RAMNIRANJAN JHUNJHUNWALA COLLEGE
GHATKOPAR (W), MUMBAI - 400 086

DEPARTMENT OF INFORMATION TECHNOLOGY
2020 - 2021

M.Sc.(I.T.) SEM I
Image Vision and Processing

Name: Pratik Rampyare Rai
Roll No.: 23

Hindi Vidya Prachar Samiti's

RAMNIRANJAN JHUNJHUNWALA COLLEGE

(AUTONOMOUS)

Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai-400086

CERTIFICATE

This is to certify that Mr/Miss/Mrs. **Pratik Rampyare Rai** with **Seat No: 23** has successfully completed the necessary course of experiments in the subject of **Image Vision and Processing** during the academic year 2020 – 2021 complying with the requirements of RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE, for the course of M.Sc. (IT) semester -I.

Internal Examiner

Date : 12th March,2021

Head of Department

College Seal

External Examiner

INDEX

NO.	PRACTICAL	PAGE
1.	Implement Basic Intensity transformation functions A. A) Image Inverse B. B) Log Transformation C. C) Power-law Transformation	
2.	Piecewise Transformation A. A) Contrast Stretching B. B) Thresholding C. C) Bit-Plane Slicing	
3.	Implement Histogram Equalization	
4.	Image filtering in Spatial Domain A. A) Low-pass Filter/Smoothing Filters (Average, Weighted Average, Median and Gaussian) A. B) High-pass Filter / Sharpening Filter (Laplacian Filter, Sobel, Robert and Prewitt Filter to detect edge)	
5.	Color Image Processing A. A) Pseudocoloring B. B) Separating the RGB Channels C. C) Color Slicing	
6.	Image Compression Techniques and watermarking A) Implement Huffman Coding B) Add a watermark to the image	
7.	Basic Morphological Transformations A) Boundary Extraction B) Thinning and Thickening and Skeletons	

Practical 1

A. Implement Basic Intensity transformation functions - Image Inverse

The negative or inverse of an image with intensity levels in the range [0, L-1] is obtained by using the negative transformation, which is given by the expression,

S = L – 1 – r

Where L – 1 (Maximum pixel value)

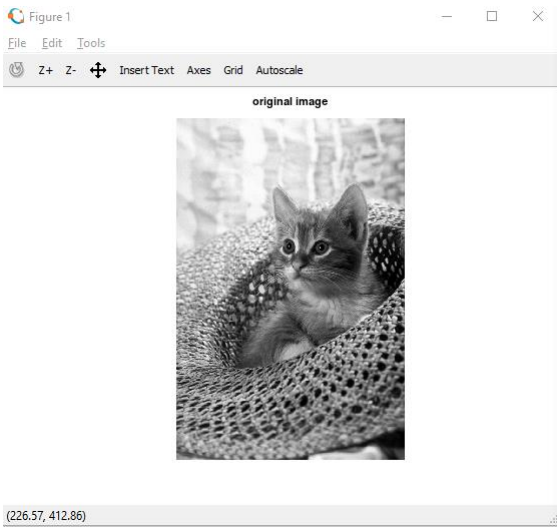
r (Pixel of an image)

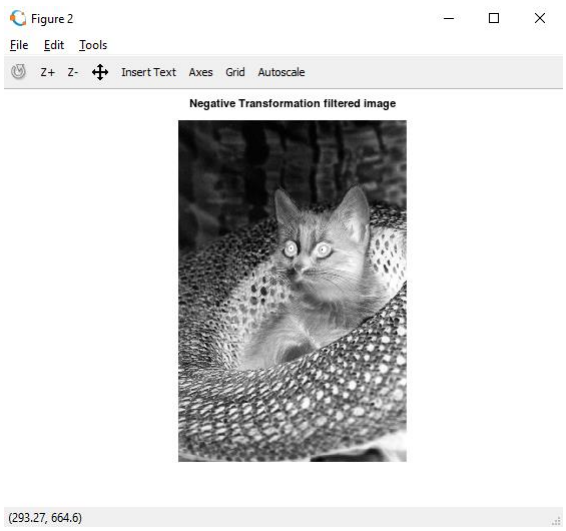
A negative image is a total inversion, in which light areas appear dark and vice versa.

Code:

```
#Practical 1 : Implement Basic Intensity transformation functions
#A) Image Inverse

pkg load image;
clear all;
close all;

img=imread('cat.png'); # "imread" is used to read the images
img=rgb2gray(img); # "rgb2gray" is used to convert the color img to gray img
img=im2double(img); # "im2double" converts the datatype to double
[row col] = size(img); # taking size of an image into matrix
for i = 1:row   #reading row value
 for j = 1:col #reading column value
   N(i,j) = 1-img(i,j);  #subs img matrix from 255 and storing in new Matrix(N)
 endfor
endfor
figure 1;
imshow(img);
title('original image');
figure 2;
imshow(N);
title('Negative Transformation filtered image');
```

Output:





B. Implement Basic Intensity transformation functions - Log Transformation

Log transformation means replacing each pixel value with its logarithm value.

The log transformations can be defined by this formula
s = c log(r + 1)
where, s and r are the pixel values of the output and the input image and c is a constant.

The value 1 is added to each of the pixel value of the input image because if there is a pixel intensity of 0 in the image, then log (0) is equal to infinity. So 1 is added, to make the minimum value at least 1.

For lower amplitudes of input image the range of gray levels is expanded.
For higher amplitudes of input image the range of gray levels is compressed

Code:

```
#Practical 1: Implement Basic Intensity transformation functions
#B) Log Transformation

pkg load image;
clear all;
close all;

pic=imread('pepperscolor.png'); # "imread" is used to read the images
pic=rgb2gray(pic); # "rgb2gray" is used to convert the colour img to gray img
pic=im2double(pic); # "im2double" converts the datatype to double
x=pic;
c=1; #Assigning constant value as 1

[row,col]=size(pic); # taking size of an image into matrix
for i=1:row   #reading row value
 for j=1:col  #reading column value
   x(i,j)=c*log(1+pic(i,j)); #Log transformation Formula
 endfor
endfor

#Subplot(m,n,p) fuction divides the current figure into an m -by- n grid and creates axes in the
position specified by p
subplot(2,2,1);
imhist(pic); title("Old Histogram"); #plots Histogram
subplot(2,2,2);
imhist(x); title("Histogram after transformation");

subplot(2,2,3);
imshow(pic); title("Before");
subplot(2,2,4);
imshow(x); title("After");
```

Output:

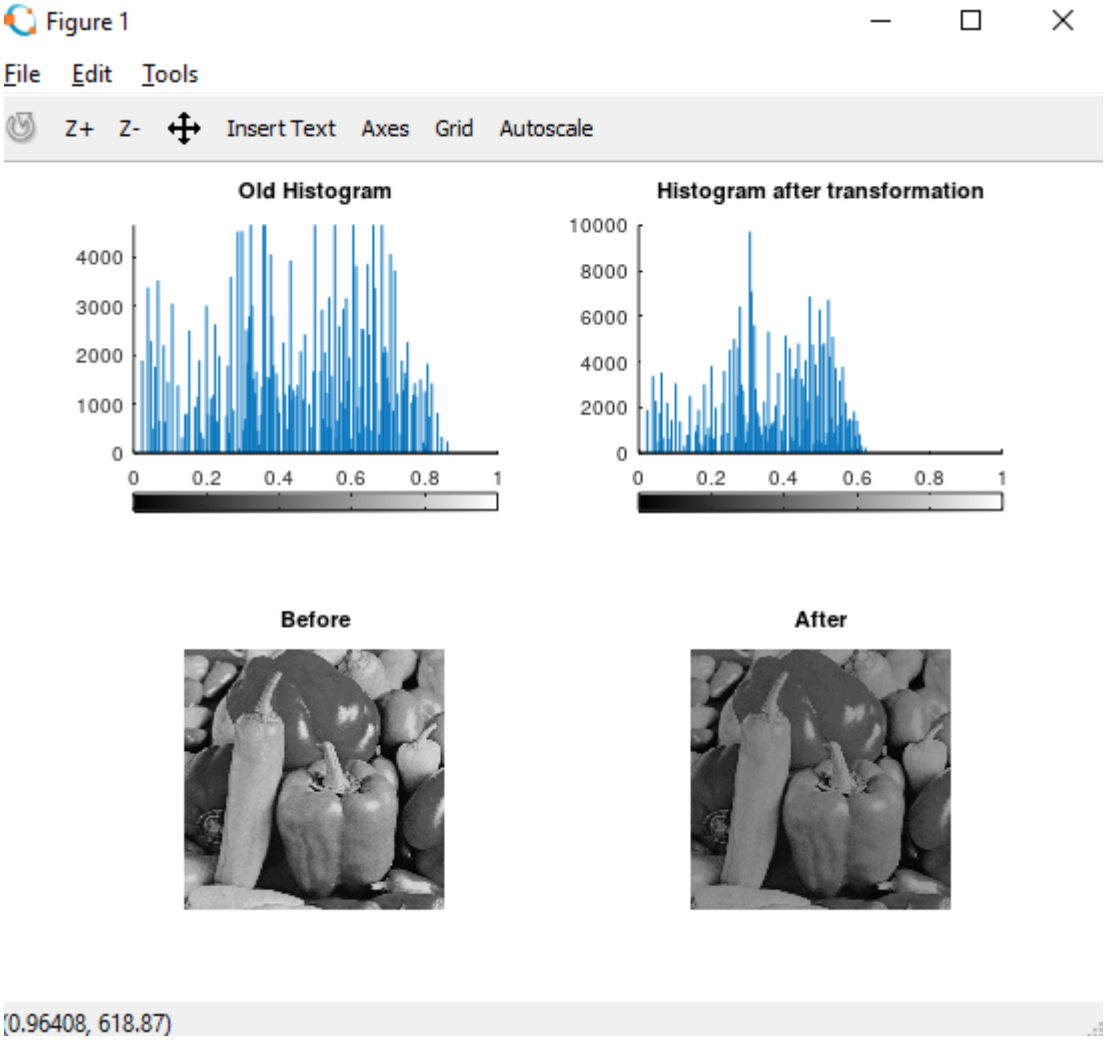


Figure 1
File Edit Tools
Z+ Z- Insert Text Axes Grid Autoscale
Old Histogram
Histogram after transformation
Before
After
(0.96408, 618.87)

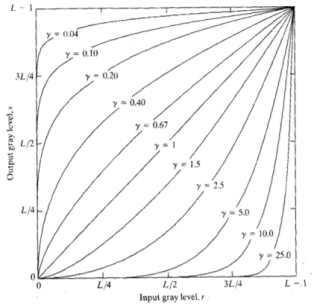
C. Implement Basic Intensity transformation functions – Power Law Transformation (Gamma)

The general form of Power law (Gamma) transformation function is

$$s = c*r^{\gamma}$$

Where, 's' and 'r' are the output and input pixel values respectively and 'c' and γ are the positive constants. Like log transformation, power law curves with γ <1 map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher input values. Similarly, for γ >1, we get the opposite result which is shown in the figure below

This is also known as gamma correction, gamma encoding or gamma compression.
The below curves are generated for r values normalized from 0 to 1 then multiplied by the scaling constant 'c' corresponding to the bit size used.



