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

DEPARTMENT OF INFORMATION TECHNOLOGY
2020 - 2021

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

Name: Sneha Ramchandra Pawar
Roll No.: 11

Hindi Vidya Prachar Samiti's

RAMNIRANJAN JHUNJHUNWALA COLLEGE

(AUTONOMOUS)

Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai-400086

CERTIFICATE

This is to certify that Miss. **Sneha Ramchandra Pawar** with Roll No. **11** has successfully completed the necessary course of experiments in the subject of **Image and Vision Processing** during the academic year **2019 – 2020** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc. (IT)** semester -I.

_____ _____

Internal Examiner External Examiner

_____ _____

Head of Department College Seal

INDEX		
Sr. No.	Practical	Sign
1.	Implement basic intensity transformation functions. A) Image Inverse B) Log Transformation C) Power - law transformation	
2.	Piecewise Transformation A) Contrast Stretching B) Thresholding C) Bit - Plane Slicing	
3.	Implement Histogram Equalization	
4.	Image filtering in Spatial Domain A) Low-pass Filter/Smoothing Filters (Average, Weighted Average, Median and Gaussian) B) High-pass Filter/Sharpening Filter (Laplacian Filter, sobel, Robert and Prewitt Filter to detect edge)	
5.	Analyze Image in Frequency Domain A) Low Pass/Smoothing Filter B) High Pass/ Sharpening Filter	
6.	Colour Image Processing. A) Pseudocoloring B) Separating the RGB Channels C) Color Slicing	
7.	Image Compression Techniques and Watermarking A) Implement Huffman Coding B) Add a watermark to the image	
8.	Basic Morphological Transfornations A) Boundary Extraction B) Thinning and Thickening C) Hole filling and Skeletons	

Practical 1

Implemet basic intensity transformation functions.
A) Implement Image Inverse:

- Black and white image inversion refers to an image processing technique where light areas are mapped to dark, and dark areas are mapped to light.
- In other words, after image inversion black becomes white and white becomes black.
- An inverted black and white image can b thought of as a digital negative of the original image.

- 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 is (maximum pixel value) & r is (Pixel of an image).

imread():
The imread() function reads images from the graphics files. A = imread(filename) reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file. A = imread(filename,fmt) additionally specifies the format of the file with the standard file extension indicated by fmt. If imread cannot find a file with the name specified by filename, it looks for a file named filename.fmt.

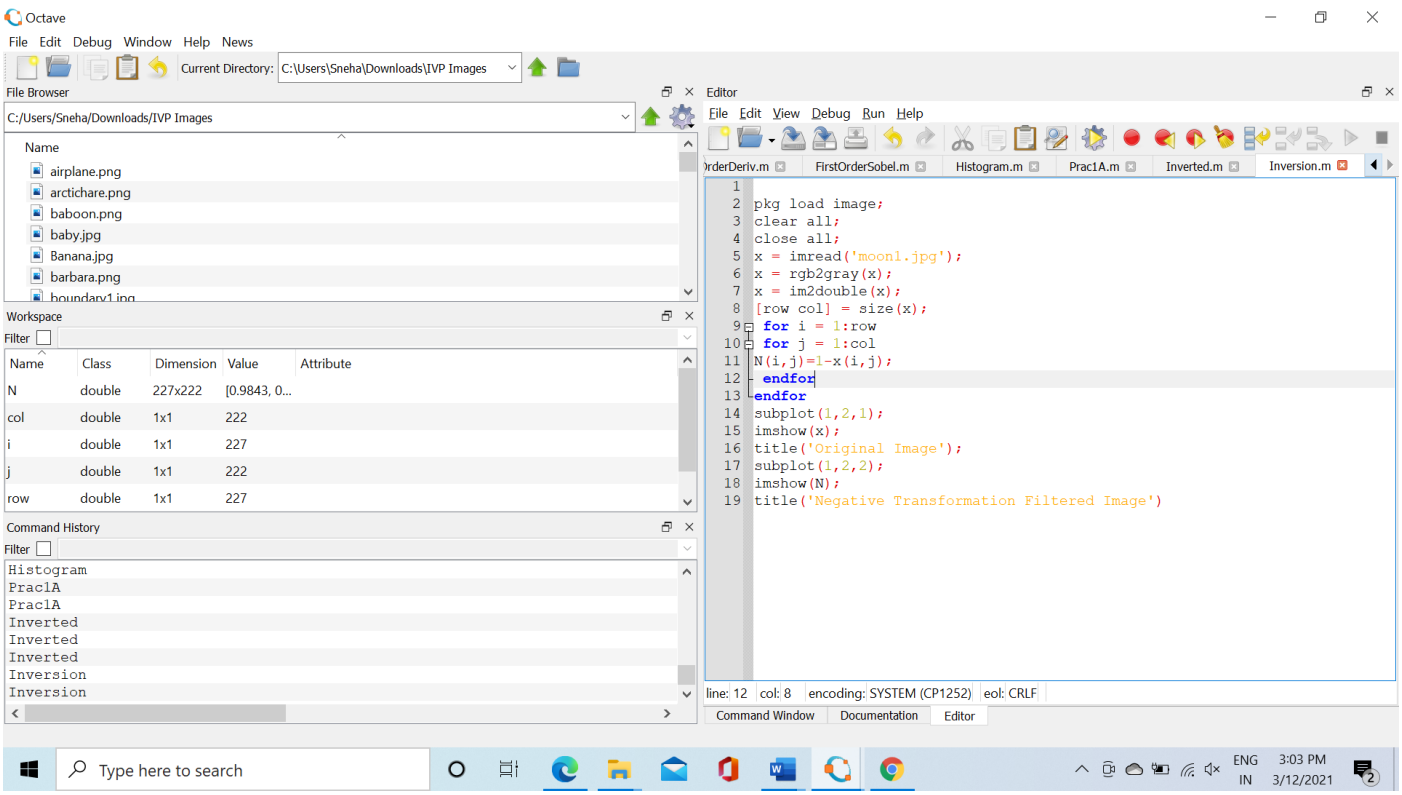
im2double():
I2 = im2double(I) I2 = im2double(I,'indexed') I2 = im2double(I) converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a binary image. im2double rescales the output from integer data types to the range [0, 1]. I2 = im2double(I,'indexed') converts the indexed image I to double precision. im2double adds an offset of 1 to the output from integer data types. subplot(): subplot(m,n,p) divides the current figure into an m-by-n grid and creates axes in the position specified by p. MATLAB® numbers subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If axes exist in the specified position, then this command makes the axes the current axes imshow(): imshow(I) displays the grayscale image I in a figure. imshow uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display

Code :

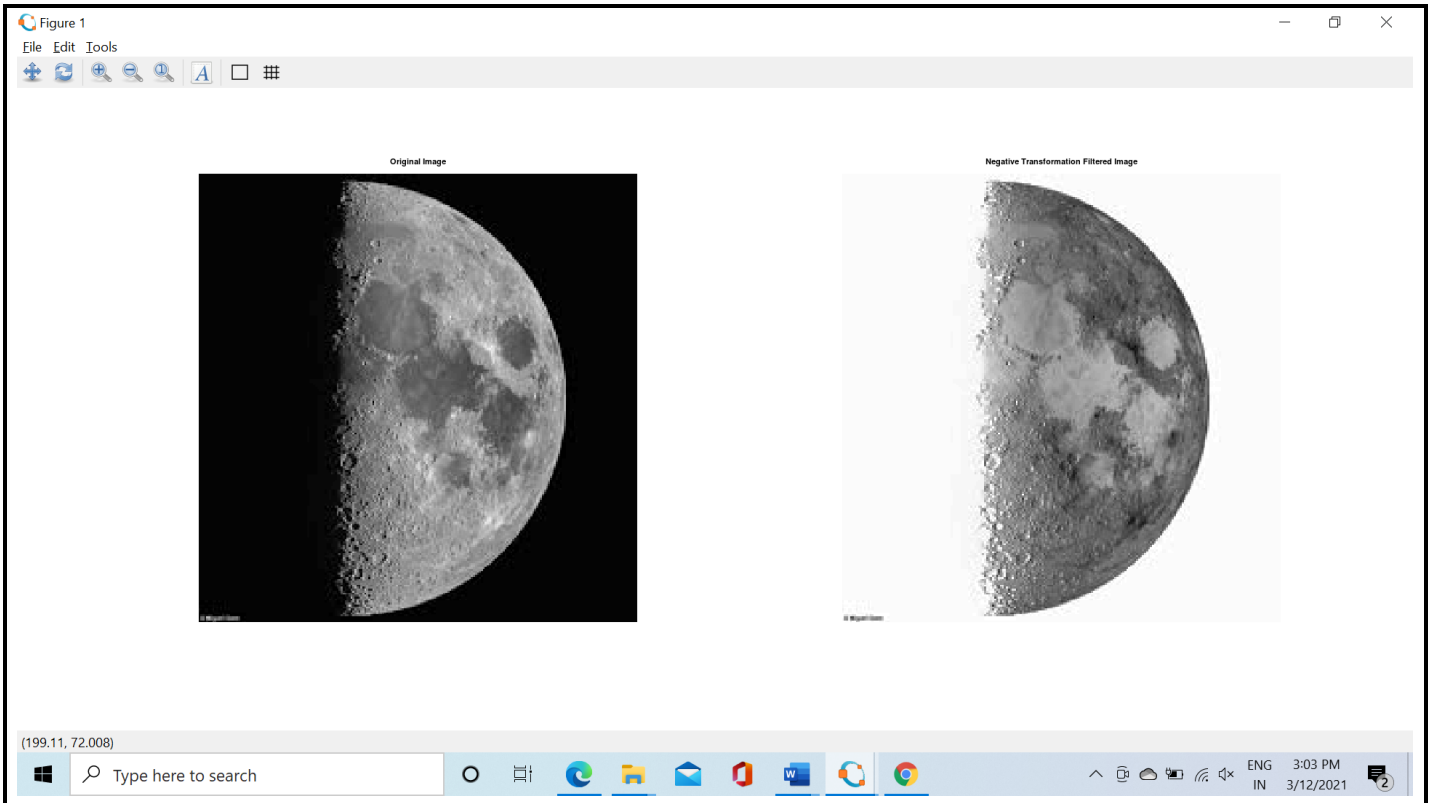
Without Using Functions:

```
pkg load image;
```

```
clear all;
close all;
x = imread('moon1.jpg');
x = rgb2gray(x);
x = im2double(x);
[row col] = size(x);
  for i = 1:row
  for j = 1:col
N(i,j)=1-x(i,j);
  endfor
endfor
subplot(1,2,1);
imshow(x);
title('Original Image');
subplot(1,2,2);
imshow(N);
title('Negative Transformation Filtered Image')
```



OUTPUT:

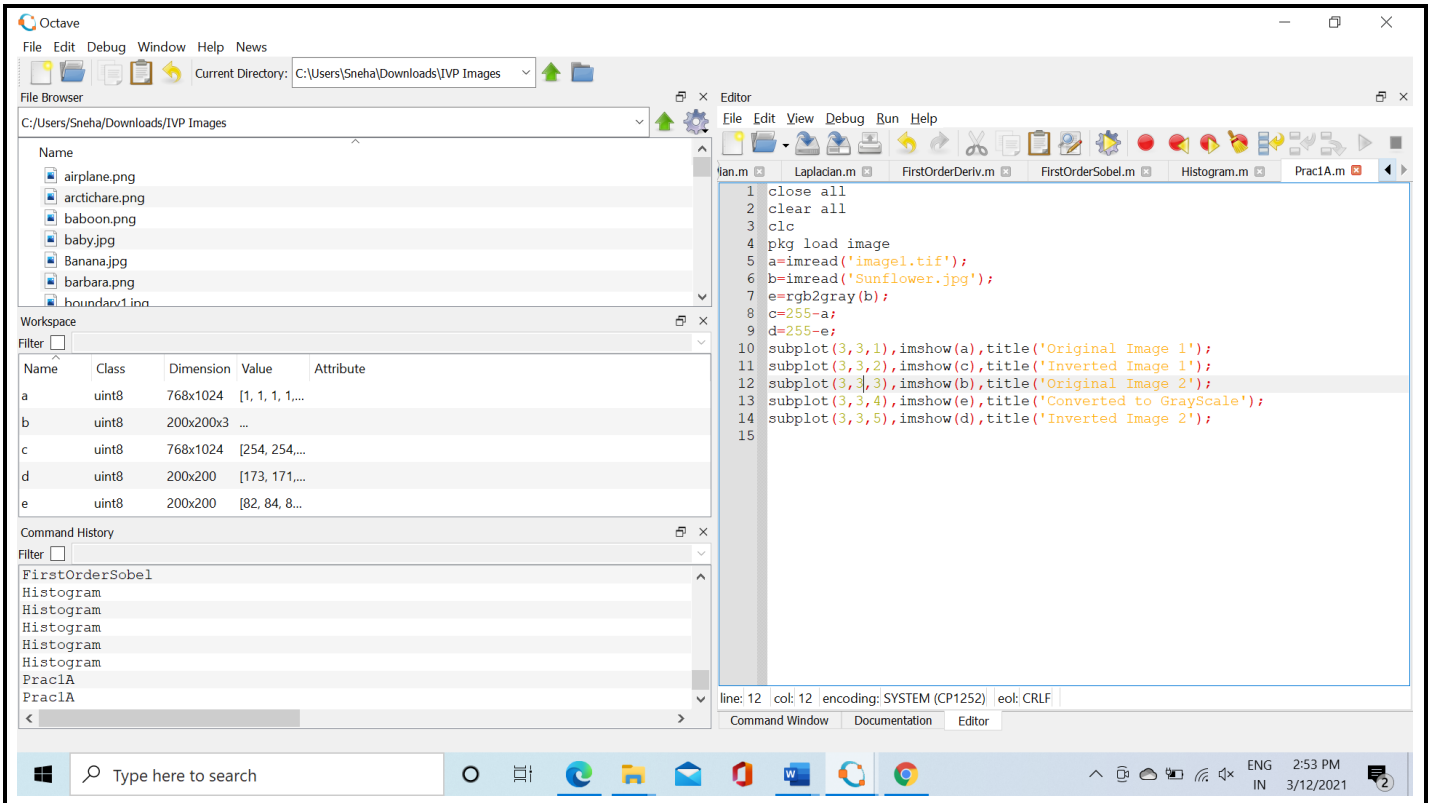


 # 1) Implement Basic Intensity Transformation Functions.

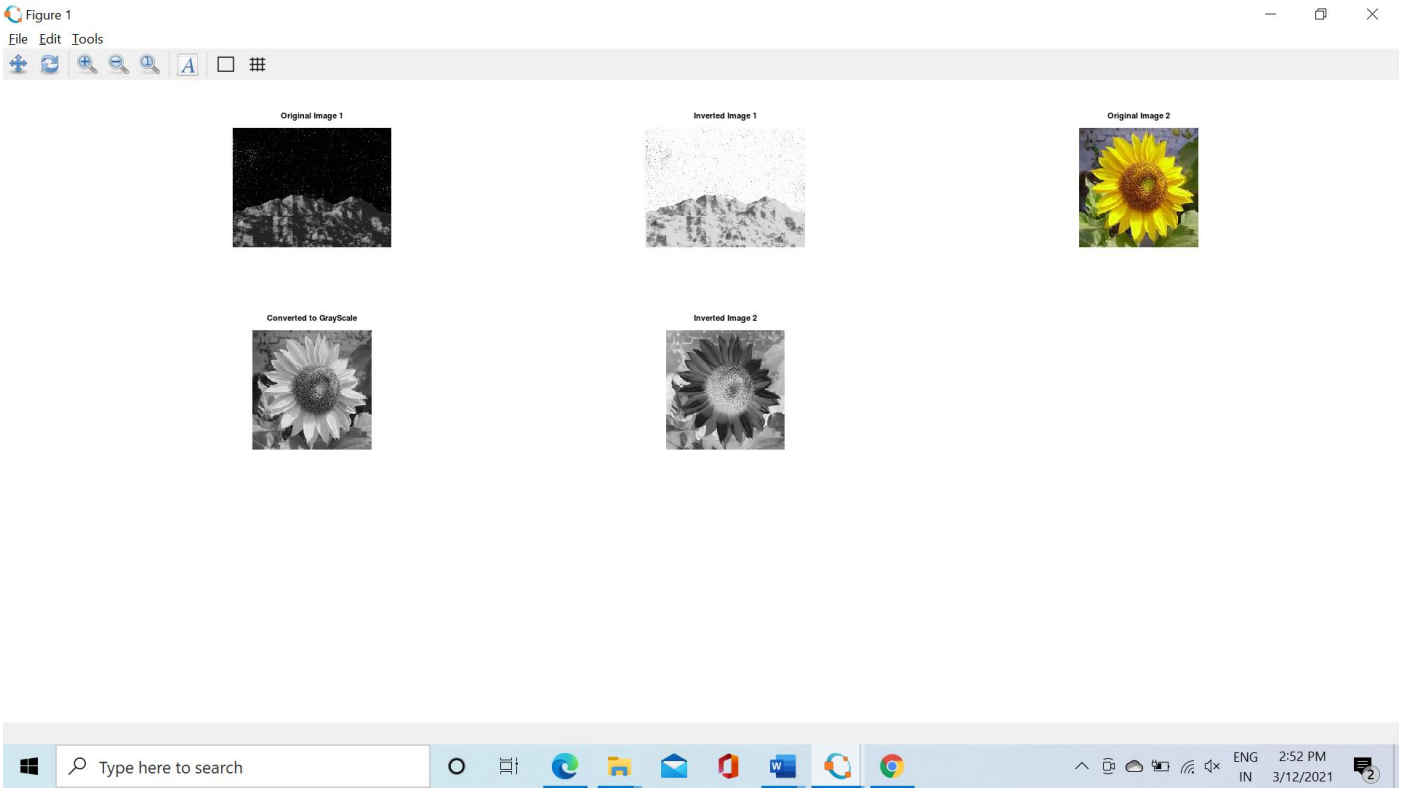
#--

 # 1. A) Image Inverse

```
close all
clear all
clc
pkg load image
a=imread('image1.tif');
b=imread('Sunflower.jpg');
e=rgb2gray(b);
c=255-a;
d=255-e;
subplot(2,2,1),imshow(a),title('Original Image 1');
subplot(2,2,2),imshow(c),title('Inverted Image 1');
subplot(2,2,3),imshow(b),title('Original Image 2');
subplot(2,2,4),imshow(e),title('Converted to GrayScale');
subplot(2,2,5),imshow(d),title('Inverted Image 2');
```



Output :



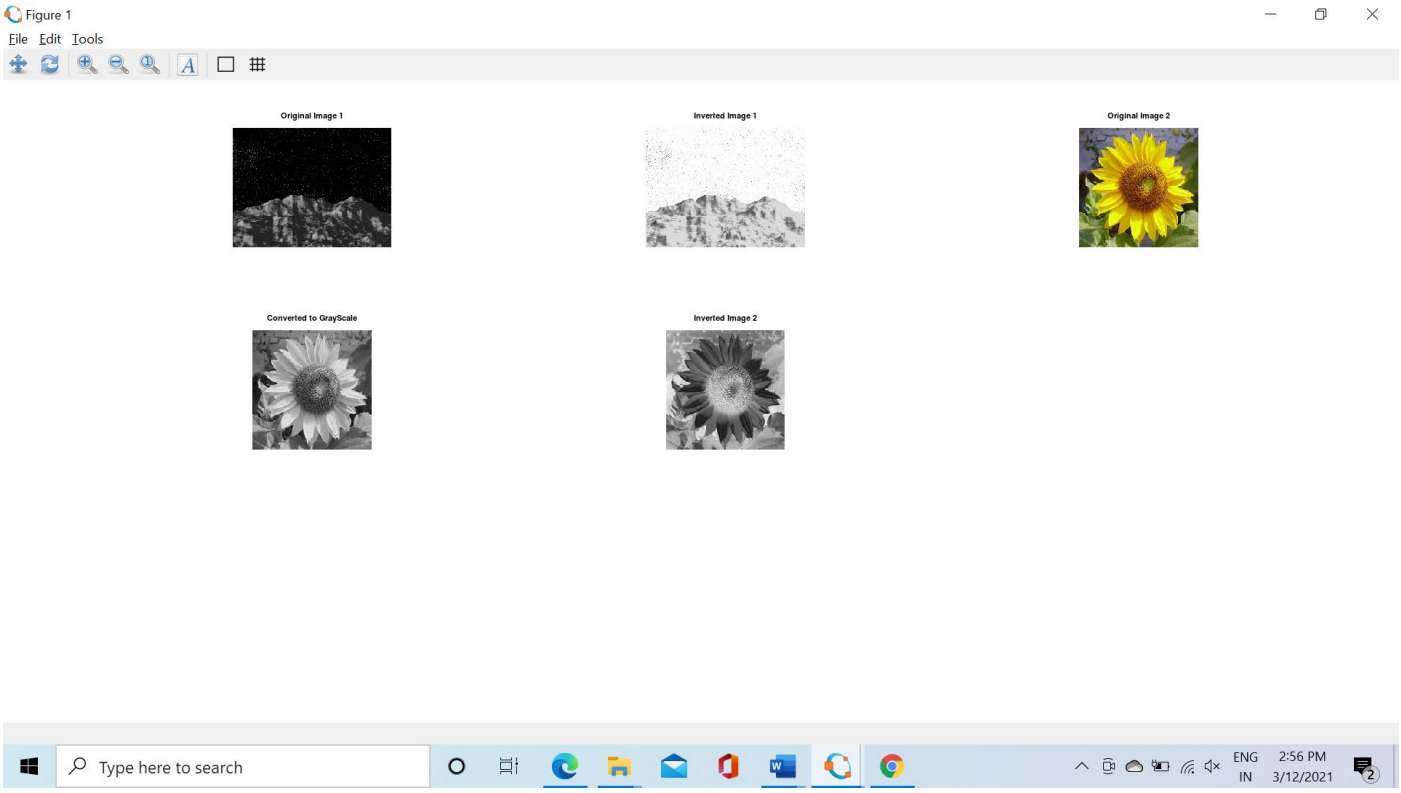
By using imcomplement function:

Code :

 # 1) Implement Basic Intensity Transformation Functions.

```
#--------------------------------------------------------------------------
# 1. A) Image Inverse by using imcomplement function
close all
clear all
clc
pkg load image
a=imread('image1.tif');
b=imread('Sunflower.jpg');
e=rgb2gray(b)
c=imcomplement(a);
d=imcomplement(e);
subplot(3,3,1),imshow(a),title('Original Image 1');
subplot(3,3,2),imshow(c),title('Inverted Image 1');
subplot(3,3,3),imshow(b),title('Original Image 2');
subplot(3,3,4),imshow(e),title('Converted to GrayScale');
subplot(3,3,5),imshow(d),title('Inverted Image 2');
```

Output :

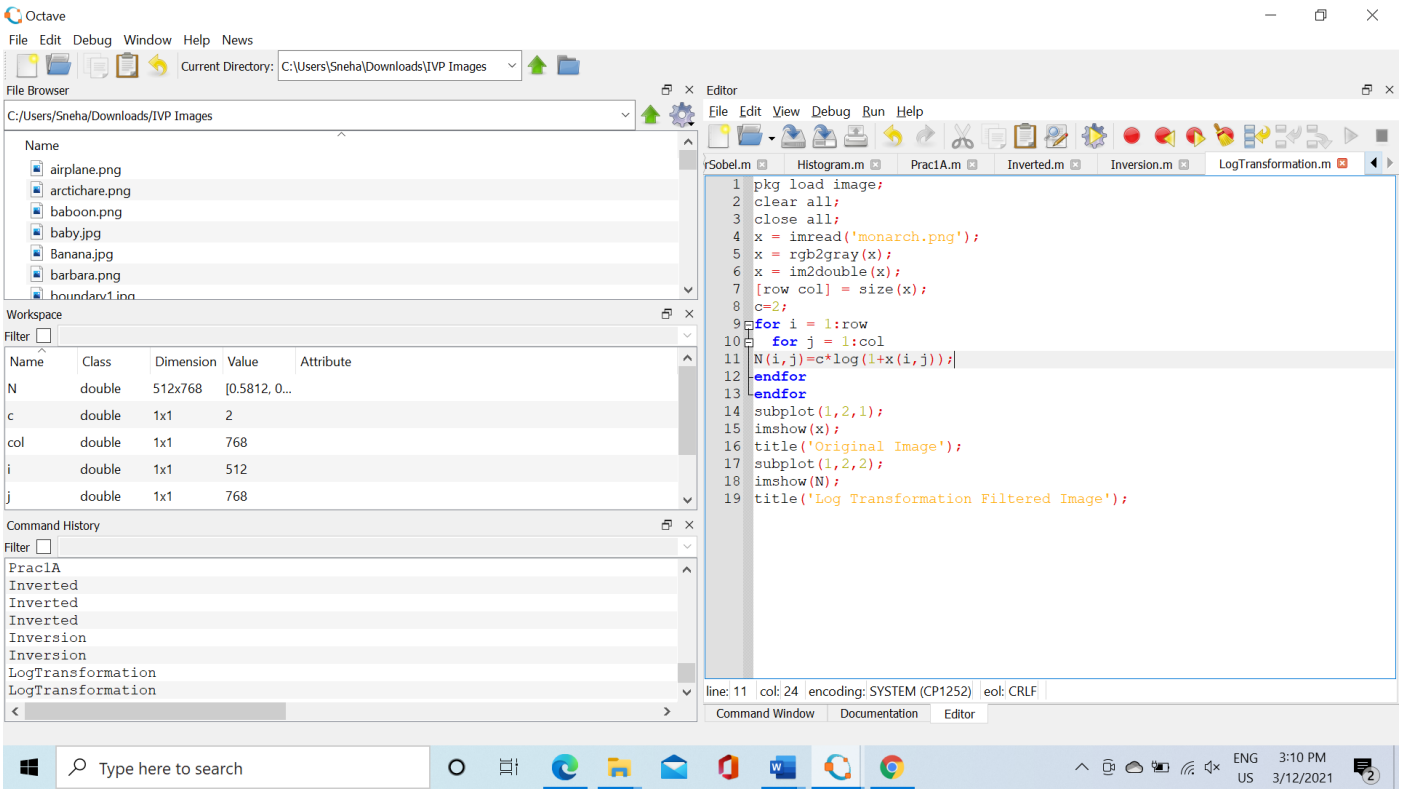


B) Log Transformation:

