BMS Institute of Technology & Management
Bengaluru

Department of AI & ML

V Semester
Computer Vision (BAI505B)

A Report on
Continuous Comprehensive Assessment
(Tutorial Programs & Micro – Project)

Academic Year
2025 – 26
(Odd Semester)

Name of the student	Advaith Arun Kashyap
USN	1BY23AI011

Marks Obtained	
Total Marks	

Prof. Sanjay M Belgaonkar
Signature of the Course Faculty

Name of the Course:	**Computer Vision**
Name of the CCA:	**Micro Projects on Computer Vision Concepts**
Maximum marks:	**20**

Brief description about the CCA

The micro projects in the **Computer Vision** course are designed to help students connect fundamental image processing concepts with real-world industrial applications. Each project links a technical topic to a practical case study from leading companies. Students begin by exploring **Linear Filters and Convolution** through noise reduction in smartphone images, learning how convolution smooths images while balancing detail preservation. In **Shift-Invariant Linear Systems**, they apply edge detection to manufacturing images, understanding how filters detect defects regardless of object position—key in automated inspection. The **Spatial Frequency and Fourier Transforms** project introduces frequency domain analysis using MRI and CT images, showing how filtering enhances or removes details for better medical diagnostics. Through **Sampling and Aliasing**, students study resolution, Moiré patterns, and anti-aliasing in video streaming and imaging systems like Netflix or CCTV. The **Filters as Templates** activity applies template matching for detecting logos in advertisements, revealing its role in brand monitoring and its challenges under variations in scale or lighting. Finally, in **Local Image Features and Gradients**, students work with lane detection in autonomous vehicles, using gradient-based methods like Sobel and Canny to identify road boundaries and obstacles. Together, these projects blend theory, coding practice, and industry relevance, equipping students with practical computer vision skills aligned with real-world AI applications.

Rubrics for Evaluation

Criteria	Excellent	Good	Needs Improvement
Implementation of Concepts (Apply) (10)	Correctly implements the CV concepts with appropriate explanation. (10)	Implements the CV concepts with minor errors or suboptimal choices. (6 – 9)	Incorrect or incomplete implementation. (0 – 5)
Innovation & Optimization (Analyze, Evaluate) (5)	Compares results effectively and explains the result analysis. (5)	Moderate analysis of results. (3 – 4)	Very poor analysis of results. (0 – 2)
Code Quality, Correctness & Documentation (5)	Code is well-structured, modular, and well-documented with clear explanations. Produces correct results, is fully reproducible, and runs without errors. (5)	Code is structured with some documentation but lacks clarity in parts. Produces mostly correct results with minor issues. (3 – 4)	Code is messy, hard to follow, and lacks documentation. Code does not run or gives incorrect results. (0 – 2)
Total Marks (20)	**20**	**10 to 17**	**Less than 10**

Index Sheet

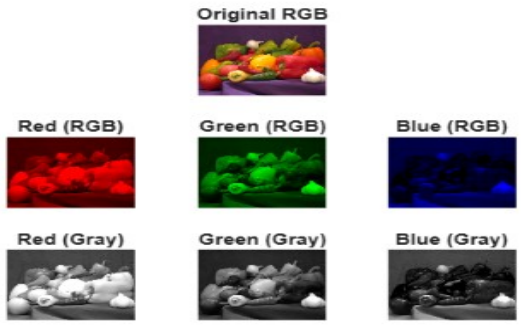
Program 1:

Program to Read an RGB Image, Convert it into Matrix Form, Split into R, G, B Components, and Display Each Channel Separately.

Code:

```
% 1. Read in-built RGB image
img = imread('peppers.png');

% 2. Convert to matrix form
img_matrix = double(img);

% 3. Split into R, G, B components
R = img(:,:,1);
G = img(:,:,2);
B = img(:,:,3);

% 4. Reconstruct RGB visualization for each channel
red_img   = cat(3, R, zeros(size(R)), zeros(size(R)));
green_img = cat(3, zeros(size(G)), G, zeros(size(G)));
blue_img  = cat(3, zeros(size(B)), zeros(size(B)), B);

% ----- Display Section -----
figure('Name','RGB Channel Visualizations','NumberTitle','off');

% Centered original image
subplot(3,3,2);
imshow(img);
title('Original RGB');

% RGB channel images
subplot(3,3,4); imshow(red_img);   title('Red (RGB)');
subplot(3,3,5); imshow(green_img); title('Green (RGB)');
subplot(3,3,6); imshow(blue_img);  title('Blue (RGB)');

% Grayscale channel views
subplot(3,3,7); imshow(R); colormap(gca, gray); title('Red (Gray)');
subplot(3,3,8); imshow(G); colormap(gca, gray); title('Green (Gray)');
subplot(3,3,9); imshow(B); colormap(gca, gray); title('Blue (Gray)');
```

Output:



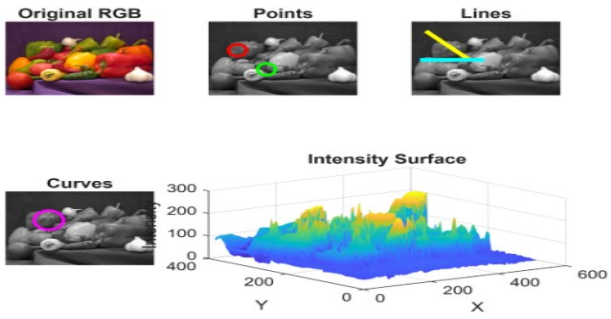
Program 2:

Program to Demonstrate Image Geometry Primitives: Points, Lines, Curves, and Surfaces

Code:

```
% ---- IMAGE GEOMETRY PRIMITIVES DEMO ----

% 1. Use a built-in online compatible image
img = imread('peppers.png');  % reliably available in MATLAB Online
gray_img = rgb2gray(img);     % grayscale for geometry illustration

figure('Name','Image Geometry Primitives','NumberTitle','off');

% Display original image
subplot(2,3,1);
imshow(img);
title('Original RGB');

% 2. Points (mark specific pixel locations)
subplot(2,3,2);
imshow(gray_img); hold on;
plot(100, 150, 'ro', 'MarkerSize', 10, 'LineWidth', 2);
plot(200, 250, 'go', 'MarkerSize', 10, 'LineWidth', 2);
title('Points');

% 3. Lines
subplot(2,3,3);
imshow(gray_img); hold on;
line([50 200], [50 200], 'Color', 'y', 'LineWidth', 3);
line([30 250], [200 200], 'Color', 'c', 'LineWidth', 3);
title('Lines');

% 4. Curves (parametric circle)
subplot(2,3,4);
imshow(gray_img); hold on;
theta = linspace(0, 2*pi, 100);
x = 150 + 50*cos(theta);
y = 150 + 50*sin(theta);
plot(x, y, 'm-', 'LineWidth', 2);
title('Curves');

% 5. 3D Surface — intensity mesh
subplot(2,3,[5 6]);
mesh(double(gray_img));

title('Intensity Surface');
xlabel('X'); ylabel('Y'); zlabel('Intensity');
```

