Indian Institute of Information Technology Nagpur

Digital Image Processing
Lab Report

Submitted By :
Harsh Deshpande (BT17ECE032)

6th Semester,
Electronics and Communication Engineering Dept.

Submitted To :
Dr. Tapan Kumar Jain
Assistant Professor
Electronics and Communication Engineering Dept.

Contents:

TASK 1 : Histogram Equalisation

Aim / Objective: To implement Histogram equalisation.

Software: MATLAB

Theory:

Histogram equalization is a method in image processing of contact adjustment using the image's histogram. Histogram equalization is the best method for image enhancement. It provides better quality of images without loss of any information. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

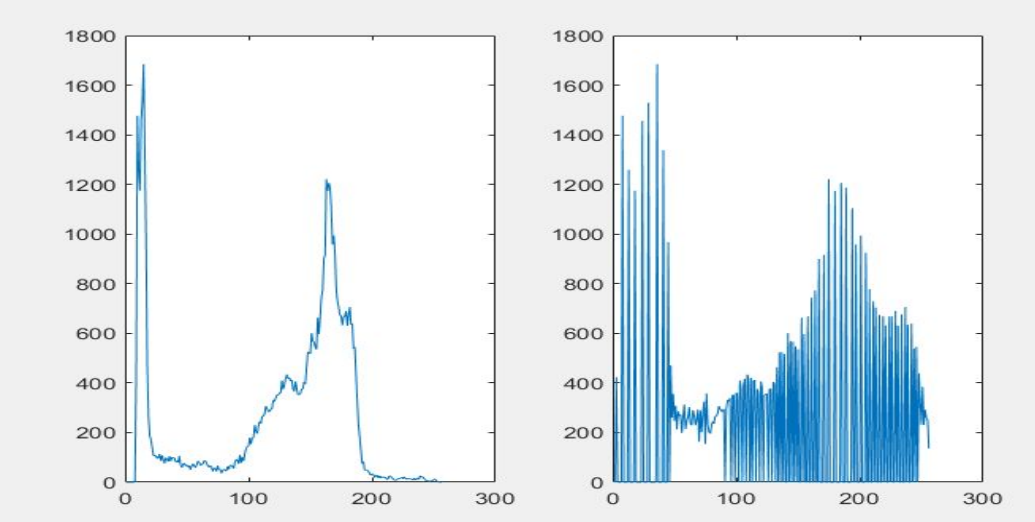
The method is useful in images with backgrounds and foregrounds that are both bright or both dark. In particular, the method can lead to better views of bone structure in x-ray images, and to better detail in photographs that are over or under-exposed. A key advantage of the method is that it is a fairly straightforward technique and an invertible operator. So in theory, if the histogram equalization function is known, then the original histogram can be recovered.

Code:

```
% Histogram_Equalization
%rd = rgb2gray(imread('cameraman.tif'));
rdi = imread('cameraman.tif');
s = size(rdi);
rdi = double(rdi);

hist1 = zeros(1,256);
for i = 1:s(1)            %Creating histogram of the original image
        for j = 1:s(2)
        for k = 0:255
        if rdi(i,j) == k
        hist1(k+1) = hist1(k+1)+1;
        end
        end
        end
end
%Calculating PDF
pdf = hist1 * (1/(s(1)*s(2)));
%Calculating CDF
cdf = zeros(1,256);
cdf(1) = pdf(1);
for i = 2:255
        cdf(i) = cdf(i-1) + pdf(i);
end
cdf = round(255*cdf);   %Round off
%Reconstructing the output image from cdf corresponding freq
rn = zeros(1,256);
for i = 1:s(1)
        for j = 1:s(2)
        for k = 0:255
        if rdi(i,j) == k
                rn(i,j) = cdf(k+1);
        end
        end
        end
end
```

```
%Creating Equalized Histogram
hist2 = zeros(1,256);
for i = 1:s(1)
        for j = 1:s(2)
        for k = 0:255
        if rn(i,j) == k
        hist2(k+1) = hist2(k+1)+1;
        end
        end
        end
end
subplot(121)
imshow(uint8(rdi));
subplot(122)
imshow(uint8(rn))
figure
subplot(121)
plot(hist1)
subplot(122)
plot(hist2)
```

Output:



Conclusion:

As you can clearly see from the images that the new image contrast has been enhanced and its histogram has also been equalized.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK1.m

TASK 2 : Display Red Green Blue color Images

Aim / Objective: To implement code for separating R-G-B components of a given image.

Software: MATLAB

Theory:

