IMPLEMENT IMAGE ENHANCEMENT TECHNIQUE

1.1 Image Smoothing

```
clc; clear all; close all;

im = imread('er.jpg');

h = fspecial('average', [3 3]);

y = imfilter(im, h);

subplot(1,2,1), imshow(im), title('Original Image');

subplot(1,2,2), imshow(y), title('Smoothed Image');
```

1.2 Image Sharpening:

```
clc; clear all; close all;

im = imread('er.jpg');

b = [-1 -1 -1; -1 8 -1; -1 -1 -1];

y = imfilter(double(im), b);

sh = double(im) + y;

subplot(1,3,1), imshow(im), title('Original Image');

subplot(1,3,2), imshow(uint8(sh)), title('Sharpened Image');

subplot(1,3,3), imshow(imsharpen(im)), title('Using imsharpen()');
```

1.3 Contrast Stretching:

1.3.1 Simple Method:

```
clc; clear all ;close all;
im = imread('er.jpg');
%auto bulit in func
stretched = imadjust(im, stretchlim(im));
%mauval
rmax=max(max(im));
rmin=min(min(im));
L=255;
d=rmax-rmin;
mf=L/d;
manual=mf.*abs(im-rmin);

figure;
subplot(2,4,1), imshow(im), title('Original Image');
subplot(2,4,2), imshow(stretched), title('Built-in Stretched');
subplot(2,4,3), imshow(manual), title('Manual Stretched');
subplot(2,4,5), imhist(im), title('Histogram Original');
subplot(2,4,6), imhist(stretched), title('Histogram Built-in');
subplot(2,4,7), imhist(manual), title('Histogram Manual');
```

1.3.2 Gamma Correction:

```
im = double(imread('er.jpg'));
g = 1.5;

im = (im - min(im(:))) / (max(im(:)) - min(im(:)));
im_gamma = im .^ (1/g);

figure;
subplot(2,1,1), imshow(uint8(im*255)), title('Original Image');
subplot(2,1,2), imshow(im_gamma), title('Gamma Corrected Image');
```

1.3.3 Sigmoidal Function:

```
im = double(imread('er.jpg'));
gain = 5;
cutoff = 0.5;

im = (im - min(im(:))) / (max(im(:)) - min(im(:)));  % Normalize to [0,1]
newim = 1 ./ (1 + exp(gain * (cutoff - im)));        % Sigmoid function

figure;
subplot(2,1,1), imshow(uint8(im * 255)), title('Original Image');
 subplot(2,1,2), imshow(newim), title('Sigmoid Enhanced Image');
```

2. HISTOGRAM EQUALIZATION

2.1 Histogram:

```
A = imread('er.jpg');
a = rgb2gray(A);
z = imhist(a);

% Manual histogram equalization
%z = imhist(a);                % Histogram
%p = z / sum(z);               % PDF
%c = cumsum(p);                % CDF
%r = floor(255 * c);           % Mapping
% Plot
figure;
subplot(3,1,1), imshow(a), title('Original Image');
subplot(3,1,2), imshow(uint8(a+3)), title('Equalized Image');
subplot(3,1,3), stem(0:255, z), title('Histogram');
```

2.2 Histogram Equalization:

```
clc; clear all ; close all;
GIm = imread('er.jpg');
```

```
% Manual Histogram Equalization
HIm = histeq(GIm + 5);
% Built-in Equalization
en = histeq(GIm);
im =cat(2,HIm,HIm,HIm);
im=rgb2gray(im);
% Plot
figure;
subplot(3,2,1), imshow(GIm), title('Original Image');
subplot(3,2,2), imhist(GIm), title('Original Histogram');
subplot(3,2,3), imshow(im), title('Manual Equalized Image');
subplot(3,2,4), imhist(im), title('Manual Histogram');
subplot(3,2,5), imshow(en), title('Built-in histeq Image');
subplot(3,2,6), imhist(en), title('Built-in Histogram');
```

2.3 Histogram Specification (Matching):

```
clc; clear; close all;
% Input and Reference images
I1 = rgb2gray(imread('img.jpg'));
I2 = rgb2gray(imread('er.jpg'));
% Histogram and CDF
h1 = imhist(I1);
h2 = imhist(I2);
cdf1 = cumsum(h1) / numel(I1);
cdf2 = cumsum(h2) / numel(I2);
% Remove duplicate values in cdf2 for interpolation
[x, idx] = unique(cdf2);
map = interp1(x, idx-1, cdf1, 'nearest', 'extrap');
% Apply mapping
matched = uint8(map(double(I1)+1));
% Display results in a single figure
figure;
subplot(2,3,1), imshow(I1), title('Input Image');
subplot(2,3,2), imshow(I2), title('Reference Image');
subplot(2,3,3), imshow(matched), title('Matched Image');

subplot(2,3,4), imhist(I1), title('Input Histogram');
subplot(2,3,5), imhist(I2), title('Reference Histogram');
```

```
subplot(2,3,6), imhist(matched), title('Matched Histogram');
```

