

1

Chapter One

List of Contents:

Chapter one Digital Image Fundamentals & Color Systems Chapter two

Image Arithmetic

Chapter three... Neighborhood Processing & Image Filters Chapter four.....

Frequency Domain: The 2D DFT

Chapter five Image Restoration and Qualification Chapter six

Image Feature Detection

Chapter seven ... Image Compression (1)

Chapter eight Image Compression (2):The 2D DCT Chapter nine

Mathematical Morphology (1) Chapter ten Mathematical

Morphology (2) Chapter eleven ... The 2D DWT: Wavelet Transform

• **References**

1. An introduction to digital image processing with MATLAB by Ala Sdair Mc Andrew, 2004
2. Digital Image Processing using MATLAB by R C Gonzales, 2007

Introduction

• *Light and colours*

Visible light is part of the electromagnetic spectrum: radiation in which the energy takes the form of waves of varying wavelength. These range from cosmic rays of very short wavelength to electric power, which has very long wavelength. Figure 1 illustrates this. The electron beams of X-rays have a shorter wavelength than visible light, and so can be used to resolve smaller objects than are possible with visible light. X-rays are of course also useful in determining the structure of objects usually hidden from view: such as bones. A further method of obtaining medical images is by the use of x-ray tomography

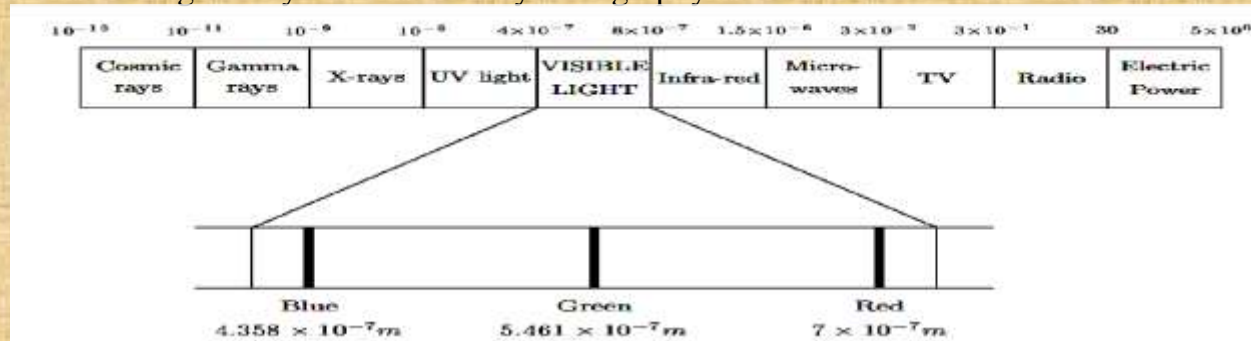


Figure 1 The electromagnetic spectrum

Image perception •

We should be aware of the limitations of the human visual system. ***Observed intensities vary as to the background.*** A single block of grey will appear darker if placed on a white background than if it were placed on a black background. That is, we don't perceive grey scales as they are, but rather as they differ from their surroundings. In figure 2 a grey square is shown on two different backgrounds. The two central squares have exactly the same intensity.



Figure 2 a gray square on different background

Digital Images •

Digital Images are electronic snapshots taken of a scene or scanned from documents, such as photographs, manuscripts, printed texts, and artwork. The digital image is sampled and mapped as a grid of dots or picture elements (pixels). Each pixel is assigned a tonal value (black, white, shades of gray or color), which is represented in binary code (zeros and ones). The binary digits ("bits") for each pixel are stored in a sequence by a computer and often reduced to a mathematical representation (compressed). The bits are then interpreted and read by the computer to produce an analog version for display or printing.

- **Resolution**

Is the ability to distinguish fine spatial detail. The spatial frequency at which a digital image is sampled (the sampling frequency) is often a good indicator of resolution. This is why dots-per-inch (dpi) or pixels-per-inch (ppi) are common and synonymous terms used to express resolution for digital images. Generally, increasing the sampling frequency also helps to increase resolution.

- **Bit Depth**

combinations: 00, 01, 10, and 11. If "00" represents black, and "11" represents white, then "01" equals dark gray and "10" equals light gray. The bit depth is two, but the number of tones that can be represented is 2^2 or 4. At 8 bits, 256 (2^8) different tones can be assigned to each pixel.

- **File Format**

It consists of both the bits that comprise the image and header information on how to read and interpret the file. File formats vary in terms of resolution, bit-depth, color capabilities, and support for compression and metadata.

- **Dynamic Range**

Is the range of tonal difference between the lightest light and darkest dark of an image. The higher the dynamic range, the more potential shades can be represented, although the dynamic range does not automatically correlate to the number of tones reproduced. For instance, high-contrast microfilm exhibits a broad dynamic range, but renders few tones. Dynamic range also describes a digital system's ability to reproduce tonal information. The varying tones for photographs may be the single most important aspect of image quality. For example, the left image has a broader dynamic range with limited tones representation, while the right one has a narrower dynamic range, but a greater numbers of tones representation



1.1 Digital Image Representation

Any image maybe defined as a *two dimensions 2D* function, $f(x,y)$ where x and y are spatial coordinates and the amplitude of $f(x,y)$ is called the intensity or gray level of the image at that point we call the image a digital image if and only if x , y , and the amplitude $f(x,y)$ are all finite discrete quantities. Mainly two operations are required to obtain a digital image: Sampling and Quantization, sampling operation is the process of digitizing the coordinate values and quantization is the process of digitizing the amplitude values.

1.2 Images and Video Frames

A digital image differs from a photo or picture in that x , y , and $f(x,y)$ values are all discrete. Usually they have taken only integer values. It can be considered they as a large array of sampled points from continues video stream, where the contents of video stream is called frames. Each of which has a particular quantized brightness of points called pixels which constitute the digital image.

1.3 Reasons of Image Processing

1.3.1 Image Enhancement

- Highlighting edges
- Obtaining the edges
- Improve the contrast
- Image noise removal
- Image blur removal

1.3.2 Image Restoration :

Reversing the damage happened to an image by a known source like:

- Noise effects
- Blur effects
- Optical distortion

1.3.3 Image Segmentation:

Segment an image means subdividing it into constituent parts or isolated objects, for example finding lines, circles, or a particular shape

4. Digital Image Types

1. True colure, Red Green Blue (RGB) image
2. Indexed image
3. Grayscale (intensity image) image
4. Binary image

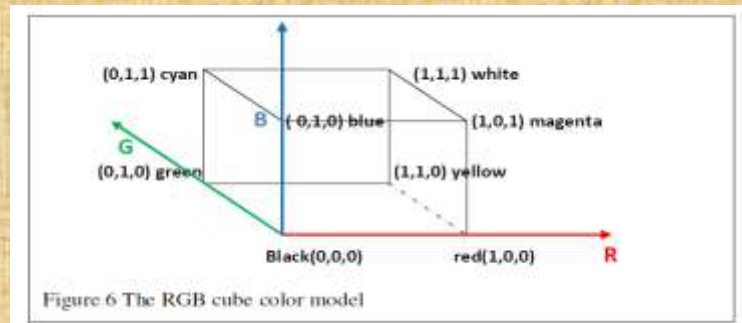
1. The RGB image:

Here each pixel has a particular colour, that colour being described by its amount of red, green, and blue. For example if a colored pixel with 8 bits size it means the red has $2^8 = 256$ (0 – 255) intensity level, and so on for green and blue pixel. (b)

This gives a total of $256^3 = 16,777, 216$ different possible colures in the pixel, e.g., *img.jpg and img.png*

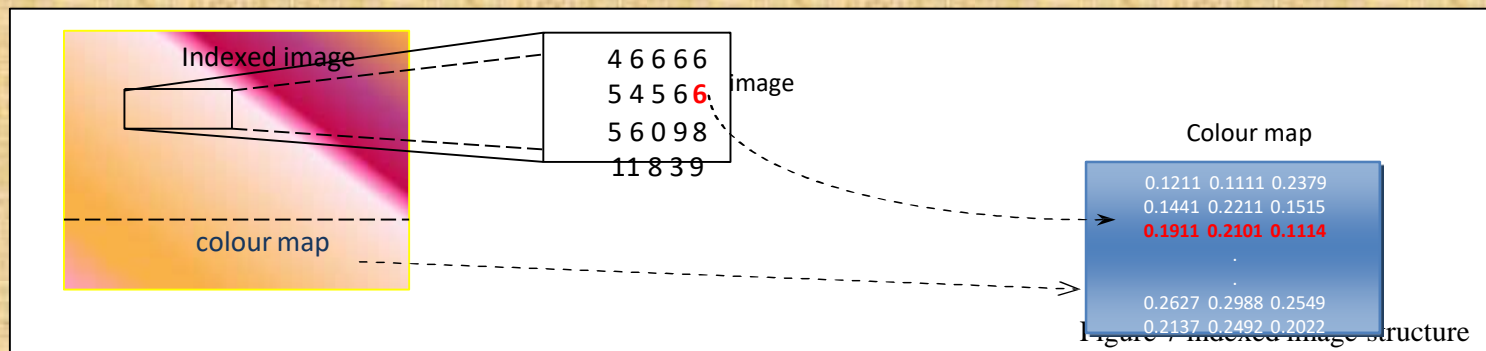


- **The RGB Color System:** to obtain any of the rgb values at a given location, we use the color cube as shown below.



2. The Indexed image:

For convenience of storage and file handling, the image has an associated color map, or color palette which is simply a list of all the colors used in that image. Each pixel has a value which does not give its color directly as for an RGB image, instead an index to the color is exist in the map. If an image has 256 colors or less, the index values will only require one byte to store. Without the color map, the image would be very dark and colorless, e.g., img.tif



Each pixel is a shade of grey, normally from 0 (black) to 255 (white) for unsigned integer data class. The range of intensities means that each pixel can be represented by eight bits or exactly one Byte. This is a very natural range for

image file handling. Other grayscale ranges are used, they are a power of 2. Such images arise in medicine like X-rays

- **RGB to Grey Image Converting:**

The RGB colored image like the pixel $(r,g,b) = (100,0,150)$ could be converted to a gray image using one of the following formula,

1. The **Average** method, simply average the values: $(R + G + B)/3$

$$gray = (100+10+150)/3 = 86.6667 = 87$$

2. The **Lightness** method, averages the most prominent and least prominent colors $(\max(R,G,B) + \min(R,G,B)) / 2$

$$gray = (250 + 10)/2 = 130$$

3. The **Luminosity** method $0.21R + 0.71G + 0.07B$. It is a more sophisticated version of the average method. It also averages the values, but it forms a weighted average to account for human perception. The fact is the humans are more sensitive to green than other colors, so green is weighted most heavily. The formula is,

$$gray = 0.21 R + 0.71 G + 0.07 B = 38.6 = 39$$

The Luminosity method tends to reduce contrast. It works best overall and it is the default method used in most applications. Some images look better using one of the other algorithms. Sometimes the four methods produce very similar result.

4. The **weighted average** method, $0.299R + 0.587G + 0.114B$, note that colors are not weighted equally. Since pure green is lighter than pure red and pure blue, it has a higher weight. Pure blue is the darkest of the three, so it receives the least weight.









$$gray = 0.299 (100) + 0.587(10) + 0.114(150) = 52.87 = 53$$

Visual Examples: The chart below shows colors on the left and equivalent grays on the right, along with the RGB values.

Pure Red (255,0,0)		Equivalent Gray (76,76,76)	
Pure Green (0,255,0)		Equivalent Gray (150,150,150)	
Pure Blue (0,0,255)		Equivalent Gray (29,29,29)	
Cyan (0,255,255)		Equivalent Gray (179,179,179)	
Magenta (255,0,255)		Equivalent Gray (105,105,105)	
Yellow (255,255,0)		Equivalent Gray (226,226,226)	
Brown (158,85,54)		Equivalent Gray (103,103,103)	
Olive (155,160,52)		Equivalent Gray (146,146,146)	
Purple (100,0,150)		Equivalent Gray (47,47,47)	

Can you convert a grayscale value back to an RGB color code?

Because many different colors can have the same grayscale equivalent, it is not possible to match a unique color to each gray value. For example, the colors below all have the same grayscale equivalent value of (100, 100, 100):

(0,170,0)		(100,100,100)	
(230,53,0)		(100,100,100)	
(80,83,240)		(100,100,100)	
(230,14,200)		(100,100,100)	

H.W//Suggest (r,g,b) values such that the difference in equivalent gray for all methods as smallest value.

3. The Binary Images

In this image type, the pixel is black, "0" value or white, "1" value. We only need one bit size to store pixel value. Such images can therefore be very efficient in terms of storage and it is suitable for text, fingerprints, or architectural planes.

Example 1:

A binary image of (512× 512) dimensions. The number of bits needs to store the file size of this image is as follows:

$$\begin{aligned} \text{File size of binary image} &= 512 * 512 = 262,144 \text{ bits} \quad \text{or} \\ &= 32,768 \text{ Byte} \\ &= 32 \text{ KB} \\ &= 0.03125 \text{ MB} \end{aligned}$$

$$\begin{aligned} \text{or File size of gray-scale image} &= 512 * 512 * 8 = 2,097,152 \text{ bits} \\ &= 262,144 \text{ Byte} \\ &= 256 \text{ KB} \\ &= 0.25 \text{ MB} \end{aligned}$$

$$\begin{aligned} \text{File size of RGB image} &= 512 * 512 * 8 * 3 = 6,291,456 \text{ bits} \quad \text{or} \\ &= 786,432 \text{ Byte} \\ &= 786 \text{ KB} \\ &= 0.75 \text{ MB} \end{aligned}$$

Example 2:

An 8bits image in jpg format at 200 dot per inch (200 dpi). The image size is (450×450) pixels. Calculate its current dimension in inch. If this image is compressed and stored in a new size of (1.5×1.5) inch with same number of pixels, calculate its new dpi.

$$\begin{aligned} \text{Current size in inch} &= 450 \text{ pixel} / 200 \text{ dpi} = 2.25 \text{ inch (for one dimension)} \\ &= 2.25 \times 2.25 \text{ inch} \end{aligned}$$

$$\text{New dpi} = 450 \text{ pixel} / 1.5 \text{ inch} = 300 \text{ dpi}$$

Colour System Models 1.5

For human beings, colour provides one of the most important descriptors of the world around us. The human visual system is particularly attuned to two things: *edges, and colour*. The human visual system is not particularly good at recognizing subtle changes in grey values. In this section we shall investigate colour briefly, and then some methods of processing colour images. The human visual system tends to perceive colour as being made up of varying amounts of red, green and blue. That is, human vision is particularly sensitive to these colours; this is a function of the cone cells in the retina of the eye. These values are called the primary colours. If we add together any two primary colours we obtain the secondary colours: for example,

Magenta (purple) = red + blue Cyan = green + blue
Yellow = red + green

A colour model is a method for specifying colours in some standard way. It generally consists of a three dimensional coordinate system and a subspace of that system in which each colour is represented by a single point. We shall investigate three systems.

1. The RGB Model

In this model, each colour is represented as three values R, G, and B, indicating the amounts of red, green and blue which make up the colour. This model is used for displays on computer screens; a monitor has three independent electron "guns" for the red, green and blue component of each colour. This model is already explained in the past sections

2. The HSV Model

Hue: The "true colour", it attributes (red, green, blue, orange, yellow, and so on) **Saturation:** The amount by which the colour has been diluted with white. The more white in the colour, the lower the saturation. So a deep red has high saturation, and a light red (a pinkish colour) has low saturation.

Value: The degree of brightness: a well lit colour has high intensity; a dark colour has low intensity. This is a more intuitive method of describing colours, and as the intensity is independent of the colour information, this is a very useful model for image processing. We can visualize this model as a cone, as shown in figure 5. Any point on the surface represents a purely saturated colour. The saturation is thus given as the relative distance to the surface from the central axis of the structure. Hue is defined to be the angle measurement from a pre-determined axis, say red.

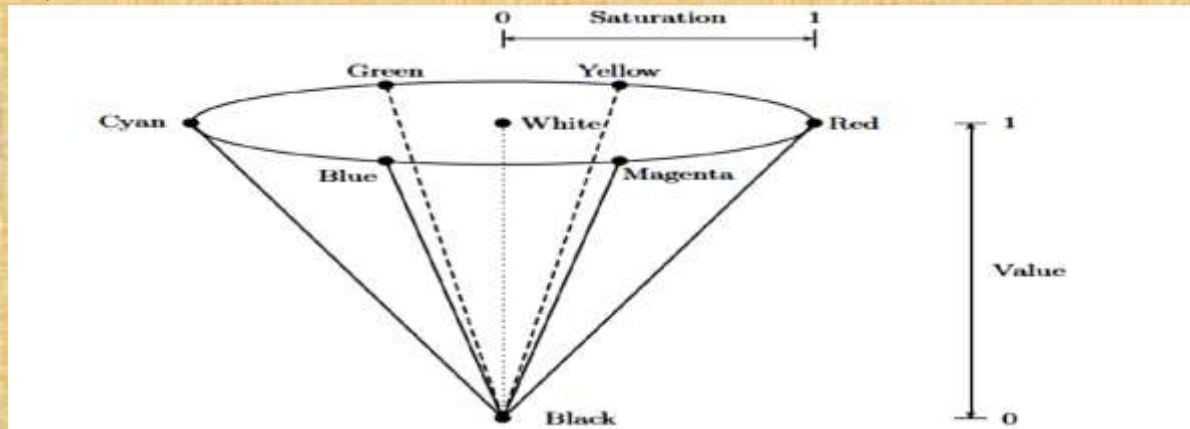


Figure 8 The HSV colour space cone

- **Conversion between RGB and HSV**

Suppose a colour is specified by its RGB values. If all the three values are equal, then the colour will be a grey scale; that is, an intensity of white. Such a colour, containing just white, will thus have a saturation of zero. Conversely, if the RGB values are very different, we would expect the resulting colour to have a high saturation. In particular, if one or two of the RGB values are zero, the saturation will be one, the highest possible value. Hue is defined as the fraction around the circle starting from red, which thus has a hue of zero. Reading around the circle in Figure 8 produces the following hues

Colour	Hue
Red	0
Yellow	0.1667
Green	0.3333
Cyan	0.5
Blue	0.6667
Magenta	0.8333

Suppose we are given three R, G, B values, which we suppose to be between 0 and 1. So if they are between 0 and 255, we first divide each value by 255. We then define:

$$\begin{aligned} V &= \max\{R, G, B\} \\ \delta &= V - \min\{R, G, B\} \\ S &= \frac{\delta}{V} \end{aligned}$$

To obtain a value for Hue, we consider several cases:

1. if $R = V$ then $H = \frac{1}{6} \frac{G - B}{\delta}$,
2. if $G = V$ then $H = \frac{1}{6} \left(2 + \frac{B - R}{\delta} \right)$,

$$3. \text{ if } B = V \text{ then } H = \frac{1}{6} \left(4 + \frac{R - G}{\delta} \right).$$

If H ends up with a negative value, we add 1. In the particular case $(R, G, B) = (0, 0, 0)$, for which both $V = \delta = 0$, we define $(H, S, V) = (0, 0, 0)$.

For example, suppose $(R, G, B) = (0.2, 0.4, 0.6)$ We have

$$V = \max\{0.2, 0.4, 0.6\} = 0.6$$

$$\delta = V - \min\{0.2, 0.4, 0.6\} = 0.6 - 0.2 = 0.4$$

$$S = \frac{0.4}{0.6} = 0.6667$$

Since $B = G$ we have

$$H = \frac{1}{6} \left(4 + \frac{0.2 - 0.4}{0.4} \right) = 0.5833.$$

Conversion in this direction is implemented by the `rgb2hsv` function. This is of course designed to be used on $m \times n \times 3$ arrays, but let's just experiment with our previous example:

```
>> rgb2hsv([0.2 0.4 0.6])
ans =
    0.5833    0.6667    0.6000
```

and these are indeed the H , S and V values we have just calculated.

1.5.3 The NTSC Model

This colour space is used for TV/Video in America and other countries where NTSC is the video standard (Australia uses PAL). In this scheme Y is the Luminance, I and Q carry the colour information. The linear conversion from YIQ to RGB is straight forward:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

and

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

The two conversion matrices are of course inverse of each others. Note the difference between Y and V :

$$Y = 0.299R + 0.587G + 0.114B \quad V = \max(R, G, B)$$

This reflects the fact that the human visual system assigns more intensity to the green component of an image than to the red and blue components.

Colour images in Matlab1.0

Since a colour image requires three separate items of information for each pixel, a (true) colour image of size $m \times n$ is represented in MATLAB by an array of size $m \times n \times 3$: a three dimensional array. We can think of such an array as a single entity consisting of three separate matrices aligned vertically.

```
>> x=imread('lily.tif');
>> size(x)

ans =

    186    230     3
```

We can isolate each colour component by the colon operator:

```
x(:,:,1)  The first, or red component
x(:,:,2)  The second, or green component
x(:,:,3)  The third, or blue component
```

These can all be viewed with `imshow`:

```
>> imshow(x)
>> figure,imshow(x(:,:,1))
>> figure,imshow(x(:,:,1))
>> figure,imshow(x(:,:,2))
```

These are all shown in figure 9. Notice how the colours with particular hues show up with high

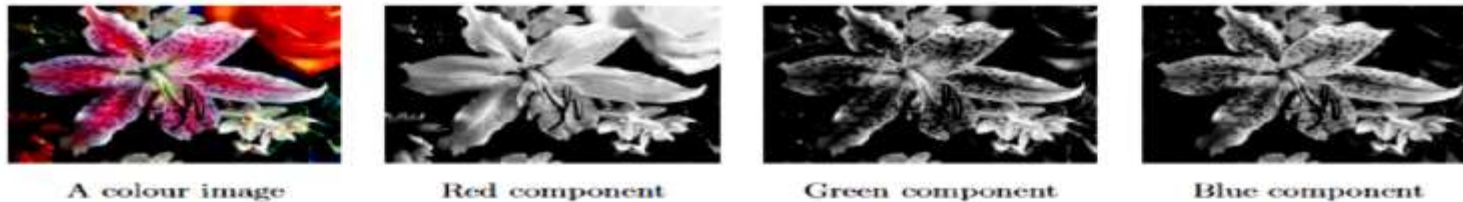


Figure 9: An RGB colour image and its components

intensities in their respective components. For the rose in the top right, and the flower in the bottom left, both of which are predominantly red, the red component shows a very high intensity for these two flowers. The green and blue components show much lower intensities. Similarly the green leaves—at the top left and bottom right—show up with higher intensity in the green component than the other two.

We can convert to YIQ or HSV and view the components again:

```
>> xh=rgb2hsv(x);
>> imshow(xh(:,:,1))
>> figure,imshow(xh(:,:,2))
>> figure,imshow(xh(:,:,3))
```

and these are shown in figure 10. We can do precisely the same thing for the YIQ colour space:

```
>> xn=rgb2ntsc(x);
>> imshow(xn(:,:,1))
```

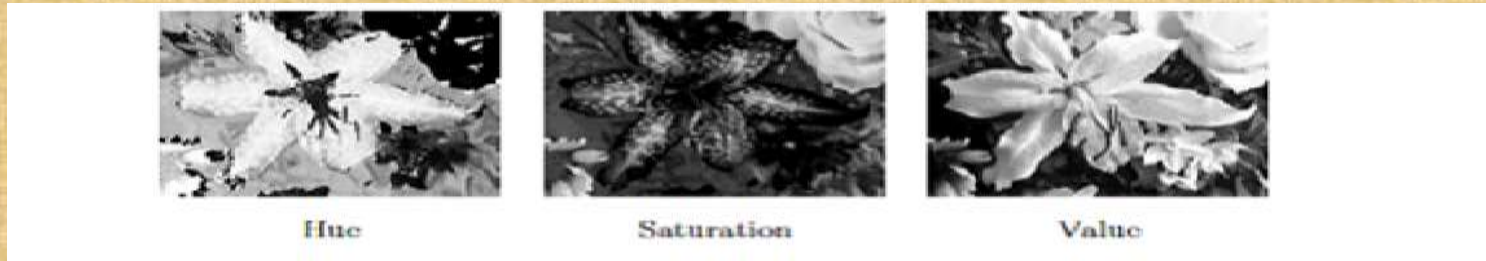



Fig. 10 The HSV Components

```
>> figure,imshow(xn(:,:,2))  
>> figure,imshow(xn(:,:,3))
```

and these are shown in figure 11. Notice that the Y component of YIQ gives a better grayscale



Fig. 11 The YIQ Components

2

Chapter Two Image Arithmetic

1. Types of Image processing

The processing operation will transform image pixels or gray values into different values and/or data class depending on the information required to perform this transformation. Generally we can divide image processing into three types:

1. Point Processing

A pixel gray value is changed without any knowledge about pixel's neighbours, like image matrix addition, subtraction, multiplication..., etc.

