**LAB FILE**

**Digital Image Processing**

**ECE 404**

**B.Tech(ECE) – VII Sem**

#### Submitted by: Faculty Name:

#### Anurag Tripathi     Ms Anupama Bhan

A12405117001,7ECE2, B. TECH



**Department of Electronics and Communication Department**

**Amity School of Engineering & Technology**

**Amity University, Noida, Uttar Pradesh**

**INDEX**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S.NO | NAME OF THE EXPERIMENT | DATE OF EXPT | DATE  OF  SUBMISSION | MAX  MARKS | MARKS  OBTAINED | SIGNATURE  OF  FACULTY |
| 1 | To perform Basic image Processing Operations | 15/07/20 | 22/07/20 | 1 |  |  |
| 2 | To write program for Histogram Modelling. | 22/07/20 | 29/07/20 | 1 |  |  |
| 3 | program for Gray level transformations | 29/07/20 | 05/08/20 | 1 |  |  |
| 4 | Image enhancement using Smoothing and sharpening spatial filters. | 05/08/20 | 19/08/20 | 1 |  |  |
| 5 | Image enhancement using frequency domain | 19/08/20 | 28/08/20 | 1 |  |  |
| 6 | Homomorphic filtering | 28/08/20 | 02/09/20 | 1 |  |  |
| 7 | Implement first derivative gradient filters | 02/09/20 | 09/09/20 | 1 |  |  |
| 8 | Laplacian filtering | 09/09/20 | 23/09/20 | 1 |  |  |
| 9 | Image restoration using weiner | 23/09/20 | 30/09/20 | 1 |  |  |
| 10 | image compression using DCT and recover the image using IDCT | 30/09/20 | 07/10/20 | 1 |  |  |
| 11 | Image segmentation using thresholding | 07/10/20 | 14/10/20 | 1 |  |  |
| 12 | Rgb to gray using 3 techniques | 21/10/20 |  | 1 |  |  |

**EXPERIMENT 1**

**AIM:**

To perform Basic image Processing Operations.

**APPARATUS/TOOLS USED:**

MATLAB

**THEORY**

An RGB image, sometimes referred to as a truecolor image, is stored in MATLAB as an m-by-n-by-3 data array that defines red, green, and blue color components for each individual pixel. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location.  A grayscale or [image](https://en.wikipedia.org/wiki/Image) is one in which the value of each [pixel](https://en.wikipedia.org/wiki/Pixel) is a single [sample](https://en.wikipedia.org/wiki/Sample_(signal)) representing only an amount of [light](https://en.wikipedia.org/wiki/Light); that is, it carries only [intensity](https://en.wikipedia.org/wiki/Luminous_intensity) information. Grayscale images, a kind of [black-and-white](https://en.wikipedia.org/wiki/Black-and-white) or gray [monochrome](https://en.wikipedia.org/wiki/Monochrome), are composed exclusively of [shades of gray](https://en.wikipedia.org/wiki/Shades_of_gray). The [contrast](https://en.wikipedia.org/wiki/Contrast_(vision)) ranges from [black](https://en.wikipedia.org/wiki/Black) at the weakest intensity to [white](https://en.wikipedia.org/wiki/White) at the strongest. A binary image is one that consists of pixels that can have one of exactly two colors, usually black and white. Binary images are also called bi-level or two-level. This means that each pixel is stored as a single bit . i.e., a 0 or 1.

**CODE:**

%**converting RGB to grayscale**

a=imread('D:\projectml\john1.jpg');

b=rgb2gray(a);

whos a

whos b

subplot(1,3,1)

imshow(a);

title('original image')

subplot(1,3,2)

imshow(b)

title('Gray scale image')

**%converting into binary**

c=im2bw(b);

whos c

subplot(1,3,3)

imshow(c);

title('binary image')

a=imread('D:\projectml\john1.jpg');

**%cropping of image**

d=imcrop(a,[60 40 90 80]);

figure;

subplot(1 ,2, 1);

imshow(d);

title('cropped image ')

a=imread('D:\projectml\john1.jpg');

**%cropping of image**

[j,rect]= imcrop(a);

subplot(2,2,2)

imshow [j,rect]

title('cropped image')

**%resizing of image**

a=imread('D:\projectml\john1.jpg');

subplot(1,2,1)

imshow(a);

title('original image');

b=imresize(a,[256,256]);

subplot(1,2,2)

imshow(b);

title('resized image')

**OBSERVATIONS:**



Figure 1.1 Our Original Image

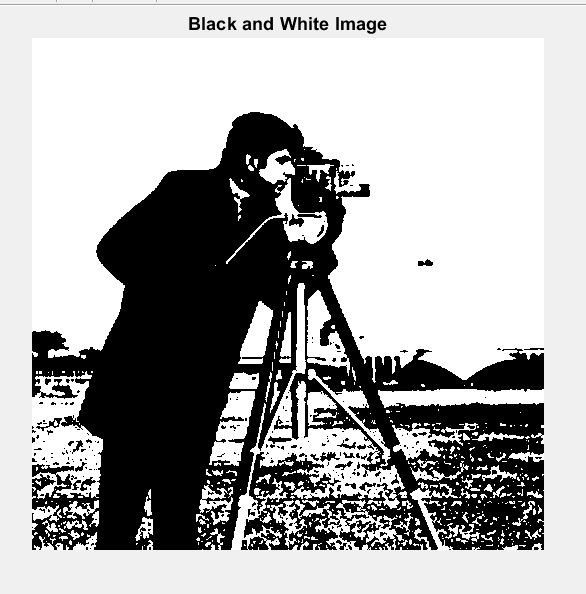


Figure 1.2 Converting RGB image to binary image.

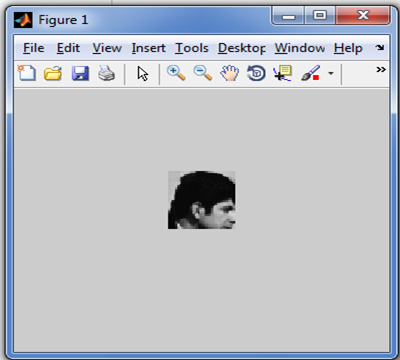
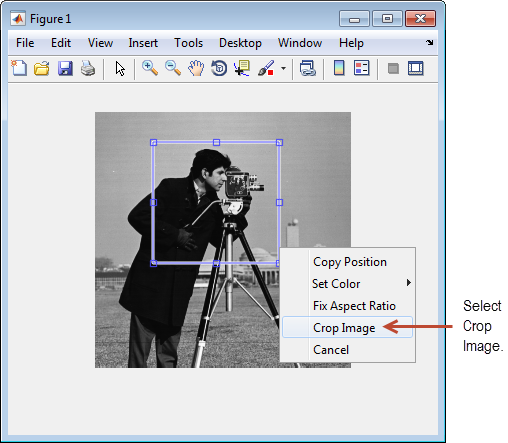
 

Figure 1.3 Cropping of image



Figure 1.4 Resizing of image

**RESULTS:**

we have successfully performed Basic image Processing Operations tools on image.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 2**

**Aim :** To write program for Histogram Modelling.

**Apparatus/Tools Required :** Matlab

**Theory :**

An image histogram is a type of [histogram](https://en.wikipedia.org/wiki/Histogram) that acts as a [graphical representation](https://en.wikipedia.org/wiki/Graphical_representation) of the [tonal](https://en.wikipedia.org/wiki/Lightness_(color)) distribution in a [digital image](https://en.wikipedia.org/wiki/Digital_image).[[1]](https://en.wikipedia.org/wiki/Image_histogram#cite_note-sutton-1) It plots the number of [pixels](https://en.wikipedia.org/wiki/Pixels) for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance.

Image histograms are present on many modern [digital cameras](https://en.wikipedia.org/wiki/Digital_cameras). Photographers can use them as an aid to show the distribution of tones captured, and whether image detail has been lost to blown-out highlights or blacked-out shadows.[[2]](https://en.wikipedia.org/wiki/Image_histogram#cite_note-2) This is less useful when using a [raw image format](https://en.wikipedia.org/wiki/Raw_image_format), as the [dynamic range](https://en.wikipedia.org/wiki/Dynamic_range#Photography) of the displayed image may only be an approximation to that in the raw file.

The [horizontal axis](https://en.wikipedia.org/wiki/Horizontal_axis) of the [graph](https://en.wikipedia.org/wiki/Graphics) represents the tonal variations, while the [vertical axis](https://en.wikipedia.org/wiki/Vertical_axis) represents the total number of pixels in that particular tone.[[1]](https://en.wikipedia.org/wiki/Image_histogram#cite_note-sutton-1)

The left side of the horizontal axis represents the dark areas, the middle represents mid-tone values and the right hand side represents light areas. The vertical axis represents the size of the area (total number of pixels) that is captured in each one of these zones.

Thus, the histogram for a very dark image will have most of its data points on the left side and center of the graph.

