FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

DEPARTMENT OF INFORMATION TECHNOLOGY

CLASS: B.E. **SEMESTER : VII**

SUBJECT: DIGITAL SIGNAL AND IMAGE PROCESSING

Experiment Number 1

Problem Definition: Point Processing Techniques

Software requirements: Matlab, C

Theory:

Point Processing Techniques

We work with single pixels i.e. 1X1 operator. New value of g(x,y) depends on operator T and the present f(x,y). Some common point processing techniques are :

Digital negative

A common example of digital negative is the displaying of an X- ray image. Negative simply means inverting gray levels i.e. black in original image will now look white and vice versa.

The transformation is given by s = (L-1)-r, where L is the number of gray levels.

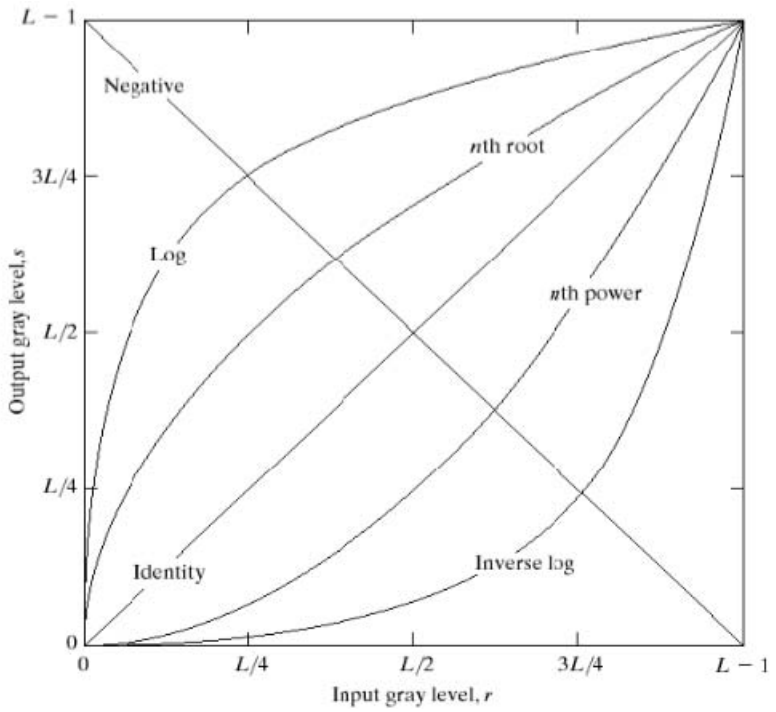


FIGURE 3.3 Some basic gray-level transformation functions used for image enhancement.

Contrast stretching

Idea is to increase the contrast of the images by making dark portions darker and the bright portions brighter. Dark levels are made darker by assigning a slope of less than one and make bright levels are made brighter by assigning slope greater than one. Slopes can be assigned depending on input image and the application.

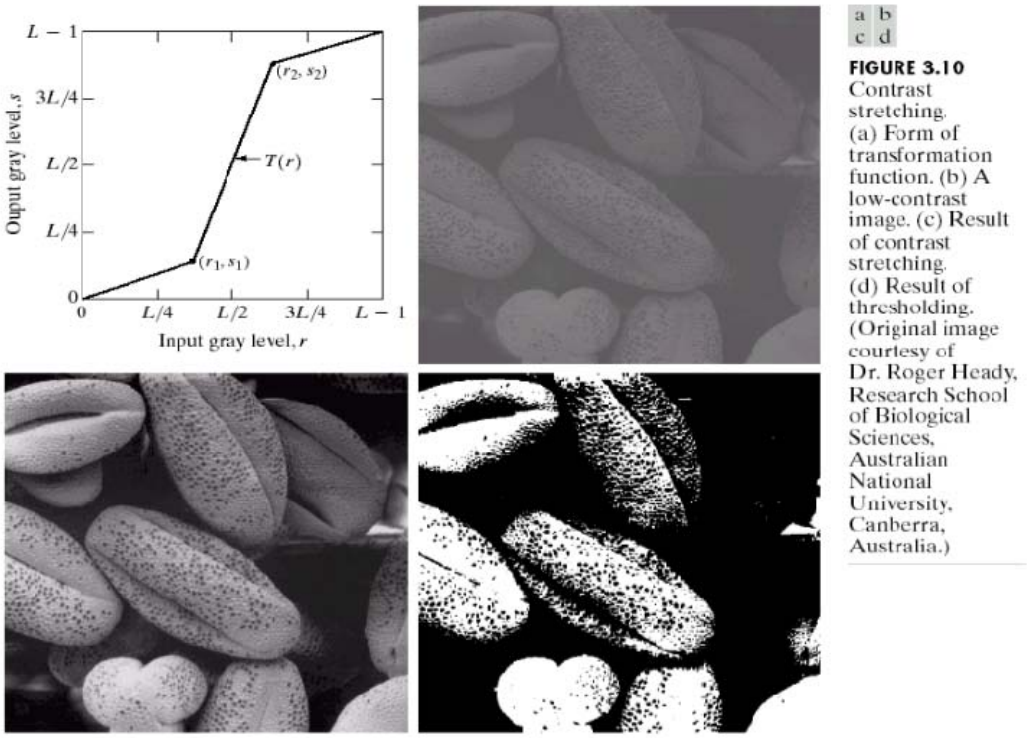
The transformation can be given as

s = l.r 0<=r<a

= m.(r-a) + v a<=r<b

= n.(r-b) + w b<=r<L-1

Where l,m, and n are the slopes.



a b
c d

FIGURE 3.10 Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)

Thresholding

Extreme contrast stretching gives thresholding. In contrast stretching if the first and the last slope are made zero, and the centre slope is increased, we get thresholding transformation.

Transformation can be given as

S = 0 if r<=a

= L-1 if r>a

Where L is the number of gray levels.

Thresholded image has a maximum contrast as it has only black and white gray levels.

Gray Level Slicing

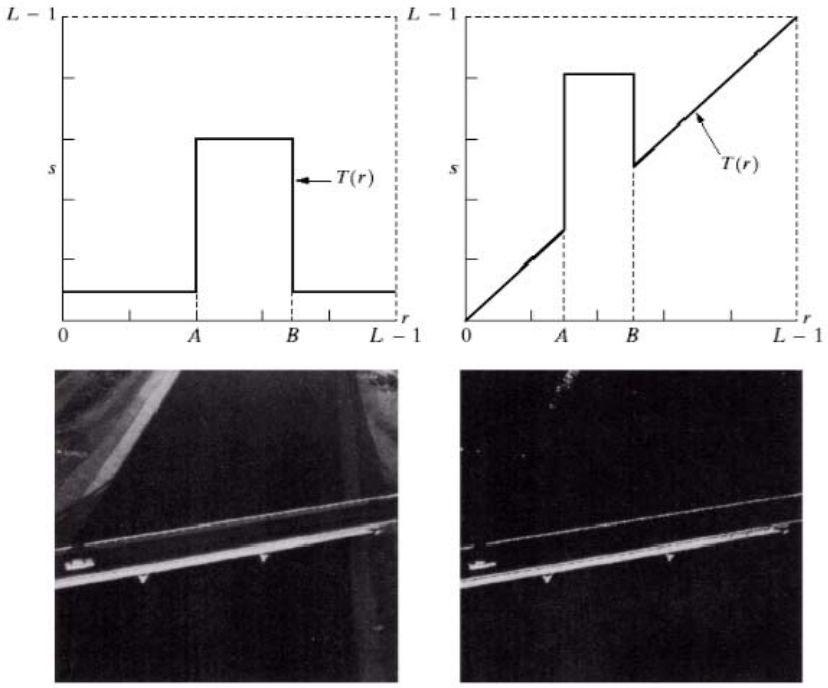
Sometimes we need to highlight a specific range of gray levels. In such cases we use a gray level slicing. It is similar to thresholding function except that here we select a band of gray level values.