2. Neighbourhood processing

To change the pixel or gray level value we only need to know the interested pixel neighbour's values, like image filtering for noise removal and features extraction.

3. Transforms

The entire image is processing and transforming as a single large block. We may need to transform an image from its spatial domain into another domain like frequency domain. Many image transforms will be consider next chapters like, Fourier, Wavelet, or Cosine transform



Fig. 2.1 Scheme of image transforming Process

2. Arithmetic Operations on Image Matrix

• Addition/Subtraction

The arithmetic operations like addition, subtraction, multiplication, and division can be implemented on image pixels individually. The result of all these operations will effect on the pixel intensity $f(x,y)$. So,

$$f(x,y)_{\text{new}} = f(x,y)_{\text{oSd}} \pm C$$

C: is any constant value

If image matrix in the range of (0 ... 255) like uint8 data class, we should rounding and clipping all values that fall out of the range. i.e;

$$f(x, y)_{\text{new}} = \begin{cases} 255 & \text{if } f(x, y) > 255 \\ 0 & \text{if } f(x, y) < 0 \end{cases}$$

Ex: Sketch the relation between the old and new pixel values after adding and subtracting a constant $C=128$

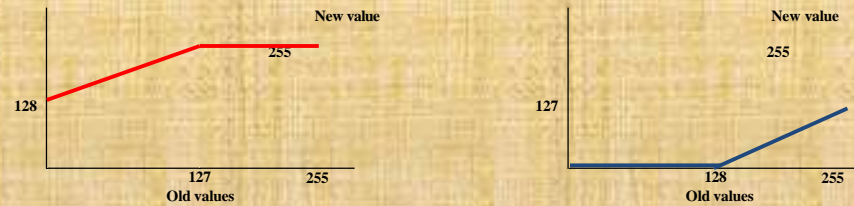


Fig. 2.2 image intensities after addition and subtraction of a constant value C

From two figures, we observe that in general adding a constant will lighten an image and subtracting a constant will darken it.

- **Multiplication/Division**

Lightening or darkening of an image can be done by multiplication operation

Ex: Sketch the relation between the old and new pixel values for the following operations.

- $f(x, y)_{\text{new}} = f(x, y)_{\text{old}} * 2$
- $f(x, y)_{\text{new}} = f(x, y)_{\text{old}} \div 2$
- $f(x, y)_{\text{new}} = (f(x, y)_{\text{old}} \div 2) + 128$

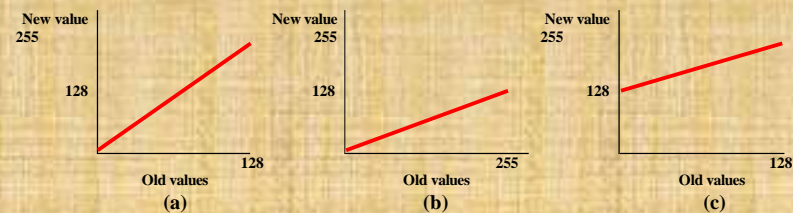


Fig. 2.3 image intensities after multiplication and division operations

Let us consider the image “*saturnstar.jpg*” as an example for the following MATLAB operations,

```
>> b = imread ('saturnstar.jpg');
>> [b,map] = rgb2ind (b,64);
>> whos b      → Name      size      byte      class
                b      984x1270  9997440   double

>> b1 = uint8( double (b) + 64 )
>> b2 = uint8 ( double (b) - 64)

or, using the built-in function

>> b3 = imadd (b2, 64)
>> b4 = imsubtract (b2,64)
```

What is the error in the following two commands?



HW//

Write a MATLAB code to read a jpg image and display the RED, GREEN, and BLUE matrices individually. Re-display the individual matrices after making the following change.

Red=Red+75 Green=Green+50 Blue=Blue-128

Image Resolution2.3

Image resolution can be divided into two types, *spatial resolution* and *effective resolution*. Spatial resolution is the density of pixels over the image, the greater the spatial resolution, the more pixels are used to display the image. Effective resolution simply, it is a spatial resolution without any repeated or duplicated of image pixels. Suppose we have a 256×256 8bits gray-scale image is saved to the matrix X. The image matrix could be resized into half, quarter...etc. as explain below,

```
>>X1= imresize(X, 0.5);
```

This command will split X matrix into even rows and columns, then a new image matrix of 128×128 will be created. From figure 2.4, the components of X1 would be the intersections of even rows and columns. The image spatial resolution is halved but its effective resolution is still same. Now, if the image is resized a gain to obtain the original 256×256 image matrix, we apply

```
>>X2= imresize(X1, 2);
```

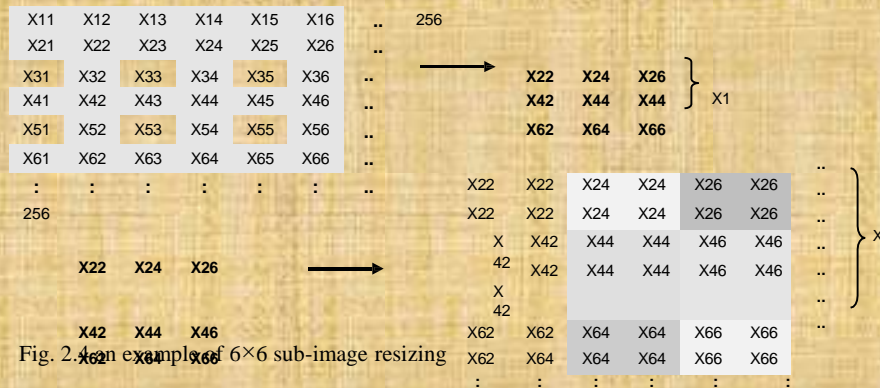


Fig. 2.4 an example of 6×6 sub-image resizing

Each pixel is duplicated four times and the spatial resolution of matrix X2 is returned to 256×256 , but the effective resolution is still halved to 128×128 . We can do all this in one line, see example below.

Commands	Effective resolution
<code>imresize (imresize (X,0.25) , 4)</code>	64×64
<code>imresize (imresize (X,1/16) , 16)</code>	16×16
<code>imresize (imresize (X,1/32) , 32)</code>	8×8

2.4 Image Matrix Complement

The complement of a gray-scale image is a negative photographic of this image. Depending on the image data class, complement of image matrix is,

1. Binary image, $0s \rightarrow 1s$ and $1s \rightarrow 0s$

```
>> m = im2bw( [1 0] ) → 1 0
```

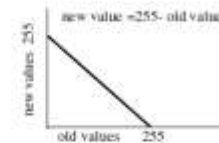
```
>> ~m → 0 1
```

2. Scaled-double image, 1 – pixel value

```
>> imcomplement (image matrix name)
```

3. Uint8 image, 256 – pixel value

```
>> imcomplement (image matrix name)
```



- Solarization

A part complementing of an image, for example by taking the complement of pixels of gray values 128 or less and leaving other pixels unchanged or we could complement the pixels which are 128 or greater and leave other pixels unchanged. See figure 2.5 below.

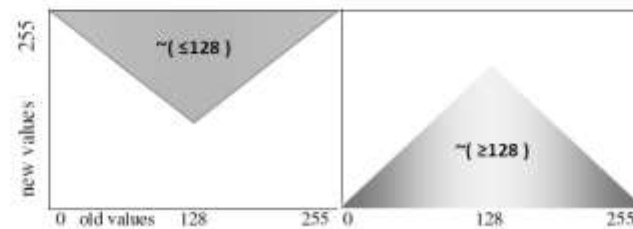


Fig. 2.5 Solarization property

HW// Write a MATLAB program to solarize a gray image using for-loop statement only

Image Enhancement: Image Histogram2.5

Image Histogram is a graph that indicates the number of occurrence (times) of a gray level (intensity) in the image. It provides important information about:

- In a dark image, the gray level would be clustered at the lowest level (left side of histogram)
- In a bright image, the gray level would be clustered at the highest level (right side of histogram)
- In a well contrasted image, the gray-level would be well spread out over the range

Ex//

```
>> I = imread(saturnstar.jpg);
>> imshow(I)
>> figure
>> imhist(I(:,1)) , axis tight /* for automatically scaled to fit all values in */
```

Given a poorly contrasted image, we can enhance its contrast by re-spreading out its gray-level and hence its histogram by using the two major methods:

where:

b_i, b_{i+1} : are the lowest and highest of target gray-level

a_i, a_{i+1} : are the lowest and highest of current gray-level

X: is the current gray-level value of image

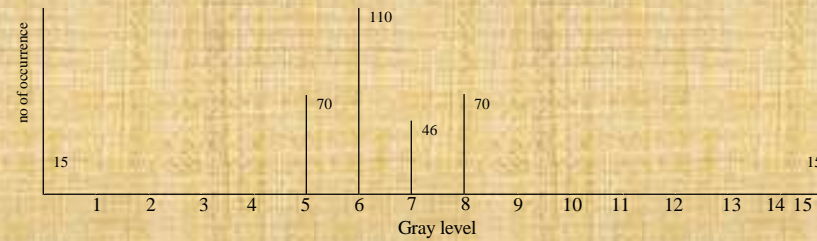
Y: is the new (re- spreading) gray-level value

Ex// Suppose an image with the following gray-level values,

Gray-level (i)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n(i)	15	0	0	0	0	70	110	46	70	0	0	0	0	0	0	15

For this image, we have

- 16 gray levels ranging from 0 to 15.
- total number of pixels, $n = 5+70+110+46+15$ or image $_{16 \times 16}$
- Image pixels are clustered in the middle



Let: $b_1 = 2$, $b_2 = 14$, $a_1 = 5$, $a_2 = 8$

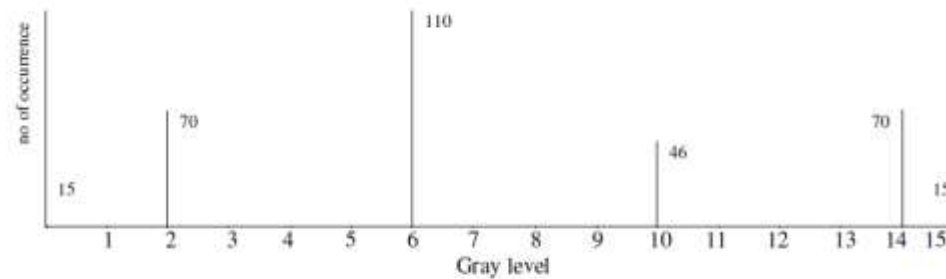
$$Y(5) = (14 - 2) / (8 - 5) * (5 - 5) + 2 = 2$$

$$Y(6) = (14 - 2) / (8 - 5) * (6 - 5) + 2 = 6$$

$$Y(7) = (14 - 2) / (8 - 5) * (7 - 5) + 2 = 10$$

$$Y(8) = (14 - 2) / (8 - 5) * (8 - 5) + 2 = 14$$

$x(i)$	$y(i)$
5	2
6	6
7	10
8	14



HW// Suppose new different values for $b_1 = 1$, $b_2 = 13$, re-spread the middle clustered values in the previous example.

2.5.2 Histogram Equalization

This method provides good results in re-distribute the intensities of image using a standard distributed formula. For any gray image, if L is the number of gray levels i.e.; for 32 gray level, $L = 32$, so the image intensity range will be: $0 \rightarrow L - 1$, and that gray level i occurs n_i times in the image. If the total number of pixel is n , this yields $n = n_0 + n_1 + \dots + n_{L-1}$ or $n = \sum n(i)$

To transform the gray levels to obtain a better contrasted image we change gray level i to $\frac{L-1}{n} \sum n_i$

Ex//

Suppose a 4-bit greyscale image has the histogram shown in the table below. Draw the histogram graph and make a necessary transformation of gray values.

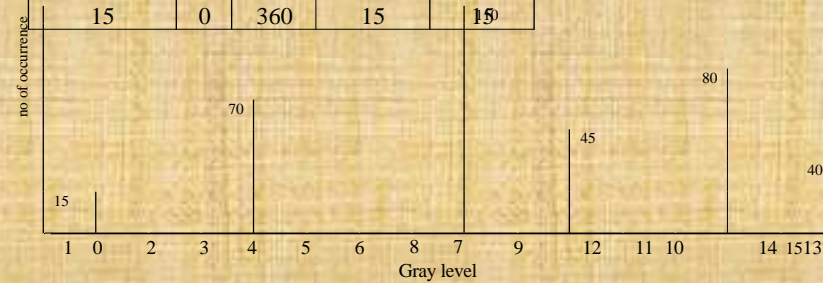
Gray-level (i)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n(i)	15	0	0	0	0	0	0	0	0	70	110	45	80	40	0	0



Histogram indicates poor contrast

With $n = 360$ we expect this image to be uniformly bright with few dark dots. To equalize this histogram, we form running total of the n_i and multiply each by $15/360 = 1/24$. We now have the following transformation of gray values obtain by reading off the first and the last columns in the table below.

Gray level (i)	$n(i)$	$\Sigma n(i)$	$\frac{15}{n(i)} \Sigma_{360}$	Rounde d value
0	15	15	0.63	1
1	0	15	0.63	1
2	0	15	0.63	1
3	0	15	0.63	1
4	0	15	0.63	1
5	0	15	0.63	1
6	0	15	0.63	1
7	0	15	0.63	1
8	0	15	0.63	1
9	70	85	3.65	4
10	110	195	8.13	8
11	45	240	10	10
12	80	320	13.33	13
13	40	360	15	15
14	0	360	15	15
15	0	360	15	15



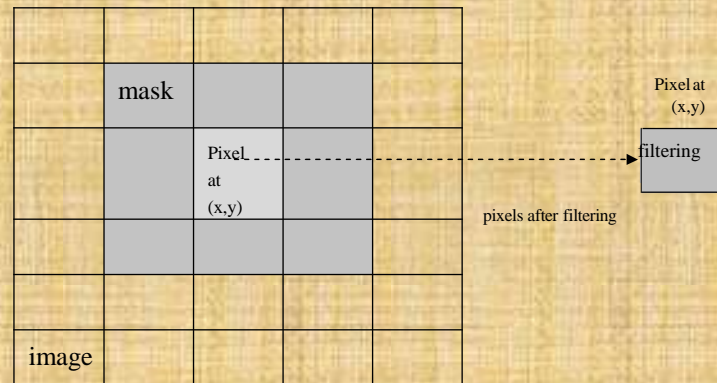
Histogram after equalization

3

Chapter Three

Neighbourhood Processing 3.1

Neighbourhood Processing is an extension of point or pixel processing, where a function is applied to the interested pixel and its neighbours. The idea is to move a mask which is normally an odd length rectangle over a given image. See figure 3.1



The combination of mask and function is called filter. If the function of the new gray value is a linear of all gray values in the mask, then the filter is called a linear filter.

3.2 Spatial Filtering Procedure

- Project the mask center over the sub-image pixel
- Compute the product of mask elements with the pixel and its neighbours using **dot Product** (\cdot) operation
- Add up the production result to find the new filtered pixel value. These operations (addition of multiplication) is called **image convolution**.

$$\sum_{s=-1}^1 \sum_{t=-1}^1 m(s,t) t(i+s, j+t)$$

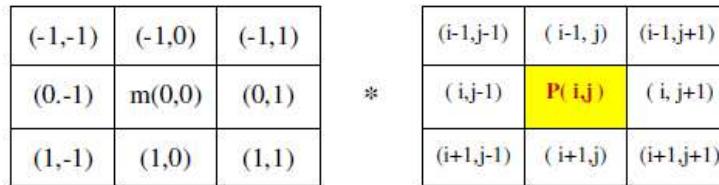


Fig. 3.2 The mask $m(s,t)_{3 \times 3}$ The sub-image of pixel $p(i,j)$

The multiplication is a dot product process between the mask and interested sub-image:

$$\begin{aligned}
 \text{new pixel} = & m(-1,-1)p(i-1,j-1) + m(-1,0)p(i-1,j) + m(-1,1)p(i-1,j+1) + \\
 & m(0,-1)p(i,j-1) + \dots + \dots + \\
 & \dots + \dots + m(1,1)p(i+1,j+1)
 \end{aligned}$$

3.3 Image Frequencies

The frequencies of an image are a measure of the amount by which gray values change with respect to the distance.

- High frequency components are characterised by large change in gray values over small distances, like; edges, and noise in the image
- Low frequency components are characterised by small or little change in the grey values. It may includes backgrounds, texture,etc

3.4 Low Pass Filters

Depending on the purpose of filtering, Filters could be classified into *Low Pass Filters (LPF)* and *High Pass Filters (HPF)*, other filters like *Band Pass Filters (BPF)* or *Band Reject Filters (BRF)* will be discussed later in chapter 4 and chapter 7. The LPF passes the low frequency components and reduces or eliminates the high frequency components

- **Average/Mean Filter**

It is one of the simplest and important linear filters. The mask size could be 3x3, 5x5, 3x5, 5x3 ,.... etc. This kind of filter is used to block the sudden changes (high frequencies) like noise and passes the low frequencies like image background. The very sharp cut-off frequency the more smeared in

the output image, so this type of filter tends to smear image edges. Any 3x3 mean filter can be represented by the following matrix.

$$\begin{array}{|c|c|c|} \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array} = 1/9 \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Fig. 3.3 Mean/Average mask structure

Ex// Apply the mean filter on the sub-image below and find the output.

$$\begin{array}{|c|c|c|} \hline a & b & c \\ \hline d & e & f \\ \hline g & h & i \\ \hline \end{array} \quad \text{new pixel}(e) = 1/9 (a + b + c + d + e + f + g + h + i)$$

Ex// In this example, we apply the MATLAB commands to filter a magic (5x5) image

```
>> I = uint8 ( 10* magic (5) )
```

170	240	10	80	150
230	50	70	140	160
40	60	130	200	220
100	120	190	210	30
110	180	250	20	90

Stochastic rows = columns = diagonal

```
>> mean2( x(1:3, 1:3) ) → 111.111 ≈ 111
>> mean2( x(1:3, 2:4) ) → 108.889 ≈ 109
```

and so, we pass the mask over other pixels, the filtered image dimension would be (3x3), why?

111	109	129
50	70	140
110	130	150
60	130	200
131	151	149
120	190	210

- **Type of Filtering Operations**

Image circumference or edge should be considered when the mask partly falls outside image. In such case, there will be lack of gray values to use in the filter function. Two different approaches deal with this problem.

First, ignore the edges: the mask is only applied to those pixels in the image as in the last example. With this example we ignored processing 16 pixels out of 25, those unprocessed pixels are the circumference or edge of $I_{(5 \times 5)}$ image. The signal length for convolution any two signals is subjected to the formula,

$$Y_{[m+n-1]} = X_m * h_n$$

and this interprets the reduction in size of the image $I_{(5 \times 5)}$ to be $I_{(3 \times 3)}$ after filtering. As a mask size is enlarged, a significant amount of information may be lost. In MATLAB this filtering operation is called, "*valid*".

Second, pad with zeros: in order to include image edge in the filtering, we surround image with zero pixels. This will return the filtered image to its original size, and the operation is, "*Same*". If a second zero padding is added the filtered image size will be larger than the origin one and the operation now is calling a "*full*" filtering operation. See figure 3.4

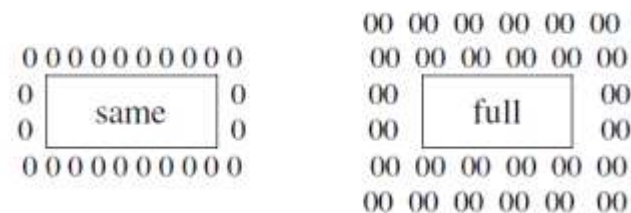


Fig. 3.4 Zeros Padding

- **Filtering in MATLAB**

filter2 (*filter*, *image*, 'shape') and *imfilter* (*image*, *filter*, 'shape')

Are the most two built-in functions they are used for image filtering, shape is optional and it describes the method for dealing with edges. It either “*valid*”, “*same*”, or “*full*” operation. The result is a matrix of double data type and the default shape is same.

Ex-1//

```
>> x = uint8( 10* magic(5) )
>> a = ones (3,3)/3 ;
>> x_same = filter2 (a, x, 'same' );
>> x_valid = filter2 (a, x, 'valid');
```

The result of the ‘same’ may also be obtained by padding x with zeros and using ‘valid’

```
>> x2 = zeros (7,7);
>> x2(2:6, 2:6) = x
>> filter2 (a, x2, 'valid');
>> Xf = filter2 (a, x, 'full')
```

The returned result, $Xf_{(7 \times 7)}$ is larger than the original $x_{(5 \times 5)}$:

where, $7 = \text{order of the matrix} + \text{order of the filter} - 1$

It does this by padding with zeros and applying the filter at all places on and around the image where the mask intersects the image matrix

Another option to create filter is by using the *fspecial* function which has many options for easy creation of many different filters.

```
>> mask = fspecial ('average', [ 5 7 ] )    % average filter (5 by 7 is created)
>> mask = fspecial ('average', [ 1 1 ] )    % The default mask is (3x3)
```

Ex-2//

```
>> c = imread ('cameraman.jpg');
>> f = fspecial('average');
>> cf = filter2 ( f, c (:, :, 1) );
>> imshow (cf); figure ; imshow ( mat2gray (cf) );
```

3.5 High Frequency Filters

In the low frequency image where the gray values are similar, the result of

Ex//

150	152	148	149		
147	152	151	150	→	11 6
142	148	149	151		-13 -5
151	149	150	148		

In this example, we considered a 4×4 block of similar pixel values. After HP filtering, the result is close to zeros of the filtered sub-image which is the expected result for low frequency components.

