

**SWAMI KESHVANAND INSTITUTE OF TECHNOLOGY,
MANAGEMENT AND GRAMOTHAN, JAIPUR**



**Hands on Lab Guide
(Lab Manual)**

**Computer Vision Lab
IV Year B.Tech. VII SEM
(Course Code: 7CAI4-22)
Session 2024-2025**

**Department of Computer Science & Engineering
SKIT, JAIPUR**



**Swami Keshvanand Institute of Technology, Management &
Gramothan, Ramnagar, Jagatpura, Jaipur-302017**

LAB MANUAL
Computer Vision Lab
(7CAI4-22)

VERSION 2.0

	<i>PREPARED BY</i>	<i>APPROVED BY</i>
NAME	Dr. Nilam	Prof. Mehul Mahrishi
DESIGNATION	Associate Professor	HOD(CSE)
SIGNATURE & REVIEW DATE		

Department of Computer Science & Engineering,
Jaipur – 302017, Rajasthan (INDIA)
E-Mail: hodcs@skit.ac.in, URL: www.skit.ac.in

LAB RULES

Responsibilities of Users

Users are expected to follow some fairly obvious rules of conduct:



Always:

- Enter the lab on time and leave at proper time.
- Wait for the previous class to leave before the next class enters.
- Keep the bag outside in the respective racks.
- Utilize lab hours in the corresponding.
- Turn off the machine before leaving the lab unless a member of lab staff has specifically told you not to do so.
- Leave the labs at least as nice as you found them.
- If you notice a problem with a piece of equipment (e.g. a computer doesn't respond) or the room in general (e.g. cooling, heating, lighting) please report it to lab staff immediately. Do not attempt to fix the problem yourself.

**Never:**

- Don't abuse the equipment.
- Do not adjust the heat or air conditioners. If you feel the temperature is not properly set, inform lab staff; we will attempt to maintain a balance that is healthy for people and machines.
- Do not attempt to reboot a computer. Report problems to lab staff.
- Do not remove or modify any software or file without permission.
- Do not remove printers and machines from the network without being explicitly told to do so by lab staff.
- Don't monopolize equipment. If you're going to be away from your machine for more than 10 or 15 minutes, log out before leaving. This is both for the security of your account, and to ensure that others are able to use the lab resources while you are not.
- Don't use internet, internet chat of any kind in your regular lab schedule.
- Do not download or upload of MP3, JPG or MPEG files.
- No games are allowed in the lab sessions.
- No hardware including USB drives can be connected or disconnected in the labs without prior permission of the lab in-charge.
- No food or drink is allowed in the lab or near any of the equipment. Aside from the fact that it leaves a mess and attracts pests, spilling anything on a keyboard or other piece of computer equipment could cause permanent, irreparable, and costly damage. (and in fact *has*) If you need to eat or drink, take a break and do so in the canteen.
- Don't bring any external material in the lab, except your lab record, copy and books.
- Don't bring the mobile phones in the lab. If necessary then keep them in silence mode.
- Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.

If you are having problems or questions, please go to either the faculty, lab in-charge or the lab supporting staff. They will help you. We need your full support and cooperation for smooth functioning of the lab.



INSTRUCTIONS

BEFORE ENTERING IN THE LAB

- All the students are supposed to prepare the theory regarding the next experiment/ Program.
- Students are supposed to bring their lab records as per their lab schedule.
- Previous experiment/program should be written in the lab record.
- If applicable trace paper/graph paper must be pasted in lab record with proper labeling.
- All the students must follow the instructions, failing which he/she may not be allowed in the lab.

WHILE WORKING IN THE LAB

- Adhere to the experimental schedule as instructed by the lab in-charge/faculty.
- Get the previously performed experiment/ program signed by the faculty/ lab in charge.
- Get the output of current experiment/program checked by the faculty/ lab in charge in the lab copy.
- Each student should work on his/her assigned computer at each turn of the lab.
- Take responsibility of valuable accessories.
- Concentrate on the assigned practical and don't play games.
- If anyone is caught red-handed carrying any equipment of the lab, then he/she will have to face serious consequences.

Marking/Assessment System

Total Marks -100

Distribution of Marks - 60 (Sessional)

Attendance	File Work	Performance	Viva	Total
10	10	30	10	60

Distribution of Marks - 40 (End Term)

Depends on Examiner

File Work	Performance	Viva	Total
10	20	10	40



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Scheme & Syllabus

IV Year- VII & VIII Semester: B. Tech. Computer Science and Engineering (AI)

7CAI4-22: Computer Vision Lab

Credit: 2

Max. Marks: 100 (IA:60, ETE:40)

0L+0T+4P

End Term Exam: 2 Hours

SN	List of Experiments
1	Implementing various basic image processing operations in python/matlab/open-CV: Reading image, writing image, conversion of images, and complement of an image.
2	Implement contrast adjustment of an image. Implement Histogram processing and equalization.
3	Use of Fourier transform for filtering the image.
4	Utilization of SIFT and HOG features for image analysis.
5	Performing/Implementing image segmentation.
6	Object detection and Recognition on available online image datasets using YOLO Model.

REFERENCE BOOKS

1	"OpenCV-Python Tutorials" (Official OpenCV documentation and tutorial), Online tutorials documentation for MATLAB and OpenCV.
2	"Python for Computer Vision Handbook" by Benjamin Root.
3	YOLO Model documentation and tutorials,



Computer Vision Lab (7CAI4-22)

List of Experiments

Implementing various basic image processing operations in Python/Matlab/open-CV:

1. Reading images, writing image, conversion of images, and complementing of an image.
2. Implement contrast adjustment of an image. Implement Histogram processing and equalization.
3. Use of Fourier transform for filtering the image.
4. Utilization of SIFT and HOG features for image analysis.
5. Performing/Implementing image segmentation.
6. Object detection and Recognition on available online image datasets using the YOLO Model.