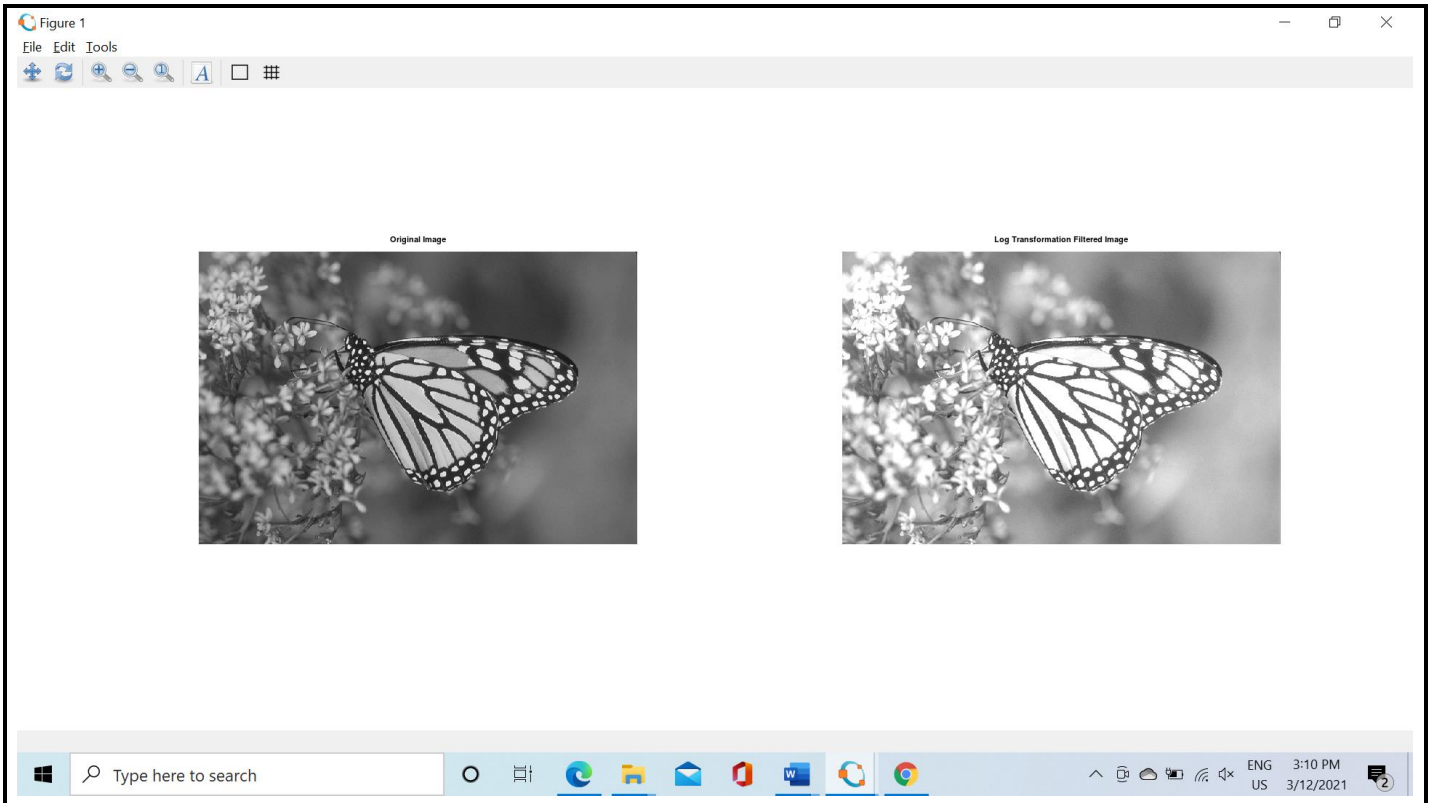
The log transformation maps a narrow range of low intensity values in the input into a wider range of output levels. We use the transformation if this type to extend the values of dark pixel in an image while compress the higher-level values. The general form of the log transformation is: $s = c \log (r + 1)$ Where c is a constant, and $r \geq 0$.

CODE:

```
pkg load image;
clear all;
close all;
x = imread('monarch.png');
x = rgb2gray(x);
x = im2double(x);
[row col] = size(x);
c=2;
for i = 1:row
    for j = 1:col
N(i,j)=c*log(1+x(i,j));
endfor
endfor
subplot(1,2,1);
imshow(x);
title('Original Image');
subplot(1,2,2);
imshow(N);
title('Log Transformation Filtered Image');
```

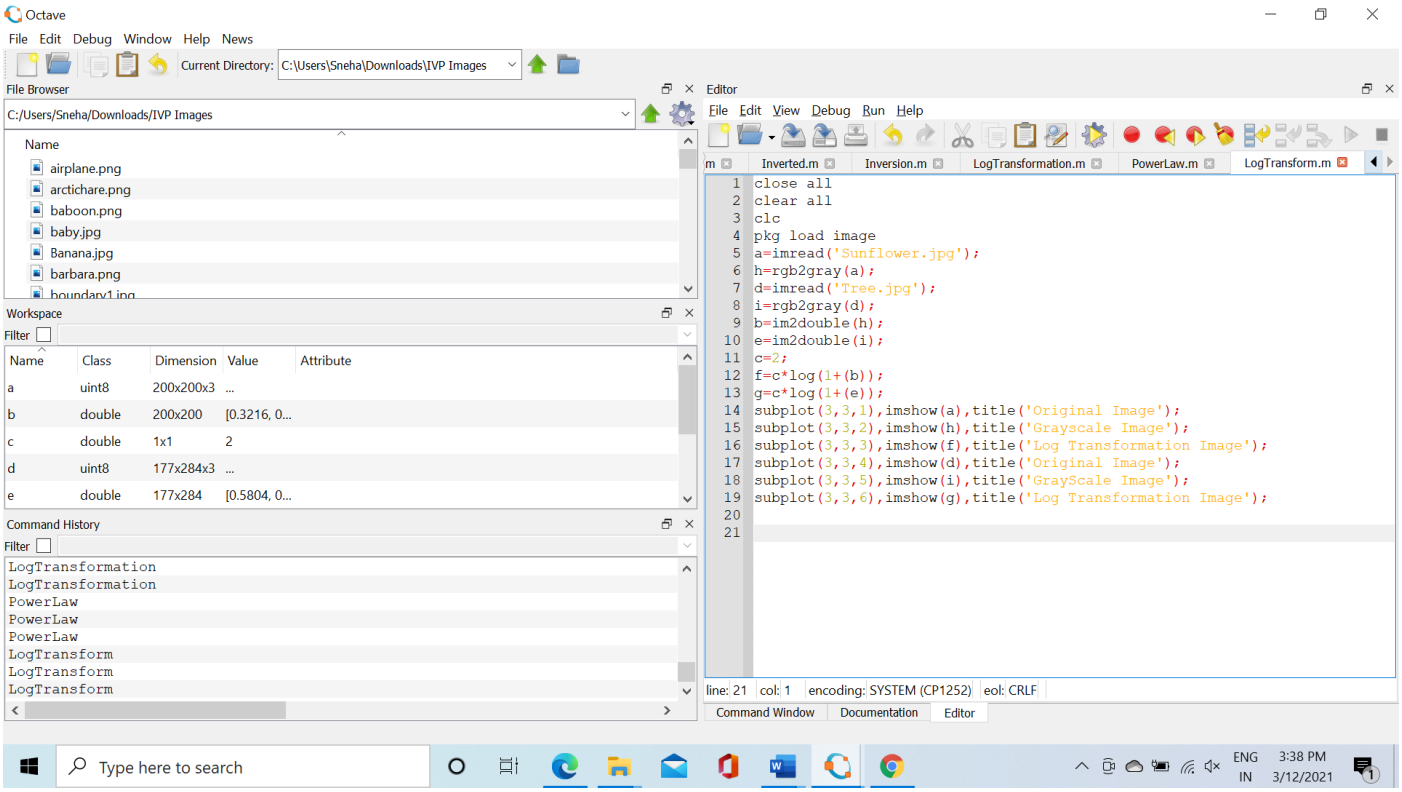


OUTPUT:

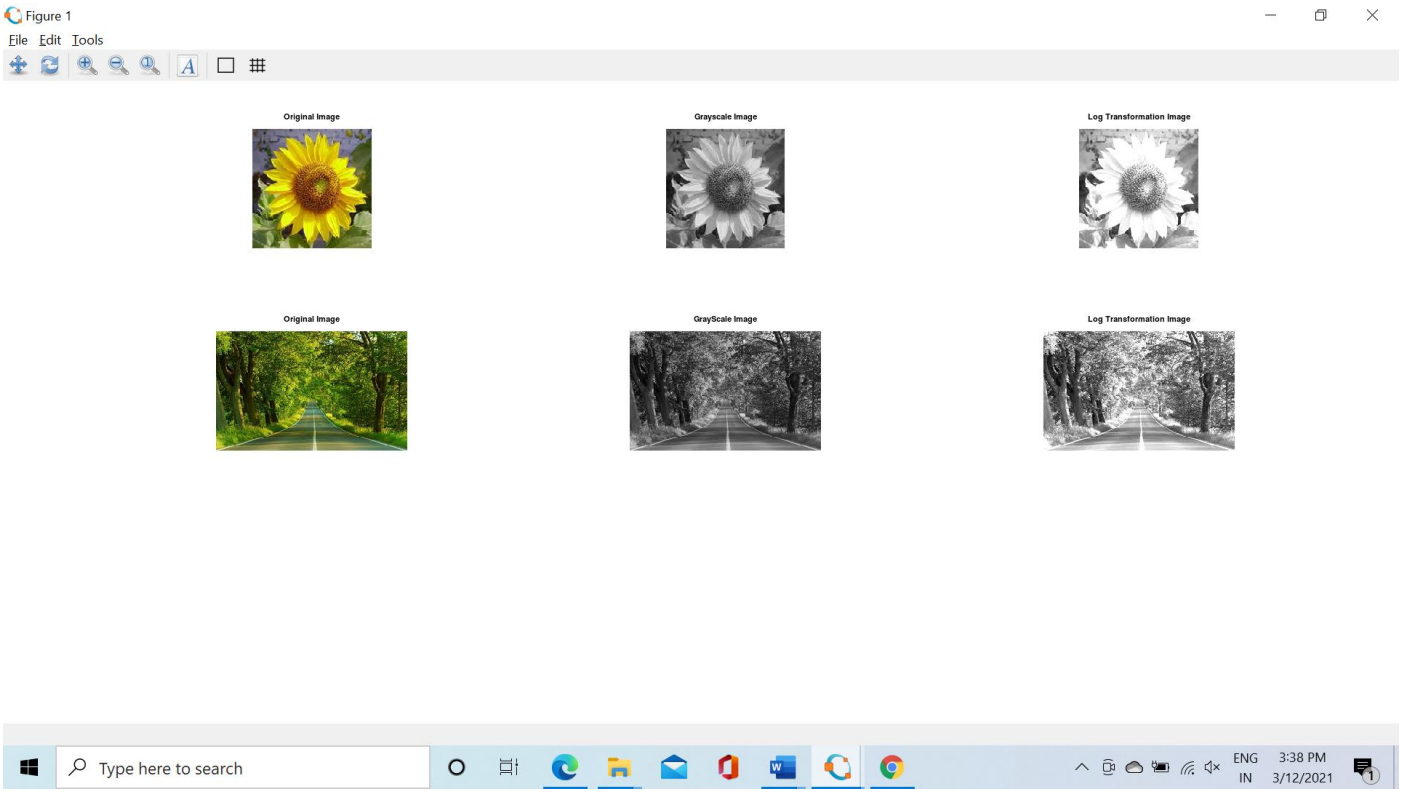


Code :

```
# 1) Implement Basic Intensity Transformation Functions.
#------------------------------------------------------------------------
# 1. B) Log Transformation
close all
clear all
clc
pkg load image
a=imread('Sunflower.jpg');
h=rgb2gray(a);
d=imread('Tree.jpg');
i=rgb2gray(d);
b=im2double(h);
e=im2double(i);
c=2;
f=c*log(1+(b));
g=c*log(1+(e));
subplot(3,3,1),imshow(a),title('Original Image');
subplot(3,3,2),imshow(h),title('Grayscale Image');
subplot(3,3,3),imshow(f),title('Log Transformation Image');
subplot(3,3,4),imshow(d),title('Original Image');
subplot(3,3,5),imshow(i),title('GrayScale Image');
subplot(3,3,6),imshow(g),title('Log Transformation Image');
```



Output :



rgb2gray():

I =rgb2gray(RGB) converts the truecolor image RGB to the grayscale image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation

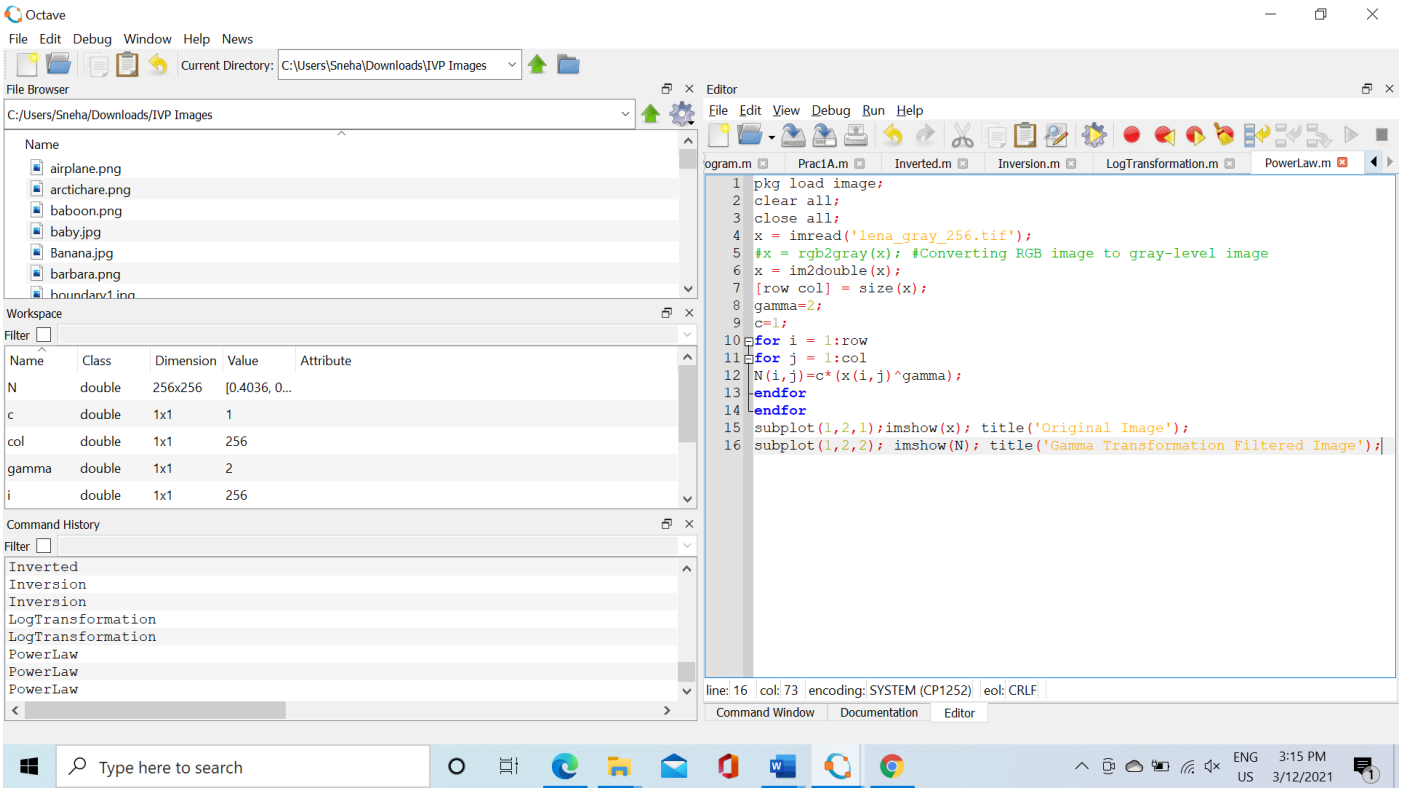
information while retaining the luminance.
imwrite():
imwrite(A,filename) writes image data A to the file specified by filename, inferring the file format from the extension. imwrite creates the new file in your current folder. The bit depth of the output image depends on the data type of A and the file format.

C) Power - law transformation

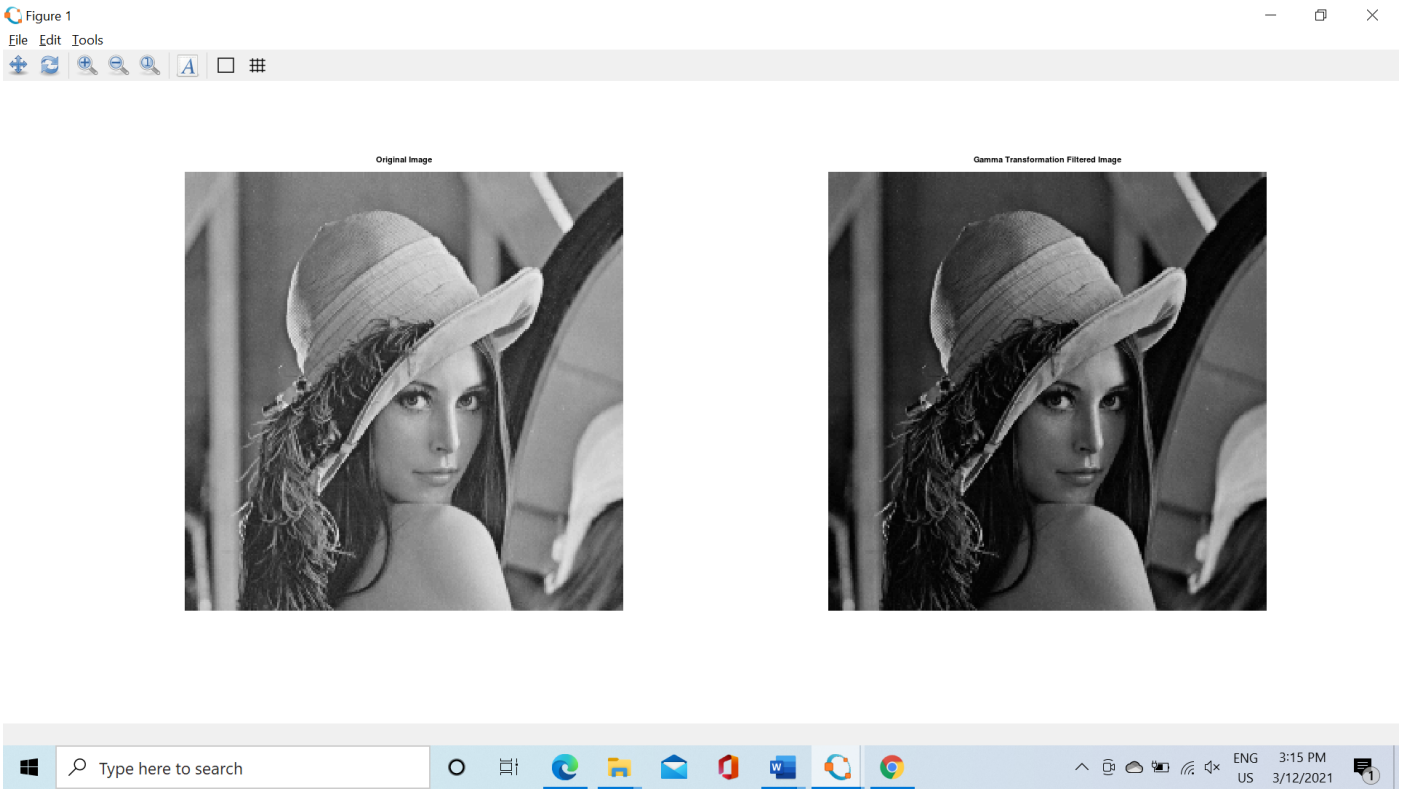
Power-law curves with fractional values of γ map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels. The nth power and nth root curves shown in below figure can be given by the expression as s = c r γ, This transformation function is also called as gamma correction. For various values of γ different levels of enhancements can be obtained. It is used to correct power law response phenomena. The different display monitors display images at different intensities and clarity. That means, every monitor has built-in gamma correction in it with certain gamma ranges and so a good monitor automatically corrects all the images displayed on it for the best contrast to give user the best experience. The gamma variation changes ratio of red green & blue along with intensity in color images. The difference between the log-transformation function and the power-law functions is that using the power-law function a family of
possible transformation curves can be obtained just by varying the λ. This process is also called a gamma correction. The Power Low Transformations can be given by the expression: s = c * r ^ γ where, s is the output pixels value r is the input pixel value c and γ are the real numbers

CODE:

```
pkg load image;
clear all;
close all;
x = imread('lena_gray_256.tif');
#x = rgb2gray(x); #Converting RGB image to gray-level image
x = im2double(x);
[row col] = size(x);
gamma=2;
c=1;
for i = 1:row
for j = 1:col
N(i,j)=c*(x(i,j)^gamma);
endfor
endfor
subplot(1,2,1);imshow(x); title('Original Image');
subplot(1,2,2); imshow(N); title('Gamma Transformation Filtered Image');
```



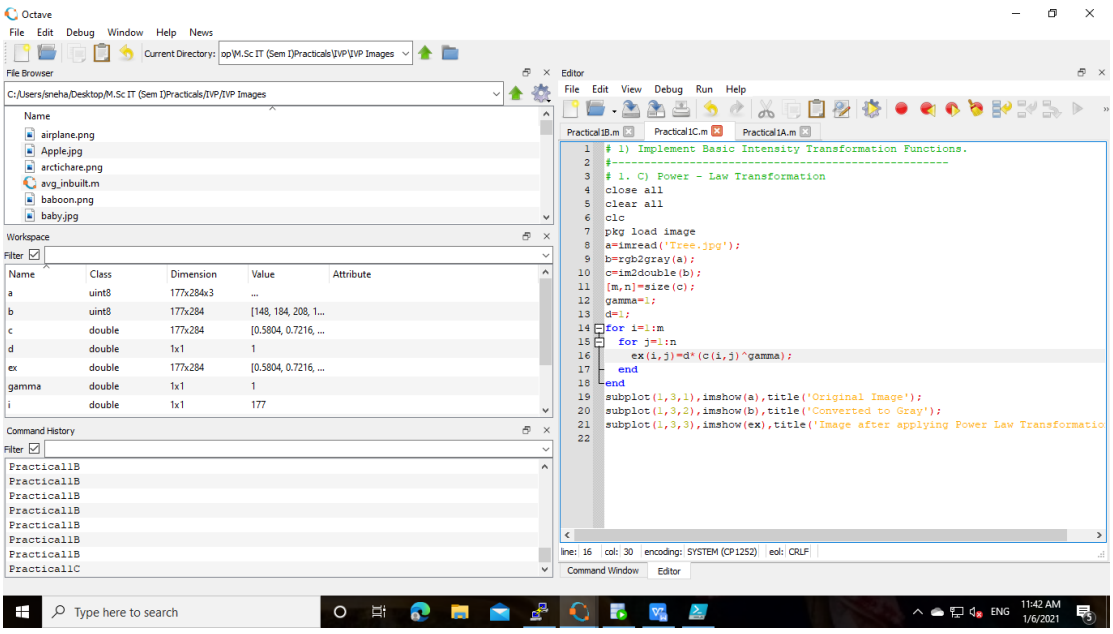
OUTPUT:



Code :

```
# 1) Implement Basic Intensity Transformation Functions.
#--------------------------------------------------------------------
```

```
# 1. C) Power - Law Transformation
close all
clear all
clc
pkg load image
a=imread('Tree.jpg');
b=rgb2gray(a);
c=im2double(b);
[m,n]=size(c);
gamma=1;
d=1;
for i=1:m
    for j=1:n
        ex(i,j)=d*(c(i,j)^gamma);
    end
end
subplot(1,3,1),imshow(a),title('Original Image');
subplot(1,3,2),imshow(b),title('Converted to Gray');
subplot(1,3,3),imshow(ex),title('Image after applying Power Law Transformation');
```



Output :


Figure 1
File Edit Tools
Original Image
Converted to Gray
Image after applying Power Law Transformation
(0.73478, 0.63921)
Type here to search
ENG
11:41 AM
1/6/2021

Practical 2

Implement Piece wise transformation functions.