- The Laplacian Filter

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. It highlights regions of rapid intensity change like edge detection.

$$\text{Laplacian } (3 \times 3) = \begin{pmatrix} 0.1667 & 0.6667 & 0.1667 \\ 0.6667 & -3.333 & 0.6667 \\ 0.1667 & 0.6667 & 0.1667 \end{pmatrix}$$

Ex-2//

```
clc,close all, clear aa;
I= imread ('C:\Program
Files\MATLAB\R2008a\toolbox\images\imdemos\office_6.jpg');
f1 = fspecial('average',[15] );
f2 = fspecial('laplacian');
If1 = filter2 ( f1 , I( : , : ,3) );
If2 = imfilter( I( : , : ,3), f2 );
imshow (I ); figure ;
imshow (mat2gray(If1)); figure;
imshow ( (50 * If2 ) )
```




Fig. 3.5
Original image a)



Low pass filtered b)
image



High pass filtered c)
image

3.6 Contrast Enhancement

The contrast of coloured image like RGB and YIQ could be enhanced using two different ways. First, the intensity matrix Y in YIQ model can be enhanced individually and the effect will be appeared on the original coloured image as Y matrix contains all information related to the intensity. The second method is to enhance the R, G, and B matrix individually in RGB model and the enhancement will appear on the original one. Suppose a “*cat.tif*” image wants to be enhanced by applying the two methods.

```
>>[x,map] = imread('cat.tif');
>>crgb = ind2rgb(x,map);
>>cntsc = rgb2ntsc(crgb);
```

Apply the histogram to the intensity components (Y) matrix

```
>> cntsc(:, :, 1) = histeq(cntsc(:, :, 1));
>>cntsc2 = ntsc2rgb(cntsc)
>> imshow(cntsc2)
```

Now, we apply the enhancement on individual RGB matrices

```
>> ctr = histeq(crgb(:, :, 1));
>> ctg = histeq(crgb(:, :, 2));
>> ctb = histeq(crgb(:, :, 3));
```

We put them all back into a single 3 dimensional array using the concatenate function “*cat()*”,

```
>> crgb2 = cat(3,ctr,ctg,ctb);
>> imshow(crgb2)
```

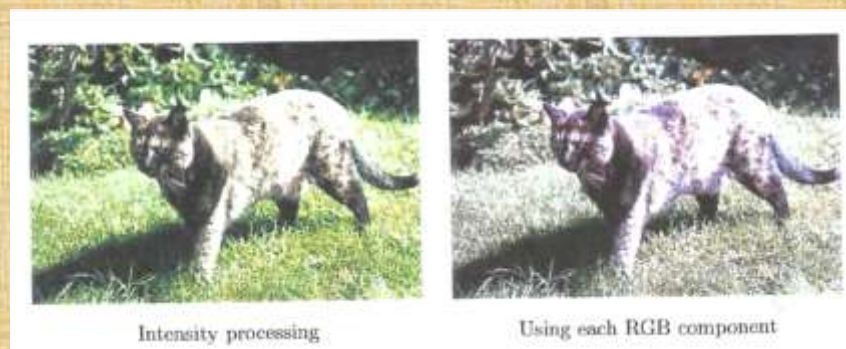


Fig. 3.6 contrast enhancement

Another important filter is called **L**aplace of **G**aussian filter “*log*” could be used for smoothing image and edge detection in the same time,

```
>> fspecial('log')
ans =
    0.0448 0.0468    0.0564 0.0468 0.0448
    0.0468 0.3167    0.7146 0.3167 0.0468
    0.0564 0.7146   -4.9048 0.7146 0.0564
    0.0468 0.3167    0.7146 0.3167 0.0468
    0.0448 0.0468    0.0564 0.0468 0.0448
```

For (3 3) mask only,

```
>> fspecial('log',3)
ans =
    0.4038 0.8021 0.4038
    0.8021 -4.8233 0.8021
    0.4038 0.8021 0.4038
```


4

Chapter Four

Frequency Domain

The 2D Discrete Fourier Transform

1. Fourier Transform Properties

- Fourier Transform (FT) is performing many tasks which would be impossible to perform in any other ways. The advantages of FT in image processing field could be summarised as:
 1. A powerful alternative method to linear spatial filtering
 2. Fast and More efficient than large spatial filters
 3. Process and isolate some particular frequencies
 4. Perform LP and HP filters in very high precision
 5. Perform Band Reject Filters for image restoration
- A periodic function may be written as the sum of sines and cosines of varying **amplitudes** and **frequencies**

$$f(x) = \sin(x) + 1/3 \sin(2x) + 1/5 \sin(4x) + 1/7 \sin(8x)$$

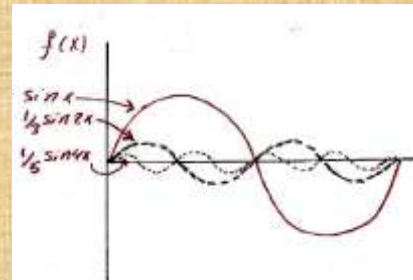


Fig 4.1 different amplitudes and frequencies for 1D periodic signals

and for a square wave

$$f(x) = \sin x + 1/3 \sin 3x + 1/5 \sin 5x + 1/7 \sin 7x$$

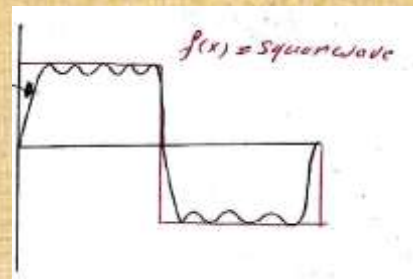


Fig 4.2 square wave components

The one dimension Discrete Fourier Transform is,

$$F(u) = a_0 + \sum_{x=1}^M f(x) \left(a_n \cos \frac{xu\pi}{M} + b_n \sin \frac{xu\pi}{M} \right)$$

or in exponential form,

$$F(u) = \sum_{x=0}^{M-1} f(x) e^{-2\pi j \left(\frac{xu}{M} \right)}$$

The inverse FT is,

$$f(x) = 1/M \sum_{u=0}^{M-1} F(u) e^{2\pi j \left(\frac{xu}{M} \right)}$$

The 2D dimensions Discrete Fourier Transform is,

$$F(u, v) = \sum_{N=0}^{N-1} \sum_{M=0}^{M-1} f(x, y) e^{-2\pi j \left(\frac{xu}{M} + \frac{yv}{N} \right)}$$

And, the inverse of 2D DFT would be,

$$f(x, y) = 1/MN \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} F(u, v) e^{2\pi j \left(\frac{xu}{M} + \frac{yv}{N} \right)}$$

- **Similarity**

The inverse and forward transforms are very similar with the exception of scale factor $1/MN$ and the negative sign.

So the term $e^{-/+2\pi j \left(\frac{xu}{M} + \frac{yv}{N} \right)}$ is look like a DFT spatial filter because:

The value of f and F are independent and could be calculated separately, so the value of $F(u, v)$ is obtained by multiplying every value of $f(x, y)$ by a fixed value, $e^{-2\pi j \left(\frac{xu}{M} + \frac{yv}{N} \right)}$ and adding up all the result and this is what a linear filter does where the convolution in spatial linear filter multiplies all elements under a mask with fixed values and adds them all up.

- **Separability**

The 2D DFT (filter elements) can be expressed as product of,

$$e^{-2\pi j\left(\frac{xu}{M} + \frac{yv}{N}\right)} = \underbrace{e^{-2\pi j\left(\frac{xu}{M}\right)}}_{\text{depends on } x \text{ \& } u \text{ only}} \cdot \underbrace{e^{-2\pi j\left(\frac{yv}{N}\right)}}_{\text{depends on } y \text{ \& } v \text{ only}}$$

so the 2D DFT can be calculated by using the separability property, we first compute the DFT for all rows and then complete the DFT of all columns of the result,

$$F(u/v) = \sum_{x,y=0}^{M-1,N-1} f(x/y) e^{-2\pi j\left(\frac{xu}{M} + \frac{yv}{N}\right)}$$



Fig . 4.3 Apply 2D DFT as a 1D DFT

- **Linearity**

$$FT(f + g) = FT(f) + F(g)$$

$$FT(kf) = k F(f)$$

noise can be reduced or removed depending on this property,

$$d = f + n$$

$$F(d) = F(f) + F(n)$$

Where: d: degraded image, f: original image, n: noise

Some noise appears on the DFT in a way that makes it easy to remove

- **Digital Convolution**

Suppose we wish to convolve the large image M with a small mask S, This method of convolution is very slow, so we pad S with zeros to be with same size of image M and do the multiplication

$$H = M * S$$

$$F(H) = F(M) \cdot F(S) \quad ((\text{convolution in time domain equals multiplication in frequency domain}))$$

and the inverse would be, $F^{-1}(H) = F^{-1}(F(M) \cdot F(S))$

- **The DC coefficients**

The dc situation is satisfied when all frequency equal to zero. That means $(u=0, v=0)$ in the transformation formula

$$F(0,0) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} f(x,y) e^{-2\pi j(\frac{x0}{M} + \frac{y0}{N})}$$

So,

$$F(0,0) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} f(x,y)$$

That means the summation of pixels in the original image equal to the DC

H.W//

The convolution of $M_{512 \times 512}$ and $S_{32 \times 32}$ is required $(512)^2 \times (32)^2$ multiplication process to be completed. How many process of multiplications do the need for 2D DFT.

- **Conjugate Symmetry & Shifting**

It is convenient to have the DC component in the center of the transformed matrix for the purpose of display. Also, the symmetry property of the Fourier transform could be obtained by substituting

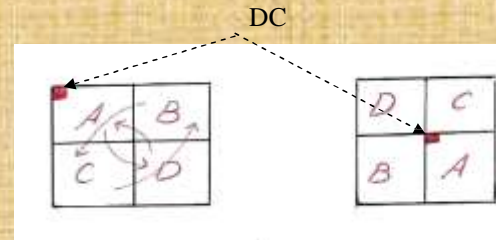
$(u = -u)$ and $(v = -v)$ in the 2D DFT formula

$$F(u, v) = \sum_{N=0}^{N-1} \sum_{M=0}^{M-1} f(x, y) e^{-2\pi j(\frac{xu}{M} + \frac{yv}{N})}$$

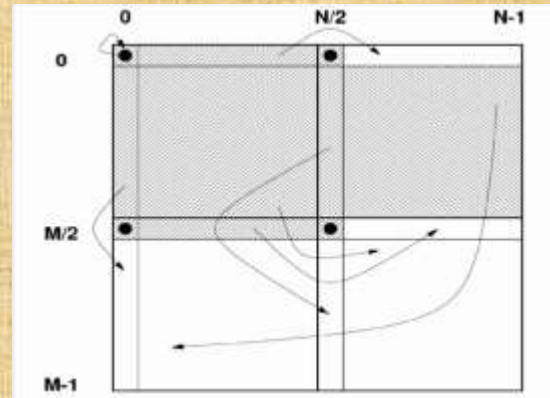
The sign will be +ve

$$F(u, v) = F(-u + pM, -v + qM)^*$$

There are mirror sub-images of conjugate spectrums between the top and bottom halves or left and right halves. The symmetry means that its information is given in just half of a transformation and the other half is redundant, see figure 4.4 a and b.



(a) DC and shifting property



- **MATLAB built-in function for 2D DFT**

- `fft2 (image matrix)`
- `ifft2 (image matrix)`
- `fftshift (image matrix)`
- `fftshift (fft2 (image matrix))`

- **Ex-1**

```
>> A = ones (8)
>> Af = fft2 (A)
```

```
Ans = 640000000
      0
      :
      0 0000000
```

An image is considered as a two dimensional function $f(x,y)$ and can be expressed as sum of **corrugation** functions having the general form,
 $Z = a \sin (bx + cy)$. In this example, no corrugations are required to form a constant.


```

• Ex-2:
>> B = [ 100 200; 100 200]; % a matrix B in this example consisting a single corrugation
>> B = repmat (B, 4,4)
>> BF = fft2(B)

```

```

ans =   100 200 100 200 100 200 100 200   9600  0  0  -3200  0  0  0  0
       100 200 100 200 100 200 100 200   0   0  0  0  0  0  0  0
       :   :   :   :   :   :   :   :   :   :   :   :   :   :   :
       100 200 100 200 100 200 100 200   0   0  0  0  0  0  0  0

```

Note: The DC = 9600 = 150*64, the mirroring of values about the dc coefficient is a sequence of the symmetry of the DFT which is required to form an edge.

There is one way to display the image spectrums in frequency domain. We stretch out the spectrum values by taking the logarithm of $|F(u,v)|$ using the MATLAB command

```
>> imshow ( mat2gray ( log ( 1+ abs ( FD image matrix ) ) ) )
```

```

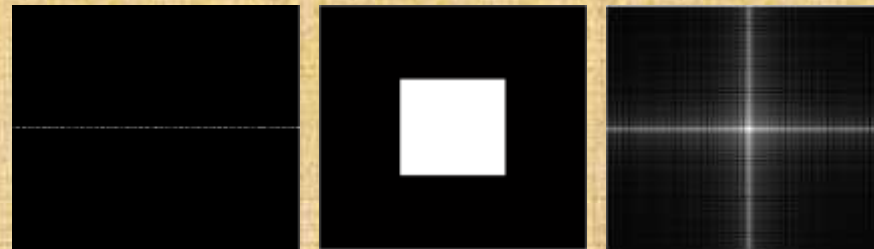
• Ex-3
>> a = [ zeros (256,128) , ones (256,128) ]; % a simple single edge image
>> af = fftshift( fft2 ( a ) );
>> imshow ( mat2gray ( log ( 1+ abs( af ) ) ) );

```

```

• Ex-4
>> c = zeros (256,256) % a single box image
>> c(78:178, 78:178) = 1;
>> cf = fftshift( fft2 ( c ) );
>> imshow (c);
>> imshow ( mat2gray ( log ( 1+ abs( cf ) ) ) );

```



af
Fig 4.5 frequency domain

c
spatial domain

cf
frequency domain

- **Ex-5** This example explains how to create a circle as it used as a frequency domain filter.

```

clc; close all; clear all

[ x, y] = meshgrid (-128 : 127 , -128 : 127);
z = sqrt ( x.^ 2 + y.^2);
c1 = (z < 5 );
c2 = (z < 25);
c3 = (z < 50);

subplot (2,3,1); imshow(mat2gray (c1)) ;title ('c1') ;
subplot (2,3,2); imshow(mat2gray (c2)) ;title ('c2') ;
subplot (2,3,3); imshow(mat2gray (c3)) ;title ('c3') ;

c1f = fftshift ( fft2 ( c1 ));
c2f = fftshift ( fft2 ( c2 ));
c3f = fftshift ( fft2 ( c3 ));

subplot (2,3,4);imshow(mat2gray (log(1+ abs(c1f)))) ;title ('c1f') ;
subplot (2,3,5);imshow(mat2gray (log(1+ abs(c2f)))) ;title ('c2f') ;
subplot (2,3,6);imshow(mat2gray (log(1+ abs(c3f)))) ;title ('c3f') ;

```

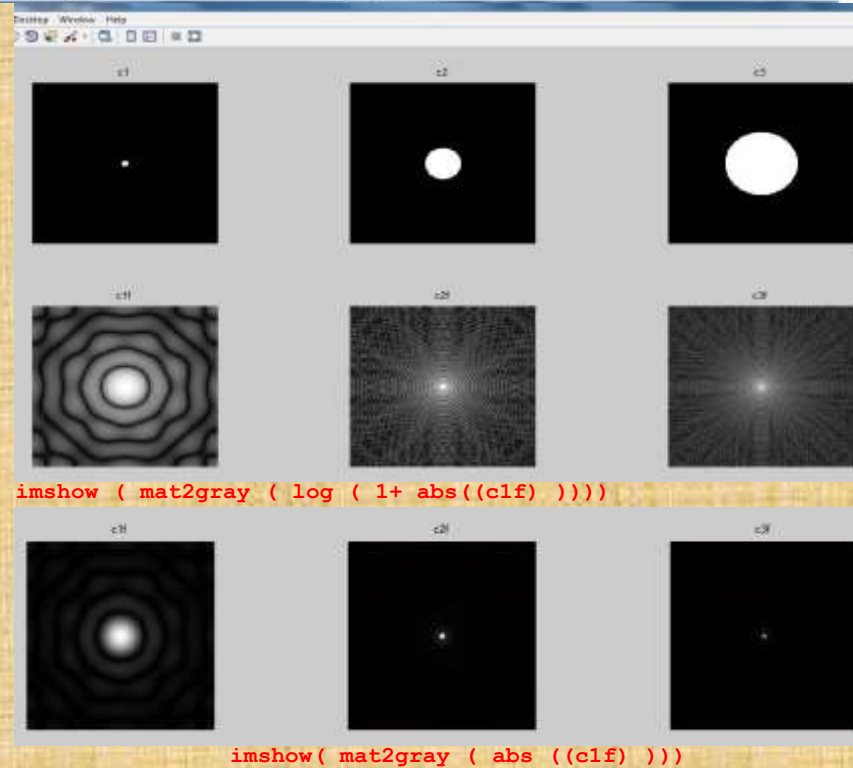


Fig. 4.6 The circle (LPF) and its transformation in the frequency domain

The `meshgrid(m , n)` of `[x,y]` will create two overlaid matrices (x and y) each one with two dimensions (`n , m`), for example:

<pre>>> [x,y] = meshgrid(1:3,1:5) Ans x = 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 y = 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5</pre>	<pre>>> [x,y] = meshgrid(2:6,1:3) ans x = 2 3 4 5 6 2 3 4 5 6 2 3 4 5 6 y = 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3</pre>
------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------

4.2 Filtering in the Frequency Domain

One of the reasons for the use of the Fourier transform in image processing is due to convolution theorem. A spatial convolution can be performed element wise multiplication of the Fourier transform by a suitable “filter matrix”

4.2.1 Ideal Low Pass Filtering

After shifting the DC coefficients of any transformed image toward the centre, all low frequency components will be moved to the centre as well. We can perform LP filtering by multiplying the transform by a matrix in such a way that centre values are maintained and values away from the centre are either removed or minimized. One way to do this is to multiply by an ideal *low-pass* matrix, which is a binary matrix, say m defined by

LP filter $\rightarrow m(x,y) = 1$ if (x,y) is closer to the centre than some value, $C_{cut\ off}$
 0 if (x,y) is further from the centre than $C_{cut\ off}$

The inverse FT of the element-wise product of any image says, I and m is,

$$F^{-1}(I . m)$$

The smallest circle radius, the sharpest $C_{cut\ off}$ frequency we obtained and the inversed image would be more smeared. See figure 4.7

4.2.2 Ideal High Pass Filtering

Just as we can perform LP filters by keeping the 2D DFT centre values and eliminating the others, so HP filters can be performed by the opposite: eliminating centre values and keep the others. This can be done with a minor modification of the preceding method of LP filtering. See figure 4.8 and 4.9

- Ex-6

```

clc; close all; clear all
[ x, y] = meshgrid (-128 : 127 , -128 : 127);
z = sqrt ( x.^ 2 + y.^2);
c1 = (z < 5); c2 = (z < 25); c3 = (z < 50);

subplot (2,3,1); imshow(mat2gray (c1) );title ('c1' ) ;
subplot (2,3,2); imshow(mat2gray (c2) );title ('c2' ) ;
subplot (2,3,3); imshow(mat2gray (c3) );title ('c3' ) ;

clf = fftshift ( fft2 ( c1 ));
c2f = fftshift ( ifft2 ( c2 ));
c3f = fftshift ( fft2 ( c3 ));

subplot (2,3,4);imshow(mat2gray (log(1+ abs(clf)))) ;title ('c1f' ) ;
subplot (2,3,5);imshow(mat2gray (log(1+ abs(c2f)))) ;title ('c2f' ) ;
subplot (2,3,6);imshow(mat2gray (log(1+ abs(c3f)))) ;title ('c3f' ) ;

```



(a) Cutoff of 5



(b) Cutoff of 30

Fig 4.7 Frequency domain LPF of cameraman image

As long as ideal filtering is required a dot product to be completed, it is very important to *match the size of filter with the size of transformed image*. The necessary MATLAB steps for size matching are:

```

I= imread ('kids.tif');
[r,c] = size (I);
[ x, y] = meshgrid (1:c , 1:r);
z = sqrt ( (x-c/2).^ 2 + (y-r/2).^2);
c = ( z > 15 )

```

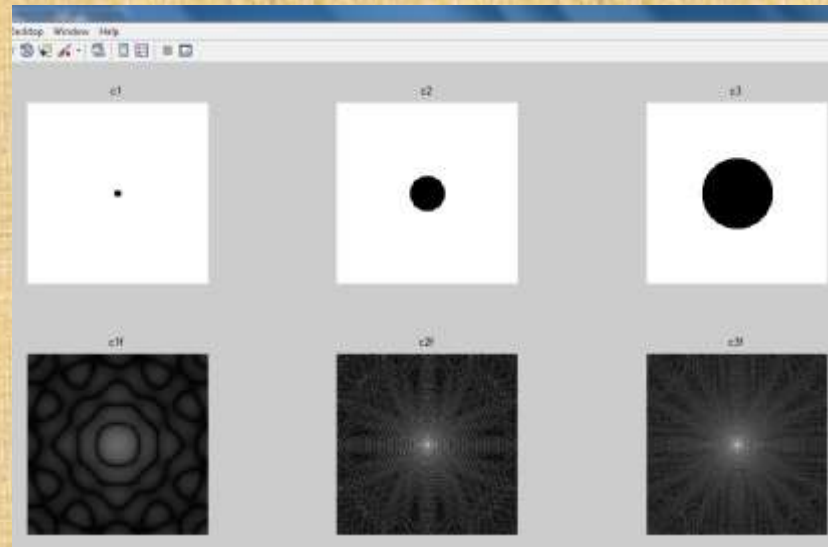


Fig. 4.8 Ideal HP filters with their transformations: $c1 > 5$, $c2 > 25$, $c3 > 50$

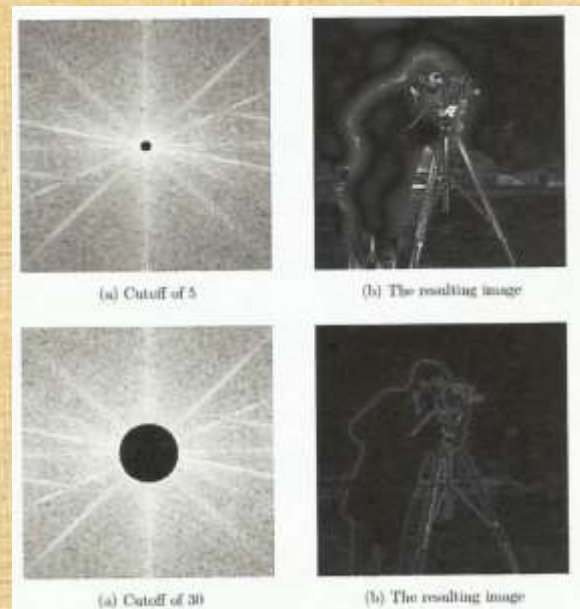


Fig. 4.9 ideal HP filtering for cameraman with cut off frequencies 5 and 30

Tutorial

The image matrix $A_{3 \times 3} = \begin{bmatrix} 5 & 0 & 7 \\ 1 & -1 & 0 \\ 2 & 3 & 4 \end{bmatrix}$ is required to be transformed to the frequency domain by applying the 2D DFT, calculate:

- a. The DC values
- b. The transformed image AF
- c. The shifted transform image AFS
- d. The mirror components
- e. Restore the original $A_{3 \times 3}$ matrix from AF
- f. Restore the original $A_{3 \times 3}$ matrix from AFS
- g. Make the DC value of ASF = 0 and recalculate the original $A_{3 \times 3}$.

Is there any difference in the $A_{3 \times 3}$ values in e, f, and g?

5

Chapter Five
Image Degradation & Restoration

salt & pepper noise

Gaussian & Speckle noise

periodic noise

blur

5.1 Image Degradation Model

Image restoration concerns about remove or reduce a degradation which have occurred during image acquisition. The reasons of degradation may include:

- noise: which are errors in the image pixel value
- blur: which is the distortion or smearing in the image because of camera motion or out of focus.

The degraded image could be successfully restored using two domains,

- neighbourhood operations, or
- frequency domain processes

let $f(x, y)$ is an image and $h(x, y)$ is a spatial filter, so

$$g(x, y) = f(x, y) * h(x, y)$$

is a convolution formula that has some form of degradation. A noise must be considered in this formula which can be modelled as an additive function to the convolution formula as follows,

$$g(x, y) = f(x, y) * h(x, y) + n(x, y)$$

in frequency domain becomes,

$$G(U, V) = F(U, V) . H(U, V) + N(U, V)$$

2. Types of Noise:

Noise is defined as any degradation in the image signal caused by external disturbance such as:

- Wireless transmission
- Satellite transmission
- Network cable transmission

Errors appear on the image output in different ways depending on the type of disturbance in the signal. If we know what type of errors in the image, we can choose the most appropriate method for reducing the effects. Cleaning an image corrupted by noise is an important area of image restoration.

5.2.1 Salt and Pepper Noise

Also, called impulse noise, shot noise, or binary noise. This degradation can be caused by sharp and sudden disturbances in the image signal. It is randomly scattered white or black or both pixels over the image.