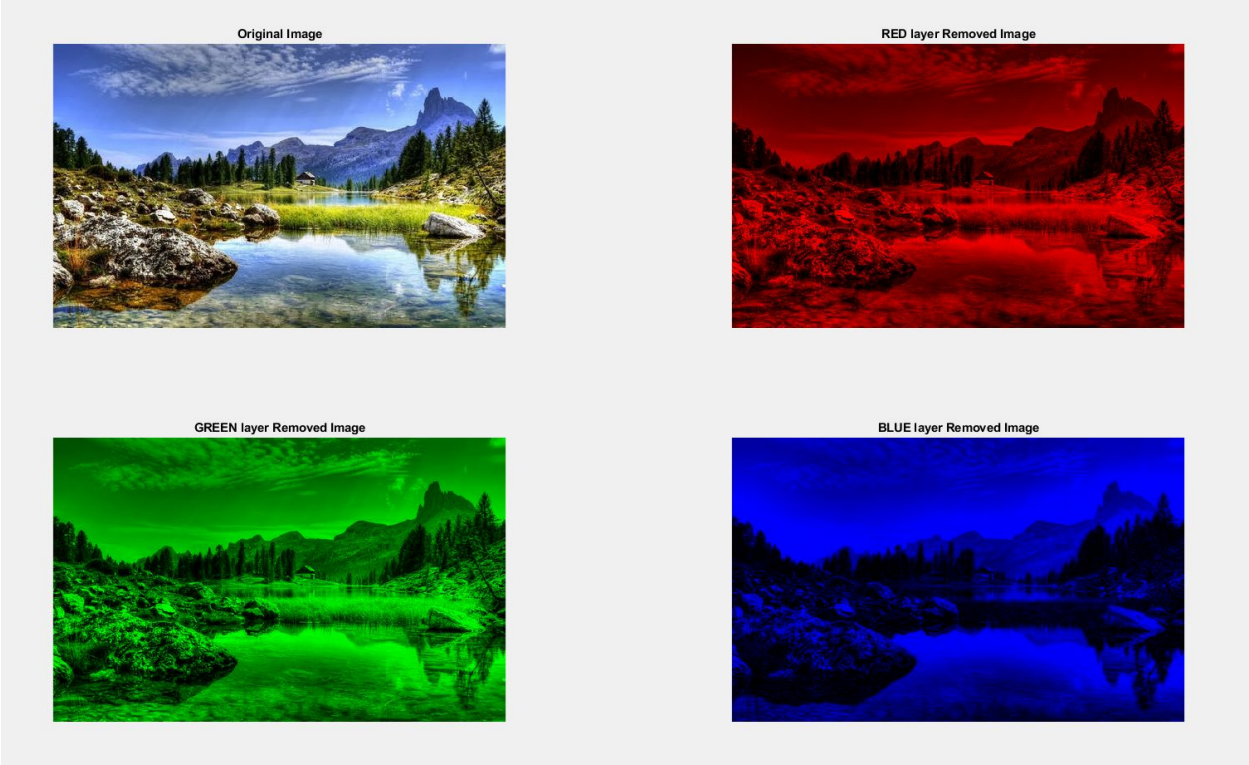
The RGB color model is an additive color model[1] in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue. The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography. Before the electronic age, the RGB color model already had a solid theory behind it, based on human perception of colors. RGB is a *device-dependent* color model: different devices detect or reproduce a given RGB value differently, since the color elements (such as phosphors or dyes) and their response to the individual R, G, and B levels vary from manufacturer to manufacturer, or even in the same device over time. Thus an RGB value does not define the same *color* across devices without some kind of color management.

Code:

```
clc; close all; clear;
I = imread('landscape.jpg');
subplot(2,2,1);
imshow(uint8(I));
title('Original Image')

R=I;
R(:,:,[2 3])= 0;
subplot(2,2,2);
imshow(uint8(R));
title('RED layer Removed Image')

G=I;
G(:,:,[1 3])= 0;
```

```
subplot(2,2,3);
imshow(uint8(G));
title('GREEN layer Removed Image')

B=I;
B(:,:,[1 2]) = 0;
subplot(2,2,4);
imshow(uint8(B));

title('BLUE layer Removed Image')
```

Output:



Conclusion:

Red Green Blue color component of colored image is separated in layers using matlab and is observed successfully.

GitHub Link:
https://github.com/Harsh-D/DIP/blob/master/DIP_TASK2.m

TASK 3 : Compression by Bit-slicing

Aim: To implement

Objective:

Software: MATLAB

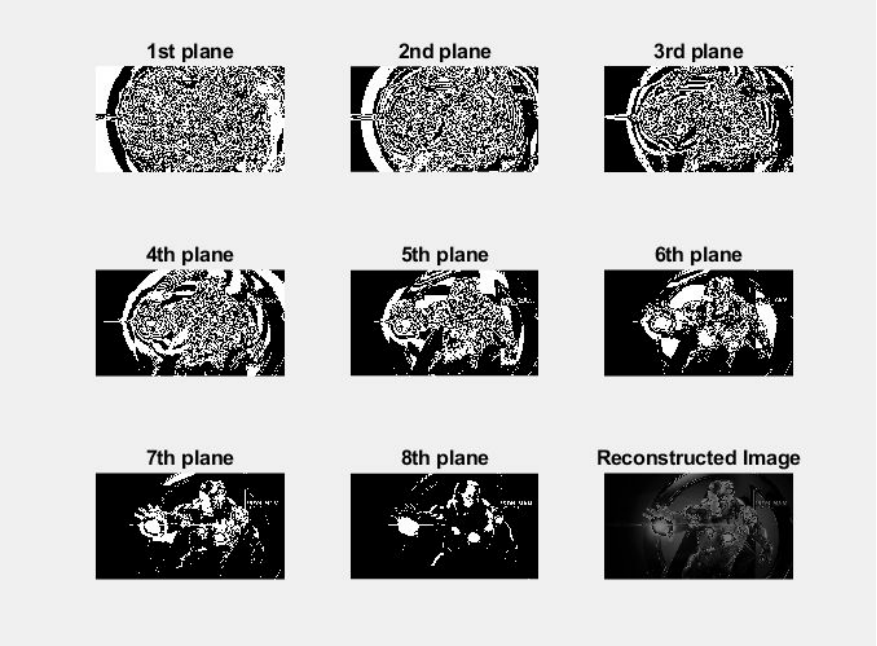
Theory:

Code:

```
clc; clear; close all;
b = imread('ironman.jpg');
a = rgb2gray(b);
[r, c, p]=size(a);
if (p==3)
      error('Input image should be Grayscale')
else

[plane1,plane2,plane3,plane4,plane5,plane6,plane7,plane8]=bitplane_code(a);
end
figure;
      subplot(3,3,1);imshow(plane1);title('1st plane');
      subplot(3,3,2);imshow(plane2);title('2nd plane');
      subplot(3,3,3);imshow(plane3);title('3rd plane');
      subplot(3,3,4);imshow(plane4);title('4th plane');
      subplot(3,3,5);imshow(plane5);title('5th plane');
      subplot(3,3,6);imshow(plane6);title('6th plane');
      subplot(3,3,7);imshow(plane7);title('7th plane');
      subplot(3,3,8);imshow(plane8);title('8th plane')
      plane8 = 0;

rec=plane1+plane2*2+plane3*4+plane4*8+plane5*16+plane6*32+plane7*64+plane8*
128;
      subplot(3,3,9);imshow(uint8(rec));title('Reconstructed Image')
```

```
function [plane1, plane2, plane3, plane4 ,plane5, plane6, plane7,
plane8]=bitplane_code(img)

[row ,col]=size(img);
b=zeros(row,col,8);

for k=1:8
      for i=1:row
      for j=1:col
            b(i,j,k)=bitget(img(i,j),k);
      end
      end
end
plane1=b(:,:,1);
plane2=b(:,:,2);
plane3=b(:,:,3);
plane4=b(:,:,4);
plane5=b(:,:,5);
plane6=b(:,:,6);
plane7=b(:,:,7);
plane8=b(:,:,8);
end
```

Output:



Conclusion:

The experiment was completed successfully. The Concept of Bit plane slicing was used and a given input image was sliced into 8 bitplanes successfully and the output was verified.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK3.m

TASK 4 : Nearest Neighbour Pixel Algorithm

Aim / Objective: To implement Nearest Neighbour Pixel Algorithm

Software: MATLAB

Code:

```
clc;clear;close all;
img = imread('img1.jpg');
[m,n] = size(img);
cimg = [zeros(m,1),img,zeros(m,1)];
rimg = [zeros(1,n+2);cimg;zeros(1,n+2)];
fimg = uint8(zeros(m,n));
for i = 2:m
    for j = 2:n
        fimg(i-1,j-1) = get_val(i,j,rimg);
    end
end
subplot(121)
imshow(img)
title('Original Image');
subplot(122)
imshow(fimg)
title('Modified image')
```

```
function val = get_val(i,j,in_img)
im = zeros(1,8);
```

```
if in_img(i,j) < in_img(i,j+1)   im(1) = 1;else im(1) =
0;end
if in_img(i,j) < in_img(i-1,j+1) im(2) = 1;else im(2) =
0;end
if in_img(i,j) < in_img(i-1,j)   im(3) = 1;else im(3) =
0;end
if in_img(i,j) < in_img(i-1,j-1) im(4) = 1;else im(4) =
0;end
if in_img(i,j) < in_img(i,j-1)   im(5) = 1;else im(5) =
0;end
if in_img(i,j) < in_img(i+1,j-1) im(6) = 1;else im(6) =
0;end
if in_img(i,j) < in_img(i+1,j)   im(7) = 1;else im(7) =
0;end
if in_img(i,j) < in_img(i+1,j+1) im(8) = 1;else im(8) =
0;end
val = bi2de(im);
end
```

Output:

Conclusion:

The experiment was conducted successfully.For a given input image, the Moore Neighbourhood concept was implemented and new modified values of each pixel were saved in a new matrix and the output image was viewed and verified

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK4.m

TASK 5 : Watermarking in any bit plane

Aim / Objective: To write code in MATLAB to insert the signature of a person as a watermark on any bit plane of the original image and observe the output.

