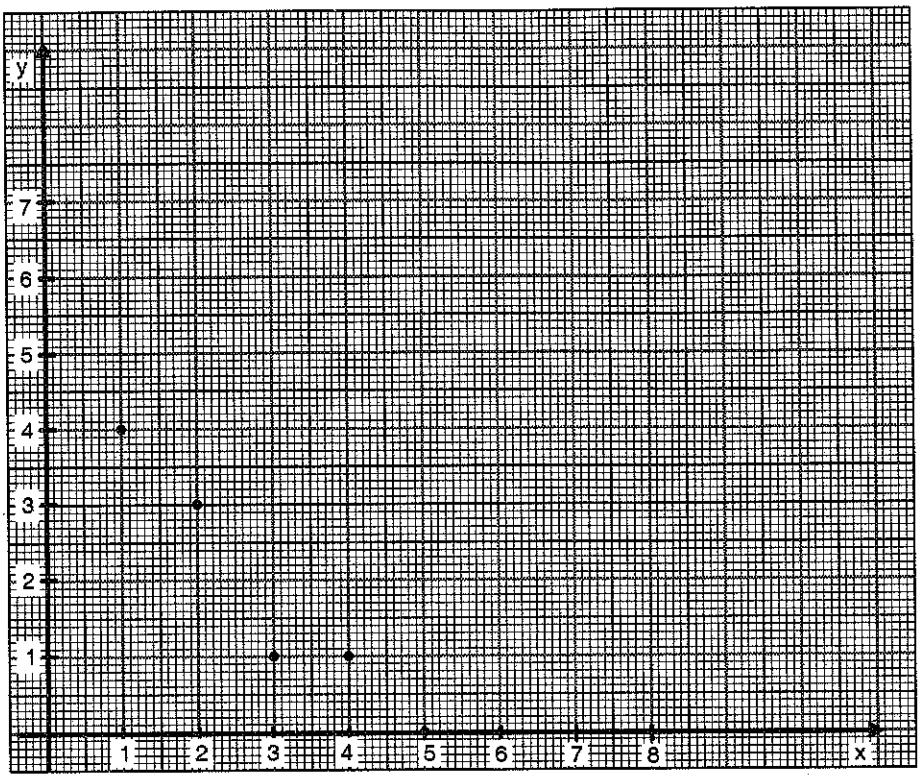


Fig. P. 7.6.2

Using the parameter space, we get

$b = -ax + y$

x	y		
1	4	$\rightarrow b = -a + 4$	...(1)
2	3	$\rightarrow b = -2a + 3$	...(2)
3	1	$\rightarrow b = -3a + 1$	...(3)
4	1	$\rightarrow b = -4a + 1$	...(4)
5	0	$\rightarrow b = -5a$	...(5)

Using Equations (1), (2), (3), (4) and (5), we draw 5 lines in the parameter space ab, each representing a point from the xy plane.

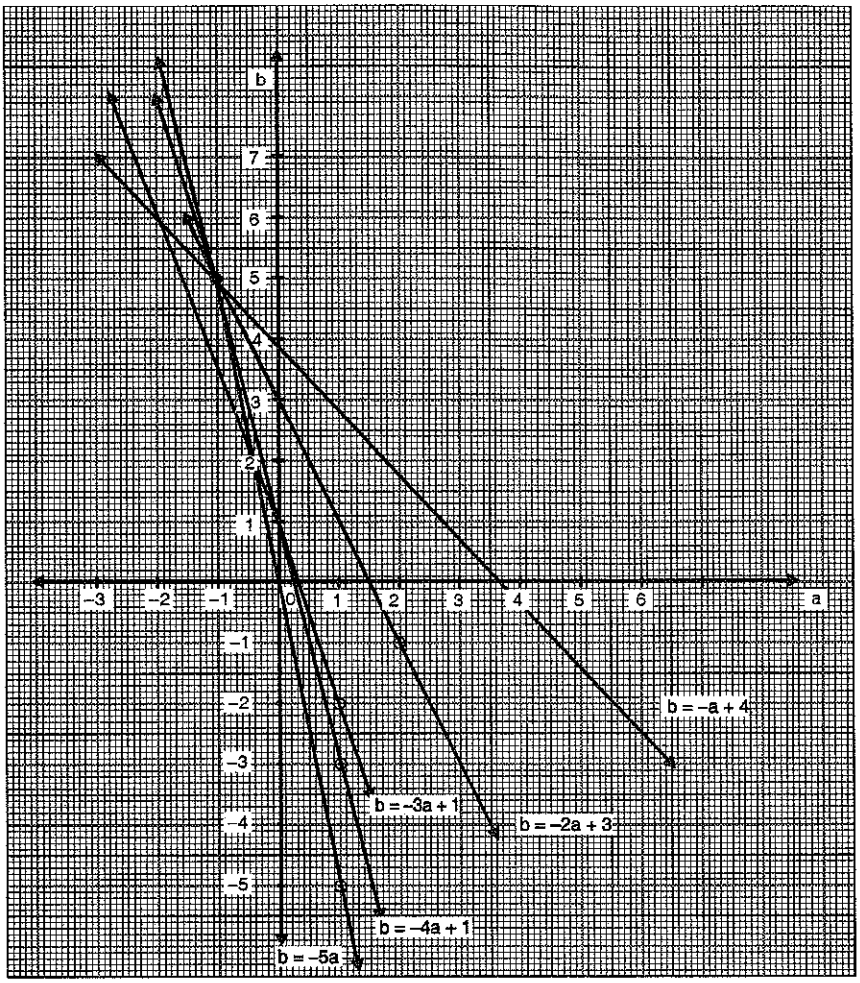


Fig. P. 7.6.2(a)

If we observe carefully, barring line b = – 3a + 1, all the other lines intersect each other at a single point (– 1, 5) i.e., a = –1, b = 5.

We take this value of a and b and use it in the equation.

$$y = ax + b$$

i.e., $y = -1x + 5$

We get the following graph.

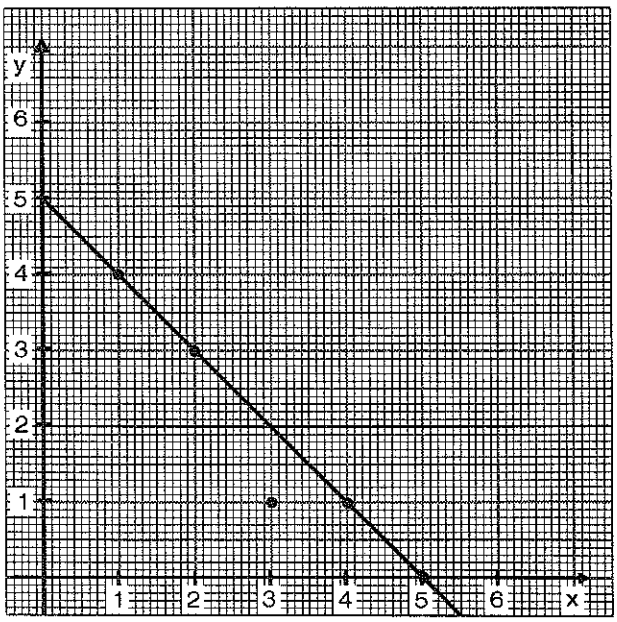


Fig. P. 7.6.2(b)

We observe that the line passes through all the given points except the (3, 1) point.

As (3, 1) did not lie on the straight line, the line representing this point in the ab plane did not intersect the other lines at the same point.

There is one problem with using the intercept equation $y = ax + b$ and $b = -ax + y$

If the points lie on a vertical line, the slope tends to infinity and hence a′ and b′ coordinates cannot be identified.

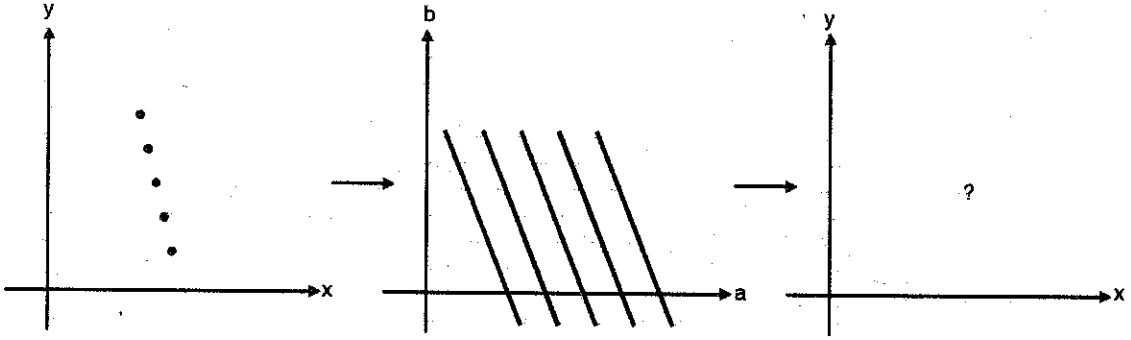


Fig. P. 7.6.2(c)

Hence this technique fails to draw lines through points which are placed vertically.

Try solving this example.

Ex. 7.6.3

Using Hough's transform, find the line passing through the maximum number of points given as, (3, 4), (0, – 4), (1, 4), (6, 12), (4, 1), (1.5, 0), (– 2, 0), (– 1, – 3), (3, – 2).

Soln. :

We write the points in a tabular form,

x	y
3	4
0	– 4
1	4
6	12
4	1
1.5	0
– 2	0
– 1	– 3
3	– 2

This example is solved using the equation of a straight line $y = ax + b$.

Changing the coordinates we get, $b = -ax + y$.

Using the given values of x and y, we get the following equations,

x	y		
3	4	$b = -3a + 4$	...(1)
0	– 4	$b = -4$	...(2)
1	4	$b = -a + 4$	...(3)
6	12	$b = -6a + 12$	...(4)
4	1	$b = -4a + 1$	...(5)
1.5	0	$b = -1.5\,a$	...(6)
– 2	0	$b = +2a$	...(7)
– 1	– 3	$b = a - 3$	...(8)
3	– 2	$b = 3a - 2$	...(9)

We draw lines using Equations (1) to (9) in the parameter space ab. Each line represents a point in the xy plane.

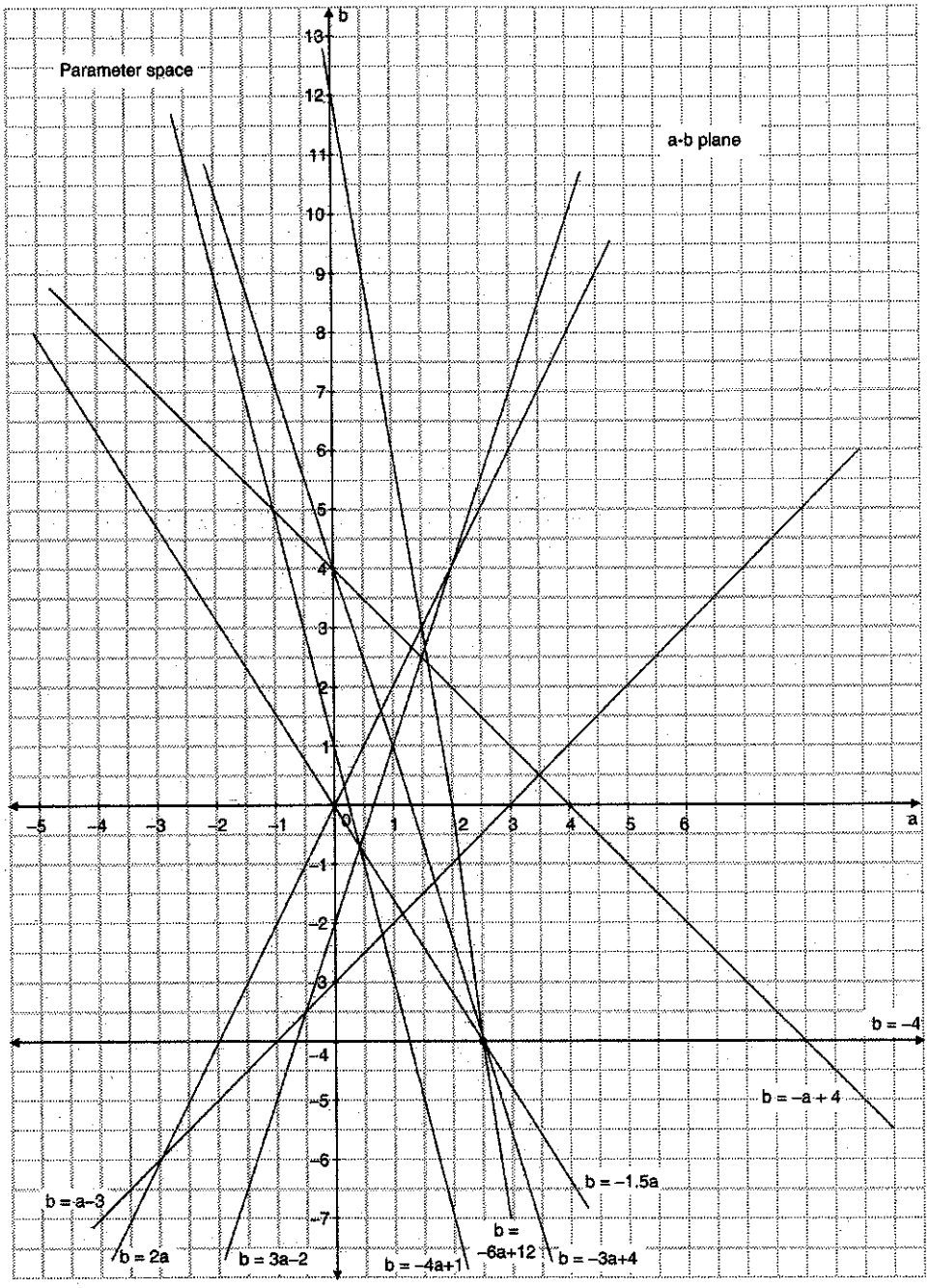


Fig. P. 7.6.3

From Fig. P. 7.6.3 it is clear that a maximum of four lines pass through point (2.6, – 4).

$\therefore$ a = 2.6, b = – 4

We take this value of a and b and use it in equation y = ax + b to draw a line.

$\therefore$ y = 2.6, x – 4

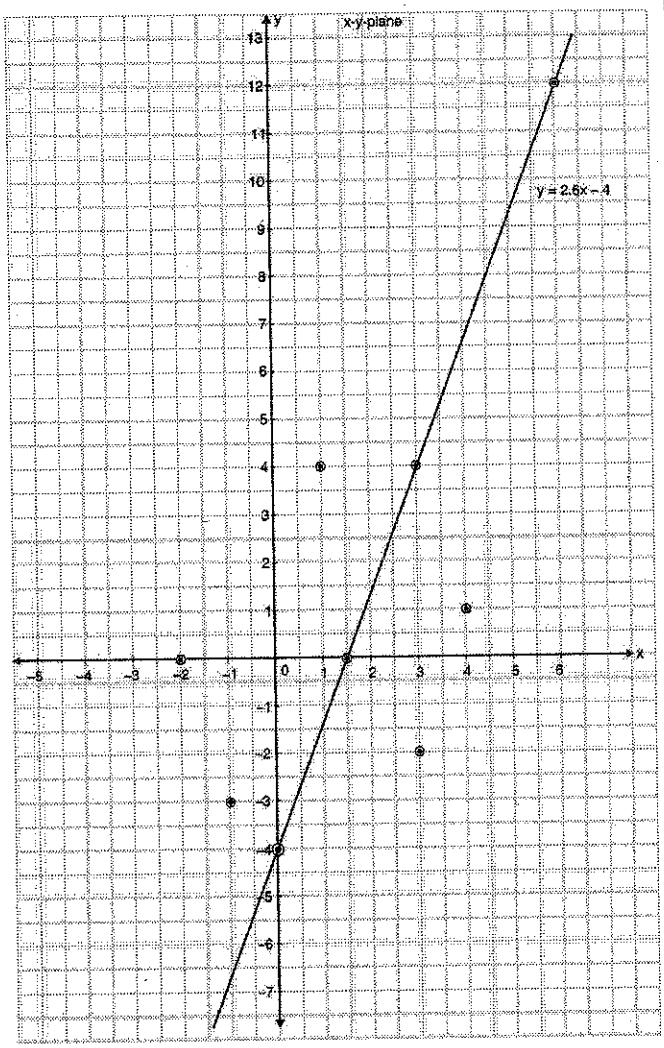


Fig. P. 7.6.3(a)

From the Fig. P. 7.6.3(a), we observe that 4 points of the given data set lie on a straight line.

Ex. 7.6.4

Draw a line joining point (2, 1), (2, 2), (2, 3), (2, 4) using Hough transform.

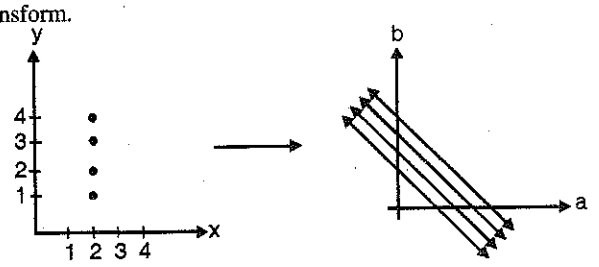


Fig. P. 7.6.4

Soln. :

Try this example and you will find that these lines do not intersect.

Hough studied this problem in detail and soon discovered that this problem could be rectified by using the normal representation of the straight line i.e., the polar coordinates (ρ, θ)

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

i.e., instead of using the equation $y = ax + b$, if we use the equation $x \cos \theta + y \sin \theta = \rho$, the problem just mentioned could be eliminated.

We shall derive the normal representation of the line equation. This must have been done in the X^{th} standard class, but that was a long time ago. It is a good idea to prove this result.

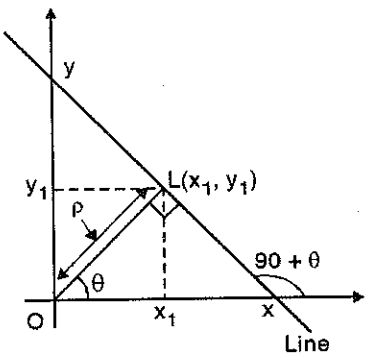


Fig. P. 7.6.4(a)

Normal equation of a straight line

Given :

(i) The perpendicular distance from the origin ρ

(ii) Angle (θ) made by the perpendicular line ρ, with the x-axis

Proof : OL is ⊥ to the given line

$$l(OL) = \rho \quad \angle LOX = \theta$$

Coordinates of L are (x_1, y_1)

Here $\frac{x_1}{\rho} = \cos \theta$ and $\frac{y_1}{\rho} = \sin \theta$

$$\therefore x_1 = \rho \cos \theta, \; y_1 = \rho \sin \theta$$

Slope of the given line is $\tan (90 + \theta)$

We know that a line is given by the equation

$$y - y_1 = m(x - x_1) \rightarrow m \text{ is the slope of the line}$$

$$\therefore \quad y - \rho \sin\theta = \tan(90 + \theta) \cdot [x - \rho \cos\theta]$$

$$y - \rho \sin\theta = -\cot\theta \cdot [x - \rho \sin\theta]$$

$$y - \rho \sin\theta = \frac{-\cos\theta}{\sin\theta} [x - \rho \cos\theta]$$

i.e. $y \sin\theta - \rho \sin^2\theta = -x \cos\theta + \rho \cos^2\theta$

$$\therefore \quad x \cos\theta + y \sin\theta = \rho[\cos^2\theta + \sin^2\theta]$$

$$\therefore \quad x \cos\theta + y \sin\theta = \rho$$

This is the equation of a straight line - normal form

Therefore, a straight line can be defined by the angle θ and the perpendicular distance ρ from the origin.

Ex. 7.6.5

Draw a line having $\rho = 5$ and $\theta = 45°$.

Soln. :

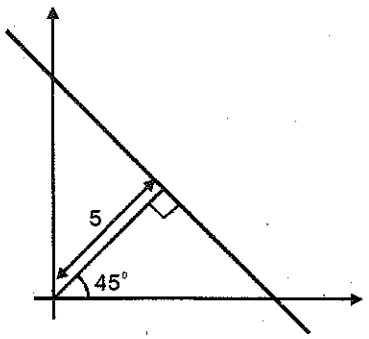


Fig. P. 7.6.5

The method of drawing the line joining the given points is the same as discussed before. The only difference is that in the parameter space we would not get a straight line. Because of the equation $x \cos\theta + y \sin\theta = \rho$, the parameter space ($\rho$-$\theta$ in this case) will have sinusoids. The point at which maximum numbers of sinusoids intersect is noted as (ρ', θ').

These values are then plugged into the equation $x \cos\theta' + y \sin\theta' = \rho'$

Values of θ are varied from – 90 to + 90 in the equation and values of ρ are plotted in the parameter space.

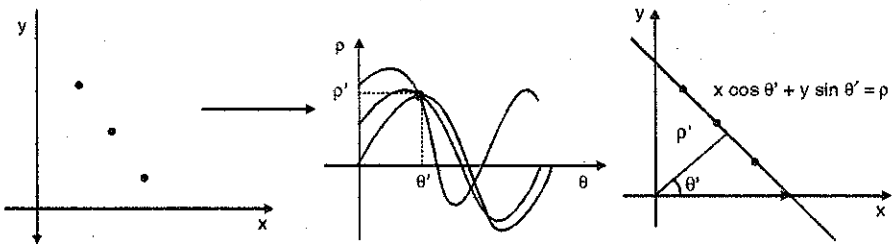


Fig. P. 7.6.5(a)

Let us solve an example to cement our thoughts.

Ex. 7.6.6

For the given image use Hough transform to form the main edges.

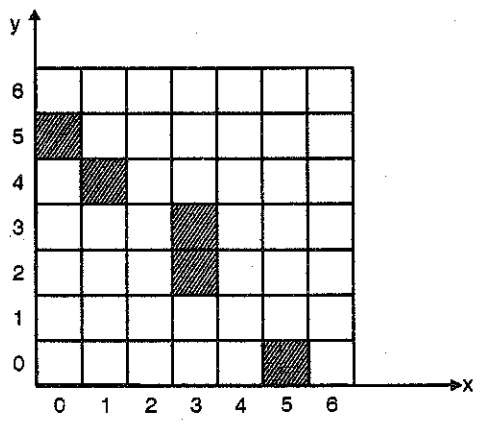


Fig. P. 7.6.6

Soln. :

The pixel coordinates of interest are (0, 5), (1, 4), (3, 2), (3, 3), (5, 0)

For each point, we shall use the equation

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

Using the first value of x and y i.e. (0, 5) we get

$$\rho = 0 \cos\theta + 5 \sin\theta$$

We now vary θ from –90 to +90 and get the values of ρ.

Similarly, equations for all the 5 points are

$(0, 5) \rightarrow \rho = 0 \cos\theta + 5 \sin\theta$

$(1, 4) \rightarrow \rho = 1 \cos\theta + 4 \sin\theta$

$(3, 2) \rightarrow \rho = 3 \cos\theta + 2 \sin\theta$

$(3, 3) \rightarrow \rho = 3 \cos\theta + 3 \sin\theta$

$(5, 0) \rightarrow \rho = 5 \cos\theta + 0 \sin\theta$

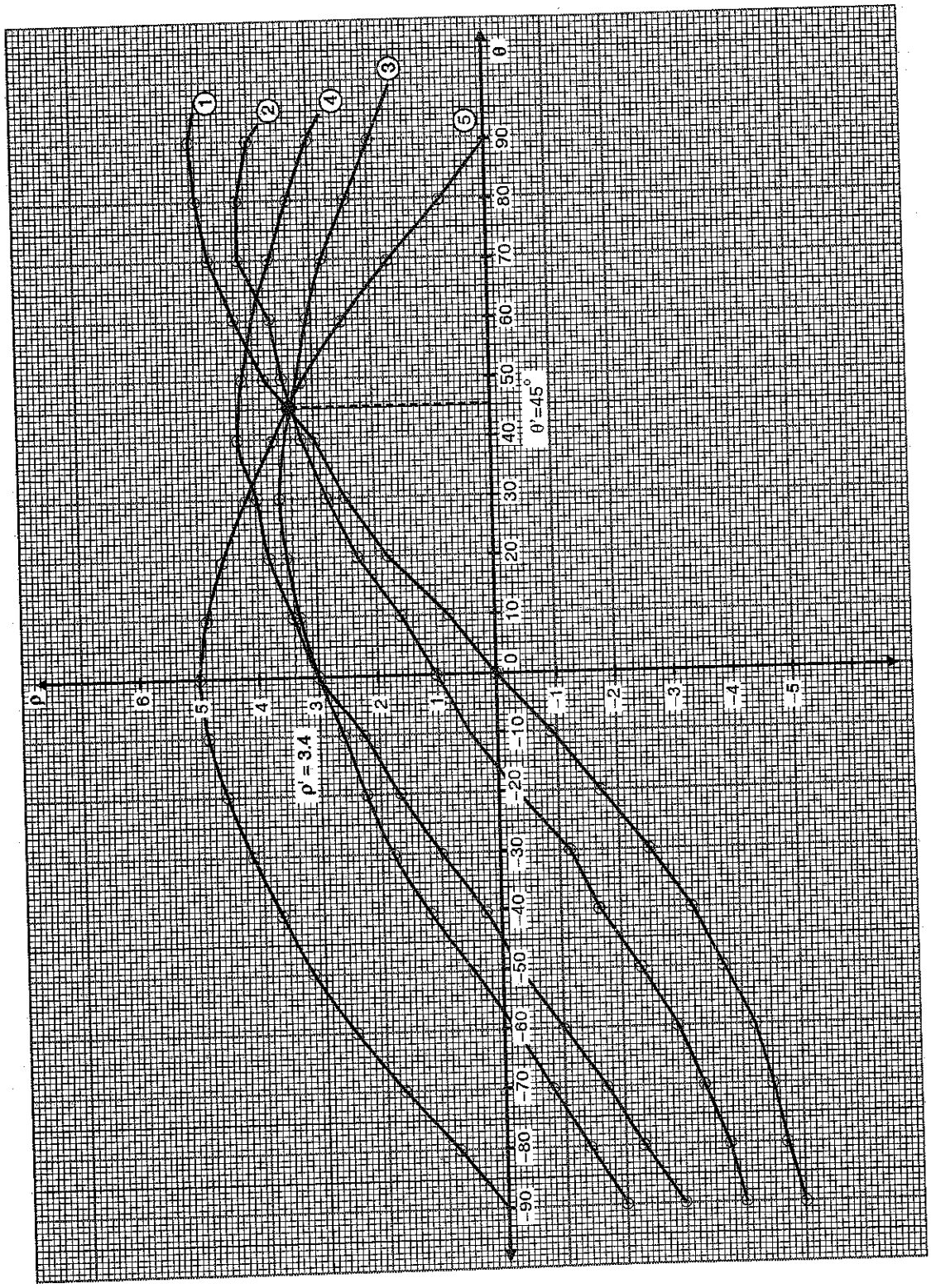


Fig. P. 7.6.6(a)

Four of the five lines intersect at the point θ = 45 and ρ = 3.4

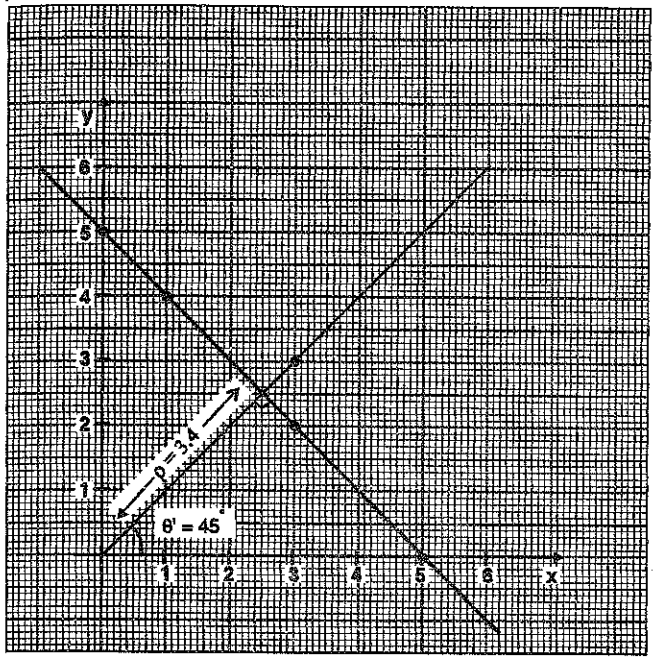


Fig. P. 7.6.6(b)

As can be seen, the line passes through 4 points. Point (3, 3) is kept outside as it does not lie on the line.

One thing to note is that lines are drawn using the equation $\rho = x \cos \theta + y \sin \theta$ and hence will be drawn from one end to the other.

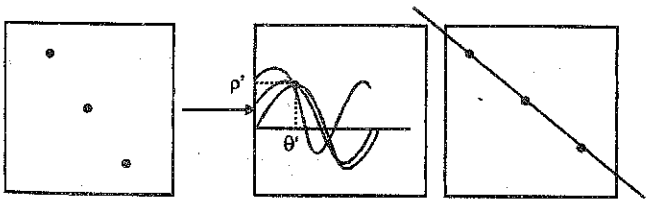


Fig. P. 7.6.6(c)

Hence after the implementation of the Hough transform, we need to trim the line drawn so that it passes only through the required points and ends at the last point.

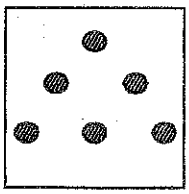


Fig. P. 7.6.6(d)

An image of the kind shown would yield 6 curves in the parameter space and would have 3 different points of intersection. These three points would form the 3 lines in the output image. This is as shown in Fig. P. 7.6.6(e).

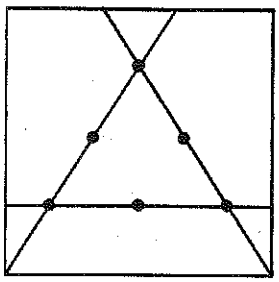
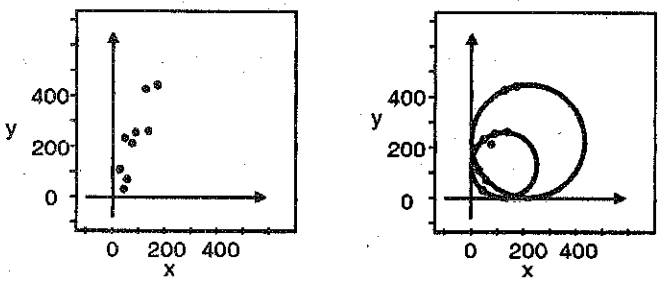


Fig. P. 7.6.6(e)

What we have discussed so far is a method of drawing straight lines using the normal equation of the line.

The same principle can be used to detect more complex curves. For example, if we want to detect all possible circular collections of points in an image, triplets of points could be used to define a circle.

We use the equation of a circle $(x - a)^2 + (y - b)^2 = r^2$ which could then be parameterized as a triplet (a, b, r). The point (a, b) is the center of the circle and r is its radius.



(i) The nine points through which arcs may pass **(ii) The data and the tangent circles found by the Hough transform**

Fig. P. 7.6.6(f)

7.7 Global Processing via Graph - Theoretic Techniques

In this technique we discuss a global approach for detecting edges. We represent edges in the form of a graph

and then search for a path which has the lowest cost. This low-cost path would eventually correspond to the most significant edge. This statement will be clear in due course of time.

We shall explain this by taking an example of a 3×3 image.

We assume two pixels at a time p and q such that p and q are 4-neighbours (refer section 7.8 for neighbours). Each edge element, defined by pixels p and q, has an associated cost.

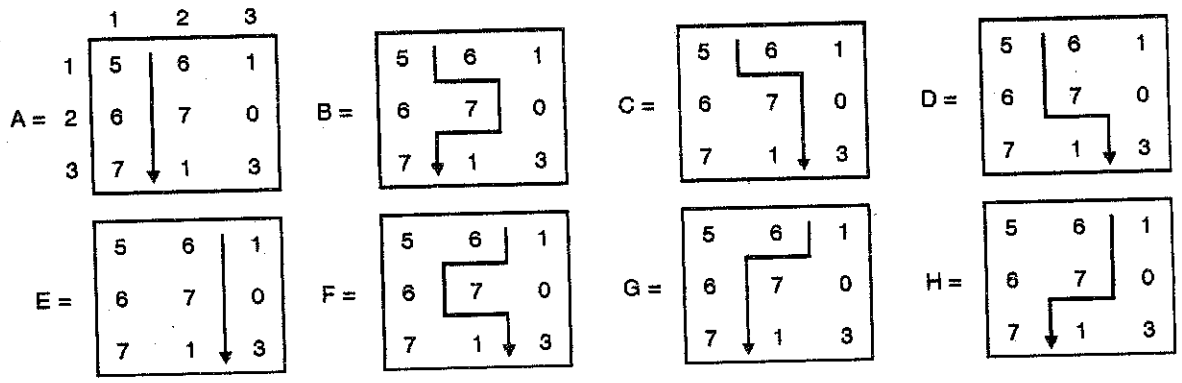
$$I = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 5 & 6 & 1 \\ 2 & 6 & 7 & 0 \\ 3 & 7 & 1 & 3 \end{array}$$

$$c(p, q) = Max(I) - [f(p) - f(q)]$$

Where Max(I) is the maximum grey level in the image.

To simplify the discussion we assume that the edges start from the top row and terminate in the last row. Due to this the first element of an edge can only be between (1, 1) and (1, 2) or (1, 2) and (1, 3).

Let p be on the right hand side of the direction of the path. All possible directions are shown below.



We draw a graph representing each edge.

Let us draw the graph for the first edge. For edge between 5 and 6 i.e., (1, 1) (1, 2), p = 5 and q = 6 (since p is on right hand of the direction).

The cost is Max(I) – [p – q] = 7 – [5 – 6] = 8. The cost of the entire edge is A is shown in Fig. 7.7.1.

Cost of A = 8 + 8 + 1 = 17.

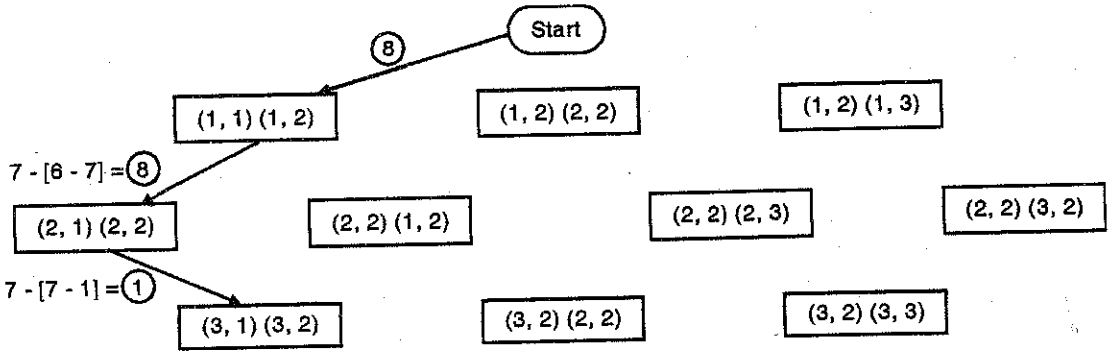


Fig. 7.7.1

Let us draw the graph for B

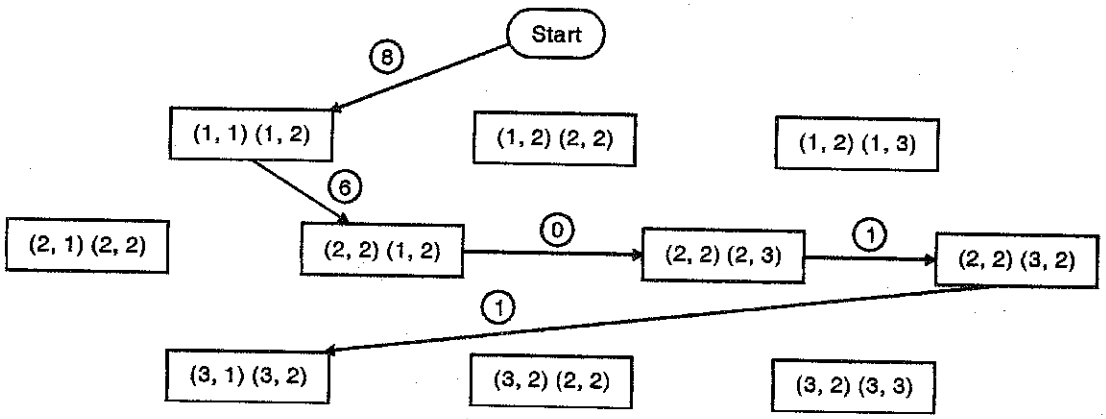


Fig. 7.7.2

Cost of B = 8 + 6 + 1 + 1 = 16.

Similarly we draw the graph for the remaining edges and calculate the cost of each

A = 17, B = 16, C = 23, D = 38, E = 11, F = 36, G = 17, H = 4

The combined graph is shown in Fig. 7.7.3.

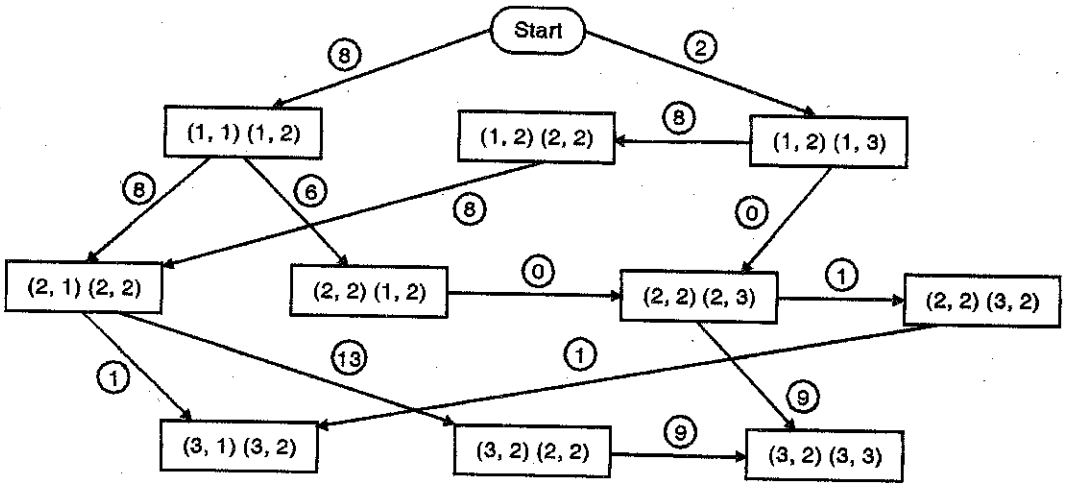


Fig. 7.7.3

The least cost path represents the edge in H. As stated earlier, the lowest cost path represents the most significant edge.

5	6	1
6	7	0
7	1	3

Note : Taking the modulus simplifies the example.

Try solving the same example by using the formula,

$C(p, q) = \text{Max}(t) - |f(p) - f(q)|$

7.8 Connectivity

Before we proceed to discuss region based segmentation, we need to know relationships between pixels in a digital image.

Consider a pixel p(x, y). This pixel p has two horizontal and two vertical neighbours which share a surface with p(x, y).

This set of four pixels is known as the 4-neighbourhood of pixel p(x, y), where each pixel is said to

be at a unit distance from p(x, y). This set of pixels is denoted as $N_4(p)$. In a similar manner, apart from these 4-neighbours, there are another four diagonal pixels which touch p(x, y) at the corners $N_D(p)$.

	p(x – 1, y)	
p(x, y – 1)	p(x, y)	p(x, y + 1)
	p(x + 1, y)	

$N_4(p)$

This set of eight pixels (The 4-neighbours as well as the diagonal pixels) is called the 8-neighbourhood of pixel p(x, y) and is denoted as $N_8(p)$

$$\therefore N_8(p) = N_4(p) + N_D(p).$$

Connectivity is an important concept that is used in segmentation. When we started off with this chapter, it was stated that segmentation is used for machine vision.

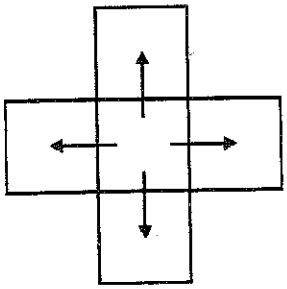
p(x –1, y – 1)		p(x –1, y + 1)
	p(x, y)	
p(x + 1, y – 1)		p(x + 1, y + 1)

$N_D(p)$

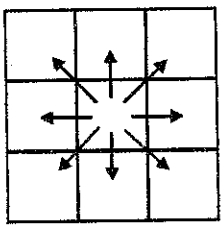
In segmentation, the machine (computer) scans the image to form different regions. This scanning of the image by the machine is based on the connectivity that the user or the code specifies. To establish if two pixels are connected, it must be determined whether they are neighbours (4-neighbours or 8-neighbours) and if their grey levels satisfy a specified criteria.

Two regions that touch only at a corner can be considered to be a single region or two distinct regions : How they are considered depends on the definition of connectivity used.

Two pixels p and q with some common criteria are said to be 4-connected if they share a side. i.e., two pixels p and q with some common criteria are 4-connected if q is in the set $N_4(p)$.



Two pixels p and q with some common criteria are said to be 8-connected if they share either a side or a corner. i.e, two pixels p and q with some common criteria are 8-connected if q is in the set $N_8(p)$.



Let us take an example. Given the criteria that p = q = 1 for the region to be connected, check whether the two region S_1 and S_2 are connected.

S_1	S_2
0	0
1 (p)	0
0	1 (q)

Even though p and q are both equal to one, these two regions will not be shown connected, if we use a 4-connectivity algorithm. This is because, in the 4-connectivity algorithm, we only check the 4-neighbours and see if any of these four neighbours has a pixel with the same value 1, in this case

If we use an 8-connectivity algorithm, S_1 and S_2 would be shown as a single connected region. Apart from 4-connectivity and 8-connectivity, we also have m-adjacent or m-connectivity (mixed connectivity). It is used to

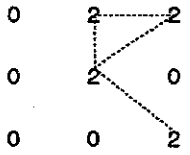
eliminate the double paths that arise when we use 8-connectivity.

Let us take an example.

Consider the image shown.

0	2	2
0	2	0
0	0	2

Here the condition used is that pixels belong to a region if their value is 2. When we use 8-connectivity here, we get



We notice that the pixel at the right hand upper corner gets connected twice due to 8-connectivity. The ambiguity is removed by using m-connectivity.

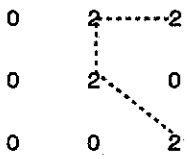
Hence m-connectivity can be defined as follows.

Two pixels p and q are said to be m-connected if

(i) q is in $N_4(p)$ or

(ii) q is in $N_D(p)$ and the set $N_4(p) \cap N_4(q)$ is empty

Using this definition we get



Any connectivity can be used as long as one is consistent. Often 8-connectivity yields results that lie closer to one's intuition.

Ex. 7.8.1

Consider two image subsets S_1 and S_2

	S_1				S_2				
0	0	0	0	0	0	0	0	2	2
2	0	0	2	0	0	2	0	0	2
2	0	0	2	0	2	2	0	0	0
0	0	2	2	2	0	0	0	0	0
0	0	2	2	2	0	0	2	2	2

For V = {2}, determine whether S_1 and S_2 are

(a) 4-connected (b) 8-connected (c) m-connected

Soln. :

V = {2} simply means that two pixels p and q are connected if their values are equal to 2.

	S_1				S_2				
0	0	0	0	0	0	0	2	2	0
2	0	0	2	0	0	2	0	0	2
2	0	0	2	0	(2) q	2	0	0	0
0	0	2	2	(2) p	0	0	0	0	0
0	0	2	2	2	0	0	2	2	2

(a) S_1 and S_2 are not 4-connected because q is not in the set of $N_4(p)$.

(b) S_1 and S_2 are eight connected because q is in the set of $N_8(p)$.

(c) S_1 and S_2 are m-connected because

(i) q is in the set $N_D(p)$ and

(ii) the set $N_4(p) \cap N_4(q)$ is empty

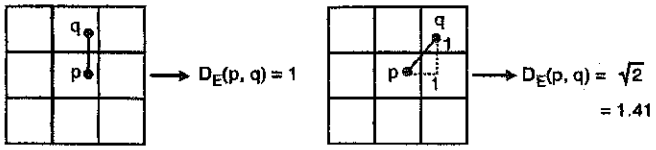
7.9 Distance Transform

Though not directly related to segmentation, Distance transform is introduced here as it is related to connectivity. Distance transforms are required in image processing projects where measurements have to be made. The distance transform provides a measure of the separation of points in an image.

(1) Euclidean Distance : Euclidean distance is the straight line distance between two pixels. If p and q are the two pixels with coordinates (x_1, y_1) and (x_2, y_2), then

$$D_E = [(x_1 - x_2)^2 + (y_1 - y_2)^2]^{1/2}$$

Diagrammatically it can be shown as



$\therefore$ for a 3×3 region, we have the Euclidian distance as

(2) City Block Distance (D_4 Distance) : For the same points $p(x_1, y_1)$ and $q(x_2, y_2)$, the city block distance is defined as

$$D_{CITY}(p, q) = D_4(p, q) = |x_1 - x_2| + |y_1 - y_2|$$

The city block distance measures the path between the pixels based on a 4-connected neighborhood. Pixels whose edges touch are 1 unit apart and pixels touching diagonally are 2 units apart.

2	1	2
1	0	1
2	1	2

(3) Chess Board Distance (D_8 Distance) : For pixels $p(x_1, y_1)$ and $q(x_2, y_2)$, the chess board distance is defined as

$$D_{CHESS}, (p, q) = D_8(p, q) = \max(|x_1 - x_2|, |y_1 - y_2|)$$

The chessboard distance measures the path between the pixels based on a 8-connected neighbourhood. Pixels whose edges or corners touch are 1 unit apart.

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

(4) D_m Distance (m-adjacency Distance) : This distance measure is based on m-adjacency. Pixels p and q are m-adjacent if

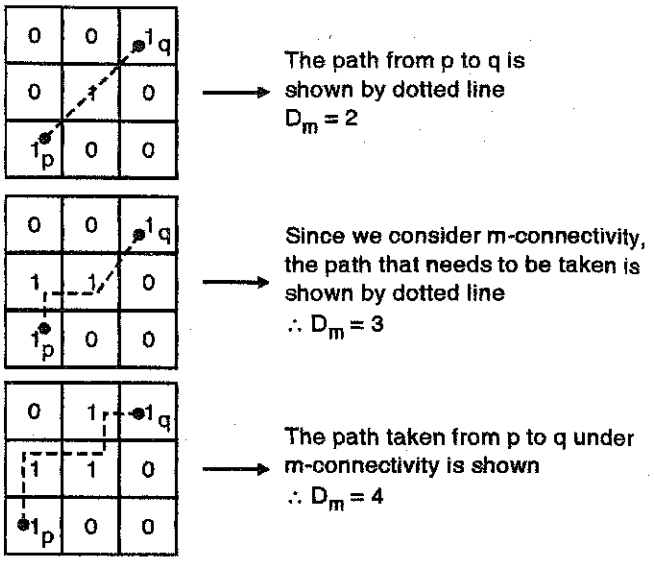
(a) q is in $N_4(p)$ or

(b) q is in $N_D(p)$ and $N_4(p) \cap N_4(q)$ is empty.

Ex. 7.9.1

Measure the D_m distance between p and q for the following images.

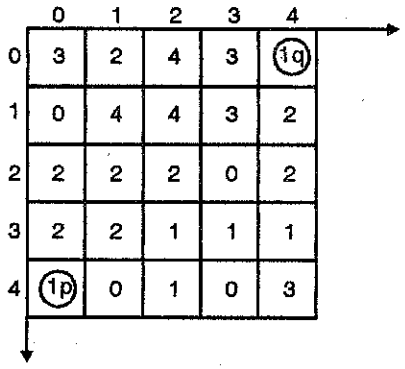
Soln. :



Ex. 7.9.2

Given below is a 5×5 image. Find out D_4, D_8.

Soln. :



$p(x_1, y_1) = p(4, 0)$

$q(x_2, y_2) = q(0, 4)$

(a) $D_4(p, q) = |x_1 - x_2| + |y_1 - y_2|$

$= |4 - 0| + |0 - 4|$

$= 4 + 4$

$D_4(p, q) = 8$

(b) $D_8(p, q) = \max(|x_1 - x_2|, |y_1 - y_2|)$

$= \max(|4 - 0|, |0 - 4|)$

$D_8(p, q) = \max(4, 4)$

$D_8(p, q) = 4$

Ex. 7.9.3

Consider the image segment shown

3	1	2	1q
2	2	0	2
1	2	1	1
1p	0	1	2

for V = {0, 1}, compute the length of the shortest m-path between p and q.

Soln. :

The path taken is shown below :

3	1	2	1
2	2	0	2
1	2	1	1
1	0	1	2

Hence the shortest m-distance is equal to 5.

Ex. 7.9.4

Let V = {0, 1}. Compute D_E, D_4, D_8 and D_m distances between two pixels p and q. Let the pixel coordinates of p and q be (3, 0) and (2, 3) respectively for the image shown. Find distance measures.

	0	1	2	3
0	0	1	1	1
1	1	0	0	1
2	1	1	1	1 q
3	1 p	1	1	1

Soln. :

V {0, 1} implies that the distance traversed can pass through 0 and 1.

(i) Euclidean distance (D_E)

In Euclidean distance, horizontal and vertical distances are equal to 1, while diagonal distances are equal to 1.41.

The path that we can traverse is shown below.

	0	1	2	3
0	0	1	1	1
1	1	0	0	1
2	1	1	1	1 q
3	1 p	1	1	1

$D_E(p, q) = 1.4 + 1 + 1$

$D_E(p, q) = 3.4$

(ii) D_4 distance

$$D_4(p, q) = |x_1 - x_2| + |y_1 - y_2|$$

Since co-ordinates of p are (3, 0) and co-ordinates of q are (2, 3).

$\therefore D_4(p, q) = |3 - 2| + |0 - 3|$

$D_4(p, q) = 4$

(iii) D_8 distance

$D_8(p, q) = \max(|x_1 - x_2|, |y_1 - y_2|)$

$= \max(|3 - 2|, |0 - 3|)$

$= \max(1, 3)$

$\therefore D_8(p, q) = 3$

(iv) D_m distance

Here we use m-adjacency neighbourhood. Pixels p and q are m-adjacent if

(a) q is in $N_4(p)$ or

(b) q is in $N_D(p)$ and $N_4(p) \cap N_4(q)$ is empty.

In the given example, we traverse the path shown below.

	0	1	2	3
0	0	1	1	1
1	1	0	0	1
2	1	1	1	1 q
3	1 p	1	1	1

$\therefore D_m(p, q) = 1 + 1 + 1 + 1$

$D_m(p, q) = 4$

7.10 Region Based Segmentation (Segmentation Based on Similarities)

When we started with the chapter on segmentation, it was stated that segmentation could be carried out in two separate ways, one based on discontinuities and the other based on similarities.

Region based segmentation is a technique in which segmentation is carried out based on the similarities in the given image. The region based approach to segmentation seeks to create regions directly by grouping together pixels which share common features into areas or regions of uniformity.

The regions that are formed using the Region based segmentation have the following properties.

Let R represent the entire image.

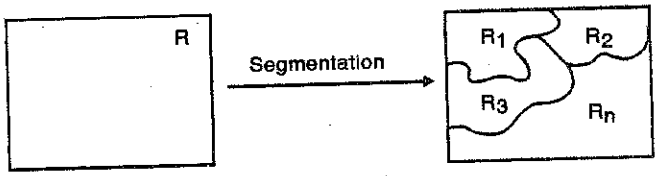


Fig. 7.10.1

We can view segmentation as a process that partitions R into sub-regions

$R_1, R_2, ...R_n$ such that

(1) $\bigcup_{i=0}^{n} R_i = R$

(2) R_i is a connected region, i = 1, 2, ...n

(3) $R_i \cap R_j = \phi$ for all i and j; $i \neq j$

(4) $P(R_i)$ = True for i = 1, 2, ...n

(5) $P(R_i \cup R_j)$ = False for $i \neq j$

Where $P(R_i)$ is the logical predicate over the points in set R and ϕ is the null set.

By predicate we mean some condition that is set.

The five conditions stated can be interpreted as follows :

(1) The union (sum) of all the regions equals the whole image. In other words all pixels in the image must be assigned to a region.

(2) Each region is contiguous and connected, (it could be either 4-connectivity or 8-connectivity or m-connectivity).

(3) The interaction of any pair of adjacent regions equals the empty set. i.e., each pixel belongs to a single region only and hence there is no overlap between regions.

(4) For each region, the uniformity predicate is true. Each region must satisfy some particular uniformity condition.

(5) For any two adjacent regions, the uniformity predicate is false. It simply means two adjacent regions do not have anything in common.

Region based segmentation can be carried out in four different ways.

(1) Region growing
(2) Region splitting
(3) Region merging
(4) Split and merge

We shall proceed to discuss each one in detail.

7.10.1 Region Growing

As the name implies, region growing is a procedure in which pixels are grouped into larger regions based on some predefined condition. The basic approach is to select a seed point (a pixel from where we begin) and grow regions from this seed pixel.

Let us pick up an arbitrary pixel (x_1, y_1) from the image that needs to be segmented. This pixel is called the seed pixel. We now examine the nearest neighbours (4 or 8 neighbours depending on whether we assume 4-connectivity or 8-connectivity) of (x_1, y_1) one by one. The neighbouring pixel is accepted in the same region as (x_1, y_1) if they together satisfy the homogeneity property of a region i.e., if both of them satisfy the predefined condition.

Once a new pixel (x_2, y_2) is accepted as a member of the current region, the neighbours of this new pixel are examined (again, 4 or 8 neighbours, depending on the connectivity assumed).

This procedure goes on recursively until no new pixel is accepted. All the pixels of the current region are given a

unique label. Now a new seed pixel is chosen and the same procedure is repeated. We continue doing this until all the pixels are assigned to some region or the other.

Let us take an example.

Ex. 7.10.1

Consider an 8 × 8 image, the grey levels range from 0 to 7. Segment this image using the region growing technique.

Soln. :

Before starting the region growing process, we need to define two things

5	6	6	6	7	7	6	6
6	7	6	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	3	2	4	7
0	0	0	0	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4

(1) What is the predicate ?

(2) Which do we take as the seed pixel ?

Both these conditions are fed in by the user. Let the predicate be

$$\max \{g(x, y)\} - \min \{g(x, y)\} < th$$

where th is the threshold.

In this case let the threshold be ≤ 3.

$$\therefore P(R_i) = \max\{g(x, y)\} - \min\{g(x, y)\} \leq 3$$

$$(x, y \in R) \qquad (x, y \in R)$$

We assume 4-connectivity. Let us start with the seed pixel 6 (encircled).

We use the above equation to get the image shown →

5	6	6	6	7	7	6	6
6	7	(6)	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	3	2	4	7
0	0	0	(0)	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4

The shaded portion gets labelled as a (say). Is segmentation complete ? The answer is NO. For segmentation, to be complete, all the pixels have to be assigned to some region. We now select 0 (encircled) as the seed pixel and label the new area as b.

We hence have →

a	a	a	a	a	a	a	a
a	a	(a)	a	a	a	a	a
a	a	a	a	b	b	a	a
a	a	a	a	b	b	a	a
b	b	b	b	b	b	a	a
b	b	b	(b)	b	b	a	a
b	b	b	b	b	b	a	a

We arrived at the above segmented image by using 6 and 0 as the seed pixels.

Note : In each of the two regions, the predicate max(g(x,y)) − min(g(x,y)) ≤ 3 in satisfied. The problem in this case is that if we change the seed pixel, the segmented image would look completely different.

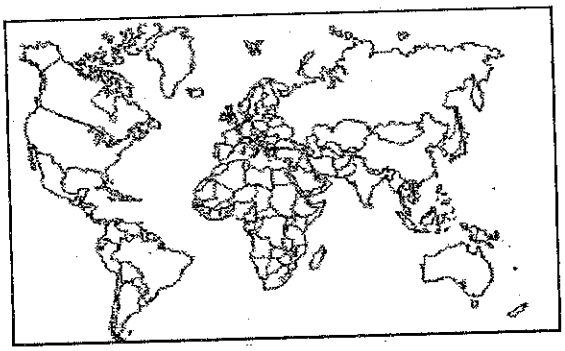
Let us take the seed pixel as 3. Let the predicate be the same i.e, th ≤ 3

Thus this image is segmented into 3 different regions by using a different seed point.

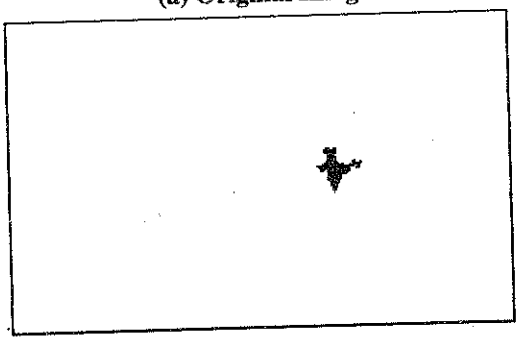
In this method, at a time only one pixel is assigned as a member of the current region because of which it is very

slow. Another problem is, as discussed, choosing a proper seed point.