Ex//

```
>> t = imread( 'twin.tif');
>> tg = rgb2gray (t);
>> tn= imnoise ( tg , 'salt&pepper', 0.2); % default noise is 10%
```

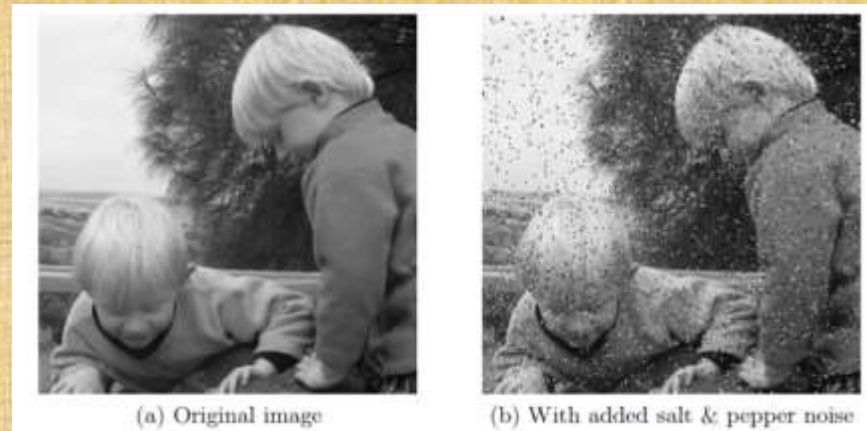


Fig. 5.1 Twin image is degraded by %20 of salt and pepper noise

5.2.2 Gaussian Noise

It is an idealized form of white noise, it is caused by random fluctuations in the signal, we can observe white noise by watching a T.V which is slightly mistuned to a particular channel. It is normally distributed and for example if I is an image matrix, N is a white or Gaussian noise, so

$$I^u = I + N \quad \text{where, } I^u \text{ is a noisy image}$$

The "Gaussian" parameter can also take optional values giving the mean and variance of the noise, the default values are mean ($= 0$) and standard deviation ($= 0.01$)

```
>> tgn = imnoise ( tg , 'gaussian' )
```


5.2.3 Speckle Noise

It is also called a multiplicative noise and it is a major problem in some radar applications. Like Gaussian noise, speckle can be modelled by random values multiplied by pixel values, $I(I + N)$. So,

$$I^u = I + IN = I(I + N)$$

N : consists of normally distributed values with mean 0 and its default value is 0.04.

```
>> tsp = imnoise (tg, 'speckle')
```

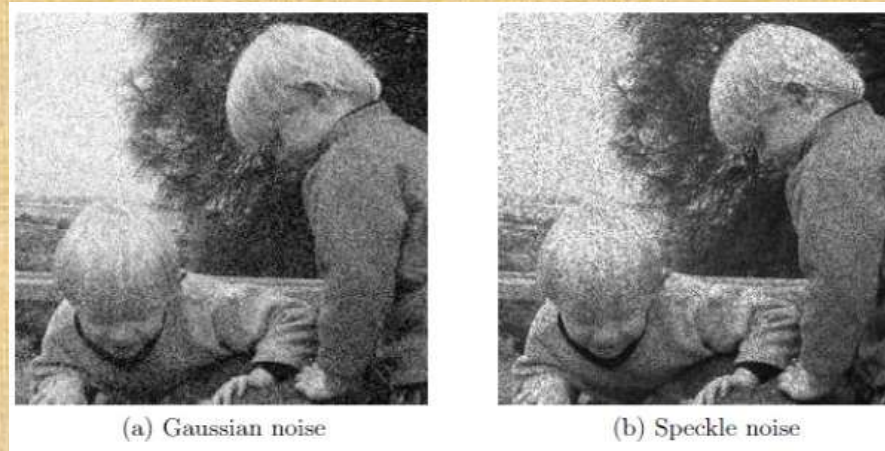


Fig. 5.2 Twins image is degraded by Gaussian and Speckle noise

5.2.4 Periodic Noise

This type of degradation has a global effect, and it is not easy to remove or decrease its effect using traditional cleaning methods. If image signal is subjected to a periodic rather than random disturbance, we might obtain a corrupted image by periodic noise. Periodic noise may occur in the image because of:

1. Image equipment
2. Network equipment
3. External disturbance of repeating nature like electric motor

The effect is look like bars over the image. All other noises like; salt & peppers or Gaussian noise could be removed using spatial filters while periodic noise requires the use of frequency domain technique to decrease the degradation effect or remove it. This is because the other forms of noise can be

modelled as local degradations while periodic noise has a global effect. We can simulate the worst effect of periodic noise using any periodic function like, exponent, cosine, or sine function:

```
>> s = size ( tg )
>> [ x , y ] = meshgrid ( 1 : s(1) , 1 : s(2) )
>> P = 1 + sine ( x/3 + y/5 )
>> tpk = ( im2double ( tg ) + P/2 ) / 2
```

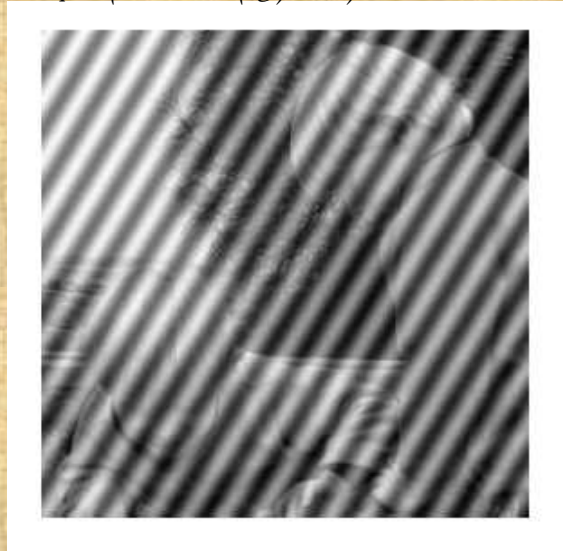


Fig 5.3 Twin image corrupted by periodic noise

5.3 Image Restoration: Noise Cleaning

5.3.1 Salt & pepper Noise Removal

The sudden change in the image pixel value for this kind of noise can be represented as a high frequency change and this high frequency can be blocked or filtered using **low pass filters**. The next sections will discuss many filtering techniques for cleaning different noisy image

- **Mean Filter:** as shown in the figure 5.4, the filtered image is not well cleaned as the noise is smeared over the image. The effect is even more pronounced if we use a large averaging filter.

```
>> fav = fspecial ('average')
>> tav = filter2 ( fav , tn )
```

- Median Filter:** a median filter is an example of a non-linear spatial filter. It is widely used as it is very effective at removing noise while preserving edges. The median is calculated by first sorting all the pixel values from the window into numerical order, and then replacing the pixel being considered with the middle (median) pixel value. If window elements are even, the middle is computed using the mean of the adjacent two pixels.

50	65	52	
63	255	58	50 52 57 58 60 61 63 65 255
61	60	57	

`> mdn = medfilt2 (tn)`



(a) 3 × 3 averaging



(b) 7 × 7 averaging



(a) Using medfilt2 twice

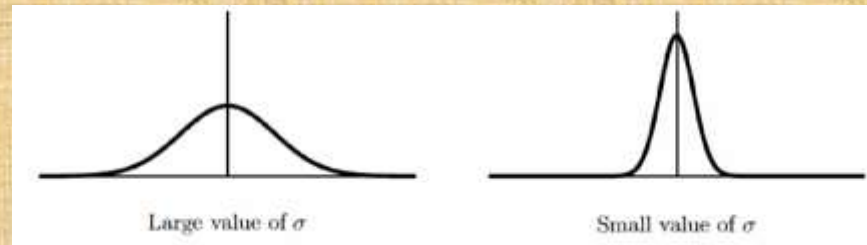


(b) using a 5 × 5 median filter

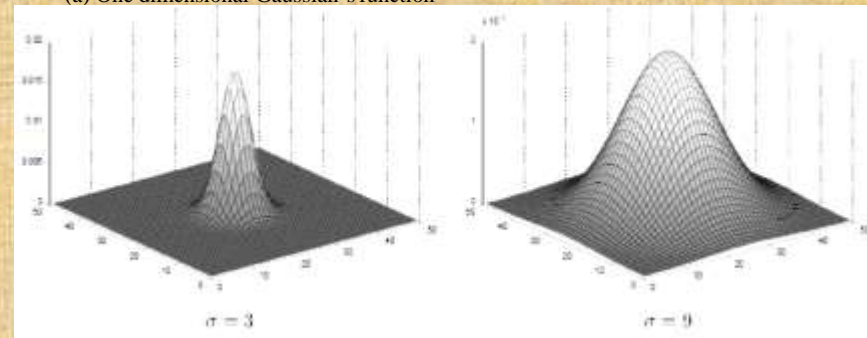
Fig 5.4 Noisy Twin image is cleaned using averaging and median filters

- **Gaussian Filter:** this filter is a class of low pass filters, and as LPFs does, it tend to smooth the image. It is based on the Gaussian probability distribution function, for two dimensions image,

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



(a) One dimensional Gaussian's function



(b) Two dimensional Gaussian's function

Fig. 5.5 (a) and (b)

Ex//

```
g = fspecial('gaussian',3,5)
```

```
g =      0.1118  0.1096
      0.109
```

6

```
      0.111  0.1141  0.1118
```

8

```
      0.109  0.1118  0.1096
```

6

```
g = fspecial('gaussian',3,3)
```

```
g = 0.1070  0.1131  0.1070
```

```
      0.1131  0.1196  0.1131
```

```
      0.1070  0.1131  0.1070
```

```
g = fspecial('gaussian',4,3)
```

```
g =      0.0623  0.0623  0.0558
```

```
      0.055
```

8

```
      0.0623  0.0623  0.0558  0.0623
```



Fig. 5.6 effects of different Gaussian filters on image

HW// The array $I_{8 \times 8}$ represents a small grayscale image. Compute the “valid” filtered images that result when the image is convolved with each of the mask (a) to (h) below.

(a)	-1 -1 0 -1 0 1 0 1 1	(b)	0 -1 -1 1 0 -1 1 1 0	(c)	-1 -1 -1 2 2 2 -1 -1 -1	(d)	-1 2 -1 -1 2 -1 -1 2 -1
(e)	-1 -1 -1 -1 8 -1 -1 -1 -1	(f)	1 1 1 1 1 1 1 1 1	(g)	-1 0 1 -1 0 1 -1 0 1	(h)	0 -1 0 -1 4 -1 0 -1 0

$$I = \begin{bmatrix} 20 & 20 & 20 & 10 & 10 & 10 & 10 & 10 \\ 20 & 20 & 20 & 10 & 10 & 10 & 10 & 20 \\ 20 & 20 & 20 & 10 & 10 & 10 & 10 & 20 \\ 20 & 20 & 10 & 10 & 10 & 10 & 20 & 10 \\ 20 & 10 & 10 & 10 & 10 & 10 & 20 & 10 \\ 10 & 10 & 10 & 20 & 10 & 10 & 20 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 20 & 10 & 20 & 20 & 10 & 10 & 20 & 20 \end{bmatrix}$$

Check your answers with MATLAB.

5.3.2 Gaussian Noise Removal

- **Mean of Multi-noisy Images:** With Gaussian noise instead of one corrupted image, we have many different Gaussian noise corruption images. If the satellite passed over the same spot many times, we will obtain many different images of the same place. Another example is in microscopy, we might take many different images of the same object.

Example:

Suppose we have 100 copies of $M + N_i$ Gaussian noisy image, so it's still true to express the formula,

$M + N_i$

where M is the matrix of original values, and N_i is a matrix of normally with mean 0. We can find the mean M' of these images by the usual ad

$$\begin{aligned} M' &= \frac{1}{100} \sum_{i=1}^{100} (M + N_i) \\ &= \frac{1}{100} \sum_{i=1}^{100} M + \frac{1}{100} \sum_{i=1}^{100} N_i \\ &= M + \frac{1}{100} \sum_{i=1}^{100} N_i \end{aligned}$$

Since N_i is normally distributed with mean 0, it can be readily shown that will be close to zero—the greater the number of N_i 's; the closer to zero.

$$M' \approx M$$

and the approximation is closer for larger number of images $M + N_i$.

% See figure 5.7

- **Averaging Filter:** The Gaussian noise has *mean* = 0, then we expect that mean filter would average noise to zero. The larger size of the filter mask, the closer to zero the value of the mean. Unfortunately mean filter tends to blur an image, however if we accept blurring against noise reduction. This method can be applied.

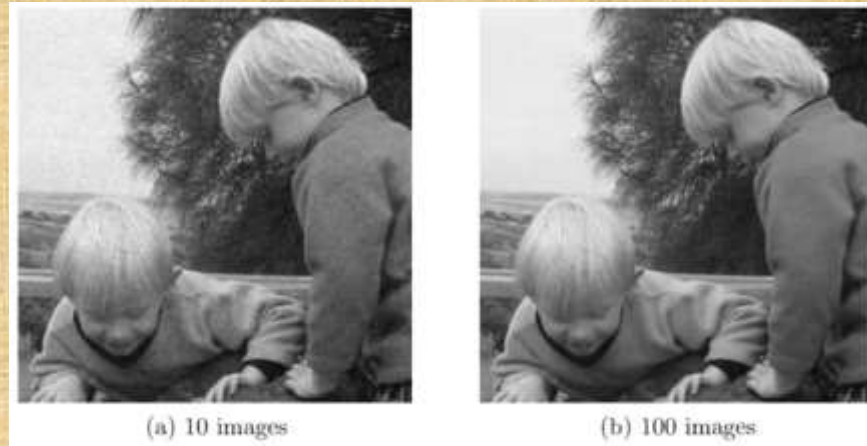


Fig. 5.7 Gaussian noise removal using the mean of multi noisy images

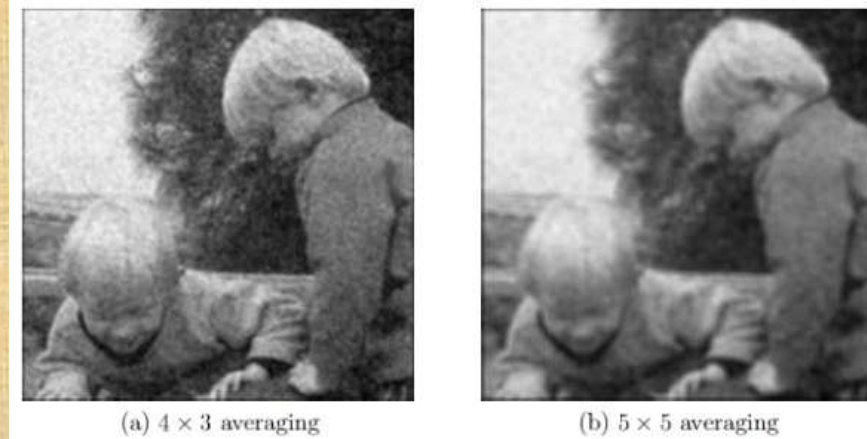


Fig. 5.8 Gaussian noise removal using mean filter

5.3.3 Periodic Noise Removal

- The Ring Filter

We have discussed how to create periodic noise bars and how to add them into an image using periodic function like *sine*. In this section we shall remove or decrease the degradation effect of this noise by designing frequency domain filters. The filtering in frequency domain like LPF and HPF are already discussed in section . Periodic noise in spatial domain would be appeared as two shiny spikes in frequency domain. The locations of those two spikes are completely related to (x and y) values in the periodic function, see figure 5.9. The next step is to design a Band Reject Filter, or commonly called Ring filter to reject the spikes. The next step is inversing the FD filtered image back it again into spatial domain. See the block diagram in Fig. 5.10.

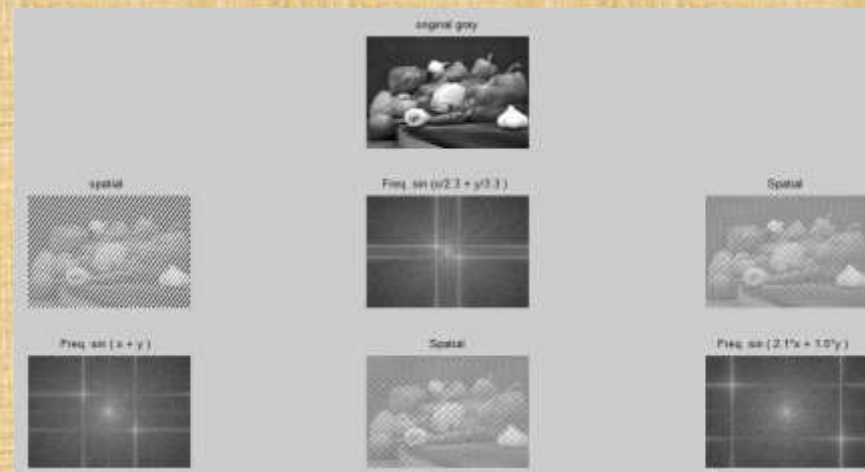


Fig. 5.9 different degradation images depending on sine ($x + y$) function

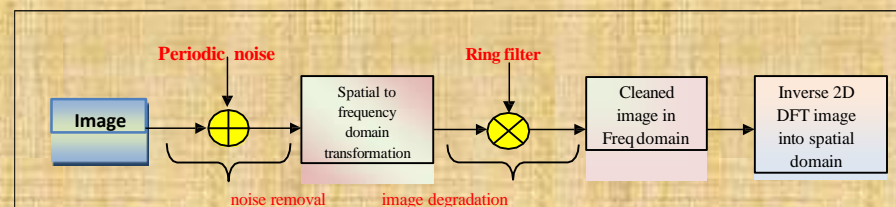


Fig. 5.10 image restoration/ periodic noise removal in frequency domain

Example: This example demonstrates periodic noise addition and removal using Band Reject Filter or commonly known a Ring Filter

```

clc; close all; clear all

I = imread ('C:\Program
            Files\MATLAB\R2008a\toolbox\images\imdemos\pears.png');
I = rgb2gray(I);

    [x,c] = size (I);
    [x, y] = meshgrid(1:c, 1:r);

    p1 = 1 + sin ( x + y );
    I2 = im2double(I) + p1 ;
    tgpf = fftshift ( fft2 (I2) );

subplot(2,3,1); imshow(mat2gray (I*1.2)) ;title ('original gray') ;
subplot(2,3,2); imshow((I2/2));title('noisy image in Spatial domain');
subplot(2,3,3);imshow (mat2gray( log ( abs(tgpf) ) ) ) ; title
                    ('noisy image in Freq. domain ' ) ;

    z = sqrt ( ( x - c/2 ).^2 + ( y - r/2 ).^2 );
    F= ( z < 135 | z > 145 );
    resf = tgpf .* F ;
    resi = ifft2 ( resf );

subplot(2,3,4);imshow (mat2gray( log (1+ abs(resf) ) ) ) ; title ('noisy
image X Ring filter ' ) ;
subplot (2,3,5);imshow (mat2gray ( log (1+ abs(resi) ) ) ) ; title
('F= ( z < 135 | z > 145 );') ;

    F2= ( z < 20 | z > 190 );
    resf2 = tgpf .* F2 ;
    resi2 = ifft2 ( resf2 );

subplot(2,3,6);imshow (mat2gray ( log (1+ abs(resi2) ) ) ) ; title
('F2= ( z < 20 | z > 190 ); ' ) ;

```

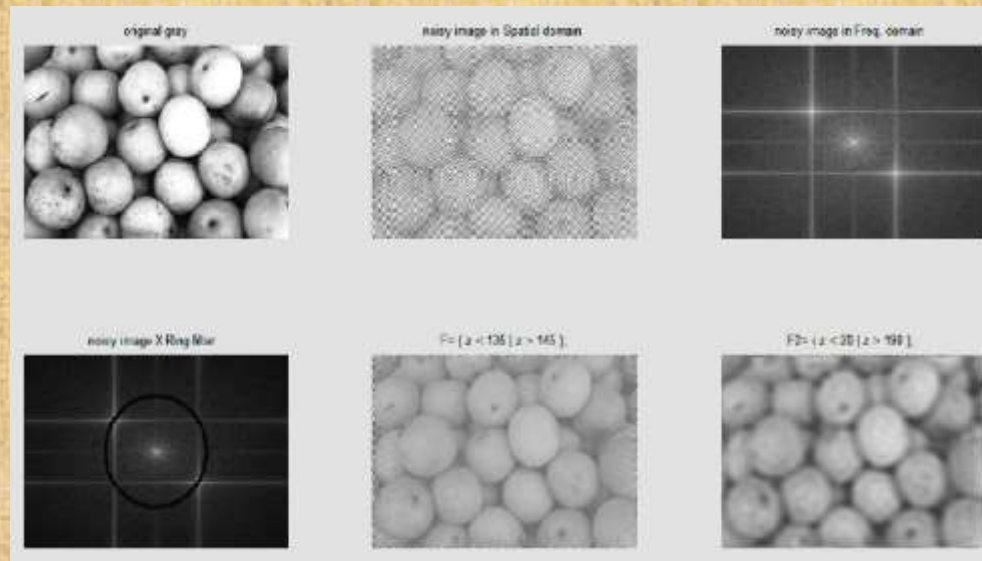



Fig. 5.11 Pears image is degraded by periodic noise and cleaned by Ring filter

- The Notch Filter
Another method can be used to clean the noisy image in the frequency domain is by setting the spike intersection lines to zero, i.e.; the interested row(s) and column(s).

HW//

Apply Notch filter method to clean the pears noisy image above and compare the result with the ring filter.

5.3.4 Winner Filter:

We know that the Gaussian noise (N) is normally distributed with mean ($\mu = 0$). Here, if we have an image X is filtered with a filter F and corrupted by an additive or Gaussian noise, the linearity of the Fourier transform is,

$$Y(i,j) = X(i,j) \cdot F(i,j) + N(i,j)$$

$$X(i,j) = [Y(i,j) - N(i,j)] / F(i,j)$$

The presence of noise can have a catastrophic effect on the inverse filtering where the noise can completely dominate the output and making the direct inverse impossible. Since we are dealing with the additive noise, the noisy image M' can be written as: $M' = M + N$ and if R represents the restored image, so our target is making R value close to M . In other words we want the result of this formula:

$$\sum (m_{i,j} - r_{i,j})^2$$

to be zero or close to zero. Filters which operates on this principle of Least Squares are called Winner filters, we can obtain x by

$$X(i,j) = \left[\frac{1}{|F(i,j)|^2 + k} |F(i,j)|^2 \right] Y(i,j)$$

Where k , is a constant and it is used to approximate the amount of noise. If the variance σ^2 of the noise is known, then $k = 2\sigma^2$ otherwise, k can be chosen by trail and error to yield to the best result. Note that when $k = 0$, then the equation will be reduced to

$$X(i,j) = \left[\frac{Y(i,j)}{F(i,j)} \right]$$

HW//

Write a MATLAB code to apply Winner formula on your Gaussian noisy image. Explain if the restored image is satisfied or not.

5.4 Motion Deblurring

Again the DFT of any filtered image could be expressed as:

$$Y(i, j) = X(i, j) \cdot F(i, j)$$

So, if we are given the $F(i, j)$ and $Y(i, j)$, we should be able to recover the DFT of the original image $X(i, j)$ by:

$$x(i, j) = Y(i, j) / F(i, j)$$

But as mentioned before the result, $x(i, j)$ may be very large values and eventually will be dominated because of some small elements of filter $F(i, j)$. There are two different methods to overcome this problem.

1st : Apply a low pass filter to the division

$$x(i, j) = \frac{Y(i, j)}{F(i, j)} L(i, j)$$

2nd : Using “division constrained” or threshold value d , so if the $|F(i, j)| < d$, we don't perform a division but just keep the original $Y(i, j)$ value, thus

$$x(i, j) = \frac{Y(i, j)}{F(i, j)} \quad \text{..... if } |F(i, j)| \geq d$$

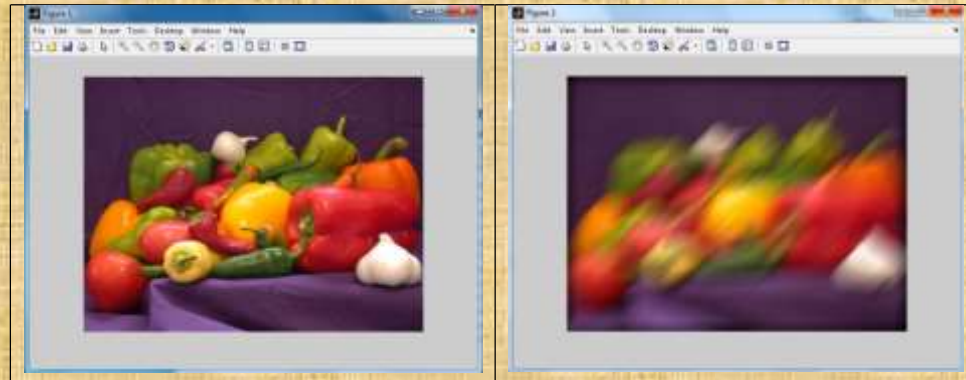
or,

$$x(i, j) = Y(i, j) \quad \text{..... if } |F(i, j)| < d$$

The removal of blur caused by motion would be a special case of filter inverse in frequency domain.

