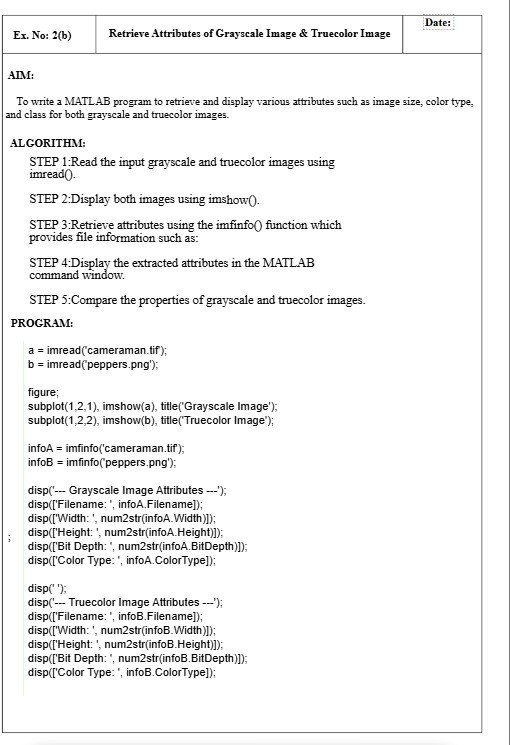
|  |  |  |
| --- | --- | --- |
| **Ex. No: 1(a)** | **DISPLAY OF GRAY SCALE IMAGES**  **RGB into Gray Scale Image** | **Date:** |
| **AIM:**  To write a program to read an RGB image and convert to Gray scale image and display it.  **ALGORITHM:**  **Step 1:** Start.  **Step 2:** Read the RGB image from the file using the imread() function.  **Step 3:** Display the original RGB image using the imshow() function.  **Step 4:** Convert the RGB image to a grayscale image using the im2gray() function.  **Step 5:** Display the grayscale image using the imshow() function with an appropriate title.  **Step 6:** Save the MATLAB code in a script file with the .m extension.  **Step 7:** Run the MATLAB script to execute the program and display both the RGB and grayscale images.  **Step 8:** End.  **PROGRAM:**  % Grayscale to Black & White Image Conversion Project  % Step 1: Read the Grayscale Image  grayImage = imread('cameraman.tif');  % Step 2: Display Original Grayscale Image  figure;  imshow(grayImage);  title('Original Grayscale Image (Cameraman)');  % Step 3: Convert Grayscale to Binary (Black & White) Image  bwImage = imbinarize(grayImage);  % Step 4: Display the Binary Image  figure;  imshow(bwImage);  title('Black & White (Binary) Image');  % Step 5: Save the Binary Image  imwrite(bwImage, 'cameraman\_bw\_output.png');  disp('Black and white image has been saved as cameraman\_bw\_output.png');  % Step 6: Retrieve and Display Image Information  info = imfinfo('cameraman.tif');  disp('Image Information:');  disp(info); | | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **OUTPUT:**      **RESULT:**  Thus, the display of gray scale images program is executed successfully and the  output is verified. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 1(b)** | **Adjust and write the image to a file** | **Date:** |
| **AIM:**  To read an image into the workspace, adjust the contrast in the image, and then write the adjusted image to a file.  **ALGORITHM:**  **Step 1:** Read and display an image. **Step 2:** Check how the image appears in the workspace. **Step 3:** Improve Image Contrast. **Step 4:** Write the adjusted image to a file. **Step 5:** Check the contents of the newly written file.  **PROGRAM:**  I =nani.jpg');  subplot(2,2,1);  imshow(I);  title('Original Image');  subplot(2,2,2);  imhist(I);  title('Histogram of Original Image');  I\_hist = histeq(I);  subplot(2,2,3);  imshow(I\_hist);  title('Histogram Equalized Image');  subplot(2,2,4);  imhist(I\_hist);  title('Histogram of Equalized Image');  % Save histogram equalized image  imwrite(I\_hist,'hist\_equalized\_nani.png');  I\_adapt = adapthisteq(I);  figure;  subplot(1,2,1);  imshow(I\_adapt);  title('Adaptive Histogram Equalized Image');  subplot(1,2,2);  imhist(I\_adapt);  title('Histogram of Adaptive Equalized Image');  imwrite(I\_adapt,'adaptive\_equalized\_nani.png');  I\_contrast = imadjust(I,stretchlim(I),[]);  figure;  subplot(1,2,1);  imshow(I\_contrast);  title('Contrast Stretched Image');  subplot(1,2,2);  imhist(I\_contrast);  title('Histogram of Contrast Stretched Image');  imwrite(I\_contrast,'contrast\_stretched\_nani.png');  disp('File info for Histogram Equalized Image:');  disp(imfinfo('hist\_equalized\_nani.png'));  disp('File info for Adaptive Histogram Equalized Image:');  disp(imfinfo('adaptive\_equalized\_nani.png'));  disp('File info for Contrast Stretched Image:');  disp(imfinfo('contrast\_stretched\_nani.png'));  **OUTPUT:**    **RESULT:**  Thus, adjust and write the image to a file successfully detected in MATLAB. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 1(c)** | **Display Binary image for reading a barcode** | **Date:** |