2.4 Local Histogram Equalization:

```
clc; clear; close all;

% Read color image

A = imread('er.jpg');

% Define local window size

M = 10;

N = 20;

% Apply adapthisteq to each channel separately

R = adapthisteq(A(:,:,1), 'NumTiles', [round(size(A,1)/M) round(size(A,2)/N)], 'ClipLimit', 0.01);

G = adapthisteq(A(:,:,2), 'NumTiles', [round(size(A,1)/M) round(size(A,2)/N)], 'ClipLimit', 0.01);

B = adapthisteq(A(:,:,3), 'NumTiles', [round(size(A,1)/M) round(size(A,2)/N)], 'ClipLimit', 0.01);

% Combine channels back into RGB image

Img = cat(3, R, G, B);

% Display results

figure;

subplot(2,2,1), imshow(A), title('Original Color Image');

subplot(2,2,2), imshow(Img), title('Local Histogram Equalized Color Image');

% Show histograms of one channel for comparison (Red channel)

subplot(2,2,3), imhist(A), title('Original Red Channel Histogram');

subplot(2,2,4), imhist(Img), title('Equalized Red Channel Histogram');
```

3. IMAGE RESTORATION

```
% Read and convert image

degraded_image = im2double(imread('er.jpg'));

% Create motion blur PSF

psf = fspecial('motion', 20, 45);

% Simulate degradation and add Gaussian noise

degraded_noisy = imnoise(imfilter(degraded_image, psf, 'conv', 'circular'), 'gaussian', 0, 0.001);
```

% Estimate noise-to-signal ratio (NSR)

```
nsr = 0.001 / var(degraded_image(:));
```

% Wiener filter restoration

```
restored_image = deconvwnr(degraded_noisy, psf, nsr);
```

% Display results

```
figure;
subplot(1,3,1), imshow(degraded_image), title('Original Image');
subplot(1,3,2), imshow(degraded_noisy), title('Noisy Degraded Image');
subplot(1,3,3), imshow(restored_image), title('Restored Image (Wiener Filter)');
```

4. IMPLEMENT IMAGE FILTERING

% Read and convert to grayscale

```
image = imread('er.jpg');
gray_image = rgb2gray(image);
```

% Filters

```
h_lowpass = fspecial('average', 3);
h_highpass = fspecial('sobel');
```

% Apply filters

```
lowpass_image = imfilter(gray_image, h_lowpass);
highpass_image = imfilter(gray_image, h_highpass);
median_filtered_image = medfilt2(gray_image);
sharp_image = imsharpen(gray_image);
gaussian_filtered_image = imgaussfilt(gray_image, 2);
```

% Display results

```
figure;
subplot(2,3,1), imshow(gray_image), title('Original Image');
subplot(2,3,2), imshow(lowpass_image), title('Low-pass Filter');
subplot(2,3,3), imshow(highpass_image), title('High-pass Filter');
subplot(2,3,4), imshow(median_filtered_image), title('Median Filter');
subplot(2,3,5), imshow(sharp_image), title('Sharpened Image');
subplot(2,3,6), imshow(gaussian_filtered_image), title('Gaussian Filter');
```

5. EDGE DETECTION USING OPERATORS (ROBERTS,PREWITTS AND SOBELS OPERATORS)

5.1 Using Built-in Functions:

```
close all; clear;
I = rgb2gray(imread('er.jpg'));
edges = {edge(I,'sobel'), edge(I,'canny'), edge(I,'prewitt'), edge(I,'log')};
titles = {'Sobel', 'Canny', 'Prewitt', 'Laplacian of Gauss'};

figure;
subplot(1,5,1), imshow(I), title('Original Image');
for k = 1:4
   subplot(1,5,k+1), imshow(edges{k}), title(titles{k});
end
```

5.2 Without using Built-in Functions:

```
A = imread('er.jpg');

C = double(rgb2gray(A));

% Sobel kernels

Gx,Gy = [-1 0 1; -2 0 2; -1 0 1]; % Gy = Gx';

% Compute gradients using convolution

Ix = conv2(C, Gx, 'valid');

Iy = conv2(C, Gy, 'valid');

% Gradient magnitude

%B = sqrt(Ix.^2 + Iy.^2);

% Thresholding

Thresh = 50;

B = uint8(B);

B(B < Thresh) = 0;

figure;subplot(1,2,1), imshow(B, []); title('Sobel Gradient');

subplot(1,2,2), imshow(~B); title('Edge Detected Image');
```