Output:



Program 3

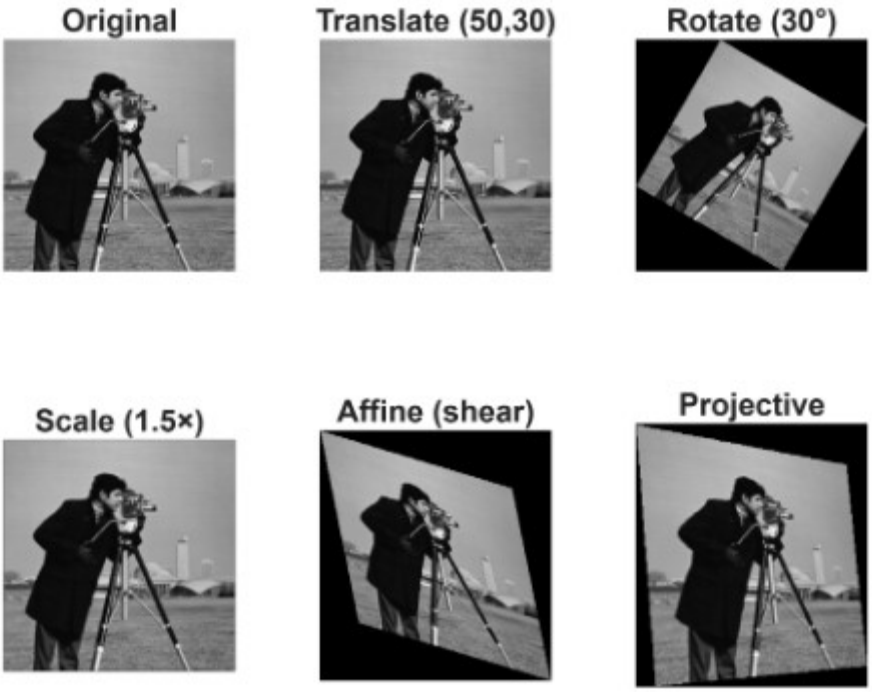
Program to Perform Geometric Transformations on Images: Translation, Rotation, Scaling, Affine, and Projective Transformations

Code:

```
% ---- Geometric Transformations Demo (MATLAB Online ready) ----

% 1. Read a reliable built-in image available in MATLAB Online
img = imread('cameraman.tif');

figure('Name','Geometric Transformations','NumberTitle','off');

subplot(2,3,1);
imshow(img);
title('Original');

% 2. Translation
tform_translate = affine2d([1 0 0; 0 1 0; 50 30 1]);
translated = imwarp(img, tform_translate);
subplot(2,3,2);
imshow(translated);
title('Translate (50,30)');

% 3. Rotation
tform_rotate = affine2d([cosd(30) sind(30) 0; -sind(30) cosd(30) 0; 0 0 1]);
rotated = imwarp(img, tform_rotate);
subplot(2,3,3);
imshow(rotated);
title('Rotate (30°)');

% 4. Scaling
tform_scale = affine2d([1.5 0 0; 0 1.5 0; 0 0 1]);
scaled = imwarp(img, tform_scale);
subplot(2,3,4);
imshow(scaled);
title('Scale (1.5×)');
```

```
% 5. Shear / general affine
tform_affine = affine2d([1 0.3 0; 0.2 1 0; 0 0 1]);
affine_img = imwarp(img, tform_affine);
subplot(2,3,5);
imshow(affine_img);
title('Affine (shear)');

% 6. Projective (perspective) transform
input_points  = [1 1; size(img,2) 1; size(img,2) size(img,1); 1 size(img,1)];
output_points = [1 1; size(img,2)-50 40; size(img,2)-30 size(img,1)-20; 20 size(img,1)];

tform_proj = fitgeotrans(input_points, output_points, 'projective');
projective_img = imwarp(img, tform_proj);

subplot(2,3,6);
imshow(projective_img);
title('Projective');
```

Output:



Program 4

Program to visualize Fourier Spectra: Demonstration using standard inbuilt MATLAB grey scale & colour images

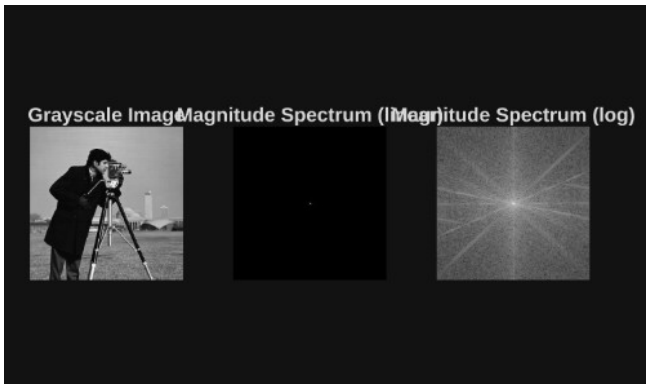
Code:

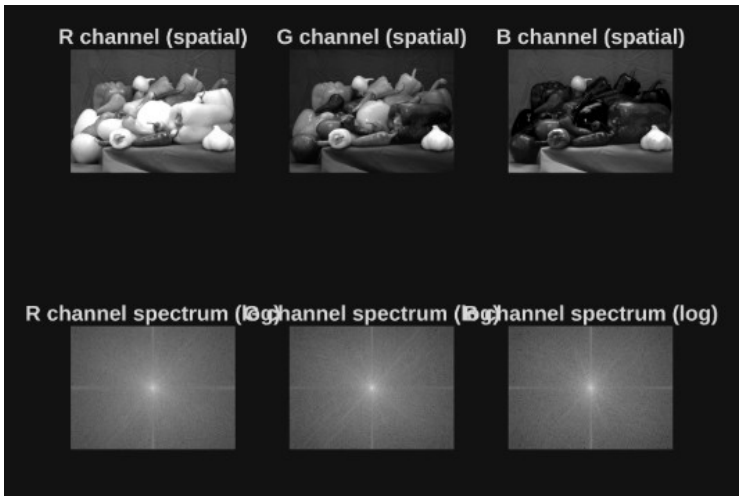
```
close all; clear; clc;

%% 1. Load standard demo images (grayscale and color)
I_gray  = imread('cameraman.tif');
I_color = imread('peppers.png');
```

```
figure;
subplot(1,2,1); imshow(I_gray);  title('Original Grayscale (cameraman)');
subplot(1,2,2); imshow(I_color); title('Original Color (peppers)');

F_gray  = fft2(double(I_gray));
F_gray_c = fftshift(F_gray);
mag_gray = abs(F_gray_c);
log_mag_gray = log(1 + mag_gray);

figure;
subplot(1,3,1); imshow(I_gray); title('Grayscale Image');
subplot(1,3,2); imshow(mag_gray, []); title('Magnitude Spectrum (linear)');
subplot(1,3,3); imshow(log_mag_gray, []); title('Magnitude Spectrum (log)');

R = I_color(:,:,1);
G = I_color(:,:,2);
B = I_color(:,:,3);

F_R = fftshift(fft2(double(R)));
F_G = fftshift(fft2(double(G)));
F_B = fftshift(fft2(double(B)));

log_R = log(1 + abs(F_R));
log_G = log(1 + abs(F_G));
log_B = log(1 + abs(F_B));

figure;
subplot(2,3,1); imshow(R);     title('R channel (spatial)');
subplot(2,3,2); imshow(G);     title('G channel (spatial)');
subplot(2,3,3); imshow(B);     title('B channel (spatial)');
subplot(2,3,4); imshow(log_R, []); title('R channel spectrum (log)');
subplot(2,3,5); imshow(log_G, []); title('G channel spectrum (log)');
subplot(2,3,6); imshow(log_B, []); title('B channel spectrum (log)');

F_gray_unshift = ifftshift(F_gray_c);
I_gray_rec = real(ifft2(F_gray_unshift));

figure;
subplot(1,2,1); imshow(I_gray); title('Original Grayscale');
subplot(1,2,2); imshow(I_gray_rec,[]); title('Reconstructed from Spectrum');
```

Output:


Original Grayscale (cameraman)
Original Color (peppers)


Grayscale Image
Magnitude Spectrum (log)


R channel (spatial)
G channel (spatial)
B channel (spatial)
R channel spectrum (log)
G channel spectrum (log)
B channel spectrum (log)


Original Grayscale
Reconstructed from Spectrum

Program 5

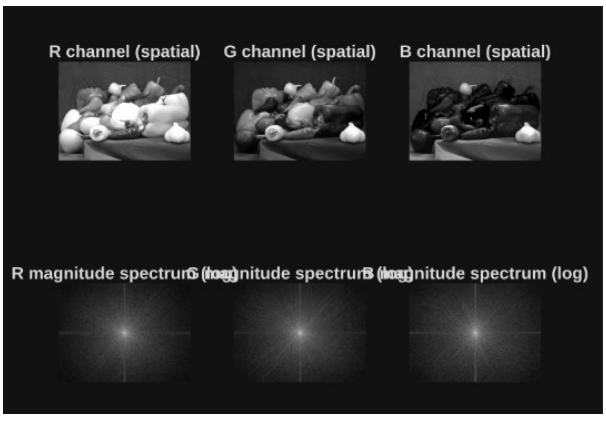
Program to find Fourier Spectrum of Colour Images with Low- pass & High-pass Filtering

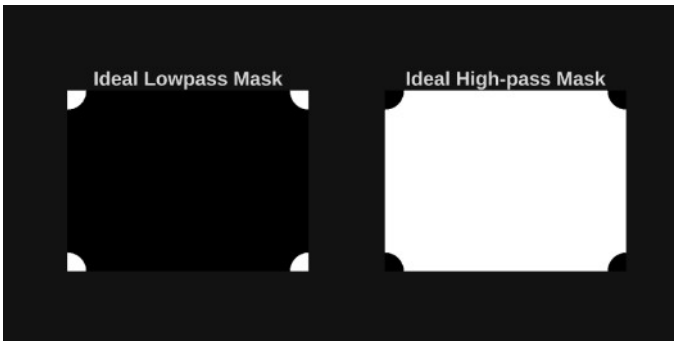
Code:

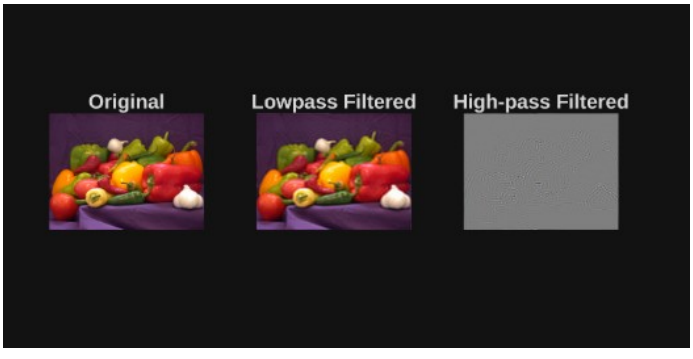
```
%% Program: Fourier Spectrum of Color Image with Lowpass & High-pass Filtering

close all; clear; clc;

%% 1. Read color image
I_color = imread('peppers.png');
I_color = im2double(I_color);

R = I_color(:,:,1);
G = I_color(:,:,2);
B = I_color(:,:,3);

figure;
imshow(I_color);
title('Original Color Image (peppers)');


%% 2. 2-D FFT and magnitude spectrum
FR = fftshift(fft2(R));
FG = fftshift(fft2(G));
FB = fftshift(fft2(B));

magR = log(1 + abs(FR));
magG = log(1 + abs(FG));
magB = log(1 + abs(FB));

figure;
subplot(2,3,1); imshow(R, []);   title('R channel (spatial)');
subplot(2,3,2); imshow(G, []);   title('G channel (spatial)');
subplot(2,3,3); imshow(B, []);   title('B channel (spatial)');
subplot(2,3,4); imshow(magR,[]); title('R magnitude spectrum (log)');
subplot(2,3,5); imshow(magG,[]); title('G magnitude spectrum (log)');
subplot(2,3,6); imshow(magB,[]); title('B magnitude spectrum (log)');



%% 3. Ideal lowpass & high-pass filters
[M, N] = size(R);

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

D  = sqrt(U.^2 + V.^2);
D0 = 40;

H_low  = double(D <= D0);
H_high = 1 - H_low;
```

```
figure;
subplot(1,2,1); imshow(fftshift(H_low),[]);  title('Ideal Lowpass Mask');
subplot(1,2,2); imshow(fftshift(H_high),[]); title('Ideal High-pass Mask');


%% 4. Lowpass filtering
FR_low = FR .* H_low;
FG_low = FG .* H_low;
FB_low = FB .* H_low;

R_low = real(ifft2(ifftshift(FR_low)));
G_low = real(ifft2(ifftshift(FG_low)));
B_low = real(ifft2(ifftshift(FB_low)));

I_low = cat(3, R_low, G_low, B_low);

%% 5. High-pass filtering
FR_high = FR .* H_high;
FG_high = FG .* H_high;
FB_high = FB .* H_high;

R_high = real(ifft2(ifftshift(FR_high)));
G_high = real(ifft2(ifftshift(FG_high)));
B_high = real(ifft2(ifftshift(FB_high)));

I_high = cat(3, R_high, G_high, B_high);

%% 6. Display results
figure;
subplot(1,3,1); imshow(I_color);      title('Original');
subplot(1,3,2); imshow(I_low,  []);   title('Lowpass Filtered');
subplot(1,3,3); imshow(I_high+0.5, []); title('High-pass Filtered');
```