Beyond Syllabus:

1. Implementation of image restoring techniques
2. Implementation of Image Intensity slicing technique for image enhancement
3. Canny edge detection Algorithm



INDEX

Sr. No.	Topic	Page Number
1	Lab Plan	
2	Lab Objective and Outcome	
3	Experiment No 1	
4	Experiment No 2	
5	Experiment No 3	
6	Experiment No 4	
7	Experiment No 5	
8	Experiment No 6	
9	Beyond Syllabus	
10	Experiment No 7	
11	Experiment No 8	
12	Experiment No 9	
13	Viva Questions	
14	References	
15	Vision Mission, PEO, PO	

LAB PLAN

Total number of experiments 9

Total number of turns required 9

Number of turns required for

Experiment Number	Turns	Scheduled Day
Exp. 1	1	Day 1
Exp. 2	1	Day 2
Exp. 3	1	Day 3
Exp. 4	1	Day 4
Exp. 5	1	Day 5
Exp. 6	1	Day 6
Exp. 7	1	Day 7
Exp. 8	1	Day 8
Exp. 9	1	Day 9

Distribution of Lab Hours:

Attendance 05 minutes

Explanation of features of language 15 minutes

Explanation of experiment 15 minutes

Performance of experiment 70 minutes

Viva / Quiz / Queries 15 minutes

Total 120 Minutes (2 Hrs.)

Lab Objective and Outcome

Objective

The primary aim of this course is to provide essential and hands-on knowledge about computer vision components and to cultivate the necessary skills for image processing, applicable both in industry and research settings.

Key objectives of the course include:

1. Introducing fundamental terminology, technology, and its practical applications.
2. Exploring the concept of image processing and its essential operations.
3. Familiarizing students with Matlab, a versatile tool for image-related operations.
4. Introducing various datasets available on different platforms, widely used for analysis in the field of image processing.
5. Guiding students in the practical implementation of image processing techniques using both online and offline image datasets.

By the end of this course, students will be equipped with a solid foundation in computer vision, image processing techniques, and the practical tools required for success in this field.

Course Outcomes

After completion of this course, students will be able to –

7CAI4-22.1	Explain the concept and Application of image processing for computer vision.
7CAI4-22.1	Illustrate key technologies and implement Histogram processing and Equalization for an image.
7CAI4-22.1	Implement and Analyze the Fourier transform for filtering the image.
7CAI4-22.1	Application for performing and implementing different methods of image segmentation
7CAI4-22.1	Design an Image processing application project using YOLO Model online datasets.

Experiment – 1

Aim: Reading images, writing image, conversion of images, and complementing of an image.

Matlab Code:

```
% Reading an image
originalImage = imread('example.jpg'); % Replace 'example.jpg' with your image file

% Display the original image
subplot(2, 2, 1);
imshow(originalImage);
title('Original Image');

% Writing the original image to a new file
imwrite(originalImage, 'original_image.jpg'); % Save as 'original_image.jpg'

% Convert the image to grayscale
grayImage = rgb2gray(originalImage);

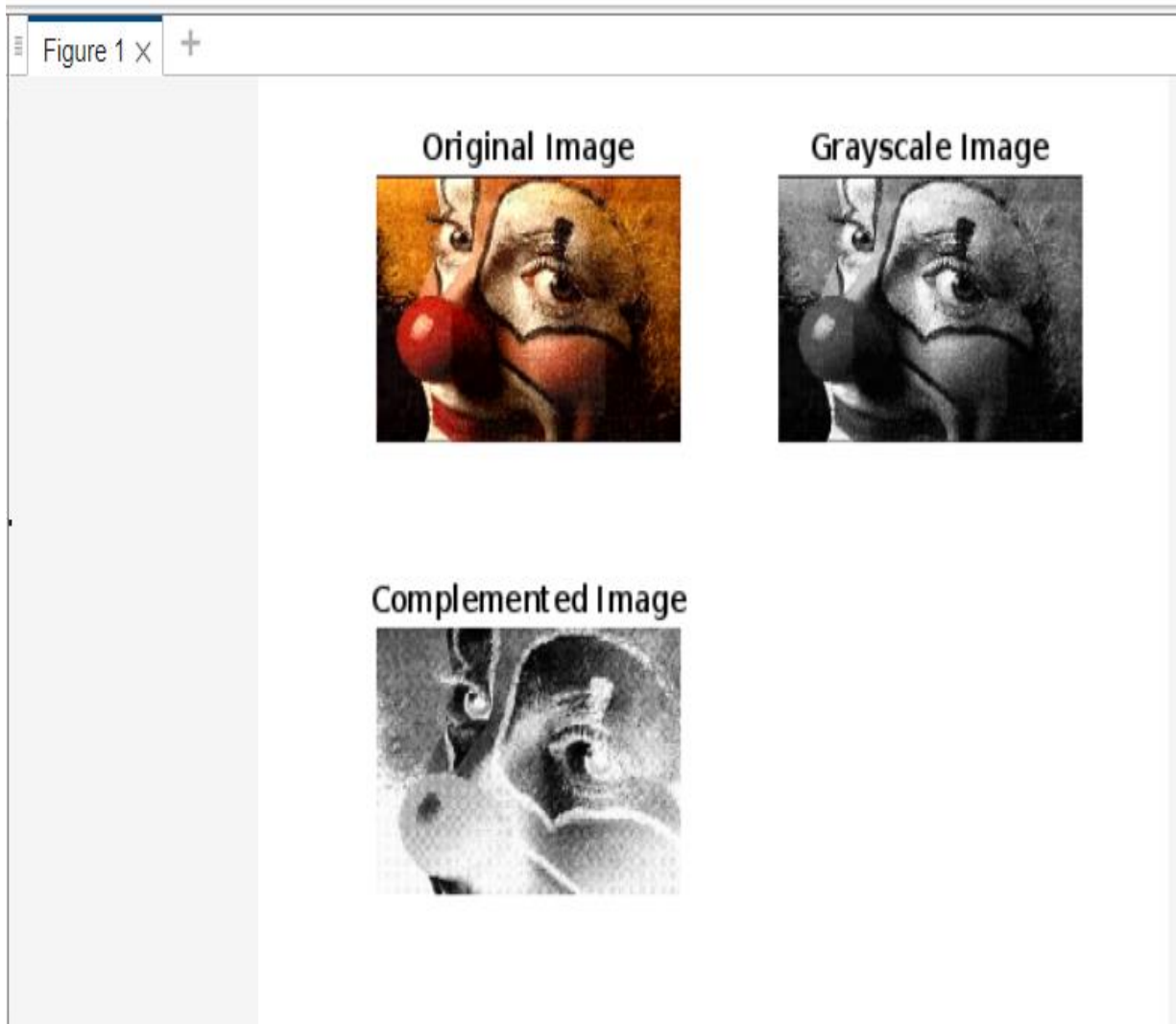
% Display the grayscale image
subplot(2, 2, 2);
imshow(grayImage);
title('Grayscale Image');

% Writing the grayscale image to a new file
imwrite(grayImage, 'grayscale_image.jpg'); % Save as 'grayscale_image.jpg'

% Complementing (inverting) the grayscale image
complementedImage = imcomplement(grayImage);

% Display the complemented image
subplot(2, 2, 3);
imshow(complementedImage);
title('Complemented Image');

% Writing the complemented image to a new file
imwrite(complementedImage, 'complemented_image.jpg'); % Save as 'complemented_image.jpg'
```