5	6	6	6	7	7	6	6
6	7	6	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	(3)	2	4	7
0	0	0	0	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4

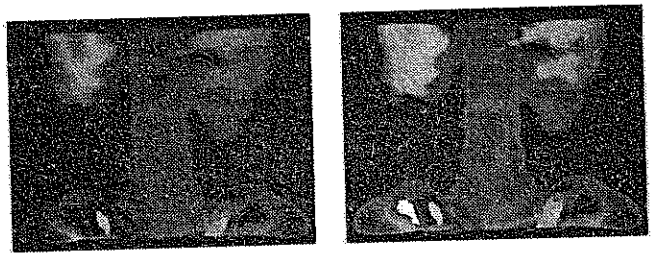


(a) Original Image



(b) Segmentation using region growing. The colours have been inverted for better visualization

Fig. 7.10.2



(a) Original image of spine **(b) Image segmentation using region growing**

Fig. 7.10.3

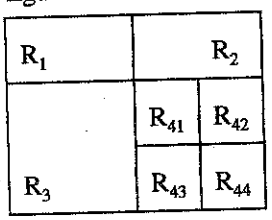
A better approach is to consider a small region at a time instead of a single pixel. This is what is done in the next method of region based segmentation.

7.10.2 Region Splitting

In this case we try to satisfy the homogeneity property where pixels that are similar are grouped together. If the grey levels present in the region do not satisfy the property, we divide the region into four equal quadrants. If the property is satisfied, we leave the region as it is. This is done recursively until all the regions satisfy the property. To explain this in terms of graph theory, we call each region a node. This node (parent node) is split into its four children (leaf nodes) if it does not satisfy the given property. If the node satisfies the property, it is left as it is. We now check each leaf node and see if these leaf nodes satisfy the given property. If yes, they are left unaltered and if no, there are further split.

This particular splitting technique has a convenient representation in the form of a quad tree. (That is, a tree in which a node has exactly four descendents).

Consider the figure shown below



In this, image R represents the entire image and hence R is the parent node. This parent node is split up into four leaf nodes, R_1, R_2, R_3 and R_4. Of these leaf nodes only R_4 does not contain pixels which satisfy some common property and hence is split again i.e., R_{41}, R_{42}, R_{43}, R_{44}. The quad tree for this division is shown.

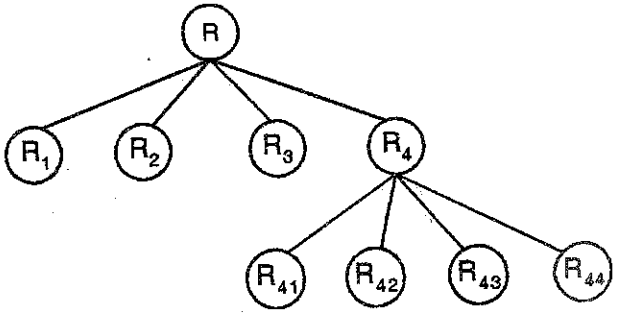


Fig. 7.10.4

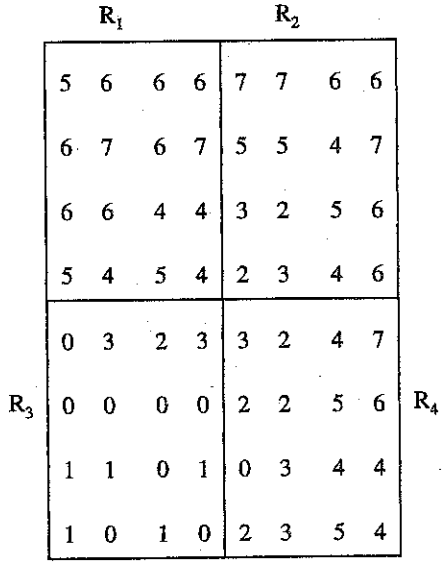
This method is applicable to images whose number of rows and number of columns are some integer power of 2.

The concepts stated here will be clear with an example.

Ex. 7.10.2

Consider the same 8 × 8 image that we worked with in region growing. Let the predicate be threshold ≤ 3. Also draw the quad tree.

5	6	6	6	7	7	6	6
6	7	6	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	3	2	4	7
0	0	0	0	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4



Soln. :

Since the entire image does not satisfy the condition of threshold ≤ 3, we split the image into four regions.

In this, R_1 and R_3 satisfy the homogeneity condition i.e., all pixels of R_1 satisfy the condition of threshold ≤ 3 in other words,

$$\max\{g(x, y)\} - \min\{g(x, y)\} \leq 3$$

$$(x, y \in R_1) \quad (x, y \in R_1)$$

$$7 - 4 \leq 3$$

Similarly, all the pixels of R_3 satisfy the same condition i.e.,

$$\max\{g(x, y)\} - \min\{g(x, y)\} \leq 3$$

$$(x, y \in R_3) \quad (x, y \in R_3)$$

$$3 - 0 \leq 3$$

But the pixels of leaf nodes R_2 and R_4 do not satisfy the homogeneity condition.

In R_2, max g(x, y) = 7 and min g(x, y) = 2

$\therefore$ max{g(x, y)} − min{g(x, y)}

$$7 - 2 \leq 3$$

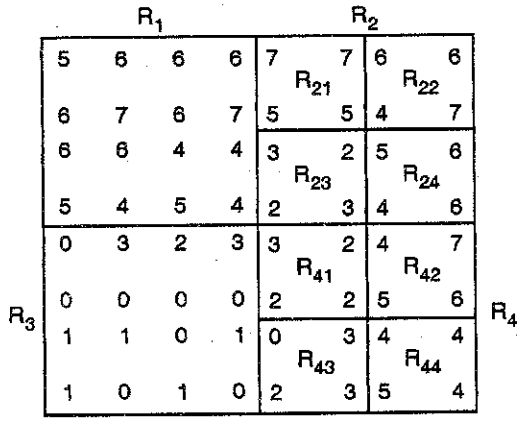
Similarly in region R_4

$$\max\{g(x, y)\} = 7, \min\{g(x, y)\} = 0$$

$$\therefore \max\{g(x, y)\} - \min\{g(x, y)\}$$

$$7 - 0 \leq 3$$

We therefore split R_2 and R_4 into 4 regions each.



We now see that each region satisfies the homogeneity condition i.e.,

$$\max\{g(x, y)\} - \min\{g(x, y)\} \leq 3$$

Hence we stop the splitting process

The quad tree is shown below.

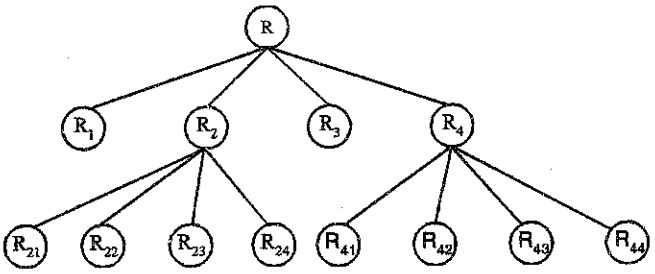


Fig. P. 7.10.2

7.10.3 Region Merging

The region merging method is exactly opposite to the region splitting method. Like the region splitting method, region merging is also applicable only to images whose number of rows and columns are an integer power of 2.

In this method, we start from the pixel level and consider each of them as a homogenous region. At any level of merging we check if the four adjacent homogenous regions arranged in a 2 × 2 fashion together satisfy the homogeneity property. If yes, they are merged to form a bigger region, otherwise the regions are left as they are.

Let us take the same 8 × 8 image as an example and perform region merging, we use the predicate

$$\max\{g(x, y)\} - \min\{g(x, y)\} \leq 3$$

$$(x, y \in R) \quad (x, y \in R)$$

We merge this image in 2 × 2 regions.

5	6	6	6	7	7	6	6
6	7	6	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	3	2	4	7
0	0	0	0	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4

Image after first step of merging

We check the four adjacent homogenous regions. Image obtained after the second step of merging is shown below :

5	6	6	6	7	7	6	6
6	7	6	7	5	5	4	7
6	6	4	4	3	2	5	6
5	4	5	4	2	3	4	6
0	3	2	3	3	2	4	7
0	0	0	0	2	2	5	6
1	1	0	1	0	3	4	4
1	0	1	0	2	3	5	4

7.10.4 Split and Merge

We have just explained region splitting and region merging. Region splitting starts with the whole image as one big region and splits this region into four quadrants. We continue splitting until all the sub-regions satisfy the predefined homogeneity property (predicate).

In region merging each pixel is taken as a small region. We merge small regions into larger regions if they satisfy the homogeneity property.

So when do we use region splitting and region merging ?

If homogeneous regions are small, the region merging technique is superior to the region growing technique. If most of the regions in an image are homogeneous, the region splitting technique is preferred to the region merging method.

In most applications, a combination of split and merge techniques are used. Together they are called the split and merge technique.

In the split and merge technique we start with a rectangular regions of size a × b pixels. We check the homogeneity property for each region. If the homogeneity property fails, we split up the region into four quadrants each of size a/2 × b/2. If the region satisfies the homogeneity

property, we merge it with the adjacent region forming a $2a \times 2b$ size region.

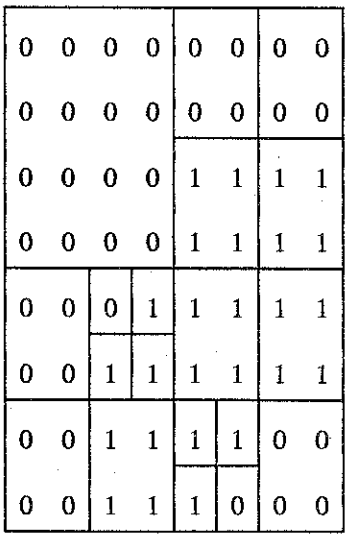
Ex. 7.10.3

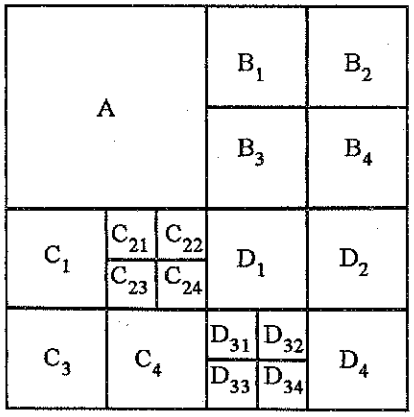
Segment the given 8×8 image using split and merge region segmentation. Here, the predicate to be used is that the value of pixels in a region has to be exactly equal.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	0	0
0	0	1	1	1	0	0	0

Soln. :

We use the split technique and split the image till each node (sub-node) has the same value of pixels. For this image, we get





On doing this, each region is uniform within itself.

A has pixels having value 0

B_1 has pixels having value 0

B_2 has pixels having value 0

B_3 has pixels having value 1

B_4 has pixels having value 1

C_1 has pixels having value 0

C_{21} has pixels having value 0

C_{22} has pixels having value 1

C_{23} has pixels having value 1

C_{24} has pixels having value 1

C_3 has pixels having value 0

C_4 has pixels having value 1

D_1 has pixels having value 1

D_2 has pixels having value 1

D_{31} has pixels having value 1

D_{32} has pixels having value 1

D_{33} has pixels having value 1

D_{34} has pixels having value 0

D_4 has pixels having value 0

We now draw the quad-tree for the image.

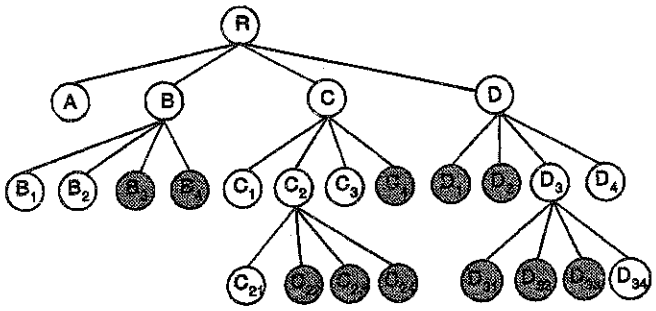


Fig. P. 7.10.3

The shaded nodes represent regions with pixel values 1.

We now merge regions that satisfy the uniformity condition. We hence get the image shown.

The split and merge algorithm can be summarized as follows :

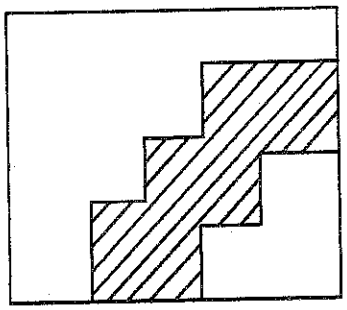


Fig. P. 7.10.3(a)

(a) Split any region into four quadrants if the uniformity predicate is not true or not satisfied.

(b) Continue this subdivision for all new sub-images until the stopping criteria is reached (usually single pixels)

(c) Merge any adjacent regions for which the uniformity condition is true.

(d) Stop when no further merging is possible.

The region based approach to segmentation seeks to create regions directly by grouping together pixels which share some common features. The regions that are formed have some uniformity condition that they satisfy. This can be a relatively straight forward task for images which have uncluttered scenes. However in most of the practical cases, images have a lot of details and hence region analysis is a complicated exercise. Region analysis can hence be too computationally expensive for typical machine vision applications. Despite this, region-based methods are of interest because they are generally less sensitive to noise than the boundary based approach.

7.11 Image Segmentation Based on Thresholding

Thresholding is one of the most important techniques for segmentation and because of its simplicity, is widely used. Thresholding has already been discussed in the chapter of image enhancement. We shall define thresholding in a more formal way here. In segmentation, thresholding is used to produce regions of uniformity within the given image based on some threshold criteria T.

Thresholding is used to produce regions of uniformity within the given image based on some threshold criteria T.

Suppose we have an image which is composed of dark objects of varying grey levels against a light background of varying grey levels as shown in Fig. 7.11.1.

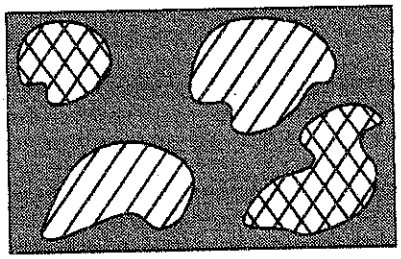


Fig. 7.11.1

The histogram of such an image would appear more or less as shown in Fig. 7.11.2.

An obvious way to extract the objects from the background is to select a threshold T that separates the two regions. Then a point (x, y) for which $f(x, y) < T$ is called an object point and for $f(x, y) > T$, it is called a background point.

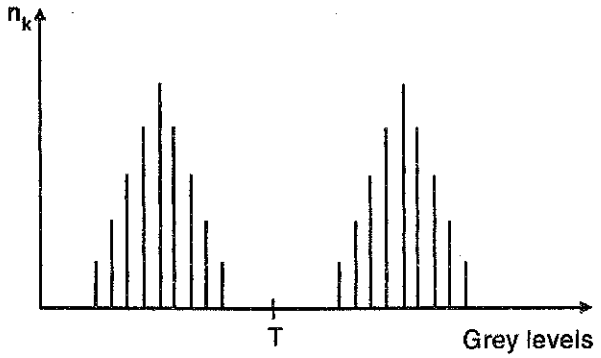


Fig. 7.11.2

Similarly we could have images whose histograms look like the one shown in Fig. 7.11.3.

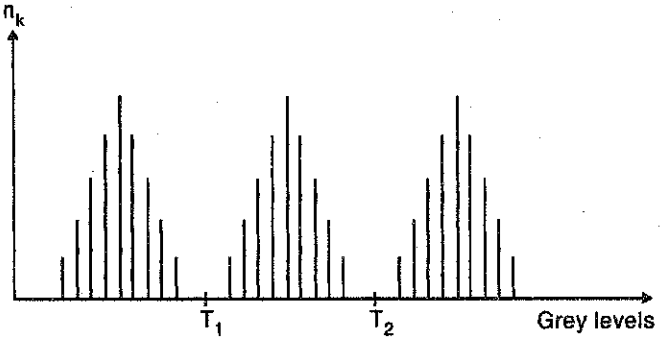


Fig. 7.11.3

i.e., an image with two different types of dark objects (large variation in their grey levels) against a bright background. In such a case, we would require two threshold values, T_1 and T_2 to separate the two objects from their background. This is known as multilevel thresholding.

The thresholding operation can thus be thought of as a test involving the function T and can be defined as

$$T = T\{x, y, A(x, y), f(x, y)\}$$

Here, f(x, y) is the grey level of the pixel at (x, y) and A(x, y) denotes some local property in the neighbourhood of this image.

Now that we are convinced that thresholding does help in separating the objects from the background, we shall proceed to explain the types of thresholding.

(1) Global thresholding

(2) Local thresholding

7.11.1 Global Thresholding

We have seen that the general thresholding function is defined as

$$T = T\{x, y, A(x, y), f(x, y)\}$$

Global thresholding is given by

$$T = T\{f(x, y)\}$$

It simply means that in global thresholding, we partition the image histogram by using a single thresholding function T. In this case A(x, y) is the entire image. Since, we have a single threshold T for the entire image, T is called the global threshold.

Segmentation is achieved by scanning the entire image pixel by pixel and labelling each pixel as an object pixel or a background pixel depending on the value of that pixel.

Global thresholding works only if the histogram of the image is partitioned property.

Steps involved in performing global thresholding.

(1) Read the given image.

(2) Plot the histogram of the given image.

(3) Based on the histogram, choose the threshold T.

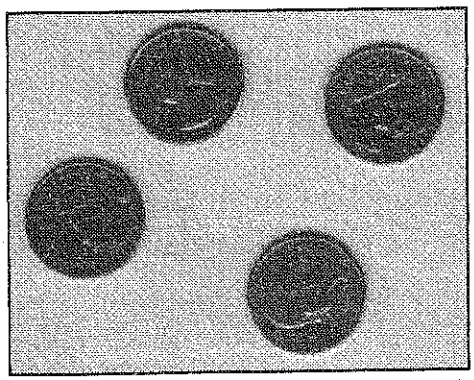
(4) Using this value of T, segment the image into objects and background.

MATLAB code for global thresholding

```
%% Global thresholding %%
%% Threshold is selected with a mouse click %%
clear all
clc
a=imread('eight.tif');
[row col]=size(a);
h=zeros(1,256);
for n=1:1:row
for m=1:1:col
```

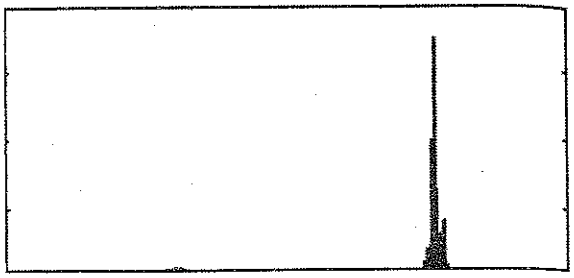
```
        t=a(n,m);       %% Takes the value of the pixel say 12
        h(t)=h(t)+1; %% Incrementing each counter h(12)
end
end
figure(1),imshow(a)
figure(2),bar(h)
[X,Y]=ginput(1);        %% select the threshold with a
                        mouse click %%
for x=1:1:col
for y=1:1:col
        if a(x,y)<X
          a(x,y)=0
        else
          a(x,y)=255;
        end
end
end
figure(3),imshow(uint8(a))
% Normalisation might be required
```

As stated earlier, this kind of thresholding can be expected to succeed, only if the histogram is well partitioned i.e., there is a marked difference in the grey level of the objects and the background.

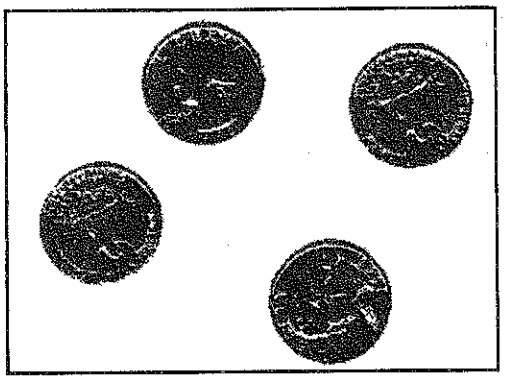


(a) Original image

Fig. 7.11.4 Continued...



(b) Histogram of original image



(c) Image obtained after threshold

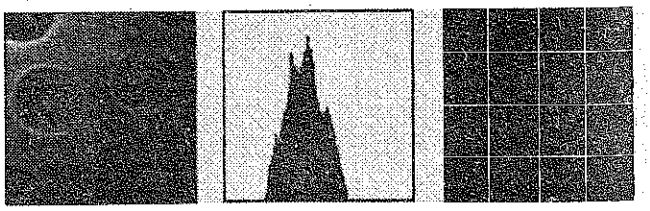
Fig. 7.11.4

7.11.2 Local Thresholding

Consider the image shown in Fig. 7.11.5(a) along with its histogram. There is no way we can segment this image based on global thresholding and separate the object from its background. In such cases, we work with local thresholds.

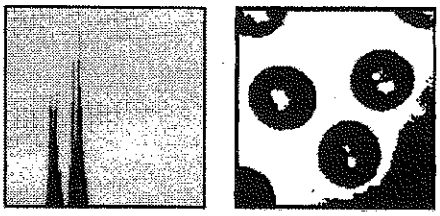
$$T = T\{A(x, y), f(x, y)\} \quad \ldots(7.11.1)$$

In this, T is dependent upon a neighbourhood property of the pixel as well as its grey level value. It simply means, we divide the entire image into subimages of size A and then use a separate threshold to segment each subimage A.



(a) (b) (c)

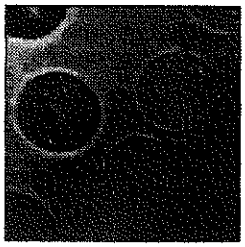
Fig. 7.11.5 Continued...



(d) (e)

Fig. 7.11.5

As can be seen, the object (coins) is obscured due to the non uniform background. We divide the image into small squares and select threshold values based on each square.



(a) Original image with non-uniform illumination

Splitting into four parts

Fig. 7.11.6

7.12 Additional Solved Examples

Ex. 7.12.1

Develop an algorithm for converting a one-pixel thick 8-path to a 4-path.

Soln. :

This can be done by defining all possible neighbourhood shapes to go from a diagonal segment to a corresponding 4-connected segment, as shown in Fig. P. 7.12.1.

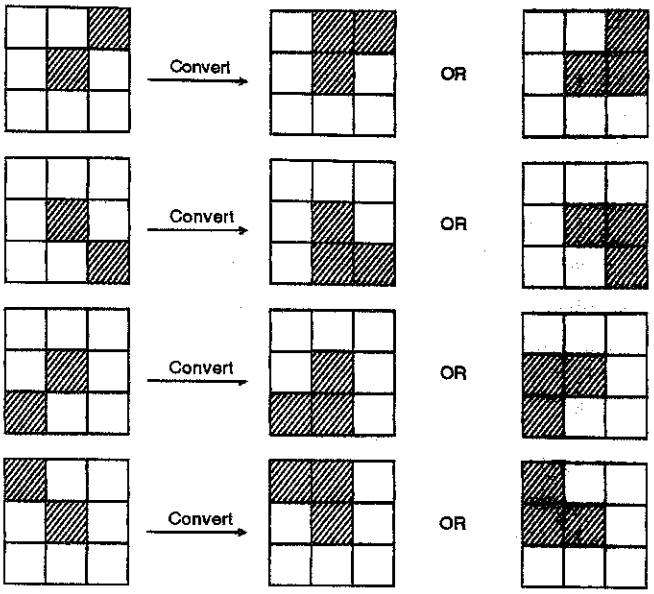


Fig. P. 7.12.1

The algorithm then simply looks for the appropriate match every time a diagonal segment is encountered.

Ex. 7.12.2

A binary image contains straight lines oriented horizontally, vertically, at 45° and at – 45°. Give a set of 3 × 3 masks that can be used to detect 1 pixel long breaks in these lines. Assume that the grey level of the lines is 1 and that the grey level of the background is 0.

Soln. :

The 4 masks that we could use are

0	0	0	0	1	0	0	0	1	1	0	0
1	–2	1	0	–2	0	0	–2	0	0	–2	0
0	0	0	0	1	0	1	0	0	0	0	1
Horizontal			Vertical			+45°			–45°		

Note that the sum of the coefficients along the required direction is zero.

Hence each mask would yield a value of 0 when centered on a pixel of an unbroken 3-pixel segment oriented in the direction of the required mask. The response of the mask would be +2 when the mask is centered on a one pixel gap in a 3-pixel segment oriented in the direction of the required mask.

Ex. 7.12.3

Show that the Laplacian mask is a Isotropic filter. (Hint : use a 3 × 3 mask)

Soln. :

By Isotropic filters we mean, filters that are rotation invariant. It means that rotating the image and then applying the filter gives the same result as applying the filter to the image first and then rotating the result.

A 3 × 3 Laplacian mask is given below

0	−1	0
−1	4	−1
0	−1	0

Consider a 3 × 3 image

1	2	1
2	4	2
4	8	4

Case 1 : Image ⟶ Convolve with Laplacian mask ⟶ Rotate

Applying the mask we get,

0	2	0
−1	2	−1
6	20	6

Rotate ⟶

6	20	6
−1	2	−1
0	2	0

Case 2 : Image ⟶ Rotate ⟶ Convolve with Laplacian mask

Rotating image we get,

4	8	4
2	4	2
1	2	1

Apply mask ⟶

6	20	6
−1	2	−1
0	2	0

We see that (1) = (2)

Hence we conclude that the Laplacian mask is an Isotropic filter (Invariant to rotation).

Note : Convolution was performed after zero padding the image.

Ex. 7.12.4

Consider the image shown below. Calculate the direction of the strongest edge.

f =

50	60	70
5	50	80
7	9	50

Soln. : We assume that the edges start from the top row and terminate at the last row. All possible edge directions are shown below.

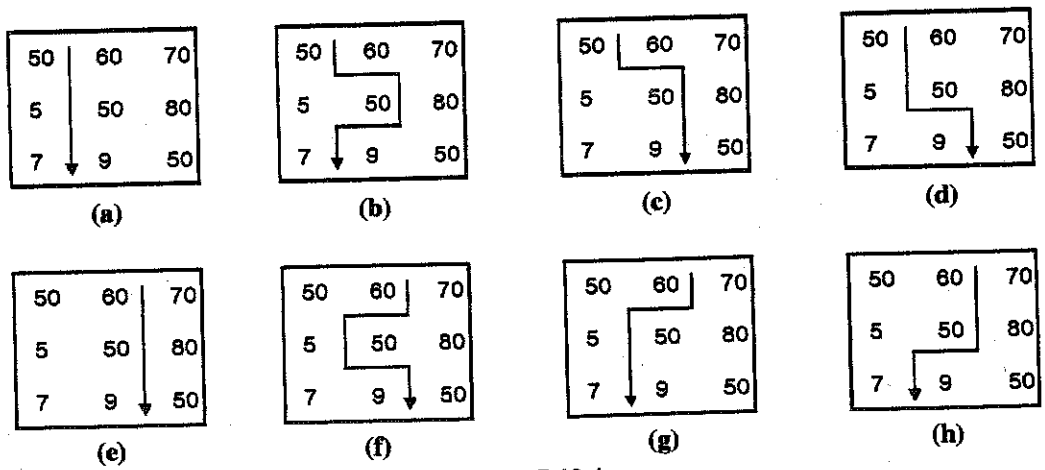


Fig. P. 7.12.4

In the earlier example we used the formula

$$C(p, q) = \text{Max}(I) - [f(p) - f(q)]$$

A easier method is to use the modulus. We shall solve the above example using the formula.

$$C(p, q) = \text{Max}(I) - |f(p) - f(q)|$$

Since we take $|f(p) - f(q)|$ we don't have to worry about the direction.

Path (a)

For edge between 50 and 60, f(p) = 50, f(q) = 60. (We could also take f(p) = 60 and f(q) = 50). Max (I) = 80

Cost across 50 and 60 is

$\therefore$ Cost $= 80 - |50 - 60| = 70$

Cost across 5 and 50 is

Cost $= 80 - |5 - 50| = 35$

Cost across 7 and 9 is

Cost $= 80 - |7 - 9| = 78$

$\therefore$ Cost of edge in (a) is

Cost $= 70 + 35 + 78 = 183$

We similarly calculate the cost of each of the edges.

Cost of edge in (b) is

$$\text{Cost} = \{80 - |50 - 60|\} + \{80 - |60 - 50|\} + \{80 - |50 - 80|\} + \{80 - |50 - 9|\} + \{80 - |7 - 9|\}$$

$$= 70 + 70 + 50 + 39 + 78 = \mathbf{307}$$

Cost of edge in (c) is

$$\text{Cost} = \{80 - |50 - 60|\} + \{80 - |60 - 50|\} + \{80 - |50 - 80|\} + \{80 - |9 - 50|\}$$

$$= 70 + 70 + 50 + 39 = \mathbf{229}$$

Cost of edge in (d) is

$$\text{Cost} = \{80 - |50 - 60|\} + \{80 - |5 - 50|\} + \{80 - |50 - 9|\} + \{80 - |9 - 50|\}$$

$$\text{Cost} = 70 + 35 + 39 + 39 = \mathbf{183}$$

Cost of edge in (e) is

$$\text{Cost} = \{80 - |60 - 70|\} + \{80 - |50 - 80|\} + \{80 - |9 - 50|\}$$

$$= 70 + 50 + 39 = \mathbf{159}$$

Cost of edge in (f) is

$$\text{Cost} = \{80 - |60 - 70|\} + \{80 - |60 - 50|\} + \{80 - |5 - 50|\} + \{80 - |50 - 9|\} + \{80 - |9 - 50|\}$$

$$\text{Cost} = 70 + 70 + 35 + 39 + 39 = \mathbf{253}$$

Cost of edge in (g) is

$$\text{Cost} = \{80 - |60 - 70|\} + \{80 - |60 - 50|\} + \{80 - |5 - 50|\} + \{80 - |7 - 9|\}$$

$$= 70 + 70 + 35 + 78 = \mathbf{253}$$

Cost of edge in (h) is

$$\text{Cost} = \{80 - |60 - 70|\} + \{80 - |50 - 80|\} + \{80 - |50 - 9|\} + \{80 - |7 - 9|\}$$

$$= 70 + 50 + 39 + 78 = \mathbf{237}$$

$\therefore$ Minimum cost is of path a = **183**, d = 183.

Hence the edge in Fig. P. 7.12.4(a) and Fig. P. 7.12.4(d) have the strongest edge.

Ex. 7.12.5

Assuming that edge starts in the first row ant ends in the last row. For the following gray level image, Sketch all possible paths and determine edge.

7	2	2
5	7	2
5	1	0

Soln. :

We shall use the formula

$$C(p, q) = \text{Max}(I) - |f(p) - f(q)|$$

All possible edge directions are shown in Fig. P. 7.12.5.

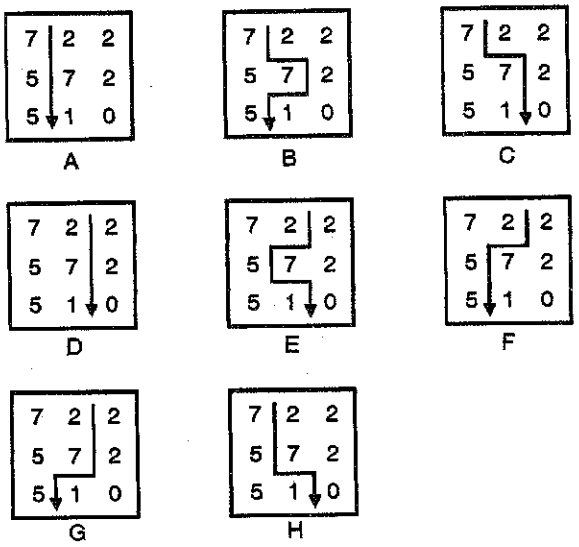


Fig. P. 7.12.5

For A, C (p,q) = (7 – 7 – 2) + (7 – 5 – 7) + (7 – 5 – 1)

= 2 + 5 + 3 = 10

∴ Cost A = 10

For B, C (p,q) = (7 – 7 – 2) + (7 – 2 – 7) + (7 – 7 – 2) + (7 – 7 – 1) + (7 – 5 – 1)

= 2 + 2 + 2 + 1 + 3 = 10

For C, C(p,q) = (7 – 7 – 2) + (7 – 7 – 2) + (7 – 7 – 2) + (7 – 1 – 0)

= 2 + 2 + 2 + 6 = 12

For D, C(p,q) = (7 – 2 – 2) + (7 – 7 – 2) + (7 – 1 – 0)

= 7 + 2 + 6 = 15

For E, C(p,q) = (7 – 2 – 2) + (7 – 2 – 7) + (7 – 7 – 5) + (7 – 7 – 1) + (7 – 1 – 0)

= 7 + 2 + 5 + 1 + 6 = 21

For F, C (p,q) = (7 – 2 – 2) + (7 – 2 – 7) + (7 – 7 – 5) + (7 – 1 – 5)

= 7 + 2 + 5 + 3 = 17

For G, C (p,q) = (7 – 2 – 2) + (7 – 2 – 7) + (7 – 7 – 1) + (7 – 5 – 1)

= 7 + 2 + 1 + 3 = 13

For H, C (p,q) = (7 – 7 – 2) + (7 – 5 – 7) + (7 – 7 – 1) + (7 – 1 – 0)

= 2 + 5 + 1 + 6 = 14

The least cost implies the strongest edge. Paths A and B have the lowest cost of 10. Hence A and B have the strongest edge.

Ex. 7.12.6

For the following given binary image R of size (256 × 256), apply split and merge technique and segment the image so that all the pixels in segmented image will have same intensity value.

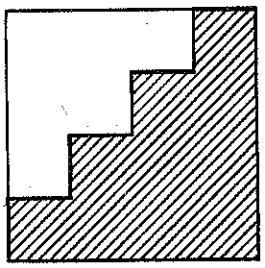


Fig. P.7.12.6

Soln. : Since the image is of size 256 × 256, we make a simple assumption and draw the image accordingly.

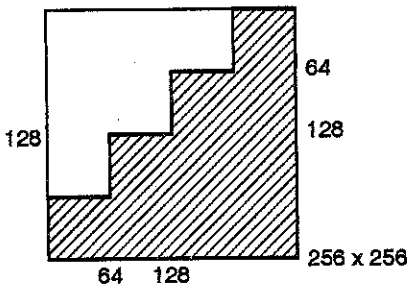
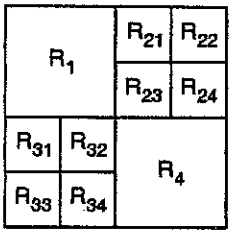


Fig. P. 7.12.6(a)

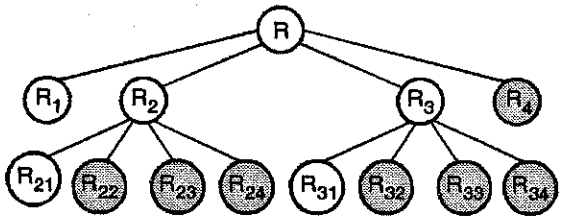
We start by splitting the image into 4 equal quadrants.

R_1	R_2
R_3	R_4

We notice that R_1 and R_4 have constant grey levels in them. We now split R_2 and R_3.



Now each of the regions formed have constant grey levels. We draw a tree diagram for the splitting technique.



Splitting gives us 10 regions.

Once we have performed splitting, we merge the regions that have the black shade as shown.

We label the shaded regions as 1 and the non shaded region as 0.

Summary

Segmentation is an essential step in almost all image analysis systems. It is imperative to understand the applications of image segmentation. Segmentation partitions an image into meaningful regions having certain characteristics unique to that region. In this, the output is an abstract representation of the input image. There exists no general segmentation algorithm which will work satisfactorily for all images. Users have to choose a particular segmentation algorithm which would be suitable for the problem in hand. Segmentation based on dissimilarities as well as segmentation based on similarities have been discussed in detail. MATLAB codes have been provided for gradient operators such as Prewitts and Sobel. Special emphasis has been given to Hough transforms which forms an important topic of edge linking. In segmentation based on similarities, topics such as region growing and region merging have been discussed.

Review Questions

Q. 1 Explain the term Image Segmentation.

Q. 2 Explain, segmentation based on discontinuities and segmentation based on similarities.

Q. 3 Given a 8 × 8 image, use a mask to detect points and lines oriented at +45° and –45°

0	0	0	0	0	0	0	0
0	255	0	255	0	0	255	0
0	0	255	0	0	0	255	0
0	255	0	0	255	0	0	0
255	0	0	0	0	255	0	0
0	0	255	0	0	0	255	0
0	0	0	0	0	0	0	255
0	0	0	0	0	0	0	0

Q. 4 Explain edge detection in detail.

Q. 5 Explain the process of image segmentation using region growing.

Q. 6 For a given image perform segmentation by region growing. Choose appropriate seeds and threshold.

7	7	7	5	2	2	7	6
6	6	6	5	2	2	7	8
5	5	5	5	0	2	6	8
5	5	0	0	1	1	8	7
5	0	0	2	1	2	6	8
0	0	0	8	8	6	7	7
0	0	8	6	6	6	8	8
8	8	5	6	4	7	8	7

Q. 7 Show that the Laplacian operator is invariant to rotation.

Q. 8 Explain the need of a LOG operator.

Q. 9 Explain a global segmentation technique which links points by determinining whether they lie on a line.

Q. 10 Explain the following edge extraction operators

(a) Sobel (b) Prewitt

(c) Roberts (d) Laplacian

(e) Fri-chen

Use these masks on an image of your choice.

Q. 11 An image is composed of one blob of mean grey level m_1=150 and variance σ_1^2 = 400 scattered on a background of mean m_2 = 255 and variance σ_2^2 = 255. The blob occupies approximately 20% of the image area. Propose a technique based on thresholding, for segmenting the blob out of the image.

Q. 12 A binary image contains straight lines oriented horizontally, vertically, at + 45° and at – 45°. Give a set of 3 × 3 masks that can be used to detect 1-pixel long breaks in these lines. Assume that the grey level of the lines is 1 and that of the background is 0.

Q. 13 An image segment is shown -

Operate it with a transform function such that

$g(x,y) = 0$ for $0 \le f(x,y) \le 3$

$= f(x,y)$ for $4 \le f(x,y) \le 12$

$= 15$ for $f(x,y) > 12$

0	0	0	1	1	3	3	3
1	1	0	1	2	3	10	8
2	0	4	5	10	14	14	7
3	1	5	6	15	14	13	6
4	5	6	10	14	13	13	5
8	9	10	12	11	10	11	11
6	7	10	1	5	7	10	11
0	1	3	0	0	1	6	6

Q. 14 Explain the technique of thresholding for segmentation.

Q. 15 Describe how Hough transform is used for boundary shape detection.

Q. 16 Given an image as shown, use Hough transform to detect the given line

0	0	0	1	1	3	3	3
1	1	0	1	2	3	10	8
2	0	4	5	10	14	14	7
3	1	5	6	15	14	13	6
4	5	6	10	14	13	13	5
8	9	10	12	11	10	11	11
6	7	10	1	5	7	10	11
0	1	3	0	0	1	6	6

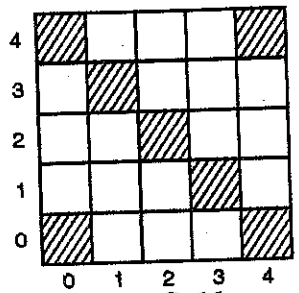


Fig. Q. 16

Q. 17 Explain the concept of compass operators.

Q. 18 Explain local, global and optimum thresholding.

Q. 19 Explain whether poorly illuminated image can be easily segmented.

Q. 20 Explain the process of image segmentation using

(a) Region Growing (b) Region Splitting

(c) Region Merging (d) Split and Merge

Q. 21 Write in detail on connectivity of pixels.

Q. 22 What is the distance transform ?

Q. 23 Explain :

(a) Euclidean distance

(b) City block distance

(c) Chess board distance

(d) D_m distance.

Q. 24 List down the properties that a region has to satisfy.

Chapter Ends...

□□□

→ An image transform can be applied to an image to convert it from one domain to another.
→ Viewing an image in frequency or Hough space domain enables the identification of the features that may not be as easily detectable in spatial domain.

CHAPTER 8

Image Transforms

8.1 Introduction

We have studied the Discrete Fourier Transform (DFT) in chapter of Image Enhancement in Frequency Domain. We have also seen specific applications where the DFT is used.

Apart from the DFT, there are a number of discrete linear transformations that prove useful in digital image processing. In this chapter, we shall develop and examine several other transforms which regularly feature in image processing literature.

The term image transform usually refers to a class of unitary matrices used for representing images. Just as a one-dimensional signal can be represented by an orthogonal series of basis functions, an image can also be expanded in terms of a discrete set of basis arrays called basis images.

8.2 Linear Transformations

8.2.1 One Dimensional Discrete Linear Transformations

If x is a N × 1 vector and T is a N × N matrix, then

$$y_i = \sum t_{i,j} \cdot x_j \qquad i = 0, 1, ..., N-1$$

$$y = Tx \qquad ...(8.2.1)$$

defines a linear transformation of the vector x. The matrix T is called the kernel matrix of the transformation and the resultant y is another N × 1 vector.

The transformation y = Tx is said to be linear because the output y is formed by a first order summation of input elements. Each element of y (y_i) is the inner product of the input vector x with the i^{th} row of T.

8.2.2 Unitary and Orthogonal Matrices

We have seen that y = Tx represents a linear transformation. Now what should be the values of T ? For a given vector length N, there are infinitely many transformation matrices T that could be used. Commonly used T are the ones that belong to a class of matrices having certain properties.

If T is a unitary matrix, then

$$T^{*\prime} = T^{-1}$$

Multiplying both sides by T, we get,

$$T \cdot T^{*\prime} = T \cdot T^{-1} \qquad ...(8.2.2)$$

$$\therefore \mathbf{T \cdot T^{*\prime} = I} \qquad ...(8.2.3)$$

Where * indicates complex conjugate of T.

Hence to check whether T is a unitary matrix we need to use the relationship $T\,T^{*\prime} = I$.

If T is unitary and has only real elements, then it is an orthogonal matrix and it follows that

$$T^{\prime} = T^{-1}$$

$$\therefore T \cdot T^{\prime} = I \qquad ...(8.2.4)$$

In other words, an orthogonal matrix is a square matrix whose transpose is its inverse.

Here the rows of T are a set of orthonormal vectors.

To cut a long story short, T is unitary if $T\,T^{*\prime} = I$ and T is orthogonal if $T \cdot T^{\prime} = I$. Always remember this.

Ex. 8.2.1

Check whether the one-dimensional DFT is an example of a unitary transform.

Soln. :

We know that the Fourier transform can be represented as F = Wf where f is the input and W is the DFT matrix. Taking N = 4, we form a DFT matrix.

(This was covered in chapter of Image Enhancement in Frequency Domain.)

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

To check whether W is unitary or not, we need to check the relationship.

$$W \cdot W^{*\prime} = I$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

$$W \qquad W^{*\prime} \qquad = \quad 4\,[I]$$

Hence the DFT is an example of a unitary transform.

4 indicates that the DFT matrix is not normalised. We compensate for this by using $\frac{1}{N}$ in the forward or inverse DFT formula.

Hence the DFT is an example of a unitary transform.

If f is an input sequence in column form, then the 1-D DFT can be computed using the formula

$$F = Wf \qquad \ldots(8.2.5)$$

8.2.3 Two Dimensional Discrete Linear Transformations

In two dimensions, a linear transformation which transforms an input image f (N × N) into the transformed matrix F (which is also N × N) is given by the equation,

$$F_{m,n} = \sum_{i=0}^{N-1} \sum_{k=0}^{N-1} f_{i,k}\ \Omega(i, k, m, n)$$

Where i, k, m and n are the discrete variables that range from 0 to N – 1. Ω (i, k, m, n) is the kernel function of the transformation.

Ω (i, k, m, n) can be thought of as an $N^2 \times N^2$ block matrix having N rows of N blocks, each of which is a N × N matrix. The blocks are indexed by m, n and the elements of each N × N submatrix by i, k.

$$\begin{array}{c} \\ m=1 \\ m=2 \\ \vdots \\ m=N \end{array} \begin{array}{c} \begin{array}{cccc} n=1 & n=2 & & n=N \end{array} \\ \begin{bmatrix} [\ ]_{i,k} & [\ ]_{i,k} & \cdots & [\ ]_{i,k} \\ [\ ]_{i,k} & [\ ]_{i,k} & \cdots & [\ ]_{i,k} \\ \vdots & \vdots & \cdots & \vdots \\ [\ ]_{i,k} & [\ ]_{i,k} & & [\ ]_{i,k} \end{bmatrix} \end{array}$$

The transformation Ω (i, k, m, n) is called separable if it can be separate into the product of row-wise and column-wise component functions i.e., if

$$\Omega(i, k, m, n) = T_{row}(i, m)\, T_{col}(k, n)$$

This simply means that the transformation can be carried out in two steps – A row wise operation followed by a column wise operation (could be the other way too).

$$\therefore F_{m,n} = \sum_{i=0}^{N-1} \left[\sum_{k=0}^{N-1} f_{i,k}\, T_{row}(i, m) \right] T_{col}(k, n)$$

Further, if the two components i.e., T_{col} and T_{row} are identical, the transform is called symmetric,

$$\Omega(i, k, m, n) = T(i, m)\, T(k, n)$$

$$\therefore F_{m,n} = \sum_{i=0}^{N-1} T(i, m) \left[\sum_{k=0}^{N-1} f_{i,k}\, T_{row}(k, n) \right]$$

$$\text{i.e., } \mathbf{F = T f T} \qquad \ldots(8.2.6)$$

The DFT matrix is both symmetric as well as separable and hence can be obtained using the above relationship.

This is the standard formula for computing the transform of a 2D signal. Here T is transformation matrix which is both symmetric and separable. If the transformation matrix is not symmetric, the equation gets modified to

$$\mathbf{F = T f T'} \qquad \ldots(8.2.7)$$

Ex. 8.2.2

Find the DFT of the given image.

$$\begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 2 \end{bmatrix}$$

Soln. :

This sum was solved in chapter of Image Enhancement in Frequency Domain (kindly go through the solution).

An easier way to solve this sum is to use the relation just discussed.

$$F = T f T$$

Since f is a 4×4 matrix, T which is the DFT matrix is given below.

$$T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 8 & 12 & 8 \\ -2 & -2 & -2 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & -2 & -2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

$$F = \begin{bmatrix} 32 & -8 & 0 & -8 \\ -8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -8 & 0 & 0 & 0 \end{bmatrix}$$

This is the same as what we had got using the conventional method.

MATLAB code for calculating the DFT.

```
%% Calculating 2D DFT using matrix method - On pseudo-image %%
clear all
clc
a =[1 2 3 4; 2 3 4 5; 2 2 2 2; 1 1 1 1];
[row col]=size(a);
const=sqrt(row*col);
for n=0:1:row-1
    for k=0:1:col-1
        W(n+1,k+1)=exp(-i*2*pi*n*k/const);
    end
end
X=W*a*W.';
ff=fft2(a) %% For comparing %%
%% Calculating 2D DFT using matrix method-real square image %%
clear all
clc
a=imread ('testpatl.tif');
a=double (a);
[row col]=size (a);
const=sqrt(row*col);
for n=0 : 1 : row-1
    for k=0:1:col-
        W(n+1,k+1)=exp(-i*2*pi*n*k/const);
    end
end
X=W*a*W.';
ff=fft2(a) %% For comparing %%
figure(1)
colormap(gray)
imagesc(fftshift(log(1+abs(X)))
                %fft shift can be implemented
figure(2)       %through program%
```

```
colormap(gray)
imagesc(fftshift(log(1+abs(f))))
```

Ex. 8.2.3

Obtain 2-D DFT of following 3×3 image.

$$f(x, y) = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Soln. :

We compute the 2-D DFT of the image using the formula

$$F = T \cdot f \cdot T$$

Here T is the DFT transformation matrix and it is symmetric. Since the size of the image is 3×3, we use a 3×3 transformation matrix.

$$T = \begin{array}{c} k \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} n \rightarrow 0 \\ \begin{bmatrix} W_3^0 & W_3^0 & W_3^0 \\ W_3^0 & W_3^1 & W_3^2 \\ W_3^0 & W_3^2 & W_3^4 \end{bmatrix} \end{array} \begin{array}{c} 1 \\ \\ \\ \end{array} \begin{array}{c} 2 \\ \\ \\ \end{array}$$

Here $W_3 = e^{\frac{-j2\pi}{3}}$

$$\therefore W_3^0 = e^{\frac{-j2\pi 0}{3}} = 1$$

$$W_3^1 = e^{\frac{-j2\pi \cdot 1}{3}} = -0.5 - 0.866j$$

$$W_3^2 = e^{\frac{-j2\pi \cdot 2}{3}} = -0.5 + 0.866j$$

$$W_3^4 = e^{\frac{-j2\pi \cdot 4}{3}} = -0.5 - 0.866j$$

$$\therefore T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -0.5 - 0.866j & -0.5 + 0.866j \\ 1 & -0.5 + 0.866j & -0.5 - 0.866j \end{bmatrix}$$

$$\therefore F = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -0.5 - 0.866j & -0.5 + 0.866j \\ 1 & -0.5 + 0.866j & -0.5 - 0.866j \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -0.5 - 0.866j & -0.5 + 0.866j \\ 1 & -0.5 + 0.866j & -0.5 - 0.866j \end{bmatrix}$$

$$\therefore F = \begin{bmatrix} 3 & 6 & 9 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -0.5 - 0.866j & -0.5 + 0.866j \\ 1 & -0.5 + 0.866j & -0.5 - 0.866j \end{bmatrix}$$

$$\text{Finally } F = \begin{bmatrix} 18 & -4.5 + 2.598j & -4.5 - 2.598j \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Ex. 8.2.4

Compute 2-D DFT of 4 x 4 grey scale image given below and then compute inverse 2-D DFT of transform coefficient.

$$f(x, y) = \begin{array}{|c|c|c|c|} \hline 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 \\ \hline \end{array}$$

Soln. :

Part A

We use the formula

$$F = T \cdot f \cdot T'$$

Where T is a 4×4 DFT matrix.

Since the DFT matrix is symmetric, we can write

$$F = T \cdot f \cdot T$$

$$\therefore F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

$$\therefore F = \begin{bmatrix} 16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Part B

We now compute the 2-D-IDFT of the obtained result.

$$f = \frac{1}{N}[T \cdot F \cdot T']$$

As T is symmetric, we have $f = \frac{1}{N}[T \cdot F \cdot T]$

Since the size of the image is 4×4, $N = 16$

$$f = \frac{1}{16}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -j & -1 & j\\ 1 & -1 & 1 & -1\\ 1 & j & -1 & -j\end{bmatrix}\begin{bmatrix}16 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -j & -1 & j\\ 1 & -1 & 1 & -1\\ 1 & j & -1 & -j\end{bmatrix}$$

$$\therefore f = \begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\end{bmatrix}$$

8.3 Discrete Cosine Transform (DCT)

The Cosine Transform published by N. Ahmed, T. Natrajan and K. R. Rao has found applications in image compression. All JPEG images use the Discrete Cosine Transform as the initial stage for compression.

Just as the Fourier transform uses sines and cosines waves to represent a signal, the DCT uses only cosine waves. Hence the DCT is purely real unlike the DFT which is complex (has magnitude and phase).

The one dimensional DCT of a sequence f(x), $0 \le x \le N-1$ is defined as,

$$F(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{\pi (2x+1) u}{2N}\right]; \quad 0 \le u \le N-1$$

Where,

$$\alpha(0) = \sqrt{\frac{1}{N}}; \quad \alpha(u) = \sqrt{\frac{2}{N}} \quad \text{for } 1 \le u \le N-1$$

The inverse transformation is given by,

$$f(x) = \sum_{u=0}^{N-1} \alpha(u) F(u) \cos\left[\frac{\pi (2x+1) u}{2N}\right]; \quad 0 \le x \le N-1$$

The corresponding 2-D DCT pair is,

$$F(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x, y) \cos\left[\frac{(2x+1) u \pi}{2N}\right] \cos\left[\frac{(2y+1) v \pi}{2N}\right]$$

for u, v = 0, 1, 2, ...N – 1 and

$$f(x, y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v) F(u, v) \cos\left[\frac{(2x+1) u \pi}{2N}\right] \cos\left[\frac{(2y+1) v \pi}{2N}\right]$$

for x, y = 0, 1, 2, ...N – 1.

The N × N Cosine Transform Matrix C = {C (u, v)} is defined as,

$$C(u, v) = \begin{cases} \sqrt{\frac{1}{N}} & u = 0, 0 \le v \le N-1 \\ \sqrt{\frac{2}{N}} \cos\left[\frac{\pi (2v+1) u}{2N}\right] & 1 \le u \le N-1, 0 \le v \le N-1 \end{cases} \quad ...(8.3.1)$$

It is this equation that we shall use to compute the DCT of a given sequence.

The cosine transform matrix is real and orthogonal but not symmetric.

$\therefore C = C^* \Rightarrow C^{-1} = C'$

$\therefore C \cdot C' = I$ ... from Equation (8.2.3)

The DCT of a column sequence f(x); $0 \le x \le N-1$ can be computed

$$F = C \cdot f \quad ...(8.3.2)$$

Similarly, since the discrete cosine transform is not symmetric, the 2-dimensional DCT of an image can be generated using Equation (8.2.7).

$$\therefore F = C f C' \quad ...(8.3.3)$$

If all this sounds confusing, a few examples would make you feel at ease.

Ex. 8.3.1

Find the DCT of the following sequence f(x) = {1, 2, 4, 7}

Soln. :

We shall find the DCT using the matrix notation.

$$F = C \cdot f$$

$$C(u, v) = \begin{cases} \sqrt{\frac{1}{N}} & u = 0, 0 \le v \le N-1 \\ \sqrt{\frac{2}{N}} \cos\left[\frac{\pi (2v+1) u}{2N}\right] & 1 \le u \le N-1, 0 \le v \le N-1 \end{cases}$$

We shall first compute C

Since N = 4, C will be N × N i.e.; 4 × 4

u \ v				
	0.5	0.5	0.5	0.5
	0.653	0.2705	− 0.2705	− 0.653
	0.5	− 0.5	− 0.5	0.5
	0.2705	− 0.653	0.653	− 0.2705

Note : C is not symmetric.

$$F = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.2705 & -0.2705 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.2705 & -0.653 & 0.653 & -0.2705 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 4 \\ 7 \end{matrix}$$

$$F = \begin{bmatrix} 7 \\ -4.459 \\ 1 \\ -0.3170 \end{bmatrix}$$

Hence the DCT of the given sequence is $\{7, -4.459, 1, -0.3170\}'$.

Ex. 8.3.2

Find the DCT of the given 4 × 4 image.

2	4	4	2
4	6	8	3
2	8	10	4
3	8	6	2

Soln. :

We use the matrix notation.

$$F = CfC'$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.2705 & -0.2705 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.2705 & -0.653 & 0.653 & -0.2705 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 & 2 \\ 4 & 6 & 8 & 3 \\ 2 & 8 & 10 & 4 \\ 3 & 8 & 6 & 2 \end{bmatrix} \times \begin{bmatrix} 0.5 & 0.653 & 0.5 & 0.2705 \\ 0.5 & 0.2705 & -0.5 & -0.653 \\ 0.5 & -0.2705 & -0.5 & 0.653 \\ 0.5 & -0.653 & 0.5 & -0.2705 \end{bmatrix}$$

$$F = \begin{bmatrix} 19 & -0.2705 & -8 & 0.653 \\ -2.6913 & -0.2498 & 2.3087 & 0.8957 \\ -3.50 & 1.4645 & 1.5 & -1.6885 \\ 0.0327 & -1.6022 & -0.9562 & -0.2448 \end{bmatrix}$$

The same result can be obtained using the DCT2 command in MATLAB.

It was stated earlier, that the DCT uses only cosine waves to represent a signal. One thing to note is that the cosine transform is not the real part of the unitary DFT. However the cosine transform of a sequence is related to the DFT of its symmetric extension.

Given below are the Cosine Transform basis functions.

This is achieved by plotting each row of the transformation matrix C. If C is 4 × 4 we would get four basic functions. Basis function for N = 16 (C is 16 × 16) is plotted in Fig. P. 8.3.2.

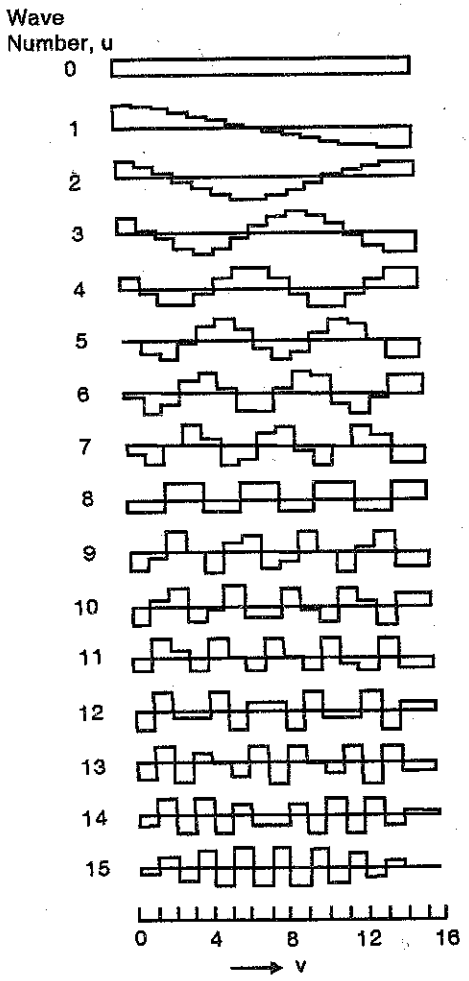


Fig. P. 8.3.2

```
%% Computation of the DCT matrix %%
N=input ('Enter the length of the DCT matrix :')
const=sqrt(2/N);
for u=0:1:N-1
for v=0:1:N-1
if u==0
 c(u+1,v+1)=1/sqrt(N);
else
a=2*v;
c(u+1,v+1)=const*cos((pi*(a+1)*u)/(2*N));
end
 end
end
%% Computation of the DCT of a given sequence %%
x=input('Enter the sequence whos DCT needs to be calculated:')
[M N]=size(x);
const=sqrt(2/N);
for u=0:1:N-1
      for v=0:1:N-1
 if u==0
   c(u+1,v+1)=1/sqrt(N);
else
  a=2*v;
  c(u+1,v+1)=const*cos((pi*(a+1)*u)/(2*N));
end
  end
end
final_dct=(c*x')'
```

```
%% Computation of the DCT of a square image %%
clear
clc
x=imread('saturn.tif'); % Image is square %
[M N]=size(x);
const=sqrt(2/N);
for u=0:1:N-1
  for v=0:1:N-1
 if u==0
      c(u+1,v+1)=1/sqrt(N);
else
a=2*v;
c(u+1,v+1)=const*cos((pi*(a+1)*u)/(2*N));
  end
 end
end
final_dct=c*x*c'
```

8.4 The Sine Transform

The Sine Transform was introduced by A.K. Jain in the paper "A Fast Karhuen - Loéve Transform for Finite Discrete Images" October 1974. The sine transform was found to be a fast algorithmic substitute for the Karhuen-Loéve transform.