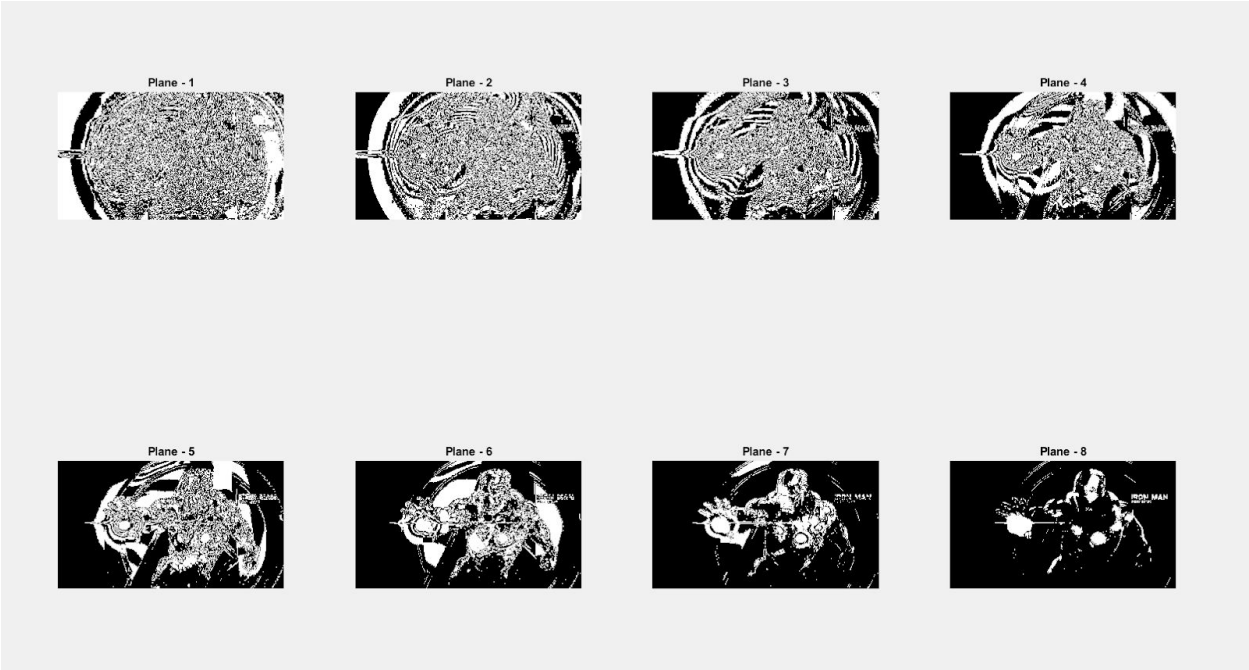
Software: MATLAB

Code:

```
clc;clear;close all;
a = rgb2gray(imread('ironman.jpg'));
sg = rgb2gray(imread('img_white.png'));
[m,n] = size(a);
b = double(a);
s = double(sg);
c = de2bi(b,8,'right-msb');
g = de2bi(s,8,'right-msb');
imshow(a)
title('Original Image')
figure
for i = 1:8
    f = reshape(bi2de(c(:,i)),m,n);
    subplot(2,4,i)
    imshow(f)
    title(['Plane - ',num2str(i)])
end
figure
for i = 1:8
    d = c;
    if i == 1
```

```
        d(:,1) = g(:,5);
        d1 = reshape(uint8(bi2de(d)),m,n);
        subplot(2,4,1)
        imshow(uint8(d1))
        title([' ',num2str(i),'st plane modified'])
    elseif i == 2
        d(:,2) = g(:,5);
        d1 = reshape(bi2de(d),m,n);
        subplot(2,4,2)
        imshow(uint8(d1))
        title([' ',num2str(i),'nd plane modified'])
    elseif i == 3
        d(:,3) = g(:,5);
        d1 = reshape(bi2de(d),m,n);
        subplot(2,4,3)
        imshow(uint8(d1))
        title([' ',num2str(i),'rd plane modified'])
    else
        d(:,i) = g(:,5);
        d3 = reshape(bi2de(d),m,n);
        subplot(2,4,i)
        imshow(uint8(d3))
        title([' ',num2str(i),'th plane modified'])
    end
end
```

Output:



Conclusion:

The experiment was conducted successfully. The LSB plane of the input image was replaced with the bit plane of this watermark image and the output was obtained and was verified.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK5.m

TASK 6 : Histogram Stretching

Aim / Objective: To write code in MATLAB for histogram stretching of an image and observe the output.