**Code**

%rotation of image

clc;

close all;

z=imread('D:\abcd.jpg');

A=rgb2gray(z)

subplot(2,4,1)

imshow(A);

title('original image');

B=imrotate(A,60);

subplot(2,4,2);

imshow(B);

title('rotated image');

%addition of salt and pepper noise in the image

C=imnoise(A,'salt & pepper',0.2);

subplot(2,4,3);

imshow(C);

title('salt and pepper noise');

%histogram plotting

subplot(2,4,4)

imhist(A);

title('histogram of image')

xlabel('pixel of gray levels')

ylabel('occurence of gray level')

subplot(2,4,5)

imhist(B);

title('histogram of rotated image')

subplot(2,4,6)

imhist(C);

title('histogram of noisy image')

xlabel('pixels of gray levels')

ylabel('occurence of gray levels')

D=imnoise(A,'gaussian');

subplot(2,4,7)

imshow(D);

title('image with gaussian noise')

subplot(2,4,8)

imhist(D)

title('histogram of gaussian noise image');

%%%%

clc;

close all;

A=imread('D:\abcd.jpg');

subplot(3,3,1)

imshow(A);

title('original image');

R=A;

R(:,:,2)=0;

R(:,:,3)=0;

subplot(3,3,2)

imshow(R)

title('red component of the original image')

R=A(:,:,1);

subplot(3,3,3);

imhist(R);

title('histogram of red plane');

s=A;

s(:,:,1)=0;

s(:,:,3)=0;

subplot(3,3,4)

imshow(s)

title('green component of the original image')

s=A(:,:,2);

subplot(3,3,5);

imhist(s);

title('histogram of green plane');

t=A;

t(:,:,1)=0;

t(:,:,2)=0;

subplot(3,3,6)

imshow(t)

title('blue component of the original image')

t=A(:,:,3);

subplot(3,3,7);

imhist(t);

title('histogram of blue plane');

**Observations-**

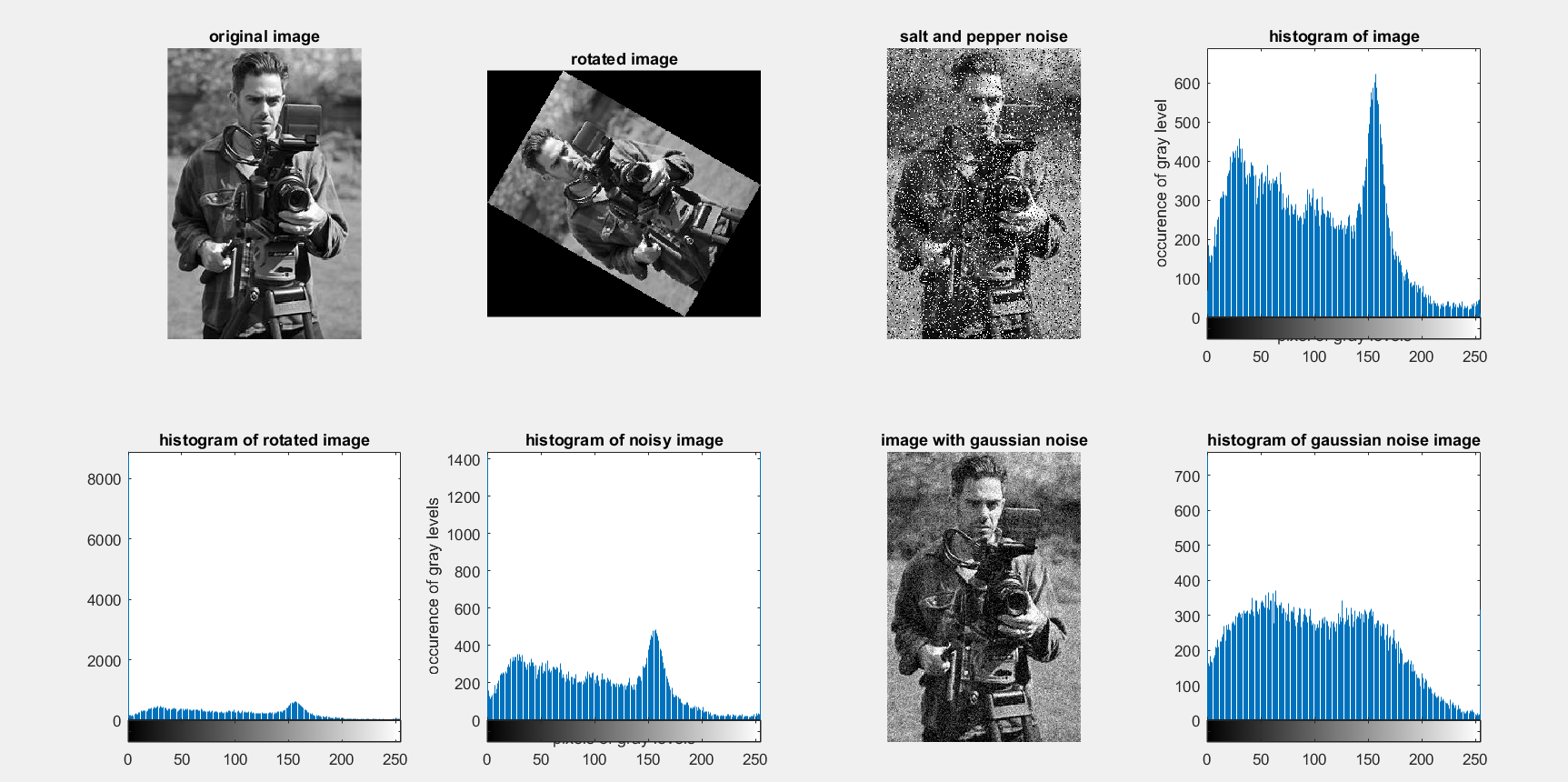


Figure – 2.1 Histogram of original,rotated,noisy images.

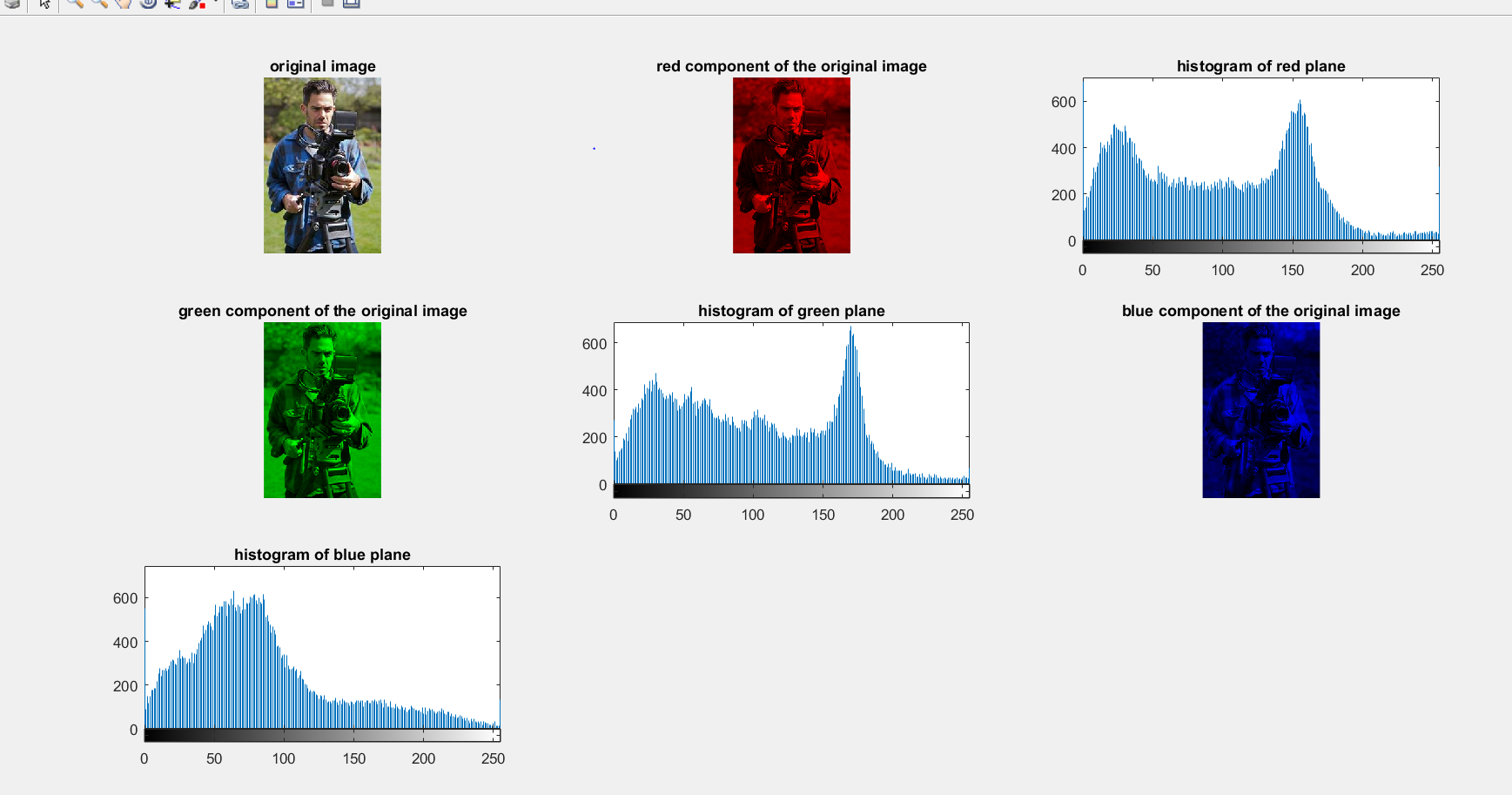


Figure – 2.2 Histogram of all channels of RGBimages.