Transformation can be given as

s = L-1 if a<=r<b

= 0 otherwise

This method is known as gray level slicing without background. This is because in this process we have completely lost the background.



a b
c d

FIGURE 3.11
(a) This transformation highlights range [*A*, *B*] of gray levels and reduces all others to a constant level. (b) This transformation highlights range [*A*, *B*] but preserves all other levels. (c) An image. (d) Result of using the transformation in (a).

Dynamic Range Compression

At times, dynamic range of the image exceeds the capability of display device. Some pixel values are so large that the other low value pixels get obscured. A simple example of such a phenomena is thatduring daytime, we cannot see stars. Something needs to be done to be able to see the small values as well. This technique of compressing the dynamic range is known as dynamic range compression. It is achieved using a log operator.

Transformation is given as $s = C \times \log(1+|r|)$. Here $(1+|r|)$ is used because when r=0, value of s will be undefined. |r| is used as Fourier transform values are complex. C is the constant.

Conclusion:

In point processing techniques only one pixel is considered at a time and modified according to the requirement.

Refer Pg. No. Refer Pg. No. 100, ‘Digital Image Processing’, Gonzales, Woods, Pearson Education, Second Edition

Code: Point Processing Techniques

% Image Negative

```
clear all
f=imread('cameraman.bmp');
a=double(f);
c=255;
b=255-a;
figure(1)
imshow(uint8(a));
figure(2);
imshow(uint8(b));
title('Image Negative');
```

% Contrast Stretching

```
clear all
f=imread('cameraman.bmp');
z=double(f);
[row col]=size(z);
s1=input('Enter the value of slope1')
s2=input('Enter the value of slope2')
s3=input('Enter the value of slope3')
lt=input('Enter Lower Threshold')
ut=input('Enter Upper Threshold')
for x=1:1:row
   for y=1:1:col
      if(z(x,y)<=lt)
         b(x,y)=s1*z(x,y);
      else if(z(x,y)<=ut)
            b(x,y)=s2*(z(x,y)-lt)+s1*lt;
         else b(x,y)=s1*(z(x,y)-ut)+s3*lt+s2*(ut-lt);
         end
      end
   end
end
figure(1)
imshow(uint8(z));
figure(2)
imshow(uint8(b));
```

% Thresholding

```
clc
p=imread('cameraman.bmp')
a=p;
[row col]=size(p)
t=input('Enter the value of Threshold')
for x=1:1:row
   for y=1:1:col
      if(a(x,y)<t)
         a(x,y)=0;
      else a(x,y)=255;
```

```
            end
        end
    end

    figure(1);
    imshow(p);
    figure(2);
    imshow(a);
    title('Thresholding');
```

% Gray Level Slicing

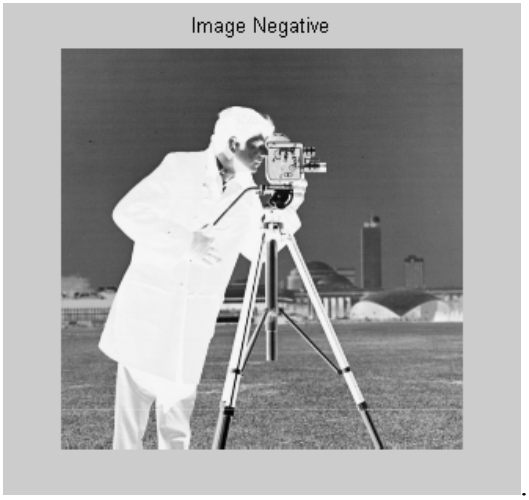
```
    clear all
    p=imread('cameraman.bmp');
    % z=rgb2gray(p);
    figure(1);
    imshow(p);
    lt=input('Enter lower threshold');
    ut=input('Enter upper threshold');
    [row col]=size(p);
    for x=1:1:row
        for y=1:1:col
            if((p(x,y)>lt)&&(p(x,y)<ut))
                p(x,y)=255;
            else
                p(x,y)=0;
            end
        end
    end

    figure(2);
    imshow(p);
    title('gray level slicing');
```

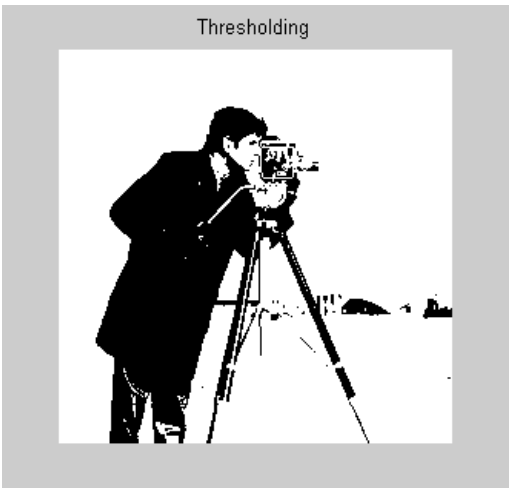
% Dynamic Range Compression

```
    clear all
    clc
    a=imread('cameraman.bmp');
    z=double(a);
    [row col]=size(z);
    for x=1:1:row
        for y=1:1:col
            c(x,y)=z(x,y)*((-1)^(x+y));
        end
    end
    d=abs(fft2(c));
    dlog=log(1+d);
    subplot(1,2,1)
    colormap(gray)
    imagesc(c)
    subplot(1,2,2)
    colormap(gray)
```

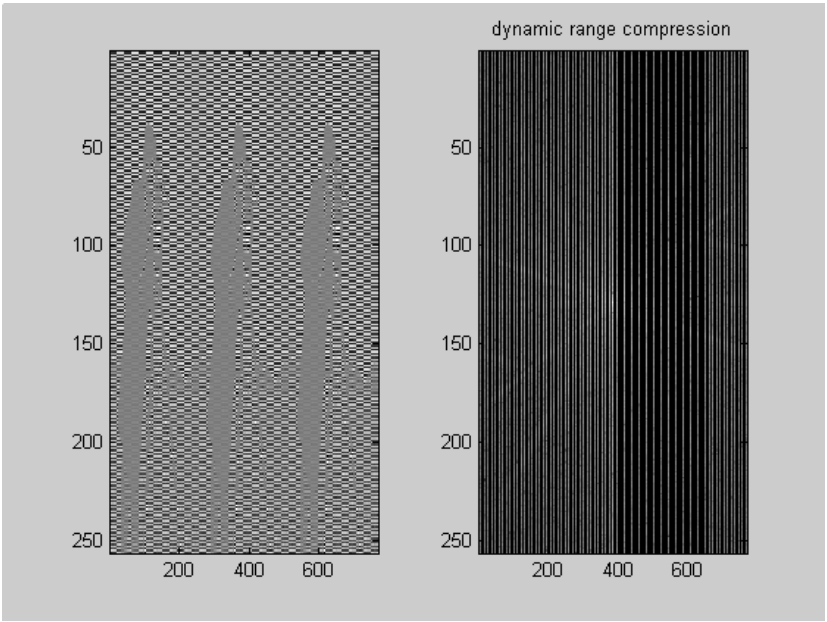
```
imagesc(dlog);
title('dynamic range compression');
```

Output: Point Processing Techniques






gray level slicing


dynamic range compression
50
100
150
200
250
200
400
600
50
100
150
200
250
200
400
600

Experiment Number 2

Problem Definition: Histogram Equalization

Software requirements: Matlab, C

Theory:

Histogram equalization

There are many applications where flat histogram is required. A perfect image is the one which has equal number of pixels in all its gray levels. The objective is to spread dynamic range and also have equal number of pixels in all gray levels. This is achieved using histogram equalization.