In this code:

1. We read an image using **imread**.
2. We display the original image.
3. We write the original image to a new file using **imwrite**.
4. We convert the original image to grayscale using **rgb2gray**.
5. We display the grayscale image.
6. We write the grayscale image to a new file.
7. We complement (invert) the grayscale image using **imcomplement**.
8. We display the complemented image.
9. We write the complemented image to a new file.

Make sure to replace '**example.jpg**' with the path to your own image file. This code covers the basic operations of reading, writing, converting, and complementing images in MATLAB.

Experiment – 2

Aim: Implement contrast adjustment of an image. Implement Histogram processing and equalization.

Code

% Image Enhancement

```
I=imread('cancercell.jpg');  
subplot(4,2,1); imshow(I); title('Original Image');  
  
g=rgb2gray(I);  
subplot(4,2,5); imshow(g); title('Gray Image');  
  
J=imadjust(g,[0.3 0.7],[]);  
subplot(4,2,3); imshow(J); title('Enhanced Image');  
  
D= imadjust(I,[0.2 0.3 0; 0.6 0.7 1],[]);  
subplot(4,2,4);imshow(D);title('Enhanced Image 2');
```

% Histogram and Histogram Equalization

```
subplot(4,2,7); imhist(g); title('Histogram of Gray Image');  
  
m=histeq(g);  
subplot(4,2,6); imshow(m); title('Equalized Image');  
subplot(4,2,8); imhist(m); title('Histogram of Equalized Image');
```

Original Image



Enhanced Image



Enhanced Image 2



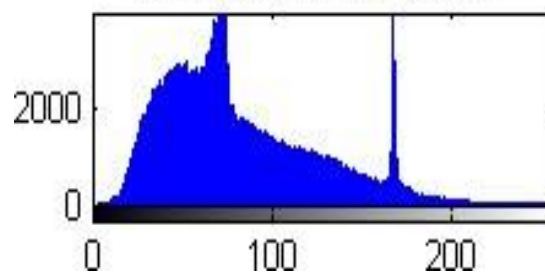
Gray Image



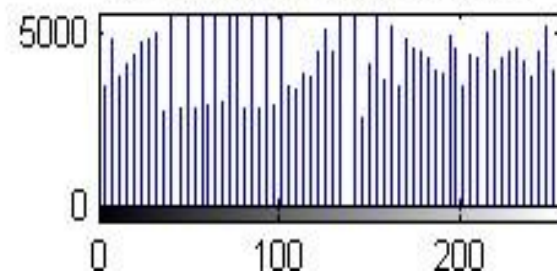
Equalized Image



Histogram of Gray Image



Histogram of Equalized Image



Experiment – 3

Aim: Use of Fourier transform for filtering the image.