**Results:** we performed and plotted the histogram modelling of rgb image.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 3**

**Aim :** To write program for Gray level transformations: Contrast stretching, log transformation, power law transformation, Gamma correction.

**Apparatus/Tools Required :** Matlab

**Theory :**

## Negative transformation-The second linear transformation is negative transformation, which is invert of identity transformation. In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image.

## During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation. This result in following image enhancement. The value of c in the log transform adjust the kind of enhancement you are looking for.

Power law transformation is also known as gamma transformation.Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their own gamma correction, that’s why they display their image at different intensity.

This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different. For example Gamma of CRT lies in between of 1.8 to 2.5, that means the image displayed on CRT is dark.

**Code**

%rotation of image

clc;

close all;

z=imread('D:\abcd.jpg');

A=rgb2gray(z)

subplot(2,4,1)

imshow(A);

title('original image');

[x,y]=size(A);

%negative

n=255-A;

figure(2)

subplot(2,4,2);imshow(n);

title('negative image')

for i=0:255,

map(i+1)=255-(i-1);

end

figure(3);plot(map);title('Plot of negative');

%For Logarithmic

d=double(A);

c=20;

l=c\*log(1+d);

figure(4);imshow(uint8(l)); title('Logarithmic Transformation');

for i=0:255,

map(i+1)=c\*log(1+i);

end

figure(5);plot(map);title('Plot of Logarithmic transform');

%\*For Power Law Transformation

c=10;

r=1.1;

p=c\*(power(d,r));

figure(6);imshow(uint8(p));title('Power Law Transform');

for i=0:255,

map(i+1)=c\*(power(double(i),r));

end

figure(7);plot(map);title('Plot of Power Law Transform');

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*For Contrast Stretching\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

r1=60;

s1=45;

r2=180;

s2=200;

m1=s1/r1;

m2=(s2-s1)/(r2-r1);

m3=(255-s2)/(255-r2);

piece=[(0:r1)\*m1,(((0:(r2-r1))\*m2)+s1),(((0:(255-r2))\*m3)+s2)];

for i=1:x,

for j=1:y,

c(i,j)=piece(a(i,j)+1);

end

end

figure(8);imshow(uint8(c));title('Contrast Stretching');

figure(9);plot(piece);title('Plot of Contrast Streching');

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*For Thresholding\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

th=125;

for i=1:x,

for j=1:y,

if(a(i,j)<th)

t(i,j)=0;

else

t(i,j)=255;

end

end

end

figure(10);imshow(uint8(t));title('Thrsholding');

map(1:th)=0;

map(th:256)=255;

figure(11);plot(map);title('Plot of Thresholding');

**Observations**



Figure 3.1 Original image

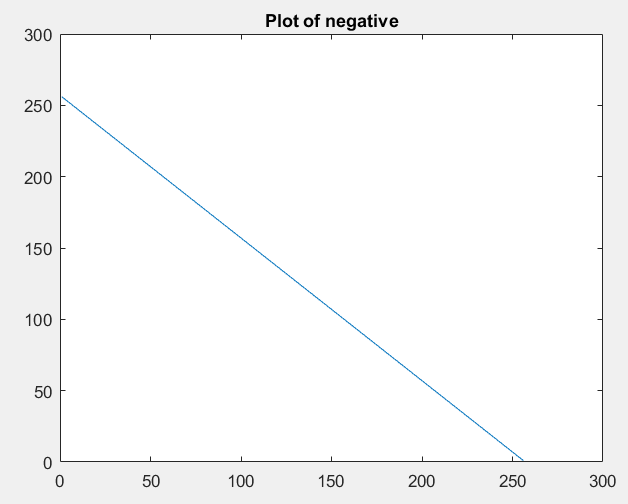
 

Figure 3.2 Negative image and its plot .

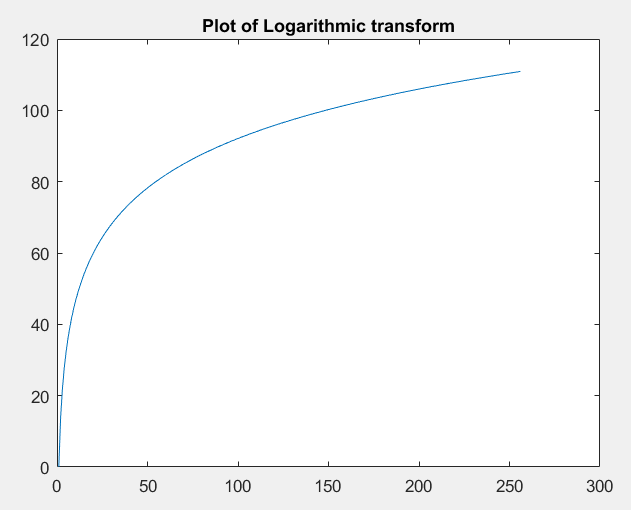
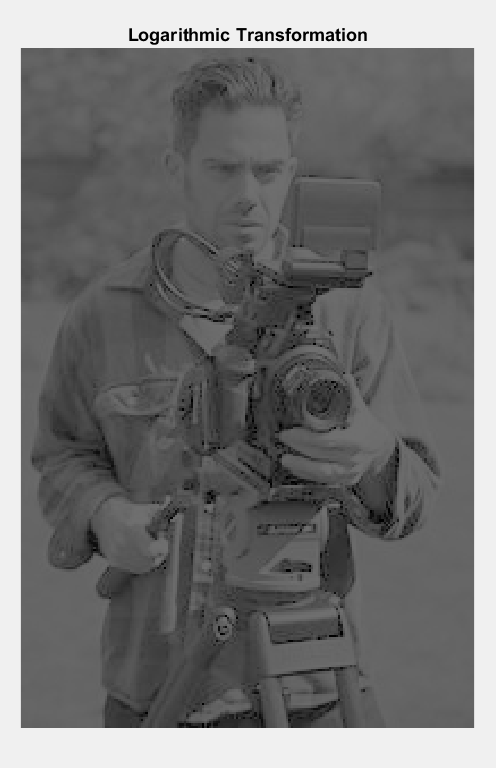


Figure 3.3 Logarithmic transformation

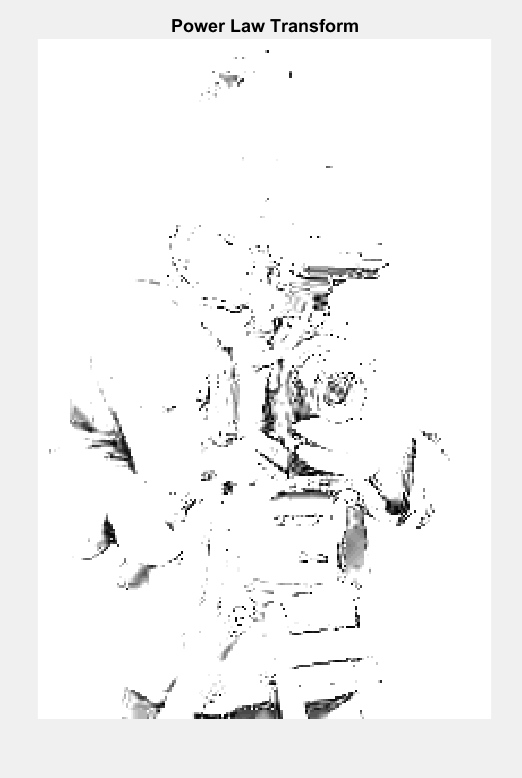
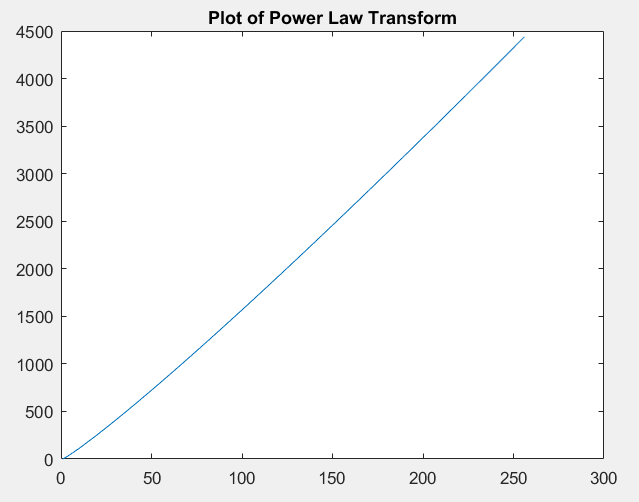
 

