Ex1:

# IMAGE SAMPLING AND QUANTIZATION

%% Task 1: Image Sampling (Resizing, Zooming, Cropping)

% Load the image

img = imread('input\_image.jpg');

% Resizing

img\_resized = imresize(img, 0.5); % Resize to 50%

figure; imshow(img\_resized); title('Resized Image');

% Zooming (using interpolation)

img\_zoomed = imresize(img, 2, 'bilinear');

figure; imshow(img\_zoomed); title('Zoomed Image');

% Cropping

img\_cropped = img(50:200, 50:200, :);

figure; imshow(img\_cropped); title('Cropped Image');

%% Task 2: Image Quantization with Dithering

% Load the image and convert to grayscale

gray\_img = rgb2gray(img);

% Without dithering

img\_quant\_no\_dither = uint8(floor(double(gray\_img) / 32) \* 32);

figure; imshow(img\_quant\_no\_dither); title('Quantized (No Dithering)');

% With dithering

img\_dithered = dither(gray\_img);

figure; imshow(img\_dithered); title('Quantized (With Dithering)');

EX2: **ANALYSIS OF SPATIAL AND INTENSITY RESOLUTION OF IMAGES**

%% Analysis of Intensity Resolution (Changing Gray Levels)

% Load the image

img = imread('/MATLAB Drive/penguin.jpeg');

gray\_img = rgb2gray(img);

% Effect of Changing Gray Levels

figure;

for i = 1:4

levels = 2^(2 \* i); % Gray levels: 4, 16, 64, 256

img\_intensity = uint8(floor(double(gray\_img) / (256 / levels)) \* (256 / levels));

subplot(2, 2, i); imshow(img\_intensity);

title(['Gray Levels: ', num2str(levels)]);

end

%% Analysis of Spatial Resolution

% Effect of Changing Spatial Resolution

figure;

for i = 1:4

scale = 0.1 \* i + 0.5; % Scaling factors: 0.6, 0.7, 0.8, 0.9

img\_resized = imresize(gray\_img, scale, 'bilinear');

img\_restored = imresize(img\_resized, size(gray\_img), 'bilinear');

subplot(2, 2, i); imshow(img\_restored);

title(['Spatial Scale: ', num2str(scale)]);

end

EX3: **INTENSITY TRANSFORMATION OF IMAGES**

a)

%% Photographic Negative and Logarithmic Transformation

% Load the image

img = imread('/MATLAB Drive/penguin.jpeg');

gray\_img = rgb2gray(img);

% Photographic Negative

img\_negative = 255 - gray\_img;

figure;

subplot(1, 2, 1); imshow(gray\_img); title('Original Image');

subplot(1, 2, 2); imshow(img\_negative); title('Photographic Negative');

% Logarithmic Transformation with c = 1, 2, 5

figure;

c\_values = [1, 2, 5];

for i = 1:3

img\_log = c\_values(i) \* log(1 + double(gray\_img));

img\_log = uint8(255 \* mat2gray(img\_log)); % Normalize to 8-bit range

subplot(1, 3, i); imshow(img\_log); title(['Log Transform (c = ', num2str(c\_values(i)), ')']);

end

b) %% Gamma Transformation

% Load the image

img = imread('/MATLAB Drive/penguin.jpeg');

gray\_img = rgb2gray(img);

% Gamma Transformation with gamma = 1, 3, 0.4

figure;

gamma\_values = [1, 3, 0.4];

for i = 1:3

img\_gamma = 255 \* ((double(gray\_img) / 255) .^ gamma\_values(i));

img\_gamma = uint8(img\_gamma); % Convert back to 8-bit

subplot(1, 3, i); imshow(img\_gamma); title(['Gamma = ', num2str(gamma\_values(i))]);

end

c) %% Contrast Stretching Intensity Transformation

% Load the image

img = imread('/MATLAB Drive/penguin.jpeg');

gray\_img = rgb2gray(img);