Code:

```
#Practical 1 : Implement Basic Intensity transformation functions
#C) Power Law Transformation

pkg load image;
clear all;
close all;

pic=imread('baby.jpg'); # "imread" is used to read the images
pic=rgb2gray(pic); # "rgb2gray" is used to convert the color img to gray img
pic=im2double(pic); # "im2double" converts the datatype to double
gamma=2; #taking constant value into "gamma" variable
c=1; #taking constant value into "c" variable

[row,col]=size(pic);  # taking size of an image into matrix
for i=1:row     #reading row value
  for j=1:col   #reading col value
    N(i,j)=c*(pic(i,j)^gamma); #power law transformation Formula
  endfor
endfor

figure 1;
imshow(pic);0
title('original Image');

figure 2;
imshow(N);
title('Gamma Transformation Image');
```

Output:

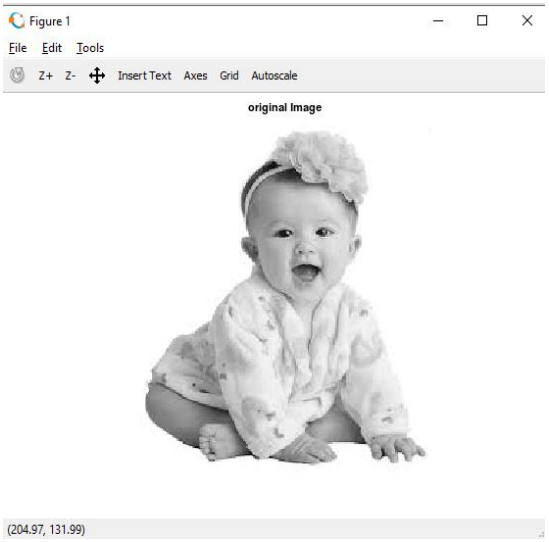


Figure 1
File Edit Tools
Z+ Z- Insert Text Axes Grid Autoscale
original Image
(204.97, 131.99)

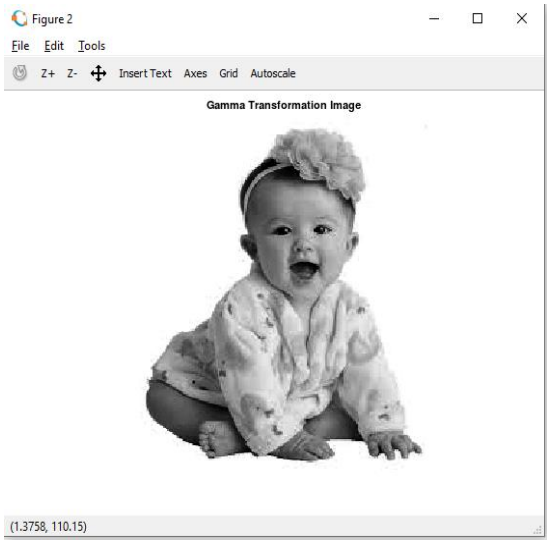


Figure 2
File Edit Tools
Z+ Z- Insert Text Axes Grid Autoscale
Gamma Transformation Image
(1.3758, 110.15)

Practical 2

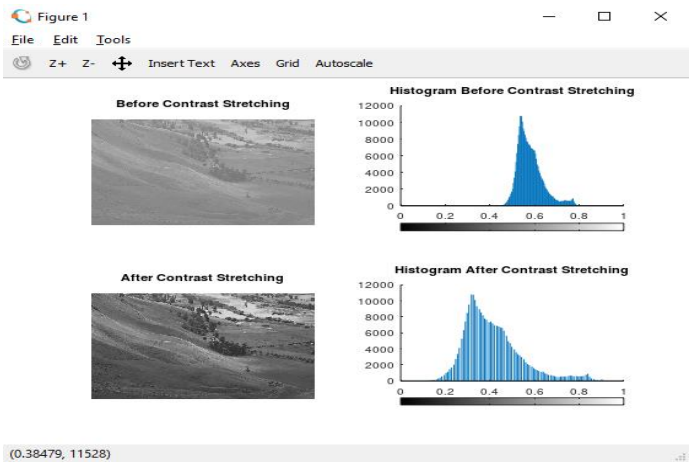
A. Piecewise transformation – Contrast Stretching(Using imadjust Function)

Contrast stretching (**often called normalization**) is a simple image enhancement technique that attempts to improve the contrast in an image by 'stretching' the range of intensity values it contains to span a desired range of values, the full range of pixel values that the image type concerned allows. Contrast stretching changes the distribution and range of the digital numbers assigned to each pixel in an image. This is normally done to accent image details that may be difficult for the human viewer to observe.

Code:

```
#Practical 2 : Piecewise transformation
#A) Contrast Stretching

pkg load image;
clear all;
close all;

pic=imread('fields.jpg'); # "imread" is used to read the images
pic=im2double(pic); # "im2double" converts the datatype to double

subplot(2,2,1);
imshow(pic);
title('Before Contrast Stretching');
subplot(2,2,2);
imhist(pic);
title('Histogram Before Contrast Stretching');

img=imadjust(pic,[0.44 0.8],[0.1 0.9]); #Contrast stretching using imadjust function

subplot(2,2,3);
imshow(img);
title('After Contrast Stretching');
subplot(2,2,4);
imhist(img);
title('Histogram After Contrast Stretching');
```

Output:



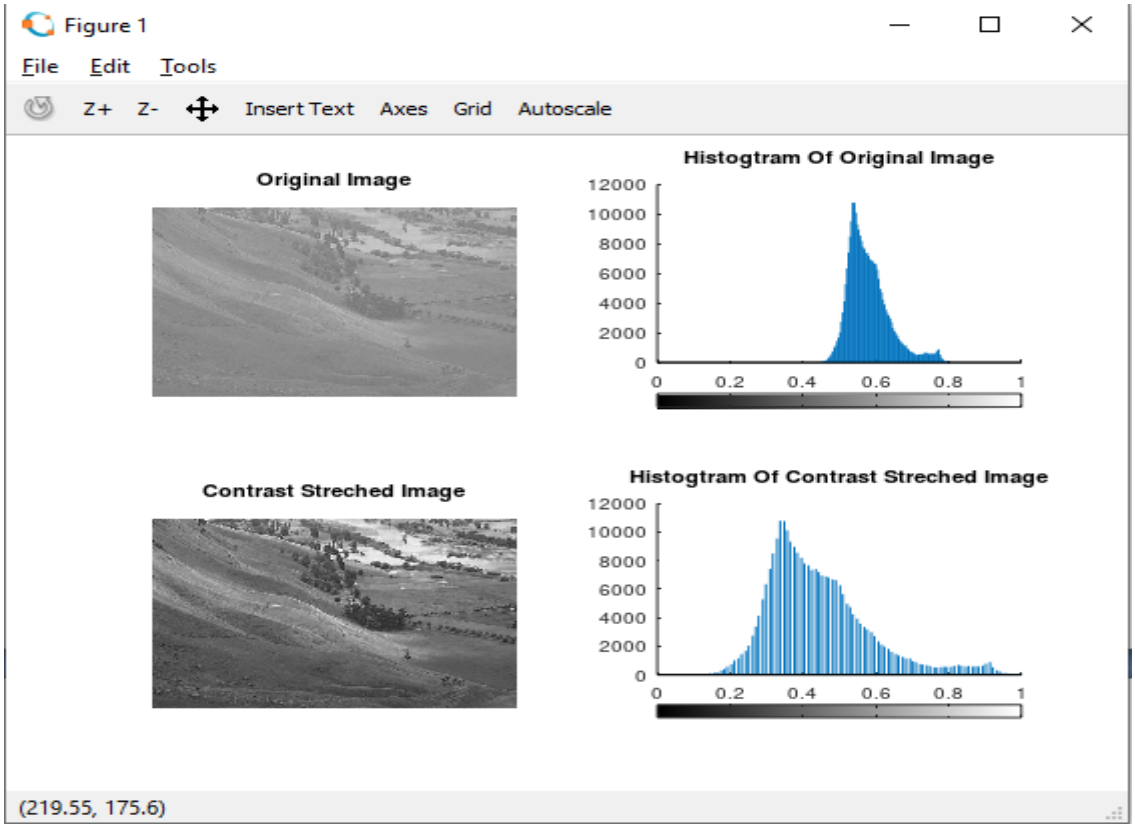
A. Piecewise transformation – Contrast Stretching(Using inputs from user)

Code:

```
#Practical 2 : Piecewise transformation
#A) Contrast Stretching using inputs from user

pkg load image;
clear all;
close all;

pic=imread('fields.jpg'); # "imread" is used to read the images
pic=im2double(pic); # "im2double" converts the datatype to double
[row col]=size(pic);
#input values from user
r1=input("Enter R1: ");
r2=input("Enter R2: ");
s1=input("Enter S1: ");
s2=input("Enter S2: ");

#Contrast Stretching formula
a = s1/r1;
b = (s2-s1)/(r2-r1);
c = (255-s2)/(255-r2);
for i=1:row
  for j=1:col
    if pic(i,j) < r1
    s(i,j) = a*pic(i,j);
    elseif pic(i,j) < r2
    s(i,j) = b*(pic(i,j)-r1)+s1;
    else
    s(i,j) = c*(pic(i,j)-r2)+s2;
    endif
  endfor
endfor

subplot(2,2,1)
imshow(pic);
title("Original Image");
subplot(2,2,2)
imhist(pic);
title('Histogtram Of Original Image');

subplot(2,2,3)
imshow(s);
title("Contrast Streched Image");
subplot(2,2,4)
imhist(s);
title('Histogtram Of Contrast Streched Image');
```

Output:

Enter R1: 0.44
Enter R2: 0.8
Enter S1: 0.1
Enter S2: 0.98



B. Piecewise transformation – Thresholding

Thresholding is a type of image segmentation, where we change the pixels of an image to make the image easier to analyze. In thresholding, we convert an image from color or grayscale into a binary image, i.e., one that is simply black and white.