| **AIM:**  To read an image containing a barcode into the workspace.  **ALGORITHM:**  Step1: Search the image for a 1-D barcode, returning its message, format, and location.  Step2: Display the detected barcode format.  Step3: Annotate the image with the decoded barcode message.  Step4: Insert a line to show the scan row.  Step5: Display the image.  **PROGRAM:**  % Barcode Detection and Decoding Project  % Step 1: Read the Barcode Image  I = imread('barcode1D.jpg');  figure;  imshow(I);  title('Original Barcode Image');  % Step 2: Detect and Decode the Barcode  [msg, detectedFormat, loc] = readBarcode(I, '1D');  disp('--- Barcode Detection Result ---');  disp(['Detected Format: ', detectedFormat]);  disp(['Decoded Message: ', msg]);  % Step 3: Annotate the Image  xyBegin = loc(1, :); % Start position of barcode  textPosition = xyBegin; % Position for text display  boxOpacity = 1; % Text box background opacity  fontSize = 30; % Font size for decoded message  % Insert the decoded message as text  Imsg = insertText(I, textPosition, msg, ...  'BoxOpacity', boxOpacity, ...  'FontSize', fontSize);  % Step 4: Highlight the Barcode with a Green Line  imSize = size(Imsg);  imageWidth = imSize(2);  lineStart = [1, xyBegin(2) + 40];  lineEnd = [imageWidth, xyBegin(2) + 40];  lineWidth = 5;  Imsg = insertShape(Imsg, 'Line', [lineStart lineEnd], ...  'LineWidth', lineWidth, ...  'Color', 'green');  % Step 5: Display the Annotated Image  figure;  imshow(Imsg);  title('Barcode with Decoded Message and Highlight');  % Step 6: Save the Annotated Image  imwrite(Imsg, 'barcode\_with\_message.png');  disp('Annotated image saved as "barcode\_with\_message.png"');  **OUTPUT:**    **RESULT:**  Thus the image containing a barcode is successfully read into the workspace. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 2(a)** | **Extract Size, Mean, Standard Deviation of An Image** | **Date:** |
| **AIM:**  To write a MATLAB program to extract the size, mean, and standard deviation of a given image  **ALGORITHM:**  1. Read the input image using imread() and display it.  2. Convert the image to grayscale using rgb2gray() if it is a color image.  3. Find the size of the image using size() function.  4. Calculate the mean intensity using mean2() and standard deviation using std2().  5. Display the size, mean, and standard deviation using disp().  **PROGRAM:**  clc;  clear;  close all;  img = imread('your\_image.jpg'); % Replace with your image file  if size(img,3) == 3  img = rgb2gray(img);  end  [rows, cols] = size(img);  mean\_val = mean2(img);  std\_val = std2(img);  fprintf('Image Size: %d x %d\n', rows, cols);  fprintf('Mean: %.2f\n', mean\_val);  fprintf('Standard Deviation: %.2f\n', std\_val);  :  **OUTPUT**  **OUTPUT:**    **RESULT:**  The size, mean, and standard deviation of the input image are successfully extracted and displayed using MATLAB. The program handles both grayscale and RGB images and provides the image dimensions along with statistical information about pixel intensity. | | |