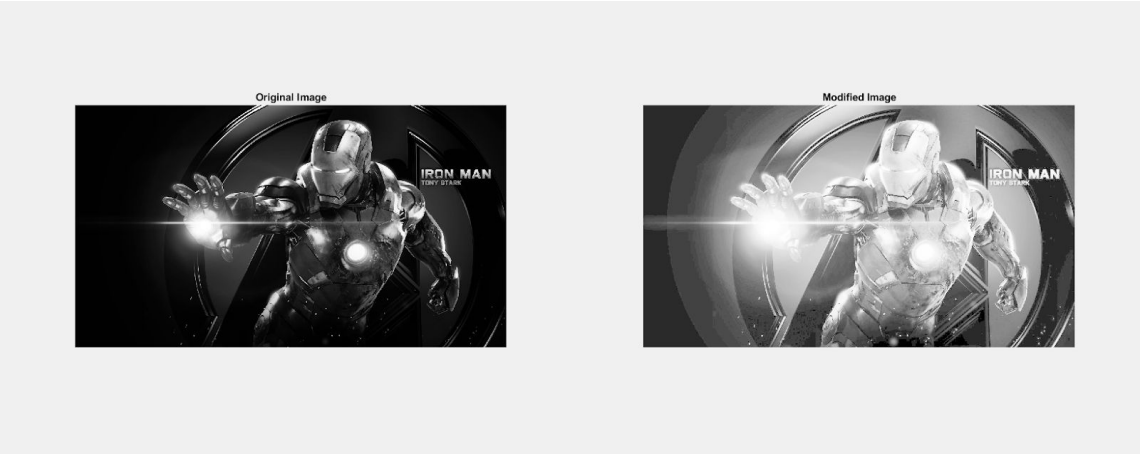
Software: MATLAB

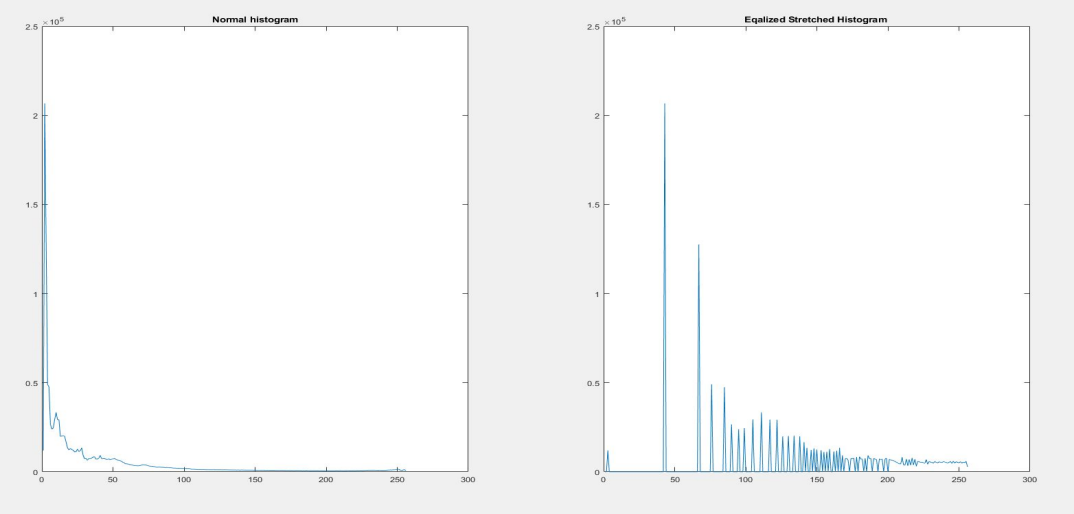
Code:

```
clc; clear; close all;
b = imread('ironman.jpg');
a = rgb2gray(b);
[r, c, p]=size(a);
if (p==3)
     error('Input image should be Grayscale')
else

[plane1,plane2,plane3,plane4,plane5,plane6,plane7,plane8]=b
itplane_code(a);
end
figure;
     subplot(3,3,1);imshow(plane1);title('1st plane');
     subplot(3,3,2);imshow(plane2);title('2nd plane');
     subplot(3,3,3);imshow(plane3);title('3rd plane');
     subplot(3,3,4);imshow(plane4);title('4th plane');
     subplot(3,3,5);imshow(plane5);title('5th plane');
     subplot(3,3,6);imshow(plane6);title('6th plane');
     subplot(3,3,7);imshow(plane7);title('7th plane');
```

```
     subplot(3,3,8);imshow(plane8);title('8th plane')
     plane8 = 0;

rec=plane1+plane2*2+plane3*4+plane4*8+plane5*16+plane6*32+p
lane7*64+plane8*128;
     subplot(3,3,9);imshow(uint8(rec));title('Reconstructed
Image')


function [plane1, plane2, plane3, plane4 ,plane5, plane6,
plane7, plane8]=bitplane_code(img)

[row ,col]=size(img);
b=zeros(row,col,8);

for k=1:8
     for i=1:row
     for j=1:col
          b(i,j,k)=bitget(img(i,j),k);
     end
     end
end
plane1=b(:,:,1);
plane2=b(:,:,2);
plane3=b(:,:,3);
plane4=b(:,:,4);
plane5=b(:,:,5);
plane6=b(:,:,6);
plane7=b(:,:,7);
plane8=b(:,:,8);
end
```

Output:





Conclusion:

The experiment was conducted successfully. Histogram stretching was used for contrast enhancement of a given input image and the output observed was verified.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK6.m

TASK 7 : Toboggan Contrast

Aim / Objective: To write code in MATLAB for implementing Toboggan Algorithm on an image and observe the output.

Software: MATLAB

Theory:

Toboggan contrast enhancement is a noniterative single-parameter linear execution time method for selectively augmenting the contrast of multispectral images of arbitrary dimensionality. Toboggan enhancement followed by contrast segmentation is compared with adaptive smoothing, all iterative, multiple parameter, parallel approach that achieves similar results. The segmentation produced by toboggan enhancement followed by contrast segmentation appears equal in quality to that of very complex optimal regional growing segmentation methods. It is concluded that toboggan enhancement is easy to understand and manipulate and is applicable to any image (multispectral, multidimensional) for which one can define a function of local discontinuity at a pixel.