Code:

```
#Practical 2 : Piecewise transformation
#B) Thresholding

pkg load image;
clear all;
close all;

pic=imread('fields.jpg'); # "imread" is used to read the images
#pic=rgb2gray(pic); # "rgb2gray" is used to convert the color img to gray img
#pic=im2double(pic); # "im2double" converts the datatype to double
threshold=150; # Defining Threshold Value

[row col]=size(pic); # taking size of an image into matrix
pic1=zeros(row,col);
for i=1:row     #reading row value
  for j=1:col   #reading col value
    if(pic(i,j))>threshold
    pic1(i,j)=1;
  else
    pic1(i,j)=0;
    endif
  endfor
endfor

subplot(2,2,1);
imshow(pic);
title('Before Thresholding');
subplot(2,2,2);
imhist(pic);
title('Histogram Before Thresholding');

subplot(2,2,3);
imshow(pic1);
title('After Thresholding');
subplot(2,2,4);
imhist(pic1);
title('Histogram After Thresholding');
```

Output:

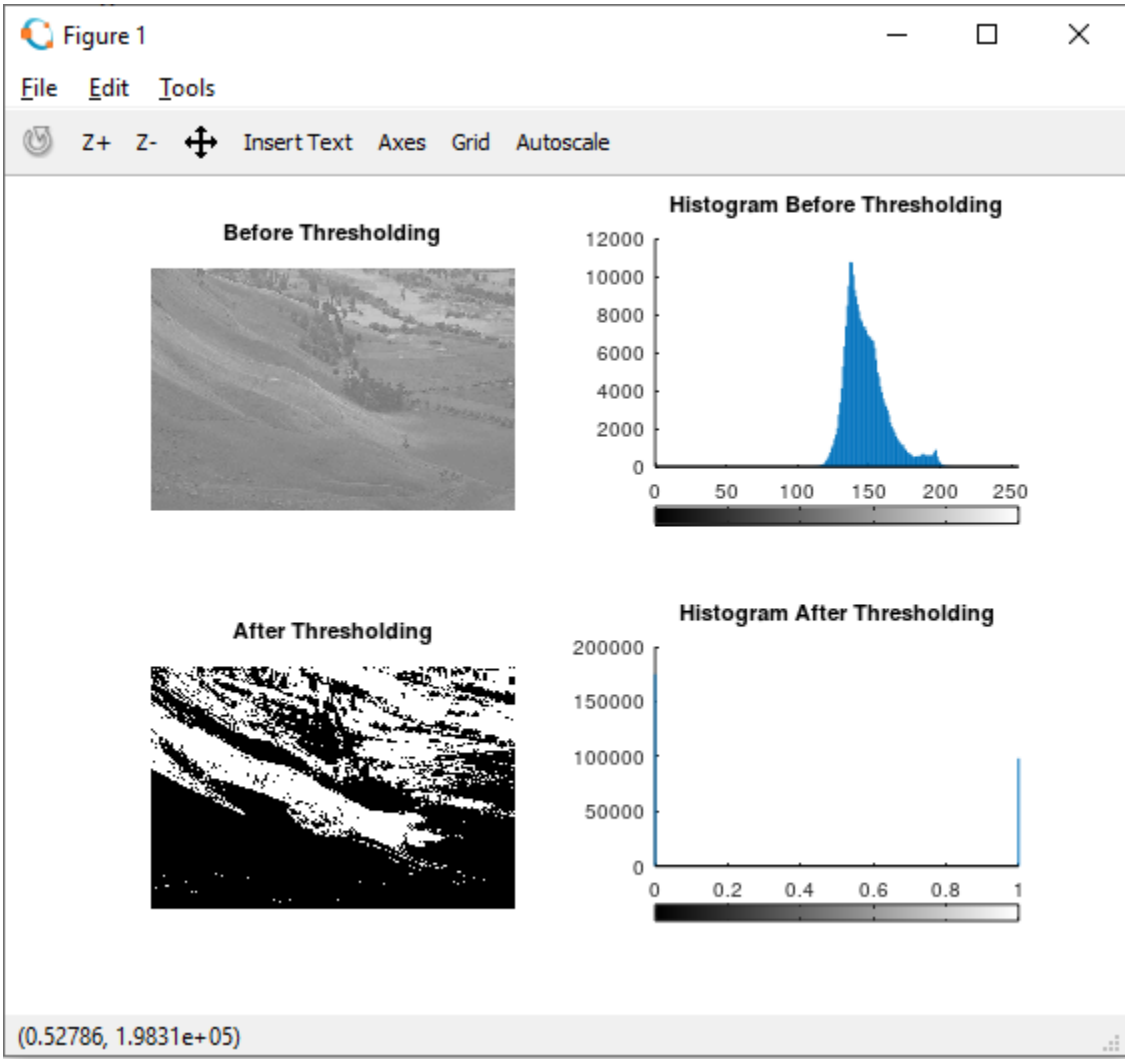


Figure 1
File Edit Tools
Z+ Z- Insert Text Axes Grid Autoscale
Before Thresholding
Histogram Before Thresholding
12000
10000
8000
6000
4000
2000
0
0 50 100 150 200 250
After Thresholding
Histogram After Thresholding
200000
150000
100000
50000
0
0 0.2 0.4 0.6 0.8 1
(0.52786, 1.9831e+05)

C. Piecewise transformation – Bit Plane Slicing

Bit plane slicing is a method of representing an image with one or more bits of the byte used for each pixel. One can use only MSB to represent the pixel, which reduces the original gray level to a binary image. The three main goals of bit plane slicing is:

- Converting a gray level image to a binary image.
- Representing an image with fewer bits and corresponding the image to a smaller size
- Enhancing the image by focussing.

Example: For instance, consider the matrix
A =[167 133 111
144 140 135
159 154 148] and the respective bit format

10100111	10000101	01101111
10010000	10001100	10000111
10011111	10011010	10010100

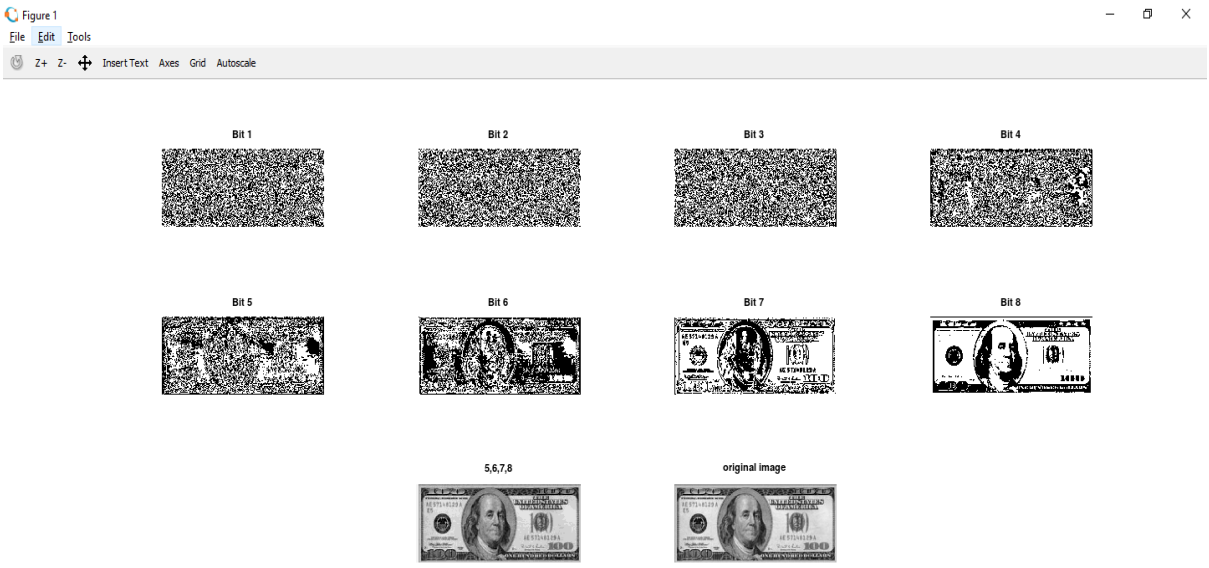
- Combine the 8 bit plane and 7 bit plane.
- For 10100111, multiply the 8 bit plane with 128 and 7 bit plane with 64
  (1x128) + (0x64) + (1x0) + (0x0) + (0x0) + (1x0) + (1x0) + (1x0) = 128
- Repeat this process for all the values in the matrix and the final result will be
  [128 128 64
  128 128 128
  128 128 128]

Code:

```
#Practical 2 : Piecewise transformation
#C) Bit Plane Slicing

pkg load image;
clear all;
close all;

pic=imread('doller.PNG'); # "imread" is used to read the images
pic=rgb2gray(pic); # "rgb2gray" is used to convert the color img to gray img
B=zeros(size(pic));

#Getting the bit at specified position
g1 = bitget(pic,1);
g2 = bitget(pic,2);
g3 = bitget(pic,3);
g4 = bitget(pic,4);
g5 = bitget(pic,5);
g6 = bitget(pic,6);
g7 = bitget(pic,7);
g8 = bitget(pic,8);
```

```
figure 1;
subplot(4,4,1)
imshow(logical(g1)); title('Bit 1');
subplot(4,4,2)
imshow(logical(g2)); title("Bit 2");
subplot(4,4,3)
imshow(logical(g3)); title('Bit 3');
subplot(4,4,4)
imshow(logical(g4));title('Bit 4');

subplot(4,4,5)
imshow(logical(g5)); title('Bit 5');
subplot(4,4,6)
imshow(logical(g6)); title("Bit 6");
subplot(4,4,7)
imshow(logical(g7)); title('Bit 7');
subplot(4,4,8)
imshow(logical(g8)); title('Bit 8');

#Reconstructing Image
B=bitset(B,5,g5);
B=bitset(B,6,g6);
B=bitset(B,7,g7);
B=bitset(B,8,g8);
B=uint8(B);
subplot(4,4,10),imshow(B); title("5,6,7,8")
subplot(4,4,11),imshow(pic); title("original image");
```

Output:



Practical 3
Implement Histogram Equalization

Histogram Equalization is a computer image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. This method usually increases the global contrast of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast.