Code:

```
% Read an image
originalImage = imread('example.jpg'); % Replace 'example.jpg' with your image file

% Convert the image to grayscale if it's in color
if size(originalImage, 3) == 3
    originalImage = rgb2gray(originalImage);
end

% Display the original image
subplot(2, 3, 1);
imshow(originalImage);
title('Original Image');

% Compute the 2D Fourier Transform of the image
fourierTransform = fft2(double(originalImage));

% Shift the zero frequency components to the center
fourierTransformShifted = fftshift(fourierTransform);

% Create a simple low-pass filter (e.g., a circular mask)
[M, N] = size(originalImage);
radius = 50; % Adjust the radius to control the filter size
[x, y] = meshgrid(1:N, 1:M);
centerX = N/2;
centerY = M/2;
mask = (x - centerX).^2 + (y - centerY).^2 <= radius^2;

% Apply the filter by multiplying it with the Fourier Transform
filteredTransform = fourierTransformShifted .* mask;

% Shift the frequency components back to their original positions
filteredTransformShifted = ifftshift(filteredTransform);

% Compute the inverse Fourier Transform to get the filtered image
```

```
filteredImage = ifft2(filteredTransformShifted);

% Display the filtered image
subplot(2, 3, 2);
imshow(abs(filteredImage), []);
title('Filtered Image');

% Display the magnitude of the Fourier Transform
subplot(2, 3, 3);
imshow(log(1 + abs(fourierTransformShifted)), []);
title('Magnitude of Fourier Transform');

% Display the filter mask
subplot(2, 3, 4);
imshow(mask, []);
title('Filter Mask');

% Display the phase of the Fourier Transform
subplot(2, 3, 5);
imshow(angle(fourierTransformShifted), []);
title('Phase of Fourier Transform');

% Display the phase of the filtered image
subplot(2, 3, 6);
imshow(angle(filteredImage), []);
title('Phase of Filtered Image');

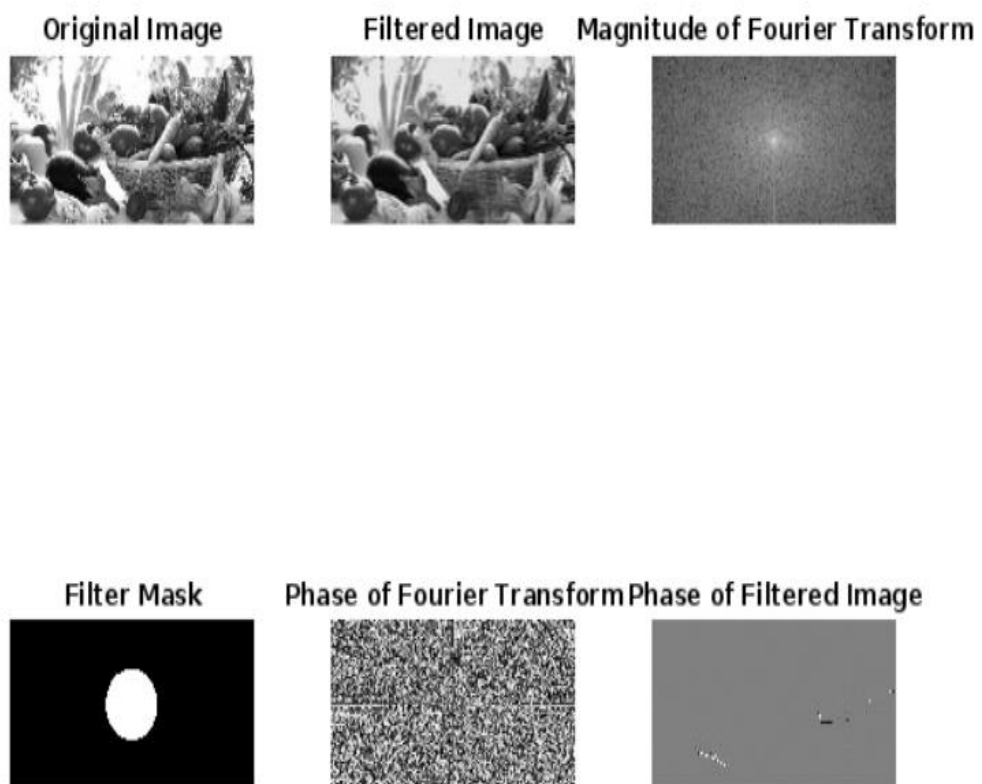
% Wait for user input before closing the figure
pause;
close;
```

In the above code, we added two additional subplots to display the phase of both the Fourier Transform and the filtered image. This will help you visualize both the magnitude and phase changes resulting from the filtering process.

Original Image:



Figure 1 × +



Experiment – 4

Aim: Utilization of SIFT and HOG features for image analysis.

MATLAB's Computer Vision Toolbox does not provide a built-in SIFT detector. Instead, you can use the VLFeat library for SIFT feature extraction. Here's an example of how to do it:

First, make sure you have the VLFeat library installed and added to MATLAB. You can download VLFeat from the official website (<https://www.vlfeat.org/>) and follow their installation instructions.

Once VLFeat is installed, you can use it for SIFT feature extraction as follows:

Code

```
% Load an image
```

```
img = imread('example.jpg'); % Replace 'example.jpg' with your image file
```

```
% Convert the image to grayscale if it's in color
```

```
if size(img, 3) == 3
```

```
    img = rgb2gray(img);
```

```
end
```

```
% Add the VLFeat toolbox to MATLAB (adjust the path as needed)
```

```
addpath('path_to_vlfeat/vlfeat-0.9.21/toolbox'); % Replace 'path_to_vlfeat' with the actual path
```

```
% Initialize VLFeat
```

```
run('vlfeat-0.9.21/toolbox/vl_setup');
```

```
% Extract SIFT features
```

```
[f, d] = vl_sift(single(img));
```

```
% Visualize SIFT features on the image
```

```
siftImage = insertMarker(img, f(1:2, :), 'Color', 'red', 'Size', 5);
```

```
% Display the image with SIFT features
```

```
imshow(siftImage);
```

```
title('Image with SIFT Features');
```

```
% Create a HOG feature extractor
```

```
hogExtractor = vision.HOGDescriptor;
```

```
% Extract HOG features
```

```
hogFeatures = extractHOGFeatures(img);
```

```
% Display HOG features
```

```
figure;
```

```
subplot(1, 2, 1);
```

```
imshow(img);
```

```
title('Original Image');
```

```
subplot(1, 2, 2);
```

```
plot(hogFeatures);
```

```
title('HOG Features');
```

```
% Perform further analysis using SIFT and HOG features as needed
```

In this code:

- We use VLFeat to extract SIFT features.
- Make sure to replace '**path_to_vlfeat**' with the actual path to your VLFeat toolbox installation.
- We then visualize the SIFT features on the image and display them.
- The HOG feature extraction part remains the same as in the previous code.

Ensure that you have VLFeat properly installed and configured in MATLAB for this code to work.

Experiment – 5

Aim: Performing/Implementing image segmentation.