Figure 3.4 Power law transformation

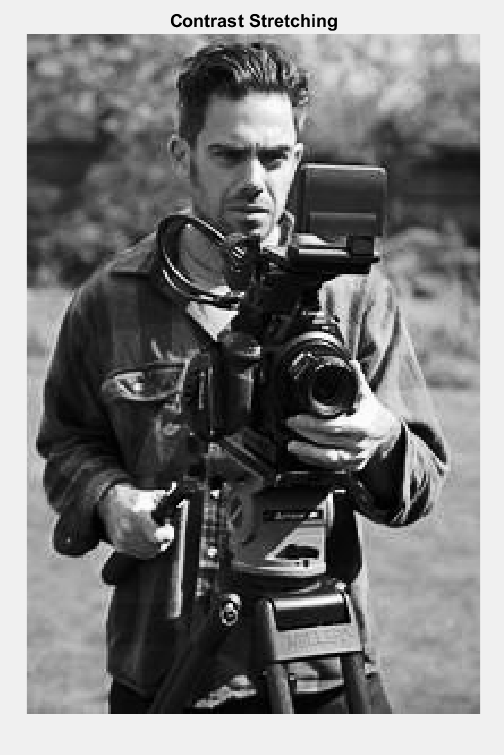
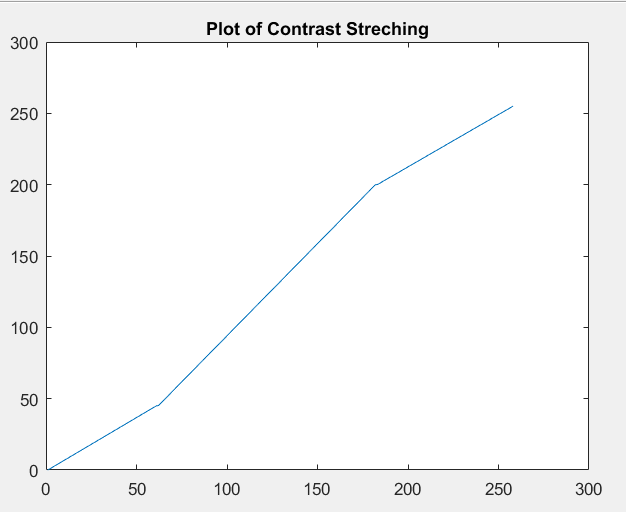
 

Figure 3.5 Contrast stretching

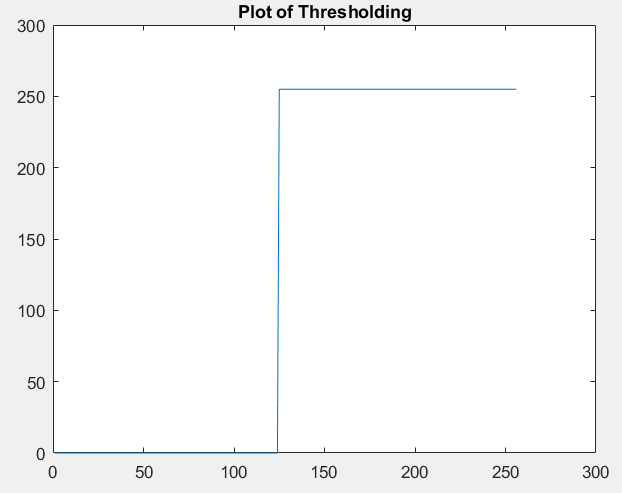
 

Figure 3.6 Thresholding

**Results:**  Gray level transformations: Contrast stretching, log transformation, power law transformation, Gamma correction is performed and plotted.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 4**

**Aim :** To write program for Image enhancement using Smoothing and sharpening spatial filters.

**Apparatus/Tools Required :** Matlab

**Theory :**

[Spatial Filtering](https://www.geeksforgeeks.org/spatial-filtering-and-its-types/) technique is used directly on pixels of an image. Mask is usually considered to be added in size so that it has a specific center pixel. This mask is moved on the image such that the center of the mask traverses all image pixels.

**Low Pass filtering:** It is also known as the smoothing filter. It removes the high-frequency content from the image. It is also used to blur an image.

**High Pass Filtering:** It eliminates low-frequency regions while retaining or enhancing the high-frequency components.

Code

clc;

close all;

z=imread('D:\abcd.jpg');

A=rgb2gray(z)

subplot(2,3,1)

imshow(A);

title('original image');

h1=1/9\*ones(3,3);

h2=1/25\*ones(5,5);

h3=1/49\*ones(7,7);

h4=1/81\*ones(9,9);

b1=conv2(double(A),double(h1),'same');

subplot(2,3,2)

imshow(uint8(b1));

title('average filter(3\*3)');

b2=conv2(double(A),double(h1),'same');

subplot(2,3,3)

imshow(uint8(b2));

title('average filter(5\*5)');

b3=conv2(double(A),double(h1),'same');

subplot(2,3,4)

imshow(uint8(b3));

title('average filter(7\*7)');

b4=conv2(double(A),double(h1),'same');

subplot(2,3,5)

imshow(uint8(b4));

title('average filter(9\*9)');

%median filtercode

clc;

close all;

z=imread('D:\abcd.jpg');

A=rgb2gray(z)

B=imnoise(A,'salt & pepper',0.03);

C=medfilt2(B,[3,3]);

D=imnoise(A,'salt & pepper');

C1=medfilt2(B,[7,7]);

subplot(3,2,1)

imshow(z);

title('original image');

subplot(3,2,2)

imshow(A);

title('grayscale image');

subplot(3,2,3)

imshow(B);

title('salt and pepper noise with filter 3\*3 ');

subplot(3,2,4)

imshow(C);

title('median filter of B ');

subplot(3,2,5)

imshow(D);

title('salt and pepper noise with filter 7\*7 ');

subplot(3,2,5)

imshow(C1);

title('median filter of D ');

%min max filter

z=imread('D:\abcd.jpg');

a=rgb2gray(z);

imshow(a);

c1=ordfilt2(double(a),1,ones(3,3));

subplot(1,2,1);

imshow(uint8(c1));

title('min filter');

c2=ordfilt2(double(a),9,ones(3,3));

subplot(1,2,2);

imshow(uint8(c2));

title('max filter');

**Observations**



Figure 4.1 Applying average filters with different masks.



Figure -4.2 Applying median filters



Figure -4.3 MIN MAX filters

**Results:** Image enhancement using Smoothing and sharpening spatial filters is performed and plotted.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 5**

**Aim :** . To write program for Image enhancement using frequency domain filters

**Apparatus/Tools Required :** Matlab

**Theory :**

Image enhancement is used when we need to focus or pick out some important features of an image. For example, we may want to sharpen the image to bring out details such as a car license plate number or some areas of an **X**-ray film. In aerial photographs, the edges or lines may need to be enhanced in order to pick out buildings or other objects. Certain spectral components of an image may need to be enhanced in images obtained from telescopes or space probes. In some cases, the contrast may need to be enhanced.

While linear filtering may be all that is required for certain types of enhancement, most useful enhancement operations are non-linear in nature.

Code

a = imread('D:\cameraman.png');

b= im2double(a);

[m,n]=size(b);

c = zeros(2\*m,2\*n);

[p,q] = size(c);

for i = 1:p

for j = 1:q

if i<=m && j<=n

c(i,j) = b(i,j);

end

end

end

subplot(3,3,1);

imshow(b);

title("Original Image");

subplot(3,3,2);

imshow(c);

title("Padded Image");

d = zeros(p,q);

for i= 1:p

for j = 1:q

d(i,j) = c(i,j).\*(-1).^(i+j);

end

end

subplot(3,3,3);

imshow(d);

title("Preprocessed Image for Calculating DFT");

e = fft2(d);

subplot(3,3,4);

imshow(e);

title("2D DFT of the preprocessed Image");

[x,y] = freqspace(p,'meshgrid');

z = zeros(p,q);

for i = 1:p

for j = 1:q

z(i,j)=sqrt(x(i,j).^2+y(i,j).^2);

end

end

H = zeros(p,q);

for i = 1:p

for j = 1:q

if z(i,j)<=0.4

H(i,j)=1;

else

H(i,j)=0;

end

end

end

subplot(3,3,5);

imshow(H);

title('Low pass filter function');

h1 = e.\*H;

subplot(3,3,6);

imshow(h1);

title("Low pass output");

h2 = ifft2(h1);

subplot(3,3,7);

imshow(h2);

title("Output image after inverse 2D DFT")

h3 = zeros(p,q);

for i = 1:p

for j = 1:q

h3(i,j)=h2(i,j).\*((-1).^(i+j));

end

end

subplot(3,3,8);

imshow(h3);

title("Post processed Image");

out = zeros(m,n);

for i = 1:m

for j=1:n

out(i,j)=h3(i,j);