- Consider a continuous function, with r being the gray levels of the image to be enhanced
- Let r be normalized to the interval [0,1] such that r = 0 represents black and r = 1 represents white
- Consider the transformation of the form s = T(r) , 0 <= r <= 1 to produce a level s for every pixel value r in the original image
- Let T(r) satisfy the following conditions

  1. T(r) is single-valued and monotonically increasing in the interval 0 <= r <= 1

  2. 0 <= T(r) <= 1 for 0 <= r <= 1
- Single-valued requirement guarantees the existence of inverse transform
- Monotonicity condition preserves the increasing order of black to white in output image
- Second condition guarantees that gray levels are preserved in output image
- Inverse transform from s back to r is denoted as $r = T^{-1}(s)$ 0 <= s <= 1
- Gray values in an image may be viewed as random variables in the interval [0, 1]
- Probability density function is used to describe a random variable
- Let $p_r(r)$ and $p_s(s)$ denote the probability density functions of random variables r and s, respectively
- For discrete values, we have

  $p_r(r_k) = n_k / n$ for $0 <= r_k <= 1$ and k = 0,1,2,……,L-1
- The discrete transformation function is given by

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} n_j / n$$

  $0 <= r_k <= 1$ and k = 0,1,2,…….,L-1
- Processed image obtained by mapping each pixel with level r_k into a corresponding pixel with level s_k

- The transformation mapping is called as Histogram equalization or histogram linearization

Conclusion:

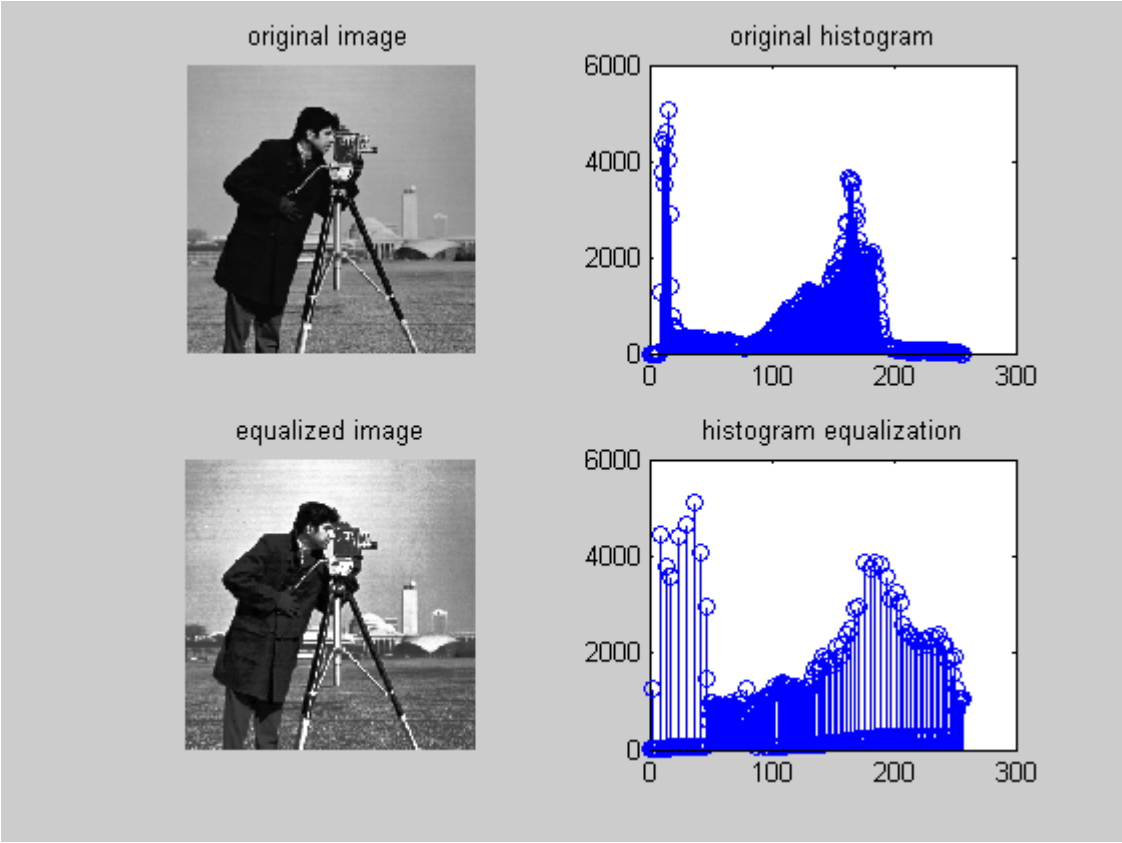
More dynamic range of gray level distribution. Fully automatic algorithm; no parameters need be specified. Simple to compute

Refer Pg. No. 113, 'Digital Image Processing', Gonzales, Woods, Pearson Education, Second Edition

Code**: Histogram Equalization**

```
p=imread('cameraman.bmp');
subplot(2,2,1);
imshow(p);
title('original image');
[m n]=size(p);
hist=zeros(1,256);
for i=1:m
   for j=1:n
      k=p(i,j);
      k=double(k);
      hist(k+1)=hist(k+1)+1;
   end
end
subplot(2,2,2);
stem(hist);
 title('original histogram');
for i=1:256
   hist(i)=hist(i)/(n*m);
end
for i=2:256
   hist(i)=(hist(i-1)+hist(i));
end
smin=hist(1);
for i=1:256
  hist(i)=round(((hist(i)-smin)*255)/(1-smin)+0.5);
end
  for i=1:m
   for j=1:n
      k=p(i,j);
      k=double(k);
     p(i,j)=hist(k+1);
   end
end
   for i=1:m
   for j=1:n
      k=p(i,j);
      k=double(k);
      hist(k+1)=hist(k+1)+1;
   end
end
subplot(2,2,3);
imshow(p);
 title('equalized image');
subplot(2,2,4);
stem(hist);
title('histogram equalization');
```

Output: Histogram Equalization



Experiment Number 3

Problem Definition: Low Pass Averaging Filter

Software requirements: Matlab, C

Theory:

Low pass filtering removes the high frequency content from the image. Mask used for low pass filtering has all coefficients positive. The standard low pass averaging 3 X 3 mask is

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Each element of the mask is the average value. A 5 X 5 or 7 X 7 mask can be used as per the requirement. Averaging filters are excellent when the image contains gaussian noise. It acheives filtering by blurring the noise. It removes the noise by blurring it till it is no longer seen.

Conclusion:

Low pass averaging filter, blurs the edges in the image. Bigger is the averaging mask more is the blurring.

Refer Pg. No. Refer Pg. No. 141, ‘Digital Image Processing’, Gonzales, Woods, Pearson Education, Second Edition

Code: Low Pass Averaging Filter

```
clear all;
clc;
aa=imread('cameraman.bmp');
a=double(aa);
[row col]=size(a);
m=input('Enter size of mask:');
n=m^2;

for i=1:1:m
for j=1:1:m
w(i,j)=1/n;
end
end
s=(m+1)/2;
for x=1:1:row
for y=1:1:col
        b(x,y)=a(x,y);
end
end
for x=s:1:row-s
for y=s:1:col-s
        b(x,y)=0;
end
end
for x=s:1:row-s
for y=s:1:col-s
        for i=1:1:m
        for j=1:1:m
        b(x,y)=a(x-s+i,y-s+j)*w(i,j)+b(x,y);
        end
        end
end
end
figure(1)
imshow(uint8(a))
figure(2)
imshow(uint8(b))
title('Low pass average filtered image');
c=conv2(a,w);
imshow(uint8(c))
```

Output: Low Pass Averaging Filter

Experiment Number 4

Problem Definition: Sharpening Filters

Software requirements: Matlab, C

Theory:

High Pass Filter

Highpass filtering eliminates the low frequency regions while retaining or enhancing the high frequency components. An image which is high passed, would have no background and would have enhanced edges. Hence high pass filters are used to sharpen blurred images.

The mask co-efficients have to be such that they have a positive value at the centre and negative values at periphery. Sum of coefficients of high pass mask has to be equal to zero. This is because, when we place this mask over the low frequency regions, the result should be zero.