Implementing image segmentation in MATLAB typically involves using various segmentation algorithms, depending on the specific requirements of your task. One commonly used algorithm is k-means clustering for color-based segmentation. Here's an example of how to perform image segmentation using k-means clustering:

Matlab Code

```
% Read an image
originalImage = imread('example.jpg'); % Replace 'example.jpg' with your image file

% Display the original image
subplot(2, 2, 1);
imshow(originalImage);
title('Original Image');

% Convert the image to double precision for k-means
imageData = double(originalImage);

% Reshape the image data into a 2D matrix
[m, n, ~] = size(imageData);
imageData = reshape(imageData, m * n, 3);

% Perform k-means clustering (adjust the number of clusters as needed)
numClusters = 4; % Change the number of clusters as per your requirement
[clusterIndices, clusterCenters] = kmeans(imageData, numClusters);

% Reshape the clusterIndices back to the original image size
segmented = reshape(clusterIndices, m, n);

% Display the segmented image
subplot(2, 2, 2);
imshow(segmented, []);
title('Segmented Image');

% Create a mask for each segment
segmentMasks = cell(numClusters, 1);
for i = 1:numClusters
    mask = (segmented == i);
```

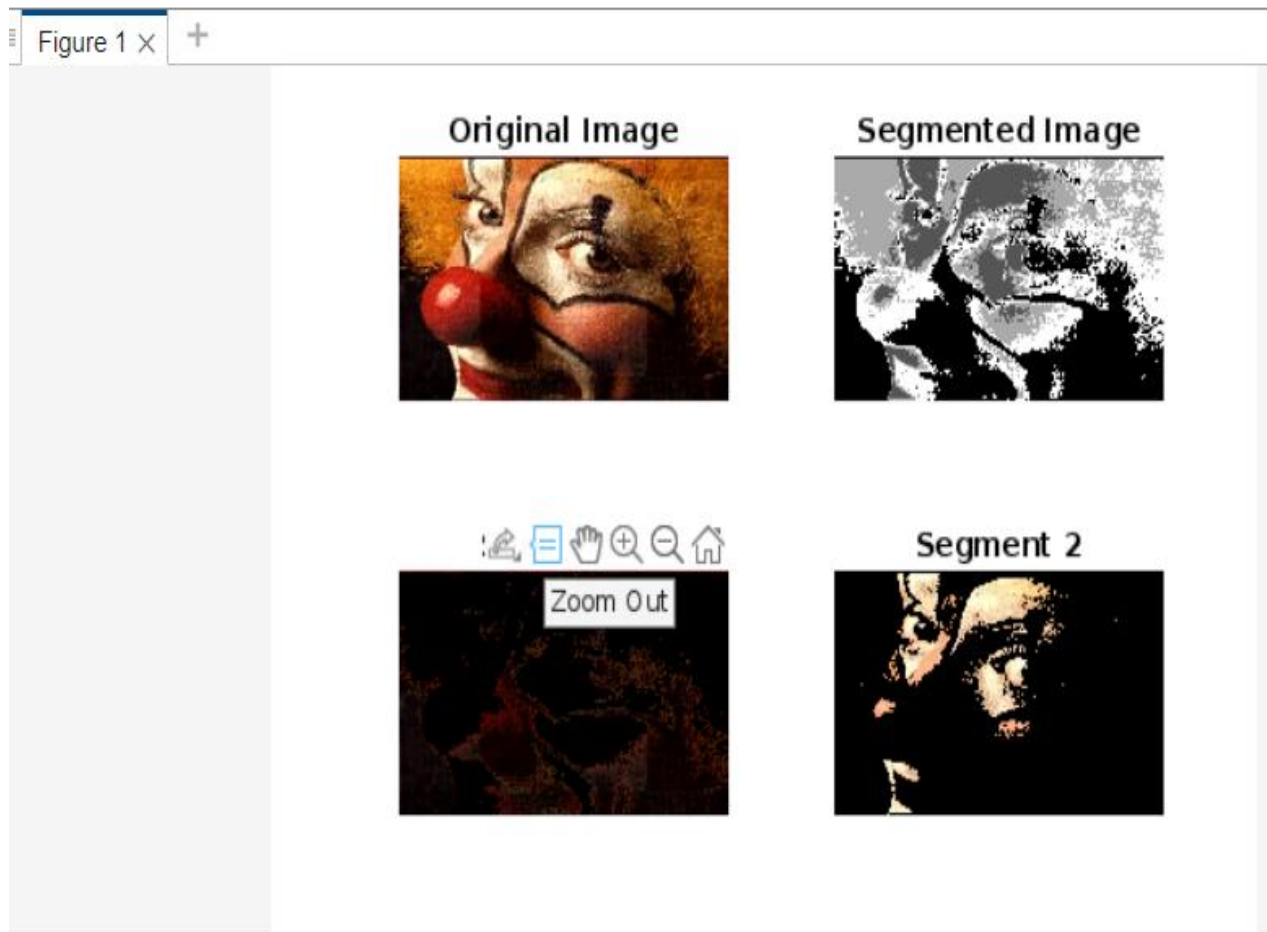
```
segmentMasks{i} = mask;
end

% Display each segmented region
for i = 1:numClusters
    subplot(2, 2, i + 2);
    segmentedRegion = originalImage;
    segmentedRegion(segmentMasks{i}, [1, 1, 3]) = 0;
    imshow(segmentedRegion);
    title(['Segment ', num2str(i)]);
end
```

In this code:

1. We read an image and display it.
2. We convert the image to double precision for k-means clustering.
3. We reshape the image data into a 2D matrix to prepare it for clustering.
4. We perform k-means clustering with a specified number of clusters.
5. We reshape the cluster indices back to the original image size.
6. We display the segmented image.
7. We create masks for each segment.
8. We display each segmented region individually.

You can adjust the **numClusters** variable to control the number of segments in the output. Feel free to experiment with different segmentation algorithms and parameters based on your specific image segmentation needs.



Experiment – 6

Aim: Object detection and Recognition on available online image datasets using the YOLO Model.

Implementing object detection and recognition using the YOLO (You Only Look Once) model in MATLAB can be complex, as it often requires pre-trained models and a deep learning framework like TensorFlow or PyTorch. However, I can provide you with a general outline of the steps involved in performing object detection and recognition using a pre-trained YOLO model. You may need to adapt this code to your specific dataset and requirements.