A) Contrast Stretching

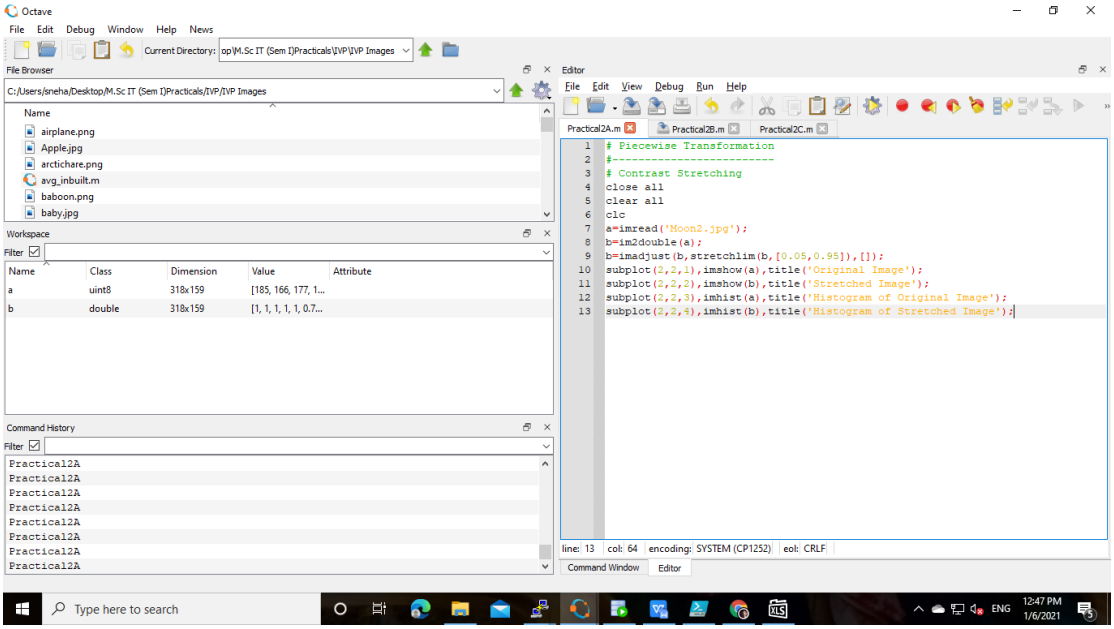
Contrast stretching is also known as normalization. It is a simple image enhancement technique. The quality of image is enhanced by stretching the range of intensity values

imadjust():

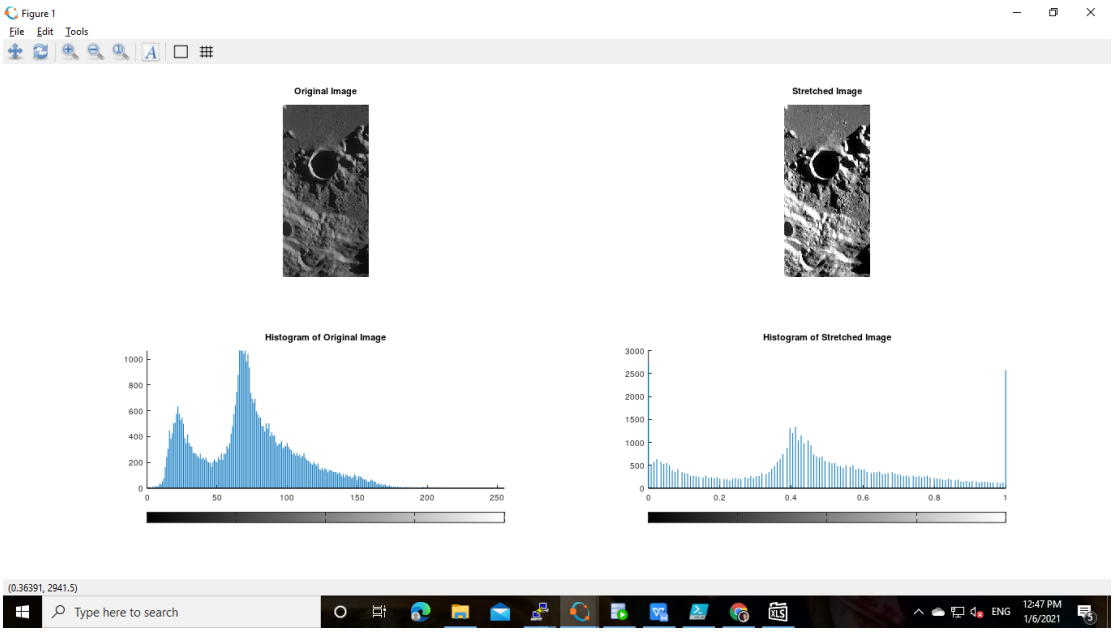
Adjust image or colormap intensity (values). Returns an image of equal dimensions to I, cmap, or RGB, with its intensity values adjusted, usually for the purpose of increasing the image contrast. The values are rescaled according to the input and output limits, low_in and high_in, and low_out and high_out respectively. The first pair sets the lower and upper limits on the input image, values above and below them being clipped. The second pair sets the lower and upper limits for the output image, the interval to which the image will be scaled after clipping the input limits. For example: imadjust (img, [0.2; 0.9], [0; 1]) will clip all values in img outside the range [0.2 0.9], and then rescale them linearly into the range [0 1].

Code :

```
# Piecewise Transformation
#------------------------
# Contrast Stretching using imadjust function
close all
clear all
clc
a=imread('Moon2.jpg');
b=im2double(a);
b=imadjust(b,stretchlim(b,[0.05,0.95]),[]);
subplot(2,2,1),imshow(a),title('Original Image');
subplot(2,2,2),imshow(b),title('Stretched Image');
subplot(2,2,3),imhist(a),title('Histogram of Original Image');
subplot(2,2,4),imhist(b),title('Histogram of Stretched Image');
```



Output :



CODE:

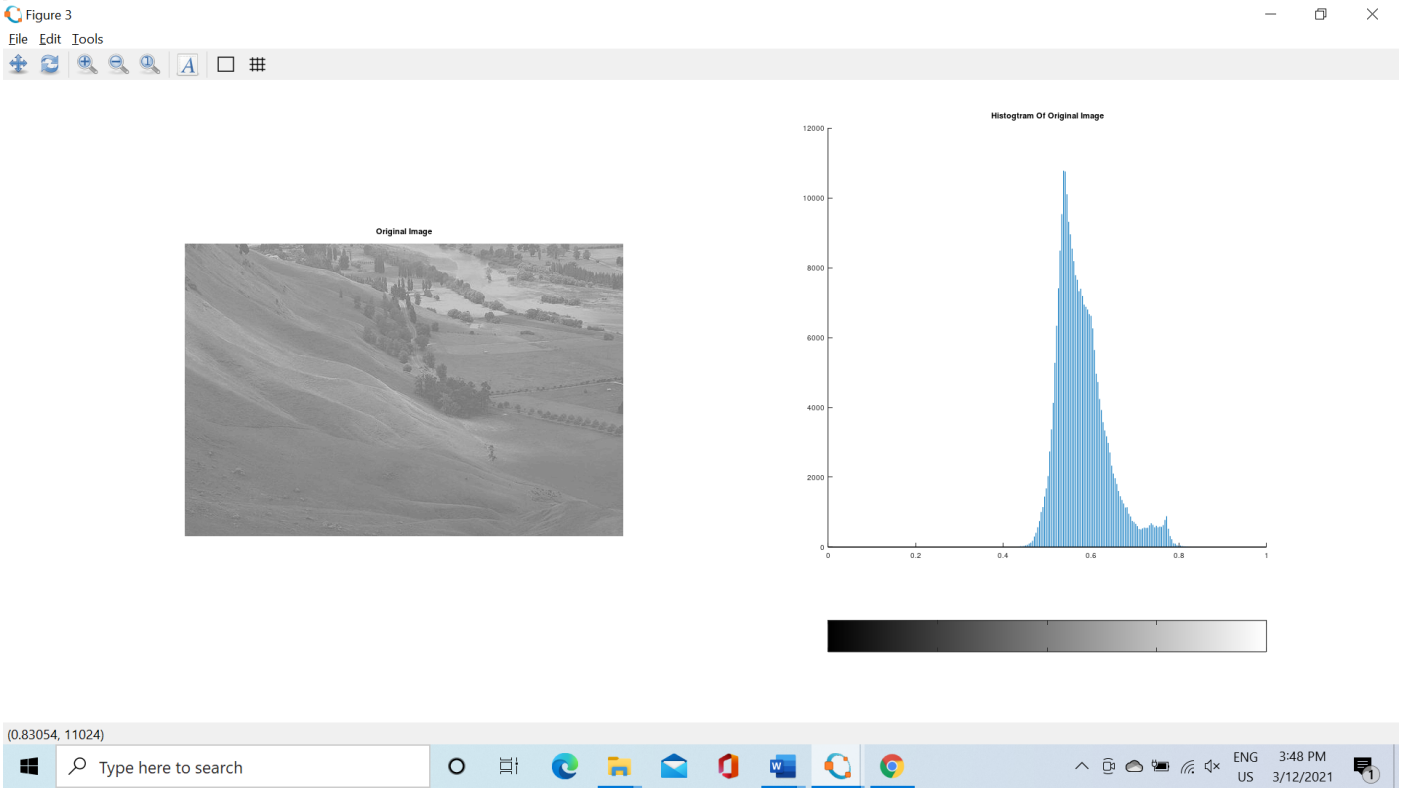
Contrast Stretching Using Inputs from user r1,r2,s1,s2

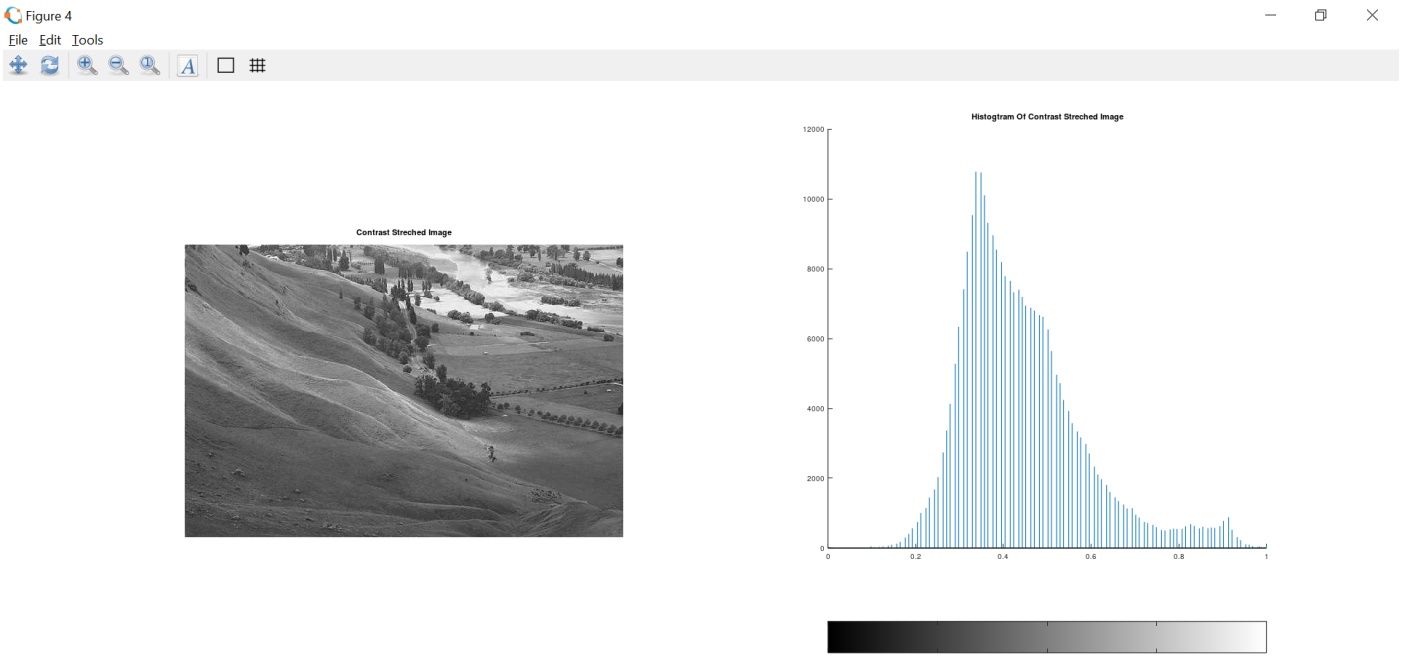
```
pkg load image;
clear all;
close all;
r = imread("fields.jpg");
#r=rgb2gray(r);
r = im2double(r);
```

```
[m n] = size(r); % Getting the dimensions of the image.
#here we are taking 4 input from user
r1=input("Enter R1: ");
r2=input("Enter R2: ");
s1=input("Enter S1: ");
s2=input("Enter S2: ");
#Calculation of contrast stretching
a = s1/r1;
b = (s2-s1)/(r2-r1);
c = (255-s2)/(255-r2);
for i=1:m
  for j=1:n
  if r(i,j) < r1
  s(i,j) = a*r(i,j);
  elseif r(i,j) < r2
  s(i,j) = b*(r(i,j)-r1)+s1;
  else
  s(i,j) = c*(r(i,j)-r2)+s2;
  endif
  endfor
endfor
#Displaying the Original and Contrast Images
figure(3);
subplot(1,2,1)
imshow(r);
title("Original Image");
subplot(1,2,2)
imhist(r);
title('Histogtram Of Original Image');
figure(4);
subplot(1,2,1)
imshow(s);
title("Contrast Streched Image");
subplot(1,2,2)
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) **Thresholding**

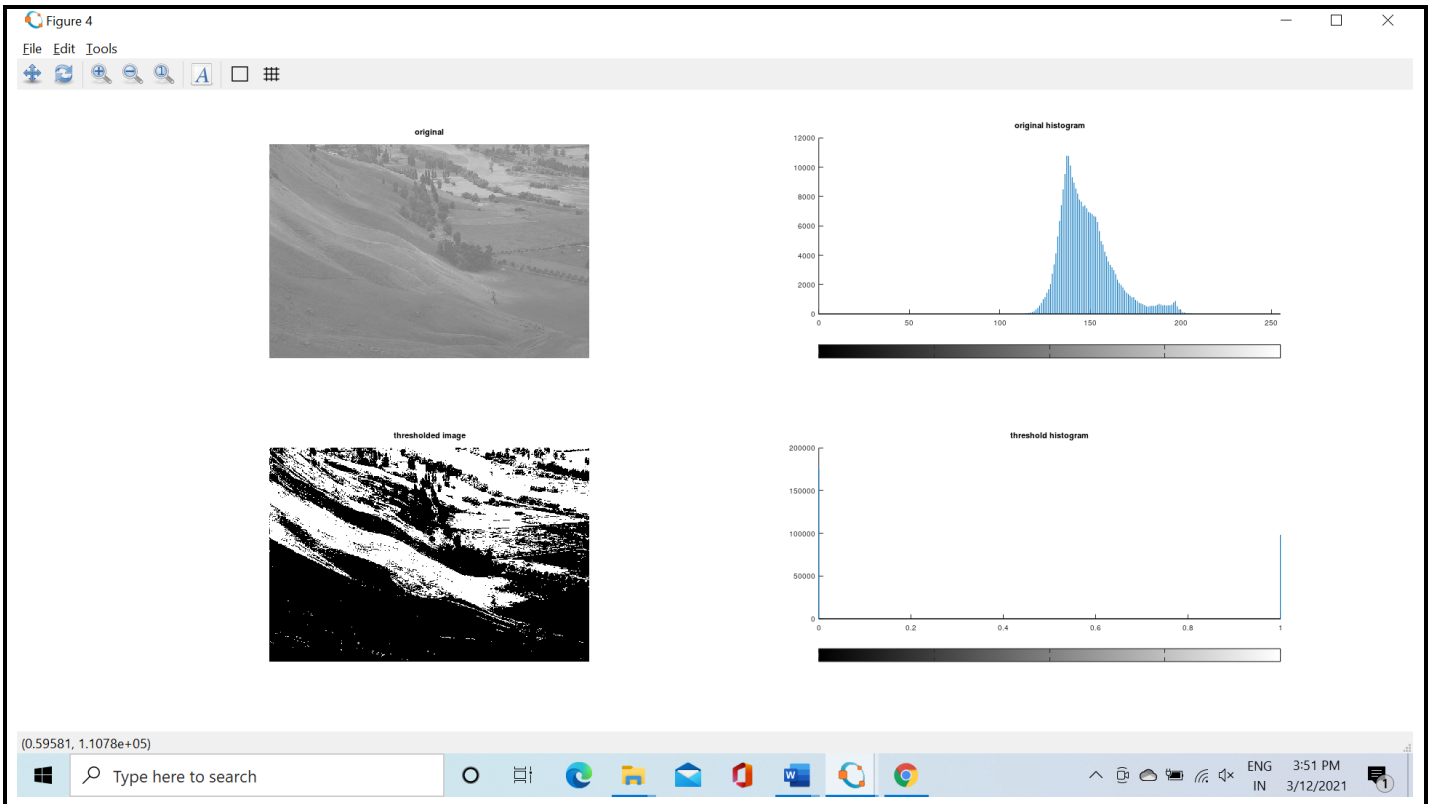
Thresholding is a method of image segmentation, in general it is used to create binary images. Thresholding is of two types namely, simple thresholding and adaptive thresholding.

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. Image thresholding is most effective in images with high levels of contrast.

CODE:

```
pkg load image;
clear all;
r=imread("fields.jpg");
#r=rgb2gray(r);
#r=im2double(r);
imhist(r);
thr=150;
[m n]=size(r);
s=zeros(m,n);
for i=1:m
     for j=1:n
  if(r(i,j))>thr
  s(i,j)=1;
  else
  s(i,j)=0;
  endif
  endfor
endfor
subplot(2,2,1); imshow(r); title("original");
subplot(2,2,2); imhist(r); title("original histogram");
subplot(2,2,3); imshow(s); title("thresholded image");
subplot(2,2,4); imhist(s); title("threshold histogram");
```

OUTPUT:



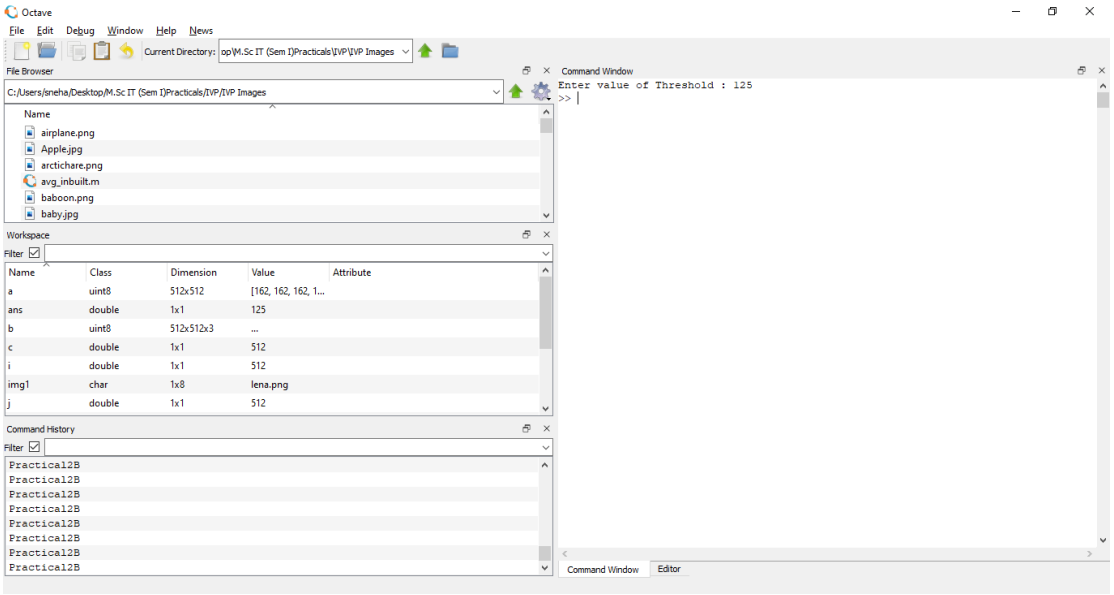
zeros(): B = zeros(n) returns an n-by-n matrix of zeros. An error message appears if n is not a scalar. B = zeros(m,n) or B = zeros([m n]) returns an m-by-n matrix of zeros.

Code :

```
# Piecewise Transformation
#------------------------
# Thresholding
close all;
clear all;
clc;
pkg load image;
[img1 path]= uigetfile('*.*');
b=imread(img1);
a=rgb2gray(b);
[r c p]=size(a);
thr=input('Enter value of Threshold : ');
for i=1:r
   for j=1:c
      if(a(i,j)>thr)
      out(i,j)=1;
```