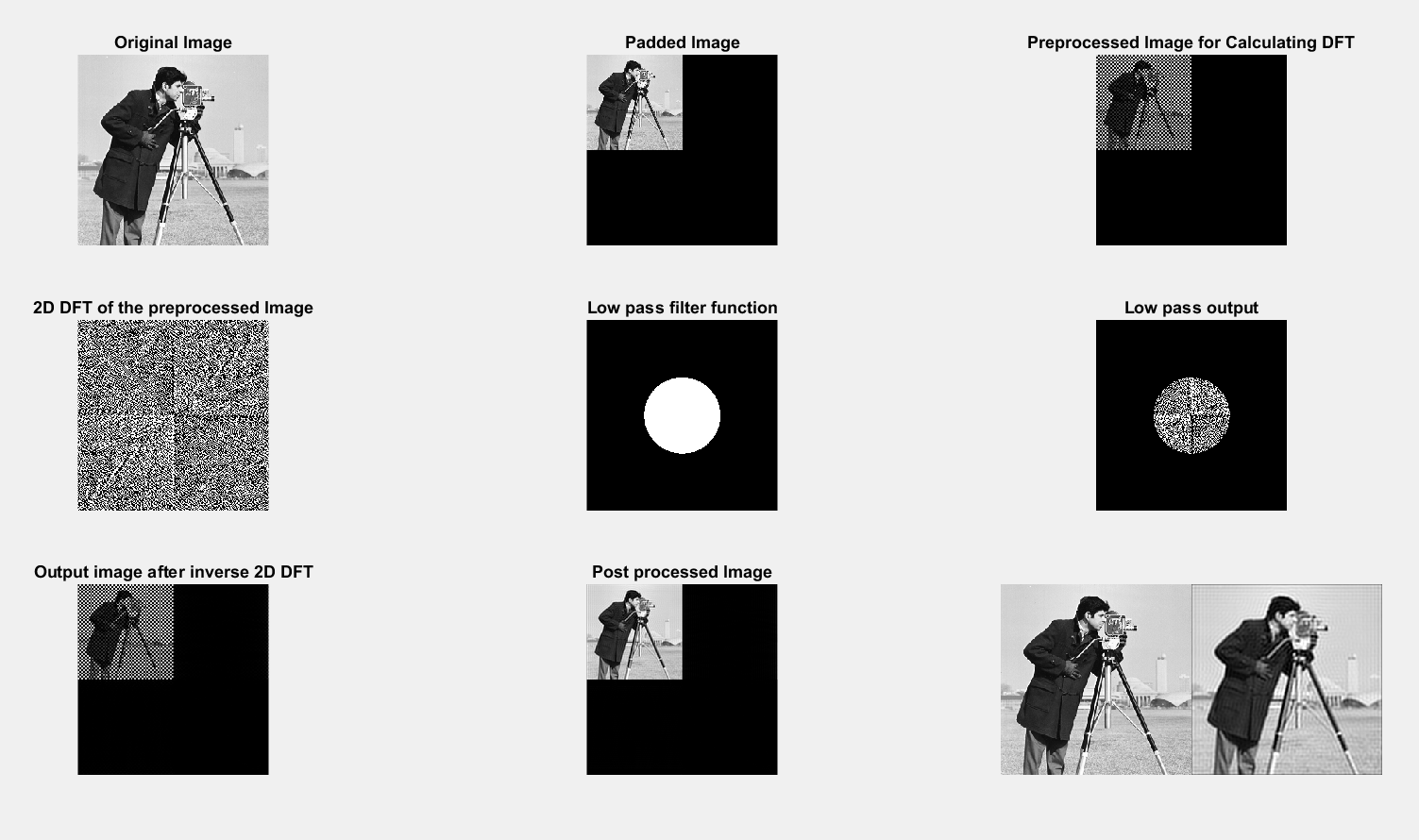
end

end

subplot(3,3,9);

imshow([b out]);

**Observations**



**Results:** Image enhancement using frequency domain filters is performed and plotted.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 6**

**Aim :** To write program for Image enhancement using Homomorphic Filtering

**Apparatus/Tools Required :** Matlab

**Theory :**

Homomorphic filtering is most commonly used for correcting non-uniform illumination in images. The illumination-reflectance model of image formation says that the intensity at any pixel, which is the amount of light reflected by a point on the object, is the product of the illumination of the scene and the reflectance of the object(s) in the scene.

Code

clc;

clear all;

img = imread('D:\xray.jpg');

img1 = rgb2gray(img);

img = double(img1);

[r,c] = size(img);

img = img+1;

lim = log(img);

fim = fft2(lim);

lowg = 0.9;

highg = 1.1;

him = homomorph(fim,lowg,highg);

ifim=ifft2(him);

eim = exp(ifim);

%plots

figure;

subplot(2,3,1);

imshow(img1);

title("Original Image");

subplot(2,3,2);

imshow(lim);

title("Logarithmic Transformed Image");

subplot(2,3,3);

imshow(uint8(fim));

title('2-D DFT of Image');

subplot(2,3,4);

imshow(him);

title("Homomorphic Filtered Image");

subplot(2,3,5);

imshow(ifim);

title("Inverse fft of the Image");

subplot(2,3,6);

imshow(uint8(eim));

title('Inverse logarithmic');

function res=homomorph(fim,lowg,highg);

dif = (highg-lowg);

sig = 15;

[r,c] = size(fim);

for i = 1:r

for j=1:c

p=-(((i-(r/2))^2)+(j-(c/2))^2)/((2\*sig^2));

k = (1/2\*3.14\*(sig^2));

term(i,j) = (1-k\*exp(p));

end

end

for i=1:r

for j=1:c

h(i,j)=(dif\*term(i,j))+lowg;

end

end

for i = 1:r

for j=1:c

res(i,j)=fim(i,j)\*h(i,j);

end

end

**Observations**

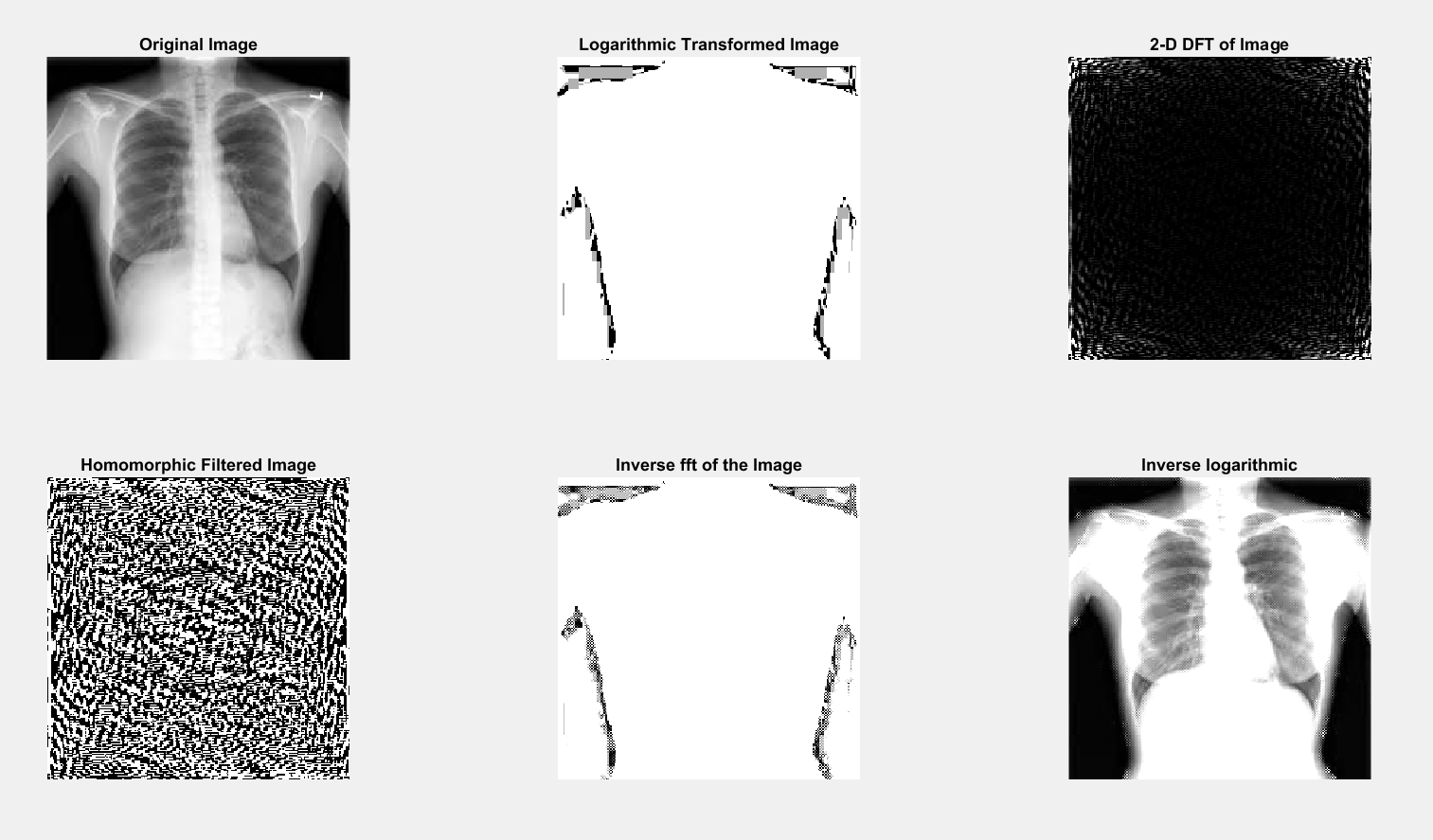


Figure 6.1 Homomorphic filtering

**Results :** Image enhancement using Homomorphic Filtering is performed and implemented.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 7**

## Aim : To write program for Image segmentation using Edge Detectors: Sobel, Prewitt, Roberts without using Matlab built-in function.