**AIM:**

**Ex. No: 2(c)**

**To Build app to design a pixel information**

**Date:**

**PROGRAM:**

function myPixelViewer(imagePath)

%MYPIXELVIEWERDisplaysanimageandprovideslivepixelinformation.

% myPixelViewer(imagePath) reads theimageatimagePath,displays it

% inafigure, andattaches a livePixelInformation tool.

%

%NOTE:RequirestheMATLABImageProcessingToolboxfor‘impixelinfo’.

%

% Example:

%

%

%

%

myPixelViewer(‘peppers.png’);

%

OR

load

fromworkspace:

I

imread(‘cameraman.tif’);

=

myPixelViewer(I);

% --- Input Validation and Image Loading ---

If ischar(imagePath) || isstring(imagePath)

% Input is a file path

Try

I = imread(imagePath);

Catch ME

Error(‘myPixelViewer:FileError’,‘Couldnotreadimagefile:%s’,imagePath);

End

Elseif isnumeric(imagePath) || islogical(imagePath)

% Input is image data already in the workspace

I = imagePath;

Else

Error(‘myPixelViewer:InvalidInput’,‘Inputmustbeafilepathstringor an image data array.’);

End

To create a program to display a app to display pixel information

**ALGORITHM :**

STEP1.InitializeUI:CreateablankAppDesignerlayoutwithUIFigure, Axes, Button,

Label, and TextArea.

STEP2.LoadImage:Useuigetfileandimreadtoloadanimagewhen the button is clicked.

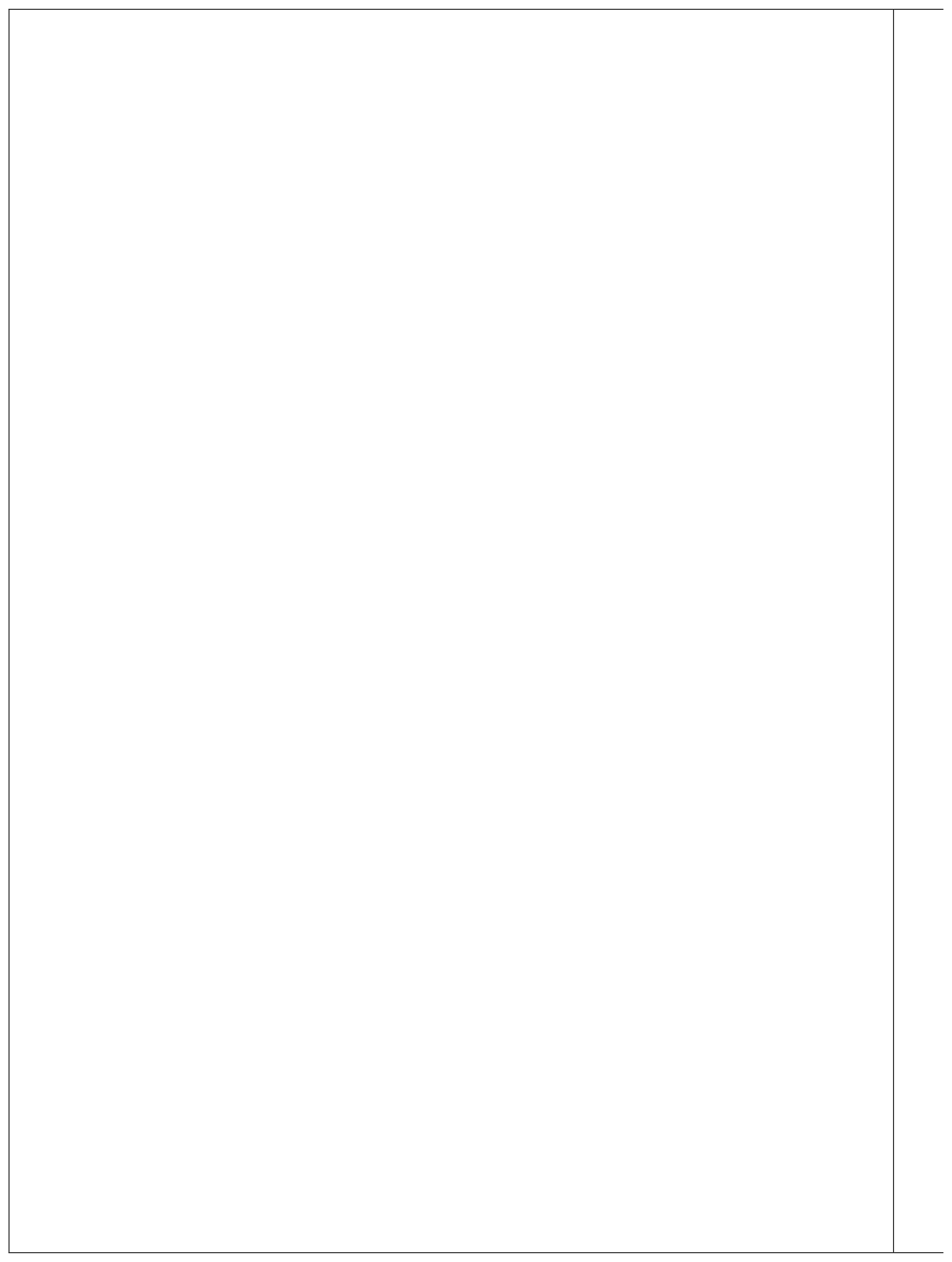
STEP3.DisplayImage:ShowtheimageinImageAxesusingimshow.