```
        else
        out(i,j)=0;
        end
    end
end
subplot(3,3,1),imshow(b),title('Input Image');
subplot(3,3,4),imhist(b),title('Histogram of an Input Image');
subplot(3,3,2),imshow(a),title('Image Converted to Gray');
subplot(3,3,5),imhist(a),title('Histogram Gray Scale Image');
subplot(3,3,3),imshow(out),title('Threshold Image');
subplot(3,3,6),imhist(out),title('Histogram of Threshold Image');
```

Output :



C) Bit Plane Slicing

Image reconstruction using n bit planes Code:

Code :

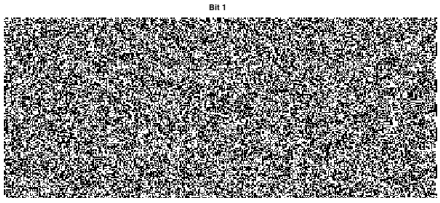
```
pkg load image
A=imread('doller.png');
g=rgb2gray(A);
B=zeros(size(g));
#Getting the bit at specified position#
g1 = bitget(g,1);
g2 = bitget(g,2);
g3 = bitget(g,3);
g4 = bitget(g,4);
g5 = bitget(g,5);
g6 = bitget(g,6);
g7 = bitget(g,7);
g8 = bitget(g,8);
figure,
subplot(2,2,1)
imshow(logical(g1));
title('Bit 1');
subplot(2,2,2)
imshow(logical(g2));
```

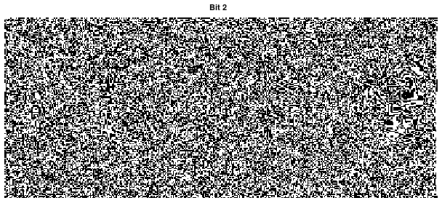
```
title("Bit 2");
subplot(2,2,3)
imshow(logical(g3));
title('Bit 3');
subplot(2,2,4)
imshow(logical(g4));
title('Bit 4');
figure,
subplot(2,2,1)
imshow(logical(g5));
title('Bit 5');
subplot(2,2,2)
imshow(logical(g6));
title("Bit 6");
subplot(2,2,3)
imshow(logical(g7));
title('Bit 7');
subplot(2,2,4)
imshow(logical(g8));
title('Bit 8');
#B=bitset(B,4,bitget(A,4));
B=bitset(B,5,g5);
B=bitset(B,6,g6);
B=bitset(B,7,g7);
B=bitset(B,8,g8);
B=uint8(B);
figure,
subplot(1,2,1),imshow(B); title("5,6,7,8")
subplot(1,2,2),imshow(g); title("original image");
```

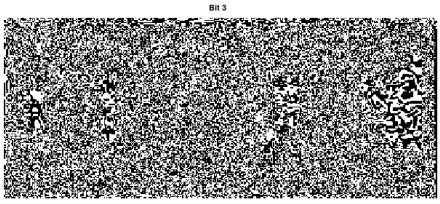
Output :

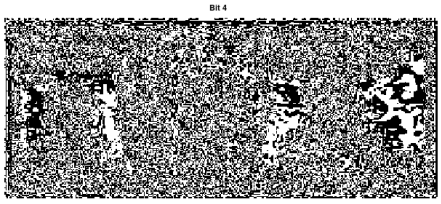
Figure 1

File Edit Tools







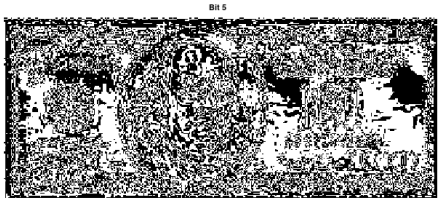


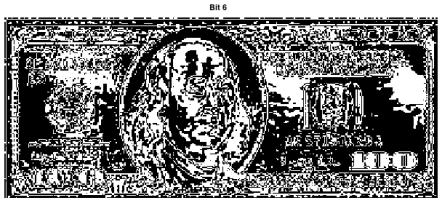
Type here to search

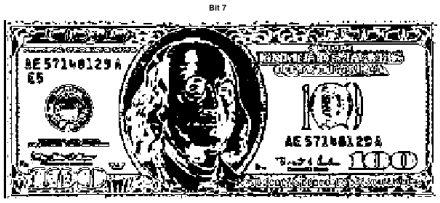
ENG IN 3:56 PM 3/12/2021

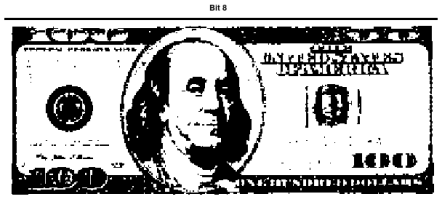
Figure 2

File Edit Tools



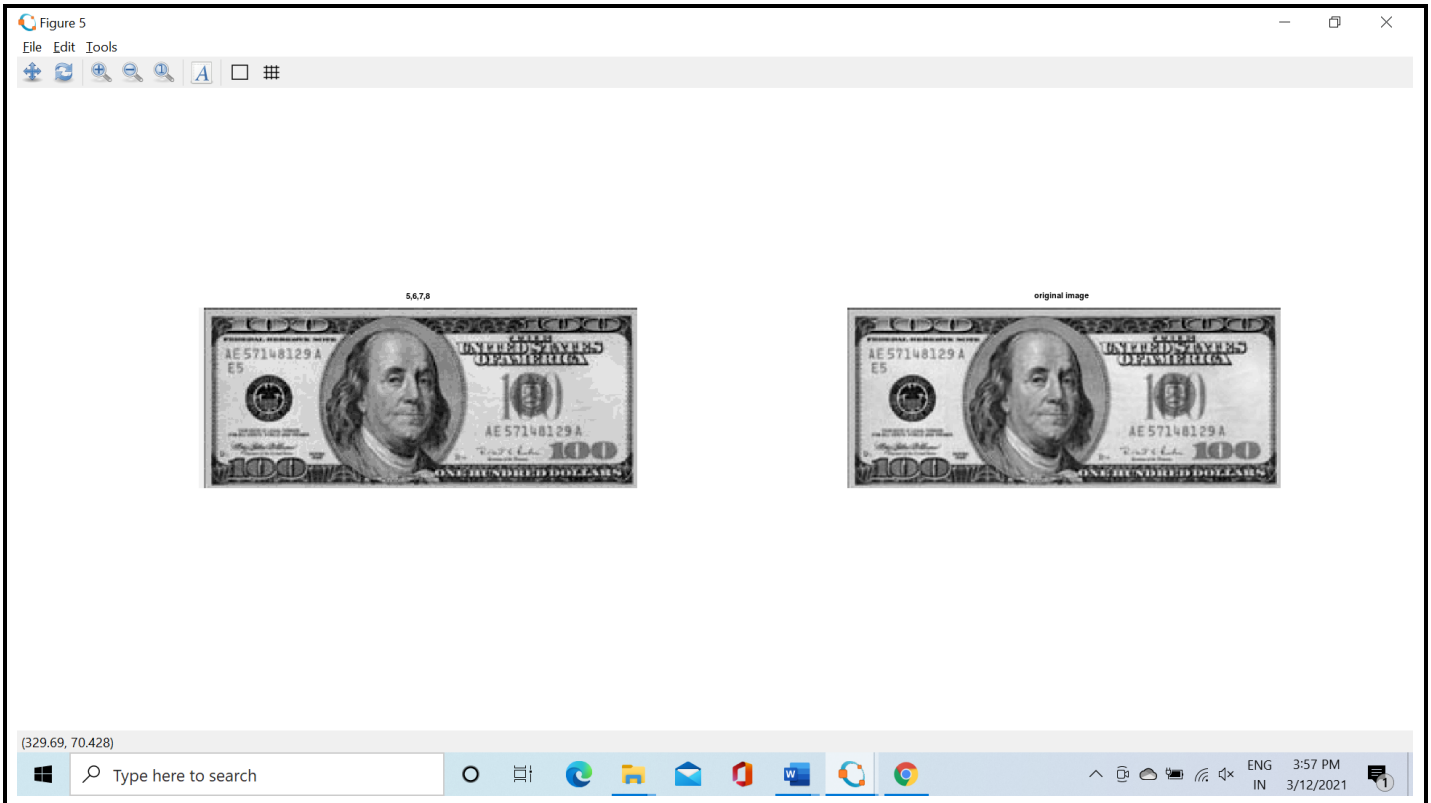






Type here to search

ENG IN 3:56 PM 3/12/2021



The nth plane in the pixels are multiplied by the constant 2^n-1 2. For instance, consider the matrix
A= A=[167 133 111
144 140 135
159 154 148] and the respective bit format

10100111	10000101	01101111
10010000	10001100	10000111
10011111	10011010	10010100

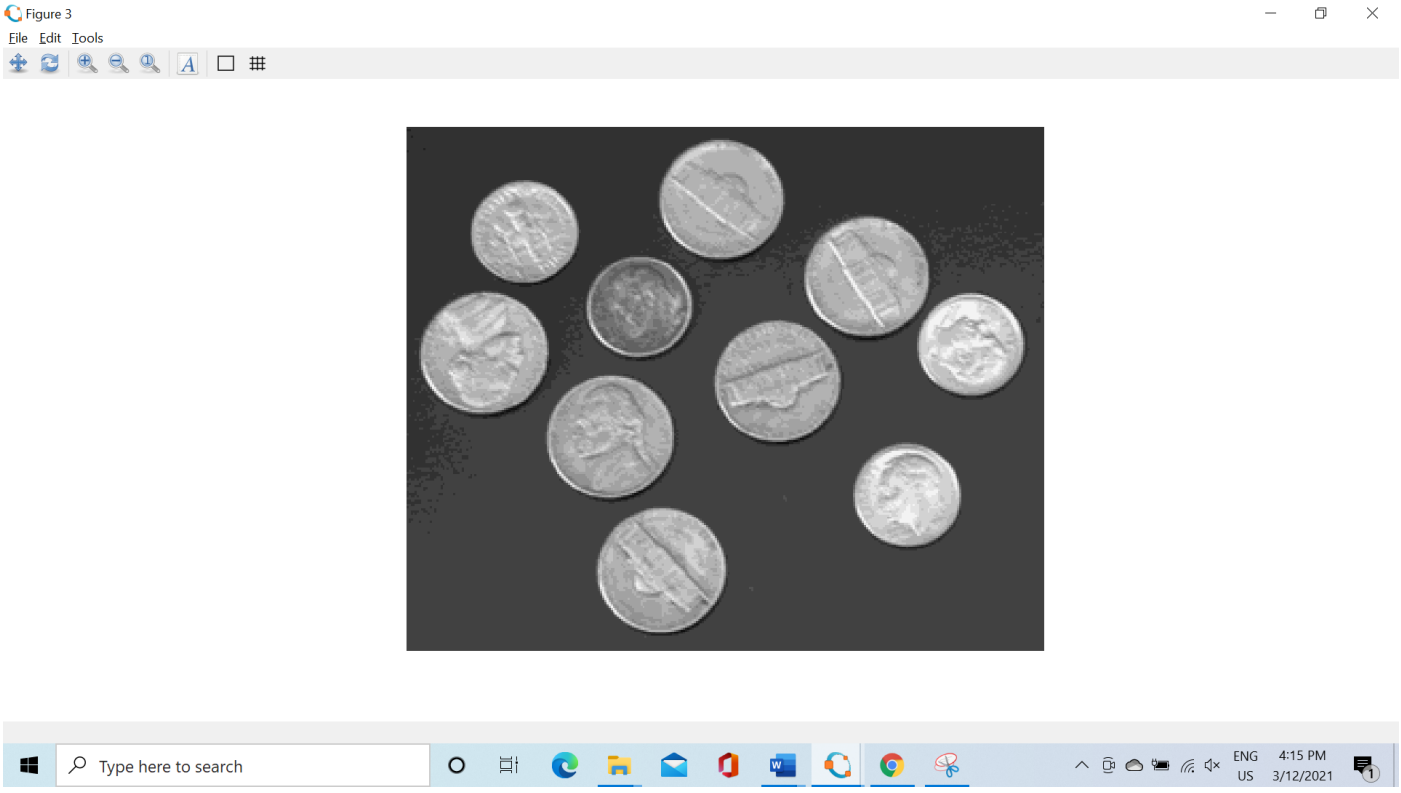
3. Combine the 8 bit plane and 7 bit plane. For 10100111, multiply the 8 bit plane with 128 and 7 bit plane with 64.\ (1x128) + (0x64) + (1x0) + (0x0) + (0x0) + (1x0) + (1x0) + (1x0) = 128
4. Repeat this process for all the values in the matrix and the final result will be
[128 128 64
128 128 128

128 128 128]

CODE:

```
pkg load image;
%Image reconstruction by combining 8 bit plane and 7 bit plane
A=imread('coins.png');
B=zeros(size(A));
B=bitset(B,7,bitget(A,7));
B=bitset(B,8,bitget(A,8));
B=uint8(B);
figure,imshow(B);
%Image reconstruction by combining 8,7,6 and 5 bit planes
A=imread('coins.png');
B=zeros(size(A));
B=bitset(B,8,bitget(A,8));
B=bitset(B,7,bitget(A,7));
B=bitset(B,6,bitget(A,6));
B=bitset(B,5,bitget(A,5));
B=uint8(B);
figure,imshow(B);
```

OUTPUT:



biget():
Get bit at specified position
Syntax
• C = bitget(A, bit)

Description C = bitget(A, bit) returns the value of the bit at position bit in A. Operand A must be an unsigned integer or an array of unsigned integers, and bit must be a number between 1 and the number of bits in the unsigned integer class of A (e.g., 32 for the uint32 class).

bitset():
Set bit at specified position

Syntax

- C = bitset(A, bit)
- C = bitset(A, bit, v)

Description

C = bitset(A, bit) sets bit position bit in A to 1 (on). A must be an unsigned integer or an array of unsigned integers, and bit must be a number between 1 and the number of bits in the unsigned integer class of A (e.g., 32 for the uint32 class).

C = bitset(A, bit, v) sets the bit at position bit to the value v, which must be either 0 or 1

Practical 3

Histogram equalization without using histeq() function.

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.

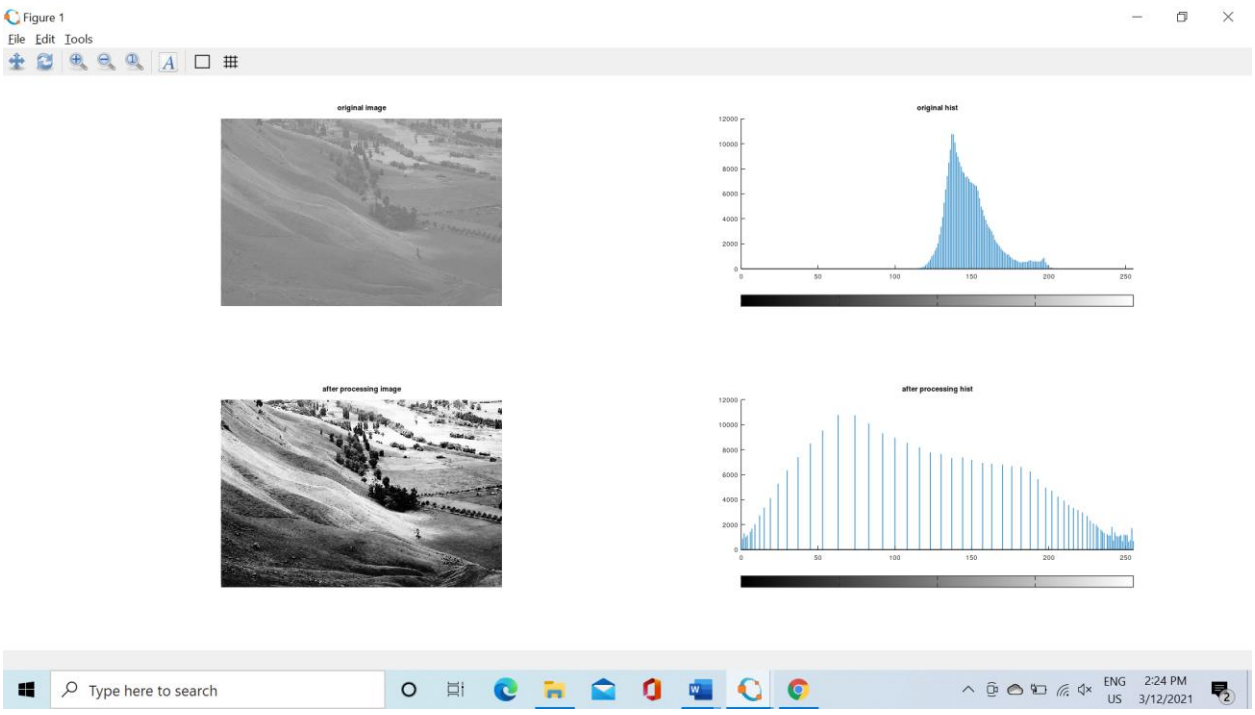
angeljohnsy@blogspot.com

CODE:

```
clear all;
close all;

pkg load image;
a=imread('fields.jpg');
#a=rgb2gray(a);
#a=a(1:10,1:10)
r=size(a,1);
c=size(a,2);
ah=uint8(zeros(r,c));
n=r*c;
f=zeros(256,1);
pdf=zeros(256,1);
cdf=zeros(256,1);
cumm=zeros(256,1);
out=zeros(256,1);
```

```
for i=1:r
   for j=1:c
      values=a(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:r
   for j=1:c
   ah(i,j)=out(a(i,j)+1);
endfor
endfor

figure,
subplot(2,2,1), imshow(a); title('original image');
subplot(2,2,2), imhist(a); title('original hist');
#he=histeq(a);
subplot(2,2,3), imshow(ah); title('after processing image');
 subplot(2,2,4), imhist(ah); title('after processing hist');
 #imhist(he);
```

OUTPUT:



round():

`Y = round(X)` rounds each element of `X` to the nearest integer. In the case of a tie, where an element has a fractional part of exactly `0.5`, the `round` function rounds away from zero to the integer with larger magnitude.

`Y = round(X,N)` rounds to `N` digits:

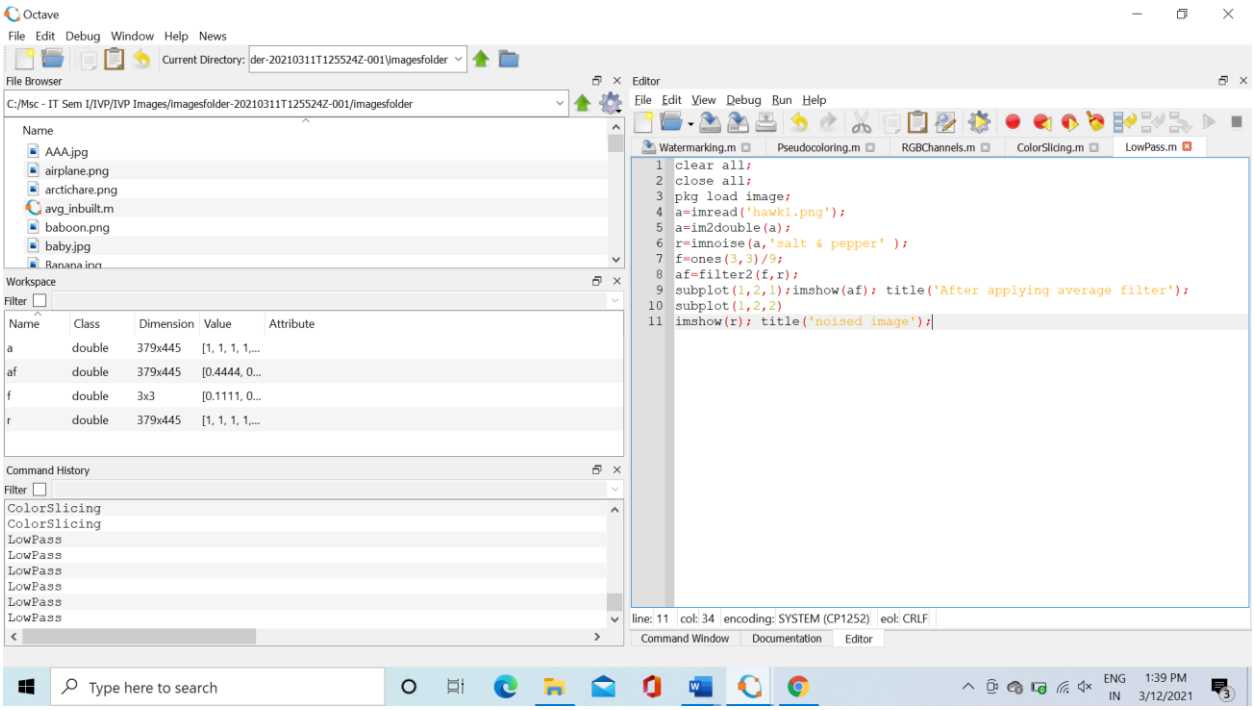
- `N > 0`: round to `N` digits to the *right* of the decimal point.
- `N = 0`: round to the nearest integer.
- `N < 0`: round to `N` digits to the *left* of the decimal point.

Practical 4

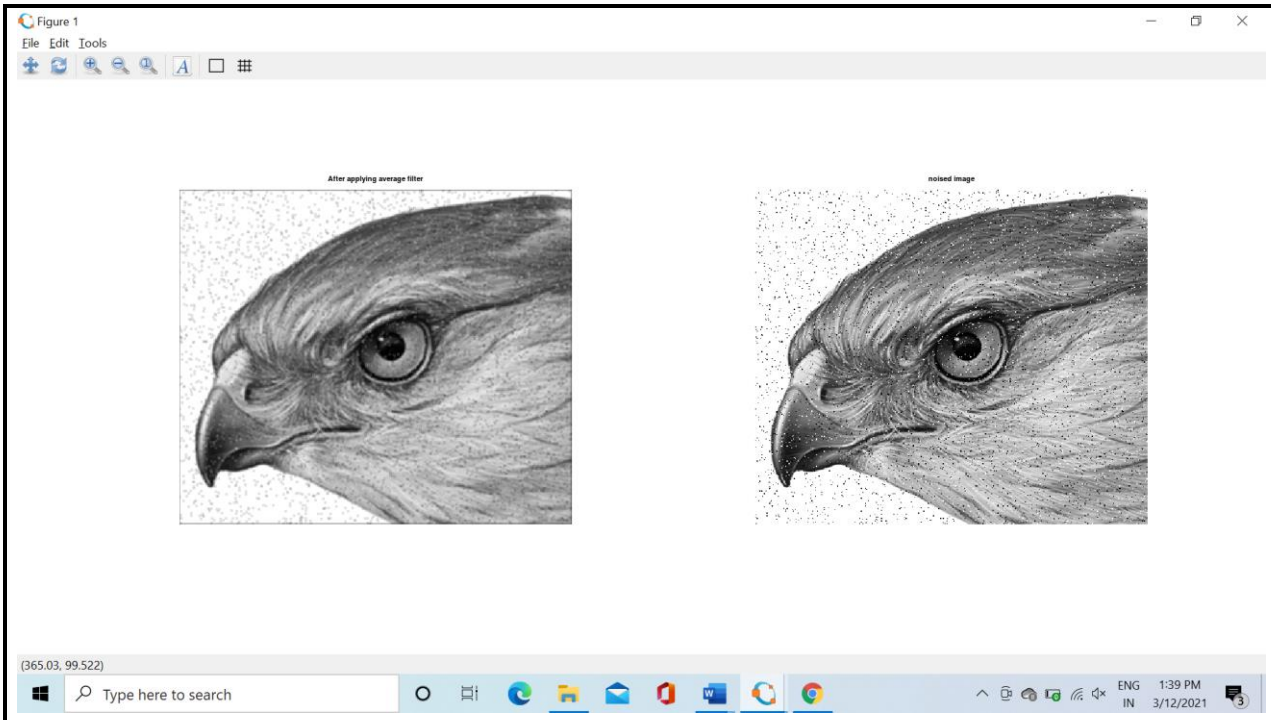
A) (Low Pass-Average filter using inbuilt functions)

CODE:

```
clear all;
close all;
pkg load image;
a=imread('hawk1.png');
a=im2double(a);
r=imnoise(a,'salt & pepper' );
f=ones(3,3)/9;
af=filter2(f,r);
subplot(1,2,1);imshow(af); title('After applying average filter');
subplot(1,2,2)
imshow(r); title('noised image');
```



OUTPUT:



imnoise():

Add noise to an image

Syntax

- **J = imnoise(I,type)**
- **J = imnoise(I,type,parameters)**

Description

J = imnoise(I,type) adds noise of a given type to the intensity image I. type is a string that can have one of these values.

Value	Description
'gaussian'	Gaussian white noise with constant mean and variance
'localvar'	Zero-mean Gaussian white noise with an intensity-dependent variance
'poisson'	Poisson noise
'salt & pepper'	On and off pixels
'speckle'	Multiplicative noise

filter2():
Perform two-dimensional linear filtering.
Y = filter2 (B, X)
Y = filter2 (B, X, SHAPE)
Apply the 2-D FIR filter B to X.
If the argument SHAPE is specified, return an array of the desired shape. Possible values are:

"full"
pad X with zeros on all sides before filtering.

"same"
unpadded X (default)

"valid"
trim X after filtering so edge effects are no included.

B) (Low pass- Average filter without using inbuilt functions)

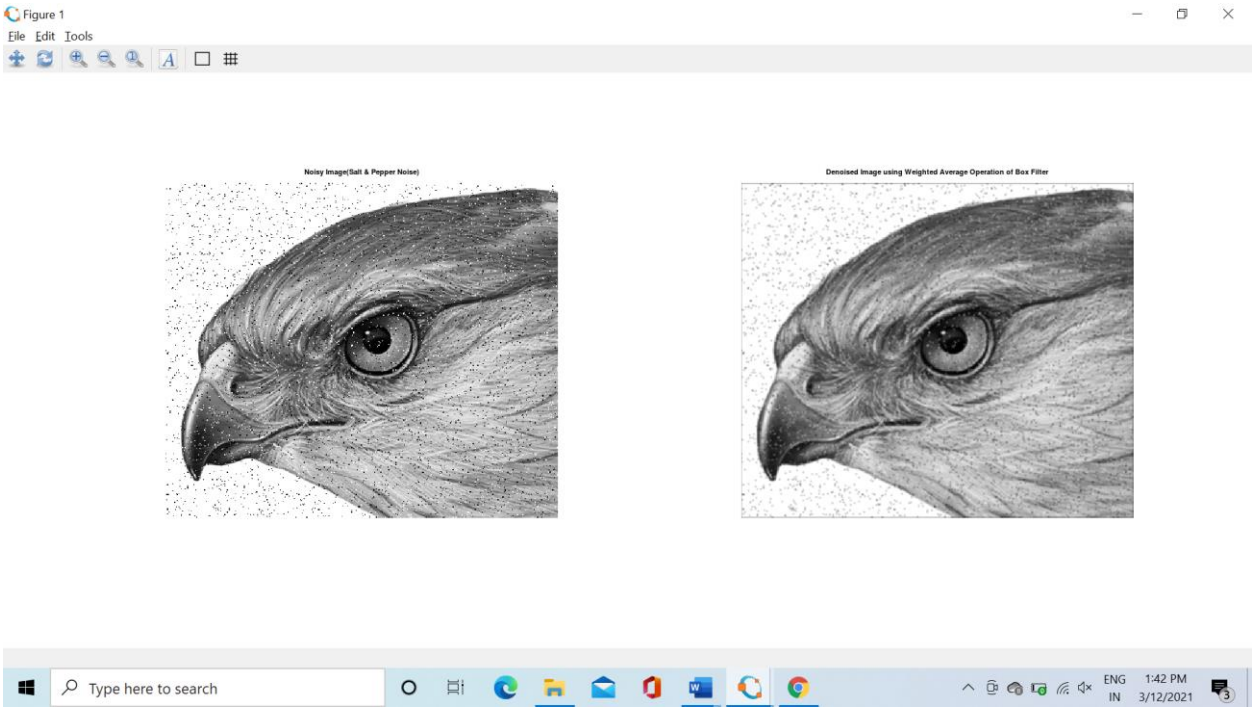
CODE:

```
close all;
pkg load image;
im=imread('hawk1.png'); % To read image
#f=rgb2gray(CIm); % To convert RGB to Grayimage
Nim=imnoise(im,'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(Nim)
[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) * (Nim);
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(Nim);
title('Noisy Image(Salt & Pepper Noise)');
subplot(1,2,2)
imshow(uint8(c));
title('Denoised Image using Weighted Average Operation of Box Filter');
```

OUTPUT:

```
ma = 379
na = 445
mb = 3
nb = 3
size_c =

   381   447
```



floor():

`Y = floor(X)` rounds each element of `X` to the nearest integer less than or equal to that element.

`Y = floor(t)` rounds each element of the `duration` array `t` to the nearest number of seconds less than or equal to that element.

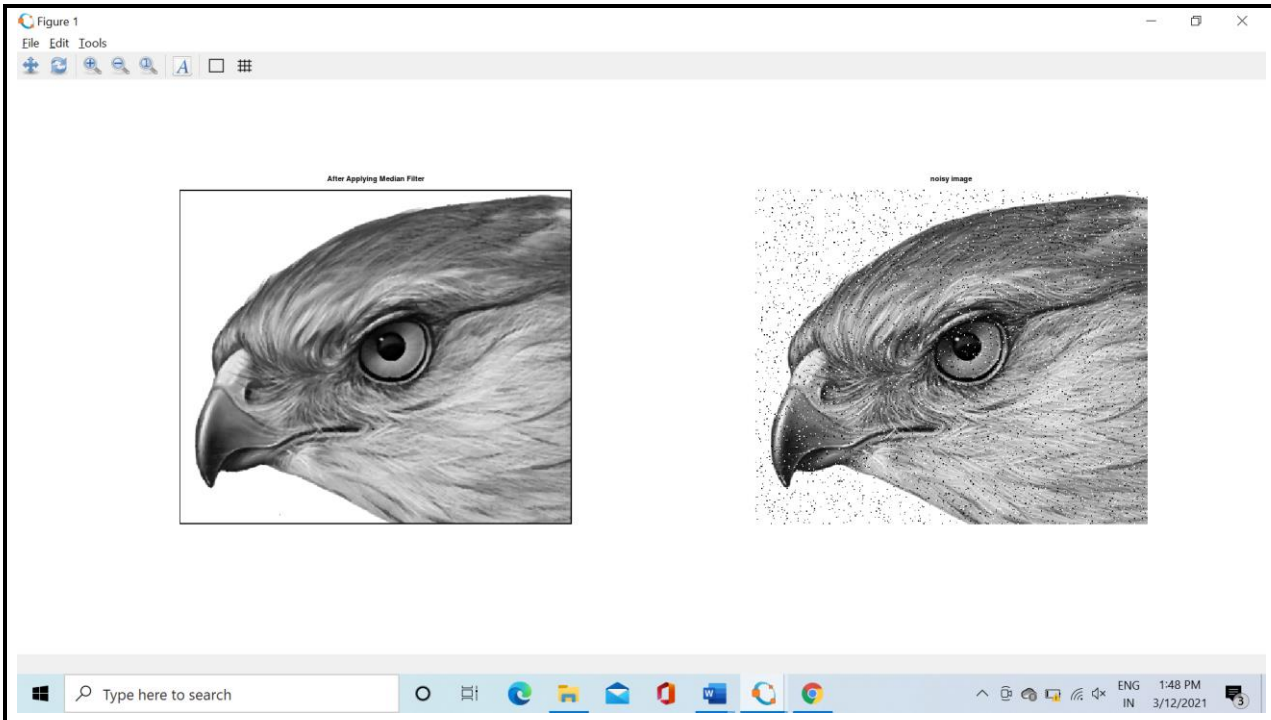
`Y = floor(t,unit)` rounds each element of `t` to the nearest number of the specified unit of time less than or equal to that element.

C) Low Pass Filter (Median):

CODE:

```
pkg load image;
# Read the image
  a=imread('hawk1.png');
  img_noisy1=imnoise(a,'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:



sort():

Sort array elements in ascending or descending order

Syntax

- **B = sort(A)**
- **B = sort(A,dim)**
- **B = sort(...,mode)**
- **[B,IX] = sort(...)**

Description

B = sort(A) sorts the elements along different dimensions of an array, and arranges those elements in ascending order.

If A is a ...	**sort(A) ...**
Vector	Sorts the elements of A.
Matrix	Sorts each column of A.
Multidimensional array	Sorts A along the first non-singleton dimension, and returns an array of sorted vectors.
Cell array of strings	Sorts the strings in ASCII dictionary order.

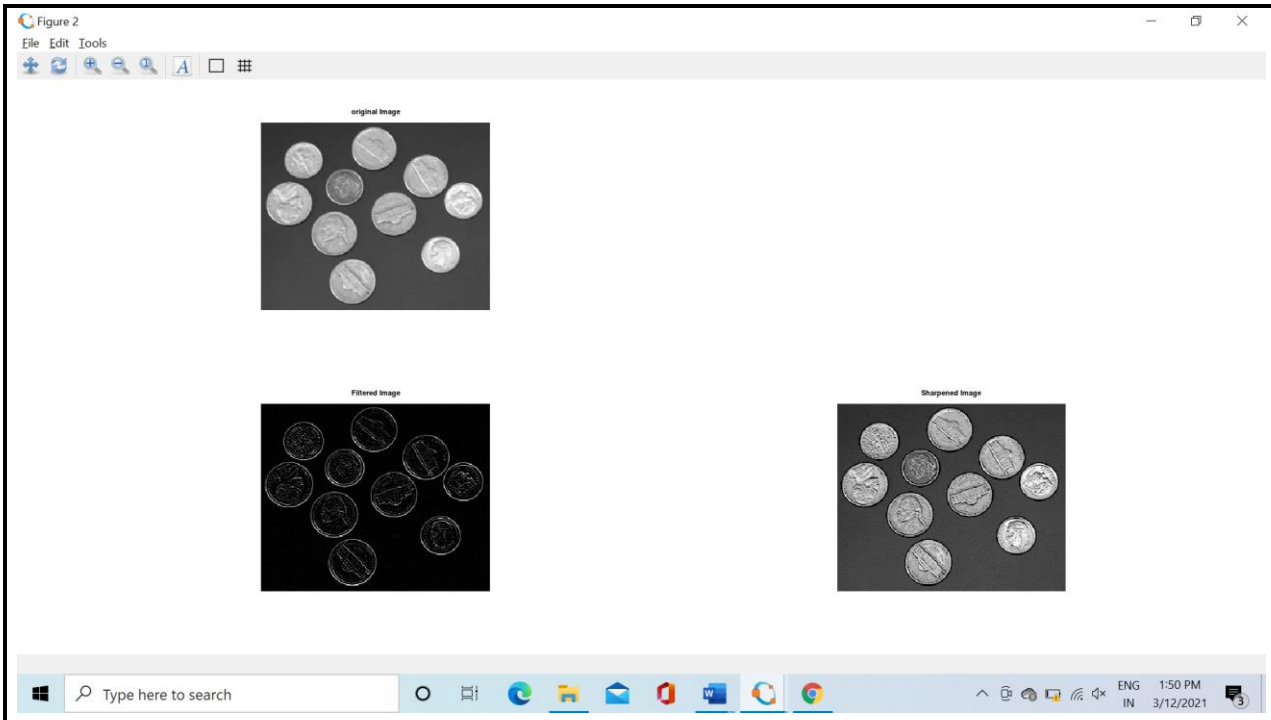
Second Order derivative (Laplacian Filter):

CODE:

Laplacian.m

```
%Input Image
clear all;
A=imread('coins.png');
size(A);
figure,
subplot(2,2,1);imshow(A); title('original Image');
%Preallocate the matrices with zeros
I1=A;
I=zeros(size(A));
I2=zeros(size(A));
%Filter Masks
F1=[0 2 0;2 -8 2; 0 2 0];
#F2=[1 1 1;1 -8 1; 1 1 1];
%Padarray with zeros
A=padarray(A,[1,1]);
A=double(A);
size(A);
%Implementation of the equation in Fig.D
for i=1:size(A,1)-2
    for j=1:size(A,2)-2

        I(i,j)=sum(sum(F1.*A(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:



padarray():
B = padarray(A,padsize) pads array A with an amount of padding in each dimension specified by padsize. The padarray function pads numeric or logical images with the value 0 and categorical images with the category <undefined>. By default, paddarray adds padding before the first element and after the last element of each dimension.
uint8():
8-bit unsigned integer arrays
y = uint8(10);

First Order Derivative -Sobel Operator for edge detection without using edge function

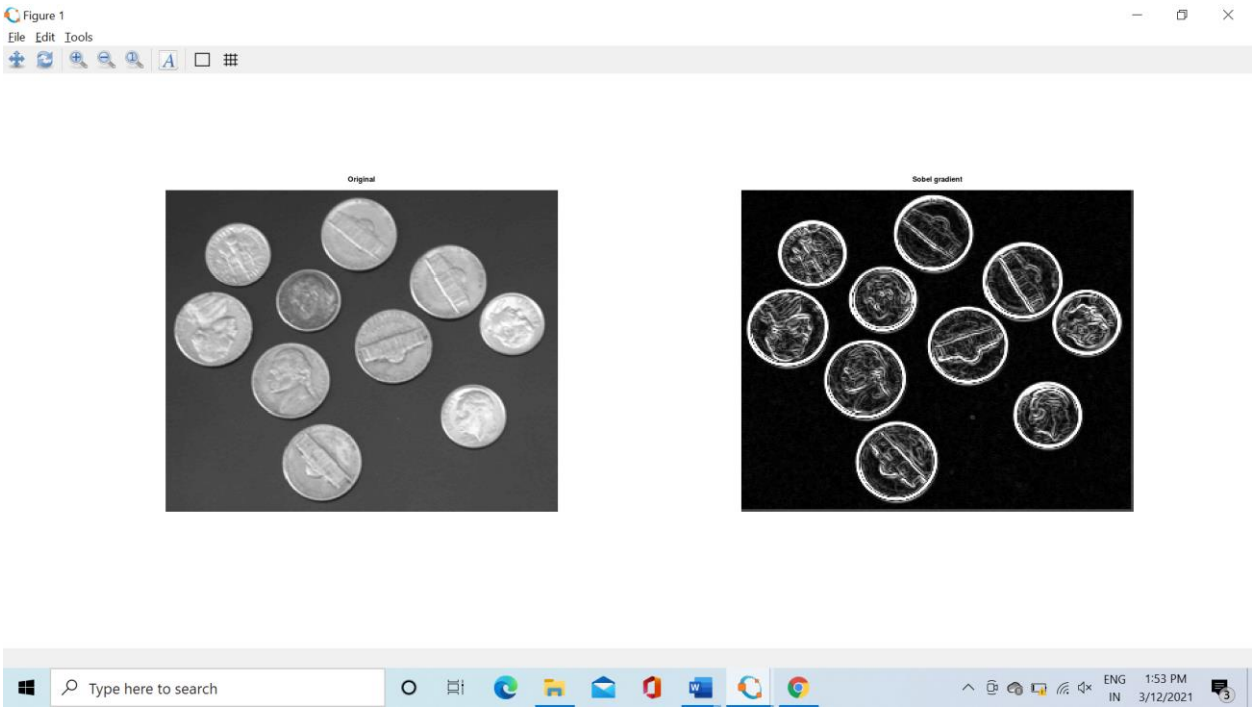
CODE:

FirstOrderDeriv.m

```
clear all;
A=imread('coins.png');
figure,
subplot(1,2,1); imshow(A); title('Original');
C=double(A);
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
        # B(i,j)=abs(Gx)+abs(Gy);
        A(i,j)=sqrt(Gx.^2+Gy.^2);

    end
```

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

OUTPUT:



abs():

Absolute value and complex magnitude

Syntax

- **Y = abs(X)**

Description

abs(X) returns an array Y such that each element of Y is the absolute value of the corresponding element of X.

sqrt():

Square root

Syntax

- **B = sqrt(X)**

Description

B = sqrt(X) returns the square root of each element of the array X. For the elements of X that are negative or complex, sqrt(X) produces complex results.

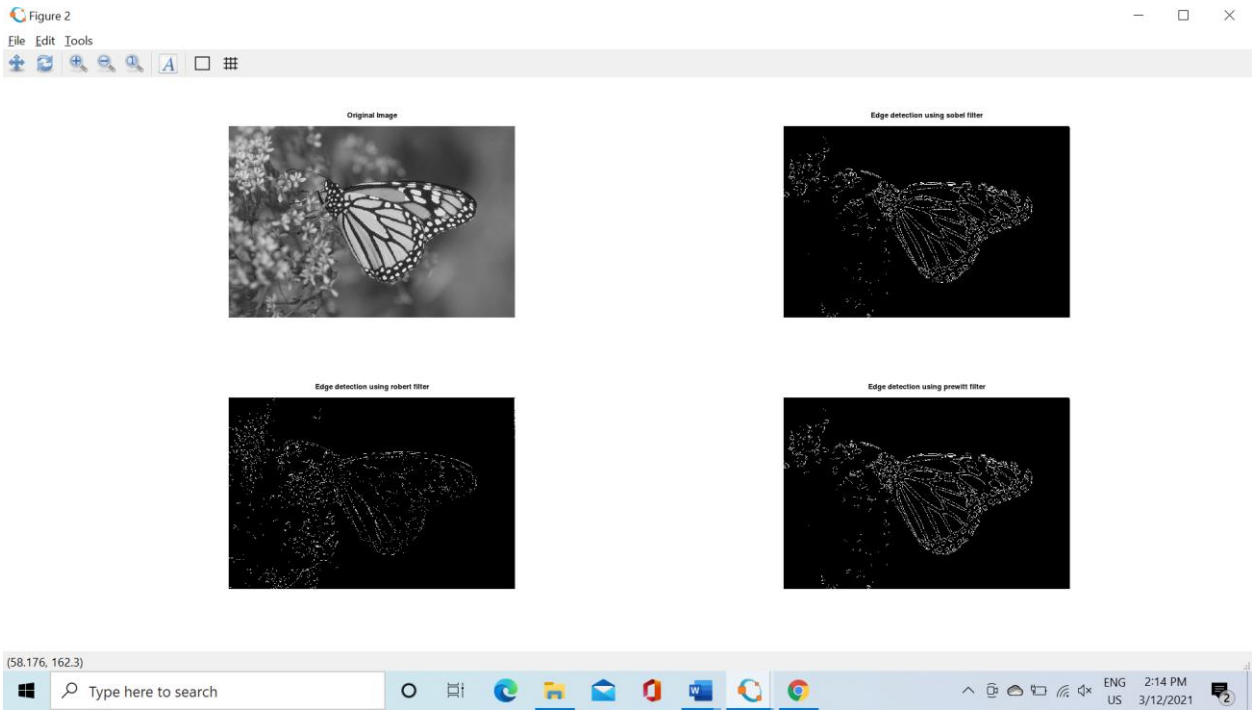
First Order Derivative -Sobel Operator for edge detection using edge function

CODE:

FirstOrderSobel.m

```
#load package of image
pkg load image;
#Take input image
img1=imread('monarch.png');
img=rgb2gray(img1);
#function to find edge using sobel filter
sobel = edge(img,'Sobel');

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:

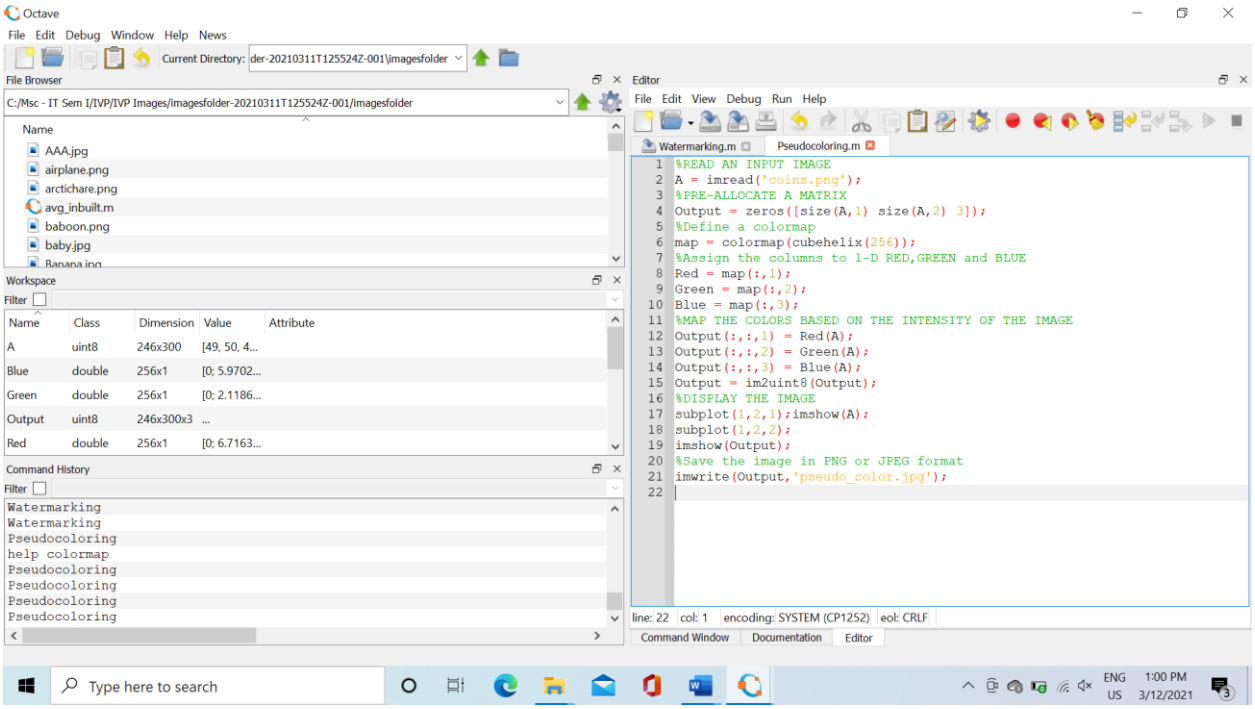


Practical 6

A) Pseudocoloring

CODE:

```
%READ AN INPUT IMAGE
A = imread('coins.png');
%PRE-ALLOCATE A MATRIX
Output = zeros([size(A,1) size(A,2) 3]);
%Define a colormap
map = colormap(cubehelix(256));
%Assign the columns to 1-D RED,GREEN and BLUE
Red = map(:,1);
Green = map(:,2);
Blue = map(:,3);
%MAP THE COLORS BASED ON THE INTENSITY OF THE IMAGE
Output(:,:,1) = Red(A);
Output(:,:,2) = Green(A);
Output(:,:,3) = Blue(A);
Output = im2uint8(Output);
%DISPLAY THE IMAGE
subplot(1,2,1);imshow(A);
subplot(1,2,2);
imshow(Output);
%Save the image in PNG or JPEG format
imwrite(Output,'pseudo_color.jpg');
```



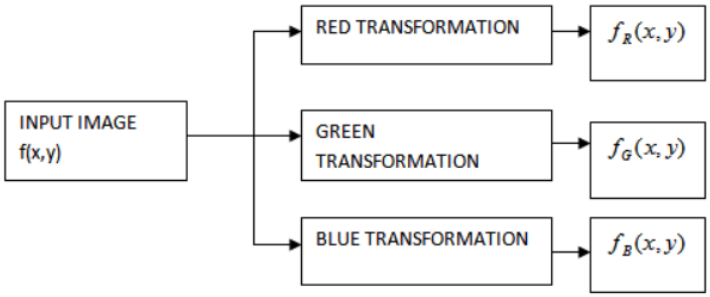
OUTPUT:

colormap():

the purpose of colormap is to define the colors of the graphics objects like image, surface and patch objects. A colormap is basically a matrix with values between 0 & 1. Colormaps can have any length, but width-wise they must have 3 columns. Each row of the matrix defines one color by using an RGB triplet.

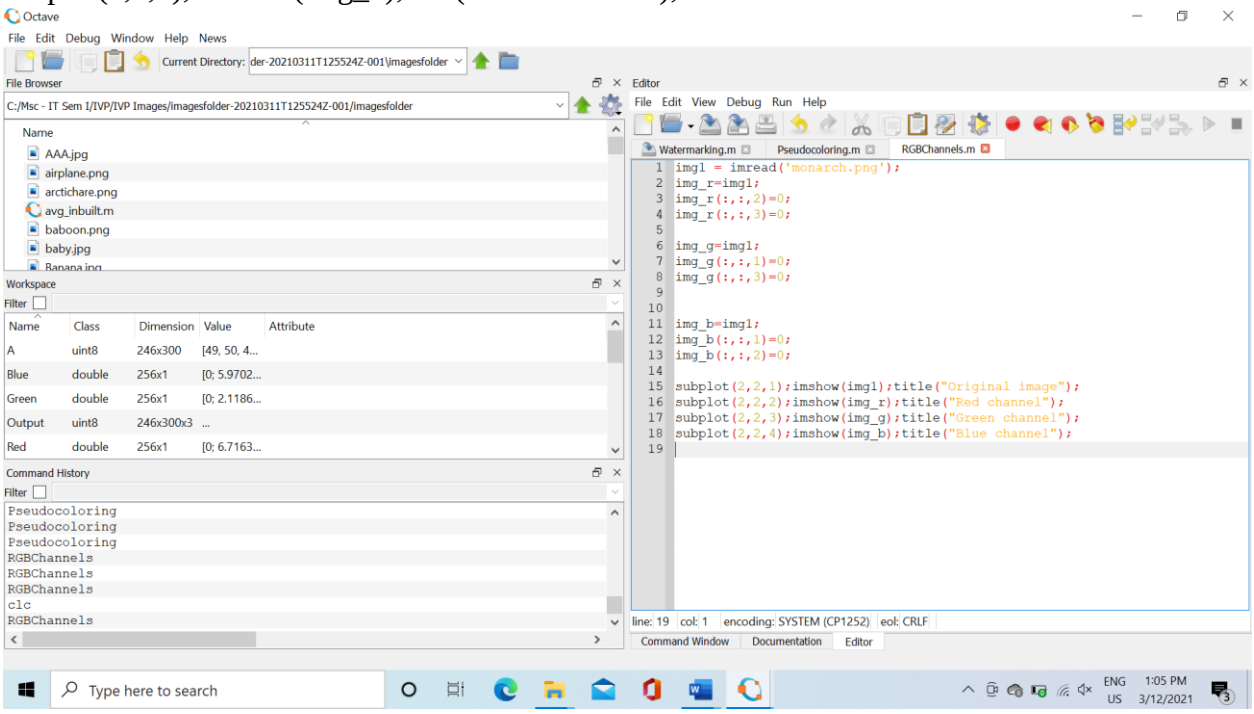
B) Separating the RGB Channels

Functional Block Diagram

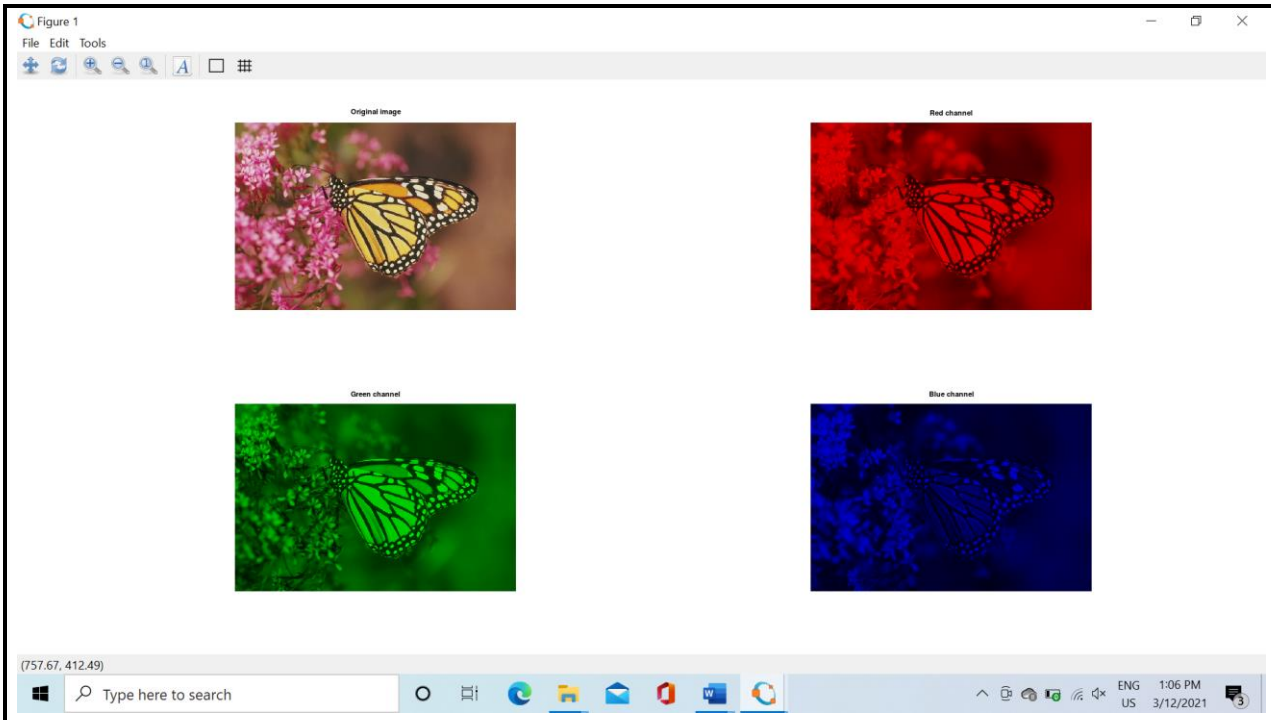


CODE:

```
img1 = imread('monarch.png');
img_r=img1;
img_r(:,:,2)=0;
img_r(:,:,3)=0;

img_g=img1;
img_g(:,:,1)=0;
img_g(:,:,3)=0;


img_b=img1;
img_b(:,:,1)=0;
img_b(:,:,2)=0;

subplot(2,2,1);imshow(img1);title("Original image");
subplot(2,2,2);imshow(img_r);title("Red channel");
subplot(2,2,3);imshow(img_g);title("Green channel");
subplot(2,2,4);imshow(img_b);title("Blue channel");
```