**Apparatus/Tools Required :** Matlab

**Theory :**

[Edge detection](https://www.mathworks.com/products/image.html) is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for [image segmentation](https://www.mathworks.com/discovery/image-segmentation.html) and data extraction in areas such as image processing, computer vision, and machine vision.

Common [edge detection algorithms](https://www.mathworks.com/help/images/ref/edge.html) include Sobel, Canny, Prewitt, Roberts, and [fuzzy logic](https://www.mathworks.com/products/fuzzy-logic.html) methods.

**Code**

%edge/gradient based filters

clc;

clear all;

image=imread('D:\cameraman.png');

imshow(image);

title('original image');

%sobel operator

z=edge(image,'sobel');

figure;

subplot(1,2,1);

imshow(z);

y=edge(z,'sobel',0.2);

subplot(1,2,2);

imshow(y);

title('sobel filtered image');

% prewitt operator

a=edge(image,'prewitt');

figure;

subplot(1,2,1);

imshow(a);

b=edge(a,'prewitt',0.2);

subplot(1,2,2);

imshow(b);

title('prewitt filtered image');

%roberts operator

d=edge(image,'roberts');

figure;

subplot(1,2,1);

imshow(d);

e=edge(d,'roberts',0.2);

subplot(1,2,2);

imshow(e);

title('roberts filtered image');

%canny operator

f=edge(image,'canny');

figure;

subplot(1,2,1);

imshow(f);

g=edge(f,'canny',0.2);

subplot(1,2,2);

imshow(g);

title('canny filtered image')

%prewitt

clear all;

image=imread('D:\cameraman.png');

maskx=[-1 -1 -1;0 0 0;1 1 1];

masky=[-1 0 1;-1 0 1;-1 0 1];

m=imfilter(image,maskx);

n=imfilter(image,masky);

p=m+n;

subplot(2,2,1);

imshow(p);

title(' output after prewitt filter');

%sobel

clear all;

image=imread('D:\cameraman.png');

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

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

a=imfilter(image,amaskx);

b=imfilter(image,bmasky);

s=a+b;

subplot(2,2,2);

imshow(s);

title('output after sobel filter');

%robert

clear all;

image=imread('D:\cameraman.png');

maskx=[1 0;0 -1];

masky=[0 -1;1 0];

m=imfilter(image,maskx);

n=imfilter(image,masky);

p=m+n;

subplot(2,2,3);

imshow(p);

title('output after robert filter');

**Observations**

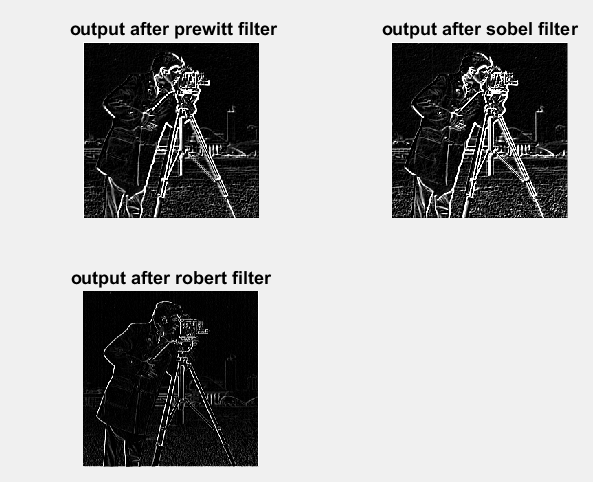


Figure .1 output of prewitt,sobel and Robert filters

**Results:** Image segmentation using Edge Detectors: Sobel, Prewitt, Roberts is performed and plotted.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 8**

**Aim :**

**Apparatus/Tools Required :** Matlab

**Theory :**

Code

%STATISTICAL PROPERTIES FOR COMPARISON OF FILTERRING

clc;

clear all;

a=imread('D:\cameraman.png');

[m,n]=size(a);

total=m\*n;

figure;

imshow(a);

title('original image');

%mean

sum=0;

a=double(a);

for j=1:m

for k=1:n

sum=sum+a(j,k);

end

end

mean=sum/total;

disp('mean is: ');

disp(mean);

%std deviation and variance

dev=0;

for j=1:m

for k=1:n

dev=dev+(a(j,k)-mean)^2;

end

end

dev=sqrt(dev/total);

disp('deviation is: ');

disp(dev);

var=dev^2;

disp('variance is : ');

disp(var);

%horizontal profile density

c=zeros(1,m);

k=m/2;

for i=1:n

c(i)=a(k,i);

end

figure;

subplot(2,1,1);

stem(c);

title('hpd');

%vpd

c=zeros(1,m);

k=m/2;

for i=1:n

c(i)=a(i,k);

end

figure;

subplot(2,1,2);

stem(c);

title('vpd');

%entropy

e=0;

p=imhist(uint8(a));

[m1 n1]=size(p);

p=p/(m\*n);

for i=1:m1

if (p(i)~=0)

e=e-p(i).\*log2(p(i));

end

end

disp('entropy is');

disp(e);

k=imnoise(a,'salt & pepper',0.03);

snr1=snr(a,k);

disp('snr is');

disp(snr1);

max1=max(a(:));

min1=min(a(:));

mse1=std(a(:));

snr2=20\*log10((max1-min1)./mse1);

psnr=20\*log10(max1)-10\*log10(mse1);

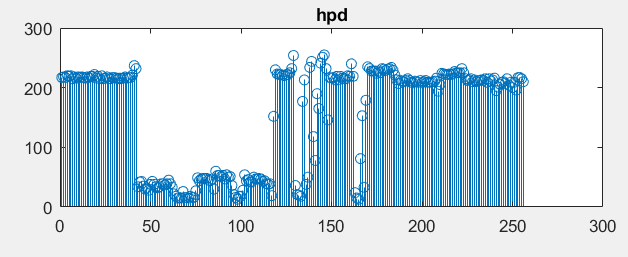
disp('psnr is');

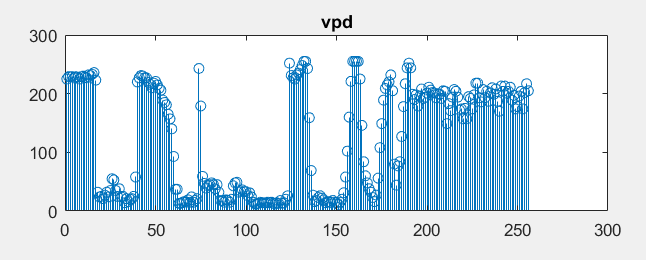
disp(psnr);

disp('snr2 is');

disp(snr2);

**Observations**



**Results:**

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 9**

**Aim :** Image restoration using wiener filter.

**Apparatus/Tools Required :** Matlab

**Theory :**

The most important technique for removal of blur in images due to linear motion or unfocussed optics is the Wiener filter. From a signal processing standpoint, blurring due to linear motion in a photograph is the result of poor sampling. Each pixel in a digital representation of the photograph should represent the intensity of a single stationary point in front of the camera. Unfortunately, if the shutter speed is too slow and the camera is in motion, a given pixel will be an amalgram of intensities from points along the line of the camera's motion. This is a two-dimensional analogy to

### G(u,v)=F(u,v).H(u,v)

where F is the fourier transform of an "ideal" version of a given image, and H is the blurring function. In this case H is a sinc function: if three pixels in a line contain info from the same point on an image, the digital image will seem to have been convolved with a three-point boxcar in the time domain. Ideally one could reverse-engineer a Fest, or F estimate, if G and H are known. This technique is inverse fitering.