The following steps are performed to obtain histogram equalization:

1. Find the frequency of each pixel value.
   Consider a matrix A= $\begin{bmatrix} 1 & 4 & 2 \\ 5 & 1 & 3 \\ 1 & 2 & 4 \end{bmatrix}$ with no of bins =5.
   The pixel value 1 occurs 3 times.
   Similarly the pixel value 2 occurs 2 times and so on.
2. Find the probability of each frequency.
   The probability of pixel value 1's occurrence = frequency (1)/no of pixels.
   i.e 3/9.
3. Find the cumulative histogram of each pixel:
   The cumulative histogram of 1 = 3.
   Cumulative histogram of 2 = cumulative histogram of 1 + frequency of 2=5.
   Cumulative histogram of 3 =
   cumulative histogram of 2+frequency of 3 = 5+1=6.
4. Find the cumulative distribution probability of each pixel
   cdf of 1= cumulative histogram of 1/no of pixels= 3/9.
5. Calculate the final value of each pixel by multiplying cdf with (no of bins);
   cdf of 1= (3/9)*(5)=1.6667. Round off the value.
6. Now replace the final values : $\begin{bmatrix} (2) & 4 & 3 \\ 5 & (2) & 3 \\ (2) & 3 & 4 \end{bmatrix}$

The final value for bin 1 is 2. It is placed in the place of 1 in the matrix.

Code:

```
#Practical 3: Histogram Equalization

pkg load image;
clear all;
close all;

pic=imread('fields.jpg'); # "imread" is used to read the images
row=size(pic,1);
col=size(pic,2);
pichist=uint8(zeros(row,col));
```

```
n=row*col;
f=zeros(256,1);
pdf=zeros(256,1);
cdf=zeros(256,1);
cumm=zeros(256,1);
out=zeros(256,1);

for i=1:row
  for j=1:col
    values=pic(i,j);
    f(values+1)=f(values+1)+1;
    pdf(values+1)=f(values+1)/n;

  endfor
endfor

sum=0; L=255; size(pdf);
for i=1:size(pdf)
  sum=sum+f(i);
  cum(i)=sum;
  cdf(i)=cum(i)/n;
  out(i)=round(cdf(i)*L);
endfor

for i=1:row
  for j=1:col
  pichist(i,j)=out(pic(i,j)+1);
endfor
endfor

figure,
subplot(2,2,1), imshow(pic); title('original image');
subplot(2,2,2), imhist(pic); title('original hist');
subplot(2,2,3), imshow(pichist); title('after processing image');
subplot(2,2,4), imhist(pichist); title('after processing hist');
```

Output:

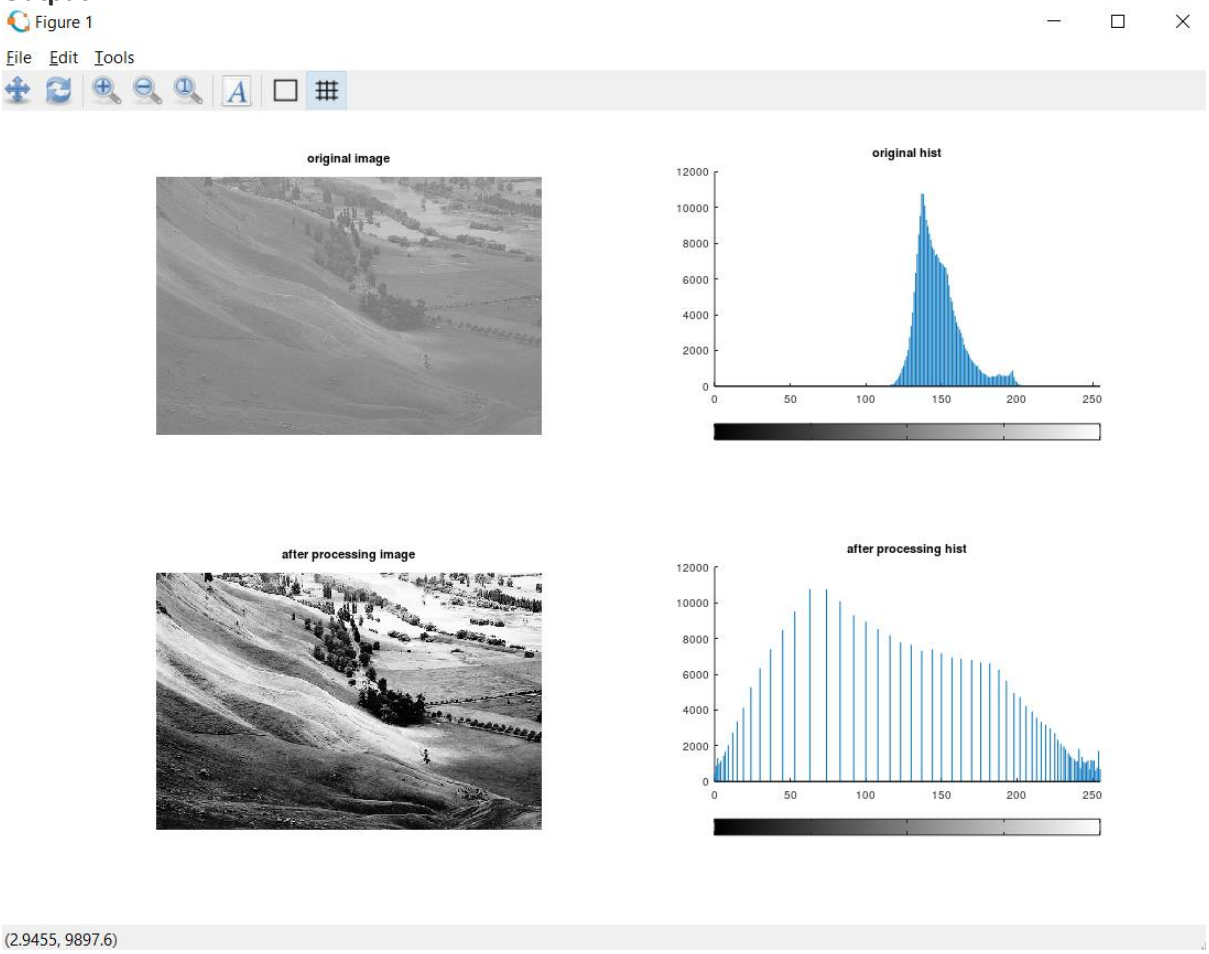


Figure 1
File Edit Tools
original image
original hist
12000
10000
8000
6000
4000
2000
0
0 50 100 150 200 250
after processing image
after processing hist
12000
10000
8000
6000
4000
2000
0
0 50 100 150 200 250
(2.9455, 9897.6)

Practical 4
Image filtering in Spatial Domain
A. Low-pass Filter/Smoothing Filters (Average filter using inbuilt functions)

Averaging filters can be applied for image denoising since the image pixel values change slowly but noise is a wide band signal.
This filters blur image edges and other details.

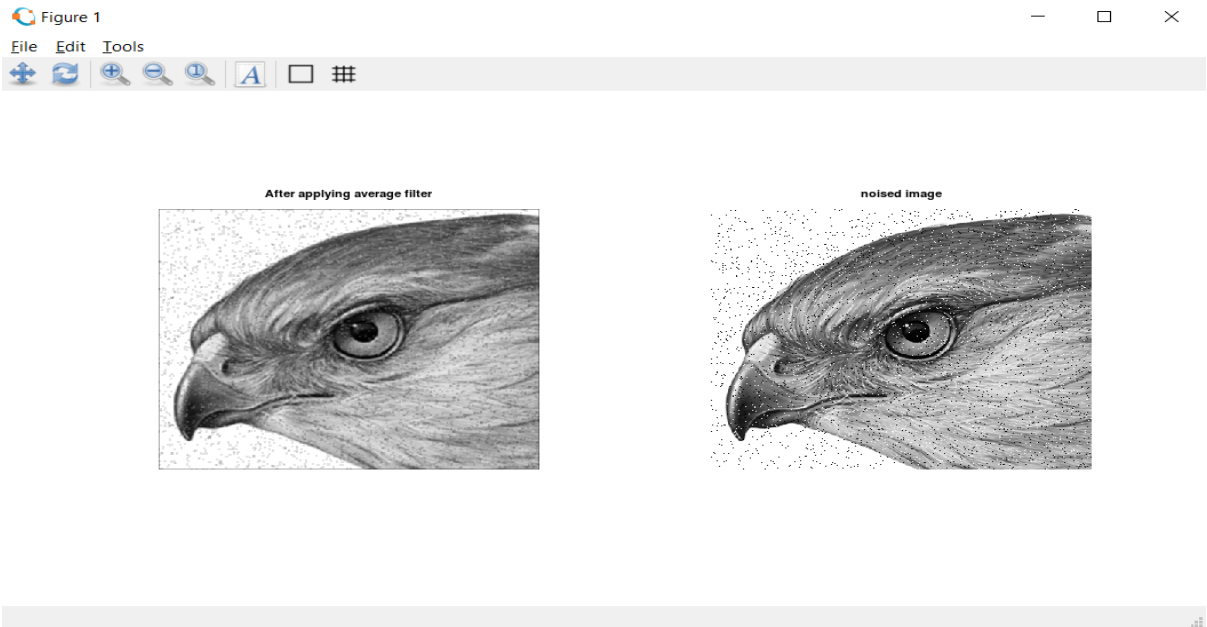
- This means that for image denoising there is a trade-off between noise remove capability and blurring of image detail.
- Larger windows remove more noise but introduce more blur.