% Contrast stretching transformation with E values = 0.4, 0.5, -1

figure;

E\_values = [0.4, 0.5, -1];

for i = 1:3

img\_contrast = 1 ./ (1 + (128 ./ double(gray\_img)).^E\_values(i));

img\_contrast = uint8(255 \* mat2gray(img\_contrast)); % Normalize to 8-bit range

subplot(1, 3, i); imshow(img\_contrast); title(['E = ', num2str(E\_values(i))]);

end

d) %% Intensity Level Slicing

% Load the image

img = imread('/MATLAB Drive/penguin.jpeg');

gray\_img = rgb2gray(img);

% Intensity level slicing (min = 100, max = 180)

img\_sliced = zeros(size(gray\_img));

img\_sliced(gray\_img >= 100 & gray\_img <= 180) = 255;

img\_sliced = uint8(img\_sliced);

% Display results

figure;

subplot(1, 2, 1); imshow(gray\_img); title('Original Image');

subplot(1, 2, 2); imshow(img\_sliced); title('Intensity Level Slicing (100-180)');

# EX4: DFT ANALYSIS OF IMAGES

% Load the image

img = imread('input\_image.jpg');

img = rgb2gray(img); % Convert to grayscale if needed

% Display original image

figure; imshow(img); title('Original Image');

% 2D DFT using FFT

img\_fft = fft2(double(img));

img\_fft\_shift = fftshift(img\_fft);

% Display magnitude spectrum

figure; imshow(log(1 + abs(img\_fft\_shift)), []); title('Magnitude Spectrum');

% Image restoration using inverse FFT

img\_restored = uint8(ifft2(ifftshift(img\_fft\_shift)));

% Display restored image

figure; imshow(img\_restored); title('Restored Image');

EX5: **TRANSFORMS (WALSH, HADAMARD, DCT, HAAR)**

clc; clear all; close all;

d = imread('/MATLAB Drive/flowers.jpeg');

e = im2double(rgb2gray(d)); % Convert to grayscale for DCT

% Resize image

e = imresize(e, [256 256]);

[M, N] = size(e);

% DCT Kernel

[u, x] = meshgrid(0:N-1);

ker1 = sqrt(2/N) \* cos(pi \* (2.\*x + 1) .\* u / (2 \* N));

ker1(1, :) = ker1(1, :) / sqrt(2);

% DCT Transformation

trans2 = ker1 \* e \* ker1'; % Corrected multiplication with transpose

% Display Results

subplot(2, 2, 1); imshow(d); xlabel('Fig 5.4 Original Image');

subplot(2, 2, 2); imshow(log(1 + abs(trans2)), []); xlabel('Fig 5.5 DCT Image');

% DCT Reconstruction

orig2 = ker1' \* trans2 \* ker1; % Corrected multiplication with transpose

subplot(2, 2, 3); imshow(orig2, []); xlabel('Fig 5.6 DCT Reconstructed Image');

% Error Calculation

error2 = abs(e - orig2); % Corrected error calculation

subplot(2, 2, 4); imshow(error2, []); xlabel('Fig 5.8 DCT Error Image');

EX6: **HISTOGRAM PROCESSING AND BASICTHRESHOLDING FUNCTIONS**

%% Task 1: Histogram Processing and Equalization

% Load the image

img = imread('/MATLAB Drive/flowers.jpeg');

gray\_img = rgb2gray(img);

% a) Plot Histogram for Original Image

figure;

subplot(2, 2, 1); imshow(gray\_img); title('Original Image');

subplot(2, 2, 2); imhist(gray\_img); title('Histogram of Original Image');

% b) Histogram Equalization

equalized\_img = histeq(gray\_img);

subplot(2, 2, 3); imshow(equalized\_img); title('Equalized Image');

subplot(2, 2, 4); imhist(equalized\_img); title('Histogram of Equalized Image');