Ex:

```
originalRGB = imread('peppers.png');
h = fspecial('motion', 50, 45);
filteredRGB = imfilter(originalRGB, h);
figure, imshow(originalRGB), figure,
imshow(filteredRGB)
```

EX: The second method can be implemented in MATLAB code below

$$x(i,j) = \frac{Y(i,j)}{F(i,j)} \quad \text{..... if } |F(i,j)| \geq d$$

or,

$$x(i,j) = Y(i,j) \quad \text{..... if } |F(i,j)| < d$$

```

clc, close all, clear all;
n=imread('image.jpg');
ng=rgb2gray(n);
[r,c]=size(ng)

blur=fspecial('motion',15);
nb=imfilter(ng,blur);

fr=zeros(1:r,1:c);
fr(1,1:15)=blur;

d=0.12;
frf=fft2(fr);
frf(abs(frf)<d)=1;
yi=ifft2((fft2(nb))./frf);
imshow(uint8(yi));

```

HW//

Write MATLAB code to implement the two methods of de-blurring on your selected image. Use different threshold values(d) and compare the result of the two methods.

6

Chapter SIX

Image Features Detection

- Image Segmentation
- Image Thresholds
- Edge-detection
- Edge-detection, the 1st derivative
- Edge-detection, the 2nd derivative
- Horizontal Edges
- Vertical Edges
- Diagonal Edges
- Hough Transform

Image segmentation6.1

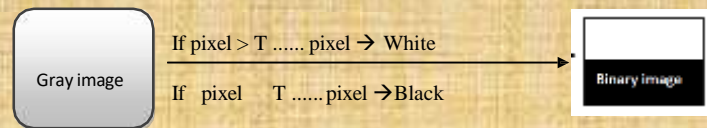
Segmentation process refers to the partitioning of image into component parts or separated objects. It can be implemented by two methods:

- Thresholding
- Edge detection

Thresholding is a vital part of image segmentation when we want to isolate object from its background and It is classified into three types,

1. Single threshold
2. Double threshold
3. Adaptive threshold

1. Single Threshold: this type uses single gray-level to detect or hide some information in the image. So an grey image can be converted to a binary one using single threshold value.



For white objects like rice, we apply the greater operator (>)

```
>> r = imread ('rice.tif');
>> imshow ( r ) , figure , imshow ( r > 100 );
```

While the less operator is used for detecting dark objects

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

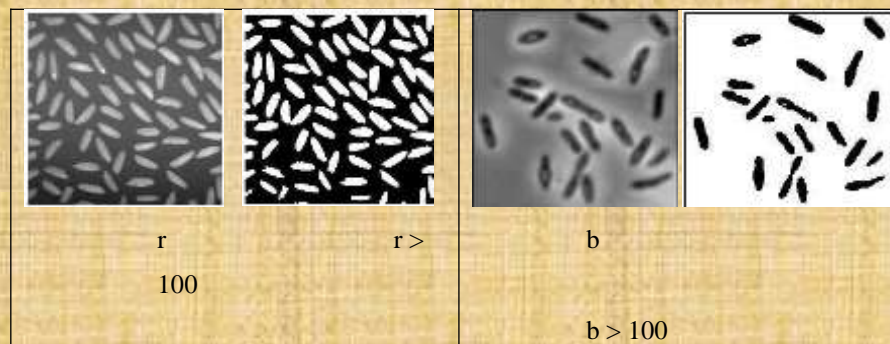


Fig. 6.2 rice and bacteria threshold

Also, the instruction: `im2bw (image matrix, threshold)` will create a binary image matrix according to threshold value.

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

2. Double Threshold

Sometimes we need more than single threshold value because of our interested objects have a range of grey-levels. So,

$T_1 < \text{pixels} < T_2 \rightarrow \text{White}$ otherwise $\text{pixels} \rightarrow \text{Black}$

```
>> [x, map] = imread (spine.tif);
>> s = uint8 ( 256 * ind2gray ( x, map ) )
>> imshow (s); figure;
>> imshow ( s > 115 & s < 125)
```

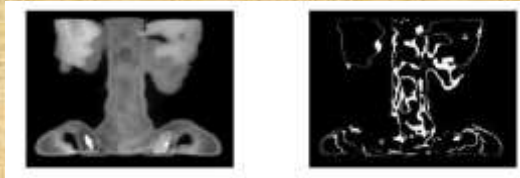


Fig. 6.3 Image needs double thresholds

3. Adaptive Threshold

This case is satisfied when objects and their background are varying in gray level.



Fig. 6.4 Image needs adaptive thresholding

Edge Detection 6.2

Edge is a local discontinuity in the pixel values which exceeds a given threshold. Also, it is defined as an image processing technique for finding the boundaries of objects within images. It works by **detecting discontinuities** in brightness. *Edge detection* is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

It includes a variety of mathematical methods that aim at identifying points in a [digital image](#) at which the [image brightness](#) changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed *edges*. The same problem of finding discontinuities in 1D signal is known as [step detection](#) and the problem of finding signal discontinuities over time is known as [change detection](#). *Edge detection* is a fundamental tool in [image processing](#), [machine vision](#) and [computer vision](#), particularly in the areas of [feature detection](#) and [feature extraction](#).

Example:

50	53	165	166
51	53	167	170
52	53	177	180
51	53	165	175

(a) vertical edge

50	53	52	53
51	53	51	53
165	166	177	180
167	170	165	175

(b) horizontal edge

Fig. 6.5 (a) and (b)

A commonly used approach to handle the problem of appropriate thresholds for thresholding is by using [thresholding](#) with [hysteresis](#). This method uses multiple thresholds to find edges. We begin by using the upper threshold to find the start of an edge. Once we have a start point, we then trace the path of the edge through the image pixel by pixel, marking an edge whenever we are above the lower threshold. We stop marking our edge only when the value falls below our lower threshold. This approach makes the assumption that edges are likely to be in continuous curves, and allows us to follow a faint section of an edge we have previously seen, without meaning that every noisy pixel in the image is marked down as an edge. Still, however, we have the problem of choosing appropriate thresholding parameters, and suitable thresholding values may vary over the image

We have computed a measure of edge strength (typically the gradient magnitude), the next stage is to apply a threshold, to decide whether edges are

present or not at an image point. The lower the threshold, the more edges will be detected, and the result will be increasingly susceptible to noise and detecting edges of irrelevant features in the image. Conversely a high threshold may miss subtle edges, or result in fragmented edges.

The general MATLAB command for image edge detection is,

```
>> edge ( image name, 'method', parameter )
```

6.3 Image Profile

The grey image profile in figure 6.6 can be plotted and the first derivative is applied to detect the discontinuity in the pixel values.

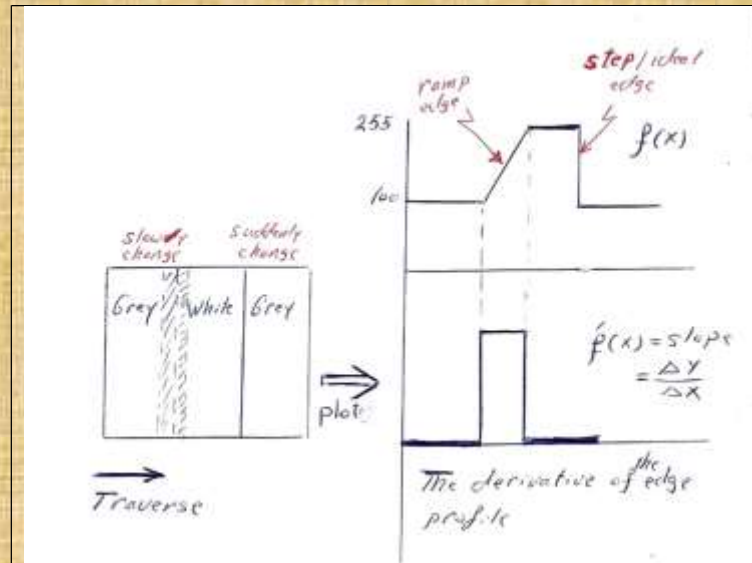


Fig. 6.6 image profile and its first derivative periods

From figure 6.6, the derivative returns zero for all constant sections of image profile and it returns non-zero in those parts of image in which different occurs. To apply continuous derivative on a discrete image we suppose that:

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \rightarrow 0} \frac{f(x) - f(x-h)}{h} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x-h)}{2h}$$

if $h=1$ we obtained $f(x+1) - f(x)$,

$$f(x) - f(x-1) \text{ and}$$

$$f(x+1) - f(x-1)$$

For two multidimensional image, we apply partial derivatives which is also called the *Gradient Vector Expression* for a function $f(x,y)$ points in the direction of its greatest increment. So, the direction of increment angle is:

$$\theta = \tan^{-1} \left(\frac{\partial f / \partial y}{\partial f / \partial x} \right)$$

and the magnitude of increment, M_g is:

$$M_g = \sqrt{\left(\frac{\partial f}{\partial y}\right)^2 + \left(\frac{\partial f}{\partial x}\right)^2}$$

Most edge detection methods are concerned with finding the magnitude of the gradient then applying a threshold to the result.

6.4 Prewitt, Roberts, and Sobel Filters

This type of filters can be used to find the horizontal and vertical edge(s) in the image. The horizontal mask $[-1 \ 0 \ 1]$ is used to detect the vertical edge, while the mask $[1 \ 0 \ -1]$ is used to find the horizontal edge. The edges in the result can be smoothed using the masks $[1 \ 1 \ 1]$ and $[1 \ 1 \ 1]$. The vertical and horizontal Prewitt edge detectors are.

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad P_y = P_{xT} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Ex: Consider the following 3×3 sub-image.

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}$$

```

x = [A] .* P_x = -a1 + a3 - a4 + a6 - a7 + a9
y = [A] .* P_y = -a1 - a2 - a3 + a7 - a8 + a9
diagonal edge = sqrt ( x2 + y2 ) ..... also it is more convenient to use either of,
diagonal edge = max { |x| , |y| } or ( |x| + |y| )

```

Example:

This example will highlight the vertical, horizontal, and diagonal edges in the integrated cct image.

```

% vertical edge
ic= imread(' ic.tif ');
px= [-1 0 1 ; -1 0 1 ; -1 0 1];
icx= filter2( px, ic);
imshow (icx /255);

% horizontal edge
py=px';
icy= filter2( py, ic);
figure, imshow (icy /255);

% diagonal edge
dig=sqrt (icx.^2 + icy.^2)
figure, imshow (dig /255);

or, using the command
>> dig = edge ( ic, 'prewitt' );
or, using binary image
bedge = im2bw ( dig / 255, 0.3 )

```

Other edge detection filters like Roberts and Sobel can be defined below:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad R_y = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Example:

Sobel filter and Roberts cross-gradient filter are slightly different edge finding filters, prove that by applying those filters on the same image source. Which one looks better? Try with different threshold values.

6.5 Edge detection – Second derivatives

The second derivative is another class of edge detection method. The Laplacian method,

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

is known as a discrete Laplacian and it is a 2nd derivative in

both directions. The mask $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ is a good example that could represent the 2nd derivative of Laplacian method. This class of filter is ideal for edge detection where, it is invariant under rotation (*isotropic filter*) that means

Laplacian + image rotation } same
image rotation + Laplacian } result

The disadvantage of all 2nd derivative is the noise sensitivity.

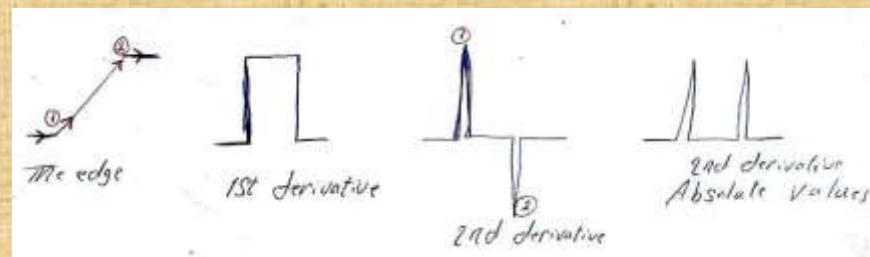


Fig 6.7 The second derivative edge

Ex:

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

The result will not be better than Prewitt or Sobel filters, so we can try with other Laplacian masks:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

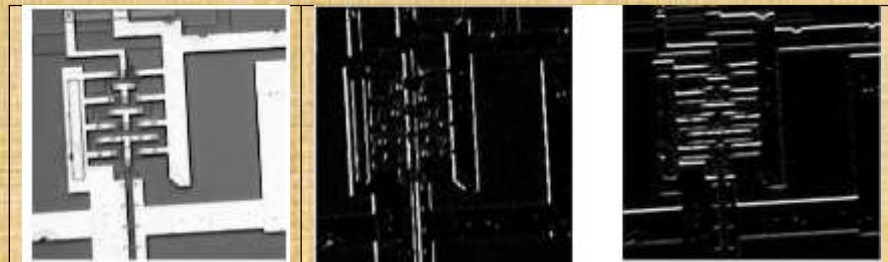


Fig 6.8 ic cct image and the vertical and horizontal edges using Prewitt filter

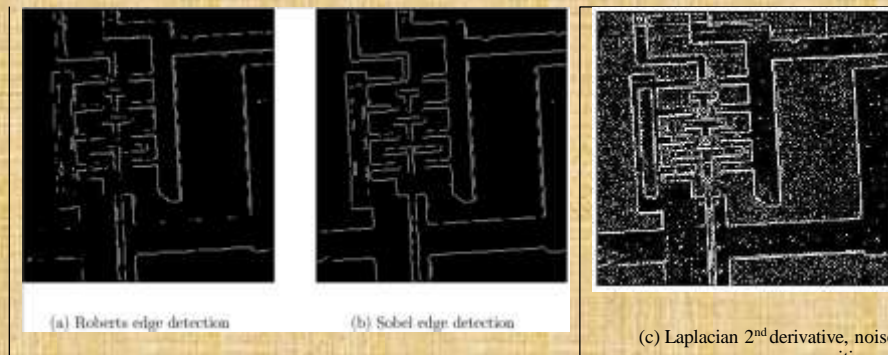


Fig 6.9 diagonal edges for Roberts, Sobel, and Laplacian filters

HW// Try to use Laplacian mask with “ALPHA” parameter. How is the result.

Zero - Crossing6.6

A more appropriate use for the Laplacian is to find the *position* of edges by locating *zero crossings*. If we look at figure 8.10, we see that the position of the edge is given by the place where the value of the filter takes on a zero value. In general, these are places where the result of the filter changes sign. For example, consider the the simple image given in figure 8.12(a), and the result after filtering with a Laplacian mask in figure 8.12(b).

30	50	30	30	50	50	50	50	50	50
30	30	30	50	30	50	50	50	50	50
30	50	200	200	200	200	200	200	50	50
30	50	200	200	200	200	200	200	30	50
30	50	200	200	200	200	200	200	30	50
30	50	200	200	200	200	200	200	30	50
30	50	200	200	200	200	200	200	30	50
30	50	30	50	200	200	200	200	30	50
30	50	30	50	200	200	200	200	30	50
30	50	30	50	30	50	50	50	50	50
30	30	30	30	30	50	50	50	50	50

(a) A simple image

-100	-30	-50	-30	-50	-30	-50	-30	-50	-100
-50	0	150	150	150	150	150	150	0	-50
-50	150	-300	-150	-150	-150	-150	-300	150	-50
-50	150	-150	0	0	0	0	-150	150	-50
-50	150	-150	0	0	0	0	-150	150	-50
-50	150	-300	-150	0	0	0	-150	150	-50
-50	0	150	300	-150	0	0	-150	150	-50
-50	0	0	150	-300	-150	-150	-300	150	-50
-30	0	0	0	150	150	150	150	0	-50
-100	-30	-50	-30	-50	-30	-50	-30	-50	-100

(b) After laplace filtering

We define the *zero crossings* in such a filtered image to be pixels which satisfy either of the following:

1. they have a negative grey value and are next to (by four-adjacency) a pixel whose grey value is positive,
2. they have a value of zero, and are between negative and positive valued pixels.

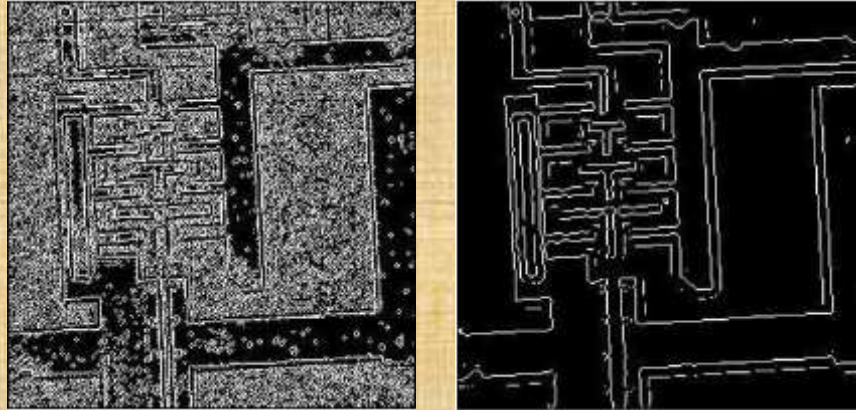
```
>> lp = fspecial ('laplace', 0);
>> icdig = edge (ic, 'zerocross', lp);
>> imshow (icdig);
```

The result is not good because of the interruption of many gray-level changes, so to eliminate this effect we smooth the image with a Gaussian filter so we have a new method for edge-detection. See figure 6.10

6.7 Marr-Hildreth Method

- Smooth the image with a Gaussian filter
- Convolve the result with a Laplacian
- Find the Zero-Crossing

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



Zero-crossing

using LoG filter first

6.8 Edge Linking: Hough Transform

Most of edge detection methods has disadvantage in linking the points along boundary or straight line. Hough transform is a way of finding boundary lines between the regions. It fits a line to those individual points, so it can be used to determine whether points lie on a curve or of a specific shape .

* Hough transform procedure

⑤

n . n . n . n . n . n . n . n . n . n .

Suppose (x, y) is a point in an image

$$\therefore y = ax + b$$

slope equation of a line

a, b are constants, x is variable
 y is a function of x

For $(x, y) = (1, 1)$

$$y = a \cdot 1 + b$$

The (a, b) array is the
"transform array"

$$\therefore b = -xa + y$$

x, y are constants, a is var
 b is a function of a

for $(x, y) = (1, 1) \Rightarrow b = -a + 1$

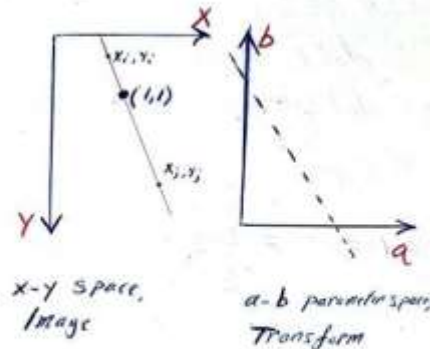
* Each point in the image is mapped onto
a line in the transform.

* each point (x_i, y_i) defines a line in the
 $a-b$ space (p (parameter space))

* points lying on the same line in the
 $x-y$ space, define lines in the parameter
space which all intersect at the same
point

* The coords of the point of intersection define the
parameters of the line in the $x-y$ space

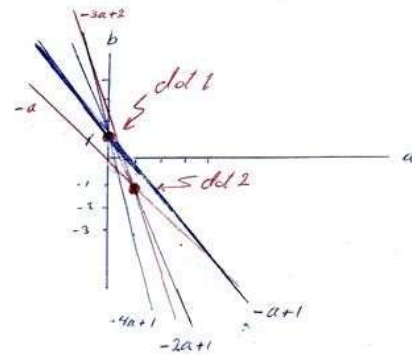
The points in the transform corresponding to the greatest number of
intersections correspond to the strongest line in the image.



Example Suppose we have an image with five points:
 $(1,0)$, $(1,1)$, $(2,1)$, $(4,1)$, and $(3,2)$

$$b = -a \cdot x + y$$

(x, y)	b
$(1, 0)$	$-a$
$(1, 1)$	$-a + 1$
$(2, 1)$	$-2a + 1$
$(4, 1)$	$-4a + 1$
$(3, 2)$	$-3a + 2$



Transform

The dots represent the maximum intersections of lines, each dot has three intersection lines.

dot 1 coordinates are $\begin{matrix} a \\ b \end{matrix} (0, 1)$

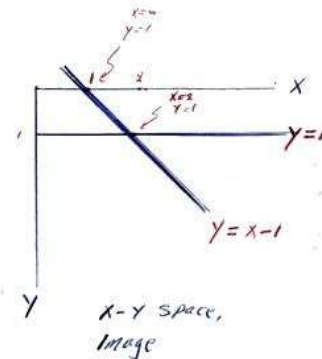
dot 2 coordinates are $(1, -1)$

The corresponding lines in $x-y$ space are:

$$\begin{matrix} a \\ b \end{matrix} (0, 1) \Rightarrow y = 0 \cdot x + 1 \Rightarrow y = 1$$

$$(1, -1) \Rightarrow y = 1 \cdot x - 1 \Rightarrow y = x - 1$$

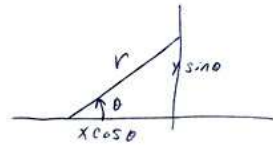
These lines are the "strongest" lines in the image in that they contain the greatest number of points



The Hough transform can't find vertical lines because the slope of vertical lines are infinite gradient and since it deals with line formula, $y = mx + c$,

$$m = \frac{\Delta y}{\Delta x \rightarrow 0} \rightarrow \infty$$

We can have another formula for the line, $r = x \cos \theta + y \sin \theta$



Now, we can implement Hough transform by: choosing a discrete set of values of r and θ , for each pixel (x, y) in the image, we compute $x \cos \theta + y \sin \theta$

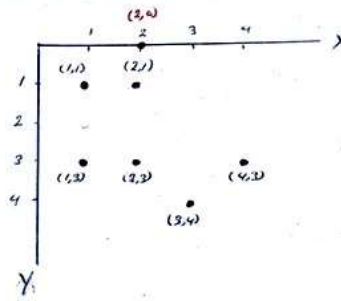
For each value of θ , we place the result in the appropriate position in the (r, θ) array. The highest values of (r, θ) in the array will correspond to strongest lines in the image

Ex Consider an image with the following pixels

We discretize θ for those values

$$-45^\circ, 0^\circ, 45^\circ, 90^\circ$$

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

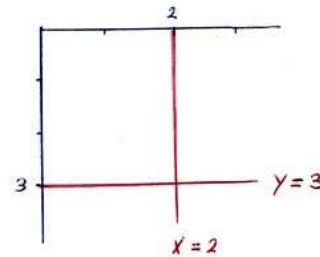


The accumulator array contains the No. of times each value of (r, θ) appears in the last table

$\theta \backslash r$	-1.4	-0.7	0	0.7	1	1.4	2	2.1	2.8	3	3.5	4	4.9
-45°	1	2	1	2		1							
0°					2		(3)			1		1	
45°						2		1	1		1		2
90°			1		2					(3)		1	

In practice this array will be very large, and can be displayed as an image. The two equal largest values occur at

$$\left. \begin{array}{l} (r, \theta) = (2, 0^\circ) \\ (r, \theta) = (3, 90^\circ) \end{array} \right\} \Rightarrow \text{Lines} \quad \begin{array}{l} x \cos 0 + y \sin 0 = 2 \text{ or } x = 2 \\ x \cos 90 + y \sin 90 = 3 \text{ or } y = 3 \end{array}$$



Lines found by
Hough Transform

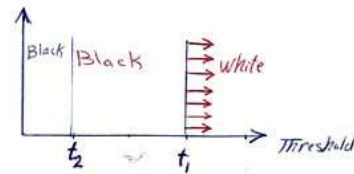
Tutorial and Solutions (7)

①

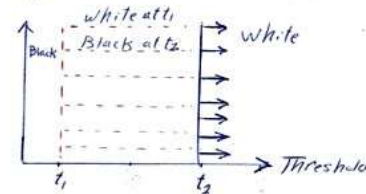
Thresholding

Q1

1. if $t_1 > t_2 \Rightarrow$ no affect of thresholding at t_2
2. if $t_2 > t_1 \Rightarrow$ Thresholding at t_2 will ignore thresholding at t_1



①



②

Q2 What threshold means? How many type of threshold?
Compare between threshold types, aided your answer by example.

Thresholding is the simplest method of image segmentation.
From a grayscale image, thresholding can be used to create binary image.