Code:

```
clc;clear;close all;
%Taking a sample image matrix
img = [13 15 17 19; 12 21 20 10; 17 11 21 19; 21 24 23 12];

%Taking the kernel to be mapped with image
kernel = [2 7 6 2; 4 4 3 2; 0 1 0 2; 3 7 6 1];
kernel1 = kernel;
[m,n] = size(img);
%Padding 256 around the image
cimg = [256*ones(m,1),img,256*ones(m,1)];
rimg = [256*ones(1,n+2);cimg;256*ones(1,n+2)];
```

```
cker = [256*ones(m,1),kernel,256*ones(m,1)];
rker = [256*ones(1,n+2);cker;256*ones(1,n+2)];
%Implementing Toboggan algo
for i = 2:m+1
    for j = 2:n+1
        kernel(i-1,j-1) = tob_pix(i,j,rimg,rker);
    end
end

function val = tob_pix(i,j,img,ker)
%Creating a vector of surrounding pixels
m = [img(i,j) img(i,j+1) img(i+1,j+1) img(i+1,j)
img(i+1,j-1) img(i,j-1) ...
    img(i-1,j-1) img(i-1,j) img(i-1,j+1)];
%Finding minimum of them to replace its mapped values in
image
mn = min(m);
p = find(m == mn);
%Replacing the value
if p == 1
    val = ker(i,j);
elseif p == 2
    val = ker(i,j+1);
elseif p == 3
    val = ker(i+1,j+1);
elseif p == 4
    val = ker(i+1,j);
elseif p == 5
    val = ker(i+1,j-1);
elseif p == 6
    val = ker(i,j-1);
elseif p == 7
```

```
    val = ker(i-1,j-1);
elseif p == 8
    val = ker(i-1,j);
elseif p == 9
    val = ker(i-1,j+1);
end
end
```

Output:

```
kernel1 =

     2     7     6     2
     4     4     3     2
     0     1     0     2
     3     7     6     1

img =

    13    15    17    19
    12    21    20    10
    17    11    21    19
    21    24    23    12

kernel =

     4     4     2     2
     1     1     2     2
     1     1     2     2
     1     1     1     1
```

Conclusion:

Hence, the Toboggan contrast enhancement technique is implemented successfully.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK7.m

TASK 8 : Edge Detection (Canny, Sobel)

Aim / Objective: To implement Canny and Sobel filter for detecting edges in the given images.

Software: MATLAB

Theory:
The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

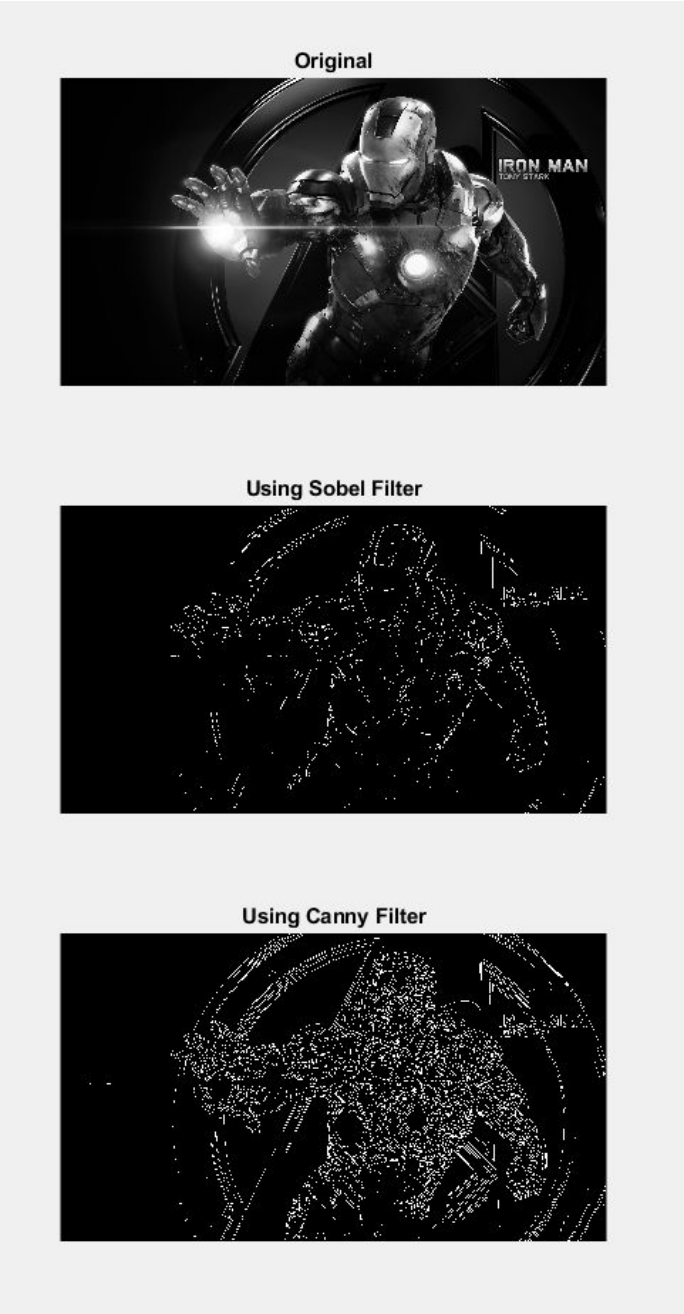
Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations.

Code:

```
% Using Sobel, canny Filter and dwt2
im = rgb2gray(imread('flower.jpg'));

subplot(131)
imshow(im)
title('Original')
ed = edge(im,'sobel');
ed1 = edge(im,'canny');
subplot(132)
imshow(ed)
title('Using Sobel Filter')
subplot(133)
```

```
imshow(ed1)
title('Using Canny Filter')
```

Output:



Conclusion:

Hence the code for implementing edge detection is executed successfully.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK8.m

TASK 9 : High pass and low pass filtering

Aim / Objective: To implement Low pass filter and high pass filter on Given Image.