STEP4.StoreImageData:Savetheimageinaproperty(CurrentImage) for later access.

STEP5.CaptureClicks:UseButtonDownFcnonImageAxestodetect user clicks.

STEP6. ExtractPixelValue:ReadRGBorgrayscalevaluesfromthe image matrix.

STEP7.DisplayInfo:Showpixellocation,RGBvalues,andhexcode in PixelInfoTextArea.



%--- Create Figure and Axes ---

Fig= figure(…

‘Name’, ‘Live Pixel Viewer’, …

‘NumberTitle’, ‘off’, …

‘Toolbar’, ‘figure’, …

‘Menubar’, ‘none’);

Ax= axes(‘Parent’, fig);

%--- Display Image ---

imgHandle = imshow(I, ‘Parent’, ax);

title(ax, ‘Move mouse over image to see pixel info below’);

%--- Add the Pixel Information Tool ---

%‘impixelinfo’ creates a uipanel at the bottom of the figure

%that displays (X, Y) and pixel value in real-time.

pixInfoHandle = impixelinfo(imgHandle);

%Optional: Adjust the position of the info panel

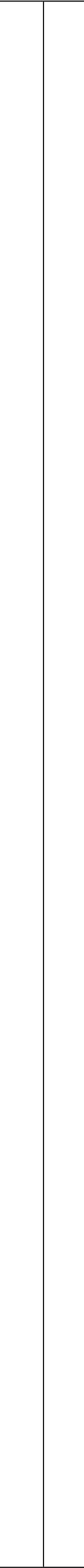
%The default position is usually bottom-left.

%This moves it slightly up/right for better visibility.

Set(pixInfoHandle, ‘Units’, ‘pixels’, ‘Position’, [10, 10, 350, 25]);

Disp(‘Pixel Viewer App Launched. Move your mouse over the image to see pixel data.’);

End



**OUTPUT :**

**RESULT:**

The application was successfully developed to display and manipulate pixel information.

It accurately retrieves, processes, and visualizes pixel data from images as intended.