The two-dimensional Sine Transform is defined as,

$$F(u, v) = \frac{2}{N+1} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \sin\left[\frac{(x+1)(u+1)\pi}{N+1}\right] \times \sin\left[\frac{(y+1)(v+1)\pi}{N+1}\right]$$

It's inverse is given by,

$$f(x, y) = \frac{2}{N+1} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u, v) \sin\left[\frac{(x+1)(u+1)\pi}{N+1}\right] \times \sin\left[\frac{(y+1)(v+1)\pi}{N+1}\right]$$

The N × N sine transform matrix $\Psi = \Psi(u, v)$ is defined as,

$$\Psi(u, v) = \sqrt{\frac{2}{N+1}} \sin\left[\frac{\pi (u+1)(v+1)}{N+1}\right] \quad 0 \le u, v \le N-1 \quad \ldots(8.4.1)$$

It is this sine transform matrix that we use for calculations.

Given below is a 4 × 4 sine transform matrix Ψ.

		v			
		0.3717	0.6015	0.6015	0.3717
		0.6015	0.3717	– 0.3717	– 0.6015
ψ =	u	0.6015	– 0.3717	– 0.3717	0.6015
		0.3717	– 0.6015	0.6015	– 0.3717

The sine transform is real, symmetric and orthogonal i.e., $\Psi^* = \Psi = \Psi' = \Psi^{-1}$. Thus the forward and inverse sine transforms are identical.

From Equation (8.2.6), we have

$$F = \Psi f \Psi$$

```
%% Code to calculate the 1-D DST %%
clear all
clc
n=[11 12 4 45];
s=size(a);
const=sqrt(2/(s(2)+1));
for n=0:1:s(2)-1
for k=0:1:s(2)-1
      si(n+1,k+1)=const*(sin(pi*(n+1)*(k+1)/(s(2)+1)));
end
end
si
result=si*a'
%% For the answer to match that of MATLAB using the DST command,we need to multiply the result with 1.5881 in case of a 4 point sequence %%
%%%% The DST matrix that we obtain is %%%%
%   0.3717    0.6015    0.6015    0.3717
%   0.6015    0.3717   -0.3717   -0.6015
%   0.6015   -0.3717   -0.3717    0.6015
%   0.3717   -0.6015    0.6015   -0.3717
%   si=conj(si)=inv(si)=si'; si*si' will be identity
```

```
%% Code to calculate the 2-D DST of a square image %%
clear all
clc
a=imread('testpatl.tif');
%a=[1 2 3 4; 3 2 4 5; 1 1 1 1;2 2 2 2]
a=double(a);
[row col]=size(a);
cc=sqrt(row*col);
const=sqrt(2/(cc+1));
for n=0:1:row-1
for k=0:1:col-1
      si(n+1,k+1)=const*(sin(pi*(n+1)*(k+1)/(cc+1)));
end
end
si;
result=si*a*si;
figure(1)
imshow(uint8(a))
figure(2)
```

```
imagesc(log(1+abs(result)))
colormap(gray)
```

The sine transform is not the imaginary part of the unitary DFT. The cosine and sine transforms are not simply the cosine and sine parts of the Fourier transform. In fact, the cosine and sine parts of the Fourier transform, individually, are not orthogonal functions.

Fast implementation of the DST is available in the paper "A Fast Computational Algorithm for Discrete Sine Transform". IEEE Trans. Commun., - 28(2), 1980 - P. Yip and K. R. Rao.

The DST is useful in certain image compression problems.

Given below are the Sine Transform basis functions N = 16. This is done by plotting each row of the sine transform matrix.

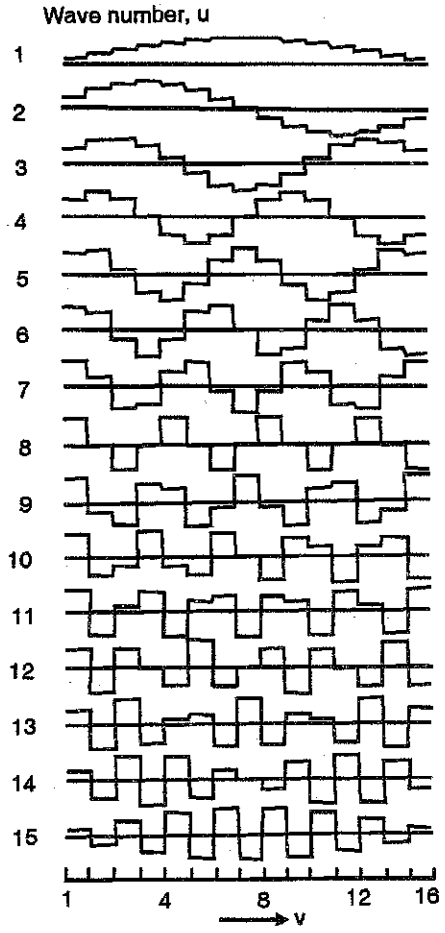
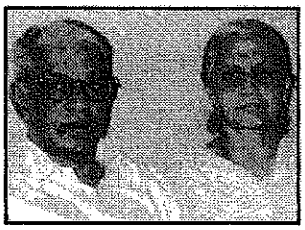
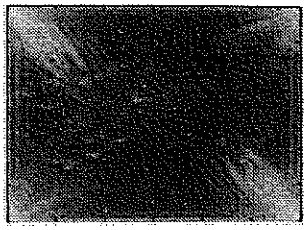


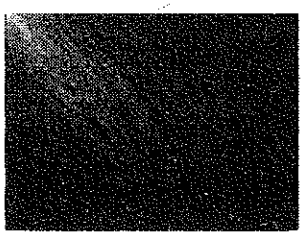
Fig. 8.4.1



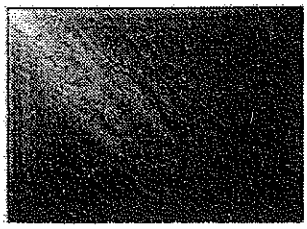
(a) Original image



(b) D.F.T. of the image



(c) D.C.T. of the image



(d) D.S.T. of the image

Fig. 8.4.2

8.5 Non-sinusoidal Transforms

The transforms studied so far viz., Fourier Transform, Discrete Cosine Transform, Sine Transforms etc. were a class of sinusoidal orthogonal functions.

We now introduce a class of non-sinusoidal orthogonal function which consists of Hadamard, Walsh and the Haar functions. These orthogonal functions consist of either square or rectangular waves. As the name suggests, these have basis functions which are not sinusoidal.

Before discussing these transforms in detail, we need to explain a new term sequency that we would encounter in subsequent discussion.

The term frequency is applied to a set of sinusoidal functions (periodic) whose zero-crossing are uniformly spaced over a interval. Frequency is defined as the number of complete cycles generated by a sinusoidal function per unit time. An alternative definition of frequency is one - half the number of zero crossings generated by a sinusoidal function per unit time (Remember: one complete cycle has 2 zero crossings).

The generalisation of frequency is achieved by defining generalised frequency as one-h average number of zero crossings per unit time. H. Harmutz introduced the term sequency to describe generalized frequency, and applied it to distinguish functions whose zero-crossing are not uniformly spaced over a given interval, and which are not necessarily periodic.

Applying the above definition of sequency to periodic and aperiodic function we obtain.

(1) Sequency of a periodic function equals one-half the number of sign changes per period.

(2) Sequency of a periodic function equals one-half the number of sign changes per unit time.

The following illustrations will make this clear.

Consider the two waveforms. Refer Fig. 8.5.1.

Both these functions, $x_1(t)$ and $x_2(t)$, have 4 zero-crossings. The sequency of each of the above functions is 2.

The above definition of sequency can also be applied to discrete functions with minor modifications.

Let $x_1(n)$ and $x_2(n)$ be the sampled versions of $x_1(t)$ and $x_2(t)$; sampled at equidistant points. If the number of sign changes of x(n) equals A, then the sequency of x(n) is defined as A/2 or (A + 1)/2, when A is even or odd respectively.

A close look at $x_1(n)$ and $x_2(n)$ reveals that for $x_1(n)$, A = 3 and for $x_2(n)$, A = 4.

Thus the sequency of $x_1(n)$ and $x_2(n)$ is equal to 2 which is the same for $x_1(t)$ and $x_2(t)$.

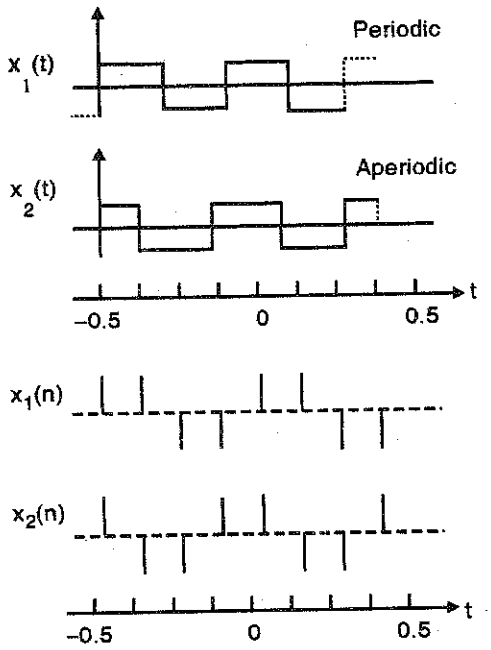


Fig. 8.5.1

8.5.1 The Kronecker Product

The Kronecker product, A × B, of two matrices A and B is obtained by multiplying B with each element a_{ij} of A and substituting the multiplied matrices, $a_{ij}B$, for the elements a_{ij} of A.

Thus if W = A × B, then

$$W = \begin{bmatrix} a_{11}B & a_{12}B & \ldots a_{1n}B \\ a_{21}B & a_{22}B & \ldots a_{2n}B \\ .. & .. & .. \\ a_{n1}B & a_{n2}B & \ldots a_{nn}B \end{bmatrix} \quad \ldots(8.5.1)$$

If A is a n × n matrix and B is a m × m matrix, then W is a mn × mn matrix.

Key to fast algorithms is to form a transform matrix through the Kronecker product of smaller matrices. We shall soon see that the Hadamard transform is generated using this Kronecker product.

Ex. 8.5.1

Given $A = \begin{bmatrix} 1 & 2 \\ 3 & 5 \end{bmatrix}$, $B = \begin{bmatrix} 2 & 5 \\ 1 & 3 \end{bmatrix}$. Find A × B and B × A.

Soln. :

$$A \times B = \begin{bmatrix} 1\begin{bmatrix}2 & 5\\ 1 & 3\end{bmatrix} & 2\begin{bmatrix}2 & 5\\ 1 & 3\end{bmatrix} \\ 3\begin{bmatrix}2 & 5\\ 1 & 3\end{bmatrix} & 5\begin{bmatrix}2 & 5\\ 1 & 3\end{bmatrix} \end{bmatrix}$$

$$\therefore A \times B = \begin{bmatrix} 2 & 5 & 4 & 10 \\ 1 & 3 & 2 & 6 \\ 6 & 15 & 10 & 25 \\ 3 & 9 & 5 & 15 \end{bmatrix}$$

Now, $$B \times A = \begin{bmatrix} 2\begin{bmatrix}1 & 2\\ 3 & 5\end{bmatrix} & 5\begin{bmatrix}1 & 2\\ 3 & 5\end{bmatrix} \\ 1\begin{bmatrix}1 & 2\\ 3 & 5\end{bmatrix} & 3\begin{bmatrix}1 & 2\\ 3 & 5\end{bmatrix} \end{bmatrix}$$

$$\therefore B \times A = \begin{bmatrix} 2 & 4 & 5 & 10 \\ 6 & 10 & 15 & 25 \\ 1 & 2 & 3 & 6 \\ 3 & 5 & 9 & 15 \end{bmatrix}$$

$$\therefore A \times B \neq B \times A$$

8.5.2 Walsh-Hadamard Transform

The Hadamard transform is based on the Hadamard matrix which is a square array having enteries of + 1 and – 1 only.

The Hadamard matrix of order 2 is given by,

$$H(2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

It is also written as $\begin{bmatrix} + & + \\ + & - \end{bmatrix}$.

The rows and columns of the Hadamard matrix are orthogonal. For orthogonality of vectors the dot product has to be zero. Let us check this for H(2).

First row → [1 1]

Second row → [1 –1]

$\therefore$ [1 1] · [1 –1]′ = 0

Similarly,

First column → $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ Second column → $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$$\therefore \begin{bmatrix} 1 \\ 1 \end{bmatrix}' \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 0$$

We have seen earlier, for a unitary transformation, the matrix inverse is given by,

$$A^{-1} = A^{*\prime}$$

A real unitary matrix is called an orthogonal matrix. For such a matrix,

$$A^{-1} = A'$$

Hence to check if a matrix A is orthogonal, we need to check if

$$A^{-1} = A'$$

Multiplying both sides by A, we get,

$$AA^{-1} = AA'$$

$$\therefore AA' = I$$

This is an important condition for orthogonality of matrices.

Ex. 8.5.2

Is the Hadamard matrix orthogonal ?

Soln. :

$$H(2) = A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\therefore A' = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\therefore A \cdot A' = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$AA' = 2\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow AA' = 2[I]$$

Hence the Hadamard matrix is orthogonal but the fact that we get a constant 2 means that it is not normalized.

A normalized 2 × 2 Hadamard transform is given by,

$$H_N(2) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$A' = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$AA' = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\therefore AA' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow AA' = [I]$$

$$\therefore H_N(2) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

is called the normalized 2 × 2 Hadamard matrix.

i.e., $H_N(2)$ is a normalized orthogonal 2 × 2 Hadamard matrix.

Hadamard matrices of order 2^n can be recursively generated through the Kronecker product.

$$H(2^n) = H(2) \times H(2^{n-1}) \quad ...(8.5.2)$$

$$\therefore H(2) = H(2) \times H(2^0)$$

$$H(2) = H(2)$$

Similarly, $H(4) = H(2) \times H(2)$

From Kronecker product we get,

$$H(4) = \begin{bmatrix} 1 \cdot [H(2)] & 1 \cdot [H(2)] \\ 1 \cdot [H(2)] & -1 \cdot [H(2)] \end{bmatrix}$$

Sign changes

$$\therefore H(4) = \left[\begin{array}{cc:cc} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \hdashline 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{array}\right] \quad \begin{matrix} 0 \\ 3 \\ 1 \\ 2 \end{matrix}$$

This is a 4 × 4 Hadamard matrix. As was the case with H(2), rows and columns of H(4) are also orthogonal. Take any two rows or any two columns and you will find that their dot product is zero. Is H(4) orthogonal and normalized ?

We shall check this by using the equation.

$$AA' = I$$

Let

$$H(4) = A$$

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$A' = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$A \cdot A' = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$A \cdot A' = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

$$A \cdot A' = 4 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A \cdot A' = 4[I]$$

Hence H(4) is orthogonal but not normalized.

Normalization is done by multiplying A with $\frac{1}{\sqrt{4}}$.

Hence, the normalized 4 × 4 Hadmard transform is given by,

$$H_N(4) = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

To check if this is normalized,

$$A \cdot A' = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

$$A \cdot A' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [I]$$

Hence $H_N(4)$ is a normalized orthogonal 4 × 4 Hadamard matrix.

We know that the Hadamard matrices of order 2^n can be recursively generated i.e.,

$$H(2^n) = H(2) \times H(2^{n-1})$$

$$\therefore H(8) = H(2) \times H(4)$$

We know, $H(2) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

and $H(4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$

Using the Kronecker product, we have,

$$H(8) = \begin{bmatrix} 1\cdot[H(4)] & 1\cdot[H(4)] \\ 1\cdot[H(4)] & -1\cdot[H(4)] \end{bmatrix}$$

$$H(8) = \left[\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ \hdashline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{array}\right]$$

Sign changes: 0, 7, 3, 4, 1, 6, 2, 5

As in the other cases, all the rows and the columns of H(8) are orthogonal i.e., the dot product of any two rows or any two columns is zero.

We shall show that H(8) is orthogonal.

Using the equation $AA' = I$, we get,

$$H(8)\,H(8)' = \begin{bmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

$$H(8)\,H(8)' = 8\,[I]$$

To normalize H(8), we multiply it by $\frac{1}{\sqrt{8}}$.

$$\therefore H_N(8) = \frac{1}{\sqrt{8}} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Sequency: 0, 7, 3, 4, 1, 6, 2, 5

is a normalized 8 × 8 Hadamard matrix. The rows of Hadamard matrix can be considered to be samples of rectangular waves with sub-period of 1/N units. The Hadamard transform basis function for N = 8 is shown in Fig. 8.5.2. From the above discussion, we realise that the Hadamard representation of signals is similar to the Fourier representation. The dotted lines are the Fourier basis. Each row of H(8) Hadamard matrix is plotted as the basis function.

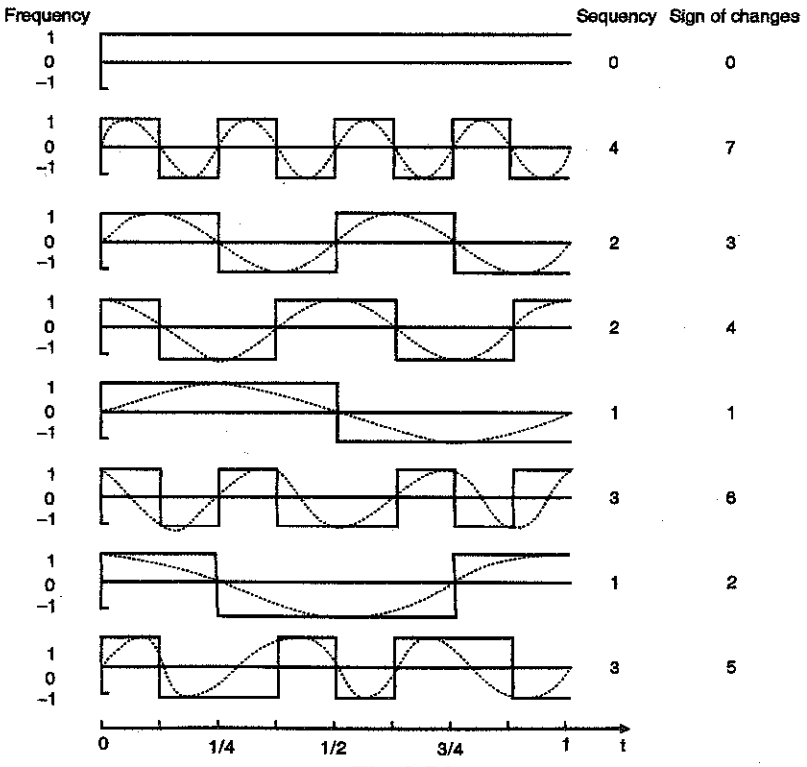


Fig. 8.5.2

If x(n) is a N-point 1-dimensional sequence of finite valued real numbers arranged in a column, then the Hadamard transformed sequence is given by,

$$X = T \cdot x$$

$$X[n] = [H(N)\, x(n)] \qquad \ldots(8.5.3)$$

H(N) is a N × N Hadamard matrix where N is the number of data points.

The inverse Hadamard transform is given by,

$$x(n) = \frac{1}{N} H(N)\, X[n]$$ where x(n) is a column vector.

Ex. 8.5.3

Compute the Hadamard transform of the data sequence $\{1, 2, 0, 3\}'$.

Soln. : Here N = 4, H(N) is a 4 × 4 matrix in this case.

$$X[n] = [H(N) \cdot x(n)]$$

$$X[n] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 0 \\ 3 \end{bmatrix}$$

$$\therefore X[n] = \begin{bmatrix} 6 \\ -4 \\ 0 \\ 2 \end{bmatrix}$$

Ex. 8.5.4

Compute the inverse Hadamard transform of the sequence $\{6, -4, 0, 2\}$.

Soln. : N = 4. The inverse Hadamard transform is

$$x(n) = \frac{1}{N} H(N) \cdot X[n]$$

$$x(n) = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ -4 \\ 0 \\ 2 \end{bmatrix}$$

$$\therefore x(n) = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 3 \end{bmatrix}$$

Hence the data sequence is $\{1, 2, 0, 3\}'$.

For a two dimensional sequence of size N × N, we compute the Hadamard transform using the standard matrix notation using Equation (8.2.6) and Equation (8.2.7).

If f is a N × N image and F is the transformed image, the Hadamard transform is given by,

$$F = T f T = [H(N)\, f\, H(N)] \qquad \ldots(8.5.4)$$

We have used Equation (8.2.6) since H(N) is symmetric.

Ex. 8.5.5

Compute the Hadamard transform of the image shown below.

2	1	2	1
1	2	3	2
2	3	4	3
1	2	3	2

Soln. : The given image is a 4 × 4 image. The transformed image is,

$$F = [H(4)\, f\, H(4)]$$

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$F = \begin{bmatrix} 6 & 8 & 12 & 8 \\ 2 & 0 & 0 & 0 \\ 0 & -2 & -2 & -2 \\ 0 & -2 & -2 & -2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$\therefore F = \begin{bmatrix} 34 & 2 & -6 & -6 \\ 2 & 2 & 2 & 2 \\ -6 & 2 & 2 & 2 \\ -6 & 2 & 2 & 2 \end{bmatrix}$$

Ex. 8.5.6

Find Hadamard transform of following image :

$$f(x, y) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Soln. : The given image is 4 × 4. We hence generate a 4 × 4 Hadamard transform matrix.

$$H(4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

The transformed image is given by the formula

$$F = [H(4) \cdot f \cdot H(4)]$$

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$\therefore F = \begin{bmatrix} 2 & 3 & 1 & 4 \\ -2 & 1 & -1 & 0 \\ 2 & 1 & 1 & 0 \\ 2 & -1 & 3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$\therefore$ Hadamard transformed image is,

$$F = \begin{bmatrix} 10 & -4 & 0 & 2 \\ -2 & -4 & 0 & -2 \\ 4 & 2 & 2 & 0 \\ 4 & 6 & -2 & 0 \end{bmatrix}$$

8.5.3 The Walsh Transform

The Walsh transform matrix is obtained from the Hadamard matrix by re-arranging the rows in increasing sign change order.

The Hadamard matrix is given below

$$H(8) = \left[\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ \hdashline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{array}\right]$$

Sign change: 0, 7, 3, 4, 1, 6, 2, 5

Re-arranging the rows of the Hadamard matrix to get an increasing order of sign changes gives us the Walsh matrix.

$$\therefore H(4) = W(8) = \left[\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ \hdashline 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{array}\right]$$

Sign change: 0, 1, 2, 3, 4, 5, 6, 7

Plotting each of the rows of the Walsh matrix gives us the Walsh basis functions.

Walsh transformation can be calculated using the matrix Equation

$$X[n] = [W(N)\, x(n)] \qquad \ldots(8.5.5)$$

The inverse is given by $x(n) = \frac{1}{N}[W(N)\, X[n]]$.

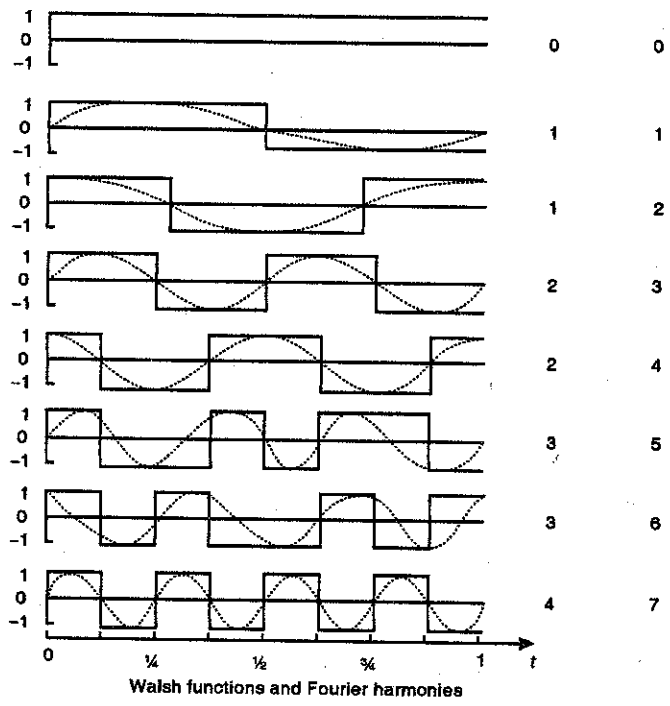


Fig. 8.5.3

Ex. 8.5.7

Compute the Discrete Walsh Transform of the data sequence {1,2,0,3}'.

Soln. : N = 4, we form a Walsh matrix of size 4 × 4.

$$W(4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

$$X[n] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \\ 2 \\ -4 \end{bmatrix}$$

$$X[n] = \{6, 0, 2, -4\}'$$

Ex. 8.5.8

Generate the Walsh transform of the given image.

2	1	2	1
1	2	3	2
2	3	4	3
1	2	3	2

Soln. : The transformed image is given by the formula $F = T \cdot f \cdot T'$.

$$F = [W(4) \cdot f \cdot W(4)']$$

Since W(4) is symmetric.

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

$$\therefore F = \begin{bmatrix} 34 & -6 & -6 & 2 \\ -6 & 2 & 2 & 2 \\ -6 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

Note : We take the scaling function in the inverse formula.

8.5.4 The Haar Transform

The Haar transform is derived from the Haar matrix. The Haar transform like most of the other transforms is separable and can be expressed in matrix form,

$$F = HfH'$$

Where f is a N × N image, H is a N × N transformation matrix and F is the resulting N × N transform. The transformation H contains the Haar basis functions $h_{pq}(x)$ which are defined over the continuous closed interval x[0, 1].

The Haar basis functions are

$$h_{00}(x) = \frac{1}{\sqrt{N}} \quad x \in [0,1]$$

and

$$h_{pq}(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2} & ; \ \frac{q-1}{2^p} \le x < \frac{q-0.5}{2^p} \\ -2^{p/2} & ; \ \frac{q-0.5}{2^p} \le x < \frac{q}{2^p} \\ 0 & ; \ \text{otherwise}, x \in [0,1] \end{cases} \quad \ldots(8.5.6)$$

Where $0 \le p < \log_2 N$, and $1 \le q \le 2^p$.

Let us now generate the transformation matrix. We shall choose different values of N.

Case I : N = 2

$$0 \le p < \log_2 N$$

$$0 \le p < 1$$

$$\therefore p = 0$$

Now, $$1 \le q \le 2^p$$

$$1 \le q \le 2^0$$

$$q = 1$$

We know,

$$h_{00}(x) = \frac{1}{\sqrt{N}} \quad x \in [0,1]$$

$$\therefore h_{00}(x) = \frac{1}{\sqrt{2}} \quad x \in [0,1] \quad \ldots(A)$$

$$h_{pq}(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{0/2} & ; \ \frac{1-1}{2^p} \le x < \frac{1-0.5}{2^p} \ (p=0, q=1) \\ -2^{0/2} & ; \ \frac{1-0.5}{2^p} \le x < \frac{1}{2^p} \\ 0 & ; \ \text{otherwise} \end{cases}$$

$$h_{01}(x) = \frac{1}{\sqrt{2}} \begin{cases} 1; & 0 \le x < 0.5 \\ -1; & 0.5 \le x < 1 \end{cases} \quad \ldots(B)$$

From Equations (A) and (B) we have,

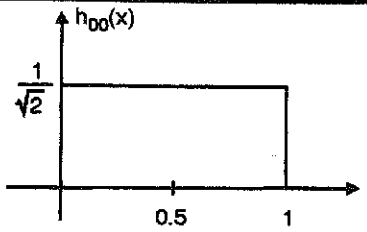


Fig. 8.5.4

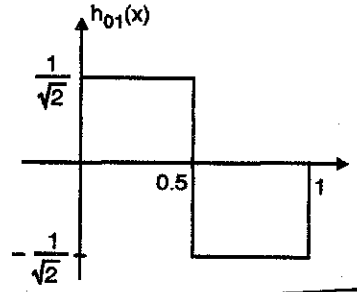


Fig. 8.5.5

Sampling the above two waveforms with N = 2, we get the discrete Haar function.

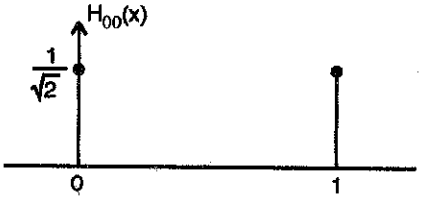


Fig. 8.5.6

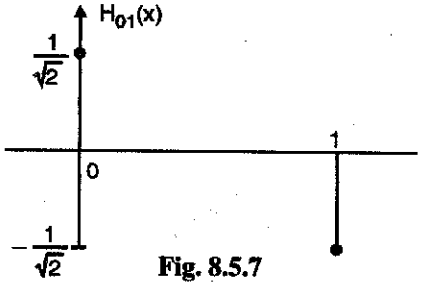


Fig. 8.5.7

Writing this in matrix form, we get,

$$\text{Haar (2)} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

This is the 2 × 2 Haar transformation matrix. It is the same as Hadamard_N (2).

Case II : N = 4

$0 \le p < \log_2 N$ and $1 \le q \le 2^p$

$0 \le p < \log_2 4$ $\quad 0 \le p < \log_2 2^2$

$0 \le p < 2$ $\quad \therefore p = 0, 1$

When,

$p = 0$ $\quad q = 1$

$p = 1$ $\quad 1 \le q \le 2$ $\quad \therefore q = 1, 2$

The Haar basis functions are,

$$h_{00}(x) = \frac{1}{\sqrt{N}}$$

$$h_{pq}(x) = \frac{1}{\sqrt{N}}\begin{cases} 2^{p/2}; & \frac{q-1}{2^p} \le x < \frac{q-0.5}{2^p} \\ -2^{p/2}; & \frac{q-0.5}{2^p} \le x < \frac{q}{2^p} \\ 0; & \text{otherwise, } x \in [0, 1] \end{cases}$$

$$h_{00}(x) = \frac{1}{\sqrt{4}}$$

$$h_{01}(x) = \frac{1}{\sqrt{4}}\begin{cases} 2^{0/2}; & \frac{1-1}{2^0} \le x < \frac{1-0.5}{2^0} \ (p=0, q=1) \\ -2^{0/2}; & \frac{1-0.5}{2^0} \le x < \frac{1}{2^0} \\ 0; & \text{otherwise} \end{cases} \quad \ldots(A)$$

$$\therefore h_{01}(x) = \frac{1}{\sqrt{4}}\begin{cases} 1; & 0 \le x < 0.5 \\ -1; & 0.5 \le x < 1 \\ 0; & \text{otherwise} \end{cases} \quad \ldots(B)$$

$$h_{11}(x) = \frac{1}{\sqrt{4}}\begin{cases} 2^{1/2}; & \frac{1-1}{2^1} \le x < \frac{1-0.5}{2^1} \ (p=1, q=1) \\ -2^{1/2}; & \frac{1-0.5}{2^1} \le x < \frac{1}{2^1} \\ 0; & \text{otherwise} \end{cases}$$

$$\therefore h_{11}(x) = \frac{1}{\sqrt{4}}\begin{cases} \sqrt{2}; & 0 \le x < 0.25 \\ -\sqrt{2}; & 0.25 \le x < 0.5 \\ 0; & \text{otherwise} \end{cases} \quad \ldots(C)$$

$$h_{12}(x) = \frac{1}{\sqrt{4}}\begin{cases} 2^{1/2}; & \frac{2-1}{2^1} \le x < \frac{2-0.5}{2^1} \ (p=1, q=2) \\ -2^{1/2}; & \frac{2-0.5}{2^1} \le x < \frac{2}{2^1} \\ 0; & \text{otherwise} \end{cases}$$

$$\therefore h_{12}(x) = \frac{1}{\sqrt{4}}\begin{cases} \sqrt{2}; & 0.5 \le x < 0.75 \\ -\sqrt{2}; & 0.75 \le x < 1 \\ 0; & \text{otherwise} \end{cases} \quad \ldots(D)$$

From Equation (A), (B), (C) and (D) we get the following waveforms. Next to the waveforms are their sampled version with sampling N = 4.

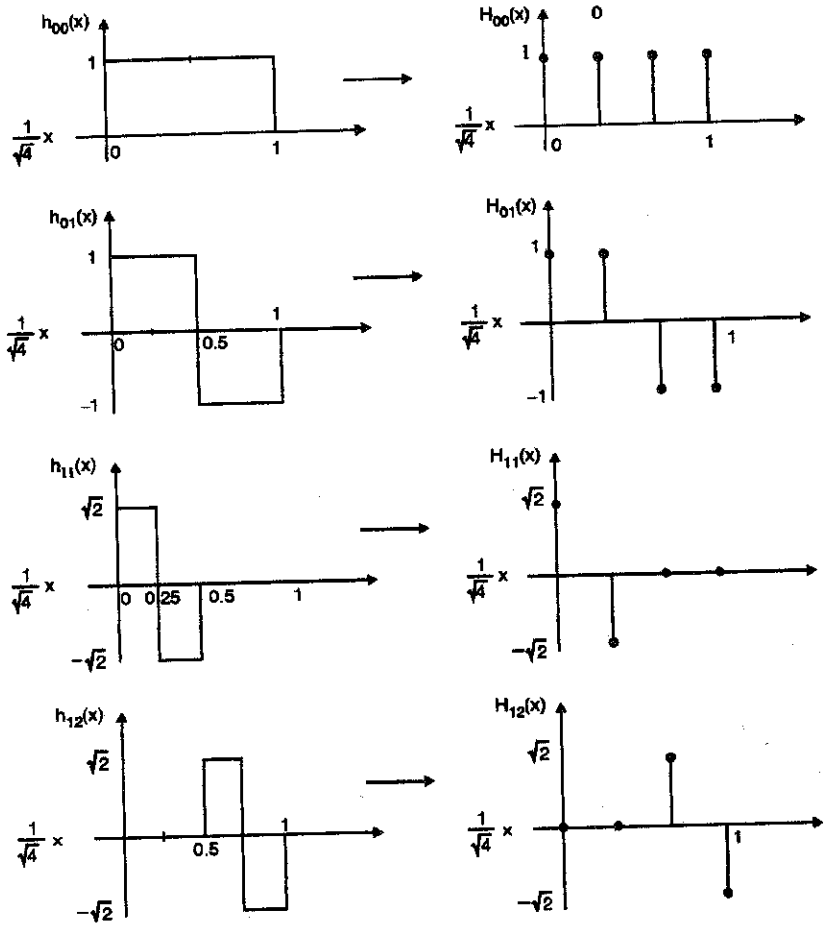


Fig. 8.5.8

Writing this in matrix form, we get,

$$\text{Haar (4)} = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

This is the 4 × 4 Haar transformation matrix.

Case III : N = 8

$$0 \le p < \log_2 N \quad \text{and} \quad 1 \le q \le 2^p$$

$$0 \le p < \log_2 8$$

$$0 \le p < \log_2 2^3$$

$$0 \le p < 3 \log_2 2$$

$$\therefore\ 0 \le p < 3$$

$$\therefore\ p = 0, 1, 2$$

When,

$$\therefore\ p = 0 \qquad 1 \le q \le 2^0 \qquad q = 1$$

$$p = 1 \qquad 1 \le q \le 2^1 \qquad q = 1, 2$$

$$p = 2 \qquad 1 \le q \le 2^2 \qquad q = 1, 2, 3, 4$$

The Haar basis functions are

$$h_{00}(x) = \frac{1}{\sqrt{N}} \quad x \in [0, 1]$$

and

$$h_{pq}(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2}; & \frac{q-1}{2^p} \le x < \frac{q-0.5}{2^p} \\ -2^{p/2}; & \frac{q-0.5}{2^p} \le x < \frac{q}{2^p} \\ 0; & \text{otherwise, } x[0, 1] \end{cases}$$

For N = 8,

$$\therefore\ h_{00}(x) = \frac{1}{\sqrt{8}} \qquad \ldots(1)$$

$$h_{01}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2^{0/2}; & \frac{1-1}{2^0} \le x < \frac{1-0.5}{2^0} \quad (p = 0, q = 1) \\ -2^{0/2}; & \frac{1-0.5}{2^0} \le x < \frac{q}{2^0} \\ 0; & \text{otherwise, } x \in [0, 1] \end{cases}$$

$$\therefore\ h_{01}(x) = \frac{1}{\sqrt{8}} \begin{cases} 1; & 0 \le x < 0.5 \\ -1; & 0.5 \le x < 1 \\ 0; & \text{otherwise} \end{cases} \qquad \ldots(2)$$

$$h_{11}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2^{1/2}; & \frac{1-1}{2^1} \le x < \frac{1-0.5}{2^1} \quad (p = 1, q = 1) \\ -2^{1/2}; & \frac{1-0.5}{2^1} \le x < \frac{1}{2^1} \\ 0; & \text{otherwise} \end{cases}$$

$$\therefore\ h_{11}(x) = \frac{1}{\sqrt{8}} \begin{cases} \sqrt{2}; & 0 \le x < 0.25 \\ -\sqrt{2}; & 0.25 \le x < 0.5 \\ 0; & \text{otherwise} \end{cases} \qquad \ldots(3)$$

$$h_{12}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2^{1/2}; & \frac{2-1}{2^1} \le x < \frac{2-0.5}{2^1} \quad (p = 1, q = 2) \\ -2^{1/2}; & \frac{2-0.5}{2^1} \le x < \frac{2}{2^1} \\ 0; & \text{otherwise} \end{cases}$$

$$\therefore\ h_{12}(x) = \frac{1}{\sqrt{8}} \begin{cases} \sqrt{2}; & 0.5 \le x < 0.75 \\ -\sqrt{2}; & 0.75 \le x < 1 \\ 0; & \text{otherwise} \end{cases} \qquad \ldots(4)$$

$$h_{21}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2^{2/2}; & \frac{1-1}{2^2} \le x < \frac{1-0.5}{2^2} \quad (p = 2, q = 1) \\ -2^{2/2}; & \frac{1-0.5}{2^2} \le x < \frac{1}{2^2} \\ 0; & \text{otherwise } x \in [0, 1] \end{cases}$$

$$\therefore\ h_{21}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2; & 0 \le x < \frac{0.5}{4} \\ -2; & \frac{0.5}{4} \le x < \frac{1}{4} \\ 0; & \text{otherwise } x \in [0, 1] \end{cases} \qquad \ldots(5)$$

$$h_{22}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2^{2/2}; & \frac{2-1}{2^2} \le x < \frac{2-0.5}{2^2} \quad (p = 2, q = 2) \\ -2^{2/2}; & \frac{2-0.5}{2^2} \le x < \frac{2}{2^2} \\ 0; & \text{otherwise} \end{cases}$$

$$\therefore\ h_{22}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2; & \frac{1}{4} \le x < \frac{1.5}{4} \\ -2; & \frac{1.5}{4} \le x < 0.5 \\ 0; & \text{otherwise } x \in [0, 1] \end{cases} \qquad \ldots(6)$$

$$h_{23}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2^{2/2}; & \frac{3-1}{2^2} \le x < \frac{3-0.5}{2^2} \quad (p = 2, q = 3) \\ -2^{2/2}; & \frac{3-0.5}{2^2} \le x < \frac{3}{2^2} \\ 0; & \text{otherwise} \end{cases}$$

$$\therefore\ h_{23}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2; & \frac{1}{2} \le x < \frac{2.5}{4} \\ -2; & \frac{2.5}{4} \le x < \frac{3}{4} \\ 0; & \text{otherwise } x \in [0, 1] \end{cases} \qquad \ldots(7)$$

$$h_{24}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2^{2/2} & ; \frac{4-1}{2^2} \le x < \frac{4-0.5}{2^2} \quad (p=2, q=4) \\ -2^{2/2} & ; \frac{4-0.5}{2^2} \le x < \frac{4}{2^2} \\ 0 & ; \text{otherwise} \end{cases}$$

$$\therefore h_{24}(x) = \frac{1}{\sqrt{8}} \begin{cases} 2 ; & \frac{3}{4} \le x < \frac{3.5}{4} \\ -2 ; & \frac{3.5}{4} \le x < 1 \\ 0 ; & \text{otherwise } x \in [0, 1] \end{cases} \quad \ldots(8)$$

From Equations (1), (2), ... (8), we can draw the following waveforms. We also draw the sampled version of the waveforms N = 8.

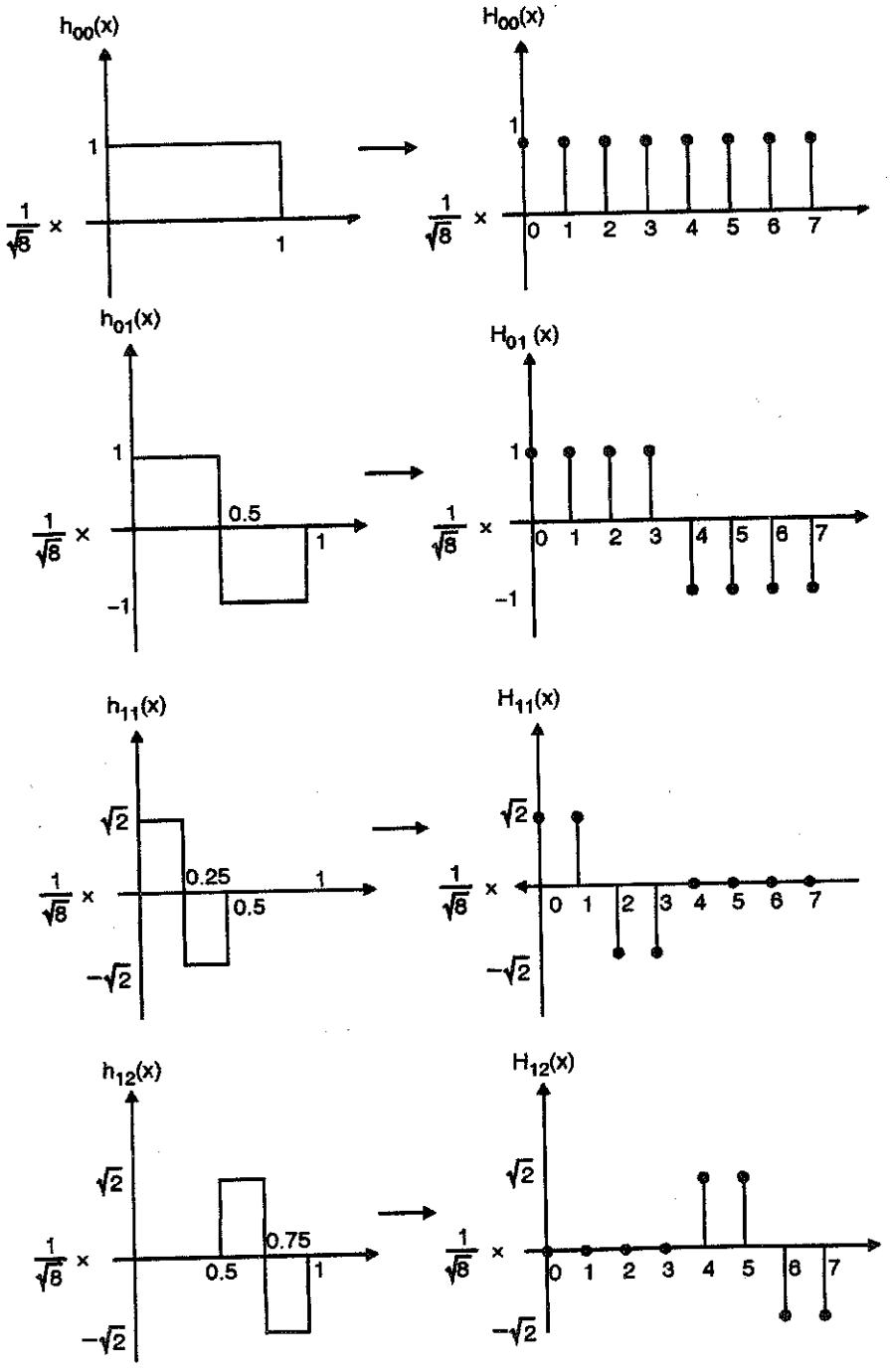


Fig. 8.5.9

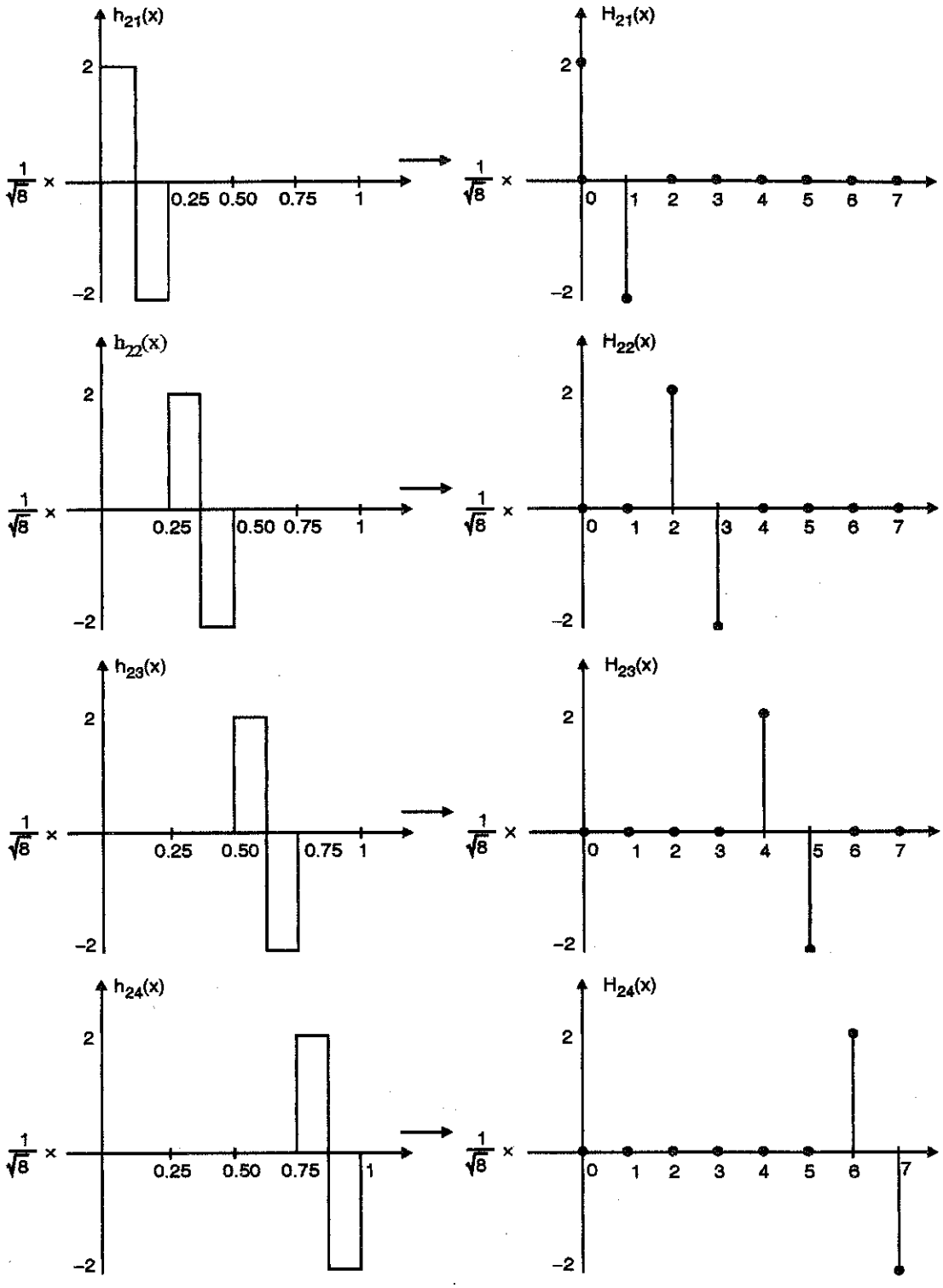

h21(x)
H21(x)
h22(x)
H22(x)
h23(x)
H23(x)
h24(x)
H24(x)
2
-2
1/√8 ×
0.25 0.50 0.75 1
0 1 2 3 4 5 6 7

Fig. 8.5.10

From the sampled version, we generate the following 8×8 Haar transformation matrix.

$$\text{Haar}(8) = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}$$

A lot of books ignore $1 / \sqrt{8}$. One should remember that $1 / \sqrt{8}$ is required to normalize the matrix.

There is an important thing to note here. Whereas the Fourier transform basis functions differ only in frequency, the Haar basis functions vary in both scale (width) and position. This gives the Haar transformation a dual scale-position nature. Such a feature distinguishes it from other transforms like the DFT, DCT, Walsh etc. Haar is used in wavelet transforms.

The advantage of this transformation is that it samples the input data sequence in coarse to fine resolution and takes the difference between adjacent pairs.

When the Haar matrix is pre-multiplied with a vector of 8 elements (input sequence), the first row finds the average of the elements; the second row takes the difference between the average of the first and second halves; the third and fourth rows take the difference between first and second quarters and that of third and fourth quarters and so on. This feature is known as the multi-resolution character of the Haar and establishes the starting point for wavelet transforms.

The Haar transform coefficients, corresponding to a data sequence $x(n) = \{x(0), x(1),...\}'$, are obtained by computing the transformation. Using the standard Equation, we get,

$$X[n] = [\text{Haar}(N) \cdot x(n)]$$

Similarly the Haar transform for an image is given by the standard formula.

$$F = [\text{Haar}(N) \cdot f \cdot \text{Haar}(N)']$$

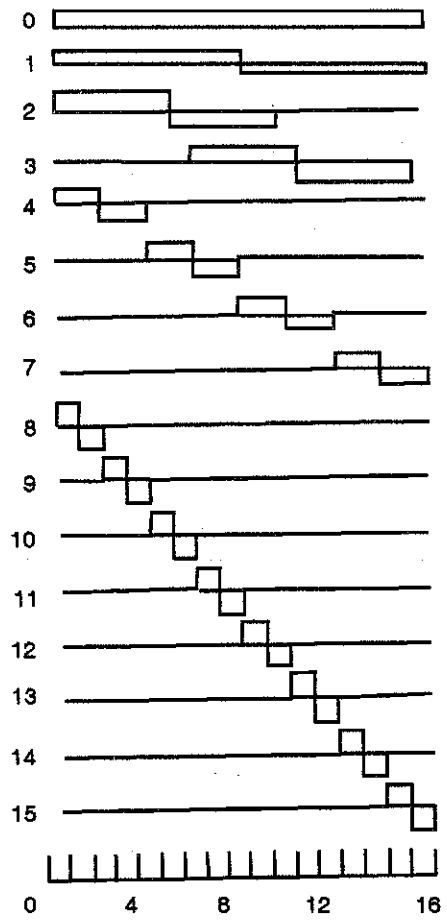


Fig. 8.5.11

Given in Fig. 8.5.11 are the Haar transform basis function for N = 16. This is done by plotting each row of Haar (16).

Ex. 8.5.9

Find the Haar transform of the following signal.

$x(n) = \{1, 2, 0, 3\}'$

Soln. :

$$X[n] = [\text{Haar}(N) \cdot x(n)]$$

$$= \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 0 \\ 3 \end{bmatrix}$$

$$X[n] = \frac{1}{\sqrt{4}} \{6, 0, -\sqrt{2}, -3, \sqrt{2}\}$$

Ex. 8.5.10

Find the Haar transform of the given pseudo image.

2	1	2	1
1	2	3	2
2	3	4	3
1	2	3	2

Soln. :

$$F = [\text{Haar}(N) \cdot f \cdot \text{Haar}(N)']$$

$$= \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}\begin{bmatrix} 2 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 2 \end{bmatrix}$$

$$\frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}'$$

$$F = \begin{bmatrix} 8.5 & -1.5 & -0.707 & 1.414 \\ -1.5 & 0.5 & 0.7071 & 0 \\ -0.7071 & 0.7071 & 1.00 & 0 \\ 1.414 & 0.00 & 0.00 & 0.00 \end{bmatrix}$$

8.5.5 The Slant Transform

The slant transform was designed to have not only a constant first basis function, but a linear second basis function as well. This statement will make sense when we draw the basis functions a little later.

The unitary kernel matrix for the slant transform is obtained by starting with a 2 × 2 Haar or Hadamard transform.

$$H_N(2) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The slant matrix for N = 4 can be written as,

$$S(4) = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ a+b & a-b & -a+b & -a-b \\ 1 & -1 & -1 & 1 \\ a-b & -a-b & a+b & -a+b \end{bmatrix}$$

Where a and b are real constants to be determined subject to the following conditions.

(i) Step size must be uniform.

(ii) S(4) must be orthogonal.

$\frac{1}{\sqrt{4}}$ is for normalized slant and can be neglected.

Let us check the step size of the first two elements of the second row of S(4) (elements of the slant vector).

$$(a+b)-(a-b) = a+b-a+b = 2b$$

The step size between the second and third elements of the second row.

$$(a-b)-(-a+b) = a-b+a-b = 2a-2b$$

The step size must be uniform.

$$\therefore\ 2b = 2a-2b$$

$$\therefore\ a = 2b$$

$$\therefore\ S(4) = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 3b & b & -b & -3b \\ 1 & -1 & -1 & 1 \\ b & -3b & 3b & -b \end{bmatrix}$$

We shall now find the value of 'a' and 'b' using the orthogonal condition.

$$\left\{\frac{1}{\sqrt{4}}[3b \quad b \quad -b \quad -3b]\right\}\left\{\frac{1}{\sqrt{4}}[3b \quad b \quad -b \quad -3b]'\right\} = 1$$

$$\therefore\ b = \frac{1}{\sqrt{5}}$$

$$\therefore\ a = 2b$$

$$\therefore\ a = \frac{2}{\sqrt{5}}$$

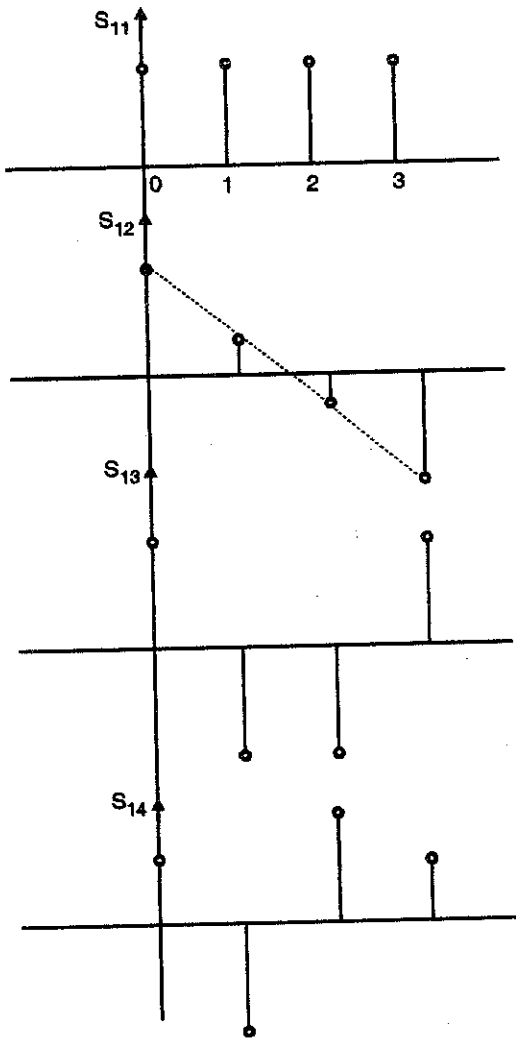


Fig. 8.5.12

Number of sign changes

$$\therefore S(4) = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix}$$

This is the 4×4 normalised slant matrix. We observe that S(4) possesses the sequence property i.e., the number of sign changes go on increasing. We draw each row separately, to get the basis vectors.