%% Task 2: Basic Thresholding

% Define Threshold Value

threshold\_value = 100;

thresholded\_img = gray\_img > threshold\_value;

% Display Thresholded Image

figure;

subplot(1, 2, 1); imshow(gray\_img); title('Original Grayscale Image');

subplot(1, 2, 2); imshow(thresholded\_img); title(['Thresholded Image (Threshold = ', num2str(threshold\_value), ')']);

EX7: **IMAGE ENHANCEMENT-SPATIAL FILTERING**

%% Image Enhancement in Spatial Domain

% Load the image

img = imread('/MATLAB Drive/flowers.jpeg');

gray\_img = rgb2gray(img);

% 1. Contrast Adjustment

contrast\_img = imadjust(gray\_img);

% 2. Sharpening using Unsharp Masking

sharpened\_img = imsharpen(gray\_img);

% 3. Smoothing using Average Filter

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

smooth\_img = imfilter(gray\_img, h);

% 4. Edge Enhancement using Laplacian Filter

laplacian\_filter = fspecial('laplacian', 0.2);

edge\_img = imfilter(gray\_img, laplacian\_filter);

% Display Results

figure;

subplot(2, 3, 1); imshow(gray\_img); title('Original Image');

subplot(2, 3, 2); imshow(contrast\_img); title('Contrast Adjusted Image');

subplot(2, 3, 3); imshow(sharpened\_img); title('Sharpened Image');

subplot(2, 3, 4); imshow(smooth\_img); title('Smoothed Image');

subplot(2, 3, 5); imshow(edge\_img); title('Edge Enhanced Image');

# EX8: IMAGE ENHANCEMENT - FILTERING IN FREQUENCY DOMAIN

%% Image Filtering in Frequency Domain

% Load the image

img = imread('/MATLAB Drive/flowers.jpeg');

gray\_img = rgb2gray(img);

% Perform FFT

img\_fft = fft2(double(gray\_img));

img\_fft\_shift = fftshift(img\_fft);

% Create a Low-Pass Filter (Ideal Filter)

[M, N] = size(gray\_img);

[u, v] = meshgrid(-N/2:N/2-1, -M/2:M/2-1);

D = sqrt(u.^2 + v.^2);

D0 = 30; % Cut-off frequency for low-pass filter

low\_pass\_filter = double(D <= D0);

% Apply Filter in Frequency Domain

filtered\_fft = img\_fft\_shift .\* low\_pass\_filter;

filtered\_img = ifft2(ifftshift(filtered\_fft));

filtered\_img = uint8(abs(filtered\_img));

% Display Results

figure;

subplot(2, 2, 1); imshow(gray\_img); title('Original Image');

subplot(2, 2, 2); imshow(log(1 + abs(img\_fft\_shift)), []); title('FFT Spectrum');

subplot(2, 2, 3); imshow(low\_pass\_filter); title('Low-Pass Filter Mask');

subplot(2, 2, 4); imshow(filtered\_img); title('Filtered Image (Low-Pass)');

EX9:

**IMAGE SEGMENTATION- EDGE DETECTION, LINE DETECTION AND POINT DETECTION**

%% Image Segmentation: Edge, Line, and Point Detection

% Load the image

img = imread('/MATLAB Drive/flowers.jpeg');

gray\_img = rgb2gray(img);

% Edge Detection

edge\_img = edge(gray\_img, 'Canny');

% Line Detection using Hough Transform

edges = edge(gray\_img, 'Canny');

[H, T, R] = hough(edges);

peaks = houghpeaks(H, 5);

lines = houghlines(edges, T, R, peaks);

% Create Image with Lines Detected

line\_img = gray\_img;

figure;

imshow(gray\_img); hold on;

for k = 1:length(lines)

xy = [lines(k).point1; lines(k).point2];

plot(xy(:, 1), xy(:, 2), 'LineWidth', 2, 'Color', 'red');

end

hold off;