OUTPUT:



C)Color Slicing

CODE:

```
clear all;
#im=input('Enter the file name);
input_image=imread('monarch.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
subplot(1,2,1);
imshow(uint8(k),[]);
title('Original Image');
subplot(1,2,2);
imshow(uint8(m),[]);
title("Pseudo filtred image");
```

OUTPUT:

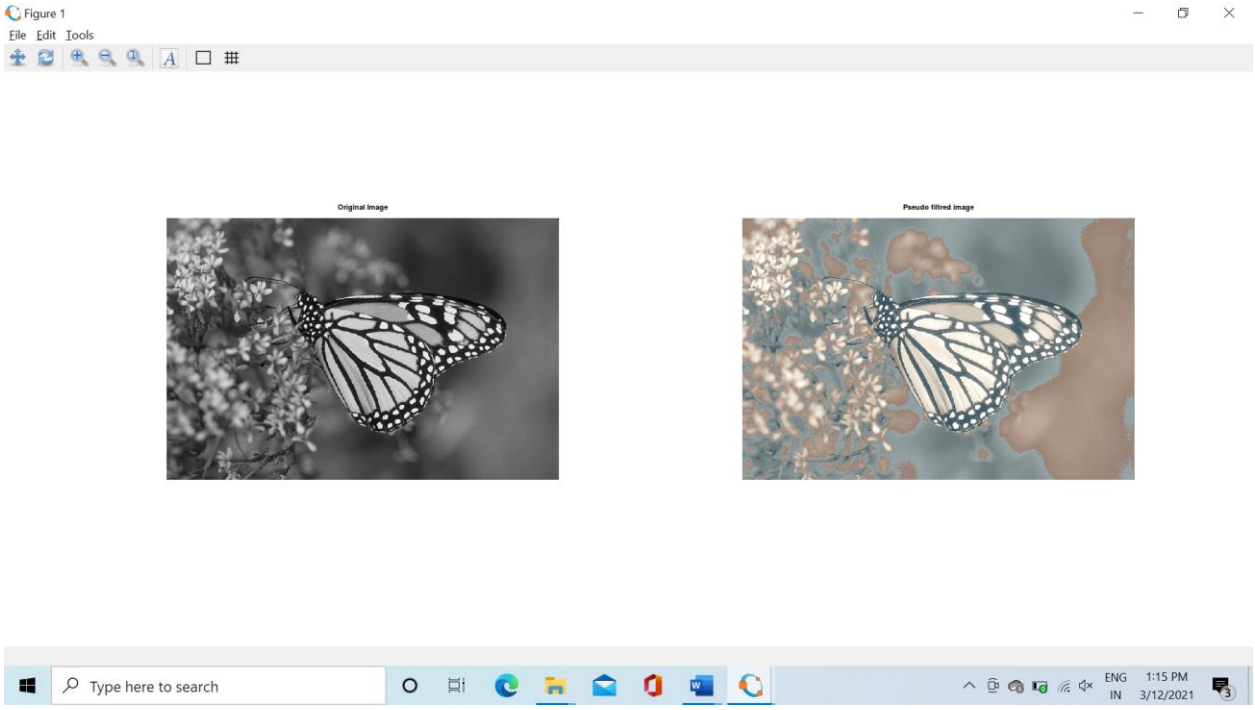


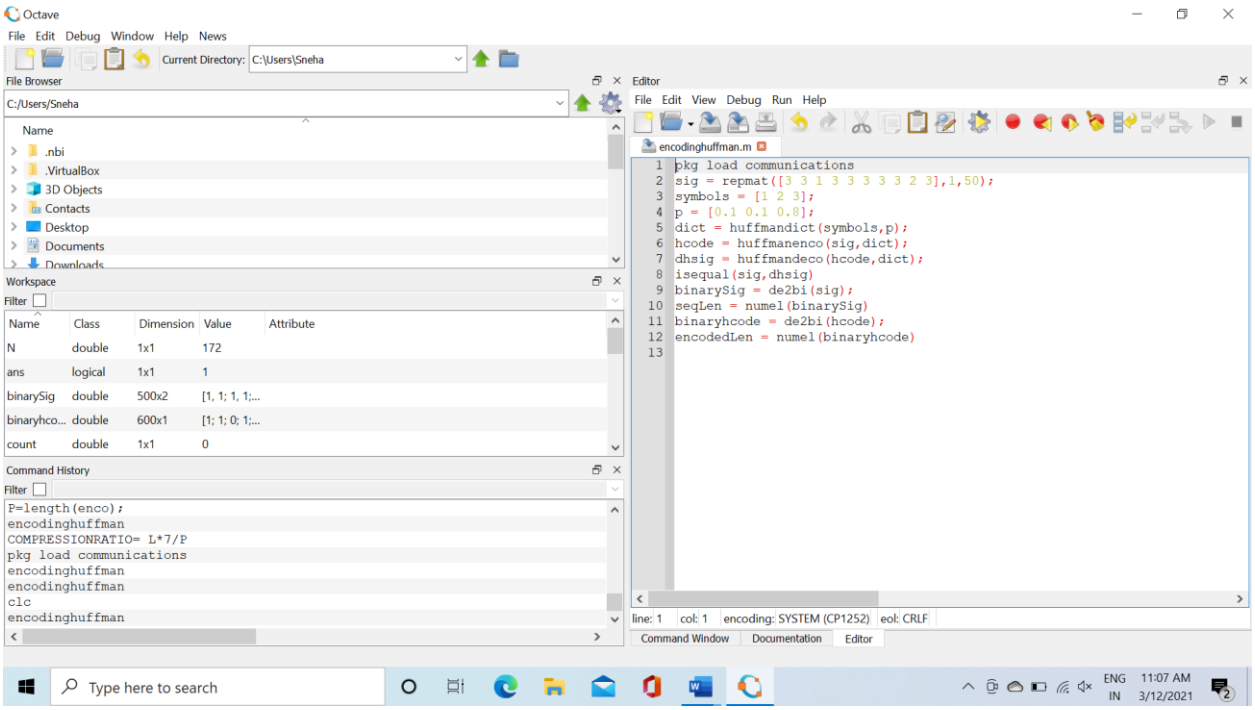
Figure 1
File Edit Tools
Original Image
Pseudo filtred image
Type here to search
ENG
IN
1:15 PM
3/12/2021

Practical 7

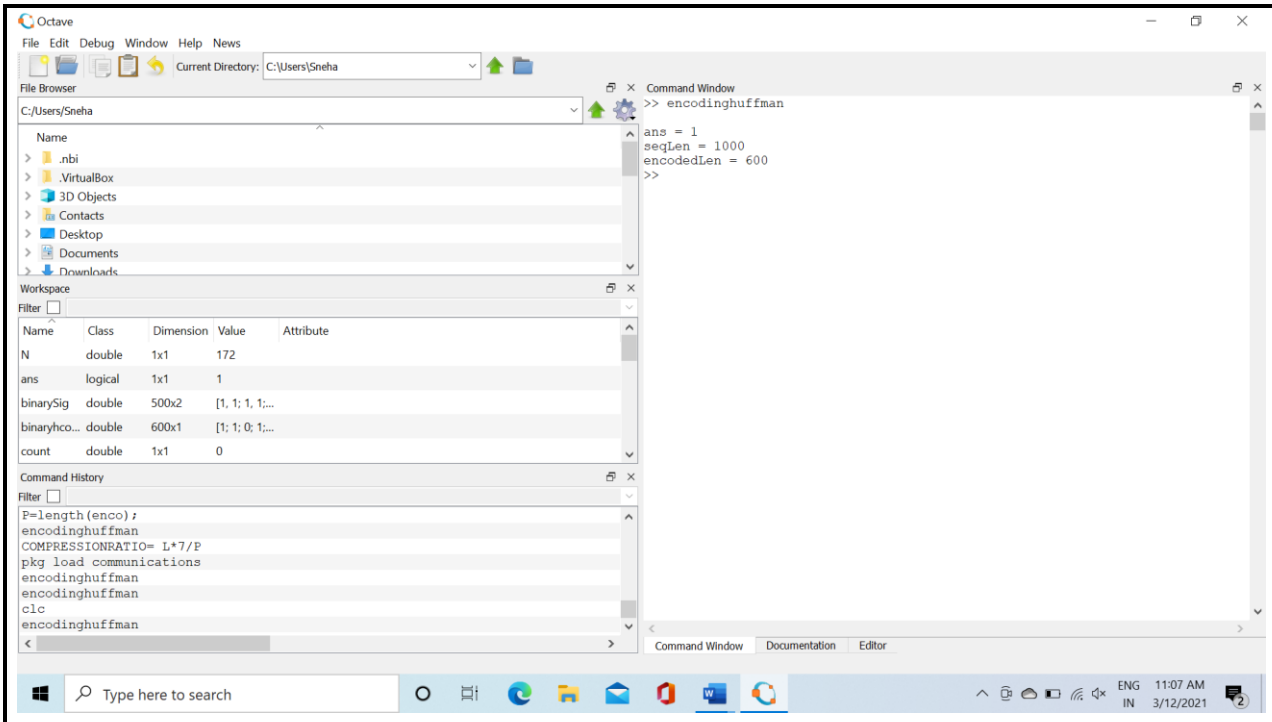
A) Implement Huffman Coding

CODE:

```
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:



repmat():

Repmat command repeats the elements of an array in output. Repetition is depended on parameter list, therefore every time we need to declare parameters within a bracket after repmat command.

Syntax:

Repmat(number,number of times)

Huffmandict():

Generate Huffman code dictionary for source with known probability model.

SYNTAX:

[dict,avglen] = huffmandict(symbols,prob)

huffmandict(symbols,prob) generates a binary Huffman code dictionary, dict, for the source symbols, symbols, by using the maximum variance algorithm. The input prob specifies the probability of occurrence for each of the input symbols. The length of prob must equal the

length of symbols. The function also returns average codeword length avglen of the dictionary, weighted according to the probabilities in the input prob.

Huffmanenco():

Encode sequence of symbols by Huffman encoding

SYNTAX:

code = huffmanenco(sig,dict)

code = huffmanenco(sig,dict) encodes input signal sig using the Huffman codes described by input code dictionary dict. sig can have the form of a vector, cell array, or alphanumeric cell array. If sig is a cell array, it must be either a row or a column. dict is an N-by-2 cell array, where *N* is the number of distinct possible symbols to encode. The first column of dict represents the distinct symbols and the second column represents the corresponding codewords. Each codeword is represented as a row vector, and no codeword in dict can be the prefix of any other codeword in dict. You can generate dict using the huffmandict function.

Huffmandeco():

Decode binary code by Huffman decoding.

SYNTAX:

sig = huffmandeco(code,dict)

sig = huffmandeco(code,dict) decodes the numeric Huffman code vector, code, by using the Huffman codes described by input code dictionary dict. Input dict is an *N*-by-2 cell array, where *N* is the number of distinct possible symbols in the original signal that encodes code. The first column of dict represents the distinct symbols, and the second column represents the corresponding codewords. Each codeword is represented as a numeric row vector, and no codeword in dict can be the prefix of any other codeword in dict. You can generate dict by using the huffmandict function and code by using the huffmanenco function. If all symbols in dict are numeric, output sig is a vector. If any symbol in dict is alphabetic, sig is a one-dimensional cell array.

de2bi():

Convert decimal numbers to binary vectors.

SYNTAX:

b = de2bi(d)

converts a nonnegative decimal integer d to a binary row vector. If d is a vector, the output b is a matrix in which each row is the binary form of the corresponding element in d.

numel():

Number of elements in array or subscripted array expression

Syntax

- **n = numel(A)**
- **n = numel(A,varargin)**

Description

n = numel(A) returns the number of elements, n, in array A.

n = numel(A,varargin) returns the number of subscripted elements, n, in A(index1,index2,...,indexn), where varargin is a cell array whose elements are index1, index2, ..., indexn.

B) **Watermarking**

CODE:

```
pkg load image;
clear all;
close all;
#Input Image where we want to apply watermark
f=imread('fruits.png');

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

fr=imresize(f,[560 560]);
subplot(1,3,1);imshow(fr);
title('Original Image with resized');

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

#Applied watermarking

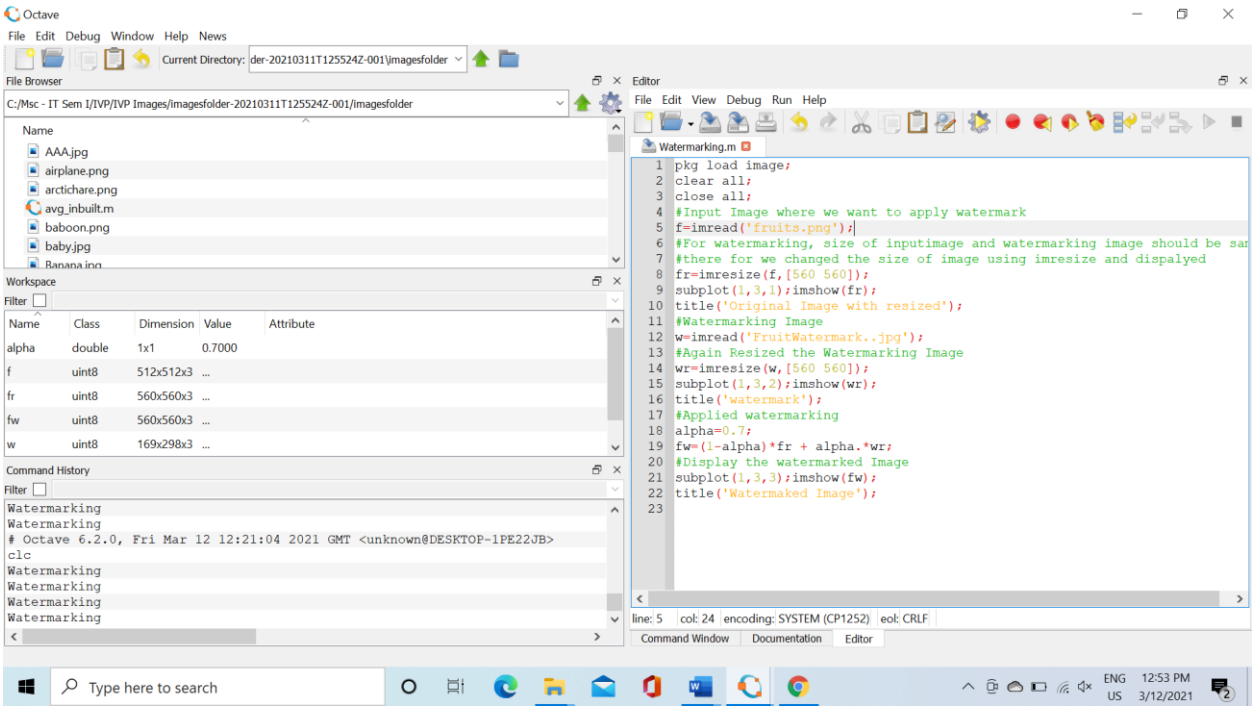
alpha=0.7;

fw=(1-alpha)*fr + alpha.*wr;

#Display the watermarked Image

subplot(1,3,3);imshow(fw);

title('Watermaked Image');



OUTPUT:

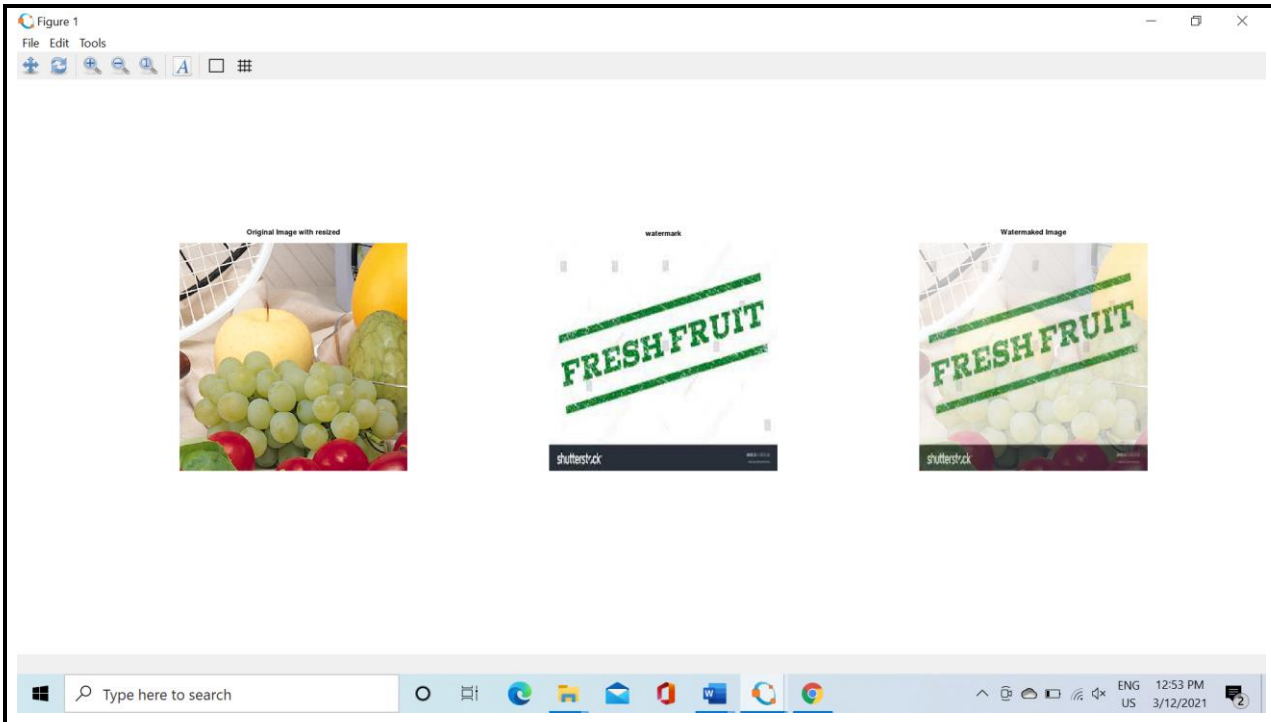


Figure 1
File Edit Tools
Original Image with resized
watermark
Watermaked Image
FRESH FRUIT
FRESH FRUIT
shutterstock
Type here to search
ENG
US
12:53 PM
3/12/2021

Practical 8

Basic Morphological Transformations

A) Boundary Extraction :

Boundary Extraction is a digital image process to extract edges from a binary image. It has also been used as the basis for extracting edges on grayscale and color images.

Morphology is known as the broad set of image processing operations that process images based on the shapes. It is also known as a tool used for extracting image components that are useful in representation and description of region shape.

The basic morphological operations are:

1. Erosion
2. Dilation

Erosion:

- Erosion shrink-ens the image pixels i.e. it is used for shrinking of element A by using element B.
- Erosion removes pixels on object boundaries.:
- The value of the output pixel is the minimum value of all the pixels in the neighborhood. A pixel is set to 0 if any of the neighboring pixels have the value 0.

1. If the value of neighborhood pixel is 0, then change the value of that pixel to 0.

By Using Erosion :

Boundary Extraction

Let A be an Image matrix and B be a structuring element.

Formula for Boundary Extraction:

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

Steps to be followed:

- Convert the image into binary image.
- Perform Erosion:

  Erode binary image A by structuring element B. (i.e)

$$(A \ominus B)$$

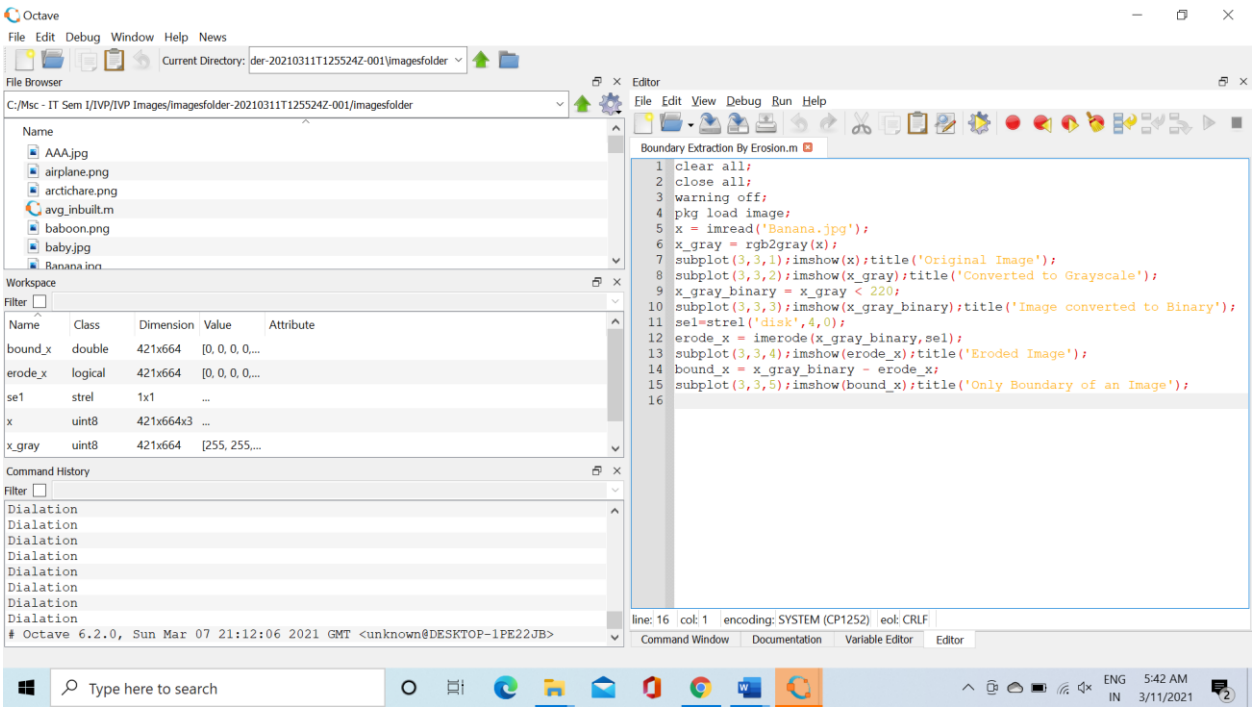
- Subtraction:

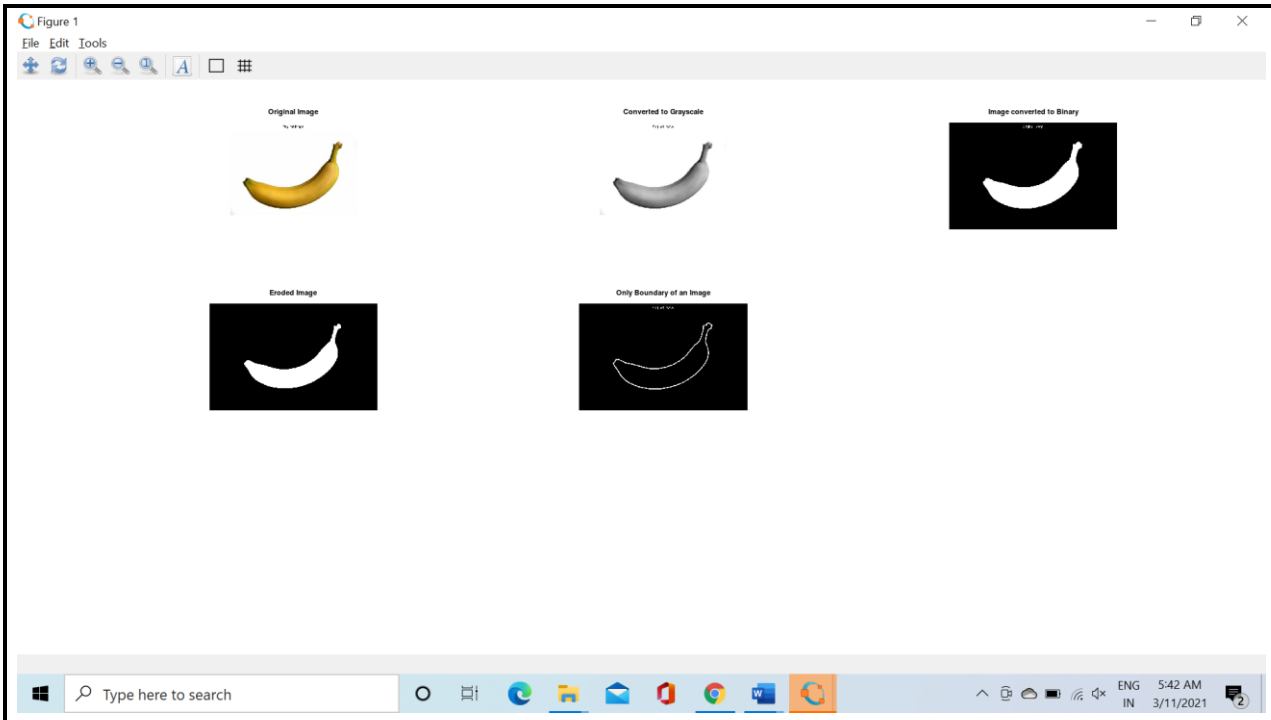
  Subtract the binary image A from the Eroded image.(i.e)

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

Boundary Extraction By Erosion.m

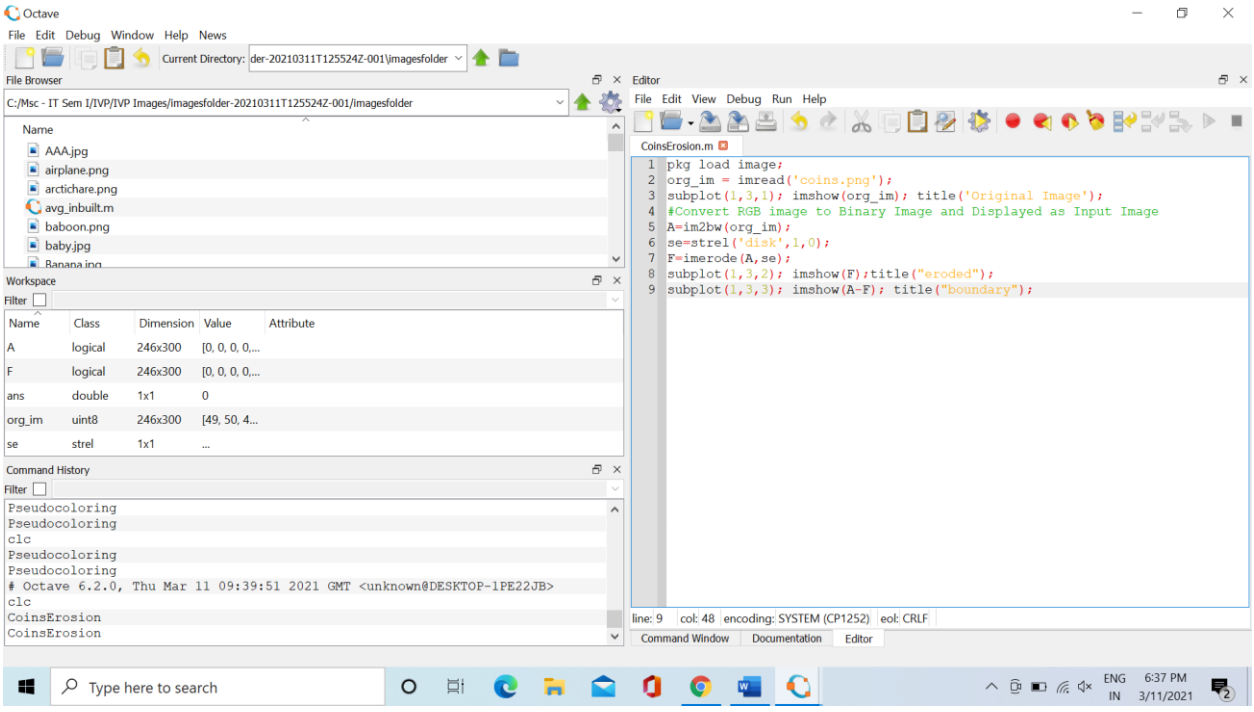
```
clear all;
close all;
warning off;
pkg load image;
x = imread('Banana.jpg');
x_gray = rgb2gray(x);
subplot(3,3,1);imshow(x);title('Original Image');
subplot(3,3,2);imshow(x_gray);title('Converted to Grayscale');
x_gray_binary = x_gray < 220;
subplot(3,3,3);imshow(x_gray_binary);title('Image converted to Binary');
se1=strel('disk',4,0);
erode_x = imerode(x_gray_binary,se1);
subplot(3,3,4);imshow(erode_x);title('Eroded Image');
bound_x = x_gray_binary - erode_x;
subplot(3,3,5);imshow(bound_x);title('Only Boundary of an Image');
```

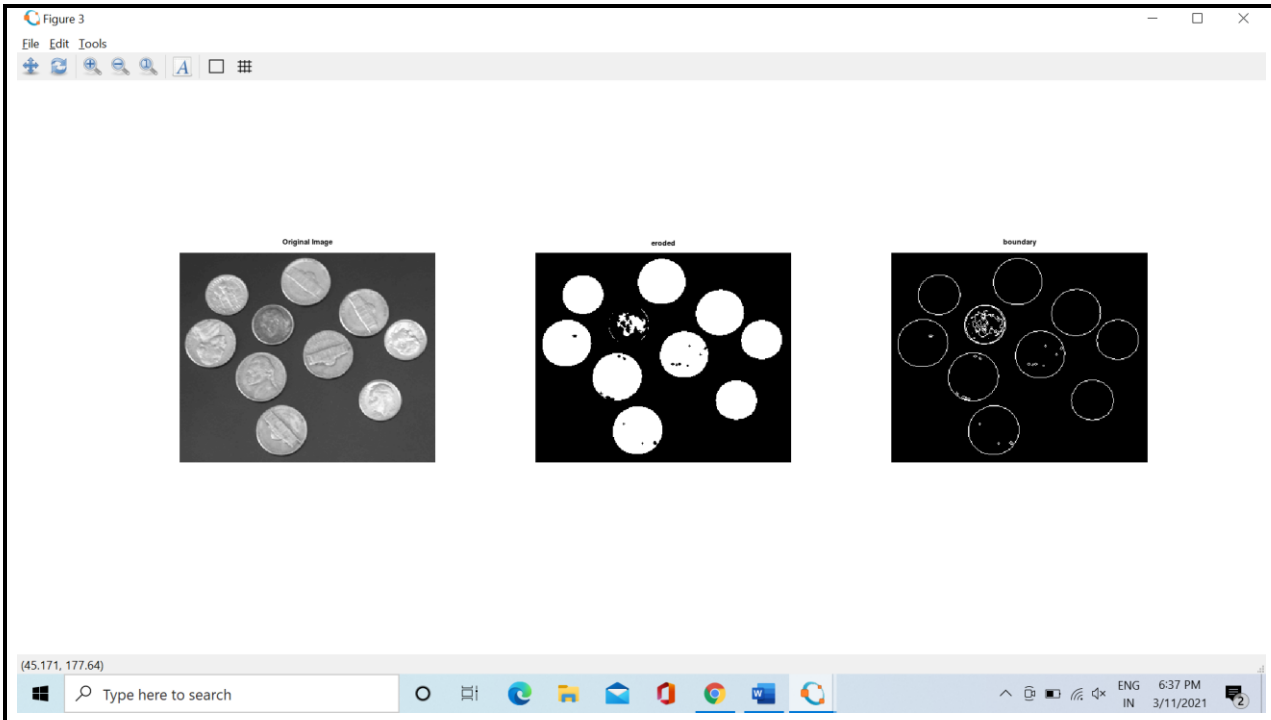




CoinsErosion.m

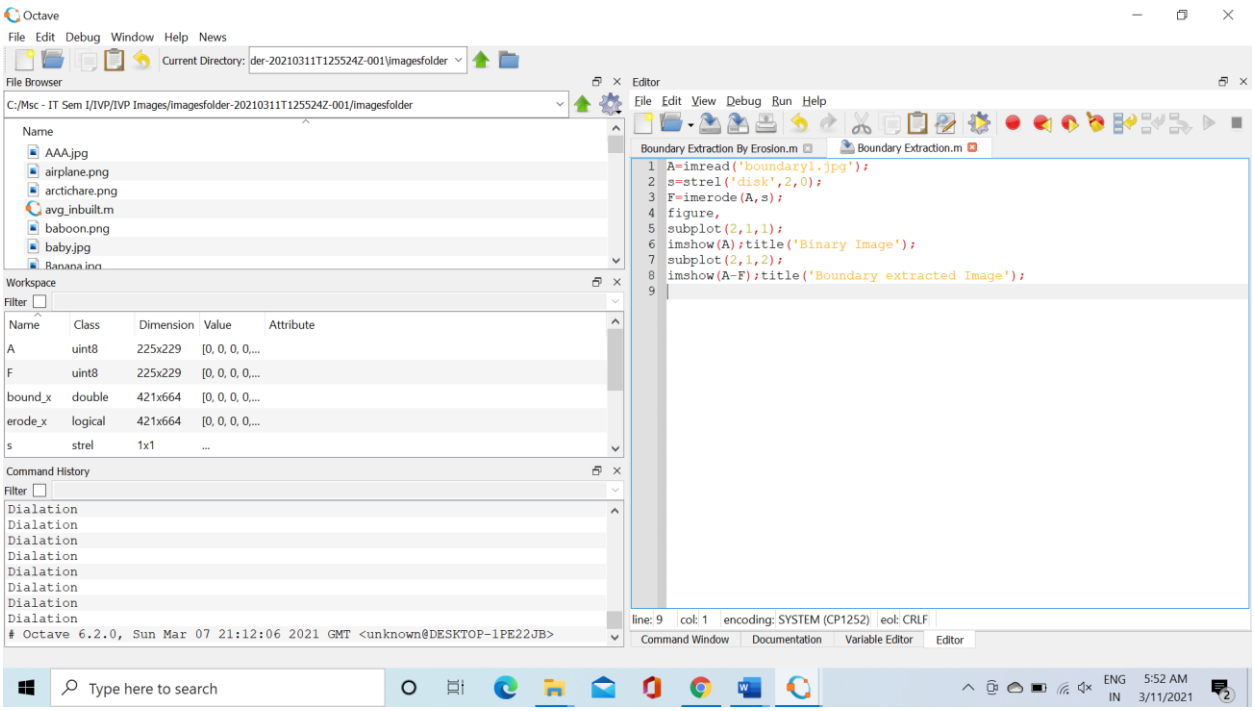
```
pkg load image;
org_im = imread('coins.png');
subplot(1,3,1); imshow(org_im); title('Original Image');
#Convert RGB image to Binary Image and Displayed as Input Image
A=im2bw(org_im);
se=strel('disk',1,0);
F=imerode(A,se);
subplot(1,3,2); imshow(F);title("eroded");
subplot(1,3,3); imshow(A-F); title("boundary");
```

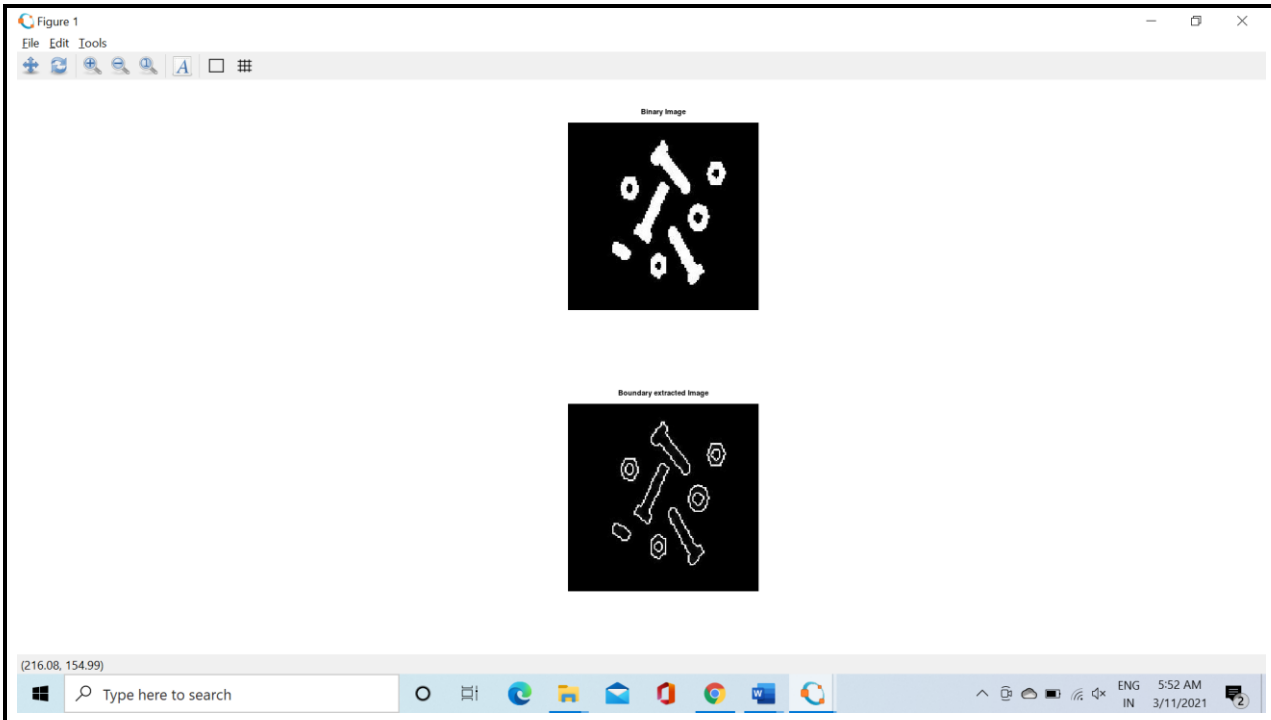




Boundary Extraction.m

```
A=imread('boundary1.jpg');
s=strel('disk',2,0);
F=imerode(A,s);
figure,
subplot(2,1,1);
imshow(A);title('Binary Image');
subplot(2,1,2);
imshow(A-F);title('Boundary extracted Image');
```





Dilation of an Image

Dilation –

- Dilation expands the image pixels i.e. it is used for expanding an element A by using structuring element B.
- Dilation adds pixels to object boundaries.
- The value of the output pixel is the maximum value of all the pixels in the neighborhood. A pixel is set to 1 if any of the neighboring pixels have the value 1.

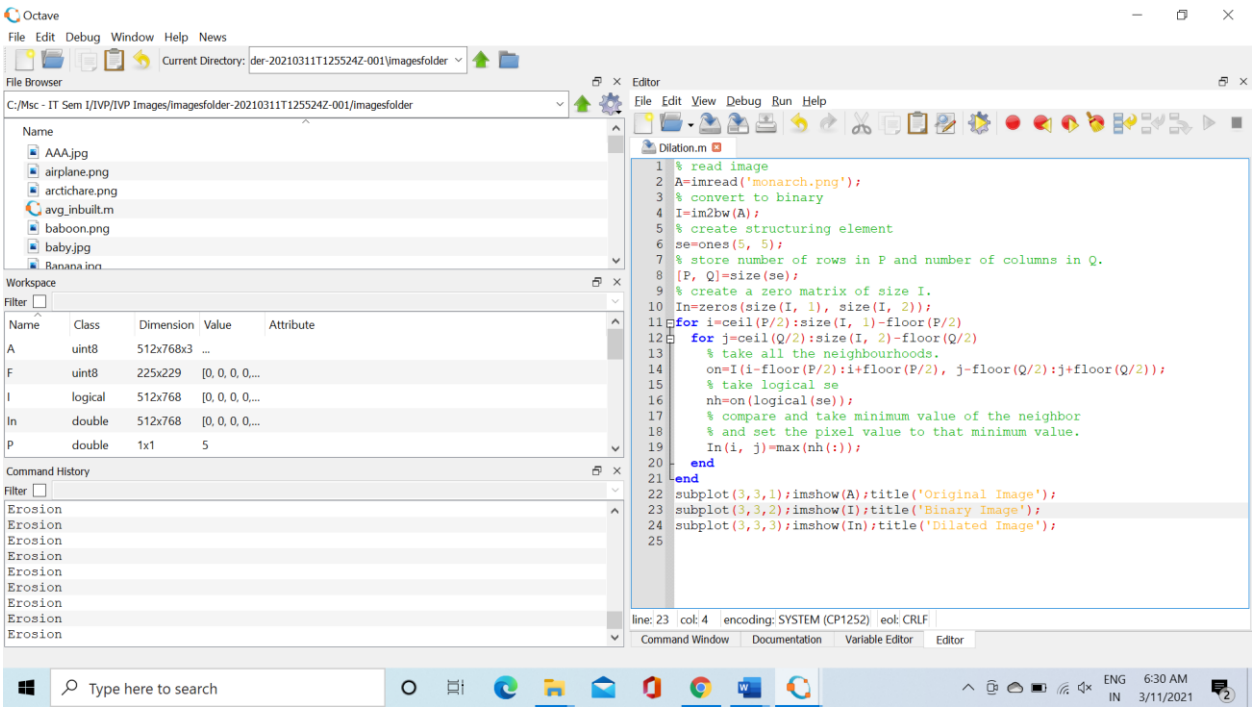
Approach:

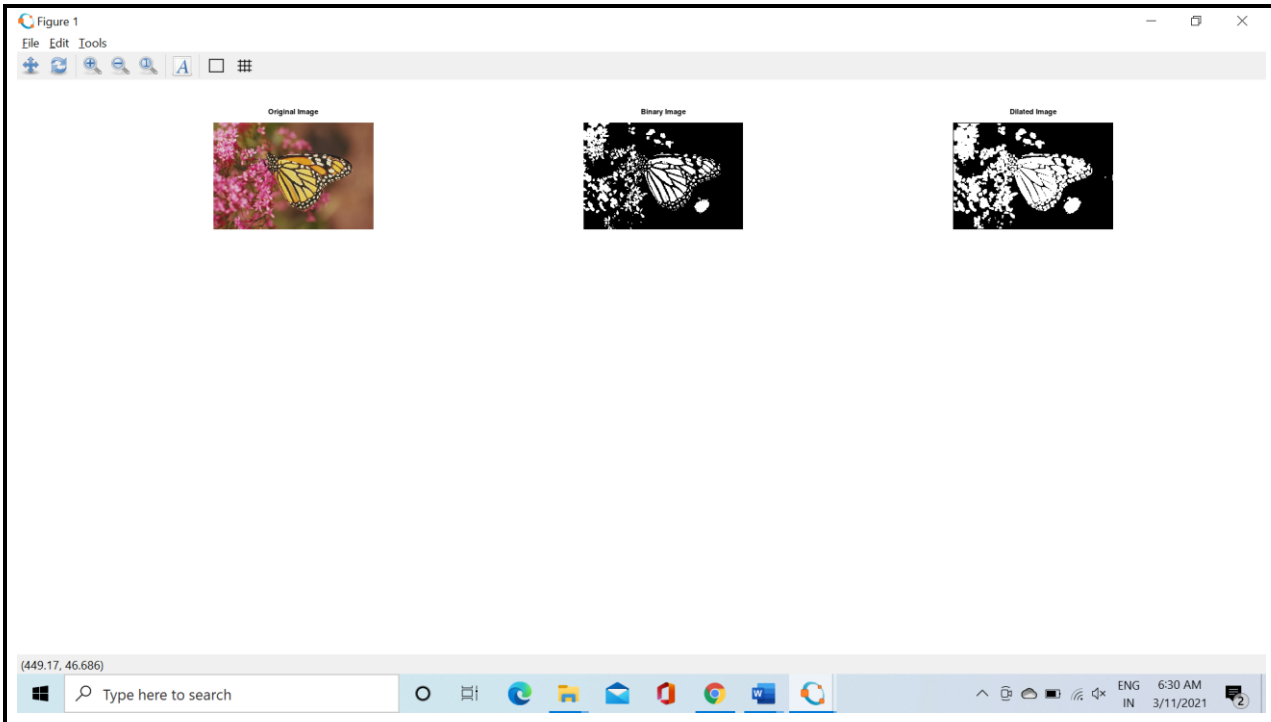
1. Read the RGB image.
2. Using function im2bw(), convert the RGB image to binary image.
3. Create a structuring element or you can use any predefined mask eg. fspecial('sobel').
4. Store the number of rows and columns in and array and loop through it.
5. Create a zero matrix of the size same as of the size of our image.
6. Leaving the boundary pixels start moving the structuring element on the image and start comparing the pixel with the pixels present in neighborhood.
7. If the value of neighborhood pixel is 1, then change the value of that pixel to 1.