1. Setup Deep Learning Framework: Ensure you have a deep learning framework like TensorFlow or PyTorch installed in MATLAB.
2. Download Pre-trained YOLO Model: Download a pre-trained YOLO model, such as YOLOv3 or YOLOv4, along with its configuration file and weights. You can find pre-trained YOLO models on the official YOLO website or other sources.
3. Load and Configure the Model: Load the pre-trained YOLO model in MATLAB using the deep learning framework. Configure the model according to the model's architecture and requirements.
4. Load and Preprocess the Image Dataset: Load the image dataset you want to perform object detection and recognition on. Preprocess the images to match the input format expected by the YOLO model (e.g., resizing, normalization).
5. Perform Object Detection: Use the pre-trained YOLO model to perform object detection on the images in the dataset. This will generate bounding boxes and class predictions for detected objects.
6. Display Detected Objects: Visualize the detected objects by drawing bounding boxes and labels on the images.

Here's a high-level example of how you might perform these steps using MATLAB and TensorFlow for object detection and recognition with a pre-trained YOLO model (assuming you have the model files and dataset ready):

Matlab Code:

```
% Load the pre-trained YOLO model and configure it (using TensorFlow)
net = importKerasNetwork('yolov3.h5'); % Replace with the path to your model
inputSize = [416 416 3]; % Input size expected by the model

% Load and preprocess the image dataset
imageFolder = 'path_to_dataset_folder'; % Replace with your dataset folder
imageFiles = dir(fullfile(imageFolder, '*.jpg')); % Assuming JPEG images
```

```
numImages = length(imageFiles);

for i = 1:numImages
    % Load and preprocess each image
    img = imread(fullfile(imageFolder, imageFiles(i).name));
    img = imresize(img, inputSize(1:2));
    img = img/255; % Normalize pixel values (if needed)

    % Perform object detection
    detections = detectYOLO(net, img);

    % Visualize and process detections (e.g., filter by confidence score)
    % Display detected objects with bounding boxes and labels
    imshow(img);
    hold on;
    for j = 1:numel(detections)
        bbox = detections(j).Location;
        label = detections(j).Label;
        score = detections(j).Confidence;
        if score > 0.5 % Adjust confidence threshold as needed
            rectangle('Position', bbox, 'EdgeColor', 'r', 'LineWidth', 2);
            text(bbox(1), bbox(2) - 10, label, 'Color', 'r', 'FontSize', 12);
        end
    end
    hold off;

    % Further processing or recognition for detected objects can be done here
end
```

Note: Please note that this is a simplified example, and you'll need to adapt it to your specific dataset and requirements. Additionally, the actual code may vary based on the pre-trained YOLO model you choose and the deep learning framework you're using in MATLAB.

Beyond Syllabus

Experiment – 7

Aim: Implementation of image restoring techniques

Image restoration is a broad field with various techniques, and the specific implementation depends on the method you want to use. Let's implement image restoration using MATLAB with an example of image denoising using the Total Variation (TV) regularization technique. In this example, we'll use a simple test image and add noise to it before restoring it.

Matlab Code:

```
% Load a sample image
originalImage = imread('cameraman.tif'); % Load a sample image

% Add noise to the image (you can adjust the noise level as needed)
noiseLevel = 25;
noisyImage = imnoise(originalImage, 'gaussian', 0, (noiseLevel/255)^2);

% Display the noisy image
subplot(1, 2, 1);
imshow(noisyImage);
title('Noisy Image');

% Image restoration using Total Variation regularization
lambda = 0.1; % Regularization parameter (adjust as needed)
numIterations = 100; % Number of iterations (adjust as needed)

% Perform TV regularization-based image restoration
restoredImage = tvdenoise(noisyImage, lambda, numIterations);

% Display the restored image
subplot(1, 2, 2);
imshow(restoredImage);
title('Restored Image');
```

```
% Define the TV denoising function (you can use an existing MATLAB implementation)
function denoisedImage = tvdenoise(inputImage, lambda, numIterations)
    u = inputImage;
    [Dux, Duy] = gradient(u);

    for k = 1:numIterations
        u_old = u;

        % Update u using TV denoising
        u = u - lambda * divergence(Dux, Duy);

        % Compute gradient
        [Dux, Duy] = gradient(u);

        % Projection step
        normDu = sqrt(Dux.^2 + Duy.^2);
        normDu(normDu == 0) = 1;
        Dux = Dux ./ normDu;
        Duy = Duy ./ normDu;

        % Ensure the solution is within the range [0, 255]
        u = min(max(u, 0), 255);

        % Check for convergence
        if norm(u - u_old, 'fro') / norm(u_old, 'fro') < 1e-6
            break;
        end
    end

    denoisedImage = u;
end
```

In this code:

1. We load a sample grayscale image (you can replace it with your own image).
2. We add Gaussian noise to the image, with the noise level adjustable by the noiseLevel variable.
3. We perform image restoration using Total Variation (TV) regularization, a popular technique for denoising.

4. The `tvdenoise` function implements the TV denoising algorithm. It iteratively updates the image to reduce noise while preserving edges.

Please note that this is a simplified example. For practical applications, you may need to fine-tune parameters, choose different restoration techniques, or use more advanced denoising algorithms depending on your specific needs and the characteristics **of your images**.

Experiment – 8

Aim: Implementation of Image Intensity slicing technique for image enhancement

Image intensity slicing is a simple technique used for enhancing specific regions of interest in an image by applying a custom intensity mapping function. Here's a MATLAB code example to implement image intensity slicing:

Matlab Code

```
% Load the image
originalImage = imread('sample_image.jpg'); % Load your image here

% Define the intensity range for enhancement
minIntensity = 100; % Minimum intensity for enhancement
maxIntensity = 200; % Maximum intensity for enhancement

% Create a copy of the original image for enhancement
enhancedImage = originalImage;

% Apply intensity slicing
enhancedImage(originalImage >= minIntensity & originalImage <= maxIntensity) = 255;

% Display the original and enhanced images side by side
subplot(1, 2, 1);
imshow(originalImage);
title('Original Image');

subplot(1, 2, 2);
imshow(enhancedImage);
title('Enhanced Image');
```

% Save the enhanced image if needed

imwrite(enhancedImage, 'enhanced_image.jpg'); % Save the enhanced image

In this code:

1. Load your input image (replace '**sample_image.jpg**' with the path to your image).
2. Define the minimum (**minIntensity**) and maximum (**maxIntensity**) intensity values for enhancement. Any pixel values within this range will be set to 255 (white) in the enhanced image.
3. Create a copy of the original image to preserve the original data.
4. Apply the intensity slicing technique by setting pixel values within the specified intensity range to 255 (white).
5. Display both the original and enhanced images side by side using subplots.
6. Optionally, save the enhanced image to a file using **imwrite** if needed.

This simple intensity slicing technique enhances the specified intensity range while keeping other parts of the image unchanged. You can adjust the **minIntensity** and **maxIntensity** values to target the specific intensity range you want to enhance.

Experiment – 9

Aim: Canny edge detection Algorithm

MATLAB code example for implementing the Canny edge detection algorithm with a sample image:

% Load the sample image

originalImage = imread('sample_image.jpg'); % Load your image here

% Convert the image to grayscale if it's not already

if size(originalImage, 3) == 3

 grayImage = rgb2gray(originalImage);

else

 grayImage = originalImage;

end

% Apply Gaussian smoothing to reduce noise and enhance edges

sigma = 1.5; % Adjust the Gaussian filter's standard deviation as needed

smoothedImage = imgaussfilt(grayImage, sigma);

% Perform Canny edge detection

```
lowThreshold = 0.1; % Adjust the low threshold as needed (suggested range: 0.1-0.4)
highThreshold = 0.3; % Adjust the high threshold as needed (suggested range: 0.3-0.7)
edgeImage = edge(smoothedImage, 'Canny', [lowThreshold highThreshold]);

% Display the original and edge-detected images side by side
subplot(1, 2, 1);
imshow(grayImage);
title('Original Grayscale Image');

subplot(1, 2, 2);
imshow(edgeImage);
title('Canny Edge Detection');

% Save the edge-detected image if needed
imwrite(edgeImage, 'canny_edges.jpg'); % Save the edge-detected image
```

In this code:

1. Load your input image (replace '**sample_image.jpg**' with the path to your image).
2. If the input image is in color, it is converted to grayscale using **rgb2gray** because Canny edge detection is typically performed on grayscale images.
3. Apply Gaussian smoothing to the grayscale image to reduce noise and enhance edges. You can adjust the **sigma** parameter to control the degree of smoothing.
4. Perform Canny edge detection using the **edge** function. Adjust the **lowThreshold** and **highThreshold** values to control the sensitivity of edge detection. A larger range between the thresholds results in fewer detected edges.
5. Display both the original grayscale image and the Canny edge-detected image side by side using subplots.
6. Optionally, save the edge-detected image to a file using **imwrite** if needed.

You can experiment with different threshold values and the standard deviation of the Gaussian filter to fine-tune the edge detection results for your specific images.

Viva Questions

Question 1: What is image processing, and why is it important?

Answer: Image processing is the manipulation and analysis of an image to extract useful information or enhance its visual quality. It's important in various fields such as medicine, security, entertainment, and more. It helps in tasks like object detection, image restoration, and feature extraction.

Question 2: What is the difference between grayscale and color images?

Answer: Grayscale images have a single channel representing pixel intensity (typically from 0 to 255), while color images have multiple channels (e.g., RGB or CMYK) representing different color components.

Question 3: How do you read an image in MATLAB, and what is the image data structure?

Answer: You can read an image in MATLAB using the **imread** function. The image is typically stored as a 2D matrix (grayscale) or a 3D matrix (color), where each element represents the pixel intensity.

Question 4: Explain the process of displaying an image in MATLAB.

Answer: You can display an image in MATLAB using the **imshow** function. For grayscale images, you use **imshow(image)**, and for color images, you use **imshow(imageRGB)**.

Question 5: What is image smoothing, and why is it used?

Answer: Image smoothing, also known as blurring, reduces noise and enhances image quality. It's achieved by applying filters like Gaussian or median filters to average or smoothen pixel values.

Question 6: How does the Canny edge detection algorithm work?

Answer: The Canny edge detection algorithm involves the following steps:

1. Smoothing the image with a Gaussian filter to reduce noise.
2. Calculating gradient magnitude and direction.
3. Suppressing non-maximum edge responses.
4. Applying hysteresis thresholding to detect edges.

Question 7: What is image thresholding?

Answer: Image thresholding is the process of converting a grayscale image into a binary image by dividing pixel values into foreground and background based on a threshold value. Pixels below the threshold are set to one value (e.g., 0), and those above are set to another (e.g., 255).

Question 8: How can you perform image resizing in MATLAB?

Answer: Image resizing can be done using the **imresize** function. You provide the input image and the desired dimensions to obtain a resized image.

Question 9: What is the purpose of morphological operations in image processing?

Answer: Morphological operations are used for processing binary images to analyze and manipulate image structures. Operations like dilation, erosion, opening, and closing help in tasks such as noise reduction, object extraction, and shape analysis.

Question 10: How can you perform image segmentation in MATLAB?

Answer: Image segmentation can be achieved using various techniques, including thresholding, clustering, and region-based methods. The choice of method depends on the specific segmentation task. MATLAB provides functions like **regionprops** for region-based segmentation analysis.

Viva questions on experiments

Experiment 1: Reading, Writing, Conversion, and Complementing Images

Question 1: How do you read an image in MATLAB?

Answer: You can read an image in MATLAB using the `imread` function. For example: `image = imread('image.jpg');`

Question 2: How can you save an image in MATLAB?

Answer: You can save an image in MATLAB using the `imwrite` function. For example: `imwrite(image, 'output.jpg');`

Question 3: How do you convert a color image to grayscale in MATLAB?

Answer: You can convert a color image to grayscale using the `rgb2gray` function. For example: `grayImage = rgb2gray(colorImage);`

Question 4: What is image complementing, and how can you complement an image in MATLAB?

Answer: Image complementing involves inverting the pixel values of an image. You can complement an image in MATLAB by subtracting each pixel value from the maximum intensity value (e.g., 255 for an 8-bit image). For example: `complementedImage = 255 - originalImage;`

Experiment 2: Contrast Adjustment, Histogram Processing, and Equalization

Question 5: How can you adjust the contrast of an image in MATLAB?

Answer: You can adjust the contrast of an image using the `imadjust` function, which allows you to stretch or compress the intensity values in the image.

Question 6: What is histogram processing and histogram equalization?

Answer: Histogram processing involves modifying the pixel values of an image based on its histogram. Histogram equalization is a specific technique that spreads out the histogram of an image to enhance its contrast.

Experiment 3: Fourier Transform for Filtering

Question 7: What is the Fourier Transform used for in image processing?

Answer: The Fourier Transform is used for frequency domain analysis in image processing. It can be used for tasks like filtering, denoising, and feature extraction.

Question 8: How can you perform Fourier Transform-based filtering on an image in MATLAB?

Answer: You can perform Fourier Transform-based filtering in MATLAB using functions like `fft2` to transform the image, apply a filter in the frequency domain, and then use `ifft2` to transform it back to the spatial domain.

Experiment 4: SIFT and HOG Features

Question 9: What are SIFT and HOG features used for in image analysis?

Answer: SIFT (Scale-Invariant Feature Transform) and HOG (Histogram of Oriented Gradients) are used for feature extraction in image analysis. SIFT is used for keypoint detection and description, while HOG is used for object detection and recognition.

Question 10: How can you extract SIFT and HOG features from an image in MATLAB?

Answer: MATLAB provides toolboxes like the Computer Vision Toolbox for extracting SIFT and HOG features from images. You can use functions like `detectSURFFeatures` for SIFT and `extractHOGFeatures` for HOG.

Experiment 5: Image Segmentation

Question 11: What is image segmentation, and why is it important?

Answer: Image segmentation is the process of dividing an image into meaningful regions or objects. It's important for object detection, recognition, and analysis.

Question 12: How can you perform image segmentation in MATLAB?

Answer: MATLAB provides various segmentation algorithms and functions, such as `regionprops` for region-based segmentation analysis and the Image Processing Toolbox for more advanced techniques.

Experiment 6: Object Detection and Recognition with YOLO

Question 13: What is YOLO (You Only Look Once), and how does it work for object detection?

Answer: YOLO is a real-time object detection system. It divides an image into a grid and predicts bounding boxes and class probabilities for objects within each grid cell in a single forward pass.

Question 14: How can you implement object detection and recognition using YOLO in MATLAB?

Answer: You can implement object detection and recognition in MATLAB using the YOLO model through deep learning frameworks like the Deep Learning Toolbox. Pretrained YOLO models are available for use.

References:

Text Books:

Fundamentals of Digital Image Processing by Anil K. Jain

- **Author: Anil K. Jain**
- This book provides a solid foundation in digital image processing and includes a wide range of topics and applications.

Reference Books:

1. **"Digital Image Processing Using MATLAB"** by Gonzalez, Woods, and Eddins
 - Authors: Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins
2. **"MATLAB for Engineers"** by Holly Moore
 - Author: Holly Moore
3. **"Digital Image Processing: An Algorithmic Approach with MATLAB"** by Uvais Qidwai and C. S. Chen
 - Authors: Uvais Qidwai and C. S. Chen
4. **"Image Processing, Analysis, and Machine Vision"** by Sonka, Hlavac, and Boyle
 - Authors: Milan Sonka, Vaclav Hlavac, and Roger Boyle
5. **"MATLAB: A Practical Introduction to Programming and Problem Solving"** by Stormy Attaway
 - Author: Stormy Attaway
6. **"Digital Image Processing: Concepts, Algorithms, and Scientific Applications"** by Bernd Jähne
 - Author: Bernd Jähne
7. **"Computer Vision: Algorithms and Applications"** by Richard Szeliski
 - Author: Richard Szeliski

VISION (SKIT)

“To Promote Higher Learning in Advanced Technology and Industrial Research to make our Country a Global Player”

MISSION (SKIT)

“To Promote Quality Education, Training and Research in the field of Engineering by establishing effective interface with Industry and to encourage Faculty to undertake Industry Sponsored Projects for Students”

QUALITY POLICY (SKIT)

“We are committed to ‘achievement of quality’ as an integral part of our institutional policy by continuous self-evaluation and striving to improve ourselves.

Institute would pursue quality in:

- All its endeavors like admissions, teaching-learning processes, examinations, extra and co-curricular activities, industry-institution interaction, research & development, continuing education, and consultancy.
- Functional areas like teaching departments, training & placement cell, library, administrative office, accounts office, hostels, canteen, security services, transport, maintenance section and all other services.”

Vision and Mission of CSE Department**VISION (CSE Department)**

“Producing quality graduates trained in the latest tools and technologies and to be a leading department in the state and region by imparting in-depth knowledge to the students in an emerging technologies in computer science & engineering.”

MISSION (CSE Department)

“Delivering Student’s resources in IT enabled domain through:

- Effective Industry interaction and project based learning
- Motivating our graduates for Employability, entrepreneurship, research and higher education
- Providing excellent engineering skills in a state-of-the-art infrastructure”

DEPARTMENT OF COMPUTER SCIENCE& ENGINEERING**Program Educational Objectives (PEOs)****Graduate of the CSE Program will be:**

- Prepared to be employed in IT industries and be engaged in learning, understanding, and applying new ideas.
- Able to apply their technical knowledge as practicing professionals or engage in graduate education.

- Prepared to be responsible computing professionals in their own area of interest.
- Working successfully in their chosen career individually and within a professional team environment.

Program Outcomes for CSE Department

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering

and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes for CSE Department

PSO1: Graduates will possess strong fundamental concepts on database technologies, Operating systems, advanced programming, Software engineering and other core subjects.

PSO2: Graduates will demonstrate an ability to design, develop, test, debug, deploy, analyze, troubleshoot, maintain, manage and secure the software.

PSO3: Graduates will possess the knowledge of institutions / organizations / companies related to computer science & engineering.