Output:







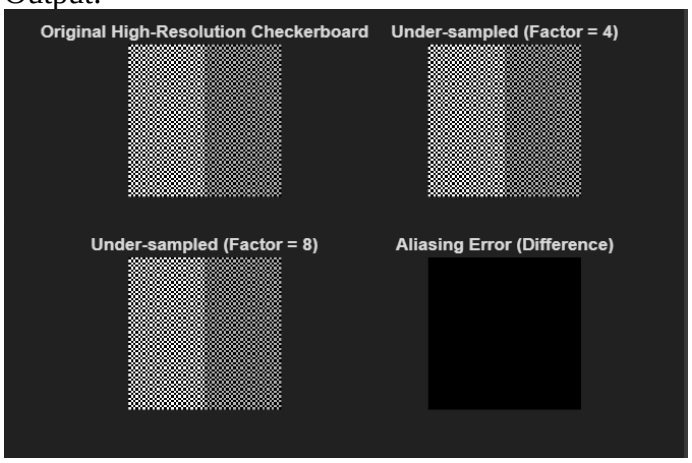
Program 6

Program to demonstrate 2D Aliasing Demo: Under sampling a Checkerboard to show how under sampling creates jaggies/Moiré patterns

Code:

```
clc;
clear;
close all;

%% Generate High-Frequency Checkerboard
N = 512;              % Image size
blockSize = 8;        % Checker block size
checker_img = checkerboard(blockSize, N/(2*blockSize));

checker_img = checker_img(1:N,1:N);

%% Under-sampling
sampling_factor1 = 4;
sampling_factor2 = 8;

undersample1 = checker_img(1:sampling_factor1:end, ...
                    1:sampling_factor1:end);

undersample2 = checker_img(1:sampling_factor2:end, ...
                    1:sampling_factor2:end);

%% Resize back to original size for visualization
upsample1 = imresize(undersample1, [N N], 'nearest');
upsample2 = imresize(undersample2, [N N], 'nearest');

%% Display Results
figure;

subplot(2,2,1);
imshow(checker_img);
title('Original High-Resolution Checkerboard');

subplot(2,2,2);
imshow(upsample1);
title('Under-sampled (Factor = 4)');

subplot(2,2,3);
imshow(upsample2);
title('Under-sampled (Factor = 8)');

subplot(2,2,4);
imshow(abs(upsample2 - checker_img), []);
title('Aliasing Error (Difference)');
```

Output:



Program 7

Program to demonstrate 8-point algorithm

Code:

```
clc;
clear;
close all;

%% Generate synthetic corresponding points
% Points in Image 1
pts1 = [ ...
   10 20;
   30 40;
   50 60;
   70 80;
   15 75;
   85 25;
   90 60;
   40 10
];

% Apply small transformation to get Image 2 points
pts2 = pts1 + randn(size(pts1)) * 2;   % add noise

%% Convert to homogeneous coordinates
x1 = [pts1 ones(8,1)];
x2 = [pts2 ones(8,1)];

%% Construct matrix A
A = zeros(8,9);
for i = 1:8
   A(i,:) = [
      x2(i,1)*x1(i,1), x2(i,1)*x1(i,2), x2(i,1), ...
      x2(i,2)*x1(i,1), x2(i,2)*x1(i,2), x2(i,2), ...
      x1(i,1),         x1(i,2),         1
   ];
end
```

```
%% Solve Af = 0 using SVD
[~,~,V] = svd(A);
F = reshape(V(:,9), [3 3])';

%% Enforce rank-2 constraint
[U,S,V] = svd(F);
S(3,3) = 0;
F = U*S*V';

%% Display Fundamental Matrix
disp('Estimated Fundamental Matrix (F):');
disp(F);
```

Output:

```
Estimated Fundamental Matrix (F):
   -0.0000    0.0011   -0.0574
   -0.0013    0.0010    0.3708
    0.0885   -0.4623    0.7985
```

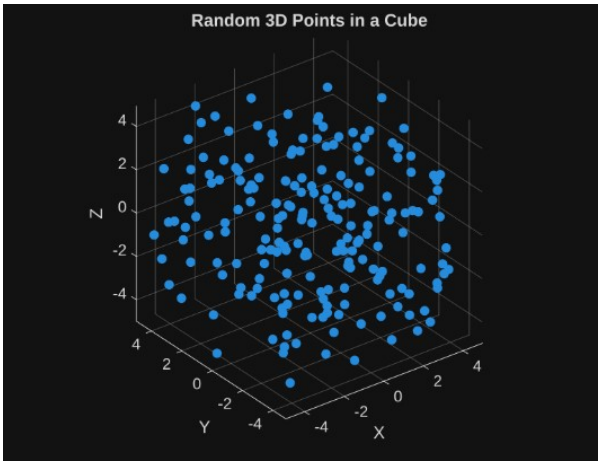
Program 8

Program to generate random 3D points in MATLAB

Code:

```
%% Program: Generate Random 3D Points

close all; clear; clc;

%% 1. Parameters
numPoints = 200;         % number of 3D points
rangeMin  = -5;          % minimum coordinate value
rangeMax  =  5;          % maximum coordinate value

%% 2. Generate random 3D points in a cube [rangeMin, rangeMax]^3
X = rangeMin + (rangeMax - rangeMin) * rand(numPoints, 1);
Y = rangeMin + (rangeMax - rangeMin) * rand(numPoints, 1);
Z = rangeMin + (rangeMax - rangeMin) * rand(numPoints, 1);

points3D = [X Y Z];       % N x 3 matrix of points

%% 3. Visualize with a 3D scatter plot
figure;
scatter3(X, Y, Z, 36, 'filled');   % 36 = marker size
grid on; axis equal;
xlabel('X'); ylabel('Y'); zlabel('Z');
title('Random 3D Points in a Cube');
```