It converts each pixel into black, white or unchanged depending on whether the original color value is within the threshold range or not.

We have three kinds of thresholding:

① single ② double ③ adaptive

depending on the target object and its background grey level.

Notes Make a comparison table among threshold types

Q3. Suppose you have a (255 x 255) JPG image "XYZ.JPG". Write a MATLAB code to read each pixel in the image and make thresholding for image values (≤ 50 OR ≥ 150) to Zero.

```

X = imread('XYZ.JPG');
X = rgb2gray(X);
for i = 1 : 255
    for j = 1 : 255
        if (X[i,j] <= 50) || (X[i,j] >= 150)
            X[i,j] = 0;
        end;
    end;
end;
imshow(X)

```

Repeat the procedure above and make thresholding of the same image values to one. What is the difference between the two figures?

Q4. The prewitt filter is similar to the Sobel in that it uses two 3x3 kernels. The two kernels are convolved with the original image to calculate the approximations of the derivatives. If we define G_x and G_y as two images that contain the horizontal and vertical derivative approximation respectively,

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * A, \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * A$$

Where A is the original source image.

Convolve the prewitt kernels to the original image
with zero padding.

10	50	10	50	10
10	55	10	55	10
10	65	10	65	10
10	50	10	50	10
10	55	10	55	10

A

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

Kernel-x

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

Kernel-y

105	0	0	0	105
170	0	0	0	-170
170	0	0	0	-170
170	0	0	0	-170
105	0	0	0	-105

Gx

-65	-75	-120	-75	-65
-15	-15	-30	-15	-15
5	5	10	5	5
10	10	20	10	10
50	70	110	75	65

Gy

$$\therefore \text{pixel} = \sqrt{g_x^2 + g_y^2}$$

$$\text{i.e., } \sqrt{(105)^2 + (-65)^2} = 123.49 \approx 124$$

$$\sqrt{(0)^2 + (-75)^2} = 75$$

$$\sqrt{(0)^2 + (-120)^2} = 120$$

$$\sqrt{(0)^2 + (75)^2} = 75$$

$$\sqrt{(105)^2 + (-65)^2} = 123.49 \approx 124$$

and so on

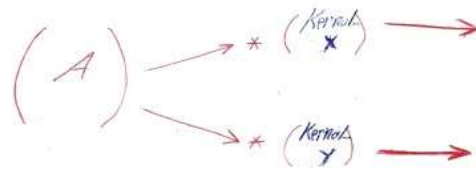
Compute the rest pixels values

Final image

Q5, The Sobel filter is used for edge detection, it works by calculating the gradient of image intensity at each pixel within the image. With

$$G_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} * A \quad , \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} * A$$

A is defined in Q4



135		0
-225		0
-235		0
-220		0
-160		0

G_x

-70	-120	-120	-120	-70

G_y

Complete the pixel values in G_x and G_y , Then Find the final image values. Compare the two final image (Q4 & Q5). What is your conclusion?

Q6, Exercises No. 7, 8, 9 p161-p162

7

Chapter Seven

Image Compression: The Huffman Coding

Chapter 7 Image Compression

- Codes and Compression
- Run-length encoding
- Huffman coding and Entropy
- Huffman coding algorithm

7.1 Codes and Compression:

CODING: Fewer bits to represent frequent symbols.

COMPRESSION: is the process of coding that will effectively reduce the total number of bits needed to represent certain information. The image compression objective is to reduce the amount of data required to represent an image. There are two main categories:

1. Lossless compression where all the information is retained and it is preferred for images of legal, scientific or political significance, where loss of data, even of apparent insignificance, could have considerable consequences. It is applied on computer program, medical images, GIS, and .gif files
2. Lossy compression where some information is lost, it is applied on TV signals, teleconference, .mp3, .jpg

Some applications require the compression scheme must be capable of reproducing the original data exactly; this is called '*lossless*'. For example, a binary executable (program) file might be compressed, but when expanded later, it had better be exactly reproduced.

Other applications do not require an exact reproduction of the original data; this is often called '*lossy*'. Images sometimes fall into this category, since they often will 'look' the same with considerable information discarded.



Fig. 7.1 A General Data Compression Scheme.

The compression ratio = B_0/B_1

Where: B_0 : number of bits before compression,

B_1 : number of bits after compression

Characters in file	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
a fixed length	000	001	010	011	100	101
a variable length	0	101	100	111	1101	1100

The fixed length-code requires 300 bits to store the file.
 $(45*3 + 13*3 + 12*3 + 16*3 + 9*3 + 5*3) = 300 = 3 * \sum (45 + 13 + \dots + 5)$

The variable-length code uses only
 $(45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4) = 224$

Compression ratio = $300/224 = 1.339$

7.4 Huffman Coding

The basic idea in Huffman coding is simple. Rather than using a fixed length code (8 bits) to represent the grey values in an image, we use a variable length code, with smaller length codes corresponding to more probable grey values. A Huffman encoder takes a block of input characters (grey values) with *fixed* length and produces a block of output bits of *variable* length. It is a fixed-to-variable length code. The algorithm assigns short codewords to those input blocks with high probabilities and long codewords to those with low probabilities.

Example 3:

A small example will make this clear. Suppose we have a 2-bit greyscale image with only four grey levels: 0, 1, 2, 3, with the probabilities 0.2, 0.4, 0.3 and 0.1 respectively. The following table shows fixed length and variable length codes for this image:

Grey value	Probability	Fixed code	Variable code
0	0.2	00	000
1	0.4	01	1
2	0.3	10	01
3	0.1	11	001

Now consider how this image has been compressed. Each grey value has its own unique identifying code. The average number of bits per pixel can be easily calculated as the expected value (in a probabilistic sense):

$$L_{av} = (0.2 * 3) + (0.4 * 1) + (0.3 * 2) + (0.1 * 3) = 1.9$$

Notice that the longest codewords are associated with the lowest probabilities. This average is indeed smaller than 2. This can be made more precise by the notion of entropy, which is a measure of the amount of information

7.5 Entropy Concept

The entropy H of an image is the theoretical minimum number of bits per pixel required to encode the image with no loss of information. It is defined by,

$$H = - \sum_{i=0}^{L-1} p_i \log_2 (p_i)$$

$$H = - (0.2 \log_2(0.2) + 0.4 \log_2(0.4) + 0.3 \log_2 (0.3) + 0.1 \log_2(0.1)) \\ = 1.8464$$

Where the index i is taken over all greyscales of the image, and p_i is the probability of grey level i occurring in the image. This means that no matter what coding scheme is used, it will never use less than 1.8464 bits per pixel. On this basis, the Huffman coding scheme given above, giving an average number of bits per pixel much closer to this theoretical minimum than 2, provides a very good result.

7.6 Huffman Algorithm

1. Determine the probabilities of each grey value in the image.
2. Form a binary tree by adding probabilities two at a time, always taking the two lowest available values.
3. Now assign 0 and 1 arbitrarily to each branch of the tree from its top.
4. Read the codes from the top down.

Example 4:

To see how this works, consider the example of a 3-bit greyscale image (so the grey values are 0_7) with the following probabilities:

Grey value	0	1	2	3	4	5	6	7
probability	0.19	0.25	0.21	0.16	0.08	0.06	0.03	0.02

For these probabilities, the entropy can be calculated as follows,

$$H = - (0.19 \log_2(0.19) + 0.25 \log_2(0.25) + 0.21 \log_2 (0.21) + 0.16 \log_2(0.16) \\ + 0.08 \log_2(0.08) + 0.06 \log_2(0.06) + 0.03 \log_2(0.03) + 0.02 \log_2(0.02)) \\ H = 2.6508$$

$$\text{The fixed length-code} = (0.19 + 0.25 + 0.21 + 0.16 + 0.08 + 0.06 + 0.03 + 0.02) * 3 \\ = 3$$

it's better to re-index them by (ascending or descending)

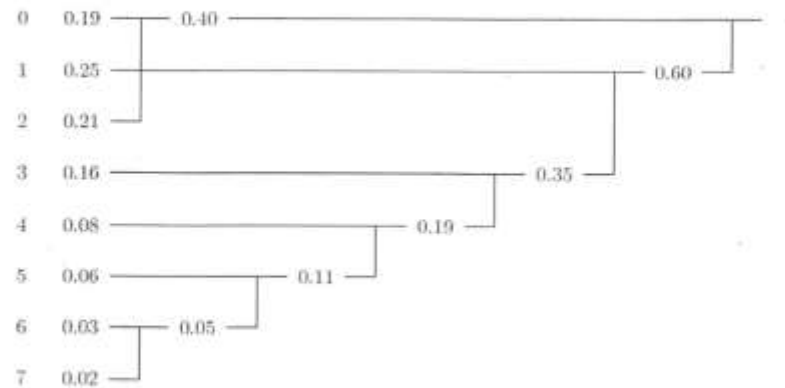


Figure 7.2 Forming the Huffman code tree

We can now combine probabilities two at a time as shown in figure 7.2. Note that if we have a choice of probabilities we choose arbitrarily. The second stage consists of arbitrarily assigning 0's and 1's to each branch of the tree just obtained. This is shown in figure 7.3.

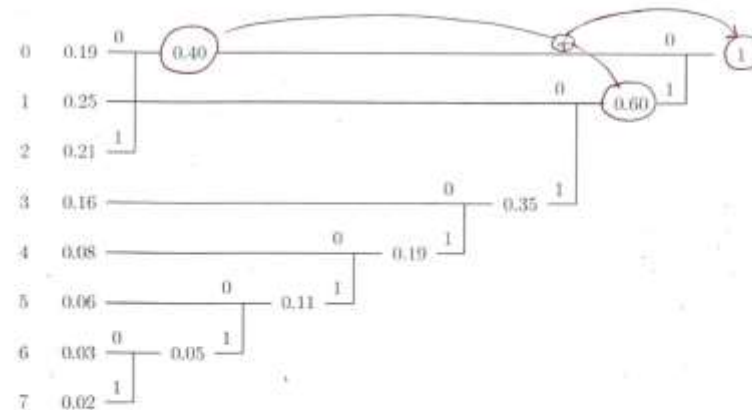


Figure 7.3 Assigning 1's and 0's to the branches

To obtain the codes for each grey value, start at the 1 on the top right, and work back towards the grey value in question, listing the numbers passed on the way. This produces:

Grey value	Huffman code
0	00
1	10
2	01
3	110
4	1110
5	11110
6	111110
7	111111

2 digit codes (for values 0, 1, 2)
3 (for value 3)
4 (for value 4)
5 (for value 5)
6 digit codes (for values 6, 7)

$$L_{avg} = (0.19 \cdot 2) + (0.25 \cdot 2) + (0.21 \cdot 2) + (0.16 \cdot 3) + (0.08 \cdot 4) + (0.06 \cdot 5) + (0.03 \cdot 6) + (0.02 \cdot 6) = 2.7$$

The compression ratio = $3/2.7 = 1.111$

which is a significant improvement over 3 bits per pixel, and very close to the theoretical minimum of 2.6508 given by the entropy. Huffman codes are uniquely decodable, in that a string can be decoded in only one way. For example, consider the string to be decoded with the Huffman code generated above.

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

There is no code word 1, or 11, so we may take the first three bits 110 as being the code for grey value 3. Notice also that no other code word begins with this string. For the next few bits, 1110 is a code word; no other begins with this string, and no other smaller string is a codeword. So we can decode this string as grey level 4. Continuing in this way we obtain:

$\underbrace{1\ 1\ 0}_3\ \underbrace{1\ 1\ 1\ 0}_4\ \underbrace{0\ 0}_0\ \underbrace{0\ 0}_0\ \underbrace{1\ 0}_1\ \underbrace{0\ 1}_2\ \underbrace{1\ 1\ 1\ 1\ 0}_5$

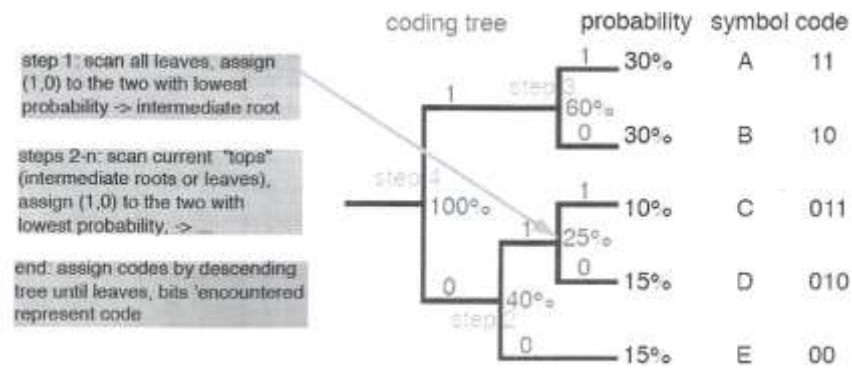
Example 5: $a_1 = 0.1, a_2 = 0.4, a_3 = 0.06, a_4 = 0.1, a_5 = 0.04, a_6 = 0.3$

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	
a_1	0.1	0.1	0.2	0.3	0.4
a_4	0.1	0.1	0.1	0.1	
a_3	0.06	0.1			
a_5	0.04				

Original source			Source reduction			
Symbol	Probability	Code	1	2	3	4
a_1	0.4	1	0.4 1	0.4 1	0.4 1	0.6 0
a_2	0.3	00	0.3 00	0.3 00	0.3 00	0.4 1
a_3	0.1	011	0.1 011	0.2 010	0.3 01	
a_4	0.1	0100	0.1 0100	0.1 01		
a_5	0.06	01010	0.1 0101			
a_6	0.04	01011				

Example 6

- Characters to be encoded: A, B, C, D, E
- probability to occur: $p(A)=0.3$, $p(B)=0.3$, $p(C)=0.1$, $p(D)=0.15$, $p(E)=0.15$



Tutorial No. 7

1. Construct a Huffman code for each of the probability tables given:

grey scale		0	1	2	3	4	5	6	7
probability	(a)	.07	.11	.08	.04	.5	.05	.06	.09
	(b)	.13	.12	.13	.13	.12	.12	.12	.13
	(c)	.09	.13	.15	.1	.14	.12	.11	.16

In each case determine the average bits/pixel given by your code.

2. From your results of the previous question, what do think are the conditions of the probability distribution which give rise to a high compression rate using Huffman coding?
3. (a) Given the following 4-bit image:
- | | | | | | | | |
|---|---|---|---|---|----|----|----|
| 0 | 4 | 4 | 4 | 4 | 4 | 6 | 7 |
| 0 | 4 | 5 | 5 | 5 | 4 | 6 | 7 |
| 1 | 4 | 5 | 5 | 5 | 4 | 6 | 7 |
| 1 | 4 | 5 | 5 | 5 | 4 | 6 | 7 |
| 1 | 4 | 4 | 4 | 4 | 4 | 6 | 7 |
| 2 | 2 | 8 | 8 | 8 | 10 | 10 | 11 |
| 2 | 2 | 9 | 9 | 9 | 12 | 13 | 13 |
| 3 | 3 | 9 | 9 | 9 | 15 | 14 | 14 |
- transform it to a 3-bit image by removing the least most significant bit plane. Construct a Huffman code on the result and determine the average number of bits/pixel used by the code.
- (b) Now apply Huffman coding to the original image and determine the average number of bits/pixel used by the code.
- (c) Which of the two codes gives the best rate of compression?

8

Chapter Eight Image Compression: Discrete Cosine Transform

Image Compression

1. The JPEG Standard:

The name “JPEG” stands for Joint Photographic Experts Group, the name of the committee that created the JPEG standard. JPEG/Exif is also the most common format saved by digital cameras. It was developed in 1974 by N. Ahmed and T. Natarajan. The JPEG is not suited for line drawings and other textual or iconic graphics, where the sharp contrast between adjacent pixels can cause noticeable artifacts. Such image is better to be saved in a lossless graphics format such as: TIFF, GIF, PNG or a raw image format. JPEG compression should not be used in scenarios where the exact reproduction of the data is required, such as some scientific and medical imaging applications and certain technical image processing work.

2. JPEG Codec Procedure

Although a JPEG file can be encoded in various ways, the encoding process consists several steps:

1. Color Space Transformation:

The representation of the colours in the image is converted from RGB to $Y' C_B C_R$ (or informally, $Y C_b C_r$) where Y' is the luma component which represents the brightness. C_B and C_R are the chrominance components for Blue and Red color.

2. Down Sampling:

Due to the densities of color and brightness sensitive receptors in the human eye, humans can see considerably more fine detail in the brightness of an image (Y component) than the Hue and color Saturation (C_b and C_r components) of an image. The spatial resolution of the chrominance data is reduced, usually by factor of 2 (called “down-sampling” or “chroma sub-sampling”). The ratios at which the down sampling is ordinary done for JPEG images are 4:4:4 (no sampling), 4:2:2 (reduction by factor of 2 in the horizontal direction), or most commonly 4:2:0 (reduction by factor of 2 in the used This reflect the fact that the eye is less sensitive to the color details than to fine brightness details. This step is some time skipped

3. Block Splitting:

After down sampling each channel must be split into of 8×8 blocks. Depending on chroma sub-sampling this yields MCU (Minimum Coded Unit) of size 8×8 blocks (4:4:4 no sampling), 16×8 (4:2:2), or most commonly 16×16 (4:2:0). If the data for a channel does not represent an integer number of blocks then the encoder must fill the

remaining area with some form of dummy data. Filling the edges with a fixed color, for example black can create ringing artifacts along the visible part of the boarder, while repeating the edge pixels is common technique that reduces such artifacts.

4. Discrete Cosine Transform :

Next, each block of (Y, C_b, C_r) component is converted to a frequency domain representation using a normalized two-dimensional DCT. Similar to a DFT, DCT produces a kind of spatial frequency domain spectrum.

5. Quantization:

The amplitudes of the frequency components are quantized. Human vision is much more sensitive to small variations in colors or brightness over large areas than to the strength of high-frequency brightness variations. Therefore, the magnitudes of the high frequency components are stored with a lower accuracy than the low-frequency components. The 8×8 quantization matrix is:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

6. The resulting data for all 8×8 blocks is further compressed with lossless algorithm, a variant of Huffman encoding.
7. The decoding process reverses these steps, except the quantization because it is irreversible

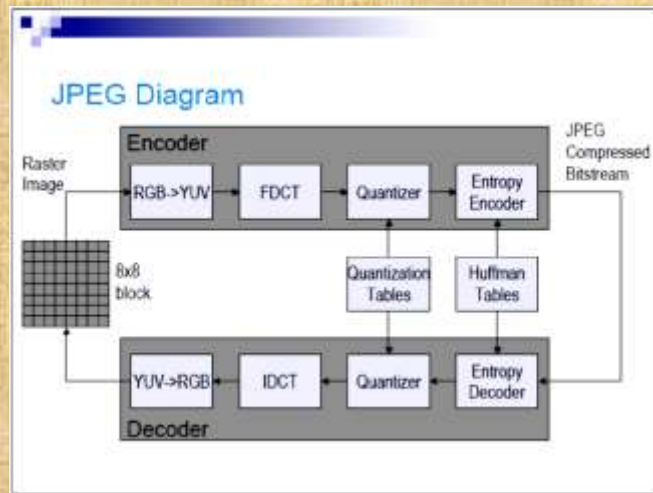


Fig. 8.1 Image compression/decompression block diagram

$$G(i, j) = c_i c_j \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(y, x) \cos \frac{(2y+1)i\pi}{2N} \cos \frac{(2x+1)j\pi}{2N}$$

$$c_i = \sqrt{1/N} \text{ if } i = 0, \quad c_i = \sqrt{2/N} \text{ otherwise. Similarly } c_j$$

Example: An example of 8×8 8-bit sub-image is,

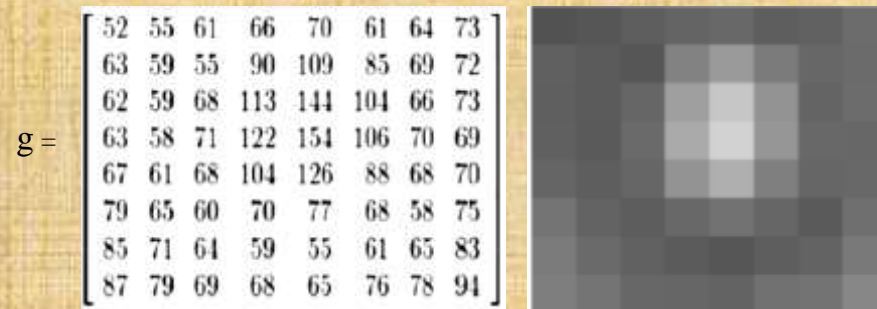


Fig. 8.2 The (8×8) sub-image picture and it matrix values as 8-bit grayscale intensities

- Before, computing the DCT of the 8×8 block, the values are shifted from a positive range to one centred around zero. For an 8bit image, each pixel value falls in the range $[0-255]$. The mid value, the 128 is subtracted from each entry to produce a data range centred around zero, so that the new entry range is $[-128,127]$.

$$g(x,y) = g(x,y) - 128$$

- This step reduces the dynamic range requirement in the DCT processing stage that follows. Mathematically, it equivalents to subtracting 1024 from the DC coefficient after performing the transform, which may be better way to perform the operation on some architectures since it involves performing only one subtraction rather than 64 of them. The step results in the following values

(a)

$$g = \begin{matrix} & \begin{matrix} \xrightarrow{x} \\ \rightarrow \end{matrix} \\ \begin{matrix} \left[\begin{array}{cccccccc} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{array} \right] \end{matrix} \\ \begin{matrix} \downarrow \\ y \end{matrix} \end{matrix}$$

(b)

$$G = \begin{matrix} & \begin{matrix} \xrightarrow{u} \\ \rightarrow \end{matrix} \\ \begin{matrix} \left[\begin{array}{cccccccc} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.80 & 1.68 \end{array} \right] \end{matrix} \\ \begin{matrix} \downarrow \\ v \end{matrix} \end{matrix}$$

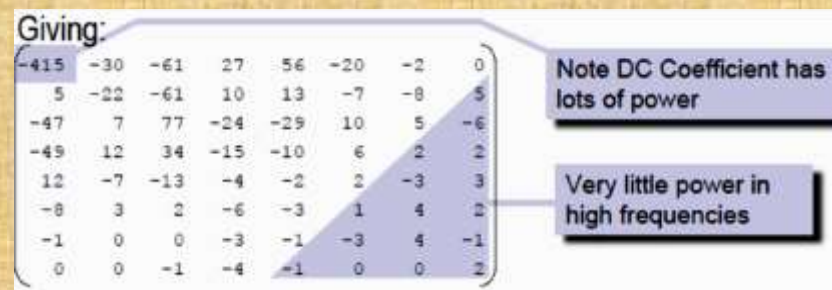


Fig. 8.3 (a) g matrix (b) transformed G matrix (c) rounded G matrix

JPEG Decoding





- Decoding is simply the reverse of encoding.
- Reverse the Huffman, RLE encodings.
- Dequantize.
- Apply inverse DCT (IDCT):

$$V(x, y) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} c_i c_j T(i, j) \cos \frac{(2i+1)\pi}{2N} \cos \frac{(2j+1)\pi}{2N}$$

- Add 128 to convert back to unsigned.

JPEG Compression ratio

- Compression ratio depends on how large the values in the quantization matrix are.
- 10:1 achievable without noticeable loss.
- 100:1 achievable, but artifacts are noticeable.

Image	Lossless compression	Lossy compression
Original		
Processed by Canny edge detector		

Mathematical morphology (1)

9.1 Introduction

Morphology, or *morphology* for short, is a branch of image processing which is particularly useful for analyzing shapes in images. We shall develop basic morphological tools for investigation of binary images, and then show how to extend these tools to greyscale images. MATLAB has many tools for binary morphology in the image processing toolbox; most of which can be used for greyscale morphology as well.

9.2 Basic ideas

The theory of mathematical morphology can be developed in many different ways. We shall adopt one standard method which uses operations on sets of points. A very solid and detailed account can be found in Haralick and Shapiro [5].

Translation

Suppose that A is a set of pixels in a binary image, and $w = (x, y)$ is a particular coordinate point. Then A_w is the set A “translated” in direction (x, y) . That is

$$A_w = \{(a, b) + (x, y) : (a, b) \in A\}.$$

For example, in figure 9.1, A is the cross shaped set, and $w = (2, 2)$. The set A has been shifted in the x and y directions by the values given in w . Note that here we are using matrix coordinates, rather than Cartesian coordinates, so that the origin is at the top left, x goes down and y goes across.