The standard 3 X 3 is given by

-1	-1	-1
-1	8	-1
-1	-1	-1

High Boost Filter

High pass filter gets rid of the complete background. But sometimes we need to enhance the edges but also retain the background. For this a modified version of high pass filter known as high boost filtering is used.

In high boost filtering, we pass some background along with high frequency content.

High pass= Original-Low pass

To pass some of the background, we multiply original image with multiplicative factor A.

High Boost=(A)Original - Low pass

=(A-1)Original + High pass

If A>1, then some of original signal is added back to high pass result.

High Boost mask can be given as

1/9

-1	-1	-1
-1	X	-1
-1	-1	-1

If A=1.1, X=8.9

Conclusion:

When we do a high pass operation, the background which is low frequency gets eliminated while edges get enhanced. High boost filtering does not eliminate the background completely.

Refer Pg. No. Refer Pg. No. 147, ‘Digital Image Processing’, Gonzales, Woods, Pearson Education, Second Edition

Code: Sharpening Filters

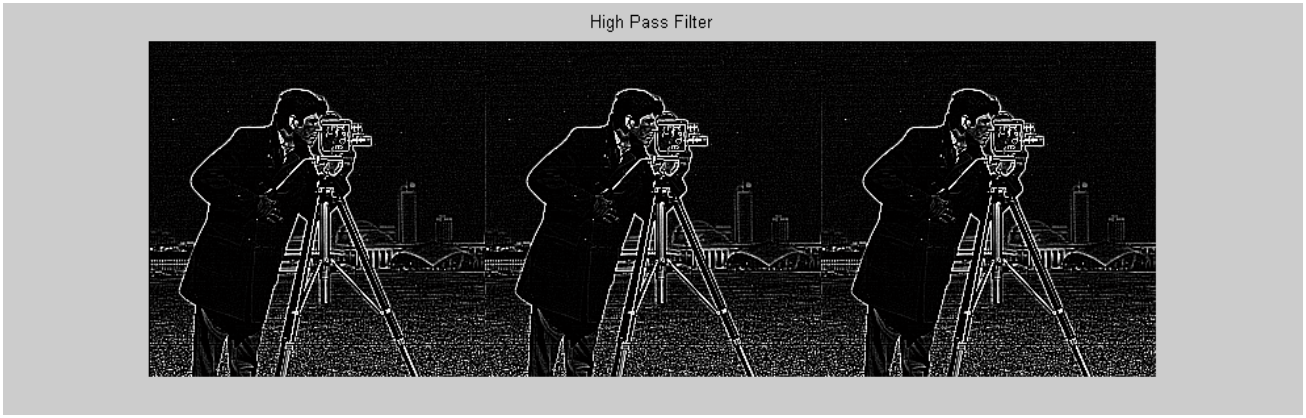
% High Pass Filter

```
clear all
clc
f=imread('cameraman.bmp ');
a=double(f);
[row col]=size(a);
w=[-1 -1 -1; -1 8 -1; -1 -1 -1]
for x=2:1:row-1
for y=2:1:col-1
a1(x,y)=w(1)*a(x-1,y-1)+w(2)*a(x-1,y)+w(3)*a(x-1,y+1)+w(4)*a(x,y-
1)+w(5)*a(x,y)+w(6)*a(x,y+1)+w(7)*a(x+1,y-1)+w(8)*a(x+1,y)+w(9)*a(x+1,y+1);
end
end
figure(1)
imshow(uint8(a))
figure(2)
imshow(uint8(a1))
title('high pass filter');
```

% High Boost Filter

```
clear all
clc
f=imread('cameraman.bmp ');
a=double(f);
[row col]=size(a);
w=[-1 -1 -1; -1 8 -1; -1 -1 -1]
for x=2:1:row-1
   for y=2:1:col-1
        a1(x,y)=w(1)*a(x-1,y-1)+w(2)*a(x-1,y)+w(3)*a(x-1,y+1)+w(4)*a(x,y-
        1)+w(5)*a(x,y)+w(6)*a(x,y+1)+w(7)*a(x+1,y-1)+w(8)*a(x+1,y)+w(9)*a(x+1,y+1);
   end
end
figure(1)
imshow(uint8(a))
figure(2)
imshow(uint8(a1))
title('High Pass Filter');
```

Ouput: Sharpening Filters



Experiment Number 5

Problem Definition: Edge detection: Roberts, Prewitts and Sobels operators

Software requirements: Matlab, C

Theory:

Edge detection is carried out by derivative approach. The basic idea of derivative approach is the computation of local derivative operator.

Roberts Operators

Here cross differences are taken instead of straight differences. These are 2 X 2 masks, so they are difficult to implement

Roberts masks= |Z5 – Z9| + |Z6-Z8|

1	0
0	-1

0	1
-1	0

Prewitts Operators

Prewitts masks = |(Z7 + Z8 + Z9) – (Z1+ Z2 + Z3)| + |(Z3 + Z6 + Z9) – (Z1 + Z4 + Z7)|

Similar weights are assigned to all neighbors of candidate pixels whose edge strength is being calculated.

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

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

Sobels Operators

Sobels masks = |(Z7 + 2Z8 + Z9) – (Z1+ 2Z2 + Z3)| + |(Z3 + 2Z6 + Z9) – (Z1 + 2Z4 + Z7)|

Higher weights are assigned to the pixels close to the candidate pixels.

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

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

Conclusion:

Sum of the coefficients each of the mask is zero. Derivative filters calculate gradient of the image. Both Prewitts and Sobels masks provide smoothing effect along with providing differentiation.

Refer Pg. No. Refer Pg. No. 594, ‘Digital Image Processing’, Gonzales, Woods, Pearson Education, Second Edition

Code: Edge detection: Roberts, Prewitts & Sobels operators

%Roberts Operator

```
clear all
clc
aa=imread('cameraman.bmp');
a=double(aa);
[row col]= size(a);
w1=[1 0; 0 -1];
w2=[0 1; -1 0];
w3=[1 1; -1 -1];
for x = 2:1:row-1;
   for y = 2:1:col-1;
      a1(x,y) = [w1(1)*a(x,y)+w1(2)*a(x,y+1)+w1(3)*a(x+1,y)+w1(4)*a(x+1,y+1)];
      a2(x,y) = [w2(1)*a(x,y)+w2(2)*a(x,y+1)+w2(3)*a(x+1,y)+w2(4)*a(x+1,y+1)];
      a3(x,y) = [w3(1)*a(x,y)+w3(2)*a(x,y+1)+w3(3)*a(x+1,y)+w3(4)*a(x+1,y+1)];
   end
end

subplot(4,3,2);imshow(uint8(a));xlabel('ORIGINAL');
subplot(4,3,4);imshow(uint8(a1));xlabel('X Gradient Image');
subplot(4,3,5);imshow(uint8(a2));xlabel('Y Gradient Image');
subplot(4,3,6);imshow(uint8(a3));xlabel('Final Image- roberts');
```