Fundamentally, an averaging filter is a low-pass filter

Code:

```
#Practical 4 : Image filtering in Spatial Domain
#A. Low-pass Filter/Smoothing Filters (Average filter using inbuilt functions)


pkg load image;
clear all;
close all;

pic=imread('hawk1.png');
pic=im2double(pic);
noiseimg=imnoise(pic,'salt & pepper' );
f=ones(3,3)/9;
avgfilter=filter2(f,noiseimg);

figure
subplot(1,2,1);imshow(avgfilter); title('After applying average filter');
subplot(1,2,2);imshow(noiseimg); title('noised image');
```

Output:



B. Low-pass Filter/Smoothing Filters (Weighted Average filter)

Code:

```
#Practical 4 : Image filtering in Spatial Domain
#A. Low-pass Filter/Smoothing Filters (Weighted Average filter)


pkg load image;
clear all;
close all;

pic=imread('hawk1.png'); % To read image
Noiseimg=imnoise(pic,'salt & pepper'); % Adding salt & pepper noise to image
w=(1/16)*[1 2 1;2 4 2;1 2 1]; % Defining the box filter mask
% get array sizes
[ma, na] = size(Noiseimg)
[mb, nb] = size(w)
% To do convolution
c = zeros( ma+mb-1, na+nb-1 );
size_c=size(c)
for i = 1:mb
  for j = 1:nb
    r1 = i
    r2 = r1 + ma - 1
    c1 = j
    c2 = c1 + na - 1
    c(r1:r2,c1:c2) = c(r1:r2,c1:c2) + w(i,j) * (Noiseimg);
  end
end
% extract region of size(a) from c
r1 = floor(mb/2) + 1;
r2 = r1 + ma - 1;
c1 = floor(nb/2) + 1;
c2 = c1 + na - 1;
c = c(r1:r2, c1:c2);
figure
subplot(1,2,1)
imshow(Noiseimg);
title('Noisy Image(Salt & Pepper Noise)');
subplot(1,2,2)
imshow(uint8(c));
title('Denoised Image using Weighted Average');
```

Output:

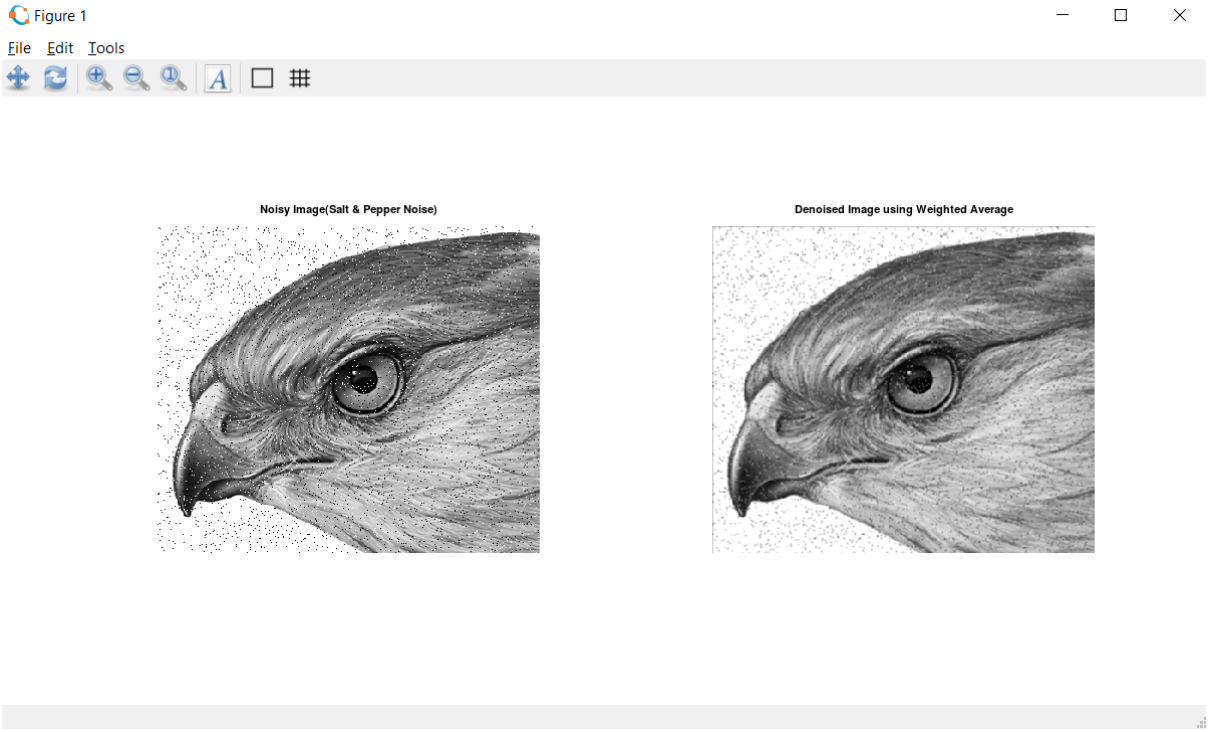


Figure 1
File Edit Tools
Noisy Image(Salt & Pepper Noise)
Denoised Image using Weighted Average

C. Low-pass Filter/Smoothing Filters (Median filter)

The **median filter** is a non-linear digital **filtering** technique, often used to remove noise from an **image** or signal. Such noise reduction is a typical pre-**processing** step to improve the results of later **processing** (for example, edge detection on an **image**).

Code:

```
#Practical 4 : Image filtering in Spatial Domain
#A. Low-pass Filter/Smoothing Filters (Median filter)

pkg load image;
clear all;
close all;

# Read the image
pic=imread('hawk1.png');
img_noisy1=imnoise(pic,'salt & pepper' );
# Obtain the number of rows and columns of the image
[m, n] = size(img_noisy1)
# Traverse the image. For every 3X3 area,
# find the median of the pixels and
# replace the center pixel by the median
img_new1 = zeros(m, n);

for i=2: m-1
    for j =2: n-1
      temp = [img_noisy1(i-1, j-1),
            img_noisy1(i-1, j),
            img_noisy1(i-1, j + 1),
            img_noisy1(i, j-1),
            img_noisy1(i, j),
            img_noisy1(i, j + 1),
            img_noisy1(i + 1, j-1),
            img_noisy1(i + 1, j),
            img_noisy1(i + 1, j + 1)] ;
       temp = sort(temp);
       img_new1(i, j)= temp(4);
    endfor
endfor
img_new1 = uint8(img_new1);
figure
subplot(1,2,1); imshow(img_new1); title('After Applying Median Filter');
subplot(1,2,2); imshow(img_noisy1);title('noisy image');
```

Output:


Figure 1
File Edit Tools
After Applying Median Filter
noisy image

D. High-pass Filter / Sharpening Filter (Laplacian Filter)

The Laplacian of an image highlights regions of rapid intensity change and is an example of a second order or a second derivative method of enhancement. It is particularly good at finding the fine details of an image. Any feature with a sharp discontinuity will be enhanced by a Laplacian operator.

Code:

```
#Practical 4 : Image filtering in Spatial Domain
#B. High-pass Filter/Sharpening Filters (Laplacian filter)

pkg load image;
clear all;
close all;


pic=imread('coins.png');
size(pic);
figure,
subplot(2,2,1);imshow(pic); title('original Image');
%Preallocate the matrices with zeros
I1=pic;
I=zeros(size(pic));
I2=zeros(size(pic));
%Filter Masks
F1=[0 2 0;2 -8 2; 0 2 0];

%Padarray with zeros
pic=padarray(pic,[1,1]);
pic=double(pic);
size(pic);
%Implementation of the equation in Fig.D
for i=1:size(pic,1)-2
   for j=1:size(pic,2)-2

      I(i,j)=sum(sum(F1.*pic(i:i+2,j:j+2)));

   end
end

I=uint8(I);


subplot(2,2,3);imshow(I);title('Filtered Image');
%Sharpenend Image
B=I1-I;
subplot(2,2,4); imshow(B);title('Sharpened Image');
```

Output:

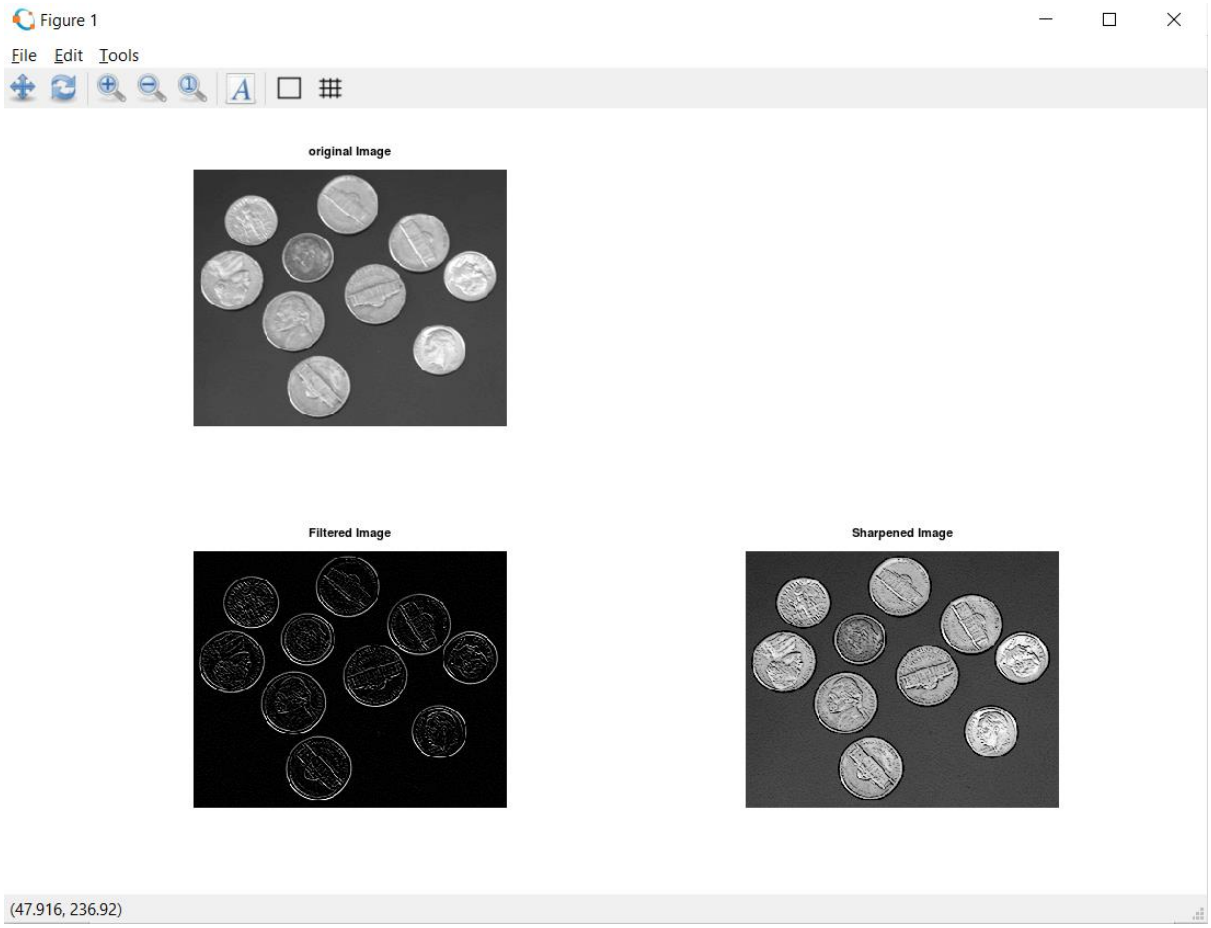


Figure 1
File Edit Tools
original Image
Filtered Image
Sharpened Image
(47.916, 236.92)

E. High-pass / Sharpening Filter (Sobel for edge detection without using edge function)

Code :

```
#Practical 4 : Image filtering in Spatial Domain
#B. High-pass Filter/Sharpening Filters (Sobel Operator without using edge function)

pkg load image;
clear all;
close all;

pic=imread('peppers.png');
figure,
subplot(1,2,1); imshow(pic); title('Original');
C=double(pic);
size(C)

for i=1:size(C,1)-2
   for j=1:size(C,2)-2
      %Sobel mask for x-direction:
      Gx=((C(i+2,j)+2*C(i+2,j+1)+C(i+2,j+2))-(C(i,j)+2*C(i,j+1)+C(i,j+2)));
      %Sobel mask for y-direction:
      Gy=((C(i,j+2)+2*C(i+1,j+2)+C(i+2,j+2))-(C(i,j)+2*C(i+1,j)+C(i+2,j)));
      %The gradient of the image
      pic(i,j)=sqrt(Gx.^2+Gy.^2);

   end
end
subplot(1,2,2); imshow(pic); title('Sobel gradient');
```

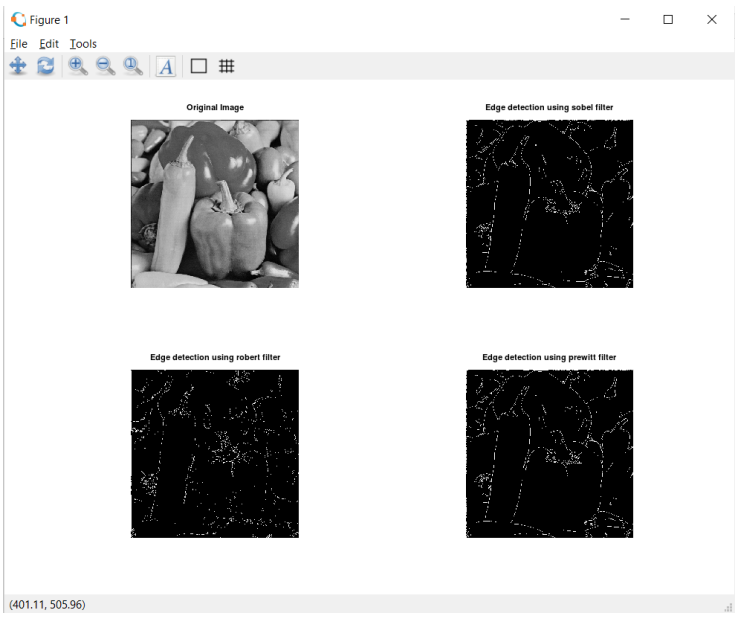
Output:

F. High-pass / Sharpening Filter (Sobel Operator for edge detection using edge function)

Code:

```
#Practical 4 : Image filtering in Spatial Domain
#B. High-pass Filter/Sharpening Filters (Sobel Operator using edge function)

pkg load image;
clear all;
close all;

#Take input image
img=imread("peppers.png");

#function to find edge using sobel filter
sobel = edge(img,'Sobel');
figure;
subplot(2,2,1)
imshow(img);
title('Original Image');
subplot(2,2,2)
imshow(sobel);
title("Edge detection using sobel filter");

#function to find edge using sobel filter
robert = edge(img,'Roberts');
prewitt = edge(img,'Prewitt');
subplot(2,2,3)
imshow(robert);
title('Edge detection using robert filter');
subplot(2,2,4)
imshow(prewitt);
title("Edge detection using prewitt filter");
```

Output:



Practical 5

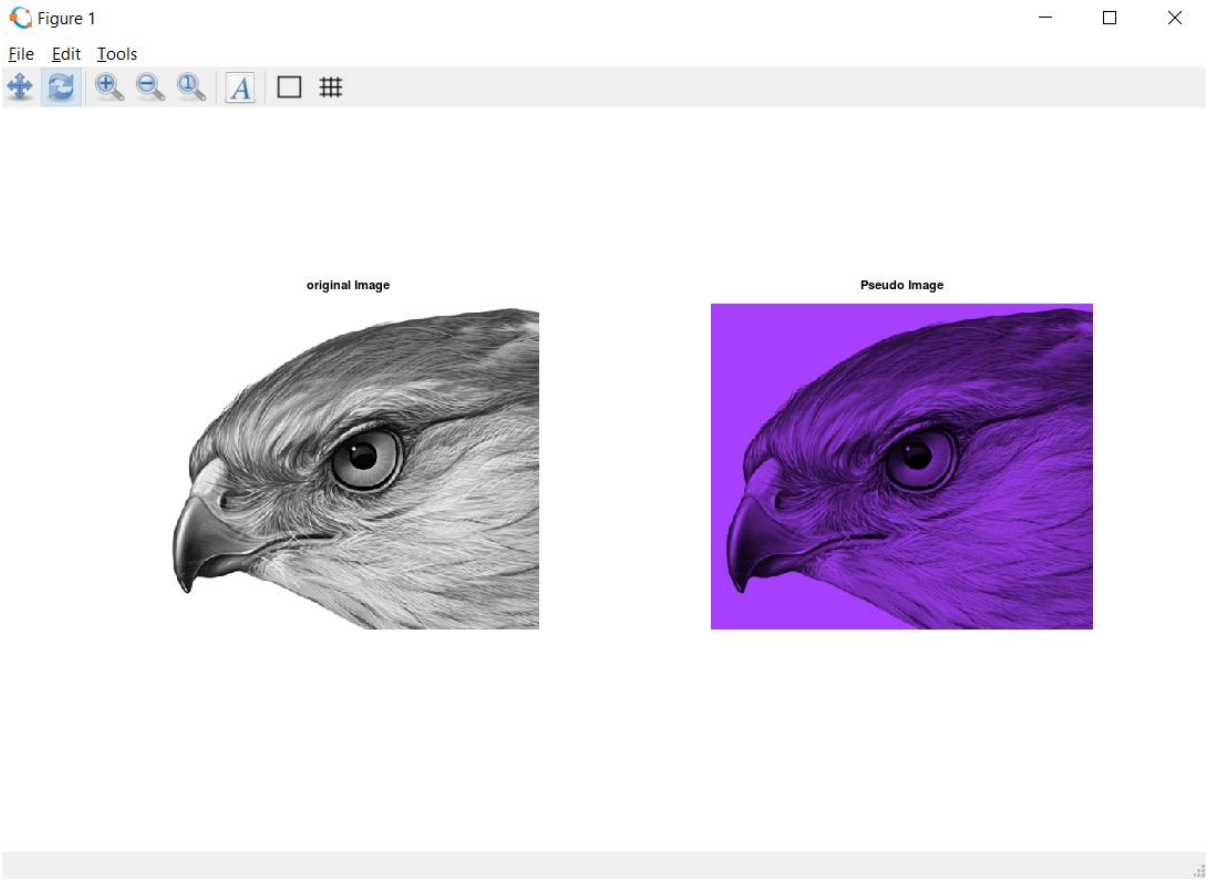
Color Image Processing (PseudoColoring)

Pseudo-color processing is a technique that maps each of the grey levels of a black and white **image** into an assigned color. ... Various color maps can give contrast enhancement effects, contouring effects, or grey level mapping (depicting areas of a given grey level).