Reflection

If A is set of pixels, then its *reflection*, denoted \hat{A} , is obtained by reflecting A in the origin:

$$\hat{A} = \{(-x, -y) : (x, y) \in A\}.$$

For examples, in figure 9.2, the open and closed circles form sets which are reflections of each other.

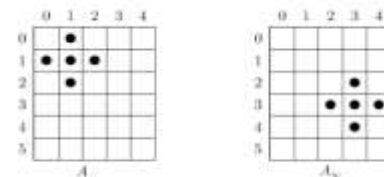


Figure 9.1. Translation

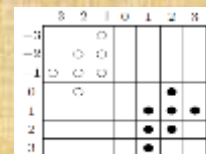


Figure 9.2. Reflection

9.3 Dilation and erosion

These are the basic operations of morphology, in the sense that all other operations are built from a combination of these two.

9.3.1 Dilation

Suppose A and B are sets of pixels. Then the *dilation* of A by B , denoted $A \oplus B$, is defined as

$$A \oplus B = \bigcup_{x \in B} A_x.$$

What this means is that for every point $x \in B$, we translate A by those coordinates. Then we take the union of all these translations.

An equivalent definition is that

$$A \oplus B = \{(x, y) + (u, v) : (x, y) \in A, (u, v) \in B\}.$$

From this last definition, dilation is shown to be commutative; that

$$A \oplus B = B \oplus A.$$

Example:

An example of a dilation is given in figure 9.3. In the translation diagrams, the grey squares show the original position of the object. Note that $A_{(0,0)}$ is of course just A itself. In this example, we have

$$B = \{(0,0), (1,1), (-1,1), (1,-1), (-1,-1)\}$$

and these these are the coordinates by which we translate A .

In general, $A \oplus B$ can be obtained by replacing every point (x, y) in A with a copy of B ; placing the $(0,0)$ point of B at (x, y) . Equivalently, we can replace every point (u, v) of B with a copy of A .

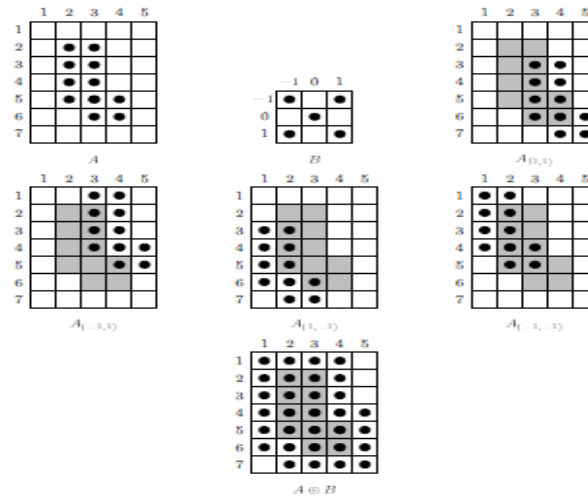


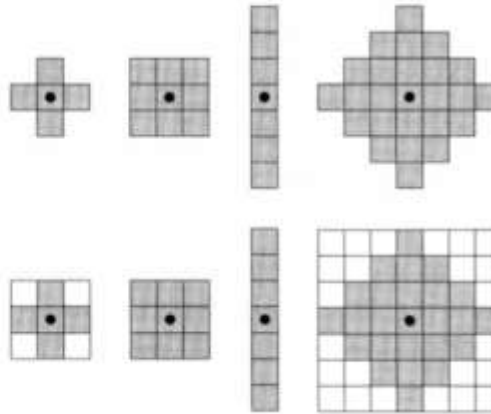
Figure 9.3: Dilation

Structuring Elements

For dilation, we generally assume that A is the image being processed, and B is a small set of pixels. In this case B is referred to as a *structuring element* or as a *kernel*.

Reflection and translation are used extensively to formulate operations based on so-called *structuring elements* (SEs): small sets of subimages used to probe an analyzed image for properties of interest.

Examples of structuring elements: shaded square denotes a member of the SE



The origins of SEs are marked by a black dot.

When working with images, SEs should be rectangular: append the smallest number of background elements.

MATLAB Commands

Dilation in MATLAB is performed with the command

```
>> imdilate(image, kernel)
```

To see an example of dilation, consider the commands:

```
>> t=inread('text.tif');  
>> sq=ones(3,3);  
>> td=imdilate(t,sq);  
>> subplot(1,2,1),imshow(t)  
>> subplot(1,2,2),imshow(td)
```

The result is shown in figure 9.5. Notice how the image has been “thickened”. This is really what dilation does; hence its name.



Figure 9.5: Dilation of a binary image

9.3.2 Erosion

Given sets A and B , the *erosion of A by B* , written $A \ominus B$, is defined as:

$$A \ominus B = \{w : B_w \subseteq A\}.$$

In other words the erosion of A by B consists of all points $w = (x, y)$ for which B_w is in A . To perform an erosion, we can move B over A , and find all the places it will fit, and for each such place mark down the corresponding $(0, 0)$ point of B . The set of all such points will form the erosion.

An example of erosion is given in figures 9.6.

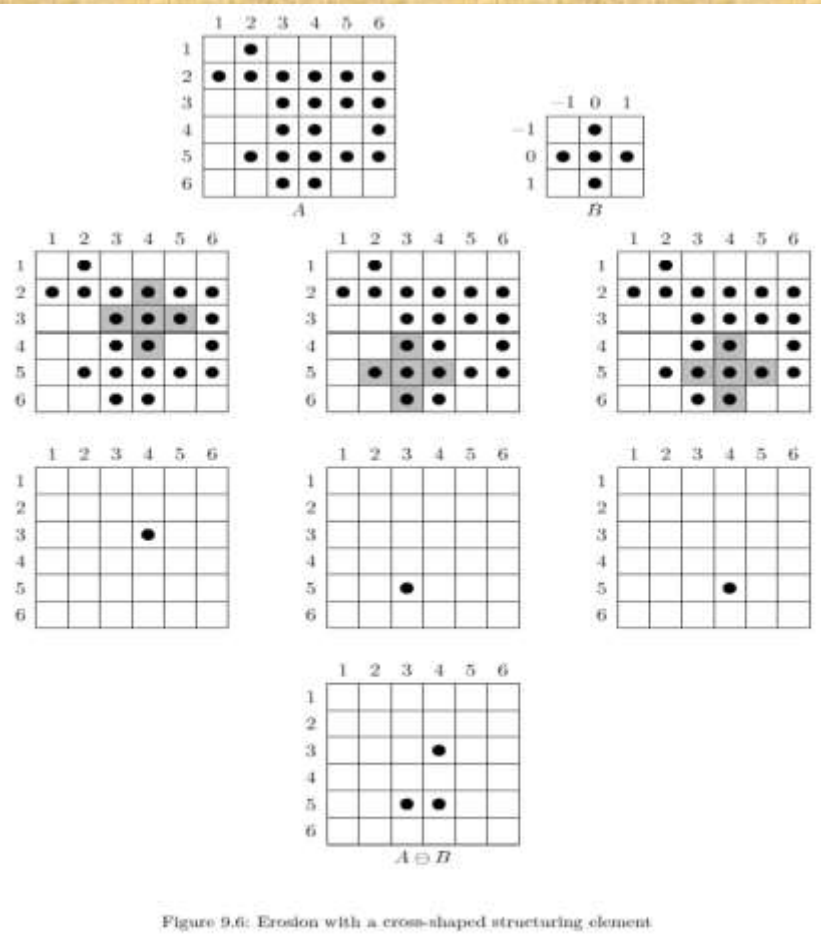


Figure 9.6: Erosion with a cross-shaped structuring element

For erosion, as for dilation, we generally assume that A is the image being processed, and B is a small set of pixels: the structuring element or kernel.

Erosion is related to *Minkowski subtraction*: the Minkowski subtraction of B from A is defined as:

$$A - B = \bigcap_{b \in B} A_b.$$

Erosion in MATLAB is performed with the command

```
>> imerode(image, kernel)
```

We shall give an example; using a different binary image:

```
>> c=imread('circbw.tif');  
>> ce=imerode(c,sq);  
>> subplot(1,2,1),imshow(c)  
>> subplot(1,2,2),imshow(ce)
```

The result is shown in figure 9.8. Notice how the image has been “thinned”. This is the expected result of an erosion; hence its name. If we kept on eroding the image, we would end up with a completely black result.

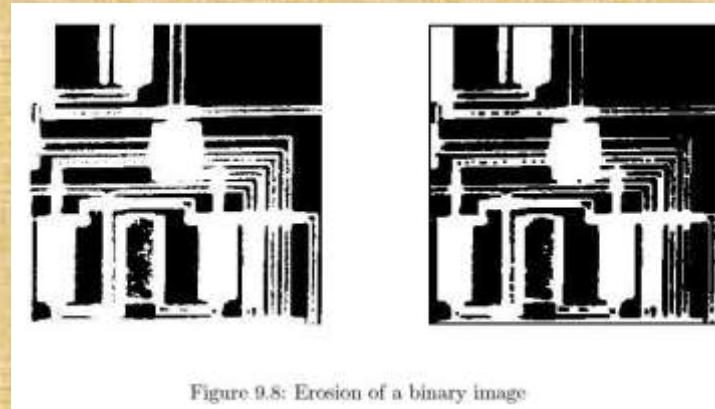


Figure 9.8: Erosion of a binary image

Relationship between Dilation and Erosion •

It can be similarly shown that the same relationship holds if erosion and dilation are interchanged; that

$$\overline{A \oplus B} = \overline{A} \ominus \overline{B}.$$

We can demonstrate the truth of these using MATLAB commands; all we need to know is that the complement of a binary image

b

is obtained using

```
>> ~b
```

and that given two images **a** and **b**; their equality is determined with

```
>> all(a(:)==b(:))
```

To demonstrate the equality

$$\overline{A \oplus B} = \overline{A} \oplus \overline{B},$$

pick a binary image, say the text image, and a structuring element. Then the left hand side of this equation is produced with

```
>> lhs=imerode(t,sq);
```

and the right hand side with

```
>> rhs=imdilate(~t,sq);
```

Finally, the command

```
>> all(lhs(:)==rhs(:))
```

should return 1, for true.

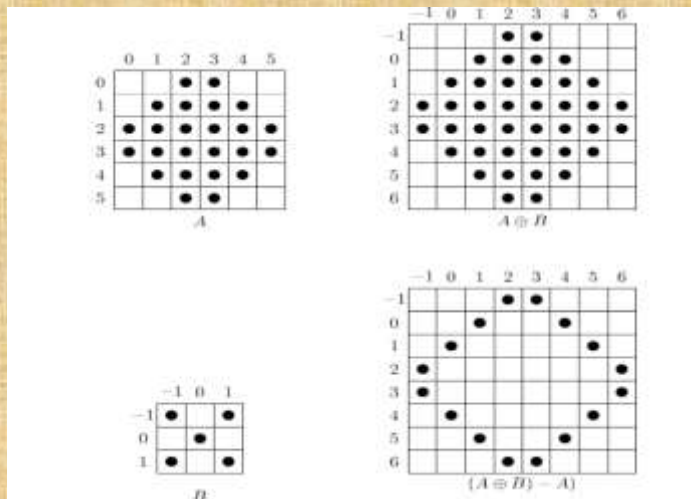
9.3.3 An application: boundary detection

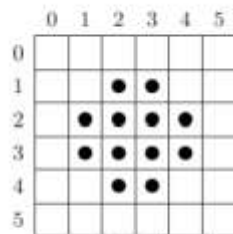
If A is an image, and B a small structuring element consisting of point symmetrically places about the origin, then we can define the boundary of A by any of the following methods:

- (i) $A - (A \oplus B)$ "internal boundary"
- (ii) $(A \oplus B) - A$ "external boundary"
- (iii) $(A \oplus B) - (A \ominus B)$ "morphological gradient"

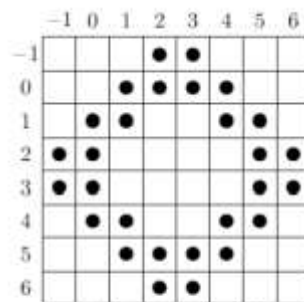
In each definition the minus refers to set difference. For some examples, see figure 9.9. Note that the internal boundary consists of those pixels in A which are at its edge; the external boundary consists of pixels outside A which are just next to it, and that the morphological gradient is a combination of both the internal and external boundaries.

To see some examples, choose the image `rice.tif`, and threshold it to obtain a binary image:





$A \ominus B$



$(A \ominus B) - (A \ominus B)$

Figure 9.9: Boundaries

Example:

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

Then the internal boundary is obtained with:

```
>> re=imerode(r,sq);
>> r_int=r&~re;
>> subplot(1,2,1),imshow(r)
>> subplot(1,2,2),imshow(r_int)
```

The result is shown in figure 9.10.



Figure 9.10: "Internal boundary" of a binary image

The external boundary and morphological gradients can be obtained similarly:

```
>> rd=imdilate(r,sq);  
>> r_ext=rd&~r;  
>> r_grad=rd&~re;  
>> subplot(1,2,1),imshow(r_ext)  
>> subplot(1,2,2),imshow(r_grad)
```

The results are shown in figure 9.11.

Note that the external boundaries are larger than the internal boundaries. This is because the internal boundaries show the outer edge of the image components; whereas the external boundaries show the pixels just outside the components. The morphological gradient is thicker than either, and is in fact the union of both.

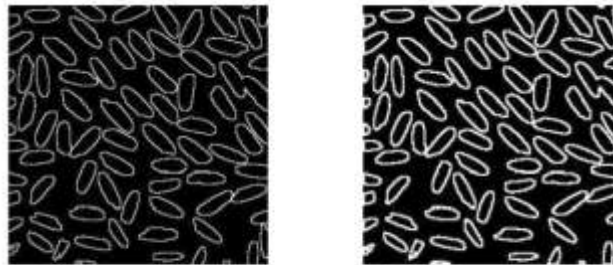


Figure 9.11: "External boundary" and the morphological gradient of a binary image

As we've seen, dilation expands the components of an image while the erosion shrinks them.

Opening generally smooths the contour of an object and eliminates thin protrusions.

Closing also tends to smooth sections of contours but fuses narrow breaks and long, thin gulfs and eliminates small holes and filling gaps in the contour.

Next chapter explains these two concepts in more detail.

Chapter 10

Mathematical morphology (2)

10.1 Opening and closing

These operations may be considered as “second level” operations; in that they build on the basic operations of dilation and erosion. They are also, as we shall see, better behaved mathematically.

10.1.1 Opening

Given A and a structuring element B , the *opening of A by B* , denoted $A \circ B$, is defined as:

$$A \circ B = (A \ominus B) \oplus B.$$

So an opening consists of an erosion followed by a dilation. An equivalent definition is

$$A \circ B = \cup \{B_w : B_w \subseteq A\}.$$

That is, $A \circ B$ is the union of all translations of B which fit inside A . Note the difference with erosion: the erosion consists only of the $(0,0)$ point of B for those translations which fit inside A ; the opening consists of all of B . An example of opening is given in figure 10.1.

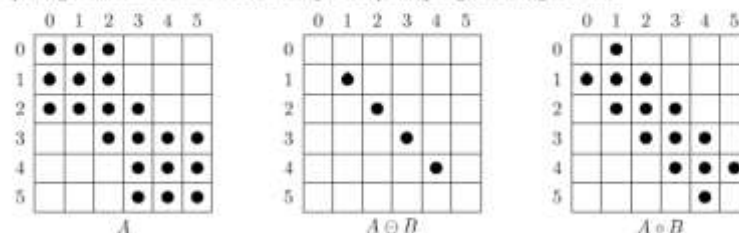


Figure 10.1: Opening

10.1.2 Closing

Analogous to opening we can define *closing*, which may be considered as a dilation followed by an erosion, and is denoted $A \bullet B$:

$$A \bullet B = (A \oplus B) \ominus B.$$

Another definition of closing is that $x \in A \bullet B$ if all translations B_w which contain x have non-empty intersections with A . An example of closing is given in figure 10.2. The closing operation satisfies

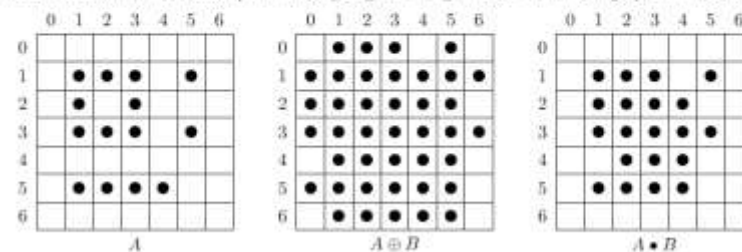


Figure 10.2: Closing

Example:

Opening and closing are implemented by the `imopen` and `inclose` functions respectively. We can see the effects on a simple image using the square and cross structuring elements.

```
>> cr=[0 1 0;1 1 1;0 1 0];
>> test=zeros(10,10);test(2:6,2:4)=1;test(3:5,6:9)=1;test(8:9,4:8)=1;test(4,5)=1
```

```
test =
```

```
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 1 1 1 1 0
0 1 1 1 1 1 1 1 1 0
0 1 1 1 0 1 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 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
```

```
>> imopen(test,sq)
```

```
ans =
```

```
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 0 1 1 1 1 0
0 1 1 1 0 1 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 0 0 0 0 0
```

```
>> imopen(test,cr)
```

```
ans =
```

```
0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 1 1 1 0 1 1 1 0 0
0 1 1 1 1 1 1 1 1 0
0 1 1 1 0 1 1 1 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 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
```

Note that in each case the image has been separated into distinct components, and the lower part has been removed completely.

```
>> inclose(test,sq)
```

```
ans =
```

```

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

```

```

>> imclose(test,cr)

ans =

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

```

With closing, the image is now fully “joined up”. We can obtain a joining-up effect with the text image, using a diagonal structuring element.

```

>> diag=[0 0 1;0 1 0;1 0 0]

diag =

0 0 1
0 1 0
1 0 0

>> tc=imclose(t,diag);
>> imshow(tc)

```

The result is shown in figure 10.3.

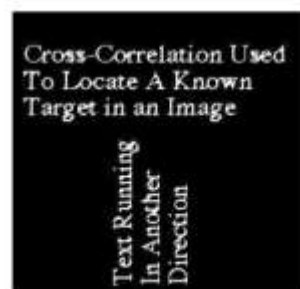


Figure 10.3: An example of closing

An application: noise removal

Suppose A is a binary image corrupted by impulse noise—some of the black pixels are white, and some of the white pixels are black. An example is given in figure 10.4. Then $A \ominus B$ will remove the single black pixels, but will enlarge the holes. We can fill the holes by dilating twice:

$$((A \ominus B) \oplus B) \oplus B.$$

The first dilation returns the holes to their original size; the second dilation removes them. But this will enlarge the objects in the image. To reduce them to their correct size, perform a final erosion:

$$(((A \ominus B) \oplus B) \oplus B) \ominus B.$$

The inner two operations constitute an opening; the outer two operations a closing. Thus this noise removal method is in fact an opening followed by a closing:

$$(A \circ B) \bullet B.$$

This is called *morphological filtering*.

Suppose we take an image and apply 10% shot noise to it:

```
>> c=imread('circles.tif');
>> x=rand(size(c));
>> d1=find(x<0.05);
>> d2=find(x>0.95);
>> c(d1)=0;
>> c(d2)=1;
>> imshow(c)
```

The result is shown as figure 10.4(a). The filtering process can be implemented with

```
>> cf1=imclose(imopen(c,sq),sq);
>> figure,imshow(cf1)
>> cf2=imclose(imopen(c,cr),cr);
>> figure,imshow(cf2)
```

and the results are shown as figures 10.4(b) and (c). The results are rather "blocky"; although less so with the cross structuring element.

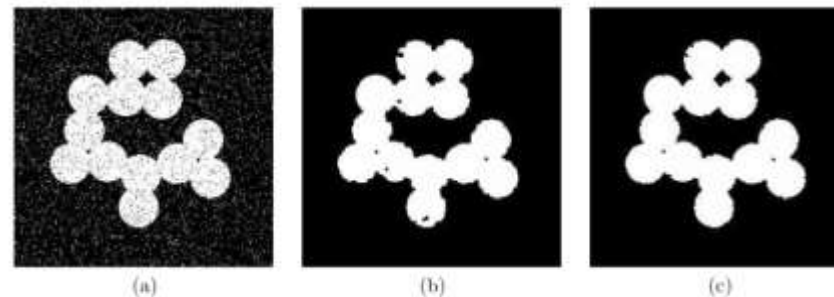
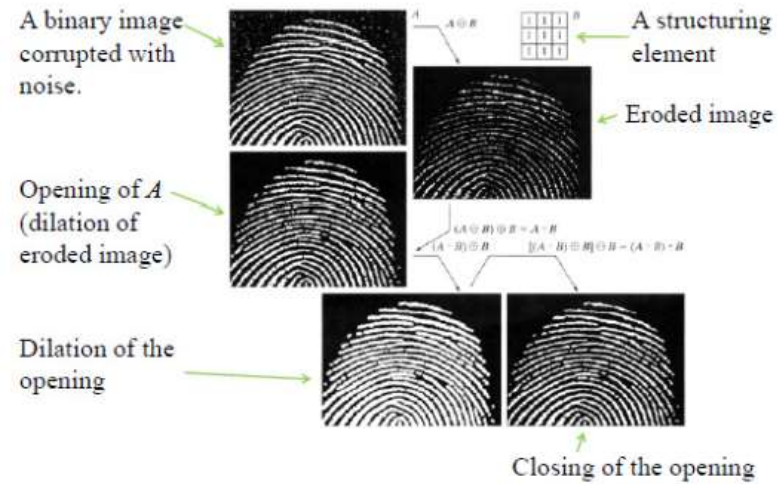


Figure 10.4: A noisy binary image and results after morphological filtering with different structuring elements.

Opening and closing



10.2 The hit-or-miss transform

This is a powerful method for finding shapes in images. As with all other morphological algorithms, it can be defined entirely in terms of dilation and erosion; in this case, erosion only.

Suppose we wish to locate 3×3 square shapes, such as is in the centre of the image A in figure 10.5.

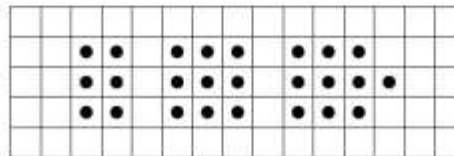


Figure 10.5: An image A containing a shape to be found

If we performed an erosion $A \ominus B$ with B being the square structuring element, we would obtain the result given in figure 10.6.

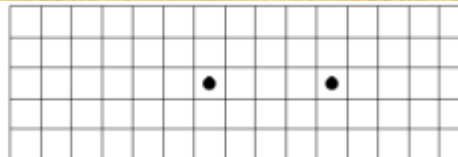


Figure 10.6: The erosion $A \ominus B$

The result contains two pixels, as there are exactly two places in A where B will fit. Now suppose we also erode the complement of A with a structuring element C which fits exactly around the 3×3 square; \bar{A} and C are shown in figure 10.7. (We assume that $(0, 0)$ is at the centre of C .)

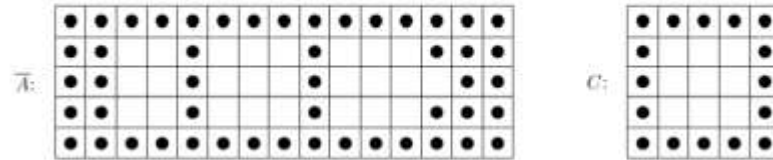


Figure 10.7: The complement and the second structuring element

If we now perform the erosion $\bar{A} \ominus C$ we would obtain the result shown in figure 10.8.

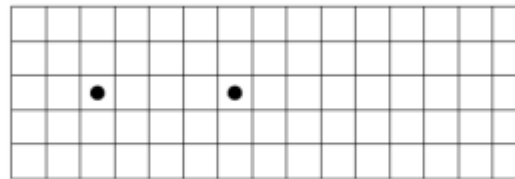


Figure 10.8: The erosion $\bar{A} \ominus C$

The intersection of the two erosion operations would produce just one pixel at the position of the centre of the 3×3 square in A , which is just what we want. If A had contained more than one square, the final result would have been single pixels at the positions of the centres of each. This combination of erosions forms the hit-or-miss transform.