Output:



Program 9

Program to demonstrate K – Means Image Segmentation

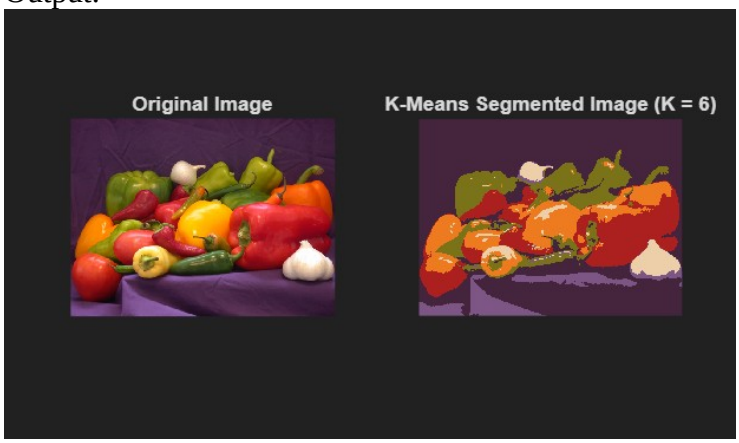
Code:

```
clc; clear; close all;

% ------------------------------------------------------------
% Step 1: Load an inbuilt MATLAB image
% ------------------------------------------------------------
img = imread('peppers.png');       % Example built-in image
img = im2double(img);              % Convert to double for processing

% Get the size
[rows, cols, ch] = size(img);

% Reshape image into N x 3 matrix (each pixel is a 3D point: R,G,B)
pixels = reshape(img, rows*cols, 3);

% ------------------------------------------------------------
% Step 2: Choose number of clusters K
% ------------------------------------------------------------
K = 6;   % You can change this to 4, 5, etc.

% ------------------------------------------------------------
% Step 3: Initialize cluster centers randomly
% ------------------------------------------------------------
rand_idx = randperm(size(pixels,1), K);
centers = pixels(rand_idx, :);      % K initial centers (RGB values)

% To store cluster assignments
cluster_idx = zeros(size(pixels,1), 1);

% ------------------------------------------------------------
% Step 4: K-Means Iteration Loop
% ------------------------------------------------------------
max_iters = 20;
```

```
for iter = 1:max_iters
   fprintf('Iteration %d\n', iter);

   % ----------------------------------------
   % Step 4.1: Assign each pixel to nearest center
   % ----------------------------------------
   for i = 1:size(pixels,1)
      % Compute distance to each center (Euclidean)
      dists = sum((centers - pixels(i,:)).^2, 2);

      % Assign pixel to closest cluster
      [~, cluster_idx(i)] = min(dists);
   end

   % ----------------------------------------
   % Step 4.2: Update cluster centers
   % ----------------------------------------
   new_centers = zeros(K, 3);
   for k = 1:K
      % Get all pixels belonging to cluster k
      cluster_pixels = pixels(cluster_idx == k, :);

      % Avoid empty clusters
      if ~isempty(cluster_pixels)
         new_centers(k, :) = mean(cluster_pixels, 1);
      else
         new_centers(k, :) = centers(k, :);
      end
   end

   % Check for convergence (centers not changing)
   if max(abs(new_centers - centers), [], 'all') < 1e-5
      disp('Converged.');
      break;
   end

   centers = new_centers; % Update centers
end

% ------------------------------------------------------------
% Step 5: Reconstruct segmented image using final cluster centers
% ------------------------------------------------------------
segmented_pixels = zeros(size(pixels));

for k = 1:K
   segmented_pixels(cluster_idx == k, :) = repmat(centers(k,:), sum(cluster_idx == k), 1);
end

% Reshape back to image format
segmented_img = reshape(segmented_pixels, rows, cols, 3);

% ------------------------------------------------------------
% Step 6: Display Results
```

```
% ------------------------------------------------------------
figure;
subplot(1,2,1);
imshow(img);
title('Original Image');

subplot(1,2,2);
imshow(segmented_img);
title(['K-Means Segmented Image (K = ' num2str(K) ')']);
```

Output:



Program 10

Program to demonstrate Mean Shift Clustering & Segmentation

Code:
Mean Shift Clustering:

```
clc; clear; close all;

% ------------------------------------------------------------
% Step 1: Load inbuilt MATLAB color image
% ------------------------------------------------------------
img = imread('peppers.png');
img = im2double(img);
[rows, cols, ch] = size(img);

% Reshape image into Nx3 array (pixels)
pixels = reshape(img, [], 3);

% ------------------------------------------------------------
% Step 2: Set Mean Shift parameters
% ------------------------------------------------------------
hs = 0.1;    % Color bandwidth (smaller = more clusters)
max_iters = 15;
threshold = 1e-3;

N = size(pixels,1);

% ------------------------------------------------------------
% Step 3: Mean Shift Mode-Seeking
```

```
% ------------------------------------------------------------
modes = pixels;     % Start each pixel at its own initial location

fprintf('Running Mean Shift Clustering...\n');

for iter = 1:max_iters
   fprintf("Iteration %d/%d\n", iter, max_iters);

   for i = 1:N
      % Compute distance from pixel i to all pixels
      dist = sqrt(sum((pixels - modes(i,:)).^2, 2));

      % Find pixels within bandwidth (hs)
      neighbors = pixels(dist < hs, :);

      % Compute mean of neighbors
      if ~isempty(neighbors)
         new_point = mean(neighbors, 1);
      else
         new_point = modes(i,:);
      end

      % Update shift
      shift = norm(new_point - modes(i,:));
      modes(i,:) = new_point;

      % Check convergence
      if shift < threshold
         continue;
      end
   end
end

% ------------------------------------------------------------
% Step 4: Group points that converge to the same mode
% ------------------------------------------------------------
fprintf("Assigning clusters based on converged modes...\n");

cluster_labels = zeros(N,1);
cluster_count = 0;
modes_final = [];

for i = 1:N
   assigned = false;
   for j = 1:cluster_count
      if norm(modes(i,:) - modes_final(j,:)) < 0.02  % threshold for merging
         cluster_labels(i) = j;
         assigned = true;
         break;
      end
   end

   if ~assigned
      cluster_count = cluster_count + 1;
```

```
        modes_final(cluster_count,:) = modes(i,:);
        cluster_labels(i) = cluster_count;
    end
end

fprintf("Total clusters found: %d\n", cluster_count);

% ---------------------------------------------------------
% Step 5: Reconstruct segmented image
% ---------------------------------------------------------
seg_img = zeros(size(pixels));

for k = 1:cluster_count
    seg_img(cluster_labels == k, :) = modes_final(k,:);
end

seg_img = reshape(seg_img, rows, cols, 3);

% ---------------------------------------------------------
% Step 6: Display Results
% ---------------------------------------------------------
figure;
subplot(1,2,1); imshow(img); title('Original Image');
subplot(1,2,2); imshow(seg_img); title('Mean Shift Segmented Image');
```