%Sobel Operator

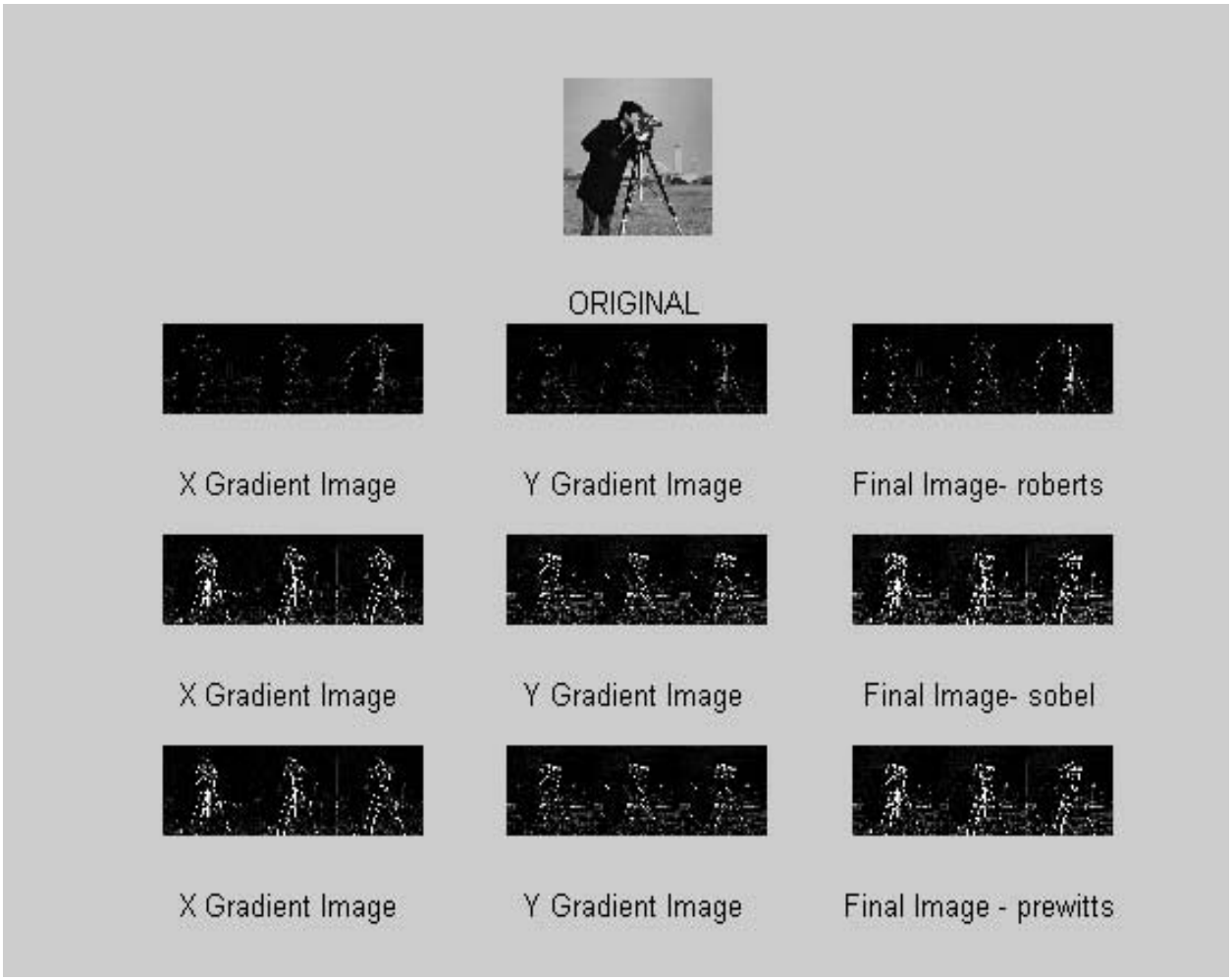
```
w1=[-1 -2 -1;  0 0 0;  1 2 1];
w2=[-1  0  1; -2 0 2; -1 0 1];
w3=[-2 -2  0; -2 0 2;  0 2 2];
for x = 2:1:row-1;
   for y = 2:1:col-1;
        a1(x,y) = [w1(1) * a(x-1,y-1) + w1(2) * a(x-1,y)+w1(3) * a(x-1,y+1) + w1(4) * a(x,y-
        1)+ w1(5)*a(x,y)+w1(6)* a(x,y+1)+w1(7) * a(x+1,y-1) + w1(8) * a(x+1,y)+w1(9) *
        a(x+1,y+1)];
      a2(x,y) = [w2(1) * a(x-1,y-1) + w2(2) * a(x-1,y)+w2(3) * a(x-1,y+1) + w2(4) * a(x,y-
      1)+ w2(5)*a(x,y)+w2(6)* a(x,y+1)+w2(7) * a(x+1,y-1) + w2(8) * a(x+1,y)+w2(9) *
      a(x+1,y+1)];
      a3(x,y) = [w3(1) * a(x-1,y-1) + w3(2) * a(x-1,y)+w3(3) * a(x-1,y+1) + w3(4) * a(x,y-
      1)+ w3(5)*a(x,y)+w3(6)* a(x,y+1)+w3(7) * a(x+1,y-1) + w3(8) * a(x+1,y)+w3(9) *
      a(x+1,y+1)];
   end
end
subplot(4,3,7);imshow(uint8(a1));xlabel('X Gradient Image');
subplot(4,3,8);imshow(uint8(a2));xlabel('Y Gradient Image');
subplot(4,3,9);imshow(uint8(a3));xlabel('Final Image- sobel');
```

%Prewitts Operator

```
w2=[-1 0 1; -1 0 1; -1 0 1];
w1=[-1 -1 -1; 0 0 0; 1 1 1];
w3=[-2 -1 0;-1 0 1;0 1 2];
for x = 2:1:row-1;
```

```
    for y = 2:1:col-1;
        a1(x,y) = [w1(1) * a(x-1,y-1) + w1(2) * a(x-1,y)+w1(3) * a(x-1,y+1) + w1(4) * a(x,y-
        1)+ w1(5)*a(x,y)+w1(6)* a(x,y+1)+w1(7) * a(x+1,y-1) + w1(8) * a(x+1,y)+w1(9) *
        a(x+1,y+1)];
      a2(x,y) = [w2(1) * a(x-1,y-1) + w2(2) * a(x-1,y)+w2(3) * a(x-1,y+1) + w2(4) * a(x,y-
      1)+ w2(5)*a(x,y)+w2(6)* a(x,y+1)+w2(7) * a(x+1,y-1) + w2(8) * a(x+1,y)+w2(9) *
      a(x+1,y+1)];
      a3(x,y) = [w3(1) * a(x-1,y-1) + w3(2) * a(x-1,y)+w3(3) * a(x-1,y+1) + w3(4) * a(x,y-
      1)+ w3(5)*a(x,y)+w3(6)* a(x,y+1)+w3(7) * a(x+1,y-1) + w3(8) * a(x+1,y)+w3(9) *
      a(x+1,y+1)];
    end
end
subplot(4,3,10);imshow(uint8(a1));xlabel('X Gradient Image');
subplot(4,3,11);imshow(uint8(a2));xlabel('Y Gradient Image');
subplot(4,3,12);imshow(uint8(a3));xlabel('Final Image - prewitts');
```

Output: Edge detection: Roberts, Prewitts & Sobels operators



Experiment Number 6

Problem Definition: Image Subtraction

Software requirements: Matlab, C

Theory:

The difference between two images f(x,y) and h(x,y), expressed as

$$g(x,y) = f(x,y) – h(x,y)$$

It is obtained by computing the difference between all pairs of corresponding pixels from f and h. The key usefulness of subtraction is the enhancement of differences between images.

One of the most commercial uses of image subtraction is in the area of medical imaging called mask mode radiography. In this case h(x,y), the mask is an X ray image of a region of a patients body captured by an intensified TV camera. The procedure consists of injecting contrast medium into patient,s bloodstream, taking a series of images of same anatomical region as h(x,y), and subtracting this mask from the series of incoming images after injection of the contrast medium. The difference between f(x,y) and h(x,y) appear in the output image as enhanced detail.