Code:

```
#Practical 5 : Color Image Processing
#A. PseudoColoring

pkg load image;
clear all;
close all;

img = imread('hawk1.png'); % Read image
figure,
subplot(1,2,1);imshow(img);title("original Image");
red = 0.66*img;
green=0.25*img;
blue = img;
pseudo_img = cat(3, red, green, blue);
subplot(1,2,2);imshow(pseudo_img), title('Pseudo Image');
```

Output:

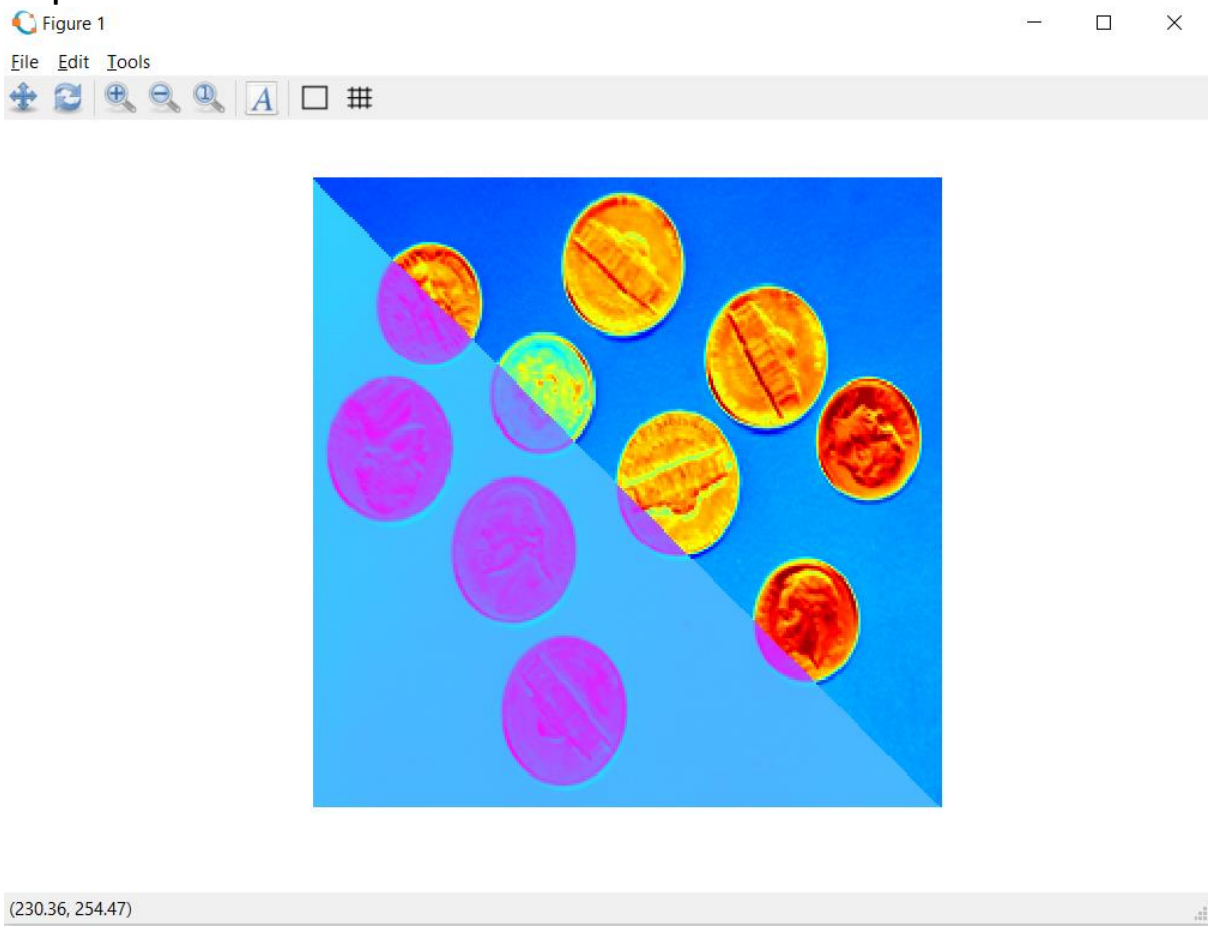


Color Image Processing (Separating the RGB Channels)

Code:

```
#Practical 5 : Color Image Processing
#B. Separating the RGB Channels

pkg load image;
clear all;
close all;

%READ INPUT IMAGE
pic = imread('coins.png');
%RESIZE IMAGE
pic = imresize(pic,[256 256]);
%PRE-ALLOCATE THE OUTPUT MATRIX
Output = ones([size(pic,1) size(pic,2)]);
%COLORMAPS
#maps={'jet(256)';'hsv(256)';'cool(256)';'spring(256)';'summer(256)';'parula(256)';'hot(256)'};
%COLORMAP 1
map = colormap(jet(256));
Red = map(:,1);
Green = map(:,2);
Blue = map(:,3);

R1 = Red(pic);
G1 = Green(pic);
B1 = Blue(pic);

%COLORMAP 2
map = colormap(cool(256));
Red = map(:,1);
Green = map(:,2);
Blue = map(:,3);

%RETRIEVE POSITION OF UPPER TRIANGLE
[x,y]=find(triu(Output)==1);

Output(:,:,1) = Red(pic);
Output(:,:,2) = Green(pic);
Output(:,:,3) = Blue(pic);
for i=1:numel(x)
      Output(x(i),y(i),1)=R1(x(i),y(i));
      Output(x(i),y(i),2)=G1(x(i),y(i));
      Output(x(i),y(i),3)=B1(x(i),y(i));
end
Output = im2uint8(Output);
%FINAL IMAGE
imshow(Output);
```

Output:



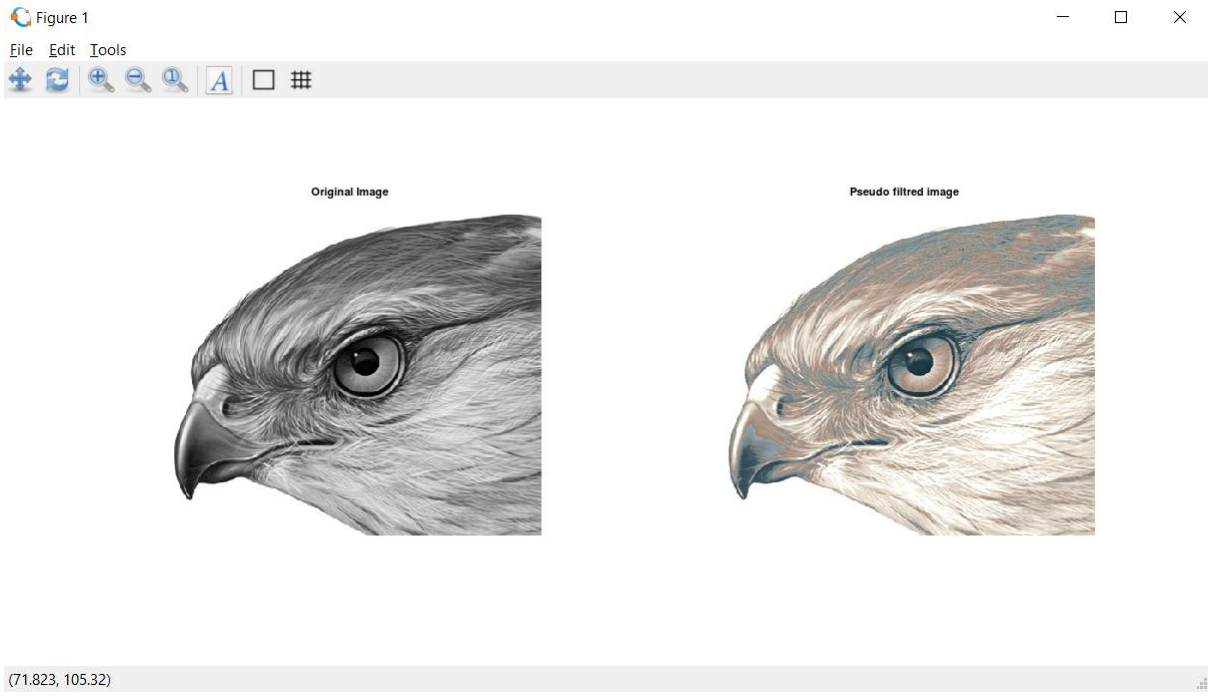
Color Image Processing (Color Slicing)

Code:

```
#Practical 5 : Color Image Processing
#C. Color slicing

pkg load image;
clear all;
close all;

input_image=imread('hawk.png');
k=rgb2gray(input_image);
[x y z]=size(k);
% z should be one for the input image
k=double(k);
for i=1:x
for j=1:y
if k(i,j)>=0 && k(i,j)<50
m(i,j,1)=k(i,j,1)+25;
m(i,j,2)=k(i,j)+50;
m(i,j,3)=k(i,j)+60;
end
if k(i,j)>=50 && k(i,j)<100
m(i,j,1)=k(i,j)+55;
m(i,j,2)=k(i,j)+68;
m(i,j,3)=k(i,j)+70;
end
if k(i,j)>=100 && k(i,j)<150
m(i,j,1)=k(i,j)+52;
m(i,j,2)=k(i,j)+30;
m(i,j,3)=k(i,j)+15;
end
if k(i,j)>=150 && k(i,j)<200
m(i,j,1)=k(i,j)+50;
m(i,j,2)=k(i,j)+40;
m(i,j,3)=k(i,j)+25;
end
if k(i,j)>=200 && k(i,j)<=256
m(i,j,1)=k(i,j)+120;
m(i,j,2)=k(i,j)+60;
m(i,j,3)=k(i,j)+45;
end
end
end
figure,
subplot(1,2,1);
imshow(uint8(k),[]);
title('Original Image');
```

```
subplot(1,2,2);
imshow(uint8(m),[]);
title("Pseudo filtred image");
```

Output:



Practical 6

Image Compression Techniques and watermarking

Implement Huffman Coding

Huffman coding is one of the basic compression methods, that have proven useful in image and video compression standards. When applying Huffman encoding technique on an Image, the source symbols can be either pixel intensities of the Image, or the output of an intensity mapping function.