Now,

S(4) can be expressed in terms of S(2) such that

$$S(4) = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 0 & 1 & 0 \\ a_4 & b_4 & -a_4 & b_4 \\ 0 & 1 & 0 & -1 \\ -b_4 & a_4 & b_4 & a_4 \end{bmatrix}\begin{bmatrix} S_2 & 0_2 \\ 0_2 & S_2 \end{bmatrix}$$

Here $a_4 = \frac{2}{\sqrt{5}}$ and $b_4 = \frac{1}{\sqrt{5}}$

$$S(4) = \frac{1}{\sqrt{4}} \times \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0.8944 & 0.4472 & -0.8944 & 0.4472 \\ 0 & 1 & 0 & -1 \\ -0.4472 & 0.8944 & 0.4472 & 0.8944 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & -1 \end{bmatrix}$$

$$S(4) = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix}$$

In a similar manner, the relation between S(4) and S(8) can be given by,

$$S(8) = \frac{1}{\sqrt{8}}\left[\begin{array}{cccc:cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ a_8 & b_8 & 0 & 0 & -a_8 & b_8 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hdashline 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ -b_8 & a_8 & 0 & 0 & b_8 & a_8 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]\left[\begin{array}{c:c} S_4 & 0_4 \\ \hdashline 0_4 & S_4 \end{array}\right]$$

$$S(8) = \frac{1}{\sqrt{8}} \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ a_8 & b_8 & 0 & 0 & -a_8 & b_8 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ -b_8 & a_8 & 0 & 0 & b_8 & a_8 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3/\sqrt{5} & 1/\sqrt{5} & -1/\sqrt{5} & -3/\sqrt{5} & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1/\sqrt{5} & -3/\sqrt{5} & 3/\sqrt{5} & -1/\sqrt{5} & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 3/\sqrt{5} & 1/\sqrt{5} & -1/\sqrt{5} & -3/\sqrt{5} \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1/\sqrt{5} & -3/\sqrt{5} & 3/\sqrt{5} & -1/\sqrt{5} \end{array}\right]$$

where a_8 and b_8 are constants.

A generalized way to obtain a slant matrix of order N in terms of the slant matrix of order N/2 is shown below.

$$S(N) = \frac{1}{\sqrt{N}} \left[\begin{array}{cc|cc} \begin{bmatrix} 1 & 0 \\ a_n & b_n \end{bmatrix} & 0 & \begin{bmatrix} 1 & 0 \\ -a_n & b_n \end{bmatrix} & 0 \\ 0 & I_2 \quad I_2 & 0 & I_2 \quad I_2 \\ \hline \begin{bmatrix} 0 & 1 \\ -b_n & a_n \end{bmatrix} & 0 & \begin{bmatrix} 0 & -1 \\ b_n & a_n \end{bmatrix} & 0 \\ 0 & I_2 \quad I_2 & 0 & -I_2 \quad -I_2 \end{array}\right] \left[\begin{array}{c|c} S(N/2) & 0 \\ \hline 0 & S(N/2) \end{array}\right]$$

The coefficients a_N and b_N can be computed using the following relations.

$$a_2 = 1$$

$$b_N = \frac{1}{\left(1 + 4a_{N/2}^2\right)^{1/2}}; \qquad N = 2, 8, 16 \ldots$$

$$a_N = 2b_N\, a_{N/2}$$

Let us find out the S(8) slant matrix. Using the above relationship, we have

$$b_8 = \frac{1}{\left(1 + 4a_4^2\right)^{1/2}}$$

But $$a_4 = \frac{2}{\sqrt{5}}$$

$$\therefore \quad b_8 = \frac{1}{\left(1 + \frac{4 \times 4}{5}\right)^{1/2}}$$

$$\therefore \quad b_8 = 0.4879$$

$$\therefore \quad a_8 = 2b_8\, a_4$$

$$a_8 = 2 \times 0.4879 \times \frac{2}{\sqrt{5}} = 0.8728$$

Substituting b_8 and a_8 in the matrix of S_8, we get,

$$S(8) = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1.5274 & 1.0910 & 0.6546 & 0.2182 & -0.6546 & -1.0910 & -1.5274 \\ 1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 0.4472 & -1.3416 & 1.3416 & -0.4472 & -1.3416 & 1.3416 & -0.4472 \\ 1.3416 & 0.4472 & -0.4472 & -1.3416 & -0.4472 & 0.4472 & 1.3416 \\ 0.6831 & -0.0976 & -0.8782 & -1.6589 & 0.8782 & 0.0976 & -0.6831 \\ 1 & -1 & -1 & 1 & 1 & 1 & -1 \\ 0.4472 & -1.3416 & 1.3416 & -0.4472 & 1.3416 & -1.3416 & 0.4472 \end{bmatrix}$$

$\frac{1}{\sqrt{8}}$ is used for normalization.

Slant matrices are used to define the Slant transform.

$X[n] = [S(N) \cdot x(n)]$

$X[n] \rightarrow$ Slant transform

$S(N) \rightarrow N \times N$ slant matrix

$x(n) \rightarrow$ 1- dimensional data of length N arranged in column.

Ex. 8.5.11

Find the slant transform of the given signal

$$x(n) = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \end{bmatrix}$$

Soln. :

Since N = 4,

We use

$$S(4) = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix}$$

The slant transform is given by,

$X[n] = [S(4) \cdot x(n)]$

$$X[n] = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \end{bmatrix}$$

$$X[n] = \frac{1}{\sqrt{4}} \begin{bmatrix} 6 \\ 0 \\ -2 \\ 0 \end{bmatrix}$$

Hence the N × N slant matrix is used to find the slant transform of a 1-dimensional signal. This discussion can be easily extended to the 2-dimensional case. Since the slant matrix is not symmetric, we use Equation (8.2.7). Given a 2-dimensional image, its normalised slant transform is calculated using the Equation (8.2.7).

$$F = S f S'$$

Ex. 8.5.12

Given a image, calculate the Slant Transform.

$$f = \begin{array}{|c|c|c|c|} \hline 2 & 2 & 2 & 2 \\ \hline 2 & 4 & 4 & 2 \\ \hline 2 & 4 & 4 & 2 \\ \hline 2 & 2 & 2 & 2 \\ \hline \end{array}$$

Soln. : Since the size of the image is 4 × 4, we use normalized S(4).

$$S(4) = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix}$$

$$F = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix} \times$$

$$\frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix}'$$

$$D = \frac{1}{4} \begin{bmatrix} 40 & 0 & -8 & 0 \\ 0 & 0 & 0 & 0 \\ -8 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Given below are the slant basis functions obtained by plotting the rows of S(16).

The Fig. P. 8.5.12 gives us a comparison of the slant transform basis vectors and the Walsh transform (dotted lines). Though we have taken N = 16, only the first 8 basis functions are shown.

Basis Images

Note : Using the Transform matrix, we have learnt how to draw the basic functions by considering the rows (1-Dimensional). We can also draw the basis image using the same matrix.

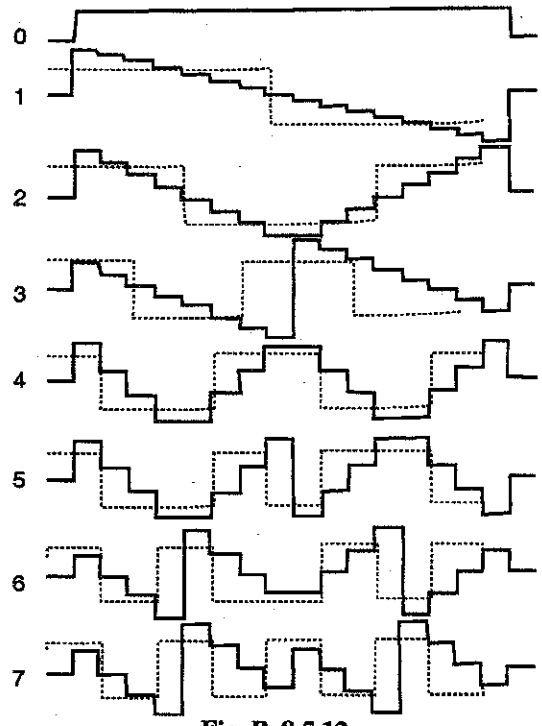


Fig. P. 8.5.12

Ex. 8.5.13

Apply slant transform and DCT transform on the given image and compare the result.

$$f(x,y) = \begin{bmatrix} 2 & 2 & 2 & 1 \\ 2 & 4 & 4 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

Image

Soln. :

A 2-dimensional transform can be computed using the formula

$F = T \cdot f \cdot T'$, where T is the transformation matrix.

(a) Slant transform

A 4×4 slant transformation matrix is given below :

$$S_4 = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & -\frac{3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & -\frac{3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix}$$

Therefore

$$F = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & -\frac{3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & -\frac{3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} 2 & 2 & 2 & 1 \\ 2 & 4 & 4 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

$$\times \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & -\frac{3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & -\frac{3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix}'$$

$$\therefore F = \frac{1}{4} \begin{bmatrix} 40 & 0 & -8 & 0 \\ 0 & 0 & 0 & 0 \\ -8 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) DCT transform

The 4×4 DCT transformation matrix is

$$C = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.2705 & -0.2705 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.2705 & -0.653 & 0.653 & -0.2705 \end{bmatrix}$$

Therefore DCT transformation is

$$F = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.2705 & -0.2705 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.2705 & -0.653 & 0.653 & -0.2705 \end{bmatrix} \begin{bmatrix} 2 & 2 & 2 & 1 \\ 2 & 4 & 4 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix} \times$$

$$\begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.2705 & -0.2705 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.2705 & -0.653 & 0.653 & -0.2705 \end{bmatrix}'$$

$$\therefore F = \begin{bmatrix} 9.75 & 0.326 & -2.25 & 0.135 \\ -0.326 & 0.426 & -0.326 & 0.176 \\ -2.25 & 0.326 & 1.75 & 0.135 \\ -0.135 & 0.176 & -0.135 & 0.073 \end{bmatrix}$$

Ex. 8.5.14

Generate the basis images of Walsh transform W_4.

$$W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix}$$

Soln. :

	0	1	2	3
	1 1 1 1	1 1 −1 −1	1 −1 −1 1	1 −1 1 −1
0	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 −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	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 −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 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
2	−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 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	1 1 −1 −1	1 −1 −1 1	1 −1 1 −1
3	−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 −1 1	1 −1 1 −1
	−1 −1 −1 −1	−1 −1 1 1	−1 1 1 −1	−1 1 −1 1

w(u, v) =

Putting + 1 as white and – 1 as black we get the basis images shown.

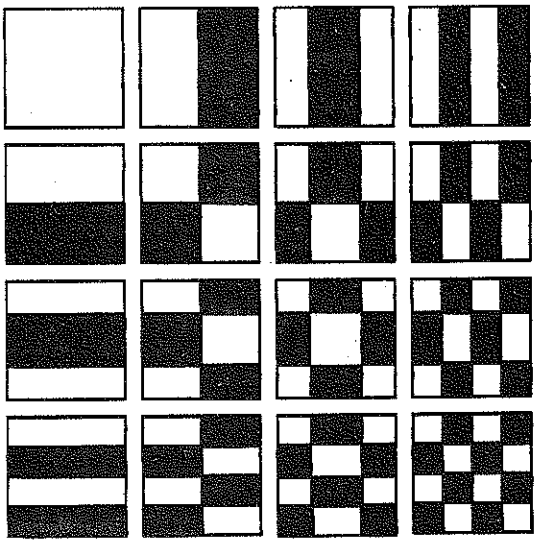


Fig. P. 8.5.14

In a similar fashion, basis images of all the transforms can be obtained.

Ex. 8.5.15

Write four Walsh sequence of length N = 4. Using their sequences, generate sixteen orthogonal Walsh patterns. For the following image, show that any four patterns from the above sixteen patterns are required to fully represent the image.

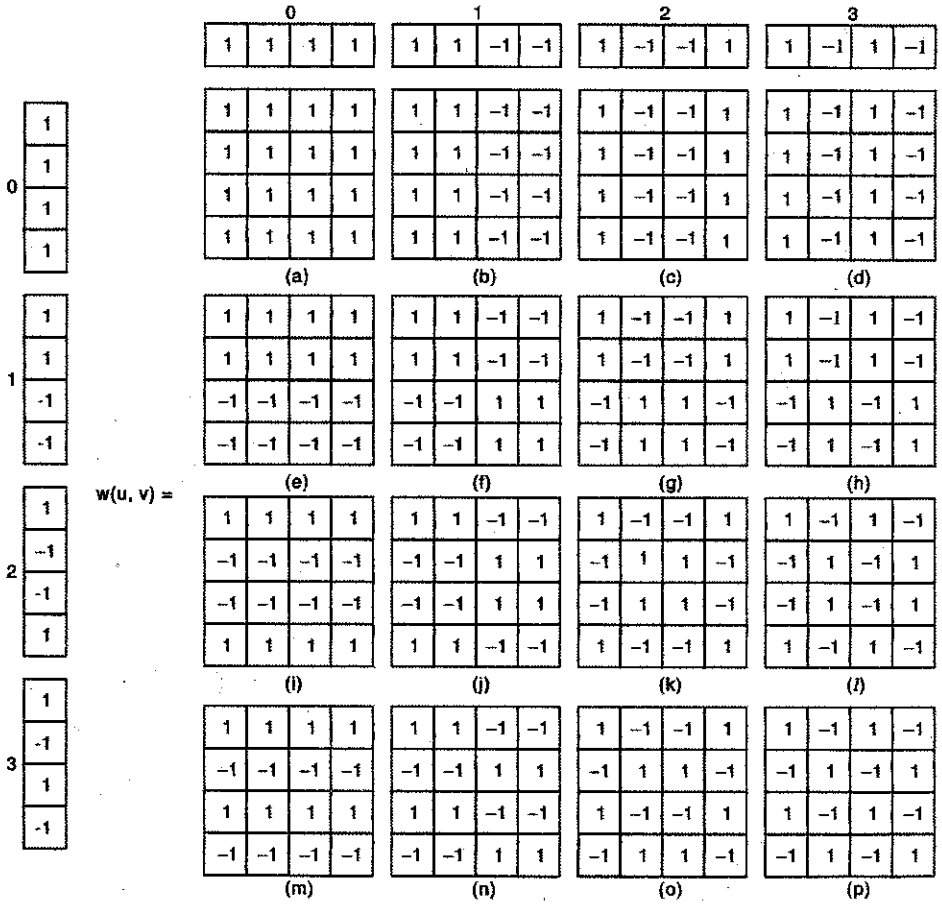
4	4	8	8
4	4	8	8
10	10	12	12
10	10	12	12

Soln. :

A 4 × 4 Walsh matrix is shown below

1	1	1	1
1	1	−1	−1
1	−1	−1	1
1	−1	1	−1

From the 4 rows of the matrix, we generate 16 orthogonal Walsh pattern.



We have learnt that to generate the Walsh transform, we need to multiply the image with each of the 16 basis images. This multiplication is a point by point multiplication

$$F(u, v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cdot w(x, y, u, v)$$

$$\therefore F(0, 0) = \begin{bmatrix} 4 & 4 & 8 & 8 \\ 4 & 4 & 8 & 8 \\ 10 & 10 & 12 & 12 \\ 10 & 10 & 12 & 12 \end{bmatrix} \cdot\times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = 136$$

On close observation we realize that all patterns except a, b, e and f will give a result = 0 when dot multiplied by the original image. Hence we need only 4 basis image to compute the Walsh transform

$$F(0, 1) = \begin{bmatrix} 4 & 4 & 8 & 8 \\ 4 & 4 & 8 & 8 \\ 10 & 10 & 12 & 12 \\ 10 & 10 & 12 & 12 \end{bmatrix} \cdot\times \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} = -24$$

$$F(1, 0) = \begin{bmatrix} 4 & 4 & 8 & 8 \\ 4 & 4 & 8 & 8 \\ 10 & 10 & 12 & 12 \\ 10 & 10 & 12 & 12 \end{bmatrix} \cdot\times \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & +1 & -1 & -1 \end{bmatrix} = -40$$

$$F(1, 1) = \begin{bmatrix} 4 & 4 & 8 & 8 \\ 4 & 4 & 8 & 8 \\ 10 & 10 & 12 & 12 \\ 10 & 10 & 12 & 12 \end{bmatrix} \cdot\times \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} = -8$$

All other terms will be zero. Hence the final Walsh transform of the image is,

$$F(u, v) = \begin{bmatrix} 136 & -24 & 0 & 0 \\ -40 & -8 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Ex. 8.5.16

Generate the Walsh transform from the basis images

$$\begin{bmatrix} 2 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 2 \end{bmatrix}$$

Soln. :

To generate the Walsh transform, we need to multiply the image with each of the 16 basis images.

$$F(u, v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cdot w(x, y, u, v)$$

$$\therefore F(0, 0) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cdot w(x, y, 0, 0)$$

$$\therefore F(0, 0) = \begin{bmatrix} 2 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 2 \end{bmatrix} \cdot\times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$F(0, 0) = (34)$$

Similarly,

$$F(0, 1) = \begin{bmatrix} 2 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 2 \end{bmatrix} \cdot\times \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix}$$

$$F(0, 1) = (-6)$$

Note : $\cdot\times \rightarrow$ Element by element multiplication.

In a similar manner, we multiply the image with every basis image (point by point). This gives us the final Walsh transform.

$$F(u, v) = \begin{array}{|c|c|c|c|} \hline 34 & -6 & -6 & 2 \\ \hline -6 & 2 & 2 & 2 \\ \hline -6 & 2 & 2 & 2 \\ \hline 2 & 2 & 2 & 2 \\ \hline \end{array}$$

This is the same result that we obtained using the standard formula.

$$F = W \cdot f \cdot W'$$

8.6 The Karhunen Loeve Transform (Hotelling Transform)

The K-L transform was proposed by Harold Hotelling (1933) and then developed by Kari Karhunen (1947) and Michel Loéve (1948). It is also referred to as the Eigenvector transform. The K-L transform is based on the statistical properties of the image and has several important properties that make it useful for image processing particularly image compression.

The goal of image compression is to store the image in fewer bits than the original. Compression in images is possible because images contain high degree of redundancies.

Since data from neighbouring pixels in an image is highly correlated, Image compression without ruining the subjective quality of the image becomes quite challenging.

Note : Even if there is a large degree of detail in an image, the grey level of a given pixel is related to its neighbourhood pixels (highly correlated).

By decorrelating this data, more data compression can be achieved. It is advantageous to remove redudancies from a decorrelated data sequence. The K-L transform performs this task of decorrelating the data. Like other orthogonal transforms, the K-L tranform, on its own, does not perform data compression. It merely decorrelates the data which facilitates high degrees of compression.

In order to understand the K-L transform which is based on the statistical properties of the image, we need to proceed with some mathematical definitions. Consider a set of N, one or multi dimensional discrete signals represented as column vectors.

$$x = [x_1, x_2, \ldots x_n]'$$

where each x_N is a vector.

The mean vector of x is defined as,

$$m_x = E\{x\}$$

Where $E\{\cdot\}$ is the Expected value and the subscript x is associated with the population of x vectors.

$$M_x = \frac{1}{N} \sum_{k=1}^{N} x_K \qquad \ldots(8.6.1)$$

The covariance matrix of the vector population is defined as

$$\text{Covariance } (x) = C_x = E\{(x - m_x)(x - m_x)'\} \qquad \ldots(8.6.2)$$

Since x is $N \times 1$ dimensional, $(x - m_x)(x - m_x)'$ and C_x are matrices of the order $N \times N$. The element C_{ii} of the covariance matrix C_x is the variance of x_i, the i^{th} component of the vector population x. The element C_{ij} of the covariance matrix C_x, is the covariance between elements x_i and x_j of the vector population x.

Note : If $C_{ij} = C_{ji} = 0$, then the elements x_i and x_j are uncorrelated.

Covariance for a vector population of size N can be approximated by the formula.

$$\text{Covariance } (x) = \frac{1}{N} \sum_{k=1}^{N} x_K \cdot x_K' - m_x \cdot m_x' \qquad \ldots(8.6.3)$$

The covariance matrix is real and symmetric. We now proceed to compute the eigen values and eigenvectors. Let v_i and λ_i be the eigenvectors and eigenvalues of C_x where $1 \leq i \leq N$. Computation of eigen values and eigenvectors is shown in the example.

The K-L transformation matrix is formed using the eigenvectors. Each eigenvector is arranged as a row of the transformation matrix. The vector corresponding to the largest eigenvalue is placed on the first row and so on.

This K-L Transform matrix, T, is orthogonal i.e.

$$T^{-1} = T'$$

$$T \cdot T' = I$$

We obtain the K-L transformed image by simply multiplying the transformation matrix with the centralised image vector $(x - m_x)$.

$$\therefore X = T \cdot (x - m_x) \quad \ldots(A)$$

This is the formula for K-L transform (Hotelling transform). The output image, X, has a few important properties. Let us start with the mean vector of the output i.e.

$$m_x = E\{X\}$$

From Equation (A), we have,

$$m_x = E\{T(x - m_x)\}$$

$$= E\{T\{x\} - Tm_x\} = TE\{x\} - ETm_x$$

$$= Tm_x - Tm_x$$

$$m_x = 0 \quad \ldots(B)$$

i.e. the mean vector of X is equal to zero. This will be confirmed when we solve a couple of sums. Let us now examine the covariance matrix of X. From Equation (8.6.2) we have

$$\text{Covariance } (X) = C_x = E\{(X - m_x)(X - m_x)'\} \quad \ldots(C)$$

Substituting Equation (A) and (B) in Equation (C) we get,

$$C_X = E\{([T(x - m_x)] - m_x)([T(x - m_x)] - m_x)'\}$$

$$C_X = E\{(Tx - Tm_x - m_x)(Tx - Tm_x - m_x)'\}$$

Since, $m_x = 0$, we get

$$C_X = E\{T(x - m_x) \cdot T'(x - m_x)'\}$$

$$\therefore C_X = T \cdot E\{(x - m_x) \cdot (x - m_x)'\} \cdot T'$$

From Equation (C), we have,

$$C_x = E\{(x - m_x) \cdot (x - m_x)'\}$$

$\therefore$ We have,

$$C_X = T \cdot C_x \cdot T' \quad \ldots(8.6.4)$$

i.e. the covariance matrix of the output, X, can be found out from the covariance matrix of the input x. It can be shown that C_X is a diagonal matrix with elements equal to the eigenvalues of C_X i.e.,

$$C_X = \begin{bmatrix} \lambda_1 & 0 & 0 \ldots\ldots & 0 \\ \vdots & \lambda_{12} & & \\ \vdots & & & \\ \vdots & \ldots\ldots\ldots\ldots & & \lambda_N \end{bmatrix}$$

This will also be observed in the problems that would follow.

The importance of this property is that, since the off diagonal elements of C_X are zero, the elements of X are decorrelated.

In addition, each eigenvalue, λ_i, is equal to the variance of the i^{th} element of X along eigenvector e_i.

It is interesting to note that C_x and C_X have the same eigenvalues as well as the same eigenvectors.

To summarise, the process of computing the K-L transform involves the following steps.

(i) Find the mean vector and the covariance matrix of x.

(ii) Find the eigenvalues and then the eigenvectors of the covariance matrix.

(iii) Create the transformation matrix T, such that the rows of T (basis functions) are the eigenvectors.

(iv) $X = T[C_x - m]$.

Solving a few examples would help us understand the concepts that are stated.

Ex. 8.6.1

Generate the K-L transform of the given image.

$$f = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & -1 & 1 & 2 \end{bmatrix}$$

Soln. :

We break f into columns i.e.

$$f_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}; \qquad f_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

$$f_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}; \qquad f_4 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

The mean vector of f is given by the formula Equation (8.6.1).

$$m_f = \frac{1}{N} \sum_{k=1}^{N} f_K$$

Where N is the number of columns (in this case N = 4).

$$\therefore \; m_f = \frac{1}{4}[f_1 + f_2 + f_3 + f_4] = \frac{1}{4}\begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

Now,

$$m_f \cdot m_f' = \frac{1}{4}\begin{bmatrix} 4 \\ 4 \end{bmatrix} \cdot \frac{1}{4}[4 \;\; 4] = \frac{1}{16}\begin{bmatrix} 16 & 16 \\ 16 & 16 \end{bmatrix}$$

Similarly,

$$\sum_{k=1}^{N} f_K \cdot f_K' = f_1 \cdot f_1' + f_2 \cdot f_2' + f_3 \cdot f_3' + f_4 \cdot f_4'$$

Now,

$$f_1 \cdot f_1' = \begin{bmatrix} 1 \\ 2 \end{bmatrix}[1 \;\; 2] = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

$$f_2 \cdot f_2' = \begin{bmatrix} 2 \\ -1 \end{bmatrix}[2 \;\; -1] = \begin{bmatrix} 4 & -2 \\ -2 & 1 \end{bmatrix}$$

$$f_3 \cdot f_3' = \begin{bmatrix} 1 \\ 1 \end{bmatrix}[1 \;\; 1] = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$f_4 \cdot f_4' = \begin{bmatrix} 0 \\ 2 \end{bmatrix}[0 \;\; 2] = \begin{bmatrix} 0 & 0 \\ 0 & 4 \end{bmatrix}$$

The covariance matrix of f is given by the formula Equation (8.6.3),

$$C_f = \frac{1}{N} \sum_{k=1}^{N} f_k \cdot f_k' - m_f \cdot m_f'$$

$$C_f = \frac{1}{N}[(f_1 \cdot f_1' - m_f \cdot m_f') + (f_2 \cdot f_2' - m_f \cdot m_f') + (f_3 \cdot f_3' - m_f \cdot m_f') + (f_4 \cdot f_4' - m_f \cdot m_f')]$$

$$\therefore \; C_f = \begin{bmatrix} 0.5 & -0.75 \\ -0.75 & 1.5 \end{bmatrix}$$

This is a simple way of computing the covariance matrix. Since the covariance matrix is real and symmetric, we can find the eigenvalues and eigenvectors.

Eigenvalues and Eigenvectors

In mathematics, there are areas where one needs to find all the numbers λ and vectors v that satisfy the equation

$$Av = \lambda v \qquad \ldots(8.6.5)$$

where A is a square matrix.

Any number λ satisfying the above equation is called an eigenvalue of matrix A. A vector v satisfying the above equation is called an eigenvector of A.

The eigenvalues λ are found by solving the Equation

$$|A - \lambda I| = 0 \qquad \ldots(8.6.6)$$

For every λ that is found as a solution of Equation (8.6.6), the corresponding set of eigenvectors v is found from Equation (8.6.5).

Coming back to the example, we find the eigenvalues and eigenvectors of C_f.

We use Equation (8.6.6) to find the eigenvalues λ.

$$|C_f - \lambda I| = 0$$

$$\left| \begin{bmatrix} 0.5 & -0.75 \\ -0.75 & 1.5 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = 0$$

$$\therefore \; \begin{vmatrix} 0.5 - \lambda & -0.75 \\ -0.75 & 1.5 - \lambda \end{vmatrix} = 0$$

$$(0.5 - \lambda)(1.5 - \lambda) - 0.5625 = 0$$

$$\therefore \; 0.75 - 0.5\lambda - 1.5\lambda + \lambda^2 - 0.5625 = 0$$

$$\lambda^2 - 2\lambda + 0.1875 = 0$$

This equation gives us the roots (eigenvalues) as $\lambda_1 = 0.0986$ and $\lambda_2 = 1.9014$.

Using λ_1 and λ_2 we can find the corresponding eigenvectors v_1 and v_2 using Equation (8.6.5) i.e.,

$$Av = \lambda v$$

In this case $A = C_f$

We find the first eigenvector (v_1) corresponding to the first eigenvalue $\lambda_1 = 0.0986$.

We use the equation

$$C_f v_1 = \lambda v_1$$

$$\begin{bmatrix} 0.5 & -0.75 \\ -0.75 & 1.5 \end{bmatrix} \begin{bmatrix} v_{10} \\ v_{11} \end{bmatrix} = 0.0986 \begin{bmatrix} v_{10} \\ v_{11} \end{bmatrix}$$

$$\text{where } v_1 = \begin{bmatrix} v_{10} \\ v_{11} \end{bmatrix}$$

$$\therefore 0.5v_{10} - 0.75v_{11} = 0.0986v_{10} \quad \ldots(a)$$

$$-0.75v_{10} + 1.5v_{11} = 0.0986v_{11} \quad \ldots(b)$$

Rearranging (a) and (b), we get,

$$0.4014v_{10} - 0.75v_{11} = 0 \quad \ldots(c)$$

$$-0.75v_{10} + 1.4014v_{11} = 0 \quad \ldots(d)$$

We could use either (c) or (d) to get a relationship between v_{10} and v_{11}. Let's use Equation (c),

$$\therefore 0.4014v_{10} = 0.75v_{11}$$

$$\therefore v_{10} = 1.8684v_{11}$$

Equation (d) would have given the same result.

Therefore eigenvector v_1 corresponding to eigenvalue λ_1 is

$$v_1 = \begin{bmatrix} 1.8685 \\ 1 \end{bmatrix}$$

Similarly we calculate the second eigenvector v_2 corresponding to eigen value $\lambda_2 = 1.9014$. We use the same equation i.e.,

$$C_f v_2 = \lambda_2 v_2$$

$$\begin{bmatrix} 0.5 & -0.75 \\ -0.75 & 1.5 \end{bmatrix} \begin{bmatrix} v_{20} \\ v_{21} \end{bmatrix} = 1.9014 \begin{bmatrix} v_{20} \\ v_{21} \end{bmatrix}$$

$$\text{where } v_2 = \begin{bmatrix} v_{20} \\ v_{21} \end{bmatrix}$$

$$\therefore 0.5v_{20} - 0.75v_{21} = 1.9014v_{20} \quad \ldots(e)$$

$$-0.75v_{20} + 1.5v_{21} = 1.9014v_{21} \quad \ldots(f)$$

Rearranging (a) and (b), we get,

$$1.4014v_{20} - 0.75v_{21} = 0 \quad \ldots(g)$$

$$-0.75v_{20} + 0.4014v_{21} = 0 \quad \ldots(h)$$

We could use either (g) or (h) to get a relationship between v_{20} and v_{21}. Let's use Equation (g),

$$-1.4014\, v_{20} = 0.75v_{21}$$

$$\therefore v_{20} = -0.5352v_{21}$$

Equation (h) would have given the same result.

$$v_2 = \begin{bmatrix} -0.5352 \\ 1 \end{bmatrix}$$

Hence the two eigenvectors are

$$v_1 = \begin{bmatrix} 1.8685 \\ 1 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} -0.5352 \\ 1 \end{bmatrix}$$

These eigenvectors need to be normalized.

$$v_{1N} = \frac{v_1}{\| v_1 \|}$$

Where v_{1N} is the normalised eigenvector and $\| \cdot \|$ is the Norm

$$v_{1N} = \frac{1}{\sqrt{(1.8685)^2 + (1)^2}} \begin{bmatrix} 1.8685 \\ 1 \end{bmatrix}$$

$$= 0.47186 \begin{bmatrix} 1.8685 \\ 1 \end{bmatrix}$$

$$\therefore v_{1N} = \begin{bmatrix} 0.8817 \\ 0.47186 \end{bmatrix}$$

Similarly we normalise v_2,

$$v_{2N} = \frac{v_2}{\| v_2 \|}$$

$$v_{2N} = \frac{1}{\sqrt{(-0.5352)^2 + (1)^2}} \begin{bmatrix} -0.5352 \\ 1 \end{bmatrix}$$

$$= 0.8817 \begin{bmatrix} -0.5352 \\ 1 \end{bmatrix}$$

$$\therefore v_{2N} = \begin{bmatrix} -0.4719 \\ 0.8817 \end{bmatrix}$$

Using v_{1N} and v_{2N} gives us the Transformation matrix that we are looking for (unlike the other transforms like DFT, the DCT, the Walsh etc., the K-L transformation

matrix depends on the values of the signal). We place the eigenvector corresponding to the largest eigenvalue on the top row.

$$\therefore \quad T = \begin{bmatrix} -0.4719 & 0.8817 \\ 0.8817 & 0.4719 \end{bmatrix}$$

This T is orthogonal i.e. it satisfies Equation (8.2.4), $T \cdot T' = I$.

$$\begin{bmatrix} -0.4719 & 0.8817 \\ 0.8817 & 0.4719 \end{bmatrix} \begin{bmatrix} -0.4719 & 0.8817 \\ 0.8817 & 0.4719 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Hence this is not only orthogonal but also normalised.

To get the final K-L Transformed image, we multiply every centralised image vector $(f - m_f)$ of the original image with the transformation matrix.

$$F = T[f - m_f]$$

where f is every column of the original image.

$$F_1 = T[f_1 - m_f]$$

$$F_1 = \begin{bmatrix} -0.4719 & 0.8817 \\ 0.8817 & 0.4719 \end{bmatrix} \left(\underbrace{\begin{bmatrix} 1 \\ 2 \end{bmatrix}}_{f_1} - \underbrace{\begin{bmatrix} 1 \\ 1 \end{bmatrix}}_{m_f} \right)$$

$$F_1 = \begin{bmatrix} 0.8817 \\ 0.4719 \end{bmatrix}$$

Similarly,

$$F_2 = T \cdot [f_2 - m_f] = \begin{bmatrix} -2.2352 \\ -0.0621 \end{bmatrix}$$

$$F_3 = T \cdot [f_3 - m_f] = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$F_4 = T \cdot [f_4 - m_f] = \begin{bmatrix} 1.3535 \\ -0.4098 \end{bmatrix}$$

Here the image, $f = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & -1 & 1 & 2 \end{bmatrix}$ is

transformed to $F = \begin{bmatrix} 0.8817 & -2.2352 & 0 & 1.3535 \\ 0.4719 & -0.0621 & 0 & -0.4098 \end{bmatrix}$

This is the final K-L Transformed image, we note that the mean vector of F, i.e.

$$m_F = \frac{1}{N} \sum_{k=1}^{N} F_k = 0$$

Computing the covariance of F is a simple task. We use Equation (8.6.4).

Covariance (F) = $T \cdot$ [covariance of f] $\cdot T'$

$$C_F = T \cdot C_f \cdot T'$$

$$\therefore \quad C_F = \begin{bmatrix} -0.4719 & 0.8817 \\ 0.8817 & 0.4719 \end{bmatrix} \begin{bmatrix} 0.5 & -0.75 \\ -0.75 & 1.5 \end{bmatrix} \times \begin{bmatrix} -0.4719 & 0.8817 \\ 0.8817 & 0.4719 \end{bmatrix}$$

$$= \begin{bmatrix} 1.9014 & 0 \\ 0 & 0.0986 \end{bmatrix}$$

Note : One could also calculate the covariance of F, as we did for f using Equation (8.6.2).

It is interesting to note that the diagonal elements are equal to the eigenvalues 0.0986 and 1.9014 found from the covariance matrix of f i.e. C_f. The non-diagonal elements are zero.

Note : MATLAB has an inbuilt command, eig, to compute the eigenvalues and eigenvector. Type >> help eig on the command window.

Ex. 8.6.2

Compute the K-L Transform of the given column vectors.

$x_1 = (1, 0, 1)'$ $x_2 = (1,1,1)'$ $x_3 = (1,1,0)'$ $x_4 = (0,0,1)'$

Soln. :

We first calculate the mean vector which is given by the Equation (8.6.1),

$$\therefore \quad m_x = \frac{1}{N} \sum_{k=1}^{N} X_k$$

$$= [x_1 + x_2 + x_3 + x_4]$$

$$m_x = \frac{1}{4} \begin{bmatrix} 3 \\ 2 \\ 3 \end{bmatrix}$$

We now calculate $m_x \cdot m_x'$

$$m_x \cdot m_x' = \frac{1}{4} \begin{bmatrix} 3 \\ 2 \\ 3 \end{bmatrix} \cdot \frac{1}{4} [3 \quad 2 \quad 3]$$

$$\therefore \quad m_x \cdot m_x' = \frac{1}{16} \begin{bmatrix} 9 & 6 & 9 \\ 6 & 4 & 6 \\ 9 & 6 & 9 \end{bmatrix}$$

We now proceed to compute $\sum_{k=1}^{N} x_k \cdot x_k'$.

$$\sum_{k=1}^{N} x_k\, x'_k = x_1 \cdot x'_1 + x_2 \cdot x'_2 + x_3 \cdot x'_3 + x_4 \cdot x'_4$$

$$x_1 x'_1 = \begin{bmatrix}1\\0\\1\end{bmatrix}[1\ \ 0\ \ 1] = \begin{bmatrix}1 & 0 & 1\\0 & 0 & 0\\1 & 0 & 1\end{bmatrix}$$

$$x_2 x'_2 = \begin{bmatrix}1\\1\\1\end{bmatrix}[1\ \ 1\ \ 1] = \begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

$$x_3 x'_3 = \begin{bmatrix}1\\1\\0\end{bmatrix}[1\ \ 1\ \ 0] = \begin{bmatrix}1 & 1 & 0\\1 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$

$$x_4 x'_4 = \begin{bmatrix}0\\0\\1\end{bmatrix}[0\ \ 0\ \ 1] = \begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 1\end{bmatrix}$$

The covariance matrix is given by the Equation (8.6.3) i.e.,

$$C_x = \frac{1}{N}\sum_{k=1}^{N} x_k \cdot x'_k - m_x \cdot m'_x$$

$$C_x = \frac{1}{N}\{[x_1 \cdot x'_1 - m_x \cdot m'_x] + [x_2 \cdot x'_2 - m_x \cdot m'_x] + [x_3 \cdot x'_3 - m_x \cdot m'_x] + [x_4 \cdot x'_4 - m_x \cdot m'_x]\}$$

$$\therefore C_x = \begin{bmatrix}0.1875 & 0.1250 & -0.0625\\0.1250 & 0.2500 & -0.1250\\-0.0625 & -0.1250 & 0.1875\end{bmatrix}$$

Where C_x is the covariance matrix.

C_x is symmetric and real and hence finding the eigenvectors is always possible.

We first find the eigenvalues. To find the eigenvalues, we use Equation (8.6.6).

$$|C_x - \lambda I| = 0$$

$$\left|\begin{bmatrix}0.1875 & 0.1250 & -0.0625\\0.1250 & 0.2500 & -0.1250\\-0.0625 & -0.1250 & 0.1875\end{bmatrix} - \lambda\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\right| = 0$$

$$\left|\begin{bmatrix}0.1875-\lambda & 0.1250 & -0.0625\\0.1250 & 0.250-\lambda & -0.1250\\-0.0625 & -0.125 & 0.1875-\lambda\end{bmatrix}\right| = 0$$

Solving the above determinant, we get the three roots (λ values) as 0.0732, 0.1250 and 0.4268.

i.e. $\lambda_1 = 0.0732$

$\lambda_2 = 0.1250$

$\lambda_3 = 0.4268$

Using the three eigenvalues, we can find the corresponding eigenvectors v_1, v_2 and v_3. We use Equation (8.6.5), i.e.,

$$Av = \lambda v$$

in this case $A = C_x$

To find the first eigenvector v_1, corresponding to the eigenvalues $\lambda_1 = 0.0732$, we use

$$C_x v_1 = \lambda_1 v_1$$

i.e.,

$$\begin{bmatrix}0.1875 & 0.125 & -0.0625\\0.125 & 0.250 & -0.1250\\-0.0625 & -0.125 & 0.1875\end{bmatrix}\begin{bmatrix}v_{10}\\v_{11}\\v_{12}\end{bmatrix} = [0.0732]\begin{bmatrix}v_{10}\\v_{11}\\v_{12}\end{bmatrix}$$

We get three equations,

$$0.1875v_{10} + 0.125v_{11} - 0.0625v_{12} = 0.0732v_{10} \quad ...(a)$$

$$0.125v_{10} + 0.25v_{11} - 0.125v_{12} = 0.0732v_{11} \quad ...(b)$$

$$-0.0625v_{10} - 0.125v_{11} + 0.1875v_{12} = 0.0732v_{12} \quad ...(c)$$

Rearranging Equations (a), (b) and (c), we get

$$0.1143v_{10} + 0.125v_{11} - 0.0625v_{12} = 0 \quad ...(d)$$

$$0.125v_{10} + 0.1768v_{11} - 0.125v_{12} = 0 \quad ...(e)$$

$$-0.0625v_{10} - 0.125v_{11} + 0.1143v_{12} = 0 \quad ...(f)$$

Solving this gives us the eigenvector v_1, corresponding to the eigen value λ_1. These are then normalised.

$$v_{1N} = \begin{bmatrix}0.5\\-0.7071\\-0.5\end{bmatrix}$$

To find the second eigenvector v_2 we use the equation,

$$C_x v_2 = \lambda_2 v_2$$

$$\text{i.e.}\begin{bmatrix}0.1875 & 0.125 & -0.0625\\0.125 & 0.25 & -0.125\\-0.0625 & -0.125 & 0.1875\end{bmatrix}\begin{bmatrix}v_{20}\\v_{21}\\v_{22}\end{bmatrix} = [0.125]\begin{bmatrix}v_{20}\\v_{21}\\v_{22}\end{bmatrix}$$

Solving this gives us the second eigenvector v_2, corresponding to the second eigen value λ_2. This is then normalised.

$$v_{2N} = \begin{bmatrix} 0.7071 \\ -0.00 \\ 0.7071 \end{bmatrix}$$

To find the third eigenvector v_3 we use the equation

$$C_x v_3 = \lambda_3 v_3$$

$$\text{i.e.} \begin{bmatrix} 0.1875 & 0.125 & -0.0625 \\ 0.125 & 0.25 & -0.125 \\ -0.0625 & -0.125 & 0.1875 \end{bmatrix} \begin{bmatrix} v_{30} \\ v_{31} \\ v_{32} \end{bmatrix} = [0.4268] \begin{bmatrix} v_{30} \\ v_{31} \\ v_{32} \end{bmatrix}$$

Solving this gives us the third eigenvector v_3, corresponding to the third eigenvalue λ_3. This is then normalised.

$$v_{3N} = \begin{bmatrix} -0.5 \\ -0.7071 \\ 0.5 \end{bmatrix}$$

Hence the three normalised eigenvectors are

$$v_{1N} = \begin{bmatrix} 0.5 \\ -0.7071 \\ -0.5 \end{bmatrix}, \quad v_{2N} = \begin{bmatrix} 0.7071 \\ 0.00 \\ 0.7071 \end{bmatrix},$$

$$v_{3N} = \begin{bmatrix} -0.5 \\ -0.7071 \\ 0.5 \end{bmatrix}$$

Using v_{1N}, v_{2N} and v_{3N} gives us the K-L Transformation matrix. We arrange them in rows. The eigenvector corresponding to the largest eigenvalue is placed at the top and so on.

$$T = \begin{bmatrix} -0.5 & -0.7071 & 0.5 \\ 0.7071 & 0.00 & 0.7071 \\ 0.5 & -0.7071 & -0.5 \end{bmatrix}$$

This T as expected is orthogonal i.e. it $T \cdot T' = I$.

To get the final K-L Transform, we multiply the centralised image vector $(x - m_x)$ by T.

i.e. $X = T[x - m_x]$

$$X_1 = T \cdot (x_1 - m_x), \quad X_2 = T \cdot (x_2 - m_x),$$

$$X_3 = T \cdot (x_3 - m_x), \quad X_4 = T \cdot (x_4 - m_x)$$

$$X_1 = \begin{bmatrix} 0.3535 \\ 0.3535 \\ 0.3535 \end{bmatrix}, \quad X_2 = \begin{bmatrix} -0.3535 \\ 0.3535 \\ -0.3535 \end{bmatrix}$$

$$X_3 = \begin{bmatrix} -0.8536 \\ -0.3535 \\ 0.1465 \end{bmatrix}, X_4 = \begin{bmatrix} 0.8536 \\ -0.3535 \\ -0.1465 \end{bmatrix}$$

Hence the final K-L transformed image is

$$X = \begin{bmatrix} 0.3535 & -0.3535 & -0.8536 & 0.8536 \\ 0.3535 & 0.3535 & -0.3535 & -0.3535 \\ 0.3535 & -0.3535 & 0.1465 & -0.1465 \end{bmatrix}$$

As stated in the earlier example, covariance of the output can be calculated by using the transformation T and the original covariance matrix.

$$T \cdot C_x \cdot T' = \begin{bmatrix} 0.4268 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 0.0732 \end{bmatrix}$$

We see that the diagonal elements are equal to the eigen values and the non-diagonal elements are equal to zero.

Another important aspect of the K-L transform is that the mean vector of X is equal to zero.

$$\text{i.e., } m_X = \frac{1}{N} \sum_{k=1}^{N} X_k = [X_1 + X_2 + X_3 + X_4] = 0$$

Reconstruction : Since the L-K transformation matrix is orthogonal, x can be reconstructed from X using the equation

$$x = T' X + m_X \qquad ...(8.6.7)$$

Suppose, instead of using all eigenvectors of C_x, we create T from only the K largest eigenvector ($K < N$). This will give us a new transformation matrix T_K which would be of the order $K \times N$. X would now be K dimensional and the reconstruction [using Equation (8.6.7)] would no longer be exact.

The new reconstructed vector $\tilde{x}$ would be

$$\tilde{x} = T_K' X + m_X \qquad ...(8.6.8)$$

This elimination of the smaller eigenvalues of C_x introduces an error i.e. $x \neq \tilde{x}$

The mean square error between x and $\tilde{x}$ is given by the formula.

$$e_{ms} = \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \lambda_i \qquad ...(8.6.9)$$

$$e_{ms} = \sum_{i=k+1}^{N} \lambda_i$$

If K = N (i.e. we use all the eigenvectors) we note that the error is zero as expected. It is simple to conclude that the error can be minimized by selecting eigenvectors associated with the largest eigenvalues.

Because of this concept of using eigenvectors associated with the largest eigenvalues to reduce the error, the K-L transform is also know as the "Principal components transform". The K-L transform is defined as a linear transformation whose basis vectors are the eigen vectors of the covariance matrix of the image data. As it diagonalises the covariance matrix, it decorrelates the data, thus making it suitable for compression. The resulting set of coefficients can be encoded using fever bits-Compression.

While the K-L transform has, theoretically, the optimal decorrelation property, it is seldom used in practice as computational requirements are very high. Unlike the other transforms given in the book, the K-L transform basis set (The transformation matrix) is image dependent. Every image will have a different K-L transform matrix. For practical purposes, sub-optimal transforms are utilised, which diagonalize the covariance matrix only approximately. The K-L transform is used as a point of reference in assessing how effective the other transforms are in decorrelating the data.

Ex. 8.6.3

Three column vectors are given below

$x_1 = [1\ 1\ 1]$, $x_2 = [-2\ 1\ 1]$, $x_3 = [0\ -1\ 1]$

Show that they are orthogonal. Also generate all possible patterns.

Soln. :

Simple condition for orthogonality is that the dot product of the two vectors should be zero. We check this condition

$$x_1 \cdot x_2' = [1\ 1\ 1] \cdot \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

$$= 0$$

$$x_1 \cdot x_3' = [1\ 1\ 1] \cdot \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$$

$$= 0$$

$$x_2 \cdot x_3' = [-2\ 1\ 1] \cdot \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$$

$$= 0$$

Hence we conclude that x_1, x_2 and x_3 are orthogonal.

Generating the patterns (basis images) is a simple task.

		x_1 [1 1 1]	x_2 [-2 1 1]	x_3 [0 -1 1]
x_1	$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -2 & 1 & 1 \\ -2 & 1 & 1 \\ -2 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 1 \\ 0 & -1 & 1 \\ 0 & -1 & 1 \end{bmatrix}$
x_2	$\begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -2 & -2 & -2 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 4 & -2 & -2 \\ -2 & 1 & 1 \\ -2 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & -2 \\ 0 & -1 & 1 \\ 0 & -1 & 1 \end{bmatrix}$
x_3	$\begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 2 & -1 & -1 \\ -2 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$

Ex. 8.6.4

Find DFT for the set of values {1, 0, 1, 2, 0, 0, 0, 1}.

Soln. :

We know that the 1-D DFT can be computed using Equation (8.2.5) which is

$$F = Wf$$

Here W is the DFT matrix and f is the 1-D signal arranged as a column. Since the given signal is of length 8, we generate a 8x8 DFT matrix and multiply it by the input signal.

$$\begin{bmatrix} 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 \\ 1.0000 & 0.7071-0.7071i & 0-1.0000i & -0.7071-0.7071i & -1.0000 & -0.7071+0.7071i & 0+1.0000i & 0.7071+0.7071i \\ 1.0000 & 0-1.0000i & -1.0000 & 0+1.0000i & 1.0000 & 0-1.0000i & -1.0000 & 0+1.0000i \\ 1.0000 & -0.7071-0.7071i & 0+1.0000i & 0.7071-0.7071i & -1.0000 & 0.7071+0.7071i & 0-1.0000i & -0.7071+0.7071i \\ 1.0000 & -1.0000 & 1.0000 & -1.0000 & 1.0000 & -1.0000 & 1.0000 & -1.0000 \\ 1.0000 & -0.7071+0.7071i & 0-1.0000i & 0.7071+0.7071i & -1.0000 & 0.7071-0.7071i & 0+1.0000i & -0.7071-0.7071i \\ 1.0000 & 0+1.0000i & -1.0000 & 0-1.0000i & 1.0000 & 0+1.0000i & -1.0000 & 0-1.0000i \\ 1.0000 & 0.7071+0.7071i & 0+1.0000i & -0.7071+0.7071i & -1.0000 & -0.7071-0.7071i & 0-1.0000i & 0.7071-0.7071i \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$F = \begin{bmatrix} 5.0000 & 0.2929-1.7071i & 0+3.0000i & 1.7071+0.2929i & -1.0000 & 1.7071-0.2929i & 0-3.0000i & 0.2929+1.7071i \end{bmatrix}$$

Summary

The term image transforms usually refers to a class of unitary matrices used for representing images. Just as a one-dimensional signal can be represented by a orthogonal set of basis functions, an image can also be expanded in terms of a set of basis arrays called as basis images. In this chapter, the preliminary concepts of some image transforms have been discussed. Transforms such as the Discrete Fourier transform, Discrete Cosine transform, Sine transform, Walsh-Hadamard transform, K-L transform etc. have been explained in detail taking pseudo-images.

Review Questions

Q. 1 State and explain the properties of conjugate symmetry and periodicity as applied to two-dimensional DFT.

Q. 2 Obtain the DFT coefficients of the following pixel array.

1	4	1	2
3	1	0	0
4	2	3	1
1	4	2	5

Q. 3 Write a 4 × 4 transform matrix for each of the following transforms

(a) DFT (b) DCT

(c) DST (d) Walsh

(e) Hadamard (f) Haar

Q. 4 Explain the performance of each of the above transforms considering the following points.

(a) Energy compaction

(b) Convolution

(c) Computational complexity

(d) Feature extraction

(e) Digital filtering

Q. 5 State and explain the difference between DFT and DCT.

Q. 6 If the kernel of an image transform is seperable and symmetric, the transform can be expressed in matrix form. Explain.

Q. 7 Write short notes on :

(a) Role of orthogonal transforms

(b) Choice of DFT v/s DCT for transform coding

(c) Application of Slant and Walsh Hadamard transform.

Q. 8 What is a 2-dimensional DFT and how is it related to a 1-dimensional DFT ?

Q. 9 If a one-dimensional N-point FFT algorithm is available, can it be used to obtain a 2-dimensional DFT of a image of size N × N ? Elaborate the procedure.

Q.10 For a given orthogonal matrix A and image u, find the transformed image

$$A_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix};\quad u = \begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}.$$

Q. 11 Develop an algorithm for a fast Hadamard transform.

Q. 12 Assuming square image arrays and the forward and inverse image transforms are separable and symmetric, show that the forward and the inverse transforms can be expressed in matrix form. What are the advantages of this formulation ?

Q. 13 Check whether the following matrix are unitary and/or orthogonal

$$A_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix};\quad A_2 = \begin{bmatrix}\sqrt{2} & j\\ -j & \sqrt{2}\end{bmatrix};$$

$$A_3 = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & j\\ j & 1\end{bmatrix}$$

Q. 14 Prove the following properties of the Kronecker products

(a) $(A + B) \otimes C = A \otimes C + B \otimes C$

(b) $(A \otimes B) \otimes C = A \otimes (B \otimes C)$

(c) $(A + B)^T = A^T \otimes B^T$

(d) Trace $[A \otimes B]$ = [Trace (A)] · [Trace (B)]

Note : Trace $= \sum_{n=m} a(n, m)$ (Sum of diagonal elements)

Chapter Ends...

❑❑❑

CHAPTER 9

Image Compression

9.1 Introduction

Data Storage and Transmission cost money. In spite of this, most digital data are not stored in a compact form. Rather, they are stored in whatever way that makes them easy to use such as: ASCII text for word processors, binary code that can be executed on a computer, BMP formats for images etc. These easy to use encoding methods require data files about twice as large as actually needed to represent the information. Data compression is the term used for various algorithms developed to address this problem.

We have seen that the amount of storage space required to store images is enormous (Remember $A \times B \times C \times D$?).

Since images are quite data intensive, reducing their size can produce results that are more ambitious than would otherwise be practical. Image compression addresses the problem of reducing the amount of data required to represent a image.

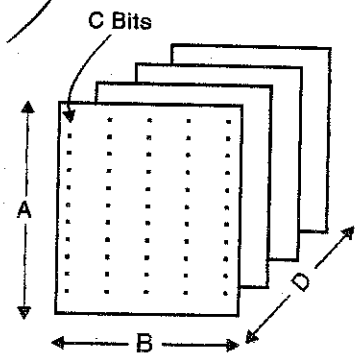


Fig. 9.1.1

9.2 Redundant and Irrelevant Data

Image files commonly contain considerable amount of data that is redundant and much that is irrelevant. Due to this, images are prime candidates for compression. Let us take a simple example to see how compression works. Mr. Rahul who is in Calcutta for a Business meeting hails from Bombay. He is supposed to be boarding the train back to Bombay today evening. He has received a telegram today morning which has the following message.

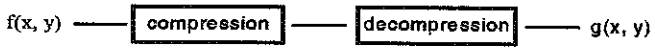
"Your wife Shilpa will receive you at C.S.T. station in Bombay at 4.30 PM tomorrow afternoon."

Now if this telegraphic message has to be shortened, we need to get rid of information that is already known. Surely Rahul knows that C.S.T. is in Bombay as he stays there. He also does not need to be reminded that his wife's name is Shilpa. When the time is written as 4:30 PM, the word afternoon can be dropped off. Eliminating all this data, we get,

"Your wife will receive you at C.S.T. Station at 4:30 PM tomorrow"

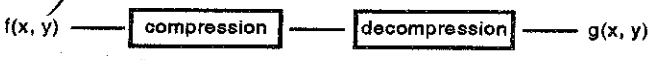
Here we have compressed the data without any loss of information. Hence data compression techniques exploit inherent redundancies and irrelevancies by transforming a data file into a smaller file from which the original file can later be reconstructed exactly or approximately. The ratio of the two file sizes (known as compression ratio) specify the degree of compression. There exists two different categories of data compression algorithms. They could either be Loss-less or they could be Lossy.

The loss-less algorithm eliminates only redundant information, so that one can recover the image exactly upon decompression of the file. In other words the restored image is identical to the original image.



For loss-less algorithms, $f(x, y) = g(x, y)$

Lossy algorithms eliminate redundant as well as irrelevant information and thus permit only an approximate reconstruction of the original image rather than an exact duplicate. Information is lost when lossy algorithms are used.



For Lossy compression $f(x, y) \neq g(x, y)$. $g(x, y)$ is smaller compared to $f(x, y)$.

Loss-less algorithms are absolutely necessary for many types of data, for example : Executable codes (EXE's), word processing files, tabulated numbers etc. In comparison, data files that represent images and other acquired signals do not have to be kept in perfect condition for storage and transmission. For images, a slight loss of fidelity is often an acceptable trade off for a much higher degree of compactness.

As one might expect, lossy compression algorithm achieve higher compression ratios. Images transmitted over the Internet are an excellent example of why image compression is so important. To understand the basic concept of compression in mathematical terms, let us consider the following example in which a 1-dimensional signal (the discussion could be extended to 2–dimensional signals as well) is represented as a vector f having N elements, i.e.,

$$f = (f_0, f_1, f_2 \ldots f_{N-1})$$

Where f_j represents the j^{th} sampled value.

If every element, f_0, f_1, f_2 ... , is coded by b bits, the total number of bits required to represent this digital signal is Nb. Now suppose that the vector f is transformed to another vector F, by a transformation matrix T i.e.,

$$F = Tf$$

Where F is represented by K bits and $K < Nb$. This may be achieved either by reducing N to N′ or by reducing b to b′. Thus the amount of data reduction is given by

$$C = \frac{Nb - K}{Nb} \cdot 100\ \% \qquad \ldots(9.2.1)$$

Notice, if $K = Nb$, $C = 0$ i.e., there is no data reduction. We can now reconstruct the digital signal f from F using inverse transformation. Let us see where we are. We have a original signal f which is represented by Nb bits. This vector is transformed to another vector F which is represented by K bits ($K < Nb$). We now reconstruct the digital signal f from F.

Any discrepancy between the original signal f and the new signal $\tilde{f}$ is the error introduced due to the process of compression. Usually amount of error increases with the reduction in the amount of data i.e., error increases as the difference between Nb and K increases. The objective of compression is to achieve maximum C without introducing objectionable error. If the error introduced is zero i.e., $\tilde{f}$ is exactly equal to f, we call it loss-less compression otherwise it is called lossy compression.

Another simple formula to measure compression ratio is

$$CR = \frac{\text{Number of bits in input image}}{\text{Number of bits in output image}} \qquad \ldots(9.2.2)$$

9.3 Error Criteria

Error criteria is required to judge the performance of a lossy compression technique. It simply means how much of error is considered to be tolerable. The error criteria can be classified into two broad groups.

(1) Objective Error (Fidelity) Criteria

(2) Subjective Error (Fidelity) Criteria.

9.3.1 Objective Error Criteria

This is a mathematical formulation to quantify the error introduced in Lossy compression. Let us denote the

original image and the reconstructed image by f(x, y) and $\tilde{f}(x, y)$ respectively. Let the physical size of both the images be M × N.