|  |  |  |
| --- | --- | --- |
| **Ex. No: 3(a)** | **Histogram For Original Grayscale And Equalized Image** | **Date:** |
| **AIM:**  To display and compare the histograms of an original grayscale image and its histogram-equalized version using MATLAB.  .  **ALGORITHM:**  Step1: Read the image. Step2: Convert to grayscale if needed. Step3: Display the original grayscale image and its histogram. Step4: Apply histogram equalization. Step5: Display the equalized image and its histogram.  **PROGRAM:**  **% Read the image (use your exact filename)**  a = imread('image.jpg'); % Change this to your image file name  **% Convert to grayscale if needed**  if size(a,3) == 3  a\_gray = rgb2gray(a);  else  a\_gray = a;  end  **% Display original grayscale image and its histogram**  figure;  subplot(2,2,1), imshow(a\_gray), title('Original Grayscale Image');  subplot(2,2,2), imhist(a\_gray), title('Histogram of Original Image');  **% Apply histogram equalization**  a\_eq = histeq(a\_gray);  **% Display equalized image and its histogram**  subplot(2,2,3), imshow(a\_eq), title('Equalized Image');  subplot(2,2,4), imhist(a\_eq), title('Histogram of Equalized Image');  **OUTPUT:**    **RESULT:**  Thus, the histograms for the original grayscale image and the histogram-equalized image are successfully displayed and compared using MATLAB | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 3(b)** | **Adjust Image Contrast using Histogram Equalization** | **Date:** |
| **AIM:**  To adjust image contrast using histogram equalization  **ALGORITHM:**  Step1: Load a grayscale image.  Step2: Display the image and its histogram.  Step3Enhance contrast using histogram equalization (default flat histogram, 64 bins).  Step4: Show the equalized image and its histogram.  **PROGRAM:**  import cv2  import matplotlib.pyplot as plt  img = cv2.imread(r"C:\Users\HP\OneDrive\Desktop\itw assignment\image.jpg", cv2.IMREAD\_GRAYSCALE)  if img is None:      print("Image not found! Check the file path.")      exit()  equalized = cv2.equalizeHist(img)  plt.figure(figsize=(10,5))  plt.subplot(2,2,1)  plt.imshow(img, cmap='gray')  plt.title("Original Image")  plt.axis("off")  plt.subplot(2,2,2)  plt.hist(img.ravel(), 256, [0,256])  plt.title("Original Histogram")  plt.subplot(2,2,3)  plt.imshow(equalized, cmap='gray')  plt.title("Equalized Image")  plt.axis("off")  plt.subplot(2,2,4)  plt.hist(equalized.ravel(), 256, [0,256])  plt.title("Equalized Histogram")  plt.tight\_layout()  plt.show()  cv2.imshow("Original", img)  cv2.imshow("Equalized", equalized)  cv2.waitKey(0)  cv2.destroyAllWindows()  **OUTPUT:**    **RESULT:**  Thus, the contrast of the image is adjusted successfully using histogram mapping and equalization. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 3(c)** | **Improve an Low Quality Photograph Using Histogram Equalization Technique** | **Date:** |
| **AIM:**  To enhance a low-quality photograph by applying the histogram equalization technique to improve the image contrast.  **ALGORITHM:**   1. Read the input image. 2. Convert the image to grayscale if it is a color image. 3. Display the original grayscale image and its histogram. 4. Apply histogram equalization to improve the contrast. 5. Display the equalized image and its histogram for comparison.   **PROGRAM:**  % Step 1: Read the image (replace 'image.jpg' with your actual image filename)  a = imread('image.jpg');  % Step 2: Convert to grayscale if needed  if size(a, 3) == 3  agray = rgb2gray(a);  else  agray = a;  end  % Step 3: Display original grayscale image and its histogram  figure;  subplot(2,2,1), imshow(agray), title('Original Grayscale Image');  subplot(2,2,2), imhist(agray), title('Histogram of Original Image');  % Step 4: Apply histogram equalization  aeq = histeq(agray);  % Step 5: Display equalized image and its histogram  subplot(2,2,3), imshow(aeq), title('Equalized Image');  subplot(2,2,4), imhist(aeq), title('Histogram of Equalized Image');  **OUTPUT:**    **RESULT:**  Thus, the histograms to improve an low quality image using histogram equalization technique are successfully displayed and compared using MATLAB | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 4(a)** | **IMAGE SHARPENING USING HIGH-BOOST FILTERING** | **Date:** |