Output:
Mean Shift Clustering:

Code:
Mean Shift Segmentation:

```
clc; clear; close all;

% ---------------------------------------------------------
% 1. Load MATLAB inbuilt image
% ---------------------------------------------------------
img = imread('peppers.png');
img = im2double(img);

[rows, cols, ch] = size(img);

% Create coordinate grid
[x, y] = meshgrid(1:cols, 1:rows);
```

```
% ---------------------------------------------------------
% 2. Create 5D feature vector: [x y R G B]
% ---------------------------------------------------------
features = [x(:), y(:), reshape(img, [], 3)];

% Number of pixels
N = size(features, 1);

% ---------------------------------------------------------
% 3. Mean Shift Parameters
% ---------------------------------------------------------
hs = 10;      % spatial bandwidth (in pixels)
hr = 0.1;     % color bandwidth (RGB distance)
max_iters = 15;
epsilon = 1e-3;   % convergence threshold

% Start modes at original feature positions
modes = features;

fprintf("Running 5D Mean Shift Segmentation...\n");

% ---------------------------------------------------------
% 4. Mean Shift Iterations
% ---------------------------------------------------------
for iter = 1:max_iters
   fprintf("Iteration %d/%d\n", iter, max_iters);

   for i = 1:N

      % Extract current mode (x, y, R, G, B)
      cur_point = modes(i, :);

      % Compute spatial distance
      spatial_dist = sqrt( (features(:,1) - cur_point(1)).^2 + ...
                           (features(:,2) - cur_point(2)).^2 );

      % Compute color distance
      color_dist = sqrt(sum((features(:,3:5) - cur_point(3:5)).^2, 2));

      % Find neighbors within spatial AND color bandwidths
      mask = (spatial_dist < hs) & (color_dist < hr);

      neighbors = features(mask, :);

      % Avoid empty neighborhood
      if ~isempty(neighbors)
         new_point = mean(neighbors, 1);
      else
         new_point = cur_point;
      end

      % Shift magnitude
      shift = norm(new_point - cur_point);
```

```
        % Update mode
        modes(i,:) = new_point;

        % Check convergence
        if shift < epsilon
            continue;
        end
    end
end

% ------------------------------------------------------------
% 5. Cluster pixels whose modes converge close together
% ------------------------------------------------------------
fprintf("Clustering modes...\n");

cluster_labels = zeros(N,1);
cluster_count = 0;
modes_final = [];

for i = 1:N
    assigned = false;

    for j = 1:cluster_count
        % If the mode is close to an existing cluster mode
        if norm(modes(i,:) - modes_final(j,:)) < 2
            cluster_labels(i) = j;
            assigned = true;
            break;
        end
    end

    if ~assigned
        cluster_count = cluster_count + 1;
        modes_final(cluster_count,:) = modes(i,:);
        cluster_labels(i) = cluster_count;
    end
end

fprintf("Total clusters found: %d\n", cluster_count);

% ------------------------------------------------------------
% 6. Construct segmented image from cluster modes
% ------------------------------------------------------------
seg_img = zeros(N, 3);

for k = 1:cluster_count
    seg_img(cluster_labels == k, :) = modes_final(k, 3:5); % RGB only
end

seg_img = reshape(seg_img, rows, cols, 3);

% ------------------------------------------------------------
% 7. Display Original and Segmented Images
```

```
% ------------------------------------------------------------
figure;
subplot(1,2,1);
imshow(img);
title('Original Image');

subplot(1,2,2);
imshow(seg_img);
title('Mean Shift Segmented Image (Color + Spatial)');
```

Output:
Mean Shift Segmentation:

Program 11

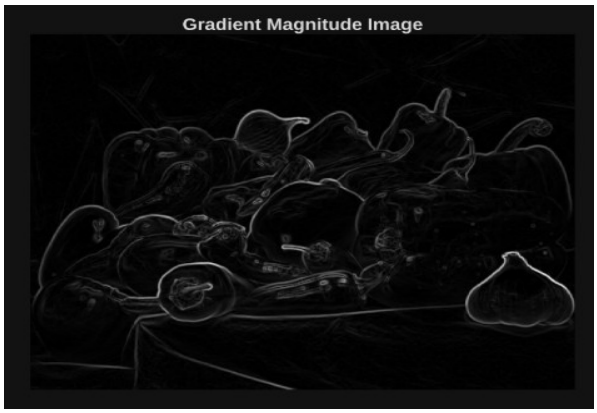
Program to demonstrate the application of Watershed algorithm