The objective error criteria most often used is the mean-squared error.

This is defined as,

$$e_{ms} = \frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \{f(x, y) - \tilde{f}(x, y)\}^2 \quad \ldots (9.3.1)$$

Sometimes the root mean squared error is preferred over the means squared error.

This defined as

$$e_{rms} = \left[\frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \{f(x, y) - \tilde{f}(x, y)\}^2 \right]^{1/2} \quad \ldots (9.3.2)$$

Since the value of the e_{rms} depends on the range of f(x, y), it is a practice to have the e_{rms} normalized. As a result of this normalisation, we get,

$$e_{ms} = \frac{\sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \{f(x, y) - \tilde{f}(x, y)\}^2}{\sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f^2(x, y)} \quad \ldots (9.3.3)$$

A simple re-arrangement of terms gives us the Signal to Noise Ratio (SNR)

$$e_{SNR} \text{ (Mean squared error)} = \frac{\sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f^2(x, y)}{\sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \{f(x, y) - \bar{f}(x, y)\}^2} \quad \ldots (9.3.4)$$

$$e_{SNR} \text{ (Root mean squared error)} = \left[\frac{\sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f^2(x, y)}{\sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \{f(x, y) - \tilde{f}(x, y)\}^2} \right]^{1/2} \quad \ldots (9.3.5)$$

Here, $\{f(x, y) - \tilde{f}(x, y)\}$ is considered as noise introduced in the reconstructed signal. Though the fidelity criteria offers a simple mechanism to evaluate the error (information loss), it cannot distinguish between small error at many pixels and large error at few pixels. This consideration is important because the same value of error may produce a widely different visual effect. Since the compressed images are ultimately viewed by human beings, measuring image quality by subjective evaluations of a human observer is often more appropriate.

MATLAB program to calculate the root-mean squared error.

```
% Calculation of mean squared error, RMS error and SNR %
clear all
clc
g=imread('sat1.tif');          %% Reading the first image %%
g1=imread('sat2.tif');  %% Reading the second image %%
g=double(g);
g1=double(g1);
squared_error=0;
                %% Initialising the mean square error to zero %
rms=0;      %% Initialising root mean square error to zero %
temp=0;             %% Required for SNR %
[row col]=size(g);
for x=1:1:row
      for y=1:1:col
            ms=(g(x,y)-g1(x,y)) ^ 2;
    squared_error=ms+squared_error;
    temp1=g(x,y)*g(x,y);  %% required for SNR formula %%
      temp=temp1+temp; %% required for SNR formula %%
    end
end
```

```
%%Formula for mean squared error%%
mean_squared_error=squared_error/(row*col)
%%Formula for root mean squared error%%
root_mean_error=sqrt(mean_squared_error)
%%Formula for signal to noise ratio%%
snr=temp/squared_error
```

9.3.2 Subjective Error Criteria

The subjective error criteria is performed as follows. The original and the reconstructed images are shown to a large group of people. Each person assigns a grade to the reconstructed image. These grades are purely subjective. Finally, based on grades assigned by the entire group, an overall grade is assigned to the reconstructed image. Compliment of this grade gives an idea of the subjective error.

9.4 Lossless Compression Techniques

Lossless data compression algorithms fall into two broad categories : Dictionary based techniques and Statistical methods. Dictionary based techniques generate a compressed file containing fixed codes, each of which represents a particular sequence of bytes in the original file.

Statistical methods on the other hand implement data compression by representing frequently occurring characters in the file with fewer bits than they do for less commonly occurring ones.

9.4.1 Dictionary Based Coding

In many applications, the image consists of recurring patterns. Most of the background information in images is a set of pixels which have a specific pattern. A very efficient approach to encode these images is to keep a list, or Dictionary, of frequently occurring patterns. When these patterns appear, they are encoded with a reference to the dictionary. What we are actually doing is splitting the image pixels into two classes - frequently occurring patterns and infrequently occurring patterns.

9.4.1(A) Run Length Encoding (RLE)

It is the simplest dictionary-based data compression technique. Image files frequently contain the same character repeated many times in a row. Images, particularly those having very few grey levels, often contain regions of adjacent pixels, all with the same grey level. Each row of such images can have long runs of the same grey value. In, such cases, one can store a code specifying the value of the grey level, followed by the length of the run, rather than storing the same value many times over.

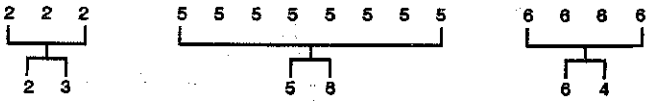
Let us take an example.

Consider the first row of an image which has the following grey values.

2	2	2	5	5	5	5	5	5	5	5	6	6	6	6
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

The first row of the image has 15 grey values.

We could store the above row in a compact manner using RLE.



The first code specifies the grey value, followed by the length of the run.

Hence the RLE of the first row is simply.

2 3 5 8 6 4

As is evident, run length encoding achieves considerable compaction in images which have a fairly constant background. The RLE eliminates *Inter-pixel redundancies*.

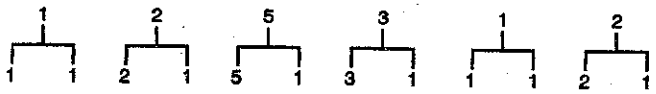
The run length encoding also has a serious problem. At times, the RLE doubles the size of the file.

Consider the following example.

Perform RLE on the following data.

1 2 5 3 1 2

In RLE, the first value specifies the grey value while the second value specifies the run. as is evident, run



∴ The RLE code is

1 1 2 1 5 1 3 1 1 1 2 1

The RLE code in this case is double that of the original sequence. Hence RLE should only be used if we have the same character grey value repeated many times in a row.

9.4.2 Statistical Coding

The images dealt with so far have been coded and saved in the natural code. That is for a 256 grey level image, grey level at f(x, y) is coded with its 8-bits binary equivalent. A grey level of a value 4 is coded as 0000 0100. Similarly for a 8 grey level image, grey value at f (x, y) is coded using 3-bits. The table that shows codes for a sampled image having grey levels varying from 0 to 7 (3-bit image) is given.

These codes are referred to as Equal Length Codes.

Input	Natural code
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Equal length codes do not make use of the statistics of data. The assumption is made that all grey levels have equal occurrence in the image. Since this possibility is unlikely, this is not an optimum for of coding. If we can develop a code such that fewer bits are assigned to grey levels having higher probability of occurrence and vice versa, we could reduce the number of bits required for transmission. Such a coding is known as Variable Length Coding or Entropy Coding and it eliminates *Coding redundancies.*

Here, we are not discarding any information as would be done in lossy compression, but simply represent more frequently occurring values with shorter codes and vice versa.

Consider an image containing L grey levels i = 0, 1 ..., L – 1. Let n_i denote the number of pixels having grey level i.

The probability of occurrence of grey level i in the image is

$$p_i = \frac{n_i}{\sum_{i=0}^{L-1} n_i} \qquad \ldots(9.4.1)$$

Let us define entropy H associated with the grey level as

$$H = -\sum_{i=0}^{L-1} p_i \log_2 p_i \qquad \ldots(9.4.2)$$

Entropy is the measure of randomness in the set of random variables. Entropy is never negative since p_i lies in the range [0,1].

Details of the above equation can be found in any book on Information theory. When all variables are equally likely i.e., $p_0 = p_1 = p_2 ..... = p_{L-1} = 1/L$, then the randomness is maximized and the entropy attains its greatest value.

Ex. 9.4.1

Given a 256 grey level image with equal probability of occurrence for each grey level. Find the maximum word length.

Soln. :

$$H = -\sum_{i=0}^{L-1} p_i \log_2 p_i$$

$$= -\sum_{i=0}^{L-1} \left(\frac{1}{256}\right) \log_2 \left(\frac{1}{256}\right)$$

$$H = 8\text{-bits/pixel}$$

This simply means that an equal-length code can be used on an image that has uniform probability density function.

9.4.3 Huffman Encoding

Huffman encoding is a loss-less statistical method that finds a variable length code with minimum redundancy. This method is named after D.A. Huffman, who developed the code in 1952 while he was a Ph.D. student at M.I.T. University. In Huffman coding, characters that occur most often, are assigned as few as one or two bits while characters whose occurrence is rare are assigned more bits. In other words, characters or symbols with higher probability of occurrence get shorter codes. Coding a stream of data using Huffman encoding is done by forming a Huffman tree.

The procedure for building this tree is simple and elegant.

(1) For a given list of characters (symbols), develop a corresponding list of probabilities or frequency counts so that each characters relative frequency of occurrence is known.

(2) Sort the list of characters according to their frequency of occurrence, with the most frequently occurring characters at the top i.e., place the probabilities in the descending order.

(3) Add the last two probabilities (the smallest two).

(4) Taking the new sum from step 3, once again arrange the probabilities in the descending order.

(5) Continue performing step 3 and step 4 till just one value is left in the probability list.

We shall take up on example to understand these steps better.

Ex. 9.4.2

Assume that we wish to transmit the set of 28 data points.

{1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3 ,3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 6, 6, 7}

Soln. :

We shall follow the steps that are given above.

Step 1 : Develop a list of probabilities.

In this case $\sum n_k = 28$

S_i	n_k	$p_i = \frac{n_k}{\sum n_k}$
1	7	0.25
2	6	0.21
3	5	0.18
4	4	0.14
5	3	0.11
6	2	0.07
7	1	0.04
$\sum n_k =$	28	

Step 2 : Arrange the probabilities in the descending order.

In this case the probabilities are already in the descending order.

S_i	p_i
1	0.25
2	0.21
3	0.18
4	0.14
5	0.11
6	0.07
7	0.04

Step 3 : Adding the last two probabilities.

S_i	P_i	
1	0.25	
2	0.21	
3	0.18	
4	0.14	
5	0.11	
6	0.07	0.11
7	0.04	

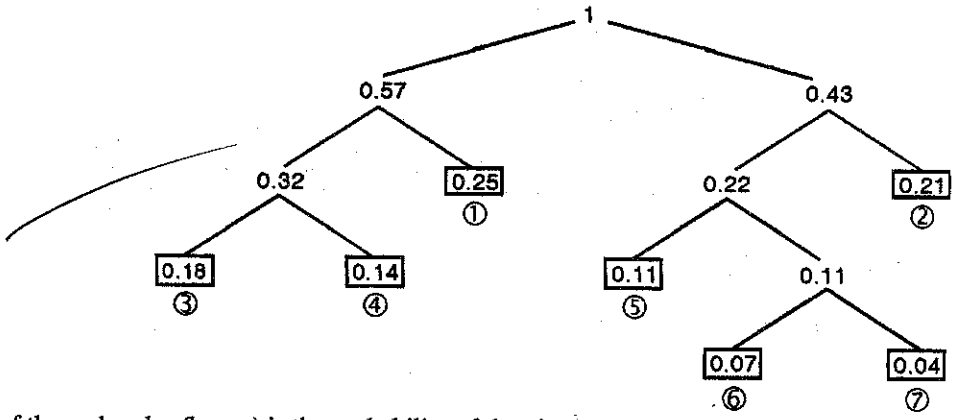
Step 4 : Arranging the new value again in the descending order.

S_i	P_i	
1	0.25	0.25
2	0.21	0.21
3	0.18	0.18
4	0.14	0.14
5	0.11	0.11
6	0.07	0.11
7	0.04	

Repeating step 3 and step 4, we get,

S_i	P_i						
1	0.25	0.25	0.25	0.32	0.43	0.57	1
2	0.21	0.21	0.22	0.25	0.32	0.43	
3	0.18	0.18	0.21	0.22	0.25		
4	0.14	0.14	0.18	0.21			
5	0.11	0.11	0.14				
6	0.07	0.11					
7	0.04						

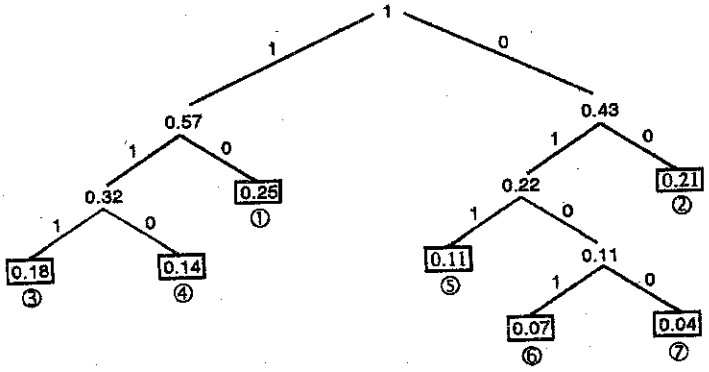
The process of merging these nodes produces a binary tree, we start from the right hand corner.



Each of the end nodes (boxes) is the probability of the given stream of data.

Now start labeling each branch. Let the left branches be assigned value 1 and the right branches be assigned value 0 (It could be the other way also).

Hence we have



Hence to find the code for say, 1 ($p_i = 0.25$), we walk along the path which leads to 1 and collect the bits attached to the path.

1 → 10

2 → 00

3 → 111

4 → 110

5 → 011

6 → 0101

7 → 0100

∴ The final table formed is given as

S_i	3-bit binary code natural code	Huffman code
1	001	10
2	010	00
3	011	111
4	100	110
5	101	011
6	110	0101
7	111	0100

Ex. 9.4.3

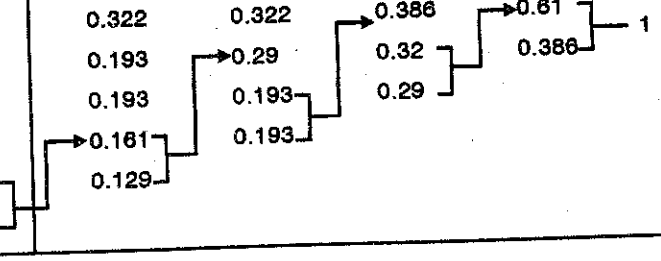
Find the Huffman code for the following stream of data.

{a, a, a, a, a, a, b, b, b, c, c, c, c, c, c, d, d, d, d, d, d, d, d, d, e, e, e, e, f, f}

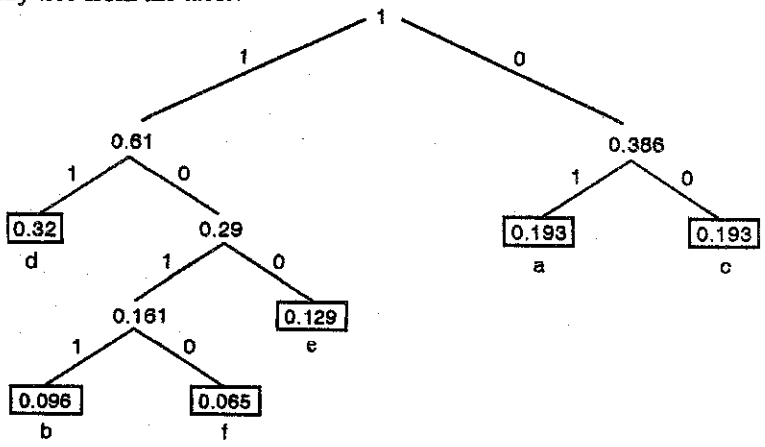
Soln. :

The total number of characters to be transmitted is 31. We shall form a table which implements all the steps.

S_i Symbols	p_i Probability	Descending Order
a	0.193	0.323
b	0.096	0.193
c	0.193	0.193
d	0.323	0.129
e	0.129	0.096
f	0.065	0.065



We now form a binary tree from the table.



We denote the left side branches as 1 and right side branches as 0 (You could also reverse this).

Hence to find the code for each letter, we move through their respective branches.

a → 01

b → 1011

c → 00

d → 11

e → 100

f → 1010

How does this kind of coding achieve compression ?

The average word length for the above example is

$$\text{Length (average)} = \sum (\text{Number of bits for each symbol}) \times (\text{Its probability})$$

$$\text{Length (average)} = 2(p(a)) + 4(p(b)) + 2(p(c)) + 2(p(d)) + 3(p(e)) + 4(p(f))$$

$$= 2(0.193) + 4(0.096) + 2(0.193) + 2(0.323) + 3(0.129) + 4(0.065)$$

$$= 2.449\text{-bits}$$

At this point, we should understand how data is transmitted across transmission channels. Each character (symbol) is represented as a eight bit group (byte) and transmitted over the channel.

Hence if we have to transmit

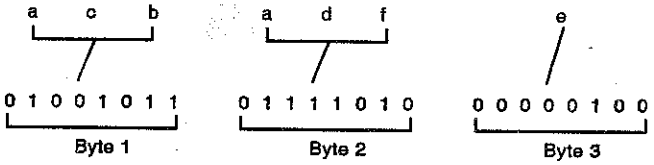
a c b a d f e

We would required 8-bits (1 byte) for each symbol. Hence 7 bytes need to be transmitted. Now with Huffman coding, things get a little different

a c b a d f e

01 00 1011 01 11 1010 100

Symbols have variable lengths depending on their probability of occurrence. While transmission, some symbols can be combined to form 8-bits (1 byte)



This is compression !!

Earlier we had to send 7 bytes but now only 3 bytes are required. The way the codes are formed ensures that no ambiguity exists while separation. To summarise, Huffman coding is one of the statistical methods used to compress data (Loss-less). In this, the frequently occurring characters in a file are represented with fewer bits than the less commonly occurring characters. This is the approach Samuel F.B. Morse used when he defined the Telegraph Code. The Morse code is made up of dots and dashes. The most often used letter in English is *e*. Hence the Morse code of e is represented by single dot (.), whereas a much less common character *z* is coded as dash dash dot dot (– – . .).

Is the Huffman coding technique optimal ?

The answer is No. We have already seen that entropy of the image is

$$H = -\sum_{i=0}^{L-1} p_i \log_2 p_i$$

Information theory establishes this as a theoretical limit to the number of bits per pixel needed to represent each grey level. Let us study the last problem on Huffman coding. According to this technique, the average length for each grey level reduced from 3-bits to 2.449-bits. We shall now check if this is the optimum answer using the formula of entropy.

a	0.193	$-0.193 \log_2 (0.193) = 0.458$
b	0.096	$-0.096 \log_2 (0.096) = 0.324$
c	0.193	$-0.193 \log_2 (0.193) = 0.458$
d	0.323	$-0.323 \log_2 (0.323) = 0.526$
e	0.129	$-0.129 \log_2 (0.129) = 0.381$
f	0.065	$-0.065 \log_2 (0.065) = 0.256$
		2.403

Hence the optimum value to which compression can take place in this example 2.403. What we get through Huffman coding is 2.449.

Ex. 9.4.4

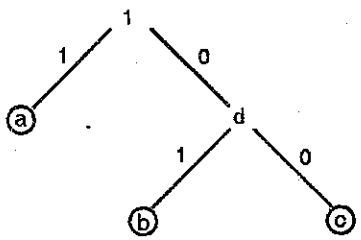
How many unique Huffman codes are there for a three symbol source ? Construct these codes.

Soln. :

Let the three symbols be A, B and C and let their probabilities be a, b and c. As stated earlier, we arrange them in the descending order and add up the last two probabilities. Let the new probability be 'd'. Let d < a

∴ Constructing the table, we get

Symbol	Probability	
A	a	a → 1
B	b	d →
C	c	



A → a → 1

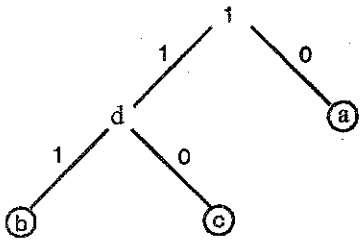
B → b → 0 1

C → c → 0 0

These are the three Huffman code for A, B and C.

If we assume that d > a, i.e.

we get,



A → a → 0

B → b → 1 1

C → c → 1 0

9.5 Arithmetic Coding

In the previous section we studied one approach-Huffman Encoding to generate variable length codes. We also noted that the Huffman codes do not yield the optimum result. Another method of generating variable length codes is the Arithmetic coding. If we recall, in Huffman coding, there is a one to one correspondence between the source symbol and the code word. The table of 3 bit binary natural code and the Huffman code generated from the first example of Huffman codes is reproduced to understand this one to one correspondence better.

S_i	3 - bit binary code natural code	Huffman code
1	001	10
2	010	00
3	011	111
4	100	110
5	101	011
6	110	0101
7	111	0100

Here every source symbol (1, 2, ...7) has a unique code (10, 00, ...0100).

It has been seen that it is more efficient to generate code words for groups or sequences rather than generating code words for each symbol. In Arithmetic coding, this one-to-one correspondence does not exist. In this technique, a sequence of source symbols is assigned a single arithmetic code word. A unique tag is generated for the sequence to be encoded. We shall now proceed to explain how this tag is generated.

Let us assume symbols from a source $S = \{ c_1, c_2.....c_n\}$. Let their probabilities be $P = \{p_1, p_2, ... p_n\}$ where p_i corresponds to the probability of occurrence of c_i. In arithmetic coding, a sequence of symbols is encoded as a number from the interval[0, 1].

The interval [0, 1] is split up into the various probabilities.

i.e. Interval $= \{ [0, p_1], [p_1, p_1 + p_2], [p_1 + p_2, p_1 + p_2 + p_3] ..... + p_n\}$

It can also be written in terms of Cumulative Density Functions (CDF).

i.e. Interval $= \{ [0, P_1], [P_1, P_2] ..... [P_{n-1}, 1]\}$

where P_i is the CDF at position i,

In course of arithmetic coding, the interval also determines the proportional division of any interval [L, U] contained in [0, 1] where L and U are the lower and upper limits. This generalisation is given by the formula

$$\text{Interval} = \{ [L, L + (U - L)P_1], [L + (U - L)P_1, L + (U - L)P_2],$$

$$[L + (U - L)P_2, L + (U - L)P_3], ... , [L + (U - L)P_{n-1}, L + (U - L) P_n] \} \quad ...(9.5.1)$$

Here P_i is the CDF at position i, If these statements are not clear, don't worry. Things will get much easier as we proceed. As stated earlier, a sequence of symbols are encoded as a single code. The probabilities of all the source symbols $\{p_1, p_2, ..., p_n\}$ are arranged within the interval [0, 1].

Based on the first symbol, say p_1, the probability range of p_1 is blown up to range of [0, 1] and the other probabilities are scaled accordingly using the formula (9.5.1). This procedure is carried out recursively till we reach the end of the symbols. Let us take an example.

Ex. 9.5.1

Consider a five symbol sequence {A, B, B, C, C} from a three symbol source code. Generate the Arithmetic code for the same.

Soln. :

The first step is to list down the probabilities of the three symbol source code. The main interval [0, 1] is spit up into initial subintervals based on the original probabilities.

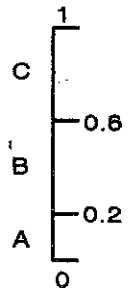
Source symbol	Probability	Initial subinterval
A	0.2	[0, 0.2]
B	0.4	[0.2, 0.6]
C	0.4	[0.6, 1]

Note : The initial subintervals are similar to the cumulative density function.

Here $P_1 = 0.2$, $P_2 = 0.6$.

The initial subintervals are generated from the formula given in Equation (9.5.1).

These are as shown.



The sequence that needs to be coded is A, B, B, C, C.

The first symbol is A hence the subinterval [0, 0.2] is blown up to the size of interval [0, 1] and the original probabilities are rescaled using Equation (9.5.1).

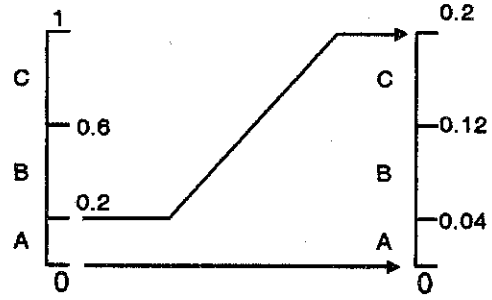


Fig. P. 9.5.1

This rescaling is shown in Fig. P. 9.5.1. The new subintervals are generated using the formula (9.5.1).

$$= [L, L + (U - L) P_1], [L + (U - L) P_1, L + (U - L) P_2]\}$$

In this case L = 0, U = 0.2

$$= \{[0, 0 + (0.2 - 0) P_1], [0 + (0.2 - 0) P_1, 0 + (0.2 - 0) P_2]\}$$

$$= \{(0,(0.2)\ (0.2)], [(0.2)\ (0.2), (0.2)\ (0.6)]\}$$

$$= [0, 0.04], [0.04, 0.12]$$

Hence the subintervals are 0 - 0.04 - 0.12 - 1. Make sure you have followed the things just discussed. This is shown in Fig. P. 9.5.1.

The next source symbol in the message is B. We now blow up the range corresponding to the position B i.e. [0.04, 0.12]. This is shown in Fig. P. 9.5.1(a) along with earlier step.

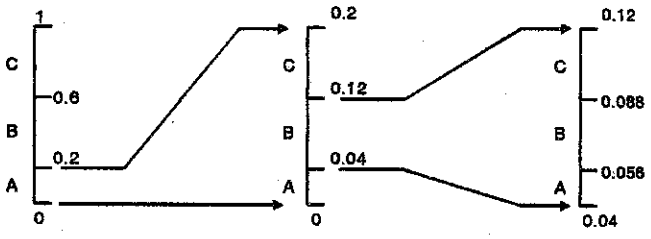


Fig. P. 9.5.1(a)

The new subintervals are generated using the same formula.

i.e. $\{[L, L + (U - L) P_1], [L + (U - L) P_1, L + (U - L) P_2]\}$

Here L = 0.04 and U = 0.12

$$[0.04, 0.04 + (0.12 - 0.04) P_1], [0.04 + (0.12 - 0.04) P_1, 0.04 + (0.12 - 0.04) P_2]$$

$$= [0.04, 0.056], [0.056, 0.088]$$

Hence the subintervals are 0.04 - 0.05 - 0.088 - 0.12.

These subintervals are shown in Fig. P. 9.5.1(a).

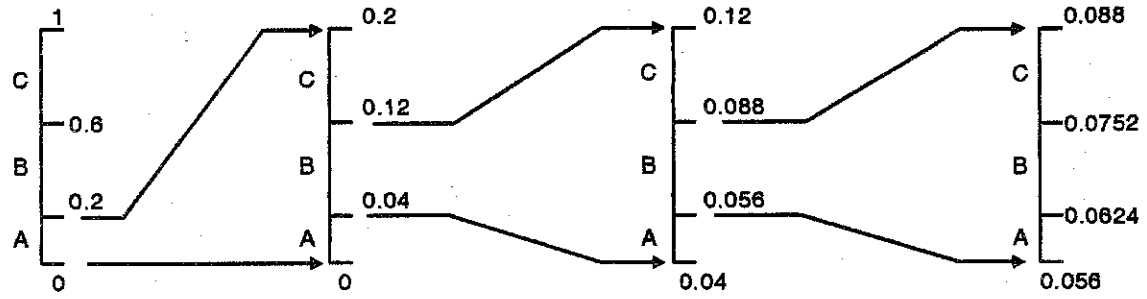


Fig. P. 9.5.1(b)

The next symbol in the message is again B. we now blow up the section corresponding to B. i.e. the range [0.056, 0.088]. This is shown in Fig. P. 9.5.1(b) along with the previous steps. The new subintervals are generated using the same formula,

$$\{ [L, L + (U - L) P_1], [L + (U - L) P_1, L + (U - L) P_2]\}$$

Here $L = 0.056$ and $U = 0.088$

$$\{[0.056, 0.056 + (0.088 - 0.056) P_1],$$

$$[0.056 + (0.088 - 0.056) P_1, 0.056 + (0.088 - 0.056) P_2]\}$$

$$= [0.056, 0.0624], [0.0624, 0.0752]$$

Hence the subintervals are 0.056 - 0.0624 - 0.0752 - 0.088

These are shown in Fig. P. 9.5.1(b). The next symbol in the message is C. We now blow up the section corresponding to C. i.e. the range [0.0752, 0.088]. This is shown in Fig. P. 9.5.1(c) along with the previous steps.

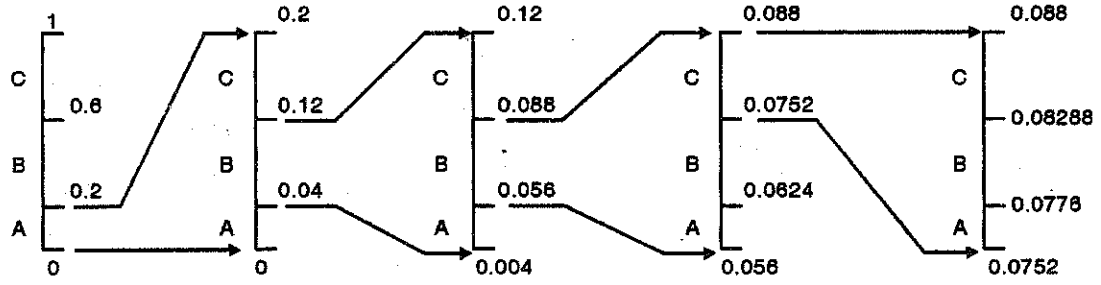


Fig. P. 9.5.1(c)

The new subintervals are generated using the Equation (9.5.1).

$$[L, L + (U - L) P_1], [L + (U - L) P_1, L + (U - L) P_2]$$

Here $L = 0.0752$ and $U = 0.088$

$$\{[0.0752, 0.0752 + (0.088 - 0.0752) P_1],$$

$$[0.0752 + (0.088 - 0.0752) P_1, 0.0752 + (0.088 - 0.0752) P_2]\}$$

$$= [0.0752, 0.07776], [0.07776, 0.08288]$$

Hence the new subintervals are 0.0752 - 0.0776 - 0.08288 - 0.088

These subintervals are shown in Fig. P. 9.5.1(d).

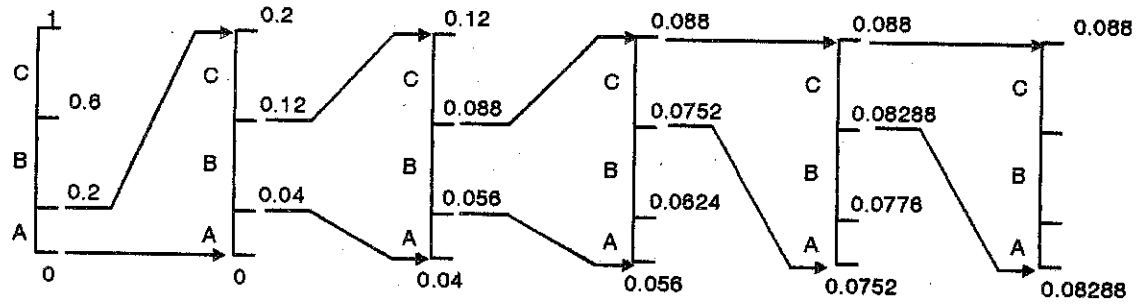


Fig. P. 9.5.1(d)

The last symbol in the message is again C. We now blow up the segment corresponding to C. i.e. the range [0.08288, 0.088]. We draw this as shown in Fig. P. 9.5.1(d) along with the previous steps. The entire message has been broken up into subintervals. Since the last symbol is C, we could take any number within the range [0.08288, 0.088] and use it as a *Tag* to code the entire sequence. Hence the tag for ABBCC could be either 0.08288 which is the lower limit, or it could be 0.088 which is the upper limit or any number from that range.

Let us take the lower limit i.e. 0.08288.

Decoding : Decoding consists of determining the subinterval of the tag and outputing the letter corresponding to the interval.

Refer to the Fig. P. 9.5.1(d) we start with the tag (codeword), in this case 0.08288. This tag causes the first subinterval to be chosen [0, 0.2]. Since this subinterval corresponds to the symbol A, the symbol A is output. The interval is now set to [0, 0.2]. In this loop it is noted that the tag 0.08288 belongs to the second subinterval of [0, 0.2] which is the interval [0.04, 0.12]. This interval corresponds to B and hence the symbol B is output. The interval is now set to [0.04, 0.12]. This procedure is continued to give us the original stream of data which is A B B C C.

These steps are summarized in the table given below.

Interval	Subinterval			Symbol
[0, 1]	[0, 0.2]	[0.2, 0.6]	[0.6, 1]	A
[0, 0.2]	[0, 0.04]	[0.04, 0.12]	[0.12, 0.2]	B
[0.04, 0.12]	[0.04, 0.056]	[0.056, 0.088]	[0.088, 0.12]	B
[0.056, 0.088]	[0.056, 0.0624]	[0.0624, 0.0752]	[0.0752, 0.088]	C
[0.0752, 0.088]	[0.0752, 0.0776]	[0.0776, 0.08288]	[0.08288, 0.088]	C

Let us solve one more example to give ourselves a little more of confidence.

Ex. 9.5.2

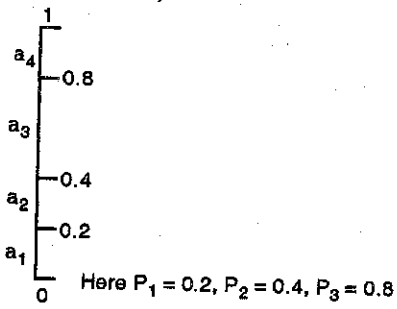
Given a four symbol source. Perform the Arithmetic coding for the signal $\{a_1, a_2, a_3, a_3, a_4\}$.

Soln. :

We list down the signal in terms of the probability of each symbol and their subintervals (cdf's)

Source symbol	Probability	Initial subinterval
a_1	0.2	[0, 0.2]
a_2	0.2	[0.2, 0.4]
a_3	0.4	[0.4, 0.8]
a_4	0.2	[0.8, 1]

This is shown below,



The first symbol of the message is a_1. We blow up the range of a_1 i.e. [0, 0.2] and create new subintervals using Equation (9.5.1).

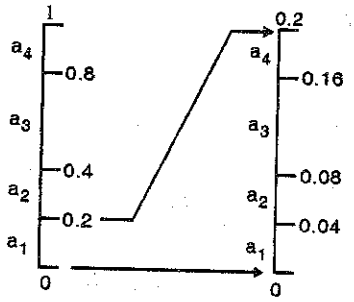


Fig. P. 9.5.2

Here L = 0 and U = 0.2

Subintervals = $\{[L, L + (U - L) P_1], [L + (U - L)P_1, L + (U - L)P_2], [L + (U - L) P_2, L + (U - L) P_3]\}$

= {[0, 0.04], [0.04, 0.08], [0.08, 0.16]}

Hence the new subintervals are 0 - 0.04 - 0.08 - 0.16 - 0.2. These subintervals are shown in Fig. P. 9.5.2.

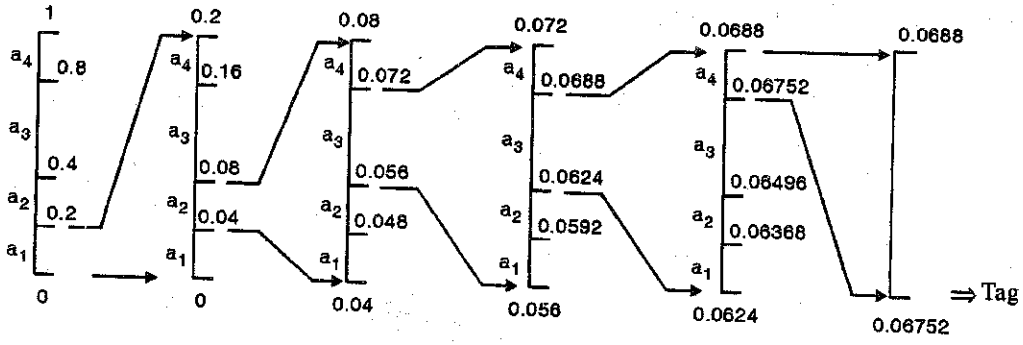


Fig. P. 9.5.2(a)

The next symbol in the message is a_2. Hence we blow up the range corresponding to a_2 i.e. [0.04 to 0.08]. We get the new subintervals. In a similar manner we blow up intervals of a_3, a_3 and a_4 and compute their subintervals. The final figure is shown below. Please try this out yourself.

Let the lower limit value i.e. 0.06752 be the Tag (codeword) which represents the message $\{a_1, a_2, a_3, a_4\}$. Decoding is the reverse operation. We start with the Tag. This causes the first subinterval [0, 0.2] to be chosen. This interval outputs a_1. The range is now set to [0, 0.2]. The tag causes the second range of [0, 0.2] which is [0.04, 0.08], to be chosen. This outputs a_2.

The entire decoding procedure is summarized in the table given below,

Interval	Subinterval				Symbol
[0, 1]	[0, 0.2]	[0.2, 0.4]	[0.4, 0.8]	[0.8, 1]	a_1
[0, 0.2]	[0, 0.04]	[0.04, 0.8]	[0.08, 0.16]	[0.16, 0.2]	a_2
[0.04, 0.08]	[0.04, 0.048]	[0.048, 0.056]	[0.056, 0.072]	[0.072, 0.08]	a_3
[0.056, 0.072]	[0.056, 0.0592]	[0.0592, 0.0624]	[0.0624, 0.0688]	[0.0688, 0.72]	a_4
[0.0624, 0.0688]	[0.0624, 0.06368]	[0.06368, 0.06496]	[0.06496, 0.06752]	[0.06752, 0.0688]	a_4

The Arithmetic coding explained in this chapter is in its very basic form. A lot of modifications have taken place over the years. Practical implementation of Arithmetic coding is explained in a very lucid manner in the book "Introduction of Data Compression" key Khalid Sayood - Elsevier Publications. Students could refer to the book for an indepth analysis of the topic.

9.6 Shannon-Fano Coding

This, like Huffman coding, is a lossless coding technique with variable length codes.

The steps involved in Shannon-Fano coding are given below :

1. The probability of the symbols are arranged in the descending order.
2. The probabilities are partitioned into two groups such that sum of the probabilities in the two groups are nearly equal.
3. The upper group is given a code 0 and the lower group is given a code 1.
4. Steps 2 and 3 are repeated till we have only one element.

We shall now discuss how to perform Step 2 and 3.

Consider a few probabilities which are arranged in the descending order.

0.6

0.2

0.1

0.1

We now need to partition this into two groups such that sum of the probabilities are as equal as possible. Let us consider two cases :

Case 1

Partition 1 → [0.6] → 0.6

Partition 2 → [0.2, 0.1, 0.1] → 0.2 + 0.1 + 0.1 = 0.4

The difference between the two partitions = 0.6 – 0.4 = 0.2

Case 2

Partition 1 → [0.6, 0.2] → 0.6 + 0.2 = 0.8

Partition 2 → [0.1, 0.1] → 0.1 + 0.1 = 0.2

The difference between the two partitions = 0.8 – 0.2 = 0.6

Since Case 1 gives us two blocks which are closer. We choose the first partition.

We assign 0 to the upper block and 1 to the lower block.

Step 1

0.6	0
0.2	1
0.1	1
0.1	1

We now partition the lower block in a similar manner.

	0.6	
$0.2 \rightarrow$	0.2	0
$0.1 + 0.1 = 0.2 \rightarrow$	0.1	1
	0.1	1

We once again assign 0 to the upper block and 1 to the lower block.

Hence we have

	Step 1	Step 2
0.6	0	-
0.2	1	0
0.1	1	1
0.1	1	1

We now partition the lower block and assign 0 and 1.

Hence we have

	Step 1	Step 2	Step 3	Code
0.6	0			0
0.2	1	0		10
0.1	1	1	0	110
0.1	1	1	1	111

Let us now take an example

Ex. 9.6.1

Given a word INDIA, construct the Shannon Fano code and compute the efficiency.

Soln. :

We compute the probabilities and arrange in the descending order.

Symbol	No. of occurrence	Probability
I	2	0.4
N	1	0.2
D	1	0.2
A	1	0.2

We now partition the probabilities and assign 0 to upper block and 1 to lower block. Step 1 is the first partition, Step 2 is the second partition and Step 3 is the third partition.

Symbol	Probability	Step 1	Step 2	Step 3	Code
I	0.4	0	-		0
N	0.2	1	0	-	10
D	0.2	1	1	0	110
A	0.2	1	1	1	111

We know,

$$\text{Efficiency} = \frac{\text{Entropy}}{\text{Average code length}}$$

$$\text{Entropy} = H = -\sum_{i=0}^{L-1} p_i \log_2 p_i$$

	I	0.4	$0.4 \log_2 (0.4) = -0.528$
	N	0.2	$0.2 \log_2 (0.2) = -0.464$
$\therefore$	D	0.2	$0.2 \log_2 (0.2) = -0.464$
	A	0.2	$0.2 \log_2 (0.2) = -0.464$
			$\therefore \Sigma = -1.92$

$$\therefore H = -\Sigma p_i \log_2 p_i$$

$\therefore H = -(-1.92)$

$\therefore H = +1.92$

Now, Average code length = Σ (Number of bits for each symbol) × (Its probability)

$\therefore$ Average code length $= 1 \times p(I) + 2 \times p(N) + 3 \times p(D) + 3\,p(A)$

$= (1 \times 0.4) + (2 \times 0.2) + (3 \times 0.2) + (3 \times 0.2)$

$\therefore$ Average code length $= 2$

$$\therefore \text{Efficiency} = \frac{\text{Entropy}}{\text{Average code length}}$$

$$\therefore \text{Efficiency} = \frac{1.92}{2} = 0.96 = 96\%$$

9.7 Lossy Compression

9.7.1 Improved Grey Scale (IGS) Quantization

We have studied in chapter of Introduction that brightness of a region as perceived by the eye does not depend on the absolute grey level values (remember Mach Bands?). This is because the human eye does not respond with equal sensitivity to all visual information. Some information is visually more important than the others.

Information which is not visually important is called *Psychovisual Redundancy.* Psychovisual redundancies exist in all images and can be eliminated without hampering the subjective quality of the image. We have seen that simply reducing the quantization (number of bits for representation), compresses the image but also produces false contouring. I.G.S. coding reduces the quantization but also reduces false contouring.

Steps involved in I.G.S. coding are,

(1) Lower 4-bits of the preceding modified pixel are added to the present pixel.

(2) The new 4 MSB's of the present pixel are taken as the I.G.S. code.

(3) Repeat step 1 and 2 after moving on to a new pixel.

(4) If MSB is 1111, then add 0000 instead of the 4 LSB's of the previous sum.

These steps will be clear with a simple example.

Ex. 9.7.1

Construct the I.G.S. code of the given grey level data set {100, 110, 124, 124, 130, 110, 200, 210}

Soln. :

From the grey level values it is clear that we would required 8-bits for their representation. Let i be the first pixel. We start by adding 0000 to the 1st pixel and then follow the steps given :

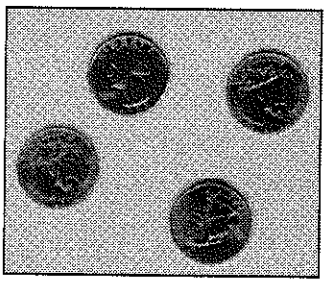
		Grey level		SUM		IGS Code
i - 1				0000	0000	–
i	100	0110	0100	0110	0100	0110
i + 1	110	0110	1110	0111	0010	0111
i + 2	124	0111	1100	0111	1110	0111
i + 3	124	0111	1100	1000	1010	1000
i + 4	130	1000	0010	1000	1100	1000
i + 5	110	0110	1110	0111	1010	0111
i + 6	200	1100	1000	1101	0010	1101
i + 7	210	1101	0010	1101	0100	1101

This type of coding reduces false contouring which is a pitfall of uniform quantization. It also eliminates psychovisual redundancies.

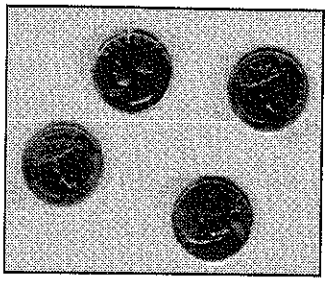
Note IGS code is a lossy form of compression as visual information is lost.

```
%% Program for I.G.S. coding %%
clear all
clc
a=imread('coins.tif');
a=double(a);
[row col]=size(a);
temp=dec2bin(0,8);
for x=1:1:row-1
    for y=1:1:col-1
```

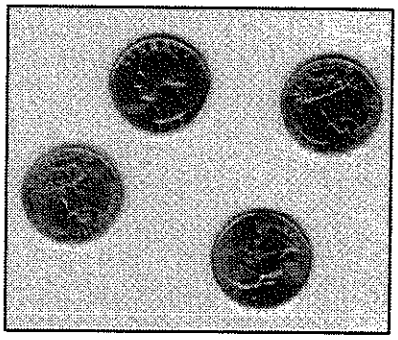
```
%%%%%%%%%%%% This is the uniform quantization loop %%%%%%%%%%
            q=dec2bin(a(x,y),8);
      q1=[q(1) q(2) q(3) q(4)];
      a1(x,y)=bin2dec(q1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The loop ends here %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
This is the IGS loop
%%%%%%%%%%%%%%%%%%%%%%%%%%
if a(x,y) >=240   %% NUMBERS > = 240 HAVE 4 MSBS AS 1
      temp2=0;
else
      temp1=[temp(5) temp(6) temp(7) temp(8)];
      temp2=bin2dec(temp1);
end
      c=a(x+1, y+1)+temp2;
      c1=dec2bin(c,8);
temp=c1;
igs=[c1(1) c1(2) c1(3) c1(4)];
code(x,y)=bin2dec(igs);
      end
end
figure (1)
imagesc((a))
colormap(gray)
figure (2)
imagesc(a1)
colormap(gray)
figure (3)
imagesc(code)
colormap(gray)
```



(a) Original image



(b) Image using uniform quantization, quantized to 4-bits



(c) Image using I.G.S. coding 4-bits

Fig. 9.7.1

Ex. 9.7.2

Compute the e_{rms} that is introduced in the above example. Also calculate the Signal to Noise Ratio (SNR).

Soln. : The IGS codes generated are given :

f (x, y)	IGS code
100	0110
110	0111
124	0111
124	1000
130	1000
110	0111
200	1101
210	1101

We convert the IGS code into its decimal equivalent. The original data was 8 bits. During IGS coding, we select only the 4 MSB values of the modified 8 bits.

Hence while converting them to decimal, we should consider their binary to decimal position weights which are $2^7\ 2^6\ 2^5\ 2^4$

$\therefore$ 0110 will be

2^7	2^6	2^5	2^4
0	1	1	0

$\therefore$ The decimal value of 0110 will be

$$0110 \xrightarrow{\text{decimal}} [(0\times 2^7)+(1\times 2^6)+(1\times 2^5)+(0\times 2^4)]$$

$$\therefore\ 0110 \xrightarrow{\text{decimal}} 92$$

f (x, y)	IGS code	Decimal equivalent $\tilde{f}$ (x, y)
100	0110	92
110	0111	112
124	0111	112
124	1000	128
130	1000	128
110	0111	112
200	1101	208
210	1101	208

e_{rms} is given by equation

$$e_{rms} = \left[\frac{1}{MN}\sum\sum\left\{ f(x,y) - \tilde{f}(x,y)\right\}^2\right]^{1/2}$$

$$= \left[\frac{1}{8}\{(100-92)^2+(110-112)^2+(124-112)^2+(124-128)^2 + (130-128)^2+(110-112)^2+(200-208)^2+(210-208)^2\}\right]^{1/2}$$

$$= \left[\frac{1}{8}\{8^2+2^2+12^2+4^2+2^2+2^2+8^2+2^2\}\right]^{1/2}$$

$$= \left[\frac{1}{8}\{64+4+144+16+4+4+64+4\}\right]^{1/2}$$

$$e_{rms} = 6.16$$

The signal to noise ratio is given by equation

$$e_{SNR} = \left[\frac{\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f^2(x,y)}{\sum_{x=0}^{N-1}\sum_{y=0}^{M-1}\{f(x,y)-\tilde{f}(x,y)\}^2}\right]^{1/2}$$

$$= \left[\frac{(100)^2+(110)^2+(124)^2+(124)^2+(130)^2+(110)^2+(200)^2+(210)^2}{304}\right]^{1/2}$$

$$e_{SNR} = 23.36$$

9.7.2 Transform Coding (JPEG Coding)

Many methods of lossy compression have been developed; however a family of techniques called Transform Coding have proven to be the most valuable.

What exactly is transform coding ?

Suppose we have an ensemble of images that need to be encoded into a compact data representation. We transform these images using some transformation, discard those pixel coefficients that are nearly zero and quantize pixel coefficients which are small there by concentrating on only those coefficients which contain the most information about the image. When these images are reconstructed using an inverse transformation, we get compressed images in which little important content would have been lost. This approach is called Transform Coding. In other words transform coding is based on a simple premise. When an image is passed through the Fourier (or any other) transform, the resulting data values will no longer be equal in their information carrying roles. It is known that the low frequency components of an image are more important than the high frequency components. Removing 40% - 50% of the bits from the high frequency components might remove, say only 5% of the encoded information.

Many different transforms have been investigated for data compression. The Fourier transform is easy to use, but does not provide adequate compression. Of all the transforms studied the Discrete Cosine Transform (DCT) comes out the winner for the amount of compression it can achieve without deteriorating the subjective quality of the image.

The DCT has better compactness than the DFT i.e., a large amount of information is stored in relatively lesser number of bits as compared to the DFT.

Let us take an example. Consider the signal shown in Fig. 9.7.2.

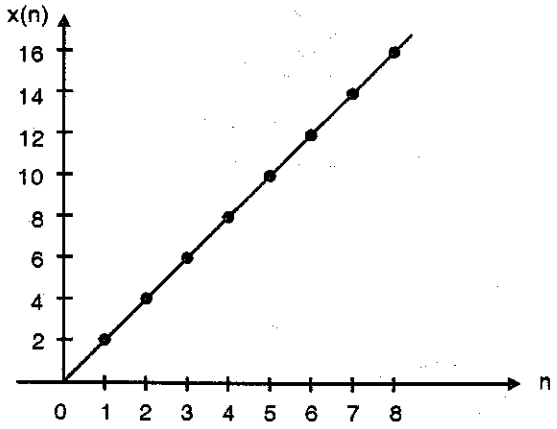


Fig. 9.7.2

We shall then repeat the entire exercise using the DCT.

X = (2, 4, 6, 8, 10, 12, 14, 16)	X = (2, 4, 6, 8, 10, 12, 14, 16)
DFT(x) = (72, – 8 + 19j, – 8 + 8j, – 8 + 3j, –8, – 8, – 3j, – 8 – 8j, 8 – 19j)	DCT (x) = (25.45, – 12.8, 0, –1.34, 0, – 0.4, 0, – 0.1)
Truncated DFT (Taking only first 5 values) = (72 , – 8 + 19j, – 8 + 8j, –8 + 3j, – 8, 0, 0, 0)	Truncated DCT (Taking only first 5 values) = (25.45 – 12.80, 0, – 1.34, 0, 0, 0,0)
IDFT + Absolute + Rounding = (7, 6, 8, 8, 10, 10, 12, 13)	IDCT = (2, 4, 6, 8, 10, 12, 14, 16)

What is illustrated above is that the first five bits of the DCT represents the original signal in a far better way as compared to the DFT. This is what we mean when we say that the DCT has better signal (energy) compactness in comparison with the DFT.

9.7.3 Joint Photographic Experts Group (JPEG)

The best example of transform coding is the JPEG (Joint Photographic Experts Group) standard for compression which uses the Discrete Cosine Transform. We shall now discuss the JPEG algorithm. The JPEG compression algorithm operates in three successive steps shown in Fig. 9.7.3 :

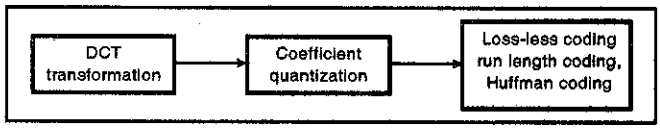


Fig. 9.7.3

These three steps combine to form a powerful compressor, capable of compressing images to less than 10 percent of their original size while losing little, if any, of their original fidelity. Consider the image shown below which has grey level values between 0 and 255 (8 bits). The JPEG compression starts by breaking the image into 8 × 8 pixel groups.

231	224	224	217	217	203	189	196
210	217	203	189	203	224	217	224
196	217	210	224	203	203	196	189
210	203	196	203	182	203	182	189
203	224	203	217	196	175	154	140
182	189	168	161	154	126	119	112
175	154	126	105	140	105	119	84
154	98	105	98	105	63	112	84

154	154	175	182	189	168	217	175
154	147	168	154	168	168	196	175
175	154	203	175	189	182	196	182
175	168	168	168	140	175	168	203
133	168	154	196	175	189	203	154
168	161	161	168	154	154	189	189
147	161	175	182	189	175	217	175
175	175	203	175	189	175	175	182

42	28	35	28	42	49	35	42
49	49	35	28	35	35	35	42
42	21	21	28	42	35	42	28
21	25	25	42	42	28	28	14
56	70	77	84	91	28	28	21
70	126	133	147	161	91	35	14
126	203	189	182	175	175	35	21
49	189	245	210	182	84	22	35

Fig. 9.7.4

These 8 × 8 pixels are treated independently during compression. Each group has 64 bytes (8 × 8 bits) which after compression, would be represented by, say, 2 to 20 bytes. In other words each 8 × 8 group is compressed using transform coding. During decompression, the inverse transform (Inverse DCT) is taken of these 2 to 20 bytes to create an *approximation* of the original group. These groups are now fitted together to get the final image.

The natural question that arises is why take a group of 8 × 8 and not say 16 × 16 ? Would there be a problem ?

The answer is No. We could very well use a 16 × 16 group. But at the time the JPEG standard was developed, 8 × 8 grouping was the maximum size that the integrated circuit technology could handle !! Hence 8 × 8 grouping was standardized. This grouping may not be changed in the future.

Each of the 8 × 8 groups is passed through a DCT program. The DCT of a 8 ×8 matrix results in a 8 × 8 spectrum. In other words, the original 64 numbers are changed to a new set of 64 numbers. All these new values are *real.* In this new group of 64 values, the low frequencies reside in the upper-left corner while the high frequencies are in the lower right corner. The DC component is at (0, 0), the upper-left most value.

Consider the following example.

44	74	13	3	61	72	70	27
50	27	21	31	2	69	73	25
21	44	61	1	2	8	48	87
64	93	63	38	19	45	55	23
32	21	37	68	59	44	12	80
96	84	58	9	6	35	45	91
73	68	45	4	37	15	72	23
41	63	4	61	63	68	89	24

DCT

DC LF

358	2	74	18	–41	–5	–23	–2
–25	–18	0	2	–26	–6	15	–8
–3	–27	–17	75	–26	14	–44	–7
10	5	16	18	26	–17	26	7
37	4	–62	30	1	–32	–40	–32
2	44	28	0	–36	–16	–20	38
–4	–31	26	–26	19	–10	28	14
–9	–14	–14	–34	18	–48	–18	7

HF

Fig. 9.7.5

Note that the DC coefficient is almost an order of magnitude greater than any other values in the DCT output matrix. In addition to this, there is a general trend in the DCT output matrix. As the elements move further down from the DC coefficient, they become lower and lower in magnitude.

```
%% Computing the DCT of 8 × 8 Blocks of an Image %%
clear all
clc
I1=imread('mansi.bmp');
I=rgb2gray(I1); %% Only if it is a colour image

fun=@dct2;
J=blkproc(I,[8 8],fun);

figure(1)
imshow(I)

figure(2)
imshow(uint8(J))
```

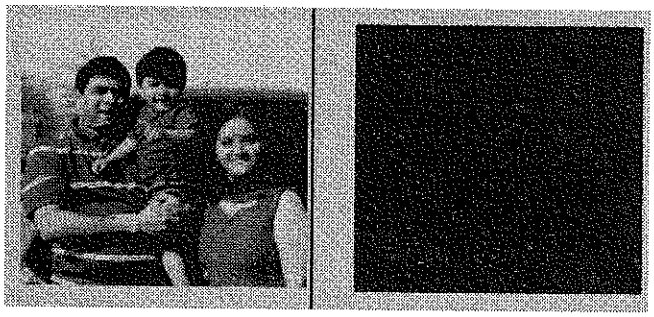


Fig. 9.7.6

This means that by performing the DCT on the input data, we have concentrated the representation of the image in the upper left coefficients of the output matrix with the lower right coefficients of the DCT matrix containing less useful information.

We started with a 8 × 8 matrix and got a new 8 × 8 matrix with a whole new set of values. Hence the DCT transformation by itself does not perform compression but produces a matrix which facilitates compression. DCT is a loss-less transformation.

We have shown in Fig. 9.7.3 that the JPEG compression is a three step procedure, the first being the DCT transformation.

As just stated, the DCT is a loss-less transformation that does not actually perform compression. It is a preparation stage for "lossy" or Coefficient Quantization stage.

Quantization is simply the process of reducing the number of bits needed to store an integer value by reducing the precession of the integer. Once a DCT image has been created, we generally reduce the precession of the coefficients more and more as we move away from the DC coefficient. Small values are made equal to zero.

The JPEG algorithm implements quantization using a quantization matrix. For each element of the DCT matrix, a separate value in the quantization matrix gives us a quantum value.

Two typical examples of quantization matrices for low compression and high compression are shown in Fig. 9.7.7.

1	1	1	1	1	2	2	4
1	1	1	1	1	2	2	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

Low compression

1	2	4	8	16	32	64	128
2	4	4	8	16	32	64	128
4	4	8	16	32	64	128	128
8	8	16	32	64	128	128	256
16	16	32	64	128	128	256	256
32	32	64	128	128	256	256	256
64	64	128	128	256	256	256	256
128	128	128	256	256	256	256	256

High compression

Fig. 9.7.7

The actual formula for quantization is quite simple

$$\text{Quantized value (i, j)} = \left\lceil \frac{\text{DCT (i, j)}}{\text{Quantization matrix (i, j)}} \right\rceil \text{ rounded to nearest integer} \qquad \ldots(9.7.1)$$

From the formula it is clear that the quantized values will be virtually zero as we move to higher values of (i, j).