Dilation.m

```
% read image

A=imread('monarch.png');

% convert to binary
I=im2bw(A);

% create structuring element
se=ones(5, 5);
```

```
% store number of rows in P and number of columns in Q.
[P, Q]=size(se);

% create a zero matrix of size I.
In=zeros(size(I, 1), size(I, 2));

for i=ceil(P/2):size(I, 1)-floor(P/2)
      for j=ceil(Q/2):size(I, 2)-floor(Q/2)

            % take all the neighbourhoods.
            on=I(i-floor(P/2):i+floor(P/2), j-floor(Q/2):j+floor(Q/2));

            % take logical se
            nh=on(logical(se));

            % compare and take minimum value of the neighbor
            % and set the pixel value to that minimum value.
            In(i, j)=max(nh(:));
      end
end

subplot(3,3,1);imshow(A);title('Original Image');
subplot(3,3,2);imshow(I);title('Binary Image');
subplot(3,3,3);imshow(In);title('Dilated Image');
```

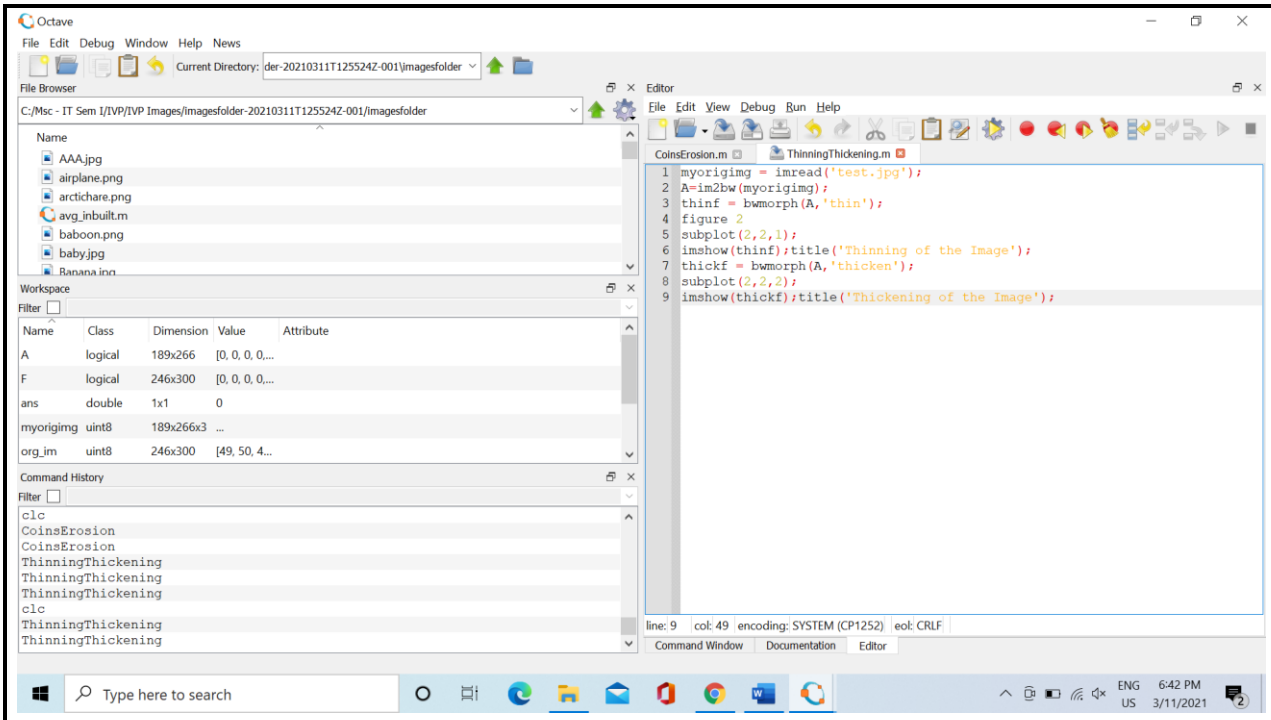


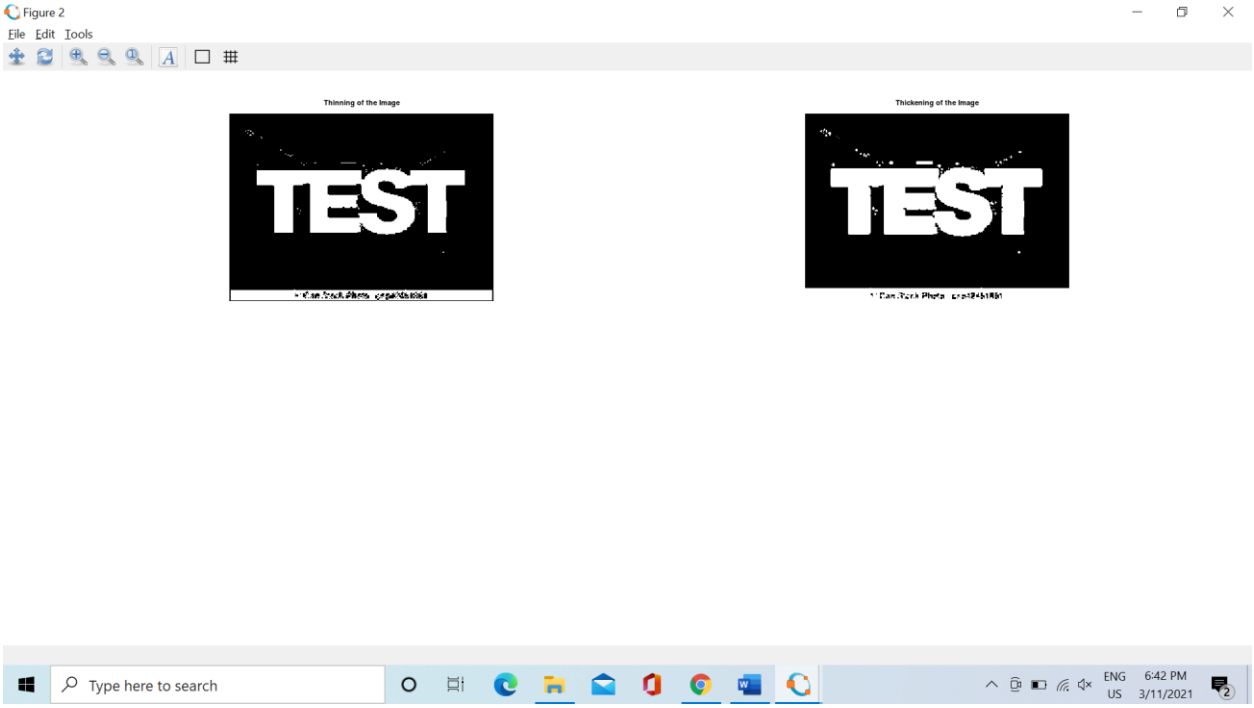


B)Thinning and Thickening :

ThinningThickening.m

```
myorigimg = imread('test.jpg');
A=im2bw(myorigimg);
thinf = bwmorph(A,'thin');
figure 2
subplot(2,2,1);
imshow(thinf);title('Thinning of the Image');
thickf = bwmorph(A,'thicken');
subplot(2,2,2);
imshow(thickf);title('Thickening of the Image');
```



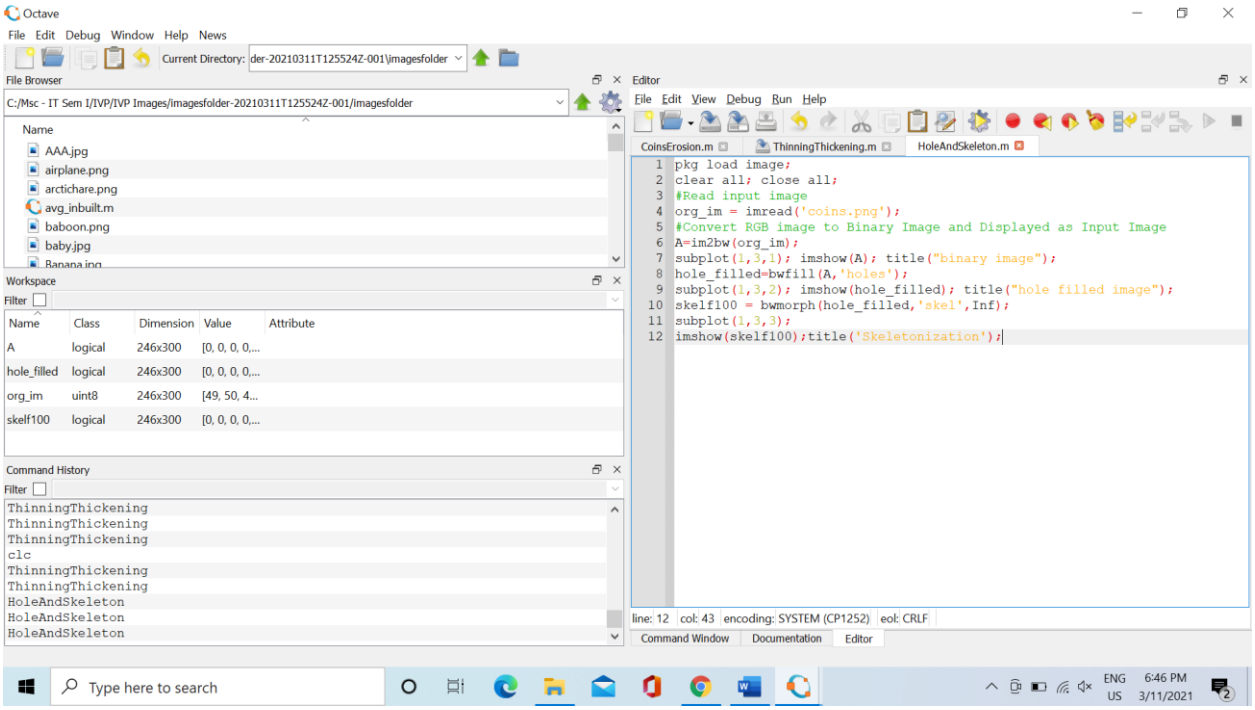


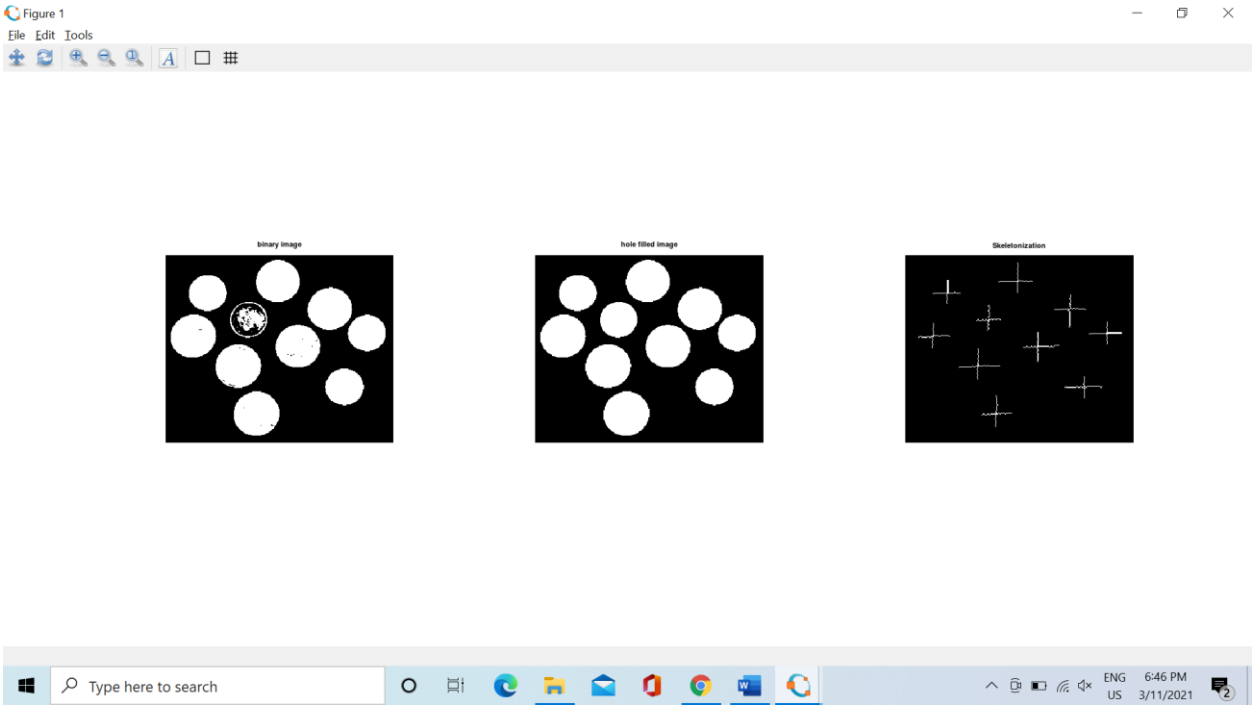
C)Hole filling and Skeletonozation

HoleAndSkeleton.m

```
pkg load image;
clear all; close all;
#Read input image
org_im = imread('coins.png');
#Convert RGB image to Binary Image and Displayed as Input Image
```

```
A=im2bw(org_im);
subplot(1,3,1); imshow(A); title("binary image");
hole_filled=bwfill(A,'holes');
subplot(1,3,2); imshow(hole_filled); title("hole filled image");
skelf100 = bwmorph(hole_filled,'skel',Inf);
subplot(1,3,3);
imshow(skelf100);title('Skeletonization');
```



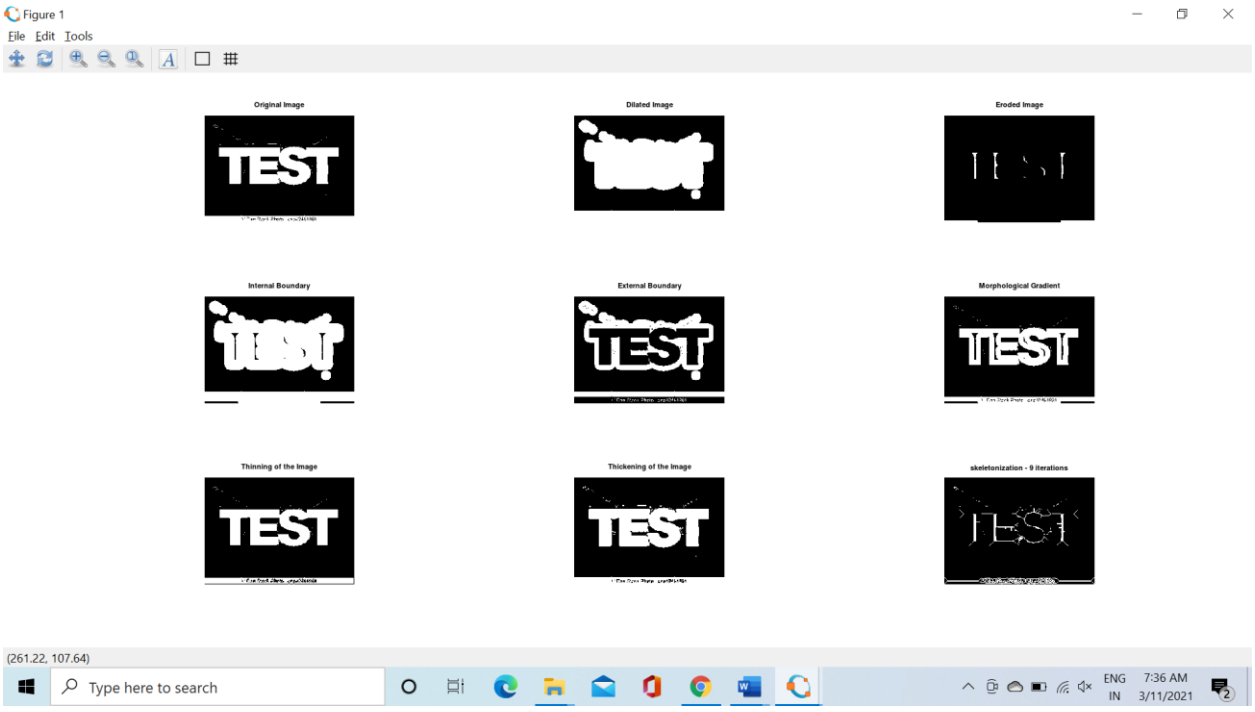


B) Thinning and Thickening Skeletonization

Test.m

```
clc;
close all;
clear all;
%read the test image%convert the image to binary image
myorigimg = imread('test.jpg');
myorigimg = im2bw(rgb2gray(myorigimg));
subplot(3,3,1);
imshow(myorigimg);title('Original Image');
%create structuring element
se = strel('disk',9,0);
%perform dilation operation using imdilate command_line_path
%Display the dilated image
mydilatedimg = imdilate(myorigimg,se);
subplot(3,3,2);
imshow(mydilatedimg);title('Dilated Image');
%perform Erosion operation using imerode command
%Display the eroded image
myerodedimg = imerode(myorigimg,se);
subplot(3,3,3);
imshow(myerodedimg);title('Eroded Image');
%Find internal Boundary
%internal Boundary = Dilated image AND NOT of eroded image
% Display internal boundary
internalboundingimg = mydilatedimg & ~myerodedimg;
subplot(3,3,4);
imshow(internalboundingimg, []);title('Internal Boundary');
%Find external boundary, external boundary = dilated image AND Not of original image
%Display external boundary
externalboundimg = mydilatedimg & ~myorigimg;
subplot(3,3,5);
imshow(externalboundimg,[]);title('External Boundary');
%Find morphological gradient
mymorphgradimg = imsubtract(myorigimg,myerodedimg);
subplot(3,3,6);
imshow(mymorphgradimg,[]);title('Morphological Gradient');

%Perform Thinning operation Using bwmorph()
thinf =bwmorph(myorigimg,'thin');
subplot(3,3,7);
imshow(thinf);title('Thinning of the Image');
%Perform Thickening operation using bwmorph()
thickf = bwmorph(myorigimg,'thicken');
```

```
subplot(3,3,8);
imshow(thickf);title('Thickening of the Image');

%Perform skeletonization operation using bwmorph() command
%with 8 iterations and display the dilated image
skelf100 = bwmorph(myorigimg,'skel',8);
subplot(3,3,9);
imshow(skelf100);title('skeletonization - 9 iterations');
```



C) Hole Filling

HoleFilling.m

```
a=imread('boundary1.jpg');
subplot(1,2,1);imshow(a);
c=imfill(a,'holes');
subplot(1,2,2);imshow(c);
```

Octave

File Edit Debug Window Help News

Current Directory: der-20210311T125524Z-001\imagesfolder

File Browser

C:/Msc - IT Sem I/IVP/IVP Images/imagesfolder-20210311T125524Z-001/imagesfolder

Name

AAA.jpg
airplane.png
arctichare.png
avg_inbuilt.m
baboon.png
baby.jpg
Banana.jpg

Workspace

Filter

Name	Class	Dimension	Value	Attribute
a	uint8	225x229	[0, 0, 0, 0,...	
c	uint8	225x229	[0, 0, 0, 0,...	
externalb...	logical	189x266	[0, 0, 0, 0,...	
hole	logical	225x229	[0, 0, 0, 0,...	
internalbo...	logical	189x266	[0, 0, 0, 0,...	

Command History

Filter

```
HoleFilling
HoleFilling
HoleFilling
HoleFilling
HoleFilling
HoleFilling
HoleFilling
HoleFilling
HoleFilling
```

Editor

File Edit View Debug Run Help

Dilation.m | Test.m | HoleFilling.m | <unnamed>

```
1 a=imread('boundary1.jpg');
2 subplot(1,2,1);imshow(a);
3 c=imfill(a,'holes');
4 subplot(1,2,2);imshow(c);
5
```

line: 5 col: 1 encoding: SYSTEM (CP1252) eol: CRLF

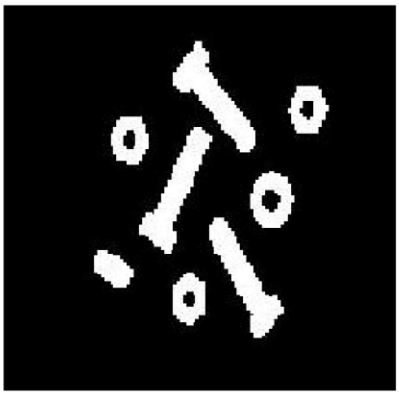
Command Window | Documentation | Editor

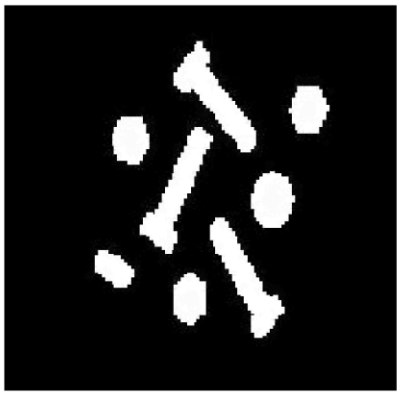
Type here to search

ENG IN 7:50 AM 3/11/2021

Figure 1

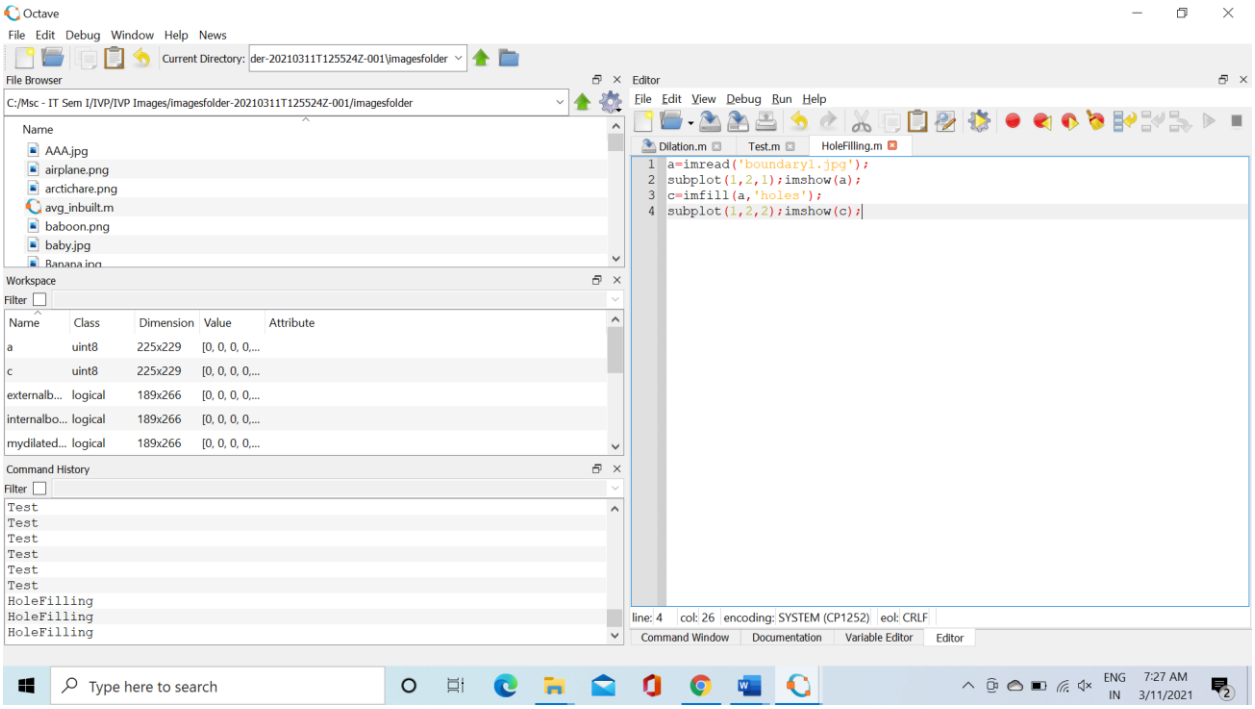
File Edit Tools





Type here to search

ENG IN 7:50 AM 3/11/2021


Octave
File Edit Debug Window Help News
Current Directory: der-20210311T125524Z-001\imagesfolder
File Browser
C:/Msc - IT Sem I/IVP/IVP Images/imagesfolder-20210311T125524Z-001/imagesfolder
Name
AAA.jpg
airplane.png
arctichare.png
avg_inbuilt.m
baboon.png
baby.jpg
Workspace
Filter
Name Class Dimension Value Attribute
a uint8 225x229 [0, 0, 0, 0,...
c uint8 225x229 [0, 0, 0, 0,...
externalb... logical 189x266 [0, 0, 0, 0,...
internalbo... logical 189x266 [0, 0, 0, 0,...
mydilated... logical 189x266 [0, 0, 0, 0,...
Command History
Filter
Test
Test
Test
Test
Test
Test
HoleFilling
HoleFilling
HoleFilling
Editor
File Edit View Debug Run Help
Dilation.m
Test.m
HoleFilling.m
1 a=imread('boundary1.jpg');
2 subplot(1,2,1);imshow(a);
3 c=imfill(a,'holes');
4 subplot(1,2,2);imshow(c);
line: 4 col: 26 encoding: SYSTEM (CP1252) eol: CRLF
Command Window
Documentation
Variable Editor
Editor
Type here to search
ENG IN
7:27 AM 3/11/2021

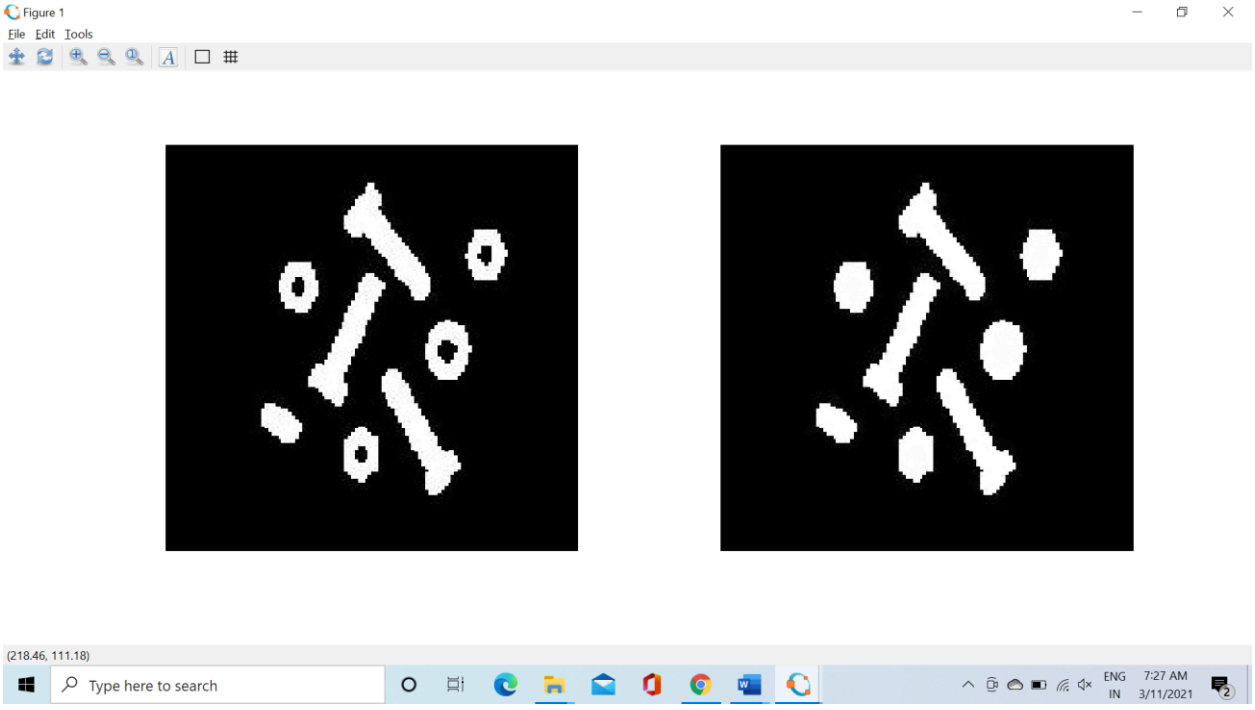


Figure 1
File Edit Tools
(218.46, 111.18)
Type here to search
ENG IN
7:27 AM 3/11/2021