% Point Detection using Laplacian Filter

laplacian\_filter = fspecial('laplacian', 0.2);

point\_img = imfilter(gray\_img, laplacian\_filter);

% Display Results

figure;

subplot(2, 2, 1); imshow(gray\_img); title('Original Image');

subplot(2, 2, 2); imshow(edge\_img); title('Edge Detection (Canny)');

subplot(2, 2, 3); imshow(line\_img); title('Line Detection (Hough Transform)');

subplot(2, 2, 4); imshow(point\_img, []); title('Point Detection (Laplacian Filter)');

ex10: **BASIC MORPHOLOGICAL OPERATION**

%% Basic Morphological Operations

% Load the binary image

img = imread('/MATLAB Drive/flowers.jpeg');

bw\_img = imbinarize(rgb2gray(img));

% Structuring Element

se = strel('disk', 5);

% Morphological Operations

dilated\_img = imdilate(bw\_img, se);

eroded\_img = imerode(bw\_img, se);

opened\_img = imopen(bw\_img, se);

closed\_img = imclose(bw\_img, se);

% Display Results

figure;

subplot(2, 3, 1); imshow(bw\_img); title('Original Binary Image');

subplot(2, 3, 2); imshow(dilated\_img); title('Dilated Image');

subplot(2, 3, 3); imshow(eroded\_img); title('Eroded Image');

subplot(2, 3, 4); imshow(opened\_img); title('Opened Image');

subplot(2, 3, 5); imshow(closed\_img); title('Closed Image');

# EX11: REGION BASED SEGMENTATION

%% Region-Based Segmentation Using MATLAB

% Load the image

img = imread('/MATLAB Drive/flowers.jpeg');

gray\_img = rgb2gray(img);

% Region Growing

seed\_point = [100, 100]; % Example seed point

region\_growing\_img = regiongrowing(gray\_img, seed\_point(1), seed\_point(2), 0.2);

% Watershed Segmentation

gradient\_img = imgradient(gray\_img);

watershed\_img = watershed(gradient\_img);

watershed\_seg = label2rgb(watershed\_img);

% Display Results

figure;

subplot(1, 3, 1); imshow(gray\_img); title('Original Image');

subplot(1, 3, 2); imshow(region\_growing\_img); title('Region Growing Segmentation');

subplot(1, 3, 3); imshow(watershed\_seg); title('Watershed Segmentation');

% Region Growing Function

function output = regiongrowing(img, x, y, threshold)

[M, N] = size(img);

output = zeros(M, N);

visited = false(M, N);

stack = [x, y];

while ~isempty(stack)

point = stack(end, :);

stack(end, :) = [];

if visited(point(1), point(2))

continue;

end

visited(point(1), point(2)) = true;

output(point(1), point(2)) = 1;

% Check neighbors

for dx = -1:1

for dy = -1:1

nx = point(1) + dx;

ny = point(2) + dy;

if nx > 0 && nx <= M && ny > 0 && ny <= N && ~visited(nx, ny)

if abs(double(img(nx, ny)) - double(img(x, y))) < threshold

stack = [stack; nx, ny];

end

end

end

end

end

end

EX12: **SEGMENTATION USING WATERSHED TRANSFORMATION**

%% Segmentation Using Watershed Transformation

% Load the image

img = imread('/MATLAB Drive/flowers.jpeg');

gray\_img = rgb2gray(img);

% Compute Gradient Magnitude

grad\_mag = imgradient(gray\_img);

% Apply Watershed Transformation

L = watershed(grad\_mag);

watershed\_img = label2rgb(L);

% Display Results

figure;

subplot(1, 3, 1); imshow(gray\_img); title('Original Image');

subplot(1, 3, 2); imshow(grad\_mag, []); title('Gradient Magnitude');

subplot(1, 3, 3); imshow(watershed\_img); title('Watershed Segmentation');