**Code**

%wiener filter for image restoration

clc;

clear all;

Ioriginal=imread('D:\cameraman.png');

figure;

imshow(Ioriginal)

title('original image')

PSF=fspecial('motion',20,10);

Idouble=im2double(Ioriginal);

blurred=imfilter(Idouble,PSF,'conv','circular');

figure;

imshow(blurred);

title('blurred image');

wnr1=deconvwnr(blurred,PSF);

figure;

imshow(wnr1)

title('restored image after restoration');

%add a zero mean gaussian noise to blurred noise

noise\_mean=0;

noise\_variance=0.0001;

blurred\_noisyimage=imnoise(blurred,'gaussian');

figure;

imshow(blurred\_noisyimage)

title('blurred and noisy image');

wnr2=deconvwnr(blurred\_noisyimage,PSF);

figure;

imshow(wnr2)

title('restoration of blurred noisy image with nsr=0');

%with realistic estimate of noise

signal\_variance=var(Idouble(:));

NSR=noise\_variance/signal\_variance;

wnr3=deconvwnr(blurred\_noisyimage,PSF,NSR);

figure;

imshow(wnr3)

title('restoration of blurred noisy image in presence of estimated noise')

max1=max(Ioriginal(:));

min1=min(Ioriginal(:));

mse1=std(Ioriginal(:));

snr2=20\*log10((max1-min1)./mse1);

psnr=20\*log10(max1)-10\*log10(mse1);

disp('psnr OF ORIGINAL IMAGE');

disp(psnr);

disp('snr2 OF ORIGINAL IMAGE');

disp(snr2);

**Observations**



Figure 9.1 Image restoration using weiner filter

**Results:** Image restoration is performed and plotted using weiner filter.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 10**

**Aim :** To write a program for image compression using DCT and recover the image using IDCT

**Apparatus/Tools Required :** Matlab

**Theory :**

A discrete cosine transform (DCT) expresses a finite sequence of [data points](https://en.wikipedia.org/wiki/Data_points) in terms of a sum of [cosine](https://en.wikipedia.org/wiki/Cosine) functions oscillating at different [frequencies](https://en.wikipedia.org/wiki/Frequency). The DCT, first proposed by [Nasir Ahmed](https://en.wikipedia.org/wiki/N._Ahmed) in 1972, is a widely used transformation technique in [signal processing](https://en.wikipedia.org/wiki/Signal_processing) and [data compression](https://en.wikipedia.org/wiki/Data_compression). It is used in most [digital media](https://en.wikipedia.org/wiki/Digital_media), including [digital images](https://en.wikipedia.org/wiki/Digital_images) (such as [JPEG](https://en.wikipedia.org/wiki/JPEG) and [HEIF](https://en.wikipedia.org/wiki/HEIF), where small high-frequency components can be discarded), [digital video](https://en.wikipedia.org/wiki/Digital_video) (such as [MPEG](https://en.wikipedia.org/wiki/MPEG) and [H.26x](https://en.wikipedia.org/wiki/H.26x)), [digital audio](https://en.wikipedia.org/wiki/Digital_audio) (such as [Dolby Digital](https://en.wikipedia.org/wiki/Dolby_Digital), [MP3](https://en.wikipedia.org/wiki/MP3) and [AAC](https://en.wikipedia.org/wiki/Advanced_Audio_Coding)), [digital television](https://en.wikipedia.org/wiki/Digital_television) (such as [SDTV](https://en.wikipedia.org/wiki/SDTV), [HDTV](https://en.wikipedia.org/wiki/HDTV) and [VOD](https://en.wikipedia.org/wiki/Video_on_demand)), [digital radio](https://en.wikipedia.org/wiki/Digital_radio) (such as [AAC+](https://en.wikipedia.org/wiki/AAC%2B) and [DAB+](https://en.wikipedia.org/wiki/DAB%2B)), and [speech coding](https://en.wikipedia.org/wiki/Speech_coding) (such as [AAC-LD](https://en.wikipedia.org/wiki/AAC-LD), [Siren](https://en.wikipedia.org/wiki/Siren_(codec)) and [Opus](https://en.wikipedia.org/wiki/Opus_(audio_format))). DCTs are also important to numerous other applications in [science and engineering](https://en.wikipedia.org/wiki/Science_and_engineering), such as [digital signal processing](https://en.wikipedia.org/wiki/Digital_signal_processing), [telecommunication](https://en.wikipedia.org/wiki/Telecommunication) devices, reducing [network bandwidth](https://en.wikipedia.org/wiki/Network_bandwidth) usage, and [spectral methods](https://en.wikipedia.org/wiki/Spectral_method) for the numerical solution of [partial differential equations](https://en.wikipedia.org/wiki/Partial_differential_equations).

Code

clc;

clear all;

I=imread('D:\cameraman.png');

I = double(I);

[r c]=size(I);

d=double(I);

if (r==c)

N=c;

else

error('ERROR');

end

%using in built function

figure('Name','DCT using inbuilt function');

subplot(1,3,1);

imshow(uint8(d));

title('Input image');

DCT\_func=dct2(d);

subplot(1,3,2);

imshow(uint8(DCT\_func));

title('DCT of the image');

IDCT\_func=idct2(DCT\_func);

subplot(1,3,3);

imshow(uint8(IDCT\_func));

title('Decompressed image');

error1=immse(I,IDCT\_func);

%without using in built function

for u=0:N-1

for v=0:N-1

if (u==0)

c(u+1,v+1)=sqrt(1/N);

else

c(u+1,v+1)=sqrt(2/N)\*cos((2\*v+1)\*pi\*u/(2\*N));

end

end

end

DCT2=c\*d\*(c');

DCT2\_inverse=(c')\*DCT2\*c;

figure('Name','DCT without inbuilt function ');

subplot(1,2,1);

imshow(uint8(DCT2));

title('DCT of the image');

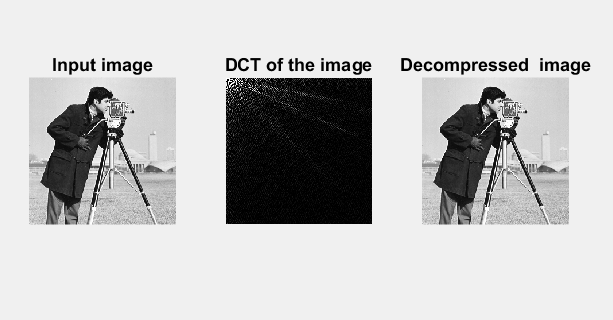
subplot(1,2,2);

imshow(uint8(DCT2\_inverse));

title('Decompressed image');

error2=immse(I,DCT2\_inverse);

**Observations**



**Results:** image compression using DCT and recover the image using IDCT is performed and plotted.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 11**

**Aim :** Image segmentation using thresholding.

**Apparatus/Tools Required :** Matlab

**Theory :**

To understand an image, it has to be divided into different meaningful parts called objects which can be easily identified and depicts some information. This division process is called image segmentation and thresholding is one of the popular techniques for image segmentation. It has low computational cost when compared to other algorithms Image thresholding works on the principle of pixel classification. It divides an image into segments depending upon the pixel attributes. This techniques applies on each pixel and by comparing it to a specific threshold value decides whether the picture belongs to an object or background. For gray images, the segmentation is carry out on the basis of image gray levels where the brighter part of an image is object and darker is background. The objects and background of gray level images can be easily identified, but the process becomes more complicated for color or textured images. So, for color and textured images requires much more attention and processing to get segmented [[1](https://ukdiss.com/examples/global-thresholding-image-segmentation-techniques.php#_ENREF_1)]. The thresholding is also affected by the noise and artefacts present in image. Usually some preprocessing steps are applied to reduce the noise and artefacts effects.

Code

%image segementation using thresholding

%simple /basic thresholding binarization

clc;

clear all;

close all;

a= imread('D:\cameraman.png');

subplot(3,3,1);

imshow(a);

title('original image');

%simple thresholding takes a default level of 0.3

level=0.3

subplot(3,3,2);

segimage1=im2bw(a,level);

imshow(segimage1);

title('simple thresholding at 0.3');

subplot(3,3,3);

imshow(a>153);

title('simple thresholding at 0.6');

temp=a;

[m n]=find(a<25);

for j=1:length(m)

temp(m(j),n(j))=0;

end

[o p]=find(a>=25 & a<=230);

for j=1:length(m)

temp(m(j),n(j))=0.4;

end

[q r]=find(a>230);

for j=1:length(m)

temp(m(j),n(j))=0;

end

subplot(3,3,4);

segimage2=im2bw(temp);

imshow(segimage2);

title('multiple thresholding between 25 & 230');

level3=graythresh(a);