Conclusion:

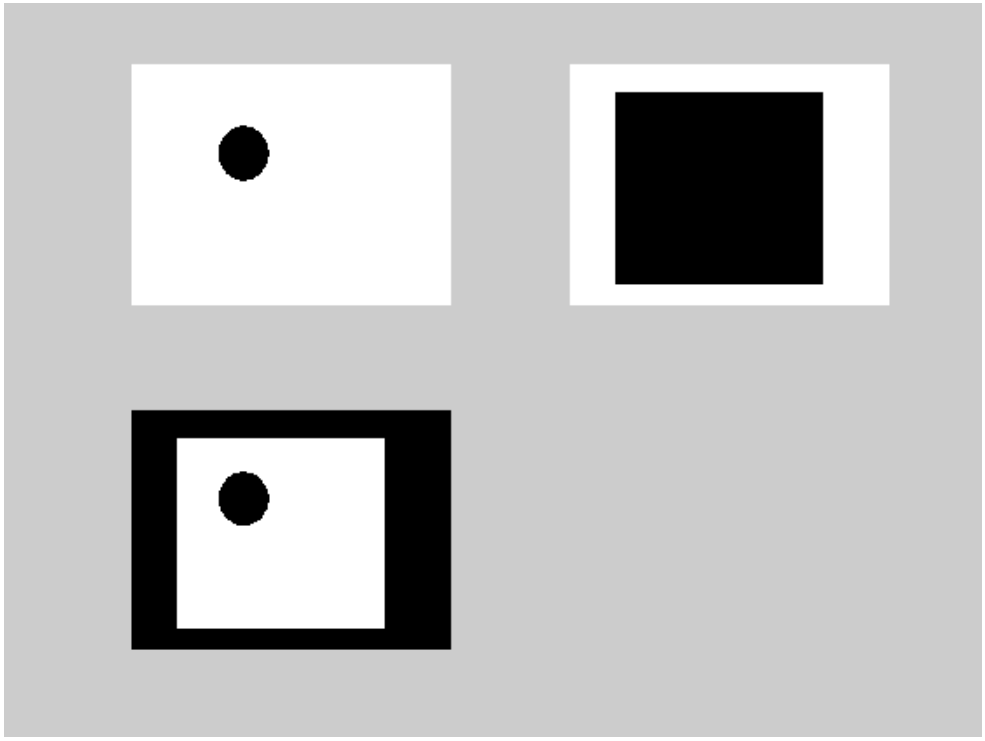
Image subtraction enhances the differences between images. These differences appear as enhanced detail.

Refer Pg. No. Refer Pg. No. 132, ‘Digital Image Processing’, Gonzales, Woods, Pearson Education, Second Edition

Code: Image Subtraction

```
clear all
clc
a=imread('im1.bmp');
z=double(a);
b=imread('imnew.bmp');
x=double(b);
        c=z-x;
subplot(2,2,1)
imshow(uint8(z))
subplot(2,2,2)
imshow(uint8(x))
subplot(2,2,3)
imshow(uint8(c))
```

Output: Image Subtraction



Experiment Number 7

Problem Definition: Dilation and Erosion

Software requirements: Matlab, C

Theory:

Dilation and Erosion

Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. By choosing the size and shape of the neighborhood, you can construct a morphological operation that is sensitive to specific shapes in the input image.

The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the *structuring element* used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The rule used to process the pixels defines the operation as a dilation or an erosion.

Rules for Dilation and Erosion

Operation	Rule
Dilation	The value of the output pixel is the *maximum* value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to the value 1, the output pixel is set to 1.
Erosion	The value of the output pixel is the *minimum* value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to 0, the output pixel is set to 0.

Processing Pixels at Image Borders (Padding Behavior)

Morphological functions position the origin of the structuring element, its center element, over the pixel of interest in the input image. For pixels at the edge of an image, parts of the neighborhood defined by the structuring element can extend past the border of the image.

To process border pixels, the morphological functions assign a value to these undefined pixels, as if the functions had padded the image with additional rows and columns. The value of these padding pixels varies for dilation and erosion operations. The following table describes the padding rules for dilation and erosion for both binary and grayscale images.

Rules for Padding Images

Operation	Rule
Dilation	Pixels beyond the image border are assigned the minimum value afforded by the data type.

Operation	Rule
	For binary images, these pixels are assumed to be set to 0. For grayscale images, the minimum value for uint8 images is 0.
Erosion	Pixels beyond the image border are assigned the *maximum* value afforded by the data type. For binary images, these pixels are assumed to be set to 1. For grayscale images, the maximum value for uint8 images is 255.

Understanding Structuring Elements

An essential part of the dilation and erosion operations is the structuring element used to probe the input image. A structuring element is a matrix consisting of only 0's and 1's that can have any arbitrary shape and size. The pixels with values of 1 define the neighborhood.

Two-dimensional, or *flat*, structuring elements are typically much smaller than the image being processed. The center pixel of the structuring element, called the *origin*, identifies the pixel of interest -- the pixel being processed. The pixels in the structuring element containing 1's define the *neighborhood* of the structuring element. These pixels are also considered in dilation or erosion processing.

Conclusion:

Dilation adds pixels to the boundaries of objects in an image and erosion reduces the number of pixels from the boundary of image. The number of pixels added or removed depends upon the size of structuring element.

Refer Pg. No. Refer Pg. No. 545, ‘Digital Image Processing’, Gonzales, Woods, Pearson Education, Second Edition

Code: Dilation and Erosion

```
clear all
a=zeros(9,10);
a(4:6,4:7)=1;
w=[1 1 1; 1 1 1; 1 1 1];
for x=2:1:8
        for y=2:1:9
                a1=[w(1)*a(x-1,y-1)       w(2)*a(x-1,y)       w(3)*a(x-1,y+1)       w(4)*a(x,y-1)
                w(5)*a(x,y) w(6)*a(x,y+1) w(7)*a(x+1,y) w(8)*a(x+1,y) w(9)*a(x+1,y+1)];
                A1(x,y)=max(a1);
                A2(x,y)=max(a1);
        end
end
display(a);
display(A1);
display(A2);
```

Output: Dilation and Erosion

a =

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

Dilation

A1 =

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

Erosion

A2 =

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

Experiment Number 8

Problem Definition: Boundary Extraction

Software: Matlab, C

Theory:

Given a binary image it is fairly simple to extract the boundary of the image.

If A is the image and B is the structuring element, then boundary extraction can be achieved using the formula:

Boundary(A)= A - (A θ B)

It simply means subtracting the eroded image of A from the original image.

If B =

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

Then A θ B would be same as A except one pixel less from all the sides. Hence A - (A θ B) would give us one pixel difference.

Conclusion:

Basic morphological operation used to extract boundary in binary images.

Refer Pg. No. Refer Pg. No. 556, 'Digital Image Processing', Gonzales, Woods, Pearson Education, Second Edition

Code: Boundary Extraction

```
clear all
a=zeros(10,10);
a(3:8,3:8)=1;
w=[1 1 1; 1 1 1; 1 1 1];
for x=2:1:9
   for y=2:1:9
                a1=[w(1)*a(x-1,y-1)      w(2)*a(x-1,y)      w(3)*a(x-1,y+1)      w(4)*a(x,y-1)
                w(5)*a(x,y) w(6)*a(x,y+1) w(7)*a(x+1,y) w(8)*a(x+1,y) w(9)*a(x+1,y+1)];
                A(x,y)=min(a1);
                 b(x,y)=a(x,y)-A(x,y)
         end
end
display(a);
display(A);
display(b);
```

Output: Boundary Extraction

a =

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

Eroded Image

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

Extracted Boundary

b =

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

Experiment Number 9

Problem Definition: Discrete Cosine Transform

Software requirements: Matlab, C

Theory:

A discrete cosine transform (DCT) expresses a sequence of finitely many data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of audio and images (where small high-frequency components can be discarded), to spectral methods for the numerical solution of partial differential equations. The use of cosine rather than sine functions is critical in these applications: for compression, it turns out that cosine functions are much more efficient (as explained below, fewer are needed to approximate a typical signal), whereas for differential equations the cosines express a particular choice of boundary conditions.

In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers.

Conclusion:

Discrete Cosine Transform uses only cosine waves to represent a signal. So DCT is purely real.