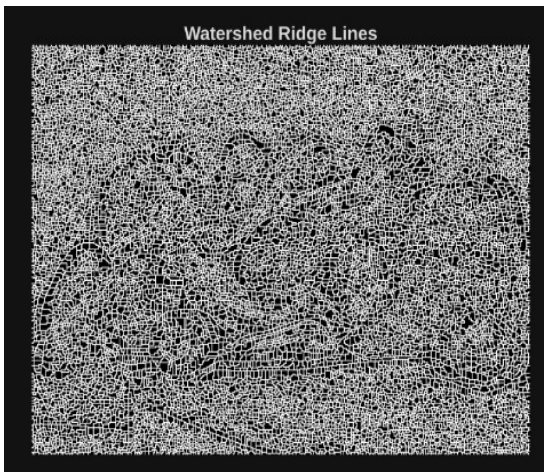
Code:

```
clc; clear; close all;

% ------------------------------------------------------------
% Step 1: Load an inbuilt MATLAB image (choose one)
% ------------------------------------------------------------
img = imread('peppers.png');     % You can change to 'onion.png', 'saturn.png', etc.
figure; imshow(img); title('Original Image');

% Convert to grayscale (watershed works on single channel)
gray = rgb2gray(img);

% ------------------------------------------------------------
% Step 2: Compute the gradient magnitude (important for watershed)
% ------------------------------------------------------------
gmag = imgradient(gray);

figure; imshow(gmag,[]);
title('Gradient Magnitude Image');

% ------------------------------------------------------------
% Step 3: Apply Watershed transform
% ------------------------------------------------------------
L = watershed(gmag);          % Label matrix of watershed regions

% Create a binary mask for watershed ridge lines
```

```
watershedBoundaries = L == 0;

figure; imshow(watershedBoundaries);
title('Watershed Ridge Lines');

% ---------------------------------------------------------
% Step 4: Superimpose the watershed boundaries on the original image
% ---------------------------------------------------------
img2 = img;
img2(:,:,1) = img(:,:,1) + uint8(watershedBoundaries)*255;  % Red boundaries

figure; imshow(img2);
title('Watershed Segmentation Result');
```

Output:





Micro Project:

3. Spatial Frequency and Fourier Transforms

Industry Case Study: MRI and CT Image Processing
Company Example: Siemens Healthineers, GE Healthcare
Activity:

- Show an MRI or CT image and its Fourier transform (frequency domain).
- Identify low-frequency regions (overall structure) vs. high-frequency regions (edges, noise).

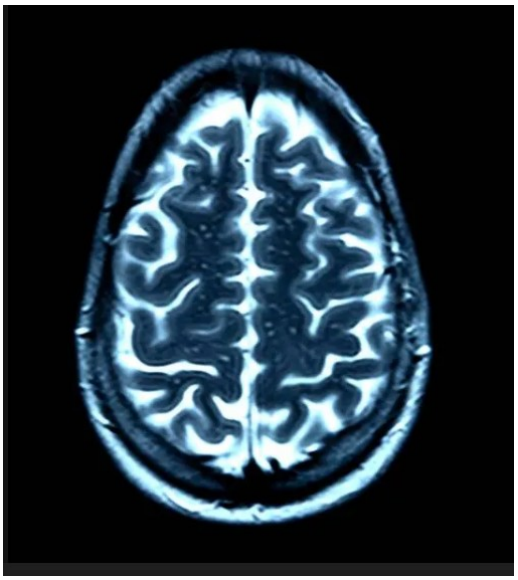
Code:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from PIL import Image


def load_grayscale_image(path, target_size=None):
    """Load an image, convert to grayscale float32 in [0, 1]."""
    img = Image.open(path).convert("L")  # L = 8-bit grayscale
    if target_size is not None:
        img = img.resize(target_size, Image.BILINEAR)
    img = np.asarray(img, dtype=np.float32)
    img = img / 255.0
    return img


def compute_fft2(image):
    """Compute centered 2D FFT and log-magnitude spectrum."""
    F = np.fft.fft2(image)
    F_shift = np.fft.fftshift(F)
    magnitude = np.abs(F_shift)
    log_magnitude = np.log1p(magnitude)  # log(1 + |F|)
    return F, F_shift, log_magnitude


def make_circular_masks(shape, low_radius_frac=0.1, high_radius_frac=0.25):
    """
    Create low-pass and high-pass circular masks.

    low_radius_frac, high_radius_frac are relative to min(H, W)/2.
    """
    h, w = shape
    cy, cx = h // 2, w // 2
    Y, X = np.ogrid[:h, :w]
    dist = np.sqrt((X - cx) ** 2 + (Y - cy) ** 2)

    max_radius = min(h, w) / 2.0
    low_radius = low_radius_frac * max_radius
    high_radius = high_radius_frac * max_radius

    low_pass_mask = dist <= low_radius
```

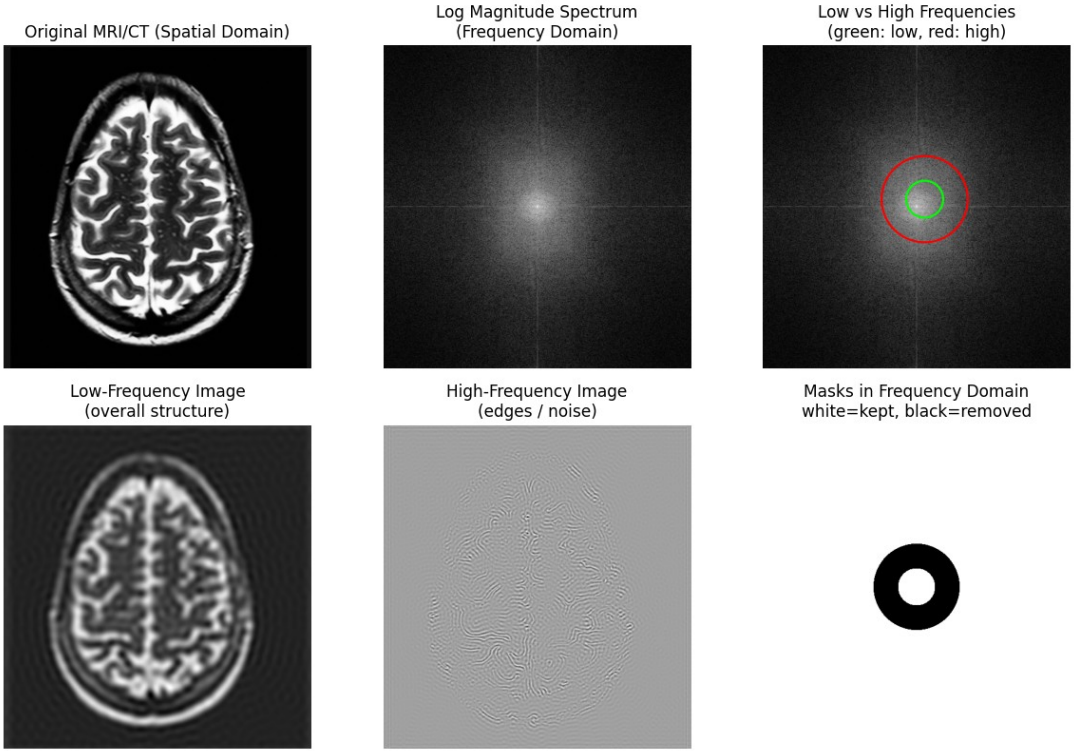
```
    high_pass_mask = dist >= high_radius