Software: MATLAB

Theory:

High-Pass Filtering (Sharpening)

A high-pass filter can be used to make an image appear sharper. These filters emphasize fine details in the image – exactly the opposite of the low-pass filter. High-pass filtering works in exactly the same way as low-pass filtering; it just uses a different convolution kernel. In the example below, notice the minus signs for the adjacent pixels. If there is no change in intensity, nothing happens.

Low-Pass Filtering (Blurring)

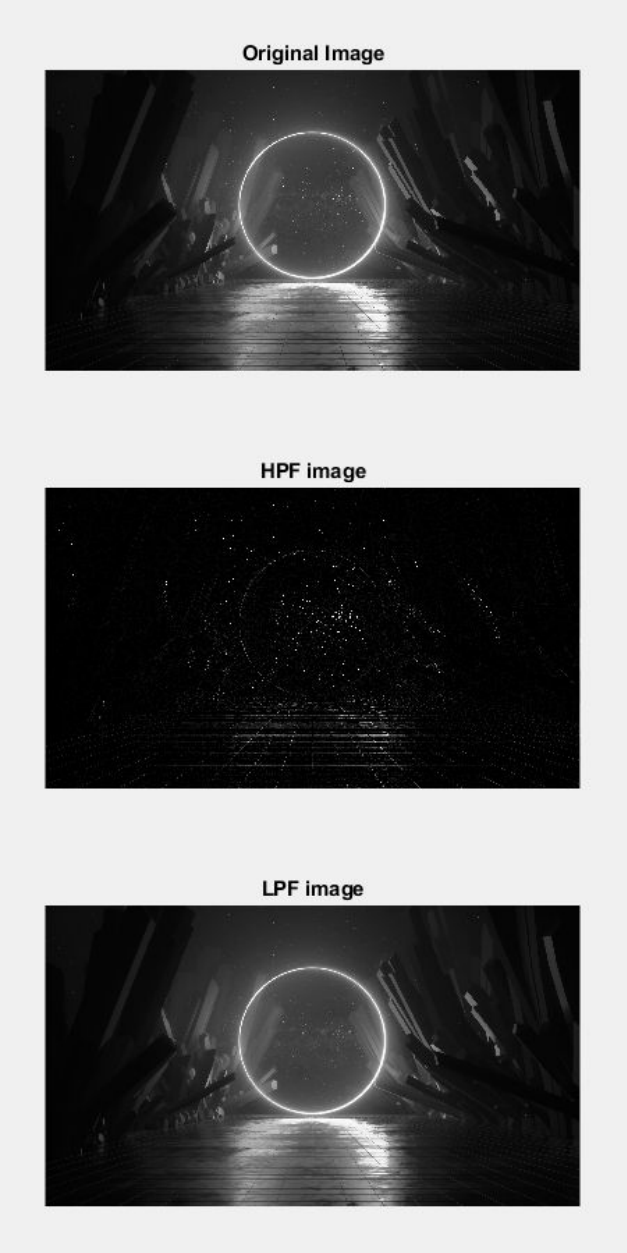
The most basic of filtering operations is called “low-pass”. A low-pass filter, also called “blurring” or “smoothing” filter, averages out rapid changes in intensity. The simplest low-pass filter just calculates the average of a pixel and all of its eight immediate neighbors. The result replaces the original value of the pixel. The process is repeated for every pixel in the image.

Code:

```
ker_hp = [-1 -1 -1;-1 8 -1;-1 -1 -1];
ker_lp = 1/9 * [1 1 1;1 1 1;1 1 1];
img = rgb2gray(imread('neon_ring.jpg'));

subplot(311)
```

```
imshow(img)
title('Original Image')

hp = imfilter(img,ker_hp);
lp = imfilter(img,ker_lp);

subplot(312)
imshow(hp)

title('HPF image')
subplot(313)

imshow(lp)
title('LPF image')
```

Output:



Conclusion:

The experiment was conducted successfully and output was verified.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK9.m

TASK 10 : Run Length Encoding

Aim / Objective: To implement Run Length encoding on a given string.

Software: MATLAB

Theory:
Run-length encoding (RLE) is a form of lossless data compression in which runs of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs. Consider, for example, simple graphic images such as icons, line drawings, Conway's Game of Life, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.

Code:

```
clc; clear;close all;
%Get the input vector
vec = randi([0 1],1,100);
st = sprintf('%d', vec);
fprintf("Original Input String: %s\n",st);

d = func_RLE(vec);
len_out = length(d);
len_ip = length(vec);

%Check for compression ratio
if (len_out / len_ip) <1
    fprintf('Positive Compression ratio = %.2d\n', len_out
/ len_ip)
else
    fprintf('Negative compression.')
end
```

```
function d = func_RLE(x)
 ind=1;
 d(ind,1)=x(1);
 d(ind,2)=1;
 for i=2 :length(x)
    if x(i-1)==x(i)
       d(ind,2)=d(ind,2)+1;
    else ind=ind+1;
         d(ind,1)=x(i);
         d(ind,2)=1;
    end
 end
end
```

Output:

```
Original Input String: 0100101110111111001000011001011000100000011010011010111110010011100010110000010010010111010111010000
Positive Compression ratio = 5.10e-01
>>
```

Conclusion:

The experiment was conducted successfully. The RLE was performed on a random input string of 100 bits and the output was verified.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK10.m

TASK 11 : Zig-Zag Pattern

Aim / Objective: To implement zigzag contrast enhancement on image.