Since the first element of the quantization matrix is 1, the first quantized value remains the same (DC component is not changed). Virtually all higher-frequency components will be rounded off to zero. Only if the high-frequency coefficients get up to unusually large values will they be encoded as non-zero.

Let us take an example

Given an 8 × 8 image, perform the DCT transformation and get quantized values using the high compression quantization matrix.

44	74	13	3	61	72	70	27
50	27	21	31	2	69	73	25
21	44	61	1	2	8	48	87
64	93	63	38	19	45	55	23
32	21	37	68	59	44	12	80
96	84	58	9	6	35	45	91
73	68	45	4	37	15	72	23
41	63	4	61	63	68	89	24

DCT + Rounding →

358	2	74	18	–41	–5	–23	–2
–25	–18	0	2	–26	–6	15	–8
–3	–27	–17	75	–26	14	–44	–7
10	5	16	18	26	–17	26	7
37	4	–62	30	1	–32	–40	–32
2	44	28	0	–36	–16	–20	38
–4	–31	26	–26	19	–10	28	14
–9	–14	–14	–34	18	–48	–18	7

1	2	4	8	16	32	64	128
2	4	4	8	16	32	64	128
4	4	8	16	32	64	128	128
8	8	16	32	64	128	128	128
16	16	32	64	128	128	256	256
32	32	64	128	128	256	256	256
64	64	128	128	256	256	256	256
128	128	128	256	256	256	256	256

Quantization matrix 2

Quantized result →

358	1	19	2	–3	0	0	0
–13	–5	0	0	–2	0	0	0
–1	–7	–2	5	–1	0	0	0
1	1	1	1	0	0	0	0
2	0	–2	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 9.7.8

The final step in the JPEG process is the coding of the quantized image. In this, the quantized image is arranged in a row using the Zig-zag sequence. The path of the zig-zag sequence is shown in Fig. 9.7.9 :

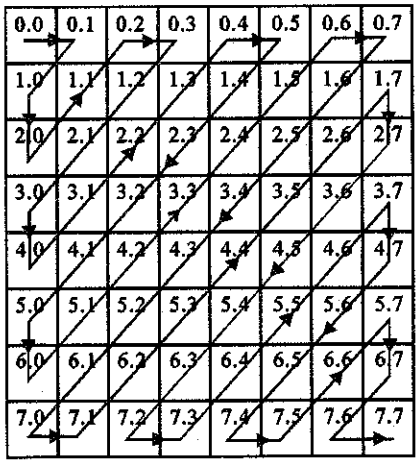
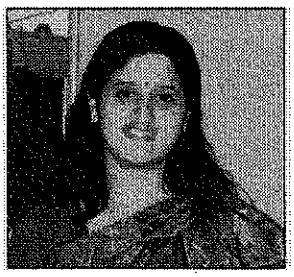


Fig. 9.7.9 : The path of zig-zag sequence on the quantized image

If we observe the zig-zag row we notice that the initial values are non-zero after which there is a long row of zeros. Huffman coding is used to code the non-zero values while Run Length Encoding (RLE) algorithm is used to code the zero values. The reason for using the zig-zag sequence is to get a long run of zeros.

These basic steps are used in all JPEG images that you see on Internet sites.

(a) Original image



(b) JPEG Compression with 10:1 compression

Fig. 9.7.10

Though any quantization matrix could be used. The JPEG group has standardised a quantization matrix. All JPEG images use the matrix shown.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

The weights are assigned according to heuristically determined perceptual or pychovisual importance. To change the compression ratio, we simply multiply this matrix by α.

$$\alpha = \begin{cases} \dfrac{50}{\text{Q-factor}}; & 1 \le \text{Q-factor} < 50 \\ 2 - \dfrac{\text{Q-factor}}{50}; & 50 \le \text{Q-factor} < 100 \end{cases} \quad \ldots(9.7.2)$$

Depending on the Q-factor (quality factor), α changes and hence the quantisation matrix $\alpha \times Q$ changes. To increase compression, α needs to be increased which in turn reduces the quality of the image. Another aspect of standardization of the JPEG standard is subtraction of the original image by 2^{n-1} (if the number of grey levels is 2^n).

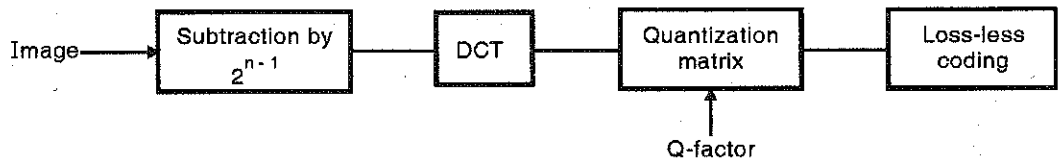


Fig. 9.7.11

If the original image is of 8-bits (0 - 255), this subtraction shifts the range to –128 - +128. This subtraction increases the compression rates without reducing the subjective quality of the image.

A typical block diagram of JPEG encoder and JPEG decoder are shown in Fig. 9.7.12.

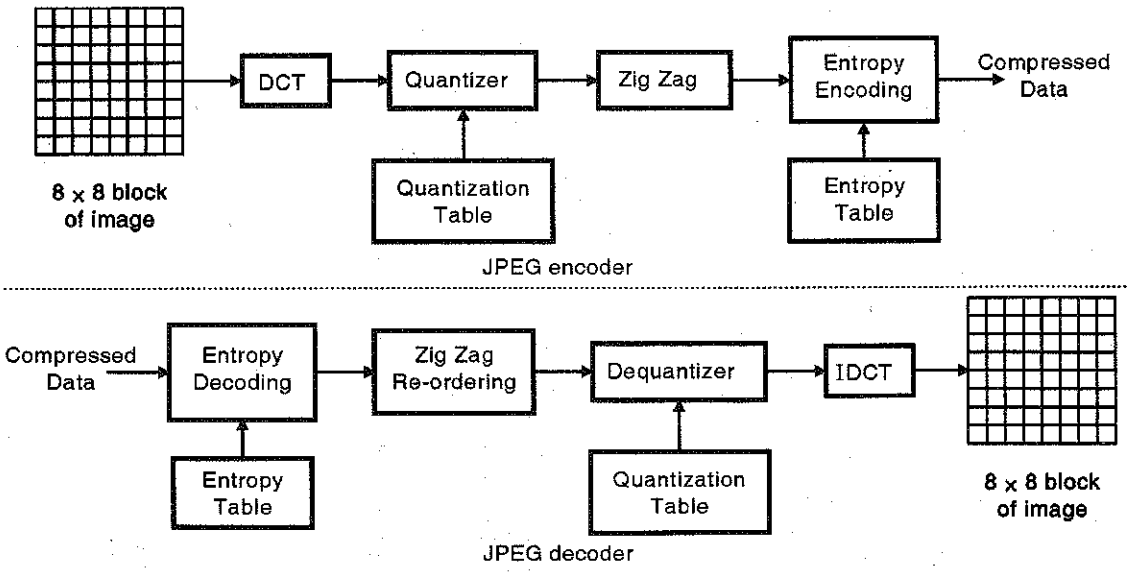


Fig. 9.7.12

9.8 JPEG 2000

JPEG 2000 is similar to the JPEG that we just discussed. It is also quite different. Instead of going through the details of JPEG 2000, which is beyond the scope of under graduate study, we shall try to understand the broad differences it has with the standard JPEG. A general Block diagram of JPEG 2000 is shown in Fig. 9.8.1.

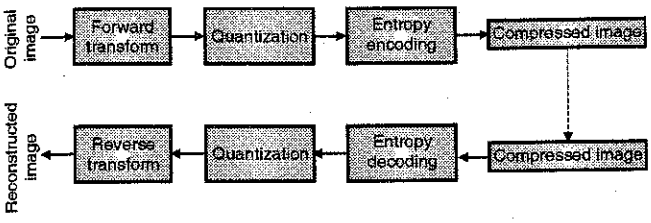


Fig. 9.8.1

Although this general block diagram looks like the one for the conventional JPEG, there are major differences in all of the processes of each block of the diagram.

The forward transform that is used in JPEG is the Discrete Cosine Transform (DCT). The forward transform that is used in JPEG 2000 is the Discrete Wavelet transform (DWT).

Wavelet decomposition uses the fact that it is possible to resolve high frequency components within a small time window and low frequency components within a large time window. This is because a low frequency component completes a cycle in a large time interval whereas a high frequency component completes a cycle in a much shorter interval.

Wavelet analysis is all about getting time as well as frequency information together, i.e. which frequency at what time.

The Pre-Entropy coding of JPEG is Run length encoding (RLE) while the Pre-Entropy coding of JPEG 2000 is Arithmetic Coding. The Entropy coding of JPEG uses the Huffman coding while the Entropy coding of JPEG 2000 utilizes Embedded Block coding with optimized truncation (EBCOT).

These differences offer a great deal of sophistication to JPEG 2000. One main advantage is that JPEG 2000 offers both lossy and lossless compression in the same file stream, while JPEG usually only utilizes lossy compression.

The use of the DWT facilitates Multi-Resolution analysis. The DWT breaks down an image into different decomposition levels. These decomposition levels contain a number of subbands, which consist of coefficients that describe the horizontal and vertical spatial frequency characteristics of the original image.

This feature allows the user to select a Region of interest (ROI) which can be viewed at a high quality, while leaving the remaining part of the image at a lower quality. This way one can view only the portions of the image that are necessary for the user. Another advantage of JPEG 2000 over JPEG is Error resilience. Error resilience measures the ability of a compression method to avoid letting errors introduced into the image file affect the quality of the image. Acquisition of images including downloading could introduce some amount of noise in the image. Hence the original image that needs to be compressed itself has a small amount of noise of noise present in it. JPEG 2000 offers significantly higher error resilience than JPEG; therefore, there is lesser chance that the image will be corrupted and its quality sacrificed in some way.

9.9 Image Compression Model

In sections 9.1 and 9.2, we discussed the need for compression and the two general types of compression viz., Lossy and Lossless. Concepts discussed in the earlier topics are combined together to form a practical image compression system. It was decided to put this topic at the end because it involves some of the earlier stated concepts. A typical image compression system is shown in Fig. 9.9.1.

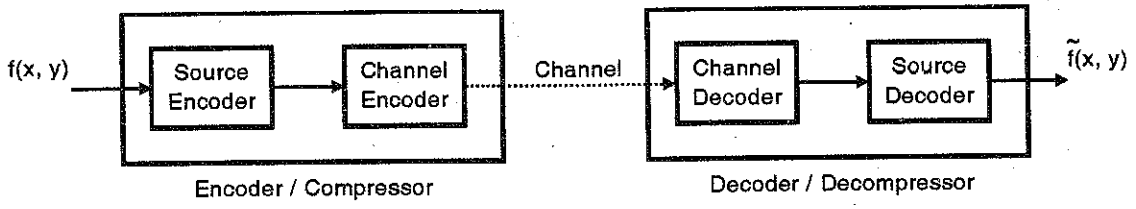


Fig. 9.9.1

As seen in Fig. 9.9.1, a practical image compression system consists of two blocks :An Encoder (Compressor) and a Decoder (Decompressor).

The image f(x, y) is fed to the encoder which, as the name suggests, encodes the image so as to make it suitable for transmission [It converts the image f(x, y) into a set of symbols]. The decoder receives this transmitted signal and reconstructs the output image f(x, y). If the system is an error free one, f(x, y) will be an exact replica of f(x, y).

The encoder and the decoder are made up of two blocks each. The encoder is made up of a Source encoder and a Channel encoder. The source encoder removes the input redundancies while the channel encoder increases the noise immunity of the source encoders output (example : Hamming codes).

The decoder consists of a channel decoder and a source decoder. The function of the channel encoder and the channel decoder is to ensure that the system is immune to noise. Hence if the channel between the encoder and the decoder is noise free, the channel encoder and the channel decoder are omitted. We shall discuss each pair in detail.

Source encoder and source decoder : We have already studied the three basic types of redundancies in an image viz, interpixel redundancies, coding redundancies and psychovisual redundancies.

We have also discussed techniques of eliminating or reducing these redundancies. Run-length encoding is used to eliminate or reduce interpixel redundancies Huffman encoding is used to eliminate or reduce coding redundancies while I.G.S. coding is used to eliminate or reduce interpixel redundancies. These are nothing but source encoders. The job of the source decoders is to get back the original signal (or modified output).

The problems that we solved on Run-length encoding, Huffman encoding and I.G.S. coding are apt examples of source encoders and source decoders.

Now that we know what source encoders and source decoders are, we shall proceed to discuss them is a very general form. A typical source encoder is shown in Fig. 9.9.2.

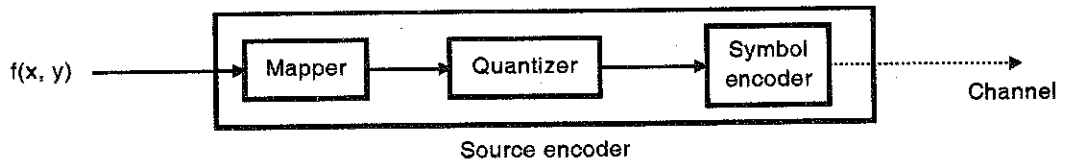


Fig. 9.9.2

The input image is passed through a mapper. The mapper reduces the interpixel redundancies. As stated earlier, Run-length encoding is an example of mapping. The Mapping stage is a lossless technique and hence is generally reversible. The output of a mapper is passed through a Quantizer block.

The quantizer block reduces the psychovisual redundancies. It compresses the data by eliminating some information and hence is an irreversible operation. We have used the quantizer block in JPEG compression. The quantizer block necessarily means Lossy compression. Hence in case of lossless compression, the quantizer block is omitted.

The final block of the source encoder is that of a symbol encoder. This block creates a variable length code to represent the output of the quantizer. The Huffman code is a typical example of symbol encoder. The symbol encoder reduces coding redundancies.

The block diagram of the source decoder is shown in Fig. 9.9.3.

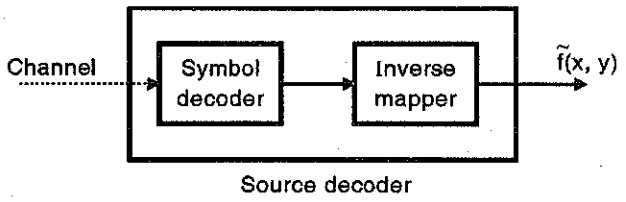


Fig. 9.9.3

The source decoder block performs exactly the reverse operation of the symbol encoder and the mapper blocks. It is important to note that the source decoder has only two blocks.

Since quantization (quantizer block) is irreversible, an inverse quantizer block does not exist. Here we have assumed that the channel is noise free and hence have ignored the channel encoder and channel decoder.

Channel encoder and decoder : The channel encoder is used to make the system immune to transmission noise. Since the output of the source encoder has very little redundancy, it is highly susceptible to noise. The channel encoder inserts a controlled form of redundancy to the source encoder output making it more noise resistant.

9.9.1 Comparison of Lossless and Lossy Compression

Sr. No.	Lossless compression	Lossy compression
1.	It removes only the redundant information.	It removes visually irrelevant data.
2.	As the name suggests, none of the information is lost.	There is always a loss of information.
3.	E_{rms} between the original image and the reconstructed images is always zero.	E_{rms} is never equal to zero.
4.	It is reversible.	It is irreversible.
5.	Compression ratio is low.	Compression ratio can be very large.
6.	Used in text files, computer files, dills, medical reports etc.	Used in speech and images.

Ex. 9.9.1

Apply Huffman coding procedure to following message assemble and determine average length of encoded message and coding efficiency (η).

The symbols (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) are emitted with probabilities of (0.22, 0.2, 0.18, 0.15, 0.10, 0.08, 0.05, 0.02).

Soln. :

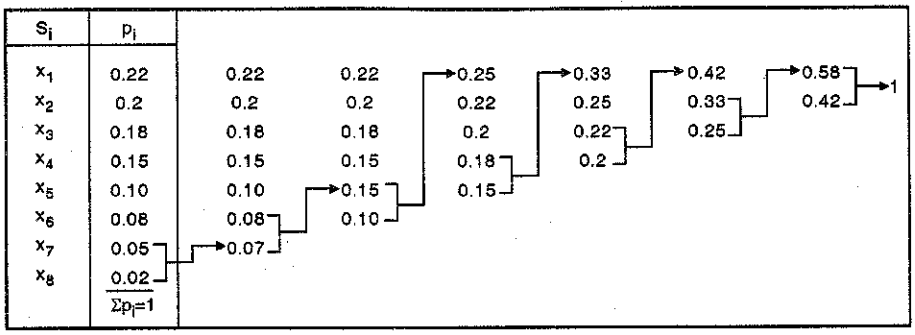
Step 1 :

The given information is as follows :

S_i Symbols	p_i Probability
x_1	0.22
x_2	0.2
x_3	0.18
x_4	0.15
x_5	0.10
x_6	0.08
x_7	0.05
x_8	0.02

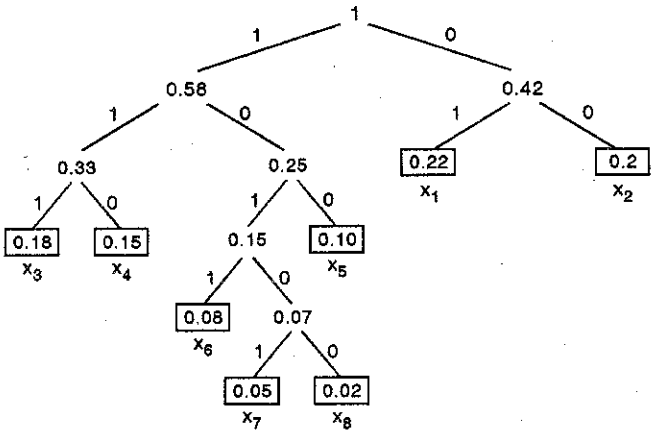
Step 2 :

The given probabilities are already in the descending order. We now create the Huffman table. In this we add the smallest 2 probabilities and again place them in the descending order.



Step 3 :

From the Huffman table, we create the Huffman tree. We start from 1 and move backwards.



Each of the end node (boxes) in the original probability of the data.

Step 4 :

We now start labelling each branch. Let the left branches be 1 and the right branches be 0. (It could be the other way too). We talk along the branches to obtain the codes.

∴ Hence the codes obtained are

Symbol	Huffman code	No. of bits
$x_1 \rightarrow$	01	2
$x_2 \rightarrow$	00	2
$x_3 \rightarrow$	111	3
$x_4 \rightarrow$	110	3
$x_5 \rightarrow$	100	3
$x_6 \rightarrow$	1011	4
$x_7 \rightarrow$	10101	5
$x_8 \rightarrow$	10100	5

Average length code = Σ (Number of bits for each symbol) × (Its probability)

∴ Average length code = 2(0.22) + 2(0.2) + 3(0.18) + 3(0.15) + 3(0.10) + 4(0.08) + 5(0.05) + 5(0.02)

∴ Average length code = 2.8 bits/symbol

Had we used the natural code, we would require 3 bits for each symbol since there are 8 symbols ($2^3 = 8$). Hence the average length code would have been 3 bits/symbol.

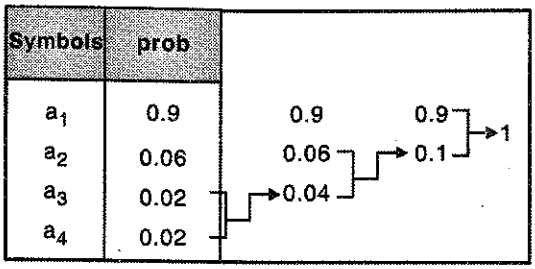
Ex. 9.9.2

Calculate the efficiency of Huffman code for the following symbols whose probability of occurrence is given below.

Symbol	Probability
a_1	0.9
a_2	0.06
a_3	0.02
a_4	0.02

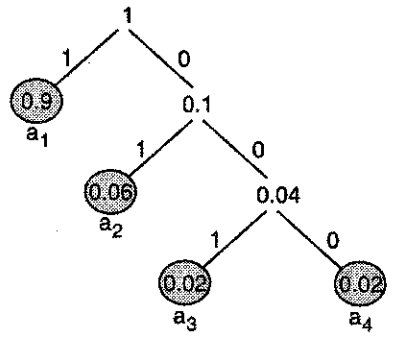
Soln. :

We generate the Huffman codes by forming the Huffman table. Here the probabilities are already in the descending order.



We now generate the Huffman tree. We start from the right hand corner of the Huffman table.

We label the left branches as 1 and right branches as 0.



∴ The Huffman codes generated are :

$a_1 \rightarrow 1$

$a_2 \rightarrow 0\ 1$

$a_3 \rightarrow 0\ 0\ 1$

$a_4 \rightarrow 0\ 0\ 0$

In Huffman coding, the symbol with the highest probability is given fewer bits for representation.

Average length code = $\sum$ (Number of bits for each symbol) × (Its probability)

$= (1 \times 0.9) + (2 \times 0.06) + (3 \times 0.02) + (3 \times 0.02)$

Average length code = 1.14 bits/symbol

Had we used Natural code, we would have required 2 bits/symbol for representing each symbol.

Coding efficiency

To calculate the coding efficiency, we first need to calculate the entropy.

$$\text{Entropy} = H = -\sum_{i=0}^{L-1} p_i \log_2 p_i$$

$$\therefore H = -(0.9 \log_2 0.9 + 0.06 \log_2 0.06 + 0.02 \log_2 0.02 + 0.02 \log_2 0.02)$$

$$\therefore H = -(-0.136 - 0.243 - 0.11 - 0.11)$$

$$\therefore H = 0.599$$

Hence entropy of the given data is 0.599 bits/symbol. This is the maximum compression that can be achieved with this data.

$$\text{Now coding efficiency} = \frac{\text{Entropy}}{\text{Average code length}} = \frac{0.599}{1.14}$$

Coding efficiency = 0.525

In terms of percentage, the coding efficiency is 52.5%

Ex. 9.9.3

A simple 4 × 4 image is represented by following matrix :

$$\begin{bmatrix} 20 & 140 & 100 & 20 \\ 20 & 140 & 100 & 20 \\ 240 & 140 & 240 & 240 \\ 240 & 140 & 240 & 240 \end{bmatrix}$$

(i) Determine the entropy of image.

(ii) Generate a simple Huffman code-book for various grey level in the image.

(iii) Coding redundancy in the image.

Soln. :

(i) The Entropy is given by the formula

$$\mathbf{H = -\sum_{i=0}^{L-1} p_i \log_2 p_i}$$

Let us begin with find the probabilities of each grey level.

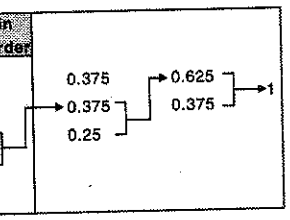
In this case, total number of pixels is 16. Hence the probabilities are as shown below

Grey level	Number of pixel	Probability	Entropy
20	04	0.25	$-0.25 \log_2 (0.25) = 0.5$
100	02	0.125	$-0.125 \log_2 (0.125) = 0.375$
140	04	0.25	$-0.25 \log_2 (0.25) = 0.5$
240	06	0.375	$-0.375 \log_2 (0.375) = 0.53$
	$\sum = 16$		$\sum = 1.905$

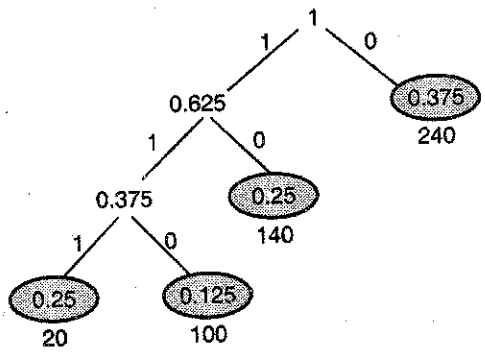
Hence the entropy of the image is 1.905.

(ii) Huffman coding

Grey level	n_k	Probability	Probability in descending order
20	04	0.25	0.375
100	02	0.125	0.25
140	04	0.25	0.25
240	06	0.375	0.125
	$\Sigma = 16$		



We now generate the Huffman tree. We label the left side branches as 1 and right side branches as 0.



Hence the Huffman code for the grey level is

20 → 111

100 → 110

140 → 10

240 → 0

Hence average code length using Huffman coding is

Code length = $\sum$ Number of bits for each symbol × Probability

Code length = $3 \times p(20) + 3 \times p(100) + 2 \times p(140) + 1 \times p(240)$

$= (3 \times 0.25) + (3 \times 0.125) + (2 \times 0.25) + (1 \times 0.375)$

Code length = 2 bits / symbol

(iii) Coding redundancy

The given image has 4 different grey levels viz. 20,100,140 and 240. We would require 2 bits to represent each of the 4 grey levels using equal length codes. Using Huffman coding also we require 2 bits/symbol. Hence the coding redundancy in the given image is 0.

Ex. 9.9.4

A simple 4 × 4 image is represented by the following matrix :

30	30	30	20
20	30	30	30
30	20	20	20
20	10	10	0

(i) Determine the entropy of image.

(ii) Efficiency of Huffman coding.

(iii) Compression ratio.

Soln. :

(i) Entropy of image

Entropy is given by the formula

$$H = -\sum_{i=0}^{L-1} p_i \log_2 p_i$$

where p_i is the probability of each grey level.

In this case, total number of pixels is 16. Hence the probabilities are as shown below.

Grey level	Number of pixels	Probability	Entropy
0	01	0.0625	$-0.0625 \log_2 (0.0625) = 0.25$
10	02	0.125	$-0.125 \log_2 (0.125) = 0.375$
20	06	0.375	$-0.375 \log_2 (0.375) = 0.53$
30	07	0.4375	$-0.4375 \log_2 (0.4375) = 0.521$
	$\Sigma n_k = 16$		$-\Sigma p_i \log_2 p_i = 1.676$

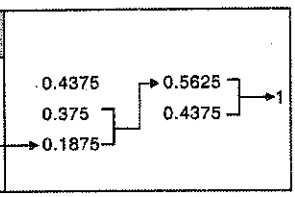
Entropy gives in the optimum value to which compression can take place.

Hence the entropy is 1.676 bits/symbol.

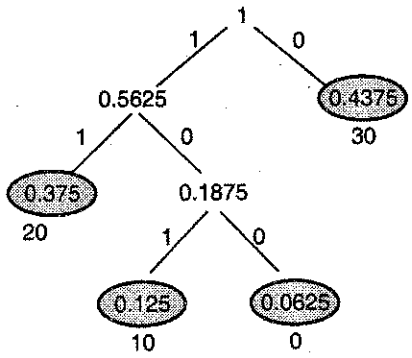
(ii) Huffman coding

We form the Huffman table.

Grey level	Probability	Arrange in descending order
0	0.0625	0.4375
10	0.125	0.375
20	0.375	0.125
30	0.4375	0.0625



We draw the Huffman tree



We label the left side branches as 1 and right side branches as 0. Hence we have,

Grey level	Huffman code
30	0
20	11
10	101
0	100

We now calculate the average word length obtained using Huffman code.

$$\text{Average length} = \sum (\text{Number of bits for each symbol}) \times (\text{Its probability})$$

$$= (1 \times p(30)) + (2 \times p(20)) + (3 \times p(10)) + (3 \times p(0))$$

$$= (1 \times 0.4375) + (2 \times 0.375) + (3 \times 0.125) + (3 \times 0.0625)$$

$$\text{Average length} = 1.75 \text{ bits/symbol}$$

Hence Huffman code requires an average of 1.75 bits/symbol for representing the image.

(iii) Compression ratio

There are 4 grey levels in the image. Without any form of compression, we should require 2 bits/symbol to represent the 4 grey levels. Using Huffman coding, we require 1.75 bits/symbol.

Hence compression ratio is given by the formula

$$\text{Compression ratio} = \frac{Nb - K}{Nb} \times 100\%$$

Where Nb is the original number of bits required and K is the number of bits required after compression

$$\therefore \quad CR = \frac{2 - 1.75}{2} \times 100\ \% = 12\ \%$$

Ex. 9.9.5

Find the arithmetic code word of the message : INDIA.

Soln. :

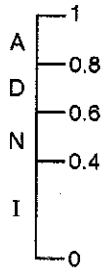
The given word INDIA has 5 letters having the following probabilities.

Source symbol	Probability	Initial subinterval
I	0.4	[0, 0.4]
N	0.2	[0.4, 0.6]
D	0.2	[0.6, 0.8]
A	0.2	[0.8, 1]

We also write down their sub-intervals (cdfs'). Since there are two I's, the probability of I is $\frac{2}{5} = 0.4$.

Here $P_1 = 0.4$, $P_2 = 0.2$, $P_3 = 0.2$

This is shown below



The first symbol is I. We blow up (zoom) the range of I and create new sub-intervals using equation.

$\{[L, L + (U - L) P_1], [L + (U - L) P_1, L + (U - L) P_2],$

$[L + (U - L) P_2, L + (U - L) P_3], ...\}$

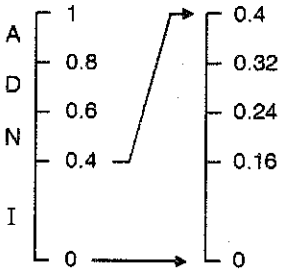


Fig. P. 9.9.5

The next symbol is N. We hence blow up the range corresponding to N which is 0.16 - 0.24 (2^{nd} interval) and create new subintervals.

The entire diagram for the word INDIA is shown below :

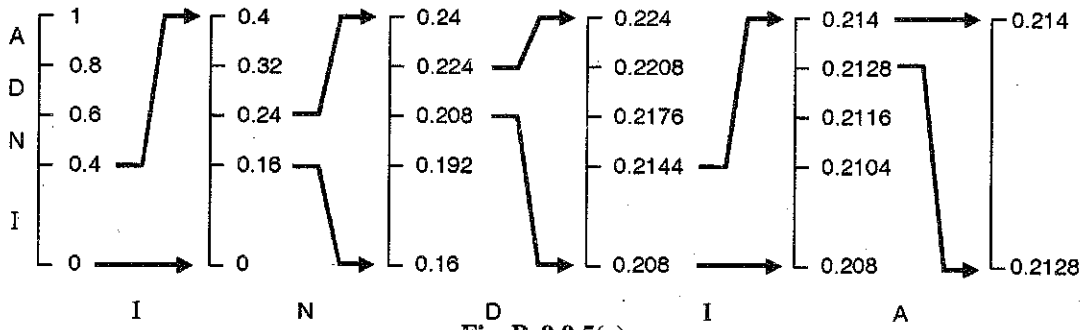


Fig. P. 9.9.5(a)

The tag word could either be 0.2128 or 0.214.

Ex. 9.9.6

Justify / contradict statement : All Image compression techniques are invertible.

Soln. : False.

If an input x(n), transformed by a function T produces an output y(n), then putting y(n) into the inverse function T^{-1} produces the output x(n), and vice versa. i.e., $y(n) = T[x(n)]$, and $x(n) = T^{-1}[y(n)]$. Here T^{-1} is called the inverse of T. Such a transformation that has an inverse is called Invertible. Not all functions have an inverse.

In image compression, after performing Lossless compression, we can get back the original image. Hence Lossless compression techniques are invertible. However, while performing lossy Compression, we discard data that is not visually significant. Hence it is impossible to get back the original image back after performing lossy compression. Hence lossy compression techniques are not invertible hence all image compression techniques are not invertible.

Summary

Users of image processing techniques have to handle enormous amounts of data. Storing and transmitting images require huge storage space as well as very wide channel capacities. Image compression deals with reducing the number of bits required to represent an image. There are compression techniques that incur losses while there are others in which losses are absolutely zero. Lossy as well as Loss-less image compression techniques have been discussed here and examples of each have been provided. Some of the compression techniques presented here are from standard signal transmission methodologies. Special attention has been paid to the JPEG standard which uses the Discrete Cosine Transform. However the treatment here is only at an elementary level and interested readers may consult relevant books on this topic for more detail.

Review Questions

Q. 1 What do we mean by redundancy in digital images ?

Q. 2 Explain the following redundancies :

(a) Psychovisual redundancy

(b) Interpixel redundancy

(c) Coding redundancy

Q. 3 Explain the role of data fidelity criteria in image compression.

Q. 4 Classify with reasons, the following data compression techniques into lossy and lossless schemes

(a) Huffman coding

(b) Run length coding

(c) DCT compression

(d) Subsampling

(e) Predictive coding.

Q. 5 Compare and contrast between lossy compression and lossless compression.

Q. 6 Run length coding is a loss-less compression technique explain.

Q. 7 Why is lossy compression not suitable for compressing executable files ?

Q. 8 A set of 8 symbols are given below, Generate a Huffman code for these symbols ?

Symbol	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
Probability	0.06	0.02	0.3	0.5	0.04	0.01	0.03	0.04

Q. 9 Generate the Huffman codes for the symbols given below.

Symbol	a_1	a_2	a_3	a_4	a_5	a_6
Probability	0.1	0.4	0.06	0.1	0.04	0.3

Q. 10 What are the fidelity criteria ? How are these used in quantifying the nature and extent of information loss ?

Q. 11 Can variable length coding procedures be used to compress histogram equalised images with in grey levels ? Explain.

Q. 12 How many unique Huffman codes can a three symbol source have ? Construct them.

Q. 13 What do we mean by image compression standards ?

Q. 14 At what stage lossy compression achieved in a JPEG image ?

Q. 15 Consider a 8 pixel line of grey scale data {12, 12, 13, 13, 10, 13, 57, 54} which has been uniformly quantized with 6 bit accuracy. Construct its 3 bit IGS code.

Q. 16 Explain the salient features of the following codes :

(a) Huffman code

(b) Lossy predictive coding

(c) Transform coding.

Q. 17 Describe briefly the features of a compression model with a neat block diagram.

Q. 18 Can a histogram equalised image be compressed further. Prove your statement by an example.

Q. 19 Is the Huffman code optimal ? Prove with an example.

Q. 20 Write an algorithm to generate :

(a) Huffman code

(b) Run-length code

(c) I.G.S. code.

Q. 21 Generate an algorithm to achieve LZW compression.

Chapter Ends...

❑❑❑

CHAPTER

10

Image Morphology

10.1 Introduction

Morphology is the science of form and structure. Morphology, in the strictest sense, denotes a branch of biology that deals with the form and structure of animals and plants. In image processing, morphology is about regions and shapes. It is used as a tool for extracting image components that are useful in representing regions and shapes. Unlike the traditional image processing techniques, morphological techniques treat an image as an ensemble of sets. Morphology can hence be defined as an interaction between two sets.

Prior to discussing the mathematics involved in morphology, it is imperative to understand some simple operations that are performed on pixels.

10.2 Arithmetic and Logical Operation

There are two different types of operations that are widely used in image processing especially in image morphology.

10.2.1 Arithmetic Operations

Arithmetic operations between two pixels a and b are denoted as follows :

(1) Addition : a + b

Image addition is used in image averaging to reduce noise. This kind of operation was performed in image enhancement.

(2) Subtracting : a – b

Image subtraction is widely used in medical imaging. A very common example is the Digital Subtraction Angiography (DSA).

Image Subtraction is basically used to get rid of background information.

(3) Image multiplication : a × b

(4) Image division : a / b

Both, Image Multiplication and Image Division are used to correct grey level shading that result from non uniformities in illumination or in the sensor used.

One thing that needs to be understood in arithmetic operations is that these are carried out on an entire image, pixel by pixel. Arithmetic operations involve only one spatial pixel location at a time and hence can be done in place. Arithmetic operations are applied to multi-valued pixels.

10.2.2 Logical Operations

Logical operations commonly used are as follows :

(1) AND : a AND b

(2) OR : a OR b

(3) COMPLEMENT : NOT a

These operations can be combined to form other logic operations. Logic operations actually apply only to binary images. (image having only two values, 0 and 1).

The AND operator gives out 1 only when both 'a' and 'b' are equal to 1.

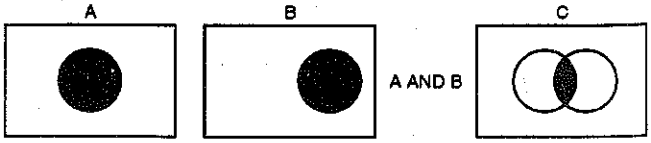


Fig. 10.2.1

The OR operator gives out 1 if either 'a' or 'b' or both are equal to 1.

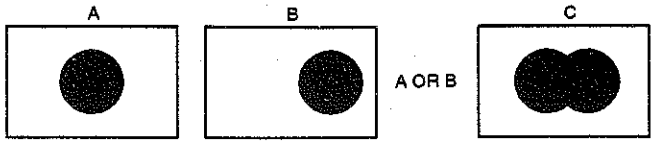


Fig. 10.2.2

The COMPLEMENT (NOT) operator gives out 1 when a = 0

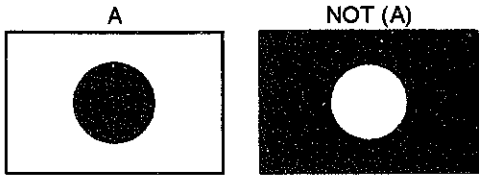


Fig. 10.2.3

Note : Shading of black colour have been inverted for better visualisation.

The AND operator applied on sets A and B is also denoted in the jargon of set theory as

$$C = A \cap B$$

i.e., C is the intersection of A and B. Similarly, the OR operator applied on sets A and B is denoted as

$$C = A \cup B$$

i.e., C is the union of A and B. Formal mathematics of set theory utilize concepts of Sets, Inclusion, Complement, Union and Intersection. A more intuitive approach based on plain english description of the set theory operations are expressions like OR, AND and NOT. Though, OR, AND and NOT are more intutive, most of the technical papers still use the set theory notations and hence it would be helpful to understand some of these.

10.3 Basic Definitions

In set theory, points are represented in lower-case italics and sets are represented as upper case italics. ϕ means an empty set, $\in$ means is a member of or belongs to, $\Rightarrow$ means implies that. The expression {p|<conditions>} means the set of points, p, such that the listed <conditions> are true.

(1) Consider the binary image shown in Fig. 10.3.1

Here the binary object A which is a subset of pixels p is contained or included within the total set of image pixels.

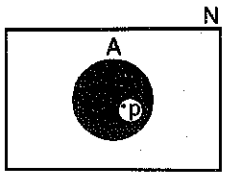


Fig. 10.3.1

$A \in N$. If p is a element of A, then p is also an element of N

i.e., $p \in A \Rightarrow p \in N$

(2) The complement of set A are those elements which are not elements of A. Hence the white background in Fig. 10.3.1 forms the complement of (A^c)

Since $p \in A \Rightarrow p \notin A^c$

(3) The union of two sets is denoted as $A \cup B$. (This is the OR operation). Here the resulting set is a group of pixels such that each pixel is an element of A or B or both.

$\therefore A \cup B = \{p | p \in A \text{ or } p \in B\}$

(4) The intersection of two sets is denoted as $A \cap B$. (This is the AND operation). Here the resulting set is a group of pixels such that each pixel belongs to both the sets A and B.

$\therefore A \cap B = \underbrace{\{p | p \in A \text{ and } p \in B\}}_{\text{< Condition >}}$

Ex. 10.3.1

Given image A and B. Find $A \cup B$ and $A \cap B$.

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 1 & 1 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

Soln. :

$A \cup B$/ (A Union B) /(A OR B) =

1	0	1	1	1
0	1	0	1	1
1	0	1	1	1
1	1	1	1	1

$A \cap B$/ (A Intersection B) /(A AND B) =

0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	0	1	1

With these basic definitions, we can now progress to some important operations of image morphology. Two sets of operations that are fundamental to image morphology are Dilation and Erosion. Dilation has the effect of filling in the valleys between spiky edges while Erosion has the effect of deleting spiky edges. Understanding the Dilation and Erosion operations are imperative as many of the morphological algorithms discussed later are based on these two fundamental operations.

10.3.1 Dilation

With A and B as two sets in Z^2 (2D-integer space), the dilation of A by B is defined as

$$A \oplus B = \{Z \mid (\hat{B})_z \cap A \neq \phi\} \quad ...(10.3.1)$$

In the above example, A is the image while B is called a structuring element.

In the equation, $(\hat{B})_z$ simply means taking the reflection of B about its origin and shifting it by Z. Hence dilation of A with B is a set of all displacements, Z, such that $(\hat{B})$ and A overlap by at least one element. Flipping of B about the origin and then moving it past image A is analogous to the convolution process. In practice flipping of B is not always done. Dilation adds pixels to the boundaries of objects in an image. The number of pixels added depends on the size and shape of the structuring element. Based on this definition, dilation can also be defined as

$$A \oplus B = \{Z \mid [(\hat{B})_z \cap A] \in A\} \quad ...(10.3.2)$$

10.3.2 Erosion

For image A and structuring element B in the Z^2 (2D integer space), Erosion is defined as

$$A \ominus B = \{Z \mid [(\hat{B})_z \in A\} \quad ...(10.3.3)$$

This equation indicates that erosion of A by B is the set of all points Z such that B, translated (shifted by Z), is a subset of A i.e., B is entirely contained within A. Erosion reduces the number of pixels from the object boundary. The number of pixels removed depends on the size of the structuring element.

There is an alternative formula used to define Dilation and Erosion.

Dilation :

It combines two sets using vector addition (or Minkowski set addition),

i.e. $(a, b) + (c, d) = (a + c, b + d)$

$\therefore A \oplus B = (p \in Z^2 \mid p = a + b, a \in A \text{ and } b \in B)$...(10.3.4)

example if

$A = \{(1, 0), (1, 1), (1, 2), (2, 2), (0, 3), (0, 4)\}$

$B = \{(0, 0), (1, 0)\}$

then

$A \oplus B = \{(1, 0), (1, 1), (1, 2), (2, 2), (0, 3), (0, 4), (2, 0), (2, 1), (2, 2), (3, 2), (1, 3), (1, 4)\}$

If A and B are represented as images, we get

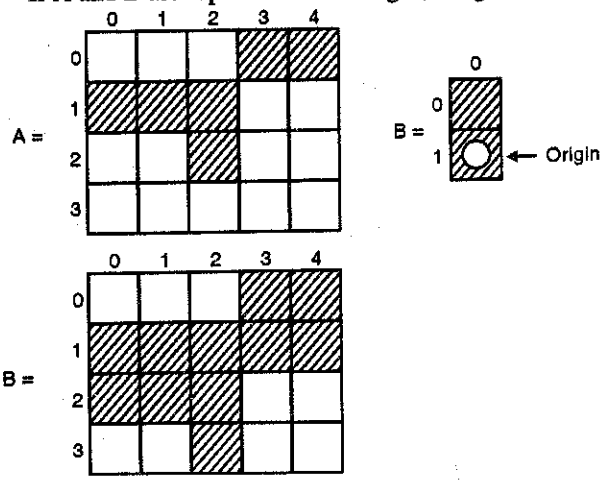


Fig. 10.3.2

Erosion : In a similar manner, Erosion combines two sets using vector subtraction of set elements.

$$A \ominus B = (p \in Z^2 \mid p + b \in A \text{ for every } b \in B) \quad \ldots (10.3.5)$$

This simply means that every point p from the image is taken and the result of erosion is given by those points p for which all possible p + b are in A.

Ex. 10.3.2

A = {(1, 0), (1, 1), (1, 2), (0, 3), (1, 3), (2, 3), (3, 3), (1, 4)}

B = {(0, 0), (1, 0)}.

Soln. :

Now (A ⊖ B) are those value of p + b that belong to A

$\therefore$ A ⊖ B = {(0, 3), (1, 3), (2, 3)}

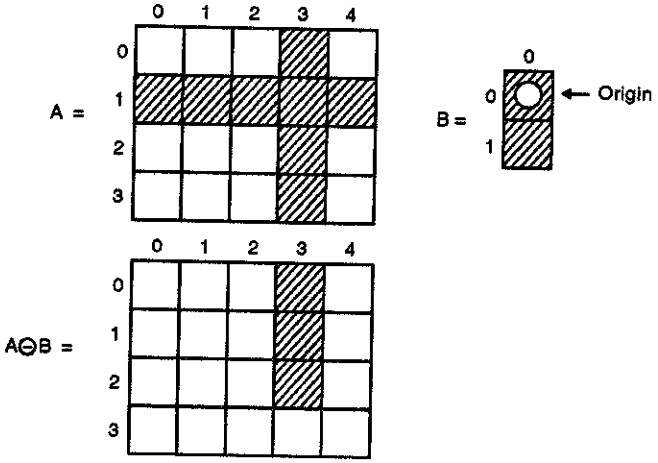


Fig. P. 10.3.2

10.4 A Simple Practical Formula for Implementing Dilation and Erosion

Dilation : Let A(x, y) be a binary image and C(x, y) be the resulting image obtained after A(x, y) has been dilated with a m × n template B(i, j). B is called the structuring element where $0 \le i \le m-1$, $0 \le j \le n-1$

B is usually a binary structure. For dilation

$$C(x, y) = \text{Maximum } \{A(x-i, y-j) \times B(i, j)\} \quad \ldots (10.4.1)$$

This simply means that if B is a binary structuring element, the output pixel is the maximum value of all the pixels in the input pixels neighbourhood multiplied by the corresponding coefficients of the structuring element. Since the structuring element is usually a binary mask, for a binary image, if any of the pixels has a value 1, the output pixel is set to 1.

Fig. 10.4.1 illustrates the dilation of a binary image. The structuring element that we use is 1 (1) 1

Consider the image shown.

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Dilation is achieved as shown.

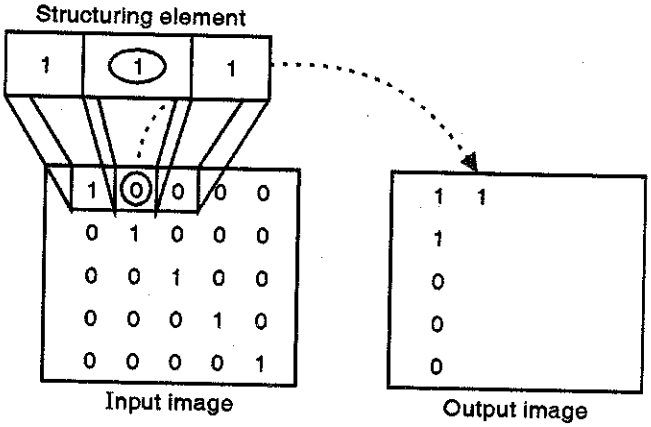


Fig. 10.4.1

Erosion : In a similar fashion, we can achieve erosion by using a simple formula which could be used in programming.

If A(x, y) is the image and B(i, j) is the structuring element, then C(x, y) which is the eroded image is given by,

$$C(x, y) = \text{Minimum } \{A(x-i, y-j) \times B(i, j)\} \quad \ldots (10.4.2)$$

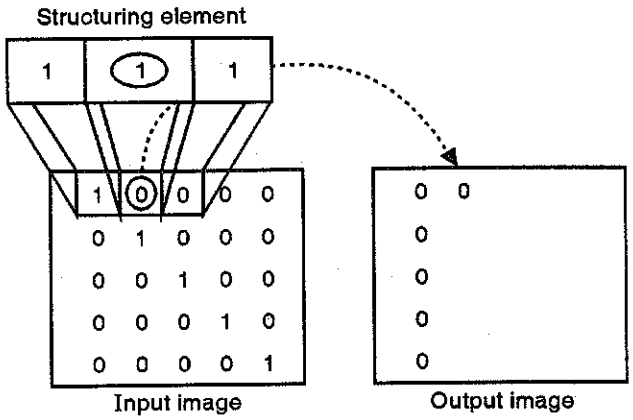


Fig. 10.4.2

The Fig. 10.4.2 shows the erosion operation. These formulae can be used only if all the coefficients of the structuring element are equal to 1. Dilation and Erosion can also be achieved on grey level images. Fig. 10.4.3 and Fig. 10.4.4 explain dilation and erosion on grey level images.

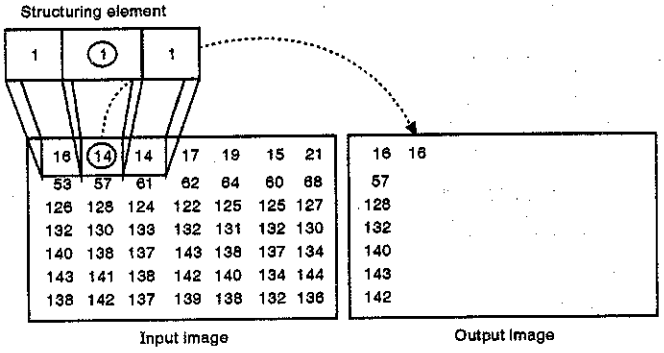


Fig. 10.4.3 : Dilation

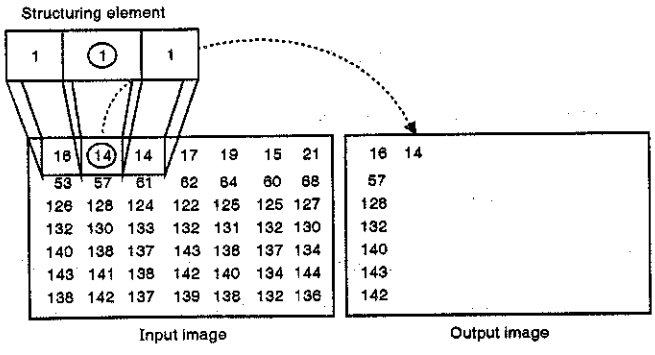


Fig. 10.4.4 : Erosion

10.5 Structuring Elements

An essential part of Dilation and Erosion operations is the structuring element. One-dimensional and Two-dimensional structuring elements consist of a matrix of 0's and 1's and its size is much smaller than the image. The center pixel of the structuring element is called the origin and it identifies the pixel which is being processed. Depending on the shape present in the input image, we need to define the structuring element. Some common shapes of structuring elements are lines, squares, diamonds, disks, balls etc.

MATLAB Program for dilation and erosion

```
%% Dilation and erosion of a pseudo image %%
clear all
clc
a=zeros(9,10);          %% Defining a 2-D array %%
a(4:6,4:7)=1;
w=[1 1 1; 1 1 1; 1 1 1];      %% Structuring element %%
for x=2:1:8
for y=2:1:9
    a1=[w(1)*a(x-1,y-1) w(2)*a(x-1,y) w(3)*a(x-1,y+1)...
        w(4)*a(x,y-1)  w(5)*a(x,y)  w(6)*a(x,y+1)...
        w(7)*a(x+1,y-1) w(8)*a(x+1,y) w(9)*a(x+1, y+1)];
    A1(x,y)=max(a1);   %% Dilation
    A2(x,y)=min(a1);   %% Erosion
end
end
a, A1, A2
```

Output of program

>> a =

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0	0	0
0	0	0	1	1	1	1	0	0	0
0	0	0	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Original image

A_1 =

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0

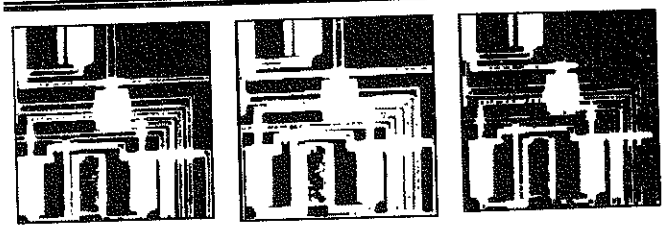
Dilation

A_2 =

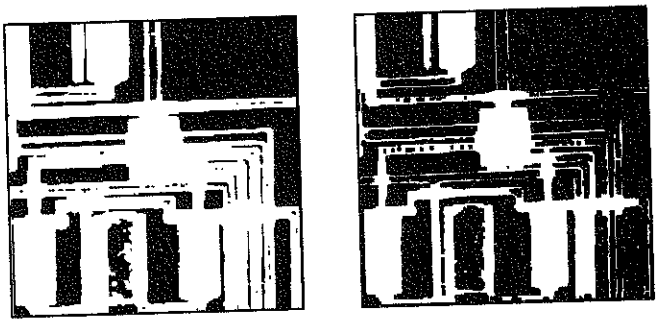
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Erosion

```
%% Dilation and erosion of real images %%
clear all
clc
a=imread('circbw.tif');
p=size(a);
%% Using the inbuilt MATLAB functions for comparision %%
s=strel('square',3);
d1=imdilate(a,s);
d2=imerode(a,s);
%% Writing our own program %%
w=[1 1 1;1 1 1;1 1 1]; %% Structuring element %%
for x=2:1:p(1)-1
for y=2:1:p(2)-1
    a1=[w(1)*a(x-1,y-1) w(2)*a(x-1,y) w(3)*a(x-1,y+1)...
      w(4)*a(x,y-1) w(5)*a(x,y) w(6)*a(x,y+1)...
      w(7)*a(x+1,y-1) w(8)*a(x+1,y) w(9)*a(x+1, y+1)];
    A1(x,y)=max(a1); % Dilation %
    A2(x,y)=min(a1); % Erosion %
end
end
figure(1),imshow(a)% Normalizing might be required
figure(2),imshow(d1)
figure(3),imshow(d2)
figure(4),imshow(A1)
figure(5),imshow(A2)
```



(a) Original image (b) Dilation using in-built function (c) Erosion using in-built function



(d) Dilation using program (e) Erosion using program

Fig. 10.5.1

Ex. 10.5.1

Given a 10 × 10 image, perform dilation using a structuring element

The image is a white block on a black background.

B = | 1 | 1 |
| 1 | 1 |

A =

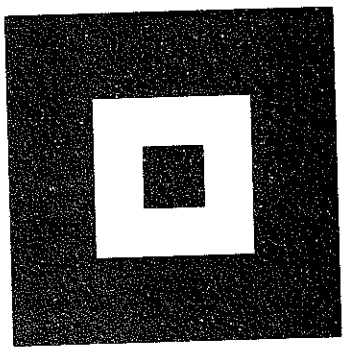


Fig. P. 10.5.1

Soln. :

A =

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

We start by placing the structuring element at the location shown.

We use formula

$$D(x, y) = \text{Maximum } \{A(x - i, y - j) \times B(i, j)\}$$

$$0 \le i \le m - 1$$

$$0 \le j \le n - 1$$

1	1
1	1

We multiply the coefficients of the structuring element with the corresponding image pixels and take the maximum value. This maximum value is placed at the origin position of B. Moving the structuring element all over the image, we get.

A ⊕ B =

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	1	1	1	1	1	1	1	0	0
3	0	1	1	1	1	1	1	1	0	0
4	0	1	1	1	1	1	1	1	0	0
5	0	1	1	1	0	1	1	1	0	0
6	0	1	1	1	1	1	1	1	0	0
7	0	1	1	1	1	1	1	1	0	0
8	0	1	1	1	1	1	1	1	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

The movement of the mask is similar to what we did in spatial filtering.

Now what have we got here ?

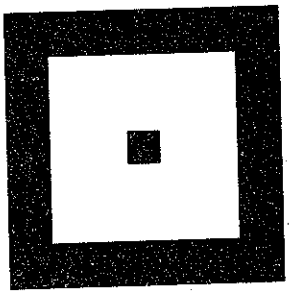


Fig. P. 10.5.1(a)

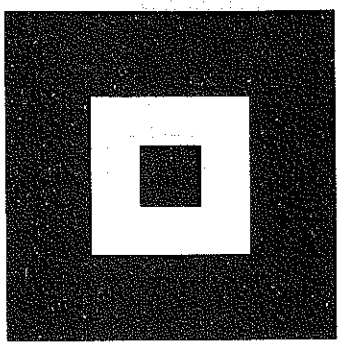
If we compare this image with the original image, we see that the size of the white block has increased and the black hole at the centre has reduced. We can thus conclude that Dilation fills the holes in the object and expands its boundaries, filling in any narrow creeks that might exist.

Ex. 10.5.2

Perform erosion on the same 10 × 10 image using the same structuring element.

Soln. :

A =



B =

1	1
1	1

Fig. P. 10.5.2

Just as in the earlier case, we place the structuring element at the left hand corner. For erosion, we use the formula,

$$E(x, y) = \text{Minimum } \{A(x - i, y - j) \times B(i, j)\}$$

$$0 \le i \le m - 1$$

$$0 \le j \le n - 1$$

A =

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Here, we multiply the mask coefficients with the image values and choose the minimum value. We place this minimum value at the position of the origin of the structuring element.

A⊖B =

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	1	1	0	0
0	0	1	0	0	0	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

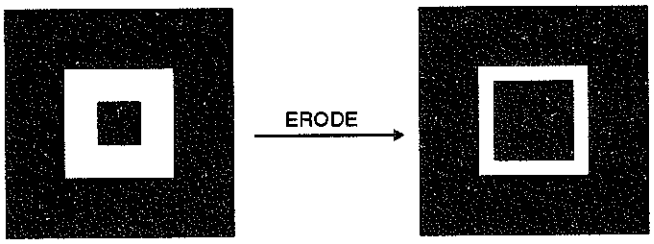


Fig. P. 10.5.2(a)

When we compare the two images, we see that the size of the white block has reduced and the black hole at the center becomes even bigger.

We can thus conclude that Erosion enlarges holes in the object, shrinks its boundary, eliminates islands and gets rid of narrow peninsulas that might exist on the boundaries. It gets rid of irrelevant data by reducing its size.

At this stage we should note that Dilation and Erosion are duals of each other, that is, the dilation of image A is equivalent of the Erosion of the complement of the image A. This simply means, that Dilation can be performed by Eroding the complement set by the same structuring element.