5.3 Simple Boundary Extraction:

```
A = imread('er.jpg');

F = imerode(A, strel('disk', 2));

figure;
```

```
subplot(2,1,1), imshow(A), title('Original Image');
subplot(2,1,2), imshow(A - F), title('Boundary Extracted Image');
```

6. IMPLEMENT IMAGE COMPRESSION

6.1.Block Truncation Coding

```
clc; clear; close all;
I = double(imread('er.jpg'));
[n,m,c] = size(I); k = 4;
out = zeros(size(I));
for b = 1:c
   for i = 1:k:n
      for j = 1:k:m
         R = i:min(i+k-1,n);
         C = j:min(j+k-1,m);
         blk = I(R,C,b);
         mn = mean(blk(:));
         out(R,C,b) = (blk > mn) * round(mean(blk(blk > mn)+0.01));
      end
   end
end
subplot(1,3,1), imshow(uint8(I)), title('Original');
subplot(1,3,2), imshow(uint8(out)), title('Encoded');
subplot(1,3,3), imshow(uint8(out)), title('Decoded');

imwrite(uint8(out),'encoded.png');
cr = dir('encoded.png').bytes / dir('er.jpg').bytes;
fprintf('Compression Ratio: %.2f\n', cr);
```

6.2 Bitplane Coding:

```
clear all; clc;
```

```
it= imread('er.jpg'); %read the image
itemp = it(:,:,1);
[r,c]= size(itemp); % get the dimensions of image
s = zeros(r,c,8); % pre allocate a variable to store 8 bit planes of the image
for k = 1:8
        for i = 1:r
                for j = 1:c
                        s(i,j,k) = bitget(itemp(i,j),k); %get kth bit from each pixel
                end
        end
end
figure,imshow(uint8(itemp));title('Original Image');
%display original image figure; %display all the 8 bit planes
subplot(3,3,1);imshow(s(:,:,8));title('8th(MSB)Plane');
subplot(3,3,2);imshow(s(:,:,7));title('7th Plane');
subplot(3,3,3);imshow(s(:,:,6));title('6th Plane');
subplot(3,3,4);imshow(s(:,:,5));title('5th Plane');
subplot(3,3,5);imshow(s(:,:,4));title('4th Plane');
subplot(3,3,6);imshow(s(:,:,3));title('3th Plane');
subplot(3,3,7);imshow(s(:,:,2));title('2nd Plane');
subplot(3,3,8);imshow(s(:,:,1));title('1st(LSB)Plane')
% reconstruct the original image from generated bit planes
rec=s(:,:,1)+s(:,:,2)*2+s(:,:,3)*4+s(:,:,4)*8+s(:,:,5)*16+s(:,:,6)*32+s(:,:,7)*64+s(:,:,8)*128;
subplot(3,3,9);imshow(uint8(rec));title('Recovered Image')%display the reconstructed image
```

7. IMAGE SUBTRACTION

```
% Read and preprocess images
img1_resized = rgb2gray(imresize(imread('er.jpg'), [256, 256]));
img2_resized = rgb2gray(imresize(imread('img.jpg'), [256, 256]));
% Subtract the images
img_diff = imabsdiff(img1_resized, img2_resized);
```

% Display the images and their difference

```
subplot(1, 3, 1), imshow(img1_resized), title('Image 1');
subplot(1, 3, 2), imshow(img2_resized), title('Image 2');
subplot(1, 3, 3), imshow(img_diff), title('Difference Image');
```

8. BOUNDARY EXTRACTION USING MORPHOLOGY

% Read and binarize the grayscale image

```
img = imbinarize(rgb2gray(imread('er.jpg')));
```

% Structuring element

```
se = strel('disk', 1);
```

% Extract boundary by subtracting eroded image from dilated image

```
boundary_img = imdilate(img, se) - imerode(img, se);
```

% Display results

```
figure,imshow(img), title('Original Binary Image');
imshow(boundary_img), title('Extracted Boundaries');
```

9. IMAGE SEGMENTATION

```
img = imread('er.jpg');
gray = rgb2gray(img);

subplot(2,3,1), imshow(img), title('Original Image');
subplot(2,3,2), imshow(imbinarize(gray, graythresh(gray))), title('Thresholding');
subplot(2,3,3), imshow(edge(gray, 'canny')), title('Edge Detection');

[~, idx] = imsegkmeans(gray, 3);
subplot(2,3,4), imshow(label2rgb(idx)), title('K-means Clustering');

grad = imgradient(gray);
ws = watershed(grad);
ws(grad == 0) = 0;
subplot(2,3,5), imshow(label2rgb(ws, 'jet', 'k', 'shuffle')), title('Watershed Segmentation');
```