    return low_pass_mask.astype(np.float32), high_pass_mask.astype(np.float32), (
        low_radius,
        high_radius,
        (cx, cy),
    )


def reconstruct_from_fft(F_shift, mask):
    """Apply mask in frequency domain and reconstruct spatial image."""
    F_shift_masked = F_shift * mask
    F_unshift = np.fft.ifftshift(F_shift_masked)
    img_rec = np.fft.ifft2(F_unshift)
    img_rec = np.real(img_rec)
    # Normalize to [0, 1] for display
    img_min, img_max = img_rec.min(), img_rec.max()
    if img_max > img_min:
        img_rec = (img_rec - img_min) / (img_max - img_min)
    else:
        img_rec = np.zeros_like(img_rec)
    return img_rec


def main():

    # 1. Load MRI/CT slice
    image_path = "mri_slice.png"  # put your MRI/CT PNG/JPG here
    img = load_grayscale_image(image_path, target_size=None)

    # 2. Compute FFT and magnitude spectrum
    F, F_shift, log_mag = compute_fft2(img)

    # 3. Build low-pass and high-pass masks
    low_mask, high_mask, (low_r, high_r, center) = make_circular_masks(
        img.shape, low_radius_frac=0.12, high_radius_frac=0.28
    )

    # 4. Reconstruct low-frequency and high-frequency images
    img_low = reconstruct_from_fft(F_shift, low_mask)
    img_high = reconstruct_from_fft(F_shift, high_mask)

    # 5. Plot results
    fig, axes = plt.subplots(2, 3, figsize=(12, 8))
    ax_orig, ax_fft, ax_fft_overlay, ax_low, ax_high, ax_masks = axes.ravel()

    # Original image
    ax_orig.imshow(img, cmap="gray")
    ax_orig.set_title("Original MRI/CT (Spatial Domain)")
    ax_orig.axis("off")

    # Magnitude spectrum
    ax_fft.imshow(log_mag, cmap="gray")
    ax_fft.set_title("Log Magnitude Spectrum\n(Frequency Domain)")
```

```
    ax_fft.axis("off")

    # Magnitude spectrum with low/high-freq rings
    ax_fft_overlay.imshow(log_mag, cmap="gray")
    cy, cx = center
    low_circle = Circle((cx, cy), low_r, edgecolor="lime", facecolor="none", linewidth=1.5)
    high_circle = Circle((cx, cy), high_r, edgecolor="red", facecolor="none", linewidth=1.5)
    ax_fft_overlay.add_patch(low_circle)
    ax_fft_overlay.add_patch(high_circle)
    ax_fft_overlay.set_title("Low vs High Frequencies\n(green: low, red: high)")
    ax_fft_overlay.axis("off")

    # Low-frequency reconstruction
    ax_low.imshow(img_low, cmap="gray")
    ax_low.set_title("Low-Frequency Image\n(overall structure)")
    ax_low.axis("off")

    # High-frequency reconstruction
    ax_high.imshow(img_high, cmap="gray")
    ax_high.set_title("High-Frequency Image\n(edges / noise)")
    ax_high.axis("off")

    # Visualize masks
    ax_masks.imshow(low_mask + high_mask, cmap="gray")
    ax_masks.set_title("Masks in Frequency Domain\nwhite=kept, black=removed")
    ax_masks.axis("off")

    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    main()
```

Input:



Output:


Original MRI/CT (Spatial Domain)
Log Magnitude Spectrum
(Frequency Domain)
Low vs High Frequencies
(green: low, red: high)
Low-Frequency Image
(overall structure)
High-Frequency Image
(edges / noise)
Masks in Frequency Domain
white=kept, black=removed

Discussion Points:

1. How frequency filtering is used to enhance or remove details:

In the frequency domain, the position of a pixel represents a specific frequency, not a spatial location.

Center of the Spectrum (Low Frequencies): These represent large, slow-changing features like the overall shape of the skull, the background intensity, and smooth gradients.

Outer Regions (High Frequencies): These represent rapid changes in intensity, which physically represent edges, sharp details, and noise.

Filtering is simply the process of creating a "mask" to multiply against the spectrum. It can:

- Remove Details: This is done by using a Low Pass Smoothing Filter, we only allow frequencies below a certain point (Generally the center of the image) to pass and block the outer regions or edges.
- Enhance Details: This is done by using a High Pass Edge Detection Filter, by blocking the center and keeping only the outer region we remove the smooth gradients.

2. Applications:

Medical Imaging (MRI & CT): Noise Reduction by Radiologists due to static noise in medical sensors, or Artifact Removal by Doctors upon applying a Notch Filter to remove interferences from the final image.

Compression: When you save a photo as a PNG/JPEG, the algorithm converts small blocks of the image into the frequency domain.

Image Reconstruction: In MRI, the machine does not take a picture of the brain directly. It captures raw data in the frequency domain. The image we see is actually the output of the process. The Fourier Transform is what the machine actually sees. To reconstruct the image for the doctor, the computer must perform an Inverse Fourier Transform. Without this math, an MRI is just a meaningless collection of frequencies.

Conclusion:

This mini-project demonstrates how a single MRI/CT slice can be decomposed into low- and high-frequency components using the 2D Fourier transform, making the abstract idea of spatial frequency concrete and visual. By showing that central k-space (low frequencies) governs global contrast while peripheral k-space (high frequencies) controls edge sharpness and fine detail, the experiment mirrors how real MRI systems rely on Fourier reconstruction and frequency-domain filtering to balance resolution and noise. The low-pass and high-pass reconstructions together highlight the core trade-off in medical image processing—denoising versus edge preservation—providing an intuitive link between classroom theory and practical design choices in scanners from companies like Siemens Healthineers and GE Healthcare.

MathWorks | *Training Services*

Course Completion Certificate

Advaith Kashyap

has successfully completed **100%** of the self-paced training course

Computer Vision Onramp

DIRECTOR, TRAINING SERVICES

4 December 2025

MathWorks | *Training Services*

Course Completion Certificate

Advaith Kashyap

has successfully completed **100%** of the self-paced training course

Object Detection with Deep Learning

DIRECTOR, TRAINING SERVICES

4 December 2025