10.6 Opening and Closing Operations

Dilation and Erosion operations are fundamental to Image Morphology. In fact, many of the morphological algorithms that would be discussed now are based on these two operations.

10.6.1 Opening

Morphological opening of an image is basically Erosion followed by a Dilation, using the same structuring element.

If A is the image and B is the structuring element then, opening of A by B is given as

$$OPEN(A, B) = D(E(A)) \rightarrow D \rightarrow \text{Dilation} \quad ...(10.6.1)$$

$$E \rightarrow \text{Erosion}$$

It is also written as

$$A \circ B = (A \ominus B) \oplus B$$

Opening generally smoothes the contours of the image, breaks down narrow bridges and eliminates thin protrusions. Thus opening isolates objects which may be just touching one another. Opening is therefore used in analysis of wear particles in engine oil, ink particles in recycled paper and most importantly in study of cells in cytology.

10.6.2 Closing

Morphological closing of an image is basically Dilation followed by Erosion, using the same structuring element.

$$CLOSE(A, B) = E(D(A)) \quad ...(10.6.2)$$

It is also written as

$$A \bullet B = (A \oplus B) \ominus B$$

Closing generally tends to fuse narrow breaks and eliminates small holes. This simplifies the process of assessing the separation of particles.

Ex. 10.6.1

Perform the opening operation on the image shown. Use the structuring element

Image is of size 10×10

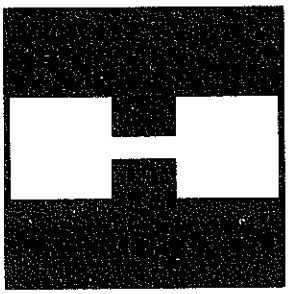


Fig. P. 10.6.1

Soln. :

The image raw data is shown.

Opening is basically Erosion followed by Dilation

$$Open = D(E(A))$$

$$A \circ B = (A \ominus B) \oplus B$$

$A =$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	1	1	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$B =$ ① / 1

Step 1 : Remember, Erosion is minimum

$$\{A(x - i, y - j) \times B(i, j)\}$$

Eroding A with B, we get

$A \ominus B =$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

We now dilating this image

$Open = (A \ominus B) \oplus B =$

0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

The modified image looks like

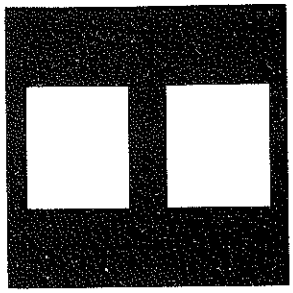


Fig. P. 10.6.1(a)

The original image had two white blocks which were connected by a thin white strip. Opening this image got rid of this strip. The size of the white blocks remain unchanged. Hence we see that opening breaks down narrow bridges or isolates objects which may just be touching one another.

Ex. 10.6.2

Perform closing operation on the image shown below. The size of the image is 10×11. The two white blocks are not connected if we consider 4-connectivity.

Use the same structuring element B =

(1)
1

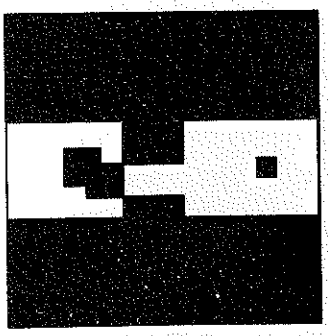


Fig. P. 10.6.2

Soln. : The image raw data is as shown. Note the two big blocks are not connected if we consider 4-connectivity.

A =

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	(0)	1	1
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

We know that closing is basically Dilation followed by Erosion.

Close = E(D(A)) $A \bullet B = (A \oplus B) \ominus B$

Remember,

Dilation = Maximum $\{A(x-i, y-j) \times B(i, j)\}$

Dilating A, we get

$A \oplus B =$

0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	1	1	1
1	1	1	1	0	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	1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Eroding $(A \oplus B)$ we have,

Closing = $(A \oplus B) \ominus B$

0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

The modified image looks like Fig. P. 10.6.2(a).

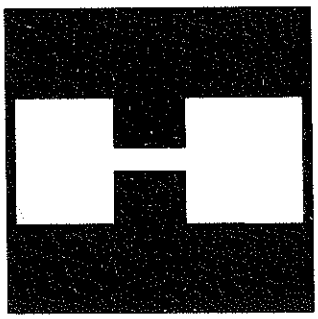
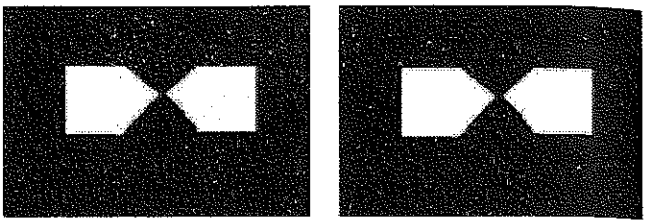


Fig. P. 10.6.2(a)

Comparing this image with the original image, we conclude that the closing operation tends to fuse narrow breaks and also eliminates small holes.

```
%% Program for opening %%
%% Opening=D(E(I)) i.e., Erode first and Dilate later
clear all
clc
a=imread('test1.bmp');
a=double(a);
p=size(a);
w=[1 1 1;1 1 1;1 1 1];      %% Structuring element %%
for x=2:1:p(1)-1
for y=2:1:p(2)-1
      a1=[w(1)*a(x-1,y-1) w(2)*a(x-1,y)  w(3)*a(x-1,y+1)...
          w(4)*a(x,y-1)   w(5)*a(x,y)    w(6)*a(x,y+1)...
          w(7)*a(x+1,y-1) w(8)*a(x+1,y)  w(9)*a(x+1,y+1)];
      A1(x,y)=min(a1);                  %% Erosion %%
end
end
q=size(A1)                  %% Modified size of the image %%
for i=2:1:q(1)-1
for j=2:1:q(2)-1
 a11=[w(1)*A1(i-1,j-1)  w(2)*A1(i-1,j)  w(3)*A1(i-1,j+1)...
          w(4)*A1(i,j-1)    w(5)*A1(i,j)    w(6)*A1(i,j+1)...
      w(7)*A1(i+1,j-1)  w(8)*A1(i+1,j)  w(9)*A1(i+1,j+1)];
      A2(i,j)=max(a11);           %% Dilation %%
end
end
figure(1),imshow(a)      %% Original %%
figure(2),imshow(A2)          %% Open %%
% Normalization might be required
```

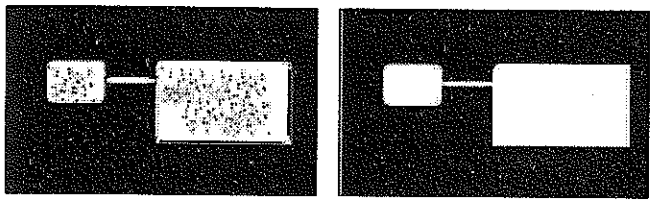


(a) Original image **(b) Image after opening**

Fig. 10.6.1

```
%% Program for closing %%
%% Closing=E(D(I)) i.e., dilate first and erode later
clear all
clc
a=imread('check.bmp');
a=rgb2gray(a);
a=double(a);
p=size(a);
w=[1 1 1;1 1 1;1 1 1];      %% Structuring element %%
for x=2:1:p(1)-1
      for y=2:1:p(2)-1
      a1=[w(1)*a(x-1,y-1) w(2)*a(x-1,y)  w(3)*a(x-1,y+1)...
          w(4)*a(x,y-1)   w(5)*a(x,y)    w(6)*a(x,y+1)...
      w(7)*a(x+1,y-1) w(8)*a(x+1,y)  w(9)*a(x+1,y+1)];
      A1(x,y)=max(a1);                  %% Dilation%%
      end
end
q=size(A1)
for i=2:1:q(1)-1
      for j=2:1:q(2)-1
```

```
a11=[w(1)*A1(i-1,j-1) w(2)*A1(i-1,j) w(3)*A1(i-1,j+1) ...
     w(4)*A1(i,j-1)  w(5)*A1(i,j)   w(6)*A1(i,j+1) ...
     w(7)*A1(i+1,j-1)w(8)*A1(i+1,j) w(9)*A1(i+1,j+1)];
     A2(i,j)=min(a11);
                    %% Erosion after dilation, closing %%
   end
end
figure(1), imshow(a)      %% Original %%
figure(2), imshow(A2)  %% Close using program %%
% Normalization might be required
```



(a) Original image **(b) Image after opening**

Fig. 10.6.2

10.7 Solved Example

Ex. 10.7.1

For the following given binary image first perform morphological opening operation and then successively apply morphological closing operation.

$$A = \begin{bmatrix} 1&0&1&0&1&0&1 \\ 1&1&0&1&1&0&1 \\ 1&1&1&1&1&1&1 \\ 1&1&1&1&1&0&0 \\ 1&1&0&1&1&1&1 \end{bmatrix} ; B = [\,(1)\ 1\,]$$

Soln. :

We first perform opening. Opening is given by the formula

$$A \circ B = (A \ominus B) \oplus B$$

i.e. $A \circ B = D[E(A)]$

We first perform Erosion followed by Dilation.

$$\therefore A \ominus B = \begin{bmatrix} 0&0&0&0&0&0&1 \\ 1&0&0&1&0&0&1 \\ 1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&0 \\ 1&0&0&1&1&1&1 \end{bmatrix} \uparrow$$

Last column is retained

Now

$$\text{Open} = [A \ominus B] \oplus B = I \begin{bmatrix} 0&0&0&0&0&1&1 \\ 1&0&1&1&0&1&1 \\ 1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&0 \\ 1&0&1&1&1&1&1 \end{bmatrix} \uparrow$$

Last column is retained

We now perform Closing on the image given in Equation (A).

$$\text{Closing} = A \cdot B = (A \oplus B) \ominus B$$

$$\text{Closing} = E\,(D\,(A))$$

Summary

Morphology, representation and description are an integral part of image analysis. Morphological applications offer a powerful approach for extracting objects from an image. Basic morphological operations such as Dliation, Erosion, Opening and Closing have been discussed.

Review Question

Q. 1 Explain the following operations :

(a) Erosion (b) Dilation

(c) Opening (d) Closing

Chapter Ends...

❑❑❑

CHAPTER 11

Colour Image Processing

11.1 Introduction

The preceding chapters dealt with grey scale images. We have also studied various algorithms to modify these greyscale images. This chapter deals with colour images. Human beings can identify thousands of colour shades and intensities as opposed to a few shades of grey. This make colour image processing a very important component of image processing. Colour is also often used as descriptor for object identification.

The basic structure of the human eye was discussed in chapter of Introduction. It was stated there that the retina is covered with photoreceptor cells (rods and cones). Rods provide us with monochromatic night vision while cones are responsible for colour vision.

It has been experimentally established that there are about 6 to 7 million cones in the human eye. These cones occur in three types, differing mainly in the photochemistry they employ to convert light into nerve impulses.

These cones divide the visible portion of the electromagnetic spectrum into three bands viz. Red, Green and Blue. These three colours are hence called the primary colours.

As shown in Fig. 11.1.1 there are three curves which give us the absorption of light by the red, green and blue cones in a human eye as a function of wavelength.

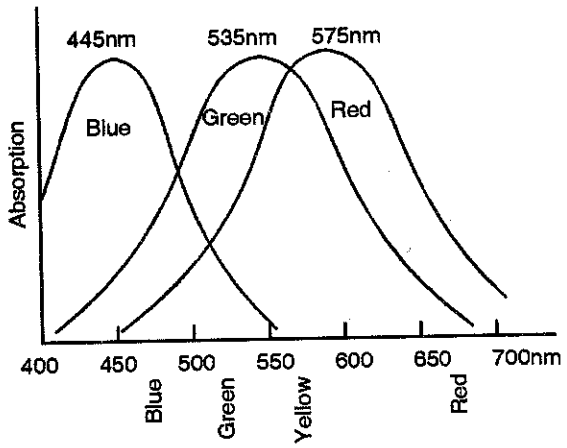


Fig. 11.1.1

Because of the anatomy of the eye, majority of the development has been devoted to tricolour systems (RGB system).

A classic example of a tri-colour system is the television. In this, the visible spectrum is divided into three colour bands Red, Green and Blue by using colour filters.

Fig. 11.1.2 is simple block diagram of a colour television camera showing generation of colour signals.

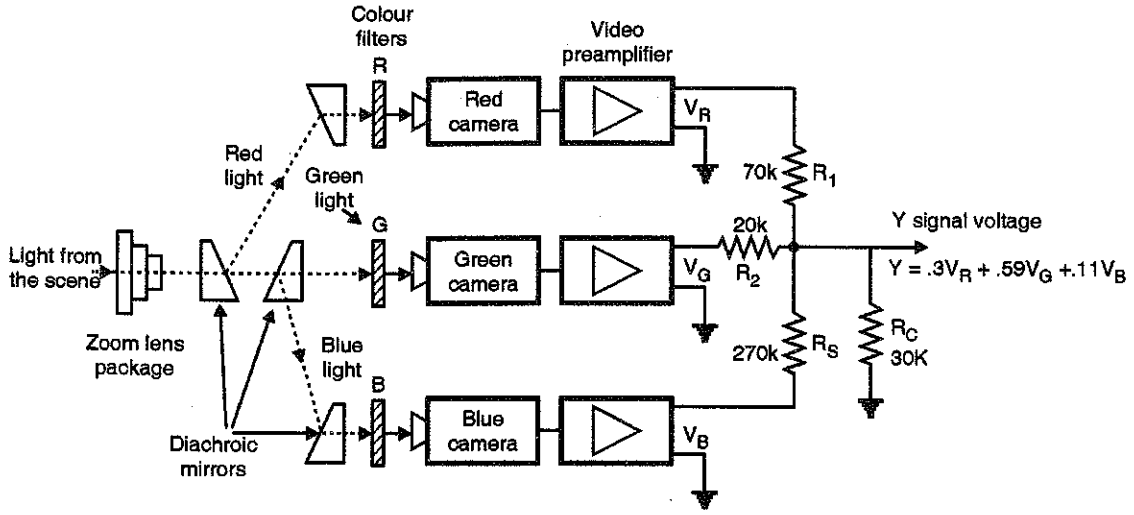


Fig. 11.1.2

The colour TV camera consists of three camera tubes. Each of these three cameras receives a filtered primary colour.

Light from the scene is split up into 3 primary colours by using three prisms. These prisms are designed as diachroic mirror. The diachroic mirror passes only one wavelength and rejects the others thus forming red, green and blue colour images. Rays from each of these diachroic mirrors is now passed through colour filters, know as trimming filters. These are then converted to video signals by the Red, Green and Blue camera tubes. Thus generating 3 colours signals or RGB signals.

Simultaneous scanning of the three camera tubes is accomplished by a master deflection oscillator and a sync generator which drives all the three tubes. Coming back to our discussion, majority of the developments have been devoted to tricolour systems-RGB system.

11.2 Colour Models

This is also known as colour space. Colour models are different ways in which colour information is stored. The most common and obvious colour model in the RGB colour model. The reason for having more than one colour model is that some operations are easier to implement if we move away from the RGB model. In the next section we discuss some important colour models.

11.2.1 RGB Colour Model

As stated earlier, this is the most common format used in image processing.

The RGB colour space is shown in Fig. 11.2.1.

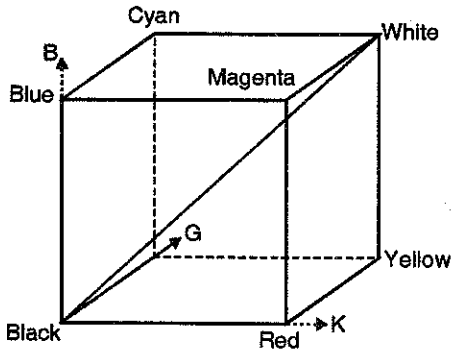


Fig. 11.2.1

Each point in the image (pixel) is represented by a point in the first quadrant of the three-space as shown.

The origin of the RGB space (Red = 0, Green = 0, Blue = 0) represents black while the opposite corner (Red = max, Green = max, Blue = max) represents white.

Line joining the two corners have equal values of Red, Blue and Green. This produces various shades of grey. The locus of all these points is called the grey line.

In the RGB model, each pixel is composed of RGB values and each of these colours require 8-bit for its representation (Full colour image). Hence each pixel is represented by 24 bits.

[R (8-bits), G (8-bits), B (8-bits)]

Total number of colours possible with a 24-bit RGB image is $(2^8)^3 = 16, 777, 216$.

A typical way of representing a RGB colour image is $M \times N \times 3$ where $M \times N$ is the physical size of the image and 3 represents the R, G, B planes i.e. an RGB image can be visualised as stack of 3 planes of size $M \times N$ each.

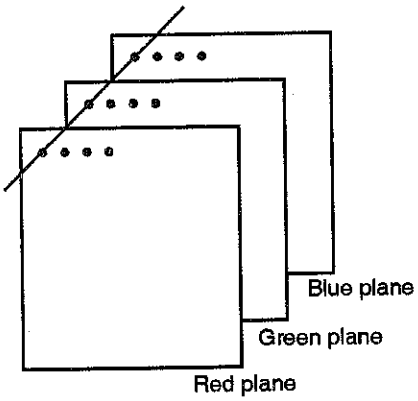


Fig. 11.2.2

If we read a colour image in Matlab and display its first pixel, this is what we get

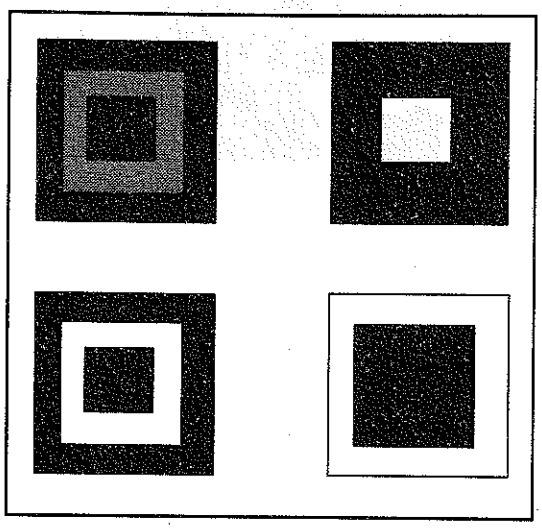
```
>>  a  = imread ('autumn.tif');
>>  a (1,1,1)  = 231
>>  a (1,1,2)  = 231
>>  a (1,1,3)  = 207
```

Here the last terms (1, 2 and 3) represent the RGB components of the pixel (1, 1).

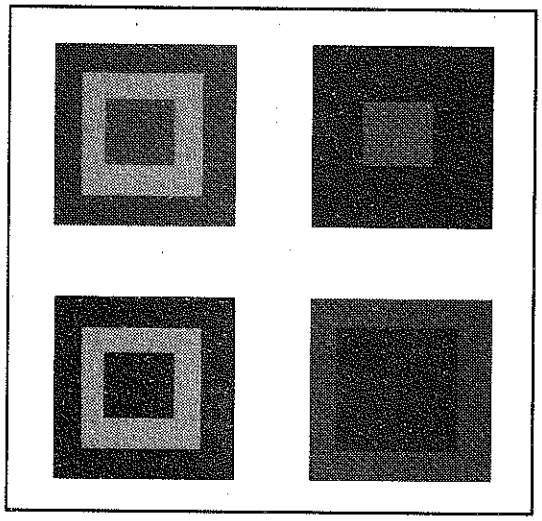
It is interesting to plot the Red, Green and Blue planes separately. Each of them appears as a gray scale image since the R, G, B planes are all 8-bits. To display each plane as a colour image, we need to display each of the planes a 24 bit images. This is done in the lower part of the program. For colour images, please refer at the end of the book.

```
%% RGB Planes

clear all

clc

a=imread('autumn.tif');

%a=imread('pallette.bmp');

a=double(a);

[row col dim]=size(a);


red=a(:,:,1);   %This would give us a grey scale image of the
                red plane

green=a(:,:,2);   %This would give us a grey scale image of
                  the green plane

blue=a(:,:,3);    %This would give us a grey scale image of
                  the blue plane


%% To display the individual planes in colour,

%% we need to store them as a 24- bit image

%% This is what is done below


plane=zeros(row,col);

RED=cat(3,red,plane,plane);

GREEN=cat(3,plane,green,plane);

BLUE=cat(3,plane,plane,blue);

figure(1)

subplot(2,2,1)

imshow(uint8(a))

subplot(2,2,2)

imshow(uint8(red))
```

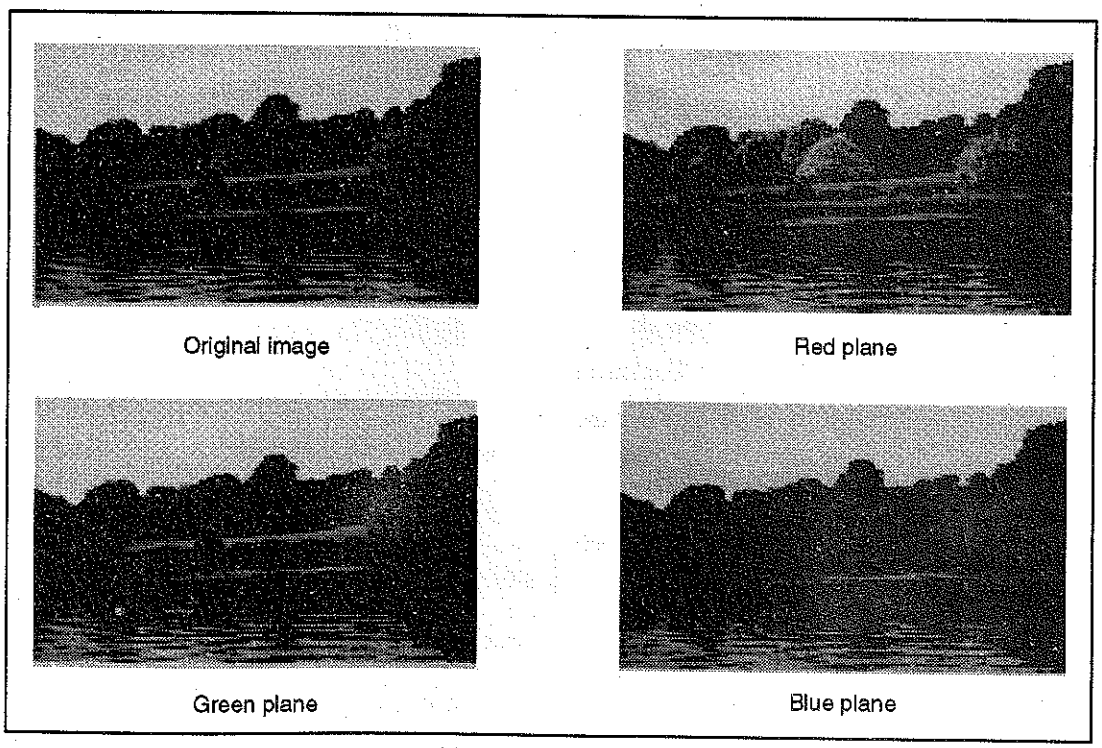
```
subplot(2,2,3)
imshow(uint8(green))
subplot(2,2,4)
imshow(uint8(blue))

figure(2)
subplot(2,2,1)
imshow(uint8(a))
subplot(2,2,2)
imshow(uint8(RED))
subplot(2,2,3)
imshow(uint8(GREEN))
subplot(2,2,4)
imshow(uint8(BLUE))
```



(a) Grey scale image



(b) Colour image



(c) Grey scale image

Fig. 11.2.3 Continued...

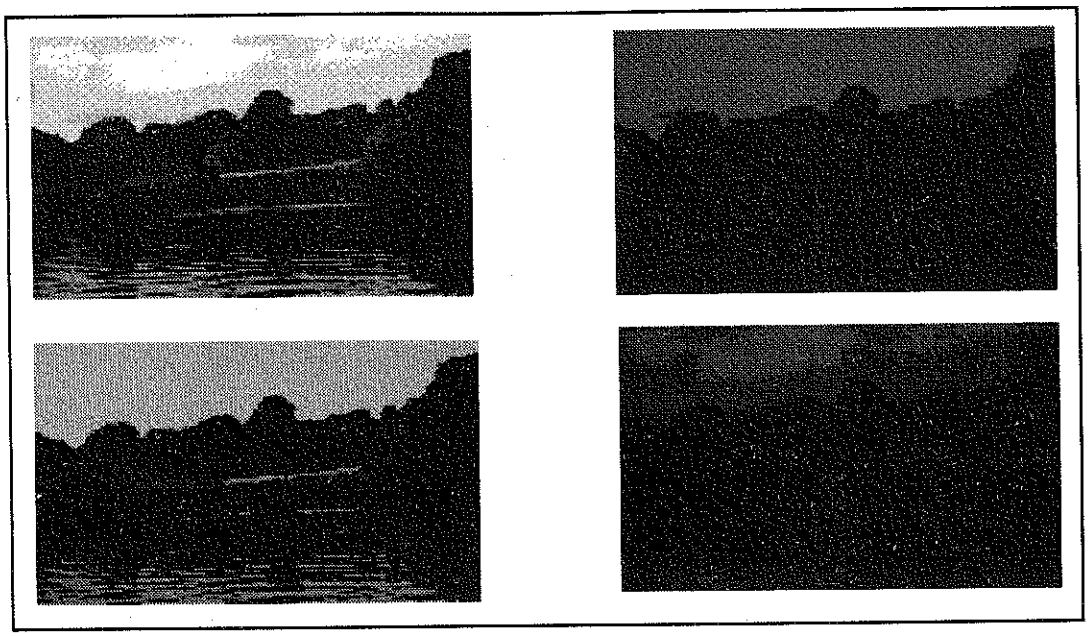


Fig. 11.2.3(d) : Colour image
(For Colour Images please refer Fig. 1(a), 1(b), 1(c) and 1(d) at the end of the book)

Apart from the RGB colour model, there are various other colour models that we shall now discuss.

The colour models that one comes across in literature are NTSC, YCbCr, CMY, CMYK and HSI.

11.2.2 NTSC Colour Model

The NTSC colour model is used in television in the United States. That is the reason why their VCD's do not normally work here, as we use the PAL format.

Similar to the RGB model, the NTSC model consists of three components viz; Luminance (Y), Hue (I) and Saturation (Q). In this model, the Luminance component represents the grey scale information while the Hue and the Saturation carry the colour information.

Computing the YIQ components from the RGB model is a simple task, Given below is the Transformation.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \ldots(11.2.1)$$

It is important to note that the sum of the first row is equal to 1, while the sum of the remaining two rows is 0.

Matlab has is inbuilt function to compute the YIQ components from an RGB model.

```
>> a = imread ('autumn tif');
>> YIQ = rbg2ntsc(a);
```

Here YIQ would be a M × N × 3 image where

YIQ (:, :, 1) will be the Luminance (Y)

YIQ (:, :, 2) will be the Hue (I)

YIQ (:, :, 3) will be the Saturation (Q)

Given below is the Matlab program required to compute the YIQ components.

```
%% RGB to NTSC
clc
clear all
a=imread('autumn.tif');
a=double(a);
```

```
[row col dim]=size(a);

red=a(:,:,1);
green=a(:,:,2);
blue=a(:,:,3);
Y=(0.299*red)+(0.587*green)+(0.114*blue);
I=(0.596*red)-(0.274*green)-(0.322*blue);
Q=(0.211*red)-(0.523*green)+(0.312*blue);

figure(1)
imshow(uint8(a))
colormap(gray)

figure(2)
imagesc(Y)
colormap(gray)

figure(3)
imagesc(I)
colormap(gray)

figure(4)
imagesc(Q)
colormap(gray)

%% Check result using in built command rgb2ntsc
```

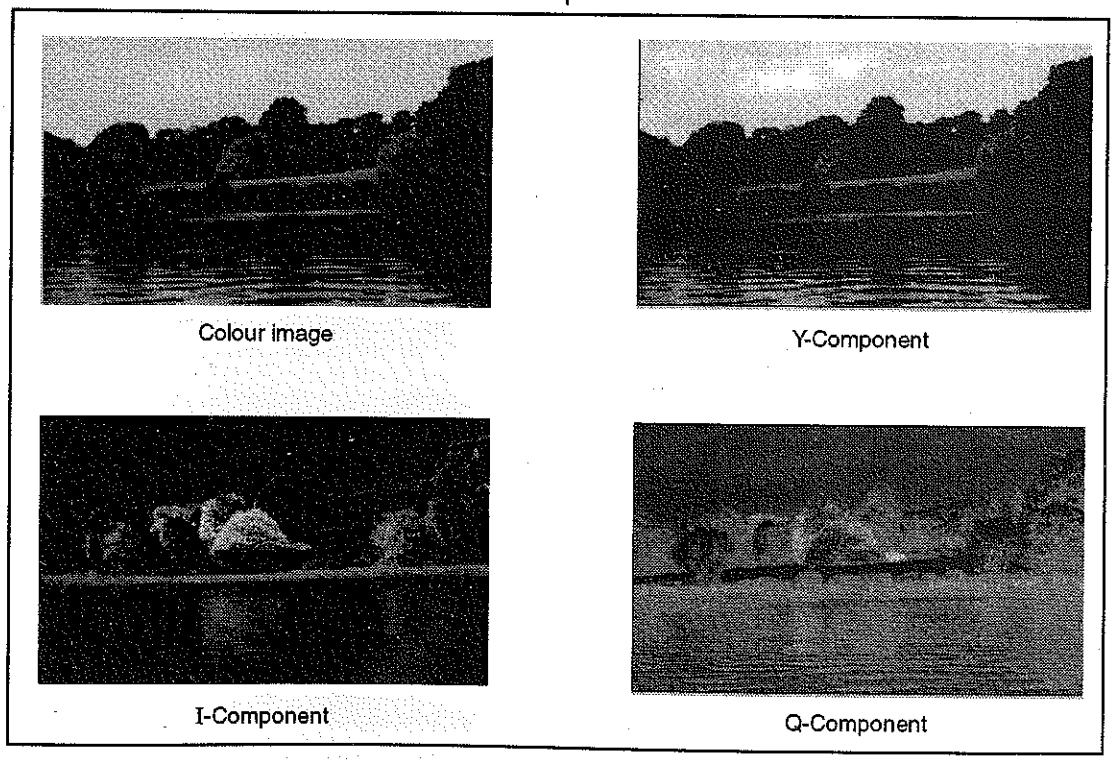


Fig. 11.2.4 : Colour image (Please refer Fig. 2 at the end of the book)

In a similar fashion, it is a simple task to get the RGB components from the YIQ components. The transformation that achieves this is given below,

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.621 \\ 1 & -0.272 & -0.647 \\ 1 & -0.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \quad \ldots(11.2.2)$$

Matlab has an inbuilt function for the transformation kindly go through the *rgb2ntsc* and *ntsc2rgb* commands in Matlab.

11.2.3 YCbCr Colour Model

This is another model which is common. It is widely used in digital video. Here Y stands for illumination and is represented by a single component. Colour information is stored as two-colour difference components, Cb and Cr.

Here Cb is the difference between the blue component and a reference value while Cr is the difference component between the red component and a reference value.

Transforming RGB values to YCbCr values is a trivial task. It is given by the transformation.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.00 \\ 112.00 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \ldots(11.2.3)$$

Matlab has an inbuilt function to achieve this, kindly go through commands rgb2ycbcr and ycbcr2rgb.

```
%% RGB to YCbCr
clc
clear all
a=imread('autumn.tif');
a=double(a);
[row col dim]=size(a);
red=a(:,:,1);
green=a(:,:,2);
blue=a(:,:,3);
Y=16+((65.481*red)+(128.553*green)+(24.966*blue));
Cb=128+((-37.797*red)-(74.203*green)+(112*blue));
Cr=128+((112*red)-(93.786*green)-(18.214*blue));

figure(1)
imshow(uint8(a))
colormap(gray)

figure(2)
imagesc(Y)
colormap(gray)

figure(3)
imagesc(Cb)
colormap(gray)

figure(4)
imagesc(Cr)
colormap(gray)
%% Check result using in built command rgb2ycbcr
```

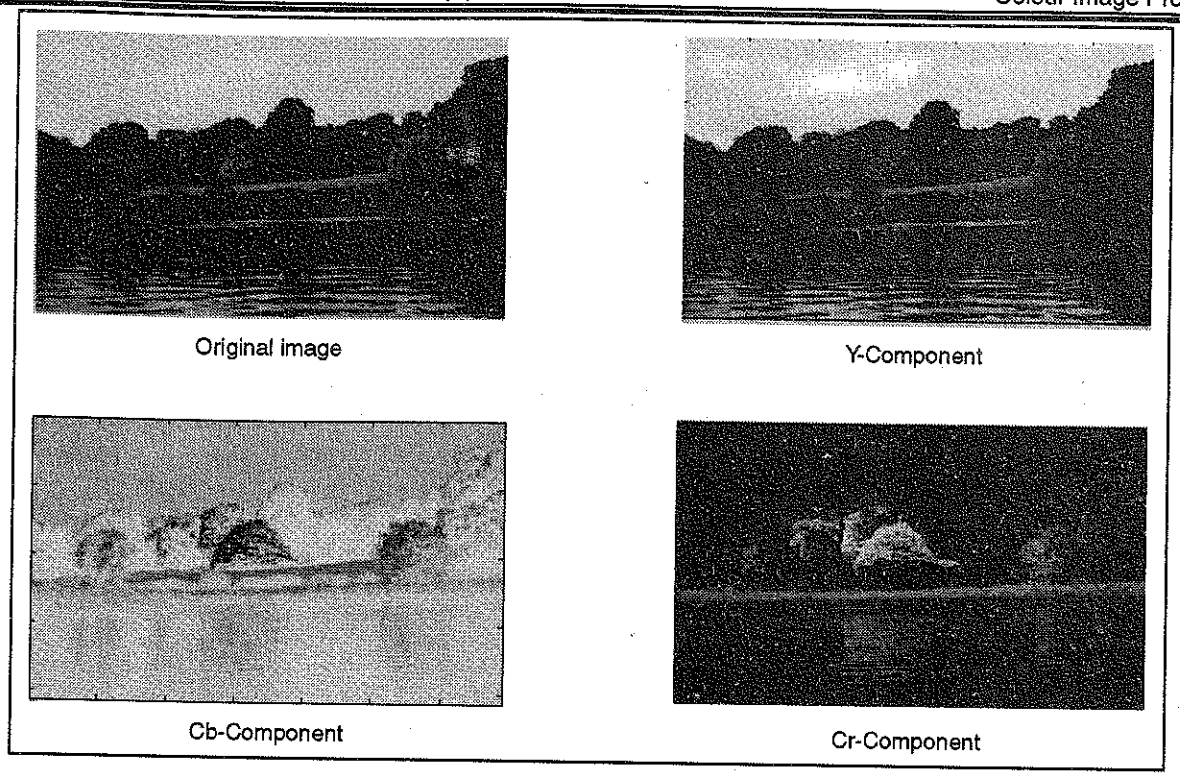


Fig. 11.2.5 : Colour image (Please refer Fig. 3 at the end of the book)

11.2.4 CMY and CMYK Models

Here C stands for cyan, M for magenta and Y for yellow. Cyan, Magenta and Yellow are secondary colours of light. While television, cameras, scanners, computer monitors work an the RGB colour model, which is a additive colour system, offset printing, digital printing, paint, photographic prints to are based on the CMY or CMYK system which is the subtractive system of colour.

C, M and Y mix to form Black (k)

When a surface coated with cyan pigment is illuminated with white light, no red light is in reflected from the surface, in other words cyan pigment subtracts red light from the reflected white light.

Transformation from RGB to CMY is a trivial task.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \ldots(11.2.4)$$

Here R, G and B values are assumed to be normalised. (for normalisation, divide the R, G, B planes by 256).

This equation simply means that light reflected from a surface coated with cyan pigment does not contain red. Similarly light reflected from surfaces coated with magenta and yellow pigments do not contain green and blue light respectively.

The only difference between the CMY format and CMYK format is K, which stand for Black. Hence the CMYK is a four colour format i.e. 3 colours of CMY and the fourth Black colour.

Matlab has an inbuilt command to compute the CMY image i.e. *incomplement*.

```
%% RGB to YCbCr
clc
clear all
a=imread('autumn.tif');
a=double(a);
[row col dim]=size(a);
```

```
red=a(:,:,1)/256;

green=a(:,:,2)/256;

blue=a(:,:,3)/256;


C=1-red;

M=1-green;

Y=1-blue;


figure(1)

imshow(uint8(a))

colormap(gray)
```

```
figure(2)

imagesc(C)

colormap(gray)


figure(3)

imagesc(M)

colormap(gray)


figure(4)

imagesc(Y)

colormap(gray)


%% Check result using in built command imcomplement
```

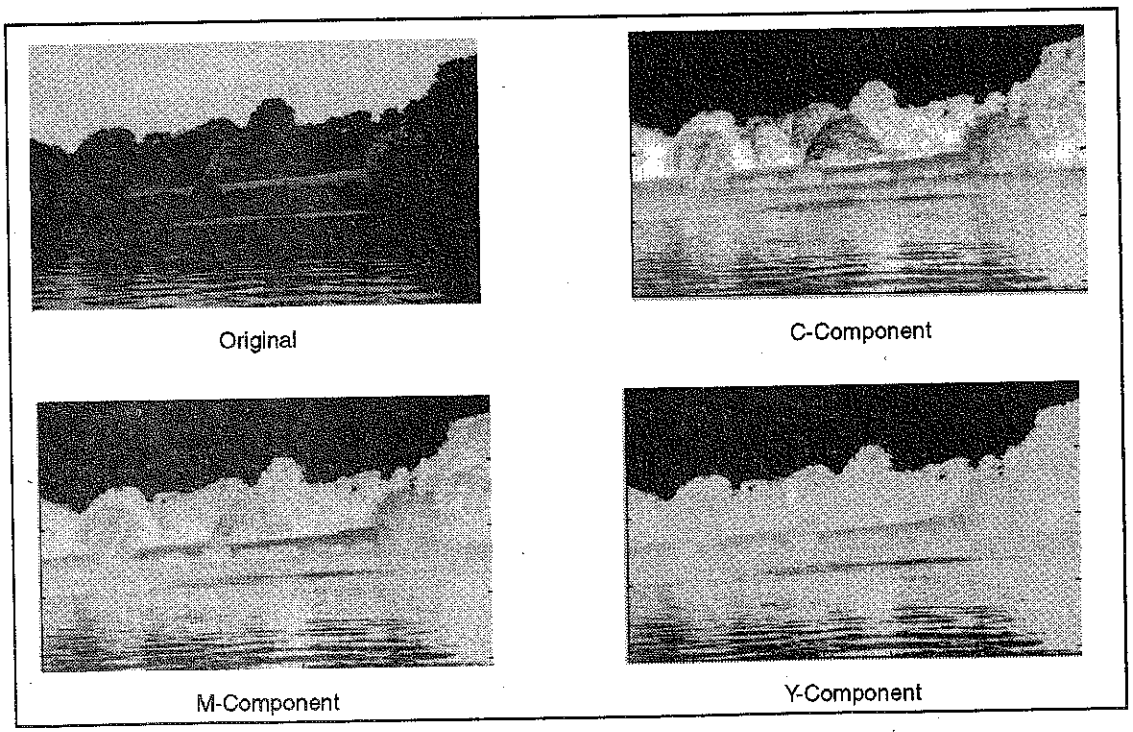


Fig. 11.2.6 : Colour image (Please refer Fig. 4 at the end of the book)

11.2.5 HSI Colour Model

HSI is an important colour model and it's design reflects the way humans see colour. For example, one does not refer to the colour of a dress by giving the percentage of the red, green and blue components!!

When we see colour, we describe it in terms of its Hue, Saturation and Intensity (Brightness).

In the HSI model, I stands for intensity and for our purpose it is simply the average of the R, G and B components. The I component specifies the brightness irrespective of the colour H stands for Hue. Hue is an attribute that describes pure colour. Hue is what the artist refers to as "pigment" it is what we think as colour. Yellow, orange, cyan etc. are examples of Hue. S stands for Saturation. This gives us the measure of the degree to which pure colour is diluted by white light.

The HSI colour model decouples the Intensity component from Hue and Saturation which are the colour carrying components. As a result, the HSI model is an ideal tool for developing image processing algorithms. Hue and saturation which are the colour information parameters can be illustrated by the colour circle.

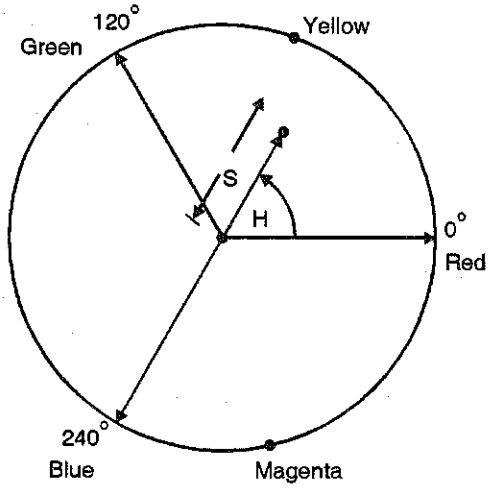


Fig. 11.2.7

Hue is expressed as an angle. Generally, a Hue of 0° is red, 120° is green and 240° is blue, Hue traverses the colours of the visible spectrum as it goes from Red to Blue i.e. from 0° to 240°. Between 240° and 360° fall the non-spectral colours (purple) that the eye perceives. The saturation parameter is the length of the segment joining the point to the center of the colour circle.

The concept of saturation is fairly easy to understand. Imagine we want to paint our house and we have a bucket of pure Red paint. At this stage we are at 0° of the colour circle. Hence this pure red corresponds to a hue of 0° and a saturation of 1. If we mix white paint in the bucket, we are actually reducing its saturation. Red becomes pink. Pink would probably correspond to a saturation of about 0.5. As we add more and more white paint, eventually, the colour in the bucket would become white.

Similarly if we add some black to the original bucket of Red, we reduce its intensity (Making it dark red). If we keep on adding Black, eventually the entire colour in the bucket would become black.

Red and Pink are different saturations of the same Hue-Red. HSI is a great format to change from one colour to another. For example, to change cyan to magenta, only H needs to be changed. In the RGB format, we would have to change all the three R, G and B values.

The colour circle that was shown is actually a part of a bigger image.

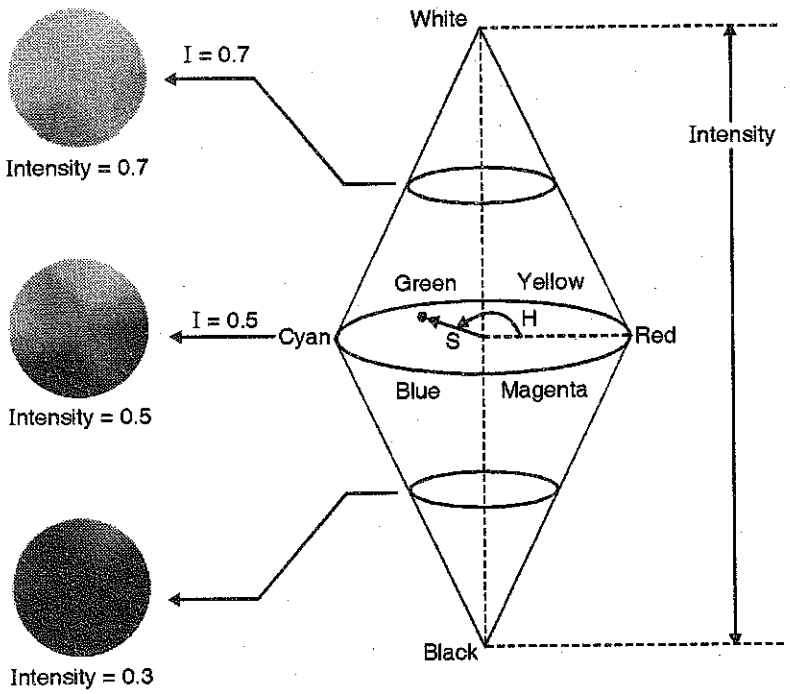


Fig. 11.2.8 : Colour image (Please refer Fig. 5 at the end of the book)

It is important to note that Hue is the angle which starts form Red. Hence H for red will be almost zero.

As we move from red, Hue (angle) starts increasing. Hence the Hue component for green will be larger than the Hue component for yellow.

To see the Hue values go to paint, click the colour tab on the menu and go to Edit colours. Now click on Define custom colours.

☞ Converting colour from RGB to HSI

Given an image in the RGB colour format, the H, S and I components are obtained using the following equations.

$$H = \begin{cases} \theta & \text{if } B \le G \\ 360 - \theta & \text{if } B > G \end{cases} \qquad \ldots(11.2.5)$$

Here,

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\}$$

Remember, θ is measured with respect to the Red axis

$$S = 1 - \frac{3}{(R+G+B)}\,[\min(R, G, B)]$$

$$I = \frac{R+G+B}{3}$$

☞ Converting HSI to RGB

The formula for converting HSI to RGB take on a different form. The values of RGB depend on the value of H. Look at the colour circle.

For $0° \le H < 120°$ (Section RED - GREEN)

$$B = I(1-S)$$

$$R = I\left[1 + \frac{S\cos H}{\cos(60° - H)}\right]$$

$$G = 3I - (R+B) \qquad \ldots(11.2.6)$$

For $120° \le H < 240°$ (Section GREEN - BLUE)

In this range, we first subtract 120°from H

i.e., $H = H - 120$

$$R = I(1-S)$$

$$G = I\left[1 + \frac{S\cos H}{\cos(60° - H)}\right]$$

$$B = 3I - (R + G)$$

For $240° \le H \le 360°$ (Section BLUE - RED)

In this range, we subtract 240° from H and then use the same equation,

i.e., $H = H - 240$

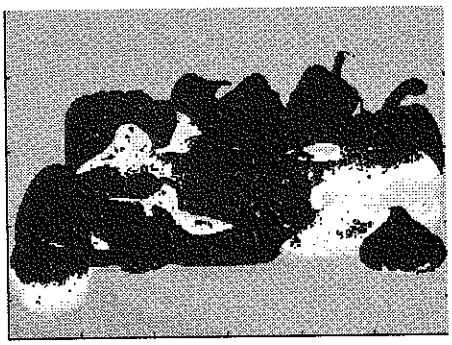
$$G = I(1 - S)$$

$$B = I\left[1 + \frac{S \cos H}{\cos(60° - H)}\right]$$

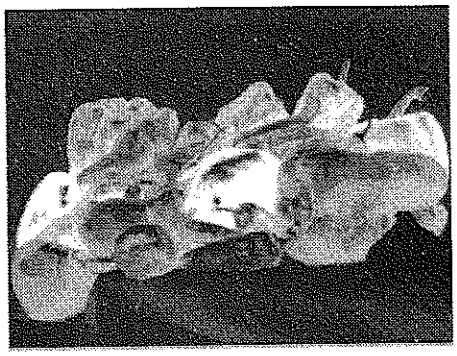
$$R = 3I - (G + B)$$

```
%% RGB to HSI
clc
clear all
a=imread('peppers.bmp');

a=double(a);
[row col dim]=size(a);

red=a(:,:,1);
[row col]=size(red);
green=a(:,:,2);
blue=a(:,:,3);

numer=0.5*((red-green)+(red-blue));
deno=sqrt((red-green).^2+(red-blue).*(green-blue));
theta=acos(numer./(deno+eps));

%%Computing Hue
for x=1:1:row
   for y=1:1:col
      if green(x,y)<= blue(x,y)
         H(x,y)=(2*pi)-theta(x,y);
      else
         H(x,y)=theta(x,y);
      end
   end
end

%% Computing Saturation and Intensity

numer=min(min(red,green),blue);
deno=red+green+blue;
deno(deno==0)=eps;
S=1-(3.*numer./deno);

I=(red+green+blue)/3;

figure(1)
imshow(uint8(a))
colormap(gray)

figure(2)
imagesc(H)
colormap(gray)

figure(3)
imagesc(S)
colormap(gray)
figure(4)
imagesc(I)
colormap(gray)
```

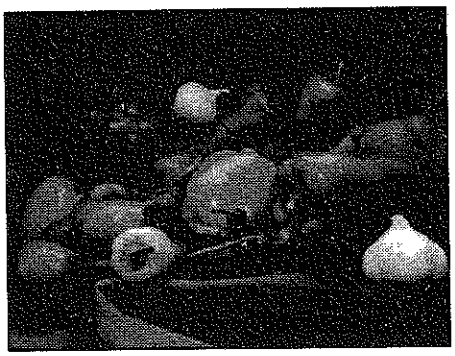
Original image



HUE component



SATURATION component



INTENSITY component

Fig. 11.2.9 : Colour image (Please refer Fig. 6 at the end of the book)

11.3 Pseudo-Colouring

Before getting into the domain of colour image processing, it is important as well as interesting to understand pseudo-colouring in grey scale images. As the name suggests, pseudo, simply means take; hence fake colouring.

It is know that the human eye can discern thousands of colours and intensities as opposed to only one or two dozen shades of grey.

Hence in pseudo colouring, grey scale images are given colour based on their values of the grey levels.

A 2-colour pseudo colour transformation is shown in Fig. 11.3.1.

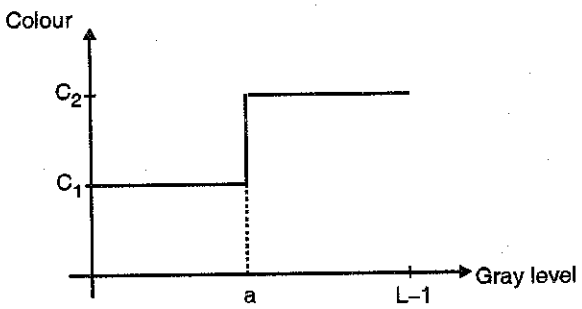


Fig. 11.3.1

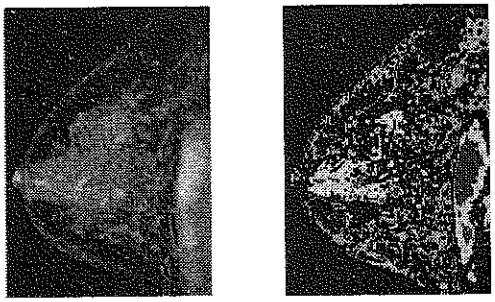
Here, all pixel values below grey level a are assigned a colour C_1 while all grey levels above grey level a are assigned a colour C_2.

C_1 and C_2 are the two colours that the user defines (they could be anything).

Instead of using to 2-colour pseudo colouring, one could use many different colours.

Given below is program which achieves pseudo colouring.

```
%%Pseudo colouring

clc

clear all


%a=imread('pseudo11.bmp');

a=imread('mri.bmp');

[row col]=size(a);

for x=1:1:row

  for y=1:1:col

    if 0<=a(x,y)&& a(x,y)<=06

      red(x,y)=0;

      blue(x,y)=0;

      green(x,y)=0;

    else if 7<=a(x,y)&& a(x,y)<=45

      red(x,y)=0;

      blue(x,y)=0;

      green(x,y)=255;

    else if 46<=a(x,y)&& a(x,y)<=66

      red(x,y)=0;

      blue(x,y)=255;

      green(x,y)=0;

    else if 67<=a(x,y)&& a(x,y)<=70

      red(x,y)=255;

      blue(x,y)=0;

      green(x,y)=200;

    else if 71<=a(x,y)&& a(x,y)<=255

      red(x,y)=255;

      blue(x,y)=0;

      green(x,y)=0;

    end

  end

 end

end

end

end

final=cat(3,red,green,blue);

figure(1)

imshow(a)

figure(2)

imshow(final)
```



Original image Pseudo coloured

Fig. 11.3.2 : Colour image (Please refer Fig. 7 at the end of the book)

11.4 Full-Colour Image Processing

We have so far seen what we mean by colour images. We have also discussed various colour models and their conversions. We shall now discuss some of the processing techniques applicable on colour images.

We are aware that a colour image in its basic form is a 24-bit image composed of 3 planes viz., Red, Green and Blue.

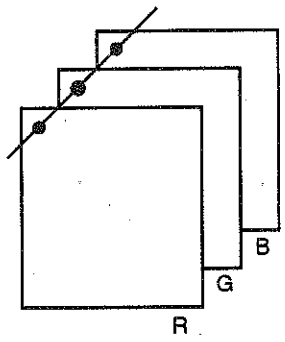


Fig. 11.4.1

There are two different approaches to colour image processing.

1. We process each of the 3 colour plane individually and then combine the modified 3 planes to get a processed colour image. This is similar to grey scale processing as each of the 3 planes is 8 bit.

2. We directly process the colour pixels. Here we consider colour pixel to be vectors. If we observe the colour cube (Fig. 11.2.1), one can interpret a colour pixel as a vector extending from the origin to any point in the RGB coordinate system.

   Hence

$$c = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\text{Or} \quad c(x, y) = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix} \qquad \ldots(11.4.1)$$

It is important to note that the results of two methodologies is not always the same.

Similar to grey scale image processing, let f(x, y) be the original colour image, g(x, y) be the modified colour image and T be the transformation function.

$$\therefore g(x, y) = T[f(x, y)]$$

The difference lies in the fact that pixel values here are triplets (RGB, HSI, CMY) or quartets (CMYK).

The earlier equation can also be written in the form

$$s_i = T_i(r_1, r_2, \ldots, r_n) \qquad i = 1, 2, 3, \ldots, n$$

for a RGB colour processing model, n = 3.

$$\therefore s_i = T_i(r_1, r_2, r_3) \qquad i = 1, 2, 3$$

Here r_i are the variables denoting colour of the input colour image while s_i are the variables denoting colour of the output colour image.

In theory, any transformation T_i, can be used on any colour model, however, some transformations are easy on certain models.

For example, suppose we want to change the intensity of an image by a factor k.

$$\text{i.e. } g(x, y) = k \cdot f(x, y)$$

If we use this transformation on a RGB colour model, we would need to transform the R, G and B plane.

$$\text{i.e. } s_i = k \cdot r_i \qquad i = 1, 2, 3$$

If we use the HSI model instead, we would only need to transform the I plane. The H and S plane would remain unchanged.

$$\text{i.e. } s_3 = k \cdot r_3$$

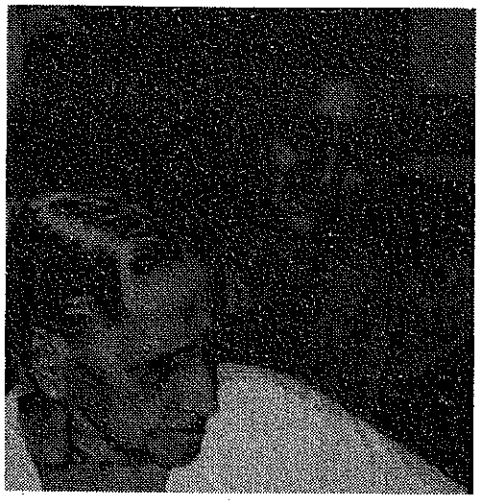
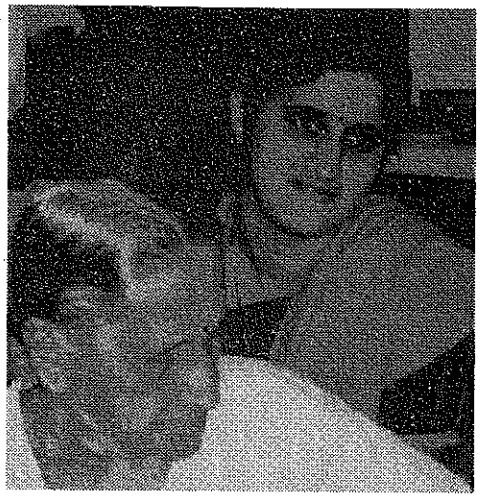


Image with poor intensity



Modifying only the I plane and leaving the H and the S planes intact

Fig. 11.4.2 : Colour image (Please refer Fig. 8 at the end of the book)

Similarly we could use various transformations which work on single pixels.

11.4.1 Colour Image Smoothing (Low Pass Averaging)

We have already understood low pass filtering (Averaging) in the spatial domain. We used a 3 × 3 mask shown below to implement the filter.

$$\frac{1}{9} \times \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

This concept can be easily extend to colour images

$$\bar{c}(x, y) = \frac{1}{K} \sum_{(x, y) \in S_{xy}} c(x, y) \qquad \ldots(11.4.2)$$

Here S_{xy} is the neighbourhood.

$\bar{c}(x, y)$ is the modified image while c (x, y) is the input image. From Equation (11.4.1)

$$c(x, y) = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix}$$

$$\therefore \bar{c}(x, y) = \begin{bmatrix} \frac{1}{K} \sum_{x, y \in S_{xy}} R(x, y) \\ \frac{1}{K} \sum_{x, y \in S_{xy}} G(x, y) \\ \frac{1}{K} \sum_{x, y \in S_{xy}} B(x, y) \end{bmatrix}$$

This transformation can be achieved using either of the two modalities, i.e. by independently smoothing each plane or by averaging the RGB vectors.