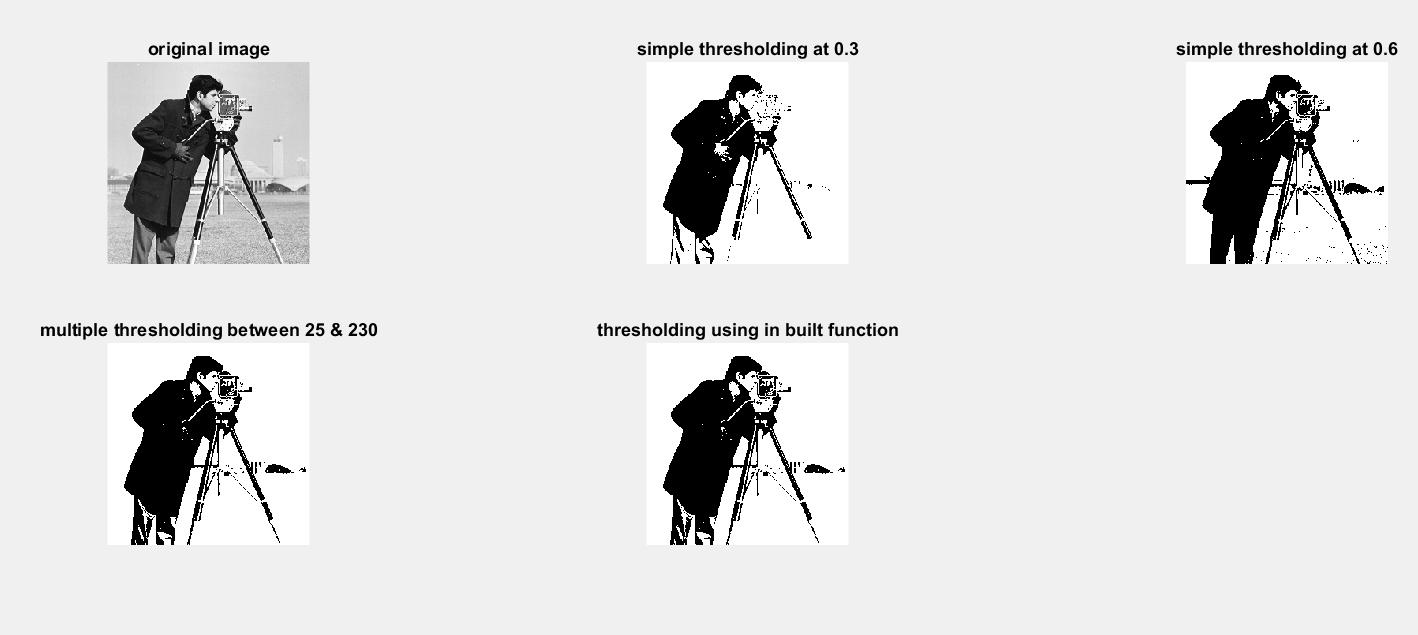
subplot(3,3,5);

segimage3=im2bw(a,level3);

imshow(segimage3);

title('thresholding using in built function')

**Observations**

Figure 11.1 Image segmentation using thresholding.

**Results:** Image segmentation is done using thresholding technique.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |

**Experiment No. 12**

**Aim :** RGB to GRAY conversion using lightness ,average and luminosity methods.

**Apparatus/Tools Required :** Matlab

**Theory :**

Humans perceive color through wavelength-sensitive sensory cells called cones. There are three different types of cones, each with a different sensitivity to electromagnetic radiation (light) of different wavelength. One type of cone is mainly sensitive to red light, one to green light, and one to blue light. By emitting a controlled combination of these three basic colors (red,green and blue), and hence stimulate the three types of cones at will, we are able to generate almost any perceivable color. This is the reasoning behind why color images are often stored as three separate image matrices; one storing the amount of red (R) in each pixel, one the amount of green (G) and one the amount of blue (B). We call such color images as stored in an RGB format.

In grayscale images, however, we do not differentiate how much we emit of the different colors, we emit the same amount in each channel. What we can differentiate is the total amount of emitted light for each pixel; little light gives dark pixels and much light is perceived as bright pixels.

To store a single colour pixel of an RGB colour image we will need 8\*3 = 24 bits (8 bit for each colour component), but when we convert an RGB image to grayscale image, only 8 bit is required to store a single pixel of the image. So we will need 33 % less memory to store grayscale image than to store an RGB image

Grayscale images are much easier to work within a variety of task like In many morphological operation and image segmentation problem, it is easier to work with single layered image (Grayscale image ) than a three-layered image (RGB colour image It is also easier to distinguish features of an image when we deal with a single layered image.

Code

% RGB to gray conversion using Lightness method , average method and

% luminous method

% Lightness method

close all;

clear all;

clc;

a = imread('D:\balloon1.jpg');

z=a;

ar = z(:, :, 1);

ag = z(:, :, 2);

ab = z(:, :, 3);

[m, n] = size(ar);

for i = 1:m

for j = 1:n

temp1 = max(ar(i, j), ag(i, j));

ma = max(temp1, ab(i, j));

temp2 = max(ar(i, j), ag(i, j));

mi = max(temp2, ab(i, j));

gr1(i, j) = (ma+mi)/2;

end

end

%Averaging method

for i = 1:m

for j = 1:n

gr2(i, j) = (ar(i, j) + ag(i, j) + ab(i, j))/3;

end

end

%luminosity method

for i = 1:m

for j = 1:n

gr3(i, j) = 0.289\*ar(i, j) + 0.5870\*ag(i, j) + 0.1140\*ab(i, j);

end

end

%using inbuilt function

gr4 = rgb2gray(a);

subplot(2,3,1);

imshow(a);

title('original colored image');

subplot(2,3,2);

imshow(gr1);

title('conversion using lightness method');

subplot(2,3,3);

imshow(gr2);

title('conversion using average method');

subplot(2,3,4);

imshow(gr3);

title('conversion using luminosity method');

subplot(2,3,5);

imshow(gr4);

title('conversion using inbuilt function');

**Observations**

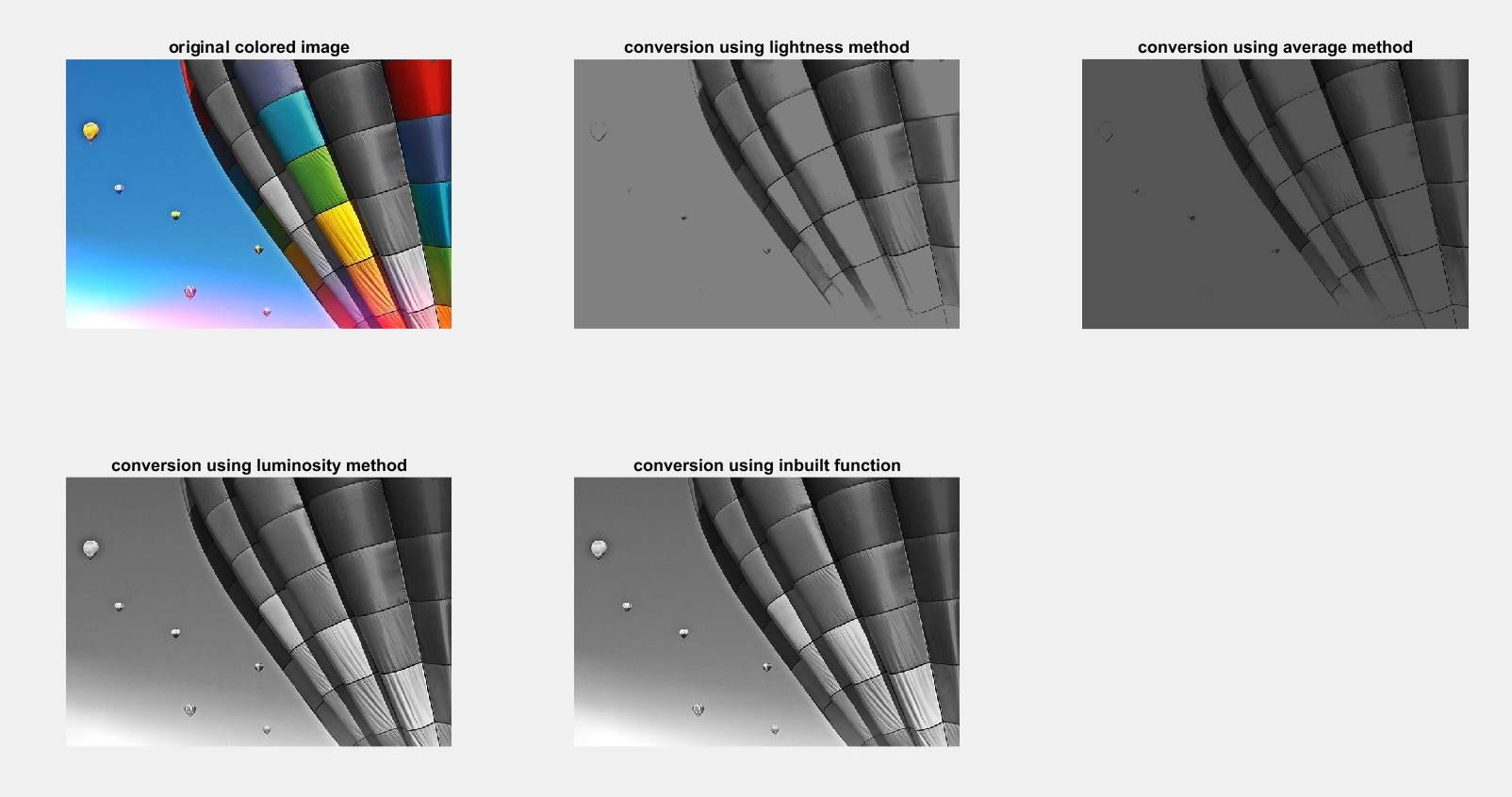


Figure 13.1 RGB to gray scale using lightness,average and luminosity method

**Results:** we have completed the conversion of the rgb to gray scale using lightness ,average and luminosity methods.

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIA | MAX MARKS | OBTAINED MARKS | COMMENTS |
| CONCEPT | 2 |  |  |
| IMPLEMENTATION | 2 |  |  |
| PERFORMANCE | 2 |  |  |
| TOTAL | 6 |  |  |