In general, if we are looking for a particular shape in an image, we design two structuring elements: B_1 which is the same shape, and B_2 which fits around the shape. We then write $B = (B_1, B_2)$ and

$$A \circledast B = (A \ominus B_1) \cap (\bar{A} \ominus B_2)$$

Example:

As an example, we shall attempt to find the hyphen in "Cross-Correlation" in the text image shown in figure 9.5. This is in fact a line of pixels of length six. We thus can create our two structuring elements as:

```
>> b1=ones(1,6);
>> b2=[1 1 1 1 1 1 1;1 0 0 0 0 0 1; 1 1 1 1 1 1 1];
>> tb1=erode(t,b1);
>> tb2=erode(~t,b2);
>> hit_or_miss=tb1&tb2;
>> [x,y]=find(hit_or_miss==1)
```

and this returns a coordinate of (41,76), which is right in the middle of the hyphen. Note that the command

```
>> tb1=erode(t,b1);
```

is not sufficient, as there are quite a few lines of length six in this image. We can see this by viewing the image tb1, which is given in figure 10.9.



Figure 10.9: Text eroded by a hyphen-shaped structuring element

10.3.3 Skeletonization

Recall that the *skeleton* of an object can be defined by the "medial axis transform"; we may imagine fires burning in along all edges of the object. The places where the lines of fire meet form the skeleton. The skeleton may be produced by morphological methods.

Why skeletonization

In real world there is a need for skeletonization of images due to following reasons:

- .1 To reduce the amount of data and time required to be processed.
- .2 Extraction of critical features such as end-points, junction-points, and connection among the components.
- .3 The vectorization algorithms often used in pattern recognition tasks also require one-pixel-wide lines as input.
- .4 Shape analysis can be more easily made on line like patterns.

Applications

- Handwritten and printed characters
- Fingerprint patterns
- Chromosomes & biological cell structures

- Circuit diagrams
- Engineering drawings.

Consider the table of operations as shown in table 10.1.

Erosions	Openings	Set differences
A	$A \circ B$	$A - (A \circ B)$
$A \ominus B$	$(A \ominus B) \circ B$	$(A \ominus B) - ((A \ominus B) \circ B)$
$A \ominus 2B$	$(A \ominus 2B) \circ B$	$(A \ominus 2B) - ((A \ominus 2B) \circ B)$
$A \ominus 3B$	$(A \ominus 3B) \circ B$	$(A \ominus 3B) - ((A \ominus 3B) \circ B)$
\vdots	\vdots	\vdots
$A \ominus kB$	$(A \ominus kB) \circ B$	$(A \ominus kB) - ((A \ominus kB) \circ B)$

Table 10.1: Operations used to construct the skeleton

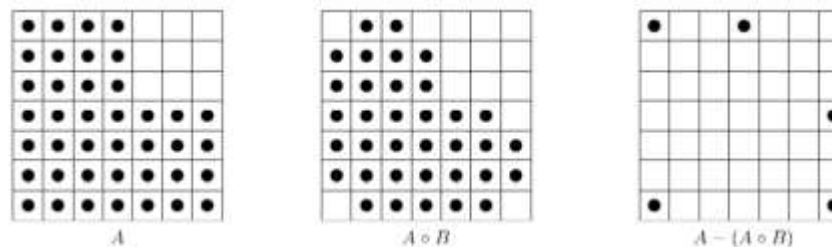
Here we use the convention that a sequence of k erosions using the same structuring element B is denoted $A \ominus kB$. We continue the table until $(A \ominus kB) \circ B$ is empty. The skeleton is then obtained by taking the unions of all the set differences. An example is given in figure 10.17, using the cross structuring element.

Since $(A \ominus 2B) \circ B$ is empty, we stop here. The skeleton is the union of all the sets in the third column; it is shown in figure 10.18. This method of skeletonization is called *Lantuéjoul's method*; for details see Serra [12].

This algorithm again can be implemented very easily; a function to do so is shown in figure 10.19. We shall experiment with the nice work image.

```
>> nk=inskel(n,sq);
>> imshow(nk)
>> nk2=inskel(n,cr);
>> figure,imshow(nk2)
```

The result is shown in figure 10.20. Image (a) is the result using the square structuring element; Image (b) is the result using the cross structuring element.



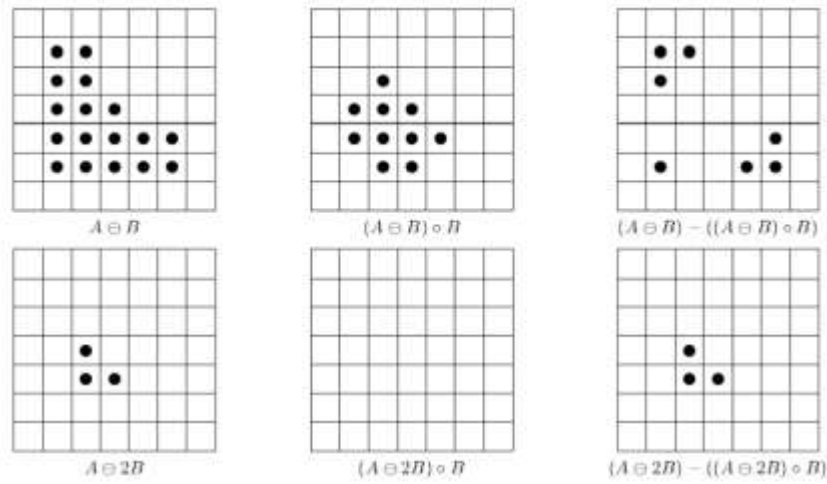


Figure 10.17: Skeletonization

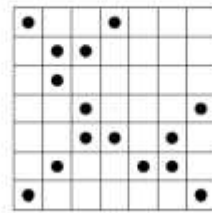


Figure 10.18: The final skeleton

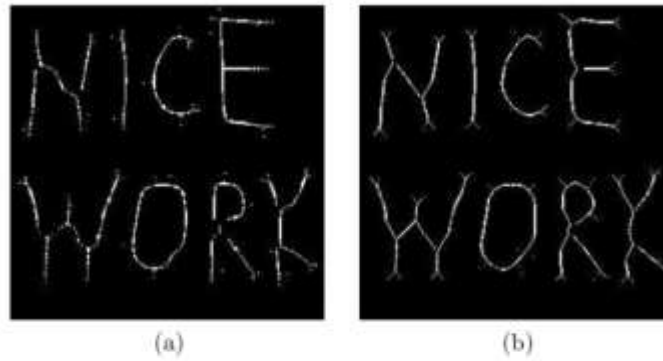


Figure 10.20: Skeletonization of a binary image

Tutorial No. 9

1. For each of the following images A and structuring elements B :

$A =$

```

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

```

```

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

```

$B =$

```

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

```

calculate the erosion $A \ominus B$, the dilation $A \oplus B$, the opening $A \circ B$ and the closing $A \bullet B$.

Check your answers with MATLAB.

2. Suppose a square object was eroded by a circle whose radius was about one quarter the side of the square. Draw the result.

3. Using the 3×3 square structuring element, compute the skeletons of

- (a) a 7 square,
- (b) a 5×9 rectangle,
- (c) an L shaped figure formed from an 8×8 square with a 3×3 square taken from a corner,
- (d) an H shaped figure formed from a 15×15 square with 5×5 squares taken from the centres of the top and bottom,
- (e) a cross formed from an 11×11 square with 3×3 squares taken from each corner.

In each case check your answer with MATLAB

4. Repeat the above question but use the cross structuring element.

11

Chapter Eleven

2D Discrete Wavelet Transform

12.1 Wavelet Concepts

Wavelet Analysis

Wavelets are short wavelike functions that can be scaled and translated. Wavelet transforms take any signal and express it in terms of scaled and translated wavelets. The resulting wavelet transform is a representation of the signal at different scales. The transform allows you to manipulate features at different scales independently, such as suppressing or strengthening some particular feature.

Mathematica provides a full-featured implementation of wavelet analysis, supporting many wavelet families, continuous wavelet transform, and several discrete wavelet transforms (standard, stationary, lifting, packet, ...). Discrete wavelet transforms directly work with arrays of any dimension as well as sounds and images, giving a symbolic representation of the transform, which can be directly processed or visualized, etc.

Scaling and Wavelet Functions

WaveletPhi — scaling function ϕ ("father wavelet") for any wavelet family

WaveletPsi — wavelet function ψ ("mother wavelet") for any wavelength family

WaveletFilterCoefficients — wavelet filter coefficients

Discrete Wavelet Families

HaarWavelet — representation of the Haar wavelet

DaubechiesWavelet — Daubechies wavelet family

Transform coefficients are obtained by projecting the 2-D input signal onto 2-D basis functions. The separable 2-D basis functions can be expressed as the product of two 1-D basis functions. Unlike two basis functions for 1-D signals at a given scale, there are four basis functions for 2-D signals as given in equation (2.11).

$$\begin{aligned}
 \phi(u, v) &= \phi(u)\phi(v) \\
 \psi_1(u, v) &= \psi(u)\phi(v) \\
 \psi_2(u, v) &= \phi(u)\psi(v) \\
 \psi_3(u, v) &= \psi(u)\psi(v)
 \end{aligned} \tag{2.11}$$

$\phi(u, v)$ can be thought of as the 2-D scaling function; $\psi_1(u, v)$, $\psi_2(u, v)$ and $\psi_3(u, v)$ are the three 2-D wavelet functions. For a 2-D input signal $x(u, v)$, the transform coefficients are obtained by projecting the input onto the four basis functions given in equation (2.11). This

2. Wavelets (Kernels) Properties

- *Scalability, Separability, Translatability*

$\phi(x, y) = \phi(x)\phi(y)$ is a one separable two dimensions scaling function

Horizontal detail is $\varphi^H(x, y) = \varphi(x)\phi(y)$

Vertical detail is $\varphi^V(x, y) = \phi(x)\varphi(y)$

Diagonal detail is $\varphi^D(x, y) = \varphi(x)\varphi(y)$

The general Wavelet formula can be represented as,

$$\phi_{j,k}(x) = 2^{\frac{j}{2}} \phi(2^{\frac{j}{2}}x - k)$$

$$\varphi_{j,k}(x) = 2^{\frac{j}{2}} \varphi(2^{\frac{j}{2}}x - k)$$

3. Wavelet Applications

- **Numerical Analysis:** like partial differentiation
- **Signal analysis:** like audio/video/image compression, texture classification and finger print
- **Control Applications:** like motion detection and tracking and robot position, Encoder/quantization de-noising
- **Audio Applications:** like speech recognition, speech enhancement, and audio de-noising

12.4 Wavelets and Compression

Wavelets are useful for **compressing signals** but they also have far more extensive uses. They can be used to **process and improve signals**, in fields such as medical imaging of particular use. They can be used to **remove noise** in an image, for example if it is of very fine scales, wavelets can be used to cut out this fine scale, effectively removing the noise. "Image compression algorithms aim to **remove redundancy** in data in a way which makes **image reconstruction** possible." This basically means that image compression algorithms try to exploit redundancies in the data; they calculate which data needs to be kept in order to reconstruct the original image and therefore which data can be 'thrown away'. Redundancy reduction is aimed at removing duplication in the image.

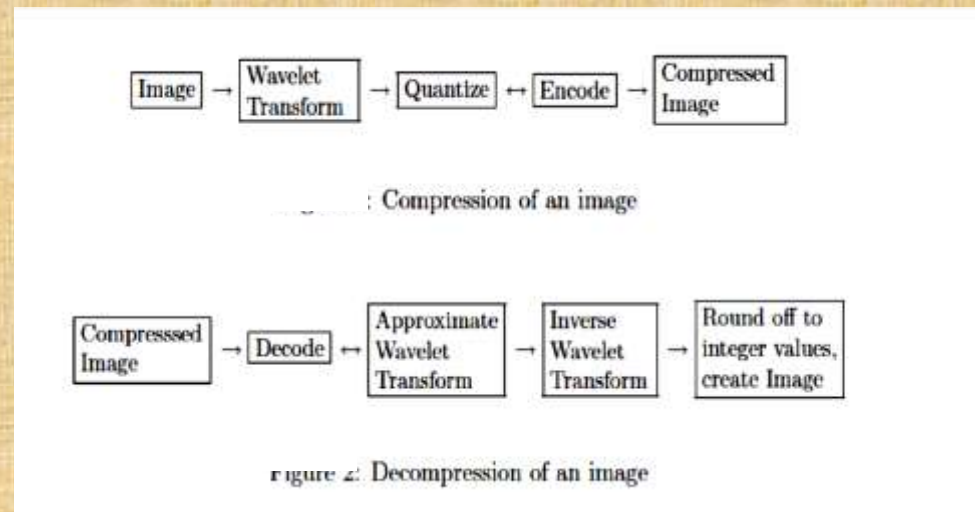


Figure 12.1 Image compression/de-compression block diagram

12.5 Wavelet Transform and Its Inverse

Two dimensional discrete wavelet transform

The DWT described in the previous section is for one dimensional (1-D) signals. Images are 2-D and are analyzed using a separable 2-D wavelet transform. A 2-D separable transform is equivalent to two 1-D transforms in series. It is implemented as a 1-D row transform followed by a 1-D column transform on the data obtained from the row transform. Figure 12.2 shows the filter bank structure for computation of a 2-D DWT and IDWT.

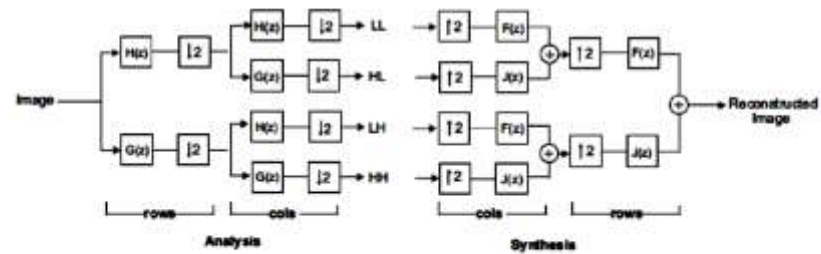


Fig. 12.2 One level filter bank for computation of 2-D DWT.

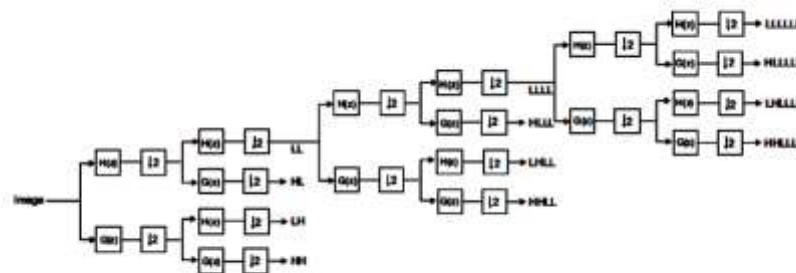


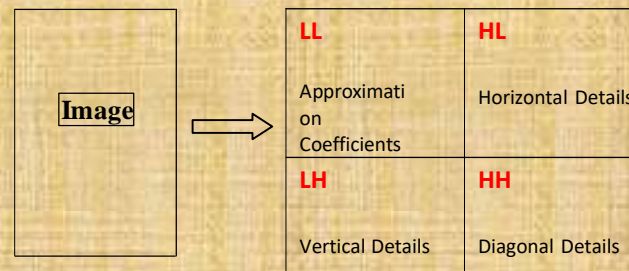
Fig. 12.3 3-level 2-D decomposition

decomposition of the *Lighthouse* image; notice that the LL subband from the first stage has been transformed into 4 subbands- the three other subbands remain unchanged. Color gray in the figure corresponds to the value zero.

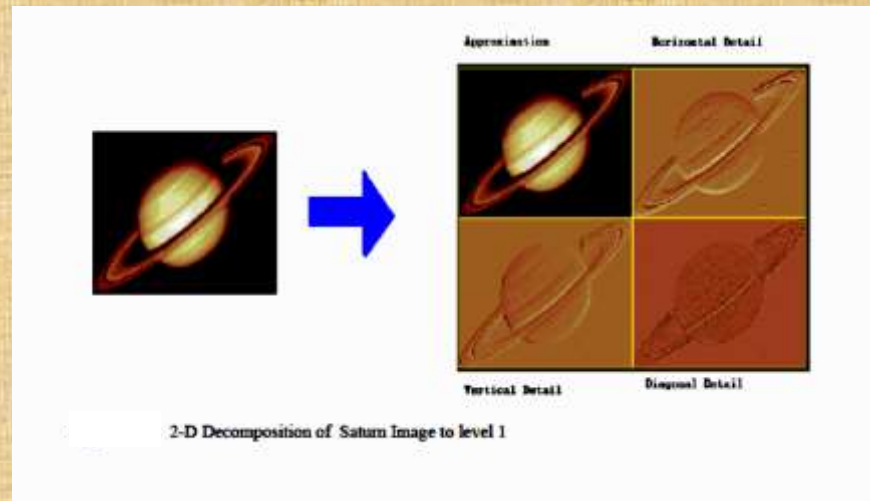
12.6 Why Down and Up Sampling ?

The low and high-pass filters $L(z)$ and $H(z)$ split the frequency content of the signal in half. It therefore seems logical to perform a down sampling with a factor two to avoid redundancy. If half of the samples of the filtered signals $c_l(k)$ and $c_h(k)$ are reduced, it is still possible to reconstruct the signal $x(k)$. The down sampling operation $(\downarrow 2)$ only the even-numbered components of the filter output, hence it is not invertible. In the frequency domain, the effect of discarding information is called aliasing. If the Shannon sampling theorem is met, no loss of information occurs. The sampling theorem of Shannon states that down sampling a sampled signal by a factor M produces a signal whose spectrum can be calculated by partitioning the original spectrum into M equal bands and summing these bands. In the synthesis bank the signals are first up sampled before filtering. The up sampling by a factor $(\uparrow 2)$ is performed by adding zeros in between the samples of the original signal. Note that first down sampling a signal and then up sampling it again will not return the original signal. The transpose of $(\downarrow 2)$, since $(\uparrow 2)$ and $(\downarrow 2)$ are transposes come in reverse order, synthesis can be performed as the transpose of the analysis. Furthermore $(\downarrow 2)(\uparrow 2) = I$, since $(\uparrow 2)$ is the right-inverse of $(\downarrow 2)$. This indicates that it is possible to obtain the original signal again with up- and down sampling. By first inserting zeros and then removing them, the original signal is obtained again.

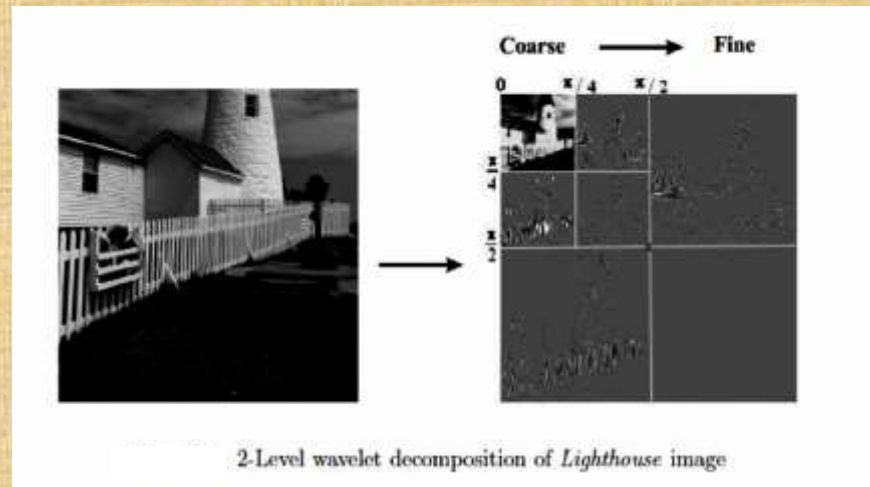
$$x = \begin{bmatrix} \cdot \\ x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ \cdot \end{bmatrix} \quad (\downarrow 2)x = \begin{bmatrix} \cdot \\ x(0) \\ x(2) \\ x(4) \\ \cdot \end{bmatrix} \quad (\uparrow 2)(\downarrow 2)x = \begin{bmatrix} \cdot \\ x(0) \\ 0 \\ x(2) \\ 0 \\ x(4) \\ \cdot \end{bmatrix} .$$



Example-1



Example-2



12.7 Why Circular Convolution?

Example -3

$x = [1\ 2\ 3\ 2\ 1]$ and $h = [3\ 2\ 1]$.

the answer from matlab of circular convolving the x and h is $[7\ 9\ 14\ 14\ 10]$. But HOW?

The MATLAB command is

```
cconv([3 2 1],[1 2 3 2 1],5)
```

where the number '5' as input indicates N=5 point circular convolution. So the number of outputs will be 5

Let see how to achieve the result.

For circular convolution flip $h=[3\ 2\ 1]$ as $h' = 1\ 2\ 3$ and find the sum of products as shown below

Step 1:

```
1 2 3 2 1
```

```
0 0 1 2 3 ( need to add two zeros since N=5 and size of h is only 3.
```

```
1x0+2x0+3x1+2x2+1x3 = 3 + 4+3 = 10
```

Step 2: (Circularly shift the second row by 1 digit to the left

```
1 2 3 2 1
```

```
0 1 2 3 0
```

```
0+2+6+6+0 = 14
```

Step 3:

```
1 2 3 2 1
```

```
1 2 3 0 0
```

```
1+4+9+0+0 = 14
```

Step4:

```
1 2 3 2 1
```

```
2 3 0 0 1
```

```
2+6+0+0+1=9
```

Step 5:

```
1 2 3 2 1
```

```
3 0 0 1 2
```

```
3+0+0+2+2=7
```

So the output is

```
[ 7 9 14 14 10]
```

12.8 Wavelet Families, Haar and Daubechies Filters

The one dimension Haar filters are:

$$\begin{aligned} \text{LOWPASS: } & \frac{1}{\sqrt{2}} [1 \ 1] \\ \text{HIGHPASS: } & \frac{1}{\sqrt{2}} [1 \ -1] \end{aligned}$$

An example of Daubechies basis vectors (there are many others) follows:

$$\begin{aligned} \text{LOWPASS: } & \frac{1}{4\sqrt{2}} [1+\sqrt{3}, \ 3+\sqrt{3}, \ 3-\sqrt{3}, \ 1-\sqrt{3}] \\ \text{HIGHPASS: } & \frac{1}{4\sqrt{2}} [1-\sqrt{3}, \ \sqrt{3}-3, \ 3+\sqrt{3}, \ -1-\sqrt{3}] \end{aligned}$$

To use the basis vectors to implement the wavelet transform, they must be zero-padded to be the same size as the image (or subimage). Also note that the origin of the

EXAMPLE 2-14

We want to use the Haar basis vectors to perform a wavelet transform on an image by dividing it into 4×4 blocks. The basis vectors need to be zero-padded so that they have a length of 4, as follows:

$$\begin{aligned} \text{LOWPASS: } & \frac{1}{\sqrt{2}} [1 \ 1 \ 0 \ 0] \\ \text{HIGHPASS: } & \frac{1}{\sqrt{2}} [1 \ -1 \ 0 \ 0] \end{aligned}$$

↑
origin

These are aligned with the image so that the origins coincide, and the result from the first vector inner product is placed into the location corresponding to the origin. Note that when the vector is zero-padded on the right, the origin is no longer to the right of the center of the resulting vector. The origin is determined by selecting the coefficient corresponding to the right of center before zero-padding.

EXAMPLE 2-15

To use the Daubechies basis vectors to do a wavelet transform on an image by dividing it into 8×8 blocks, we need to zero-pad them to a length of 8, as follows:

$$\begin{aligned} \text{LOWPASS: } & \frac{1}{4\sqrt{2}} [1+\sqrt{3}, \ 3+\sqrt{3}, \ 3-\sqrt{3}, \ 1-\sqrt{3}, \ 0, \ 0, \ 0, \ 0] \\ \text{HIGHPASS: } & \frac{1}{4\sqrt{2}} [1-\sqrt{3}, \ \sqrt{3}-3, \ 3+\sqrt{3}, \ -1-\sqrt{3}, \ 0, \ 0, \ 0, \ 0] \end{aligned}$$

↑
origin