Given below is the program in which smoothing is achieved by averaging the 3 colour planes independently.

```
%% Colour Image Smoothing with 3 × 3 averaging mask

clc
clear all
a=imread('peppers.bmp');
a=double(a);
[row col dim]=size(a);
w=[1 1 1;1 1 1;1 1 1]/9;
red=a(:,:,1);
```

```
green=a(:,:,2);

blue=a(:,:,3);

for x=2:1:row-1

  for y=2:1:col-1

     red_f(x,y)=w(1,1)*red(x-1,y-1)+
                w(1,2)*red(x-1,y)+w(1,3)*red(x-1,y+1)+...

 w(2,1)*red(x,y-1)+w(2,2)*red(x,y)+w(2,3)*red(x,y+1)+...

 w(3,1)*red(x+1,y-1)+w(3,2)*red(x+1,y)+w(3,3)*red(x+1,y+1);


 green_f(x,y)= w(1,1)*green(x-1,y-1)+
               w(1,2)*green(x-1,y)+
               w(1,3)*green(x-1,y+1)+...
               w(2,1)*green(x,y-1)+w(2,2)*green(x,y)+
               w(2,3)*green(x,y+1)+...
               w(3,1)*green(x+1,y-1)+
               w(3,2)*green(x+1,y)+
               w(3,3)*green(x+1,y+1);

blue_f(x,y)= w(1,1)*blue(x-1,y-1)+
             w(1,2)*blue(x-1,y)+w(1,3)*blue(x-1,y+1)+...
```

```
w(2,1)*blue(x,y-1)+w(2,2)*blue(x,y)+

w(2,3)*blue(x,y+1)+...

w(3,1)*blue(x+1,y-1)+w(3,2)*blue(x+1,y)+

w(3,3)*blue(x+1,y+1);


  end

end


final=cat(3,red_f,green_f,blue_f);


figure(1)

imshow(uint8(a))

figure(2)

imshow(uint8(final))
```

Original image

Image smoothing using averaging mask

Fig. 11.4.3 : Colour image (Please refer Fig. 9 at the end of the book)

11.4.2 Colour Image Sharpening (High Pass Filtering)

We have seen high pass filtering in the special domain. We used different masks which gave us prominent edges.

We use a Laplacian operator to perform colour image sharpening. In a RGB colour system we have

$$\nabla^2 [c(x, y)] = \begin{bmatrix} \nabla^2 R(x, y) \\ \nabla^2 G(x, y) \\ \nabla^2 B(x, y) \end{bmatrix}$$

As for image smoothing, we can perform sharpening by implementing the Laplacian on the 3 colour planes individually. We could use a Laplacian mask shown below :

0	−1	0
−1	4	−1
0	−1	0

If we are interested in preserving the low frequency components, we make sure that the sum of the coefficients of the mask is not zero. We hence use the mask given below

0	−1	0
−1	5	−1
0	−1	0

```
%% Colour Image Sharpening

clc
clear all
a=imread('peppers.bmp');
a=double(a);
[row col dim]=size(a);
w=[0 -1 0;-1 5 -1;0 -1 0];
red=a(:,:,1);
green=a(:,:,2);
blue=a(:,:,3);
for x=2:1:row-1
  for y=2:1:col-1
    red_f(x,y)= w(1,1)*red(x-1,y-1)+
                w(1,2)*red(x-1,y)+w(1,3)*red(x-1,y+1)+...
                w(2,1)*red(x,y-1)+w(2,2)*red(x,y)+
                w(2,3)*red(x,y+1)+...
w(3,1)*red(x+1,y-1)+w(3,2)*red(x+1,y)+
w(3,3)*red(x+1,y+1);

    green_f(x,y)= w(1,1)*green(x-1,y-1)+
                  w(1,2)*green(x-1,y)+
                  w(1,3)*green(x-1,y+1)+...
                  w(2,1)*green(x,y-1)+w(2,2)*green(x,y)+
                  w(2,3)*green(x,y+1)+...
                  w(3,1)*green(x+1,y-1) +
                w(3,2)*green(x+1,y)+w(3,3)*green(x+1,y+1);

blue_f(x,y)= w(1,1)*blue(x-1,y-1)+
             w(1,2)*blue(x-1,y)+w(1,3)*blue(x-1,y+1)+...
             w(2,1)*blue(x,y-1)+w(2,2)*blue(x,y)+
             w(2,3)*blue(x,y+1)+...
             w(3,1)*blue(x+1,y-1)+w(3,2)*blue(x+1,y)+
             w(3,3)*blue(x+1,y+1);

  end
end

final=cat(3,red_f,green_f,blue_f);

figure(1)
imshow(uint8(a))
figure(2)
imshow(uint8(final))
```

Original image

Image after sharpening

Fig. 11.4.4 : Colour image (Please refer Fig. 10 at the end of the book)

11.5 Colour Segmentation

As we know, segmentation simply means partitioning an image into different regions. Colour segmentation implies segmenting an image based on colour.

Colour segmentation can be achieved using the HSI colour model as well as the RGB colour model.

11.5.1 Segmentation using HSI Model

The HSI model is the most obvious way of segmenting an image based on colour. The biggest advantage that the HSI model has is that colour is represented in the hue image (which is 8-bit).

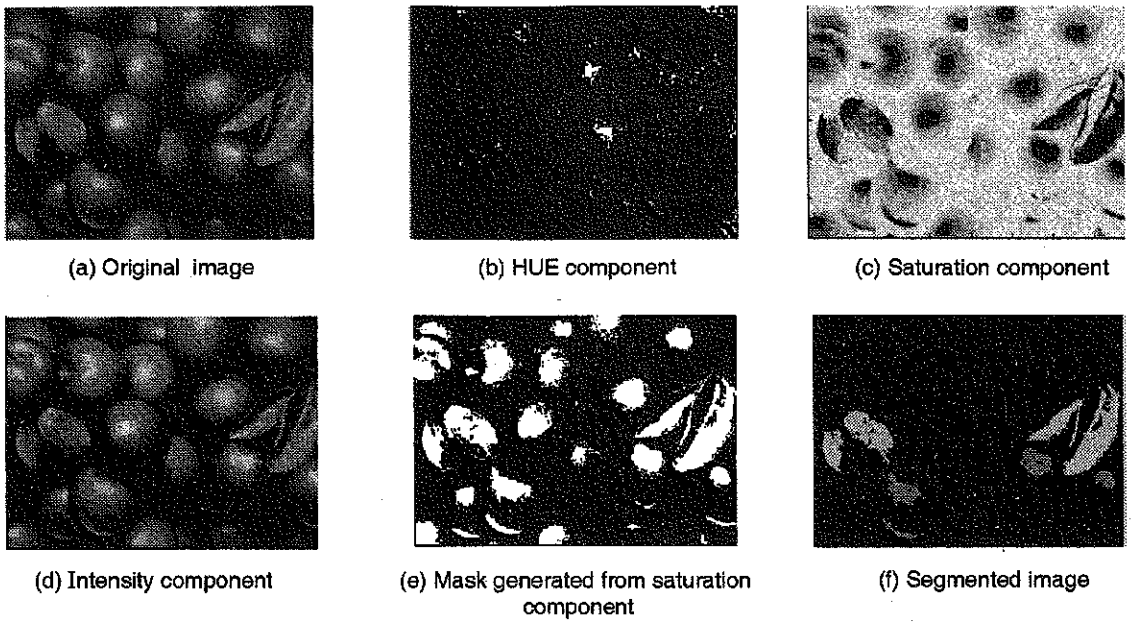
The saturation image plane is used as a mask to further isolate regions of interest in the hue image. Since the Intensity plane carries no information about the colour, it is seldom used for colour segmentation.

Refer Fig. 11.5.1. Assume we need to isolate the green leaves from the image Fig. 11.5.1 (b), Fig. 11.5.1 (c), and Fig. 11.5.1(d) are the H, S and I planes respectively.

We convert the S-plane into a binary image and use it as a mask on the H-plane we perform element by element multiplication to extract the green leaves from the image.

```
%% Segmentation using HSI model

clc

clear all

%a=imread('peppers.bmp');

%a=imread('greens.jpg');

%a=imread('FISK1.bmp');

a=imread('fruits2.bmp');


a=double(a);

[row col dim]=size(a);


red=a(:,:,1);


green=a(:,:,2);

blue=a(:,:,3);


numer=0.5*((red-green)+(red-blue));

deno=sqrt((red-green).^2+(red-blue).*(green-blue));

theta=acos(numer./(deno+eps));


%%Computing Hue

for x=1:1:row

  for y=1:1:col

    if green(x,y)< blue(x,y)
```

```
        H(x,y)=(2*pi)-theta(x,y);
    else
        H(x,y)=theta(x,y);
    end
  end
end

%% Computing Saturation and Intensity

 numer=min(min(red,green),blue);
 deno=red+green+blue;
 deno(deno==0)=eps;
 S=1-(3.*numer./deno);

 I=(red+green+blue)/3;

figure(1)
imshow(uint8(a))
colormap(gray)

figure(2)
imagesc(H)
colormap(gray)

figure(3)
imagesc(S)
colormap(gray)

figure(4)
imagesc(I)
colormap(gray)

M=max(max(S));
for x=1:1:row
  for y=1:1:col
    if S(x,y)>0.6 %This value needs to be set by observing
the saturation componets
        S1(x,y)=0;
    else
        S1(x,y)=1;
    end
  end
end
figure(5)
imagesc(S1)
colormap(gray)

jj=50*(H.*S1);
figure(6)
imshow(uint8(jj))

colormap(gray)
```



(a) Original image (b) HUE component (c) Saturation component

(d) Intensity component (e) Mask generated from saturation component (f) Segmented image

Fig. 11.5.1 : Colour image (Please refer Fig. 11 at the end of the book)

11.5.2 Segmentation using RGB Model

The procedure of colour segmentation is straightforward using the RGB model. Every colour pixel in the image has an R, G and B value. The next colour pixel is said to belong to the same region as the first pixel if its RGB value is within the range of the earlier pixel. (This range is specified by the user). In other words, we measure the similarity between the two pixels. The easiest way to do that is to use the Euclidean distance. Let p and q be the two colour pixels. The Euclidean distance between them is

$$D(p, q) = \| p - q \|$$

$$= [(p - a)^T \cdot (p - a)]^{1/2}$$

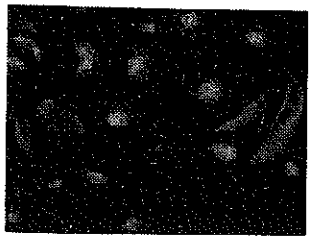
(∵ p, and q are vectors – R, G, B)

$$D(p, q) = [(p_R - q_R)^2 + (p_G - q_G)^2 + (p_B - q_B)^2]^{1/2}$$

Here R, G, B denote the Red, Green and Blue components.

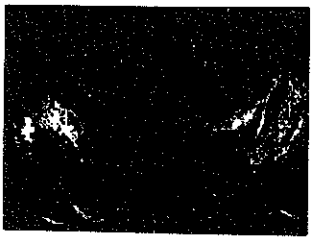
As was with Region based segmentation, this technique is computationally expensive.

```
%% Segmentation using RGB model
clc
clear all

a=imread('fruits2.bmp');
b=rgb2gray(a);
[row col]=size(b);
const=100; %% Would depend on the colour to be segmented
a=double(a);
red=a(:,:,1);
green=a(:,:,2);
blue=a(:,:,3);
for x=1:1:row
  for y=1:1:col
    D(x,y)=sqrt((red(x,y)-const) ^ 2+(green(x,y)-
          const) ^ 2+(blue(x,y)-const) ^ 2);
    %pause(5)
    if D(x,y)>(60-10) && D(x,y)<(60+10)
      DD(x,y)=255;
    else
      DD(x,y)=0;
    end
  end
end

figure(1)
imshow(uint8(a))

figure(2)
imshow(uint8(D))

figure(3)
imshow(uint8(DD))
```



Colour image

Distance image



Segmented image

Fig. 11.5.2 : Colour image (Please refer Fig. 12 at the end of the book)

Summary

This chapter deals with colour image processing. Human beings can identify thousands of colour shades and intensities as opposed to a few shades of grey. This makes colour image processing very important. Various colour models have been discussed in details. Pseudo colour processing which is used regularly is thermography is also discussed. The last section deals with true-colour processing. Smoothing, sharpening and segmentation have been discussed. Programs have been provided for each of the techniques.

Review Questions

Q. 1 Draw the block diagram of a colour television and explain.

Q. 2 Explain RGB model in detail.

Q. 3 Explain various colour models.

Q. 4 What is the HSI colour model ?

Q. 5 Explain Pseudo colouring.

Q. 6 Explain Image smoothing and Image sharpening.

Q. 7 Segmentation using HSI and RGB model which would yield better results. Explain.

Chapter Ends...

❑❑❑

Appendix A

2D Signals and Systems

A.1 Introduction

A signal is defined as any physical quantity that varies with time, space or any other independent variable. Anything that carries some information can be called a signal. Examples of signals that we encounter frequently in our daily life are speech, music, picture, video etc. A 1-Dimensional signal has one dependent variable and one independent variable .

For example $v(t) = 15t$

Here voltage v is the dependant variable and time t is the independent variable.

Similarly, a 2-Dimensional signal has one dependent variable and two independent variable. For example ,

$$f(x, y) = 10x + 12y$$

Here f is the brightness in the image and x, y are the spatial coordinates.

We will confine ourselves to 2-Dimanesional (2-D) signals only

A.2 Classification of 2-D Signals

Discrete time signals have different characteristics.

1. Separable signals

A 2-D signal f(x, y) is separable if it can be represented as a function of two 1-D signals.

$$f(x, y) = f_1(x)\, f_2(y)$$

2. Periodic signals

An image f(x, y) is called periodic with period (N_1, N_2) where both are positive, if and only if $f(x, y) = f(x + N_1, y) = f(x, y + N_2)$

A.3 Classification of 2-D Systems

A discrete time system is a device or an algorithm that operates on a discrete time signal. It is mathematically represented as

$$g(x, y) = T[f(x, y)]$$

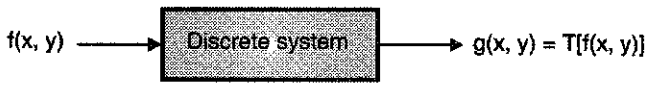


Fig. A.3.1

Discrete time systems are classified according to their general characteristics. They are :

1. Static and Dynamic system (Memory less and Memory systems)
2. Linear and Non-linear system
3. Shift Variant and Shift invariant system
4. Stable and Unstable system

1. Static and Dynamic systems

A system is said to be static if the output g (x, y)of the system depends only on the present input and not on the past or future input. Static systems are also known as Memory less systems. If the output g(x, y) depends on the present as well as the past inputs, then it is referred to as a dynamic system, also known as Memory systems.

Given below are a few examples of static system,

We know that the general representation of a systems is $g(x, y) = T[f(x, y)]$

(i) $g(x, y) = 4 f(x, y)$ → Static system

(ii) $g(x, y) = 0.5 f(x, y) - 0.2 f(x - 1, y) + 0.3 f(x - 1, y - 1)$ → Dynamic system

2. Linear and Non-Linear Systems

A system is said to be linear if it satisfies the superposition principle. Superposition principle states that the response of the system to a weighted sum of signals is equal to the corresponding weighted sum of output of the system to individual input signals.

i.e. $T[a f_1(x, y) + b f_2(x, y)] = aT[f_1(x, y)] + bT[f_2(x, y)]$

A block diagram will make things clearer.

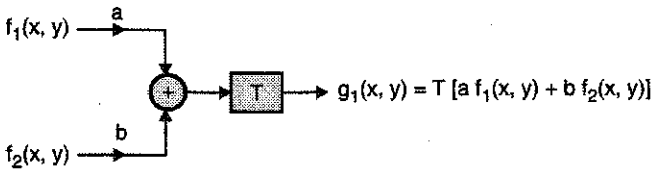


Fig. A.3.2

and

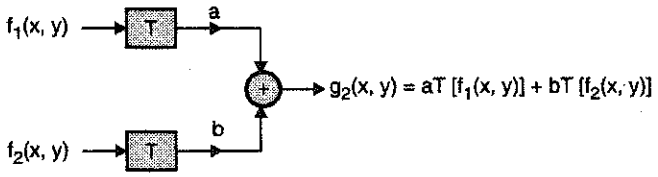


Fig. A.3.3

A system is said to be linear only if $g_1(x, y) = g_2(x, y)$

3. Time Variant and Time Invariant Systems

A system is Shift invariant if its input-output characteristics do not change with time. If g (x, y) is the response of the system to input f(x, y) , then g (x – n, y – m) will be the response of the system to input f (x – n, y – m) i.e. if the input gets shifted by (n, m) samples, then the output also shifts by (n, m) samples.

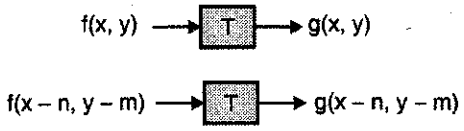


Fig A.3.4

4. Stable and Unstable systems

A system is said to be stable if for a bounded input, the system produces a bounded output (BIBO condition). This will happen if and only if its impulse response is absolutely summable i.e.,

$$\sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} |h(x, y)| < \infty$$

A.4 Convolution

The convolution operation is one of the most important operations in engineering and is explained in detail in chapter 4. 1-D Convolution is given by the formula

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$$

where x is the input, h is the impulse response and y is the output.

It is represented by the notation

$$y(n) = x(n) * h(n)$$

Similarly, 2-D convolution is given by the formula

$$g(x, y) = \sum_{m} \sum_{n} f(m, n)\, h(x-m, y-n)$$

and is represented as

$$g(x, y) = f(x, y) * h(x, y)$$

In Convolution, two signals are given. We take any one o the two signals and flip it. We the place the two signals one over the other, multiply corresponding elements and add the result. We then move the flipped signal across the other signal step by step and repeat the same procedure.

We will consider an example of 1-D signals.

Ex. A.4.1

Obtain linear convolution of following sequences :

$x(n) = \{1, 2, 1, 2\}$ and $h(n) = \{1, 1, 1\}$

Soln. :

If the sequences are given in terms of 'n' then obtain x (k) and h (k) by replacing 'n' by 'k'.

$\therefore x(k) = \{1, 2, 1, 2\}$

and $h(k) = \{1, 1, 1\}$

Since the origin is not mentioned, we consider the first position as the default origin.

$$\therefore x(k) = \{\underset{\uparrow}{1}, 2, 1, 2\}$$

and $$h(k) = \{\underset{\uparrow}{1}, 1, 1\}$$

Convolution is given by the formula,

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k) \qquad \ldots(1)$$

Now we have to decide range of 'n' and 'k'.

Range of 'n'

Basically to calculate y (n) we should know from which value of 'n'. We should start (lower range of n) and what is ending value of 'n' (higher range of n). We will use following notations to decide range of y (n).

y_l = Lowest range of y (n)

y_h = Highest range of y (n)

x_l = Lowest range of x (k)

x_h = Highest range of x (k)

h_l = Lowest range of h (k)

h_h = Highest range of h (k)

Now use following formulae to calculate range of y (n)

$$y_l = x_l + h_l \text{ and } y_h = x_h + h_h$$

From the given sequences we have,

$x_l = 0, x_h = 3, h_l = 0$ and $h_h = 2$

$$\therefore \quad y_l = x_l + h_l = 0 + 0 = 0$$

and $$y_h = x_h + h_h = 3 + 2 = 5$$

Thus range of y (n) is from y (0) to y (5).

Range of 'k'

The value of k in the summation sign will be always same as the sequence x (k). Thus in this case the range of k is from k = 0 to 3. So Equation (1) becomes,

$$y(n) = \sum_{k=0}^{3} x(k)\, h(n-k) \qquad \ldots(2)$$

Now we will calculate output y (n) by putting different values of n from 0 to 5.

Calculation of y (0)

Putting n = 0 in Equation (2) we get,

$$y(0) = \sum_{k=0}^{3} x(k)\,(0-k)$$

$$y(0) = \sum_{k=0}^{3} x(k)\, h(-k) \qquad \ldots(3)$$

Now we have to perform calculations using graphical method. Here h (– k) indicates folded version of h (k).

Step I : Sketch x (k).

Sketch of x (k) is as shown in Fig. P. A.4.1(a).

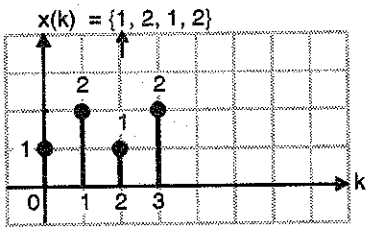


Fig. P. A.4.1(a) : Sequence x (k)

Step II : Sketch h (k)

Sketch of h (k) is shown in Fig. P. A.4.1(b).

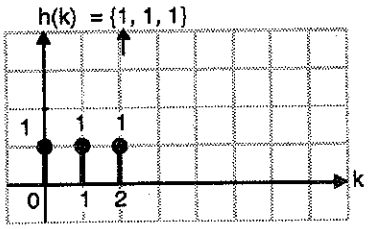


Fig. P. A.4.1(b) : Sequence h (k)

Step III : Fold h (k) to obtain h (– k).

Sketch of h (– k) is shown in Fig. P. A.4.1(c).

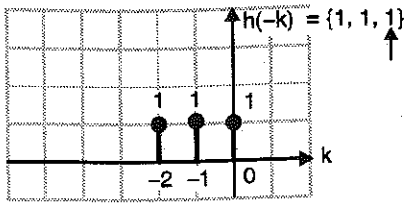


Fig. P. A.4.1(c) : Folded sequence h (– k)

Step IV : Obtain multiplication of x (k) and h (– k). This multiplication takes place on sample to sample basis as shown in Fig. P. A.4.1(d).

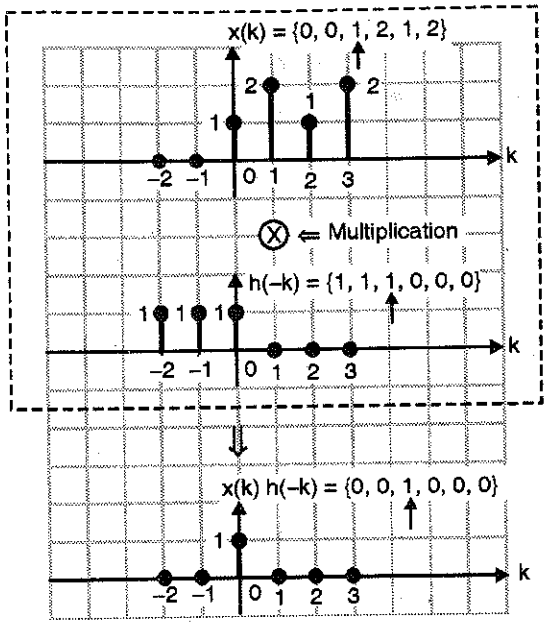


Fig. P. A.4.1(d) : Product of x (k) and h (– k)

Step V : Take the summation of all product terms.

In this case as result of multiplication we have only one sample at n = 0.

$\therefore y(0) = 1$

Calculation of y (1)

Putting n = 1 in Equation (2) we get,

$$y(n) = \sum_{k=0}^{3} x(k)\,h(1-k) \qquad \ldots(4)$$

h (1 – k) can be written as h (– k + 1). Thus Equation (4) becomes,

$$y(n) = \sum_{k=0}^{3} x(k)\,h(-k+1) \qquad \ldots(5)$$

Step I : Now h (– k + 1) indicates delay of h (– k) by '1' sample. So shift the sequence in Fig. P. A.4.1(c) towards right by '1' sample. This sequence is shown in Fig. P. A.4.1(e).

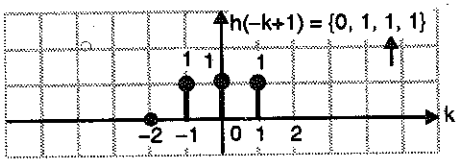


Fig. P. A.4.1(e) : Sequence h (– k + 1)

Step II : Obtain multiplication of x (k) and h (– k + 1). It is shown in Fig. P. A.4.1(f).

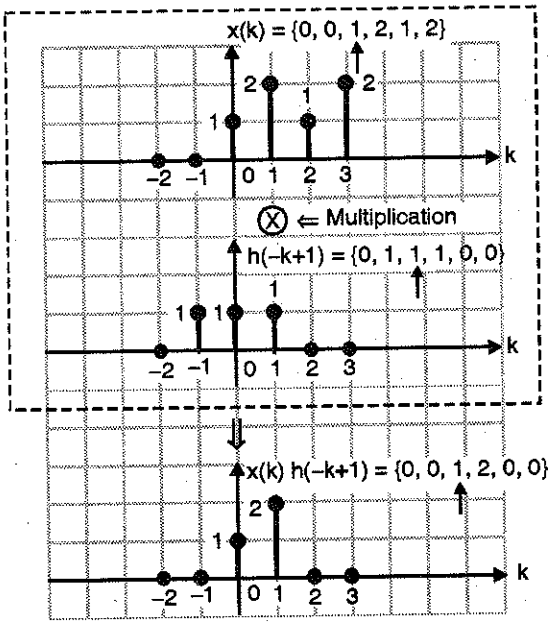


Fig. P. A.4.1(f) : Product of x (k) and h (– k + 1)

Step III : Now according to Equation (5) we have to add all product terms shown in Fig. P. A.4.1(f) to obtain y (n).

$\therefore y(1) = (1\times1)+(2\times1) = 1+2$

$\therefore y(1) = 3$

Calculation of y (2)

Putting n = 2 in Equation (2) we get,

$$y(2) = \sum_{k=0}^{3} x(k)\,h(2-k) \qquad \ldots(6)$$

The term h (2 – k) is same as h (– k + 2).

$$\therefore\ y(2) = \sum_{k=0}^{3} x(k)\,h(-k+2) \qquad \ldots(7)$$

Step I : Here h (– k + 2) indicates delay of h (– k) by '2' samples. This is obtained by shifting sequence in Fig. P. A.4.1(c) towards right by '2' samples. This sequence is shown in Fig. P. A.4.1(g).

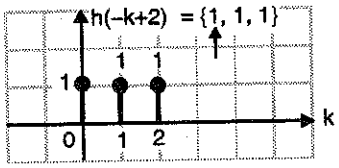


Fig. P. A.4.1(g) : Sequence h (– k + 2)

Step II : Obtain multiplication of x (k) and h (– k + 2). It is shown in Fig. P. A.4.1(h).

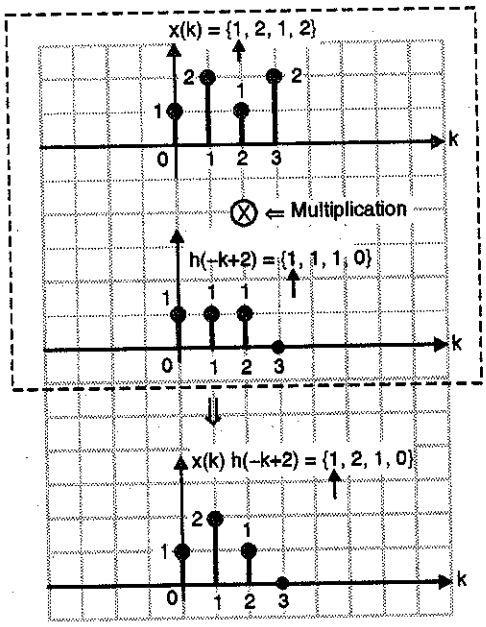


Fig. P. A.4.1(h) : Product of x (k) and h (– k + 2)

Step III : Add all product terms to obtain y (2).

$$\therefore\ y(2) = (1\times1)+(1\times2)+(1\times1) = 1+2+1$$

$$\therefore\ \mathbf{y(2) = 4}$$

Calculation of y (3)

Putting n = 3 in Equation (2) we get,

$$y(3) = \sum_{k=0}^{3} x(k)\,h(3-k) \qquad \ldots(8)$$

The term h (3 – k) is same as h (– k + 3)

$$\therefore\ y(3) = \sum_{k=0}^{3} x(k)\,h(-k+3) \qquad \ldots(9)$$

Step I : Here h (– k + 3) indicates delay of h (– k) by '3' samples. It is shown in Fig. P. A.4.1(i).

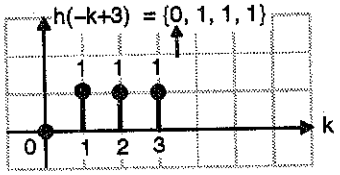


Fig. P. A.4.1(i) : Sequence h (– k + 3)

Step II : Obtain multiplication of x (k) and h (– k + 3). This sequence is shown in Fig. P. A.4.1(j).

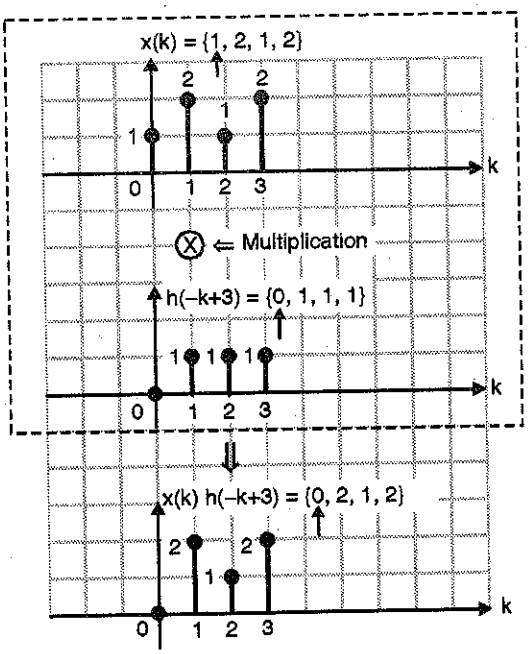


Fig. P. A.4.1(j) : Product of x (k) and h (– k + 3)

Step III : Add all product terms to obtain y (3).

$$\therefore\ y(3) = (0\times1)+(1\times2)+(1\times1)+(2\times1)$$

$$= 0+2+1+2$$

$$\therefore\ \mathbf{y(3) = 5}$$

Calculation of y (4)

Putting n = 4 in Equation (2) we get,

$$y(4) = \sum_{k=0}^{4} x(k)\, h(4-k) \qquad \ldots(10)$$

The term h (4 – k) is same as h (– k + 4)

$$\therefore\ y(4) = \sum_{k=0}^{4} x(k)\, h(-k+4) \qquad \ldots(11)$$

Step I : Here h (– k + 4) indicates delay of h (– k) by '4' samples. It is shown in Fig. P. A.4.1(k).

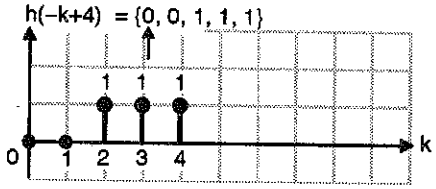


Fig. P. A.4.1(k) : Sequence h (– k + 4)

Step II : Obtain multiplication of x (k) and h (– k + 4). This sequence is shown in Fig. P. A.4.1(*l*).

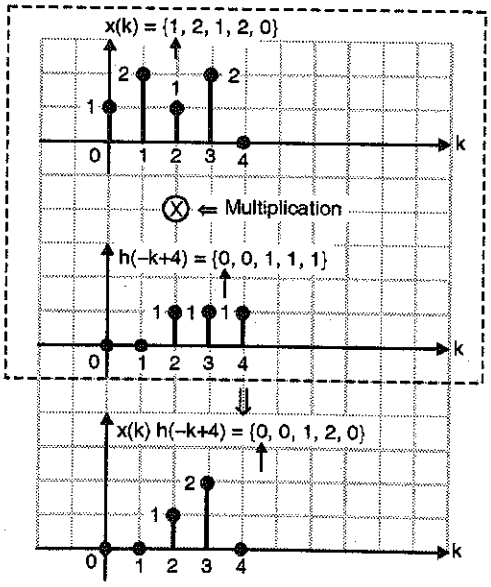


Fig. P. A.4.1(*l*) : Product of x (k) and h (– k + 4)

Step III : Add all product terms to obtain y (4).

$$\therefore y(4) = (1\times0)+(2\times0)+(1\times1)+(2\times1)+(0\times1)$$

$$= 0+0+1+2+0$$

$$\therefore y(4) = 3$$

Calculation of y (5)

Putting n = 5 in Equation (2) we get,

$$y(5) = \sum_{k=0}^{3} x(k)\, h(5-k) \qquad \ldots(12)$$

The term h (5 – k) is same as h (– k + 5)

$$\therefore\ y(5) = \sum_{k=0}^{3} x(k)\, h(-k+5) \qquad \ldots(13)$$

Step I : Here h (– k + 5) indicates delay of h (– k) by '5' samples. It is shown in Fig. P. A.4.1(m).

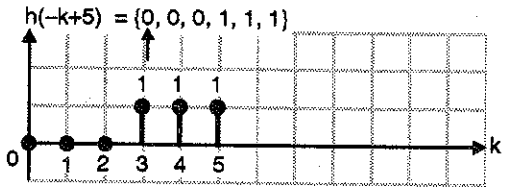


Fig. P. A.4.1(m) : Sequence h (– k + 5)

Step II : Obtain multiplication of x (k) and h (– k + 5). This sequence is shown in Fig. P. A.4.1(n).

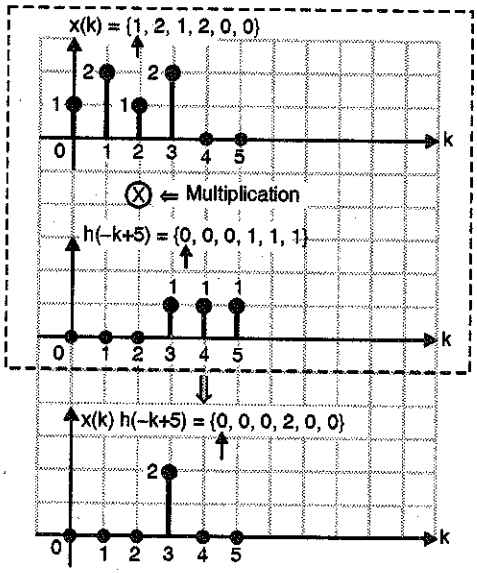


Fig. P. A.4.1(n) : Product of x (k) and h (– k + 5)

Step III : Add all product terms to obtain y (5).

$$\therefore y(5) = (1\times0)+(2\times0)+(1\times0)+(2\times1)+(0\times1)+(0\times1)$$

The result of convolution , y (n) is represented graphically as shown in Fig. P. A.4.1(o).

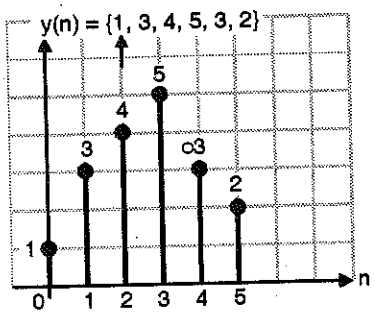


Fig. P. A.4.1(o) : Sketch of y (n) = { 1, 3, 4, 5, 3, 2 }

2-D convolution is explained in detail in the chapter Image Enhancement in the Spatial domain.

Ex. A.4.2

Perform discrete convolution on the following image arrays. Assume that the left bottom corner elements are at the origin in both cases. f_1 and f_2 are two images.

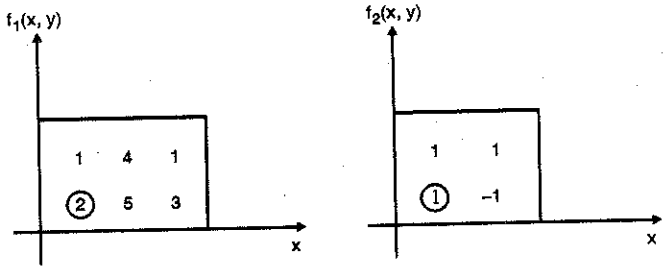


Fig. P. A.4.2(a)

Soln. :

$$f_1(x, y) * f_2(x, y) = \sum_m \sum_n f_1(m, n)\, f_2(x - m, y - n)$$

Let $f_1(x, y) * f_2(x, y) = g(x, y)$

$$g(x, y) = \sum_m \sum_n f_1(m, n)\, f_2(x - m, y - n)$$

$$g(0, 0) = \sum_m \sum_n f_1(m, n)\, f_2(-m, -n)$$

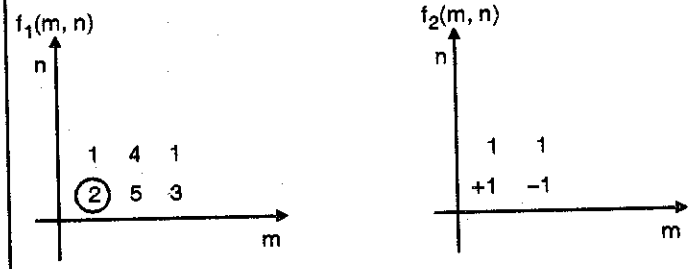


Fig. P. A.4.2(b)

The basic shift operations are

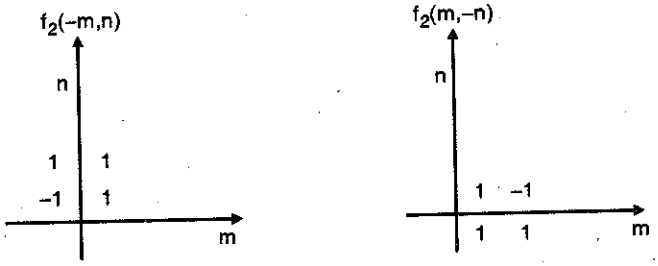
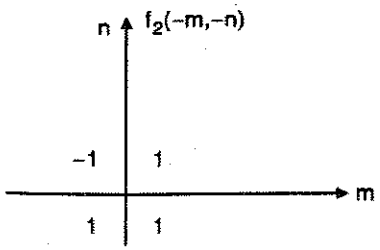


Fig. P. A.4.2(c)

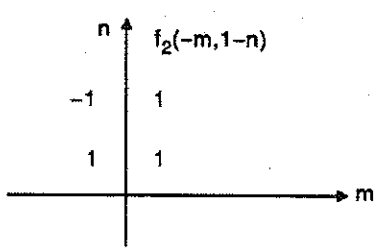
$f_2(-m, n) \Rightarrow$ flipping about m-axis



$g(0,0) = \sum_m \sum_n f_1(m, n)\, f_2(-m, -n)$

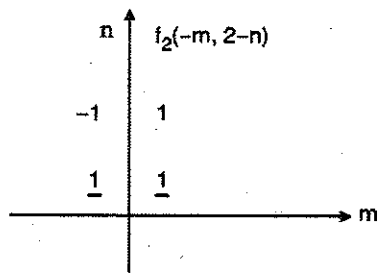
$g(0,0) = 2 \times 1 = 2$

$f_2(m, -n) \Rightarrow$ flipping about n-axis



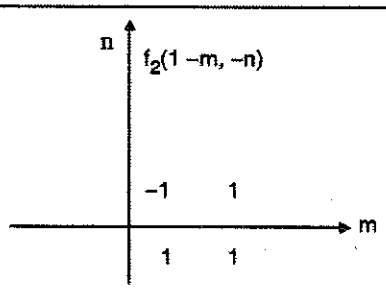
$g(0,1) = \sum_m \sum_n f_1(m, n)\, f_2(-m, 1-n)$

$g(0,1) = 1 \times 1 + 2 \times 1 = 3$



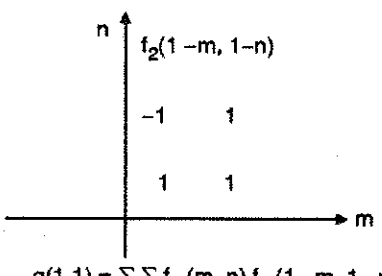
$g(0,2) = \sum_m \sum_n f_1(m, n)\, f_2(-m, 2-n)$

$g(0,2) = 1 \times 1 = 1$



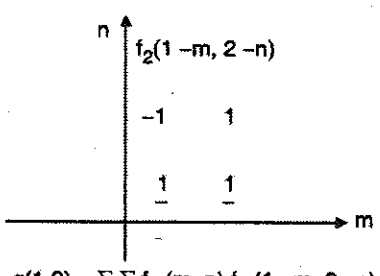
$g(1,0) = \sum \sum f_1(m, n)\, f_2(1-m, -n)$

$g(1,0) = 2 \times -1 + 5 \times 1 = 3$



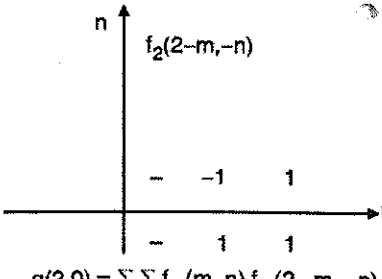
$g(1,1) = \sum_m \sum_n f_1(m, n)\, f_2(1-m, 1-n)$

$g(1,1) = 1 \times -1 + 4 \times 1 + 2 \times 1 + 5 \times 1 = 10$



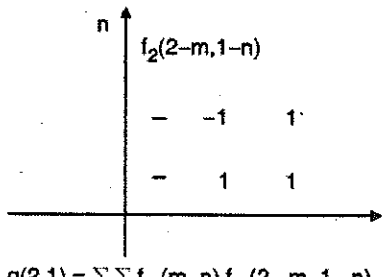
$g(1,2) = \sum_m \sum_n f_1(m, n)\, f_2(1-m, 2-n)$

$g(1,2) = 1 \times 1 + 4 \times 1 = 5$



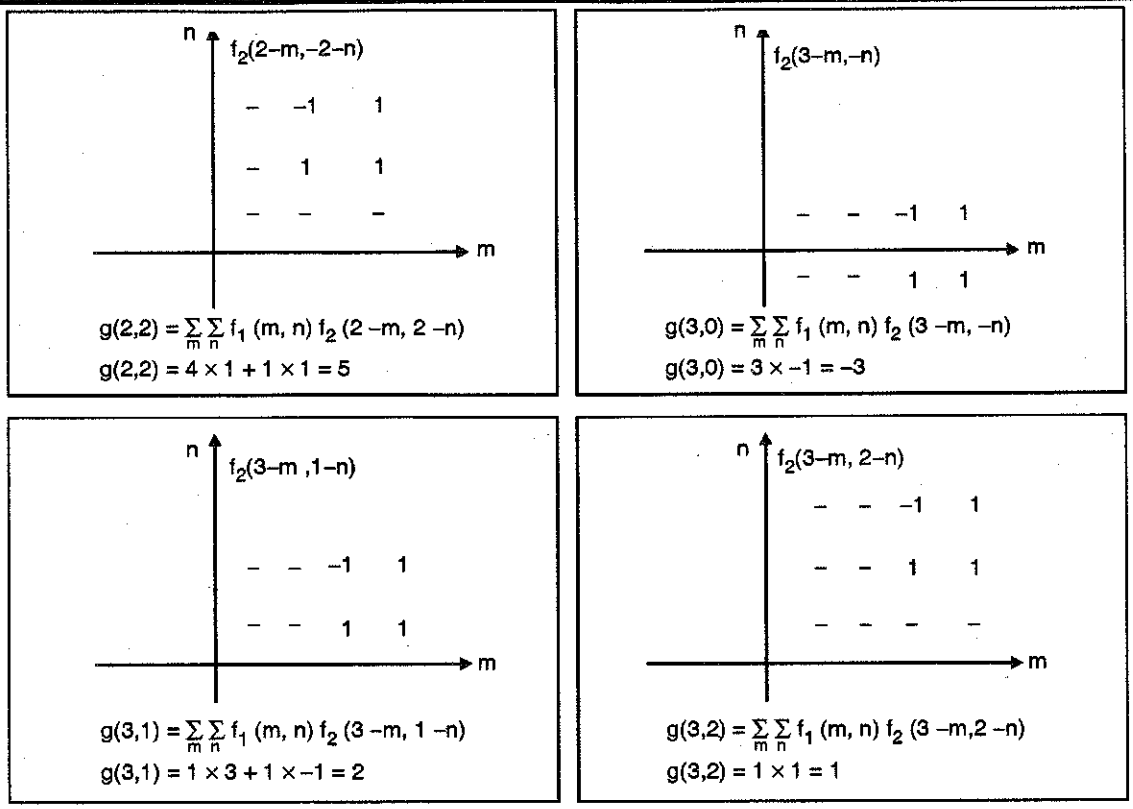
$g(2,0) = \sum_m \sum_n f_1(m, n)\, f_2(2-m, -n)$

$g(2,0) = 5 \times -1 + 3 \times 1 = -2$



$g(2,1) = \sum_m \sum_n f_1(m, n)\, f_2(2-m, 1-n)$

$g(2,1) = 4 \times -1 + 1 \times 1 + 5 \times 1 + 3 \times 1 = 5$



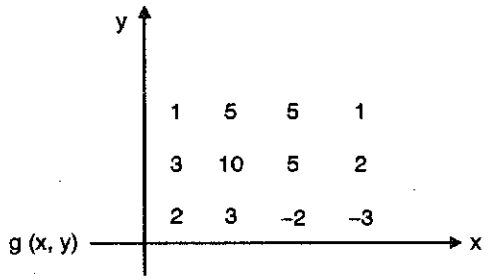


Fig. P. A.4.2(d)

Another method : It is convenient to represent digital images as matrices and exploit the benefits of linear algebra. We modify the matrix so as to suit convolution.

Let F_1 and F_2 be the two images that need to be convolved :

Let F_1 be a $m_1 \times n_1$ image and F_2 be a $m_2 \times n_2$ image. We zero pad F_1 and F_2 so that both of them are of size $(m_1 + m_2 - 1) \times (n_1 + n_2 - 1)$.

Ex. A.4.3

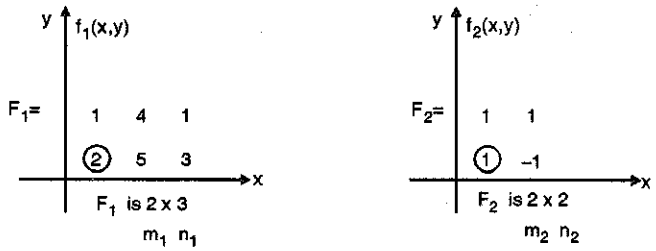


Fig. P. A.4.3(a)

Soln. :

Zero padding F_1 and F_2 so that both are of size

$$(m_1 + m_2 - 1) \times (n_1 + n_2 - 1) = 3 \times 4$$

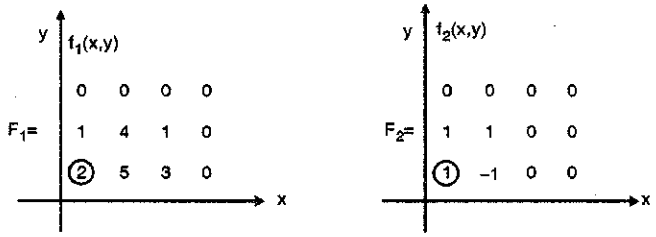


Fig. P. A.4.3(b)

Now, of these two, arrange one as a circular matrix and the other one as a row stacking matrix. Let the first element of these be their respective origins, then

$$G(x, y) = F_1(x, y) F_2(x, y)$$

F_1 | F_2 | G

F_1												F_2	G
(2)	0	0	0	0	1	4	1	0	3	5		(1)	(2)
5	2	0	0	0	0	1	4	1	0	3		-1	3
3	5	2	0	0	0	0	1	4	1	0		0	-2
0	3	5	2	0	0	0	0	1	4	1		0	-3
1	0	3	5	2	0	0	0	0	1	4		1	3
4	1	0	3	5	2	0	0	0	0	1		1	10
1	4	1	0	3	5	2	0	0	0	0		0	5
0	1	4	1	0	3	5	2	0	0	0		0	2
0	0	1	4	1	0	3	5	2	0	0		0	1
0	0	0	1	4	1	0	3	5	2	0		0	5
0	0	0	0	1	4	1	0	3	5	2		0	5
0	0	0	0	0	1	4	1	0	3	5	2	0	1

We see that the results are the same as obtained from the previous method.

$\therefore$ G (x, y)

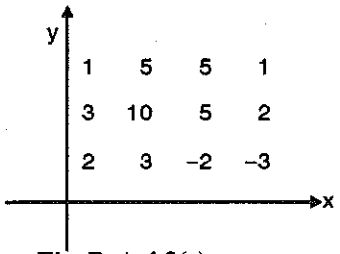


Fig. P. A.4.3(c)

A.5 Correlation

Correlation is basically used when we want to compare two signals. It is a very important operation in signal processing and image processing. Correlation is a measure of the degree to which two signals are similar.

Correlation as a mathematical operation closely resembles convolution.

Correlation of two separate signals is known as cross correlation, while correlation of the signal with itself is known as auto correlation.

We shall explain correlation in terms of 1-D signals.

1. **Cross correlation :** Correlation between two signals x(n) and y(n) is called cross correlation. It is given by the formula

$$r_{xy}(l) = \sum_{n=-\infty}^{+\infty} x(n)\, y(n-l)$$

2. **Auto correlation :** Many times it is necessary to correlate the signal with itself for example determination of time delays between transmitted and received signal.

Hence the two signals are generated from the same source. It is given by the formula

$$r_{xx}(l) = \sum_{n=-\infty}^{+\infty} x(n)\, x(n-l)$$

Correlation is exactly similar to convolution except that in correlation, we do not flip one signal.

Ex. A.5.1

Compute the cross - correlation between

$x(n) = \{1, 1, \underset{\uparrow}{0}, 1\}$ and

$y(n) = \{4, -3, -2, \underset{\uparrow}{1}\}$

Soln. :

The cross-correlation of x (n) and y(n) is given by,

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)\, y(n-l) \qquad \ldots(1)$$

Range of 'n'

In Equation (1) sequence x (n) is not changed but sequence y (n) is delayed by 'l'. Thus range of summation ('n') will be same as x (n).

Thus range of 'n' is from – 2 to 1.

Putting these values in Equation (1) we get,

$$r_{xy}(l) = \sum_{n=-2}^{1} x(n)\, y(n-l) \qquad \ldots(2)$$

Now we will decide range of 'l' as follows :

Range of 'l'

Observe the given sequences. Here maximum value of y (n) is y (1). So in Equation (2) y (n – l) should have maximum value equals to y (1). If we get y (n – l) = y (2) then, result of summation will be zero.

Now starting value of 'n' in summation is n = – 2.

Thus y (n – l) = y (1) maximum $\Rightarrow$ n – l = 1 maximum that means,

$$-2 - l = 1 \Rightarrow l = -2 - 1 = -3$$

So we will start from $l = -3$

Now from the given sequence; lowest value of y (n) is y (– 2). Thus from Equation (2), we have y (n – l) = y (– 2). Now according to Equation (2) we should stop the summation at n = 1.

Thus y (n – l) = y (– 2) $\Rightarrow 1 - l = -2 \Rightarrow l = 1 + 2 = 3$

Thus we will stop at $l = +3$.

So range of l is from – 3 to + 3.

Calculation of $r_{xy}(-3)$

Putting $l = -3$ in Equation (2) we get

$$r_{xy}(-3) = \sum_{n=-2}^{1} x(n)\, y(n+3) \qquad \ldots(3)$$

Expanding the summation,

$$r_{xy}(-3) = x(-2)\, y(-2+3) + x(-1)\, y(-1+3) + x(0)\, y(0+3) + x(1)\, y(1+3)$$

$$\therefore \quad r_{xy}(-3) = x(-2)\, y(1) + x(-1)\, y(2) + x(0)\, y(3) + x(1)\, y(4) \qquad \ldots(4)$$

Now we have given sequences,

$$x(n) = \{1, 1, 0, 1\} \quad \text{and} \quad y(n) = \{4, -3, -2, 1\}$$
$$\uparrow \qquad\qquad\qquad\qquad \uparrow$$

$x(-2) = 1$ and $y(-2) = 4$

$x(-1) = 1$ and $y(-1) = -3$

$x(0) = 0$ and $y(0) = -2$

$x(1) = 1$ and $y(1) = 1$

Putting these values in Equation (4) we get,

$$r_{xy}(-3) = (1\times 1) + (1\times 0) + (0\times 0) + (1\times 0)$$
$$= 1 + 0 + 0 + 0$$

$$\therefore \quad \mathbf{r_{xy}(-3) = 1}$$

Calculation of $r_{xy}(-2)$

Putting $l = -2$ in Equation (2) we get,

$$r_{xy}(-2) = \sum_{n=-2}^{1} x(n)\, y(n+2) \qquad \ldots(5)$$

Expanding the summation,

$$r_{xy}(-2) = x(-2)\, y(-2+2) + x(-1)\, y(-1+2) + x(0)\, y(2) + x(1)\, y(1+2)$$

$$\therefore \quad r_{xy}(-2) = x(-2)\, y(0) + x(-1)\, y(1) + x(0)\, y(2) + x(1)\, y(3)$$

$$\therefore \quad r_{xy}(-2) = (1\times(-2)) + (1\times 1) + (0\times 0) + (1\times 0)$$
$$= -2 + 1$$

$$\therefore \quad \mathbf{r_{xy}(-2) = -1}$$

Calculation of $r_{xy}(-1)$

Putting $l = -1$ in Equation (2) we get,

$$r_{xy}(-1) = \sum_{n=-2}^{1} x(n)\, y(n+1) \qquad \ldots(6)$$

Expanding the summation,

$$r_{xy}(-1) = x(-2)\, y(-2+1) + x(-1)\, y(-1+1) + x(0)\, y(0+1) + x(1)\, y(1+1)$$

$$\therefore r_{xy}(-1) = x(-2)\, y(-1) + x(-1)\, y(0) + x(0)\, y(1) + x(1)\, y(2)$$

$$r_{xy}(-1) = [1 \times (-3)] + [1 \times (-2)] + [0 \times 1] + [1 \times 0] = -3 - 2$$

$$\therefore \mathbf{r_{xy}(-1) = -5}$$

Calculation of $r_{xy}(0)$

Putting $l = 0$ in Equation (2) we get,

$$r_{xy}(0) = \sum_{n=-2}^{1} x(n)\, y(n) \qquad \ldots(7)$$

Expanding the summation,

$$\therefore r_{xy}(0) = x(-2)\, y(-2) + x(-1)\, y(-1) + x(0)\, y(0) + x(1)\, y(1)$$

$$= [1 \times 4] + [1 \times (-3)] + [0 \times (-2)] + [1 \times 1]$$

$$= 4 - 3 + 0 + 1$$

$$\therefore \mathbf{r_{xy}(0) = 2}$$

Calculation of $r_{xy}(1)$

Putting $l = 1$ in Equation (2) we get,

$$r_{xy}(1) = \sum_{n=-2}^{1} x(n)\, y(n-1) \qquad \ldots(8)$$

Expanding the summation,

$$r_{xy}(1) = x(-2)\, y(-2-1) + x(-1)\, y(-1-1) + x(0)\, y(0-1) + x(1)\, y(1-1)$$

$$r_{xy}(1) = x(-2)\, y(-3) + x(-1)\, y(-2) + x(0)\, y(-1) + x(1)\, y(0)$$

$$= [1 \times 0] + [1 \times 4] + [0 \times (-3)] + [1 \times (-2)] = 0 + 4 + 0 - 2$$

$$\therefore \mathbf{r_{xy}(1) = 2}$$

Calculation of $r_{xy}(2)$

Putting $l = 2$ in Equation (2) we get,

$$r_{xy}(2) = \sum_{n=-2}^{1} x(n)\, y(n-2) \qquad \ldots(9)$$

Expanding the summation,

$$r_{xy}(2) = x(-2)\, y(-2-2) + x(-1)\, y(-1-2) + x(0)\, y(0-2) + x(1)\, y(1-2)$$

$$= x(-2)\, y(-4) + x(-1)\, y(-3) + x(0)\, y(-2) + x(1)\, y(-1)$$

$$= [1 \times 0] + [1 \times 0] + [0 \times 4] + [1 \times (-3)]$$

$$= 0 + 0 + 0 - 3$$

$$\therefore \mathbf{r_{xy}(2) = -3}$$

Calculation of $r_{xy}(3)$

Putting $l = 3$ in Equation (2) we get,

$$r_{xy}(3) = \sum_{n=-2}^{1} x(n)\, y(n-3) \qquad \ldots(10)$$

Expanding the summation,

$$r_{xy}(3) = x(-2)\, y(-2-3) + x(-1)\, y(-1-3) + x(0)\, y(0-3) + x(1)\, y(1-3)$$

$$= x(-2)\, y(-5) + x(-1)\, y(-4) + x(0)\, y(-3) + x(1)\, y(-2)$$

$$= [1 \times 0] + [1 \times 0] + [0 \times 0] + [1 \times 4]$$

$$= 0 + 0 + 0 + 4$$

$$\therefore \mathbf{r_{xy}(3) = 4}$$

Now the final sequence is written as,

$$r_{xy}(l) = \{ r_{xy}(-3), r_{xy}(-2), r_{xy}(-1), r_{xy}(0), r_{xy}(1), r_{xy}(2), r_{xy}(3) \}$$
$$\uparrow$$

$$\therefore \quad \mathbf{r_{xy}(l) = \{1, -1, -5, 2, 2, -3, 4\}}$$
$$\uparrow$$

Now that arrow is marked at the value of $r_{xy}(0)$.

2-D correlation is given by the formula,

$$r_{f1,f2}(i, j) = \sum_m \sum_n f_1(m, n) f_2(m + i, n + j)$$

A.6 Z-transform

The z-transform which was introduced by Ragazzini and Zedels in 1952 is a powerful mathematical tool for the analysis of Linear Time Invariant discrete systems. The Z-transform converts a discrete time domain signal (which is a sequence of numbers) into a complex frequency domain representation.

The Z-transform can be considered as a discrete equivalent of the Laplace transforms.

Just as analog filters are designed using the Laplace transform, digital filters are developed using the Z-transform.

The 1-Dimensional Z-transform of a discrete time signal x(n) is defined as,

$$X(z) = \sum_{n=-\infty}^{+\infty} x(n) z^{-n} \qquad \ldots(1)$$

where $z = re^{j\omega}$ which is a complex variable.

In a similar manner, 2-Dimensional Z- transform is given by the formula

$$X(z_1, z_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1, n_2) z_1^{-n1} z_2^{-n2} \qquad \ldots(2)$$

where z_1 and z_2 are complex variables.

❑❑❑