Software: MATLAB

Theory:

The discrete cosine transform is a technique for converting a signal into elementary frequency components. It is widely used for extracting the features. The one dimensional DCT is useful in processing of one-dimensional signals such as speech waveforms. For analysis of the two-dimensional (2-D) signals such as images, a 2-D version of the DCT is required.The DCT works by separating images into parts of differing frequencies .

Code:

```
clc;close all;
im = [1 2 6 7;3 5 8 13;4 9 12 14;10 11 15 16]';
%Creating output matrix
[Out_vec,index] = deal(zeros(size(im,1)*size(im,2),1),1);

%Traversing the zig-zag pattern
for i = 1:size(im,1)
    if mod(size(im,1),2) == 0
        if i < size(im,1)
            if mod(i,2) == 0
                k = i;
                for j = 1:i
                    Out_vec(index)=im(k,j);
                    [k,index] = deal(k-1,index+1);
                end
            elseif mod(i,2) ~= 0
```

```
            k = 1;
            for j = i:-1:1
                Out_vec(index) = im(k,j);
                [k,index] = deal(k+1,index+1);
            end
        end
    elseif i == size(im,1)
        k = i;
        for j = 1:i
            Out_vec(index)=im(k,j);
            [k,index] = deal(k-1,index+1);
        end
        for j = 2:size(im,1)
            if mod(j,2) == 0
                k = j;
                for l = size(im,1):-1:j
                    Out_vec(index) = im(k,l);
                    [k,index] = deal(k+1,index+1);
                end
            elseif mod(j,2) ~= 0
                for l = size(im,1):-1:j
                    Out_vec(index) = im(l,j);
                    [j,index] = deal(j+1,index+1);
                end
            end
        end
    end
elseif mod(size(im,1),2) ~= 0
    if i < size(im,1)
        if mod(i,2) == 0
            k = i;
            for j = 1:i
```

```
                Out_vec(index)=im(k,j);
                [k,index] = deal(k-1,index+1);
            end
        elseif mod(i,2) ~= 0
            k = 1;
            for j = i:-1:1
                Out_vec(index) = im(k,j);
                [k,index] = deal(k+1,index+1);
            end
        end
    elseif i == size(im,1)
        k = 1;
        for j = i:-1:1
            Out_vec(index) = im(k,j);
            [k,index] = deal(k+1,index+1);
        end
        for j = 2:size(im,1)
            if mod(j,2) == 0
                k = j;
                for l = size(im,1):-1:j
                    Out_vec(index) = im(l,k);
                    [k,index] = deal(k+1,index+1);
                end
            elseif mod(j,2) ~= 0
                for l = size(im,1):-1:j
                    Out_vec(index) = im(j,l);
                    [j,index] = deal(j+1,index+1);
                end
            end
        end
    end
end
```

```
end
```

Output:

```
im =

     1     3     4    10
     2     5     9    11
     6     8    12    15
     7    13    14    16


Out_vec =

     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
    11
    12
    13
    14
    15
    16
```

Conclusion:

The experiment was conducted successfully and output was verified.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK11.m

TASK 12 : Watermarking by DWT2

Aim / Objective: To implement matlab code for Watermarking using DWT2

Software: MATLAB

Code:

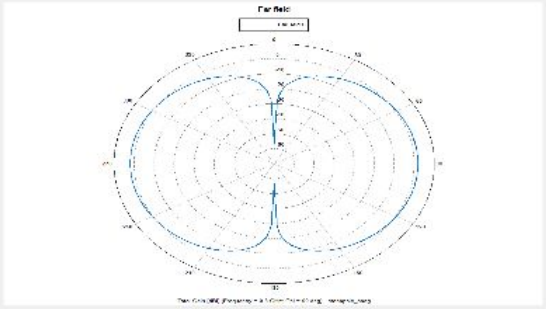
```
watermark('task12_1.jpg','task12_2.png')

function y= watermark(A,B)
%read input image
img=imread(A);
[m, n, p]=size(img);
subplot(3,1,1)
imshow(img);
title('Original Image');

%Get the dwt2 values of that image using Haar Wavelet
[host_LL,host_LH,host_HL,host_HH]=dwt2(img,'haar');

%Read the signature image
water_mark=imread(B);

%Resize it as same as input image
water_mark=imresize(water_mark,[m n]);

subplot(3,1,2)
imshow(water_mark)
title('Watermark');
```

```
%Get the dwt2 values of watermark image using Haar wavelet
[water_mark_LL,water_mark_LH,water_mark_HL,...
    water_mark_HH]=dwt2(water_mark,'haar');
%Taking approximate coefficients of both the images and
adding them
water_marked_LL = host_LL + (0.03*water_mark_LL);
%Reconstructing the image using inverse dwt transform
watermarked=idwt2(water_marked_LL,host_LH,host_HL,host_HH,'
haar');
subplot(3,1,3)
imshow(uint8(watermarked));
title('Watermarked image');
y='Watermarked successfully';
end
```

Output:

Original Image

Watermark



Watermarked image

Conclusion:

The watermarking of the input image with signature watermark was performed successfully.

GitHub Link:

https://github.com/Harsh-D/DIP/blob/master/DIP_TASK12.m