AIM OF THE OPEN-ENDED EXPERIMENT:

This project aims to develop a comprehensive image processing pipeline using MATLAB to analyze and manipulate digital images. The system performs operations like noise reduction, edge detection, segmentation, and frequency-domain analysis, serving as a practical tool for understanding core concepts in image processing.

REQUIREMENTS:

- **Software:** MATLAB (with Image Processing Toolbox)
- **Hardware:** Standard computer (Minimum: 4GB RAM, i3 Processor)
- **Sample Images:** JPEG/PNG format (e.g., sample.jpg)

THEORY:

Image processing is a crucial domain in computer vision, enabling tasks such as feature extraction, that enhances, analyzes, and extracts meaningful features from images. The following steps detail the core techniques used in the system.

Image Processing Pipeline

Step 1: Image Acquisition

The process starts with **loading an image** using MATLAB's imread() function. The image is displayed for reference before processing.

Step 2: Grayscale Conversion

To simplify computations and reduce complexity, the image is converted to grayscale using the luminance-based transformation:

$$I_{gray} = 0.299R + 0.587G + 0.114B$$

where **R, G,** and **B** represent the red, green, and blue color channels, respectively.

Step 3: Noise Removal with Gaussian Filtering

To enhance image quality and remove high-frequency noise, a **Gaussian filter** is applied:

$$G(x,y) = e^{-x2+y2/2a2}/(2\pi\sigma^2)$$

where $\boldsymbol{\sigma = 2}$ controls the degree of smoothing. This ensures that edges are preserved while reducing unwanted noise.

Step 4: Edge Detection using Canny Algorithm

Edges are critical for object recognition. The **Canny edge detector** detects edges using:

1. **Gradient Computation** (using Sobel filters)
2. **Non-Maximum Suppression** (removing weak edges)
3. **Double Thresholding & Hysteresis** (linking strong and weak edges)

This produces a binary image where edges are clearly defined.

Step 5: Image Segmentation using Thresholding

Thresholding is applied using **Otsu's method**, which selects an optimal threshold based on image histogram analysis. The result is a **binary image** where objects are separated from the background.

Step 6: Object Analysis using Region Properties

After segmentation, properties such as **area, perimeter, and centroid** of objects are measured using the regionprops() function. These metrics are useful for object classification and feature analysis.

Step 7: Image Saving

The processed images, including edges and thresholded results, are saved using imwrite() for further analysis or reporting.

3. Advanced Image Processing Techniques

Step 8: Contrast Enhancement using Histogram Equalization

To improve visibility in low-contrast images, **histogram equalization** is applied. This redistributes pixel intensities to enhance contrast. The transformation follows:

$$S_k = (L-1) \sum p_r(r_j)$$

where **L** is the number of intensity levels, and **p_r (r_j)** is the probability distribution of pixel intensities.

Step 9: Morphological Operations (Erosion and Dilation)

Morphological transformations are used to refine the binary image:

- **Erosion**: Removes small noise elements and shrinks objects. Defined as:
$$A \ominus B = \{z | (B)_z \subseteq A\}$$
- **Dilation**: Expands objects, filling small holes. Defined as:
$$A \oplus B = \{z | (Bz) \cap A \neq \emptyset\}$$

A **disk-shaped structuring element** is used to perform these operations.

Step 10: Fourier Transform for Frequency Analysis

The **Fourier Transform** converts the image from the spatial domain to the frequency domain for pattern and texture analysis:

$$F(u,v) = \sum\sum f(x,y) e^{-j2\pi((Nux+Mvy)/MN)}$$

where **F(u,v)** represents the frequency components of the image. The **log magnitude spectrum** is used to visualize the dominant frequency patterns.

CODE OF DEVELOPMENT:

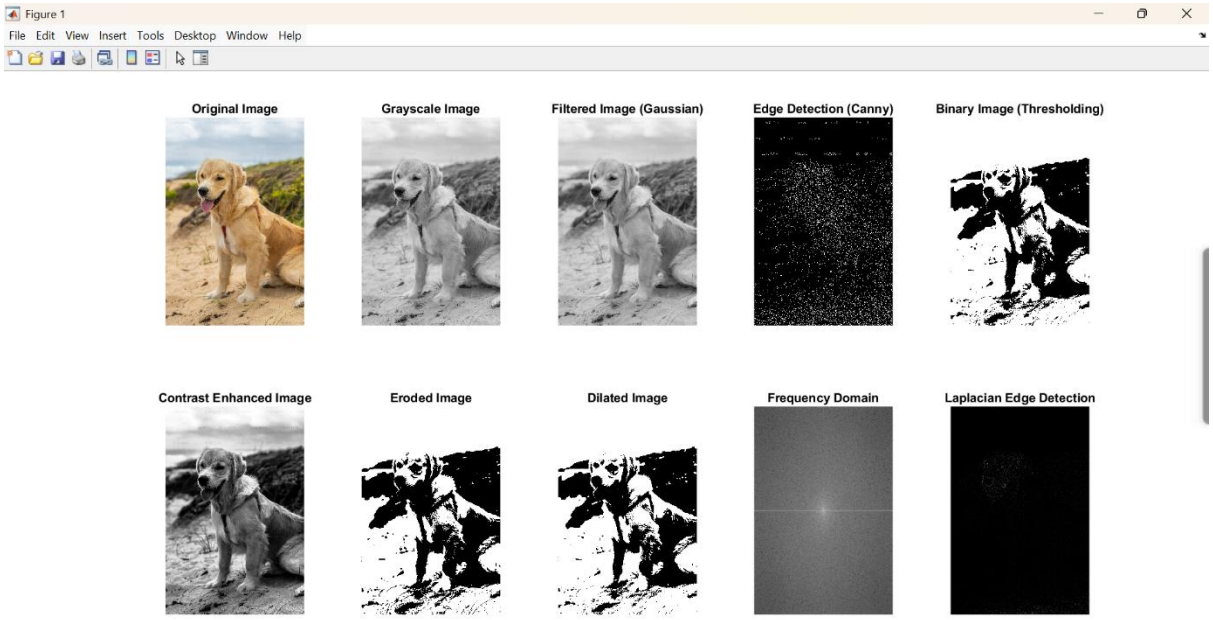
```
% Image Processing Pipeline

% 1. Load an image
img = imread('sample.jpg');

% Check if the image is RGB, convert to grayscale only if necessary
if size(img,3) == 3
    gray_img = rgb2gray(img);
else
    gray_img = img; % Already grayscale
end

% Create a single figure for all outputs
figure;
```

```
% Set the figure size to be larger
set(gcf, 'Position', [100, 100, 1200, 600]); % [left, bottom, width, height]

% 2. Display the original image
subplot(2, 5, 1); % 2 rows, 5 columns, position 1
imshow(img);
title('Original Image');

% 3. Convert to grayscale
subplot(2, 5, 2); % Position 2
imshow(gray_img);
title('Grayscale Image');

% 4. Apply Gaussian filter to remove noise
filtered_img = imgaussfilt(gray_img, 2);
subplot(2, 5, 3); % Position 3
imshow(filtered_img);
title('Filtered Image (Gaussian)');

% 5. Detect edges using Canny edge detector
edge_img = edge(filtered_img, 'Canny');
subplot(2, 5, 4); % Position 4
imshow(edge_img);
title('Edge Detection (Canny)');

% 6. Segment the image using thresholding
bw_img = imbinarize(filtered_img);
subplot(2, 5, 5); % Position 5
imshow(bw_img);
title('Binary Image (Thresholding)');

% 7. Measure properties of objects in the binary image
props = regionprops(bw_img, 'Area', 'Perimeter', 'Centroid');
disp('Object Properties:');
disp(props);

% 8. Save the processed image
imwrite(edge_img, 'processed_image.jpg');

% Advanced Techniques (Optional)

% 9. Histogram Equalization for Contrast Enhancement
enhanced_img = histeq(gray_img); % Enhance contrast using histogram equalization
subplot(2, 5, 6); % Position 6
imshow(enhanced_img);
title('Contrast Enhanced Image');

% 10. Morphological Operations (Erosion and Dilation)
se = strel('disk', 2); % Create a disk-shaped structural element with radius 2
eroded_img = imerode(bw_img, se); % Perform erosion on binary image
dilated_img = imdilate(bw_img, se); % Perform dilation on binary image
subplot(2, 5, 7); % Position 7
imshow(eroded_img);
title('Eroded Image');
subplot(2, 5, 8); % Position 8
```

```
imshow(dilated_img);
title('Dilated Image');

% 11. Fourier Transform for Frequency Domain Analysis
freq_domain = fft2(gray_img); % Compute 2D Fourier Transform
freq_shifted = fftshift(freq_domain); % Shift zero-frequency component to center
magnitude_spectrum = log(1 + abs(freq_shifted)); % Compute log magnitude spectrum
subplot(2, 5, 9); % Position 9
imshow(mat2gray(magnitude_spectrum)); % Normalize for better visualization
title('Frequency Domain');

% 12. Laplacian Edge Detection
laplacian_img = imfilter(gray_img, fspecial('laplacian', 0.2));
subplot(2, 5, 10); % Position 10
imshow(abs(laplacian_img), []); % Display absolute values for proper intensity scaling
title('Laplacian Edge Detection');

% Adjust figure layout for better visualization
set(gcf, 'Color', 'w'); % Set background color to white
```

OBSERVATIONS AND RESULTS:

Image No.	Image Name	Description
1	Original Image	The raw input image before any processing.
2	Grayscale Image	The image is converted from RGB to grayscale for further processing.
3	Filtered Image (Gaussian)	Image after applying a Gaussian filter to reduce noise and smoothen details.
4	Edge Detection (Canny)	Detected edges using the Canny edge detection method, highlighting object boundaries.
5	Binary Image (Thresholding)	The image is converted into a binary format (black & white) using thresholding.
6	Contrast Enhanced Image	Image processed with histogram equalization to improve contrast.
7	Eroded Image	Morphological erosion is applied to remove small noise and shrink objects.
8	Dilated Image	Morphological dilation is applied to expand object boundaries and fill small gaps.
9	Frequency Domain	The image is transformed into the frequency domain using a 2D Fourier Transform.
10	Laplacian Edge Detection	Edges detected using the Laplacian filter, emphasizing intensity transitions.



CONCLUSION:

The Image Processing Pipeline project successfully demonstrates the application of various image processing techniques to analyze and enhance digital images. By integrating preprocessing, filtering, edge detection, segmentation, morphological transformations, and frequency domain analysis, this pipeline provides a comprehensive framework for extracting meaningful information from images.

Through this project, we achieved the following key objectives:

- **Improved Image Quality** using noise reduction and contrast enhancement techniques.
- **Accurate Feature Extraction** through edge detection and segmentation methods.
- **Efficient Object Analysis** using morphological operations.
- **Advanced Image Representation** with Fourier Transform for frequency domain analysis.

These techniques have wide-ranging applications in medical imaging, remote sensing, industrial automation, object detection, and artificial intelligence-based vision systems. The project establishes a strong foundation for further advancements in deep learning-based image analysis, real-time object recognition, and autonomous vision systems.

Moving forward, this pipeline can be enhanced by integrating adaptive thresholding, deep learning-based feature extraction, and real-time video processing, making it more applicable to modern-day computer vision and AI-driven technologies.