EX13: **ANALYSIS OF IMAGES WITH DIFFERENT COLOR MODELS**

a) %% Image Conversion Program

% Load the RGB image

img = imread('/MATLAB Drive/flowers.jpeg');

% Convert RGB to Grayscale

gray\_img = rgb2gray(img);

% Create a sample matrix and convert it to Grayscale

matrix\_img = uint8(rand(256, 256) \* 255); % Example random matrix

gray\_from\_matrix = mat2gray(matrix\_img);

% Convert Grayscale to Binary

binary\_img = imbinarize(gray\_img);

% Display Results

figure;

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

subplot(2, 2, 2); imshow(gray\_img); title('Grayscale Image');

subplot(2, 2, 3); imshow(gray\_from\_matrix); title('Matrix to Grayscale Image');

subplot(2, 2, 4); imshow(binary\_img); title('Binary Image');

b) %% RGB Image Component Extraction

% Load the RGB image

img = imread('/MATLAB Drive/flowers.jpeg');

% Extract Red, Green, and Blue Components

red\_channel = img(:,:,1);

green\_channel = img(:,:,2);

blue\_channel = img(:,:,3);

% Display Results

figure;

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

subplot(2, 2, 2); imshow(red\_channel); title('Red Channel');

subplot(2, 2, 3); imshow(green\_channel); title('Green Channel');

subplot(2, 2, 4); imshow(blue\_channel); title('Blue Channel');

c) %% RGB to HSI Conversion and Display

% Load the RGB image

img = imread('/MATLAB Drive/flowers.jpeg');

% Convert RGB to HSI

hsi\_img = rgb2hsv(img);

% Extract Hue, Saturation, and Intensity Components

hue = hsi\_img(:,:,1);

saturation = hsi\_img(:,:,2);

intensity = rgb2gray(img); % Intensity as the grayscale equivalent

% Display Results

figure;

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

subplot(2, 2, 2); imshow(hue); title('Hue Component');

subplot(2, 2, 3); imshow(saturation); title('Saturation Component');

subplot(2, 2, 4); imshow(intensity); title('Intensity Component');

ex15: **IMAGE COMPRESSION TECHNIQUES**

%% Image Compression Using DCT and IDCT

% Load the image

img = imread('/MATLAB Drive/flowers.jpeg');

gray\_img = rgb2gray(img);

% Perform DCT (Discrete Cosine Transform)

DCT\_img = dct2(gray\_img);

% Apply Compression by Zeroing Small Coefficients

threshold = 10;

compressed\_img = DCT\_img;

compressed\_img(abs(DCT\_img) < threshold) = 0;

% Perform Inverse DCT to Reconstruct Image

reconstructed\_img = idct2(compressed\_img);

% Display Results

figure;

subplot(2, 2, 1); imshow(gray\_img); title('Original Grayscale Image');

subplot(2, 2, 2); imshow(log(abs(DCT\_img)),[]); title('DCT of Image');

subplot(2, 2, 3); imshow(log(abs(compressed\_img)),[]); title('Compressed DCT Image');

subplot(2, 2, 4); imshow(uint8(reconstructed\_img)); title('Reconstructed Image');

ex16 **IMAGE RESTORATION**

%% Image Restoration Using MATLAB

% Load the original image

img = imread('/MATLAB Drive/flowers.jpeg');

gray\_img = rgb2gray(img);

% Add Noise to Image

noisy\_img = imnoise(gray\_img, 'gaussian', 0, 0.01);

% Apply Wiener Filter for Restoration

restored\_img = wiener2(noisy\_img, [5 5]);

% Display Results

figure;

subplot(1, 3, 1); imshow(gray\_img); title('Original Grayscale Image');

subplot(1, 3, 2); imshow(noisy\_img); title('Noisy Image');

subplot(1, 3, 3); imshow(restored\_img); title('Restored Image');