Code:

```
#Practical 6 : Image Compression Techniques and watermarking
#A. Implement Huffman Coding

pkg load image;
clear all;
close all;

pkg load communications
sig = repmat([3 3 1 3 3 3 3 3 2 3],1,50);
symbols = [1 2 3];
p = [0.1 0.1 0.8];
dict = huffmandict(symbols,p);
hcode = huffmanenco(sig,dict);
dhsig = huffmandeco(hcode,dict);
isequal(sig,dhsig)
binarySig = de2bi(sig);
seqLen = numel(binarySig)
binaryhcode = de2bi(hcode);
encodedLen = numel(binaryhcode)
```

Output:

```
>> prac6aa

ans = 1
seqLen = 1000
encodedLen = 600
```

Implement Watermarking

Code:

```
#Practical 6 : Image Compression Techniques and watermarking
#B. Watermarking

pkg load image;
clear all;
close all;

#Input Image where we want to apply watermark
pic=imread('lena_color_512.tif');

#For watermarking, size of input image and watermarking image should be same
#there for we changed the size of image using imresize and displayed

picresized=imresize(pic,[560 560]);
figure;
subplot(1,3,1);
imshow(picresized);
title('Original Image with resized');

#Watermarking Image
w=imread('watersample.jpg');
#Again Resized the Watermarking Image
wr=imresize(w,[560 560]);
subplot(1,3,2);imshow(wr);
title('watermark');

#Applied watermarking
alpha=0.7;
fw=(1-alpha)*picresized + alpha.*wr;
#Display the watermarked Image
subplot(1,3,3);imshow(fw);
title('Watermaked Image');
```

Output:

Practical 7

Basic Morphological Transformations

Boundary Extraction

If A is an image and structuring element is B then Boundary Extraction can be given as,

Boundary (A) = A - (A Θ B)

It means subtracting the erode image of A from the original Image. Let A =

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

If B=

1	1	1
1	1	1
1	1	1

Then AΘB would be same as A except one pixel, A Θ 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	0	0
0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Now Boundary (A) =

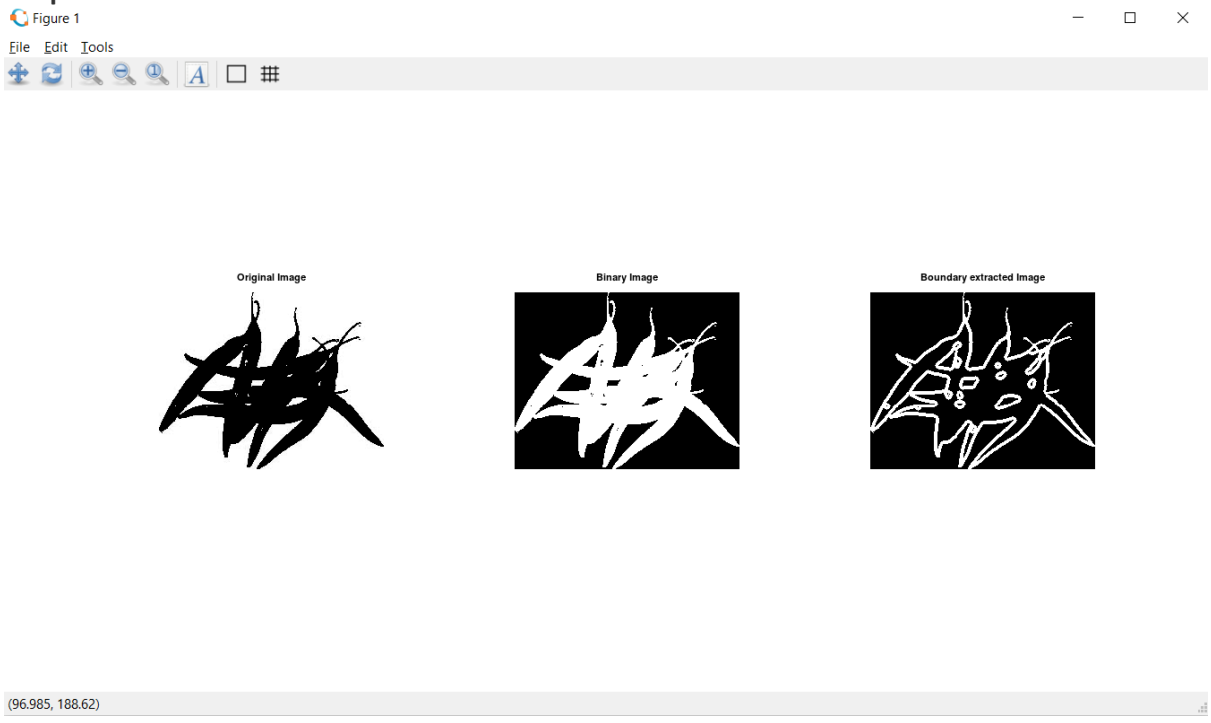
0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0
0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0

Hence, it would give a pixel difference.

Code:

```
#Practical 7 :Basic Morphological Transformations
#A) Boundary Extraction

pkg load image;
clear all;
close all;

pic=imread('pepper.jpg');
pic=rgb2gray(pic);
pic(pic<225)=0;
se=strel('disk',4,0);%Structuring element
binaryimg=~im2bw(pic);%binary Image

erodedimg=imerode(binaryimg,se);%Erode the image by structuring element

figure,
subplot(1,3,1);
imshow(pic);title('Original Image');
subplot(1,3,2);
imshow(binaryimg);title('Binary Image');
%Difference between binary image and Eroded image
subplot(1,3,3);imshow(binaryimg-erodedimg);title('Boundary extracted Image');
```

Output:



Thinning, Thickening and Skeletonization

Thinning is a morphological operation that is used to remove selected foreground pixels from binary **images**, somewhat like erosion or opening.

Thickening is a morphological operation that is used to grow selected regions of foreground pixels in binary **images**, somewhat like dilation or closing.

Skeletonization is a process for reducing foreground regions in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels.

Code:

```
#Practical 7 :Basic Morphological Transformations
#B) Thinning,Thickening and skeletonization

pkg load image;
clear all;
close all;

% Read the test Image
% Convert the image to binary image

myorigimg = imread('monarch.png');
myorigimg = im2bw(rgb2gray(myorigimg));
subplot(2, 2, 1);
imshow(myorigimg);title('Original image');

% Perform Thinning operation using bwmorph() command
% Display the dilated image

thinf = bwmorph(myorigimg,'thin');
subplot(2,2,2);
imshow(thinf);title('Thinning of the Image');

% Perform Thickening operation using bwmorph()command
% Display the dilated image

thickf = bwmorph(myorigimg,'thicken');
subplot(2,2,3);
imshow(thickf);title('Thickening of the Image');

% Perform Skeletonozation operation using bwmorph()command
% with 8 iterations and display the dilated image

skelf100 = bwmorph(myorigimg,'skel',9);
subplot(2,2,4);
imshow(skelf100);title('Skeletonization - 9 iterations');
```

Output:

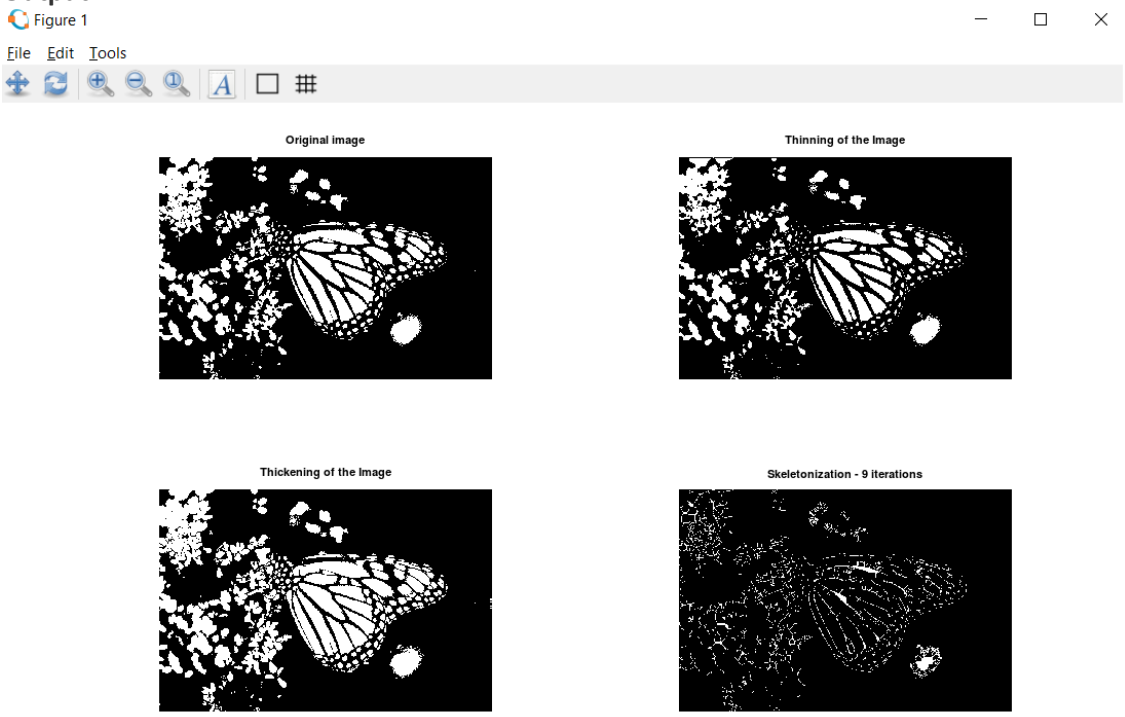


Figure 1
File Edit Tools
Original image
Thinning of the Image
Thickening of the Image
Skeletonization - 9 iterations