Refer Pg. No. 150, Anil K. Jain, ‘ Fundamentals of Digital Image Processing’, PHI Publication

Code: Discrete Cosine Transform

```
clear all
a=[2 4 4 2 ; 4 6 8 3 ; 2 8 10 4 ; 3 8 6 2];
[row col]=size(a);
const=sqrt(2/col);
for n=0:1:col-1
   for k=0:1:col-1
      if n==0
         c(n+1,k+1)=1/sqrt(col);
      else
         b=2*k;
         c(n+1, k+1)=const*cos((pi*(b+1)*n)/(2*col));
      end
   end
end
display(a);
X=c*a*c.';
display(X);
```

Output: Discrete Cosine Transform

a =

2 4 4 2

4 6 8 3

2 8 10 4

3 8 6 2

X =

19.0000 -0.2706 -8.0000 0.6533

-2.6924 -0.2500 2.3097 0.8964

-3.5000 1.4651 1.5000 -1.6892

0.0328 -1.6036 -0.9567 -0.2500

Experiment Number 10

Problem Definition: Discrete Fourier Transform

Software requirements: Matlab, C

Theory:

Discrete Fourier transform (DFT) is a specific kind of Fourier transform, used in Fourier analysis. It transforms one function into another, which is called the frequency domain representation, or simply the *DFT*, of the original function. But the DFT requires an input function that is discrete and whose non-zero values have a limited (*finite*) duration.

The input to the DFT is a finite sequence of real or complex numbers (with more abstract generalizations discussed below), making the DFT ideal for processing information stored in computers. In particular, the DFT is widely employed in signal processing and related fields to analyze the frequencies contained in a sampled signal, to solve partial differential equations, and to perform other operations such as convolutions or multiplying large integers.

The sequence of N complex numbers $x_0, ..., x_{N-1}$ is transformed into the sequence of N complex numbers $X_0, ..., X_{N-1}$ by the DFT according to the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

where i is the imaginary unit

The inverse discrete Fourier transform (IDFT) is given by

$$x_n - \frac{1}{N} \sum_{k-0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n - 0, \ldots, N-1.$$

Conclusion:

Discrete Fourier Transform uses sine and cosine waves to represent a signal. So DFT has purely real and imaginary terms.

Refer Pg. No. 145, Anil K. Jain, ‘ Fundamentals of Digital Image Processing’, PHI Publication

Code: Discrete Fourier Transform

```
clear all
a=[1 2 3 4 ; 2 3 4 5 ; 2 2 2 2 ; 1 1 1 1];
[row col]=size(a);
const=sqrt(row*col);
for n=0:1:row-1
   for k=0:1:col-1
   W(n+1,k+1)=exp(-i*2*pi*n*k/const);
   end
end
display(a);
X=W*a*W.';
display(X);
```

Output: Discrete Fourier Transform

a =

1	2	3	4
2	3	4	5
2	2	2	2
1	1	1	1

X =

36.0000	-4.0000 + 4.0000i	-4.0000 - 0.0000i	-4.0000 - 4.0000i
2.0000 -10.0000i	-0.0000 + 4.0000i	-2.0000 + 2.0000i	-4.0000 - 0.0000i
0 - 0.0000i	0.0000 + 0.0000i	-0.0000 + 0.0000i	-0.0000 + 0.0000i
2.0000 +10.0000i	-4.0000 - 0.0000i	-2.0000 - 2.0000i	0.0000 - 4.0000i

Experiment Number 11

Problem Definition: Linear and Circular Convolution

Software: Matlab, C

Theory:

Convolution is an integral concatenation of two signals. It has many applications in numerous areas of signal processing. The most popular application is the determination of the output signal of a linear time-invariant system by convolving the input signal with the impulse response of the system. Convolving two signals is equivalent to multiplying the Fourier transform of the two signals.

For discrete time signals $x(n)$ and $h(n)$, the integration is replaced by a summation

$$y(n) = x(n) \circledast h(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$$

Conclusion:

To determine the response of any relaxed, discrete time, linear time invariant system to any input signal, convolution sum can be used

Code: Linear Convolution

```
FUNCTION conv(x,h)
close all
clear all
x=input('Enter x:   ')
h=input('Enter h:   ')
m=length(x);
n=length(h);
X=[x,zeros(1,n)];
H=[h,zeros(1,m)];
for i=1:n+m-1
Y(i)=0;
for j=1:m
if(i-j+1>0)
Y(i)=Y(i)+X(j)*H(i-j+1);
else
end
end
end
Y
stem(Y);
ylabel('Y[n]');
xlabel('----->n');
```

Code: Circular Convolution

```
function y=circular_convolution(x,h)

x= input ('enter the first sequence');

h= input ('enter the second sequence');

N1= length(x);

N2= length(h);

N=max(N1,N2);%length of sequence

x=[x zeros(1,N-N1)];  %modified first sequence

h=[h zeros(1,N-N2)];  %modified second sequence

for n=0:N-1;

   y(n+1)=0;

   for i=0:N-1

     j=mod(n-i,N);

     y(n+1)=y(n+1)+x(i+1)*h(j+1);  %shifting and adding
```

```
    end
end
%to display and plot circular convolution
n=1:N;
disp('output sequence of circular convolution');
disp(y);%to view output in command window
pause;
stem(n,y);%plotting circular convolution
grid minor;
xlabel('time index');
ylabel('amplitude');
title('circular convolution sequence of x and h');
```

Experiment Number 12

Problem Definition: Write a program to implement Run Length Encoding

Software: Turbo C

Theory:

- Compression
  Compression is the process of encoding information using fewer bits than an unencoded representation would use, through use of specific encoding schemes.

- Types of compression
  Broadly classified as lossless and lossy compression.
  Lossless compression schemes are reversible so that the original data can be reconstructed, while lossy schemes accept some loss of data in order to achieve higher compression.

- Run length encoding
  Run-length encoding (RLE) is a very simple form of data compression in which runs of data are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, relatively simple graphic images, line drawings, and animations. It is not useful with files that don't have many runs as it could potentially double the file size.

- Example of RLE
  For example, consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. Let us take a hypothetical single scan line, with B representing a black pixel and W representing white:

  WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWW
  WWWWWWWWWWWWBWWWWWWWWWWWWW

  If we apply the run-length encoding (RLE) data compression algorithm to the above hypothetical scan line, we get the following:

  12W1B12W3B24W1B14W

  Interpret this as twelve W's, one B, twelve W's, three B's, etc.

Algorithm:

1. Set the previous symbol equal to an unmatchable value.
2. Read the next symbol from the input stream.
3. If the symbol is an EOF exit.
4. Write out the current symbol.
5. If the symbol is an does not match the previous symbol, set the previous symbol to the current symbol, and go to step 2.
6. Read and count additional symbols until a non-matching symbol is found. This is the run length.
7. Write out the run length.
8. Write out the non-matching symbol.

9. Set the previous symbol to the non-matching symbol, and go to step 2.

Conclusion:

RLE is a lossless compression algorithm that only offers decent compression ratios in specific types of data.

Code: Write a program to implement Run length encoding

```
#include<stdio.h>
#include<conio.h>
void value();
void disp();
void rle();
int a[100][100],b[100][100],m,n,i,j;
void main()
{
	clrscr();
	printf("\nEnter the size of image(m x n):\n");
	printf("m:");
	scanf("%d",&m);
	printf("\nn:");
	scanf("%d",&n);
	value(m,n);
	disp(m,n,a);
	for(i=0;i<n;i++)
		for(j=0;j<m;j++)
			a[i][j]=a[i][j]+1;
	printf("\nRLE CODE:\n");
	rle(m,n,a);
	printf("\n\n");
	getch();
}
void rle(int m,int n,int a[100][100])
{
	int c;
	for(j=0;j<n;j++)
	{
		printf("\n");
		c=1;
		for(i=0;i<m;i++)
		{
			if(a[j][i]==a[j][i+1])
			{
				c++;
			}
			else
			{
				printf("(0X%.2x)",c);
				printf("(0X%.2d)",(a[j][i])-1);
				c=1;
			}

		}
	}
}

void value(int m,int n)
{
 for(i=0;i<n;i++)
```

```
        {
                printf("\nEnter %dth row\n",i);
                for(j=0;j<m;j++)
                {
                        printf("\t\t");
                        scanf("%d",&a[i][j]);
                        if((a[i][j]!=0) && (a[i][j]!=1))
                        {
                                printf("\n\nEnter correct values.Enter values again.");
                                printf("\n\n-----------------------------------------\n\n");
                                value(m,n);
                        }
                }
        }
}

void disp(int m,int n,int a[100][100])
{
        printf("\n\n\nImage :\n\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                        printf("%d  ",a[i][j]);
        printf("\n");
        }
}
```

Output: Write a program to implement a Run length encoding

Enter the size of image(m x n):
m:20
n:3

Enter 0th row
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

Enter 1th row

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

Enter 2th row

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

Image :

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

RLE CODE:

(0X12)(0X01)(0X02)(0X00)

(0X10)(0X00)(0X04)(0X01)

(0X05)(0X01)(0X0b)(0X00)(0X04)(0X01)

Experiment Number 13

Problem Definition: Segmentation using Region Growing

Software: Turbo C/ Java

Theory: Segmentation refers to the process of partitioning a digital image into multiple segments (sets of pixels, also known as superpixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.

The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s).

Algorithm:

1) Select a seed point (a start pixel).
2) Select its nearest neighbours (4 or 8 neighbours) one by one and check if they satisfy homogeneity property of a region. ie if both pixel satisfy predefined condition.
3) Once a new pixel is accepted as member of current region, the neighbours of this new pixel are examined.
4) This procedure is recursively applied until no new pixel is accepted. All the pixels of current region are given a unique label. Now a new seed pixel is chosen and the same procedure is repeated.
5) We continue doing this until all the pixels are assigned to some region or other.

Code : To implement Region growing

```
function J=regiongrowing(I,x,y,reg_maxdist)
% This function performs "region growing" in an image from a specified
% seedpoint (x,y)
%
% J = regiongrowing(I,x,y,t)
%
% I : input image
% J : logical output image of region
% x,y : the position of the seedpoint (if not given uses function getpts)
% t : maximum intensity distance (defaults to 0.2)
%
% The region is iteratively grown by comparing all unallocated neighbouring pixels to the region.
% The difference between a pixel's intensity value and the region's mean,
% is used as a measure of similarity. The pixel with the smallest difference
% measured this way is allocated to the respective region.
% This process stops when the intensity difference between region mean and
% new pixel become larger than a certain treshold (t)
%
% Example:
%
% I = im2double(imread('medtest.png'));
% x=198; y=359;
% J = regiongrowing(I,x,y,0.2);
% figure, imshow(I+J);
%
% Author: D. Kroon, University of Twente


if(exist('reg_maxdist','var')==0), reg_maxdist=0.2; end
if(exist('y','var')==0), figure, imshow(I,[]); [y,x]=getpts; y=round(y(1)); x=round(x(1)); end
```

```
J = zeros(size(I)); % Output
Isizes = size(I); % Dimensions of input image

reg_mean = I(x,y); % The mean of the segmented region
reg_size = 1; % Number of pixels in region

% Free memory to store neighbours of the (segmented) region
neg_free = 10000; neg_pos=0;
neg_list = zeros(neg_free,3);

pixdist=0; % Distance of the region newest pixel to the regio mean

% Neighbor locations (footprint)
neigb=[-1 0; 1 0; 0 -1;0 1];

% Start regiogrowing until distance between regio and posible new pixels become
% higher than a certain treshold
while(pixdist<reg_maxdist&&reg_size<numel(I))

   % Add new neighbors pixels
   for j=1:4,
      % Calculate the neighbour coordinate
      xn = x +neigb(j,1); yn = y +neigb(j,2);

      % Check if neighbour is inside or outside the image
      ins=(xn>=1)&&(yn>=1)&&(xn<=Isizes(1))&&(yn<=Isizes(2));

      % Add neighbor if inside and not already part of the segmented area
      if(ins&&(J(xn,yn)==0))
            neg_pos = neg_pos+1;
            neg_list(neg_pos,:) = [xn yn I(xn,yn)]; J(xn,yn)=1;
```

```
    end
  end

  % Add a new block of free memory
  if(neg_pos+10>neg_free), neg_free=neg_free+10000; neg_list((neg_pos+1):neg_free,:)=0; end

  % Add pixel with intensity nearest to the mean of the region, to the region
  dist = abs(neg_list(1:neg_pos,3)-reg_mean);
  [pixdist, index] = min(dist);
  J(x,y)=2; reg_size=reg_size+1;

  % Calculate the new mean of the region
  reg_mean= (reg_mean*reg_size + neg_list(index,3))/(reg_size+1);

  % Save the x and y coordinates of the pixel (for the neighbour add proccess)
  x = neg_list(index,1); y = neg_list(index,2);

  % Remove the pixel from the neighbour (check) list
  neg_list(index,:)=neg_list(neg_pos,:); neg_pos=neg_pos-1;
end

% Return the segmented area as logical matrix
J=J>1;
```

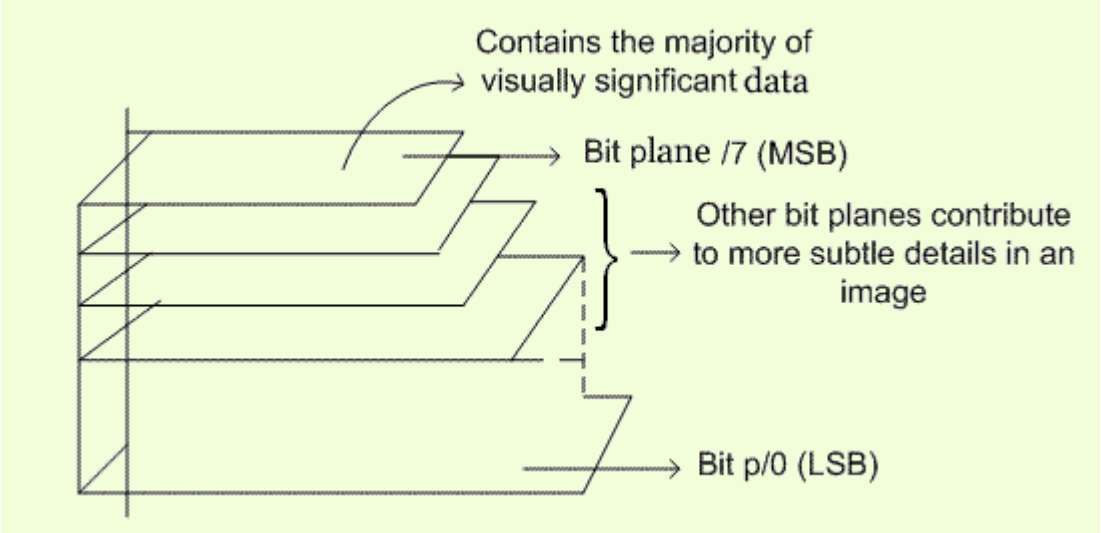
Experiment No 14

Problem Definition: Implement Digital Watermarking

Theory:

Highlighting the contribution made to total image appearance by specific bits might be desired. Suppose that each pixel in an image is represented by 8 bits. Imagine the image is composed of 8, 1-bit planes ranging from bit plane1-0 (LSB)to bit plane 7 (MSB).

In terms of 8-bits bytes, plane 0 contains all lowest order bits in the bytes comprising the pixels in the image and plane 7 contains all high order bits.



For watermarking, we replace few LSB planes of the original image with MSB planes of image to be watermarked.

Algorithm:

1) Start.
2) Read the original image on which image is to be watermarked and the image to be watermarked .
3) Perform bit slicing to extract every bit plane of the both images.
4) Replace few LSB planes of original image with MSB planes of other image.
5) Stop.