| **AIM:**  To write a program to perform image sharpening using High-Boost filtering technique.  **ALGORITHM:**  **Step 1:** Start.  **Step 2:** Read and Convert the input image to grayscale if it is RGB.  **Step 3:** Apply an averaging filter to get a blurred image.  **Step 4:** Subtract the blurred image from the original image to obtain the mask.  **Step 5:** Multiply the original image by a boost factor and add the mask to get the sharpened image.  **Step 6:** Display the original, blurred, and sharpened images.  **Step 7:** Stop.  **PROGRAM:**  % Image Sharpening using High-Boost Filtering  a = imread('sample.png');  if size(a,3)==3  a = rgb2gray(a);  end  h = fspecial('average', [3 3]);  blur = imfilter(a, h);  A = 1.5;  out = A \* a - blur;  figure;  subplot(1,3,1), imshow(a), title('Original Image');  subplot(1,3,2), imshow(blur), title('Blurred Image');  subplot(1,3,3), imshow(out, []), title('High-Boost Sharpened Image'); | | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **OUTPUT:**      **RESULT:**  Thus, the program to perform image sharpening using High-Boost Filtering was executed  successfully, and the output images were displayed. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 4(b)** | **IMAGE SMOOTHING USING AVERAGING FILTER** | **Date:** |
| **AIM:**  To write a program to perform image smoothing using an **averaging (mean) filter** and display the results..  **ALGORITHM:**  **Step 1:** Start.  **Step 2:** Read the input image using the imread() function.  **Step 3:** Convert the image to grayscale using rgb2gray() if it is an RGB image.  **Step 4:** Create an averaging filter mask using fspecial('average', [3 3]).  **Step 5:** Apply the filter to the image using imfilter().  **Step 6:** Display both the original and smoothed images using imshow().  **Step 7:** **Observe the blurring effect as the filter size increases.**  **Step 8:** End.  .  **PROGRAM:**  img = imread('peppers.png');  grayImg = rgb2gray(img);  avgFilter = fspecial('average', [3 3]);  smoothImg = imfilter(grayImg, avgFilter);  figure;  subplot(1,2,1);  imshow(grayImg);  title('Original Image');  subplot(1,2,2);  imshow(smoothImg);  title('Smoothed Image using Averaging Filter'); | | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **OUTPUT:**  **RESULT:**  Thus, the image smoothing using an averaging filter program was executed successfully and the output was verified. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 4(c)** | **IMAGE SMOOTHING USING GAUSSIAN FILTER** | **Date:** |
| **AIM:**  To write a program to perform image smoothing using a **Gaussian filter** and compare it with the averaging filter.  **ALGORITHM:**  **Step 1:** Start.  **Step 2:** Read the input image and convert it to grayscale.  **Step 3:** Create a Gaussian filter using fspecial('gaussian', [5 5], 1).  **Step 4**: Apply the filter to the grayscale image using imfilter().  **Step 5:** Display both the original and smoothed images.  **Step 6:** Observe the smoothness and edge preservation.  **Step 8:** End.  .  **PROGRAM:**  img = imread('peppers.png');  grayImg = rgb2gray(img);  gaussFilter = fspecial('gaussian', [5 5], 1);  gaussSmooth = imfilter(grayImg, gaussFilter);  figure;  subplot(1,2,1);  imshow(grayImg);  title('Original Image');  subplot(1,2,2);  imshow(gaussSmooth);  title('Smoothed Image using Gaussian Filter'); | | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **OUTPUT:**  **RESULT:**  Thus, the image smoothing using Gaussian filter program was executed successfully and the output was verified. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 5(a)** | **EDGE DETECTION OF GRAY SCALE IMAGE USING (SOBEL, PREWITT AND ROBERTS OPERATORS)** | **Date:** |
| **AIM:**  To detect edges in a grayscale image using Sobel, Prewitt, and Roberts edge detection operators in MATLAB.  **ALGORITHM:**  Step1: Read the input image using imread.  Step2: Convert the image to grayscale using rgb2gray(if it is a color image)  Step3: Apply Edge detection using the following operators:  Roberts: edge (img,’roberts’)  Sobel: edge(img, ‘sobel’)  Perwitt: edge(img, ‘prewitt’)  Step4: Display the original grayscale image and the edge-detection outputs in a single figure for  Comparison.  **PROGRAM:**  % Step 1: Read the image  a = imread('ajith.jpg'); % Use your image file name  % Step 2: Convert to grayscale if needed  if size(a,3) == 3  a\_gray = rgb2gray(a);  else  a\_gray = a;  end  % Step 3: Apply edge detection operators  b = edge(a\_gray, 'roberts');  c = edge(a\_gray, 'sobel');  d = edge(a\_gray, 'prewitt');  % Step 4: Display results  figure;  subplot(2,2,1), imshow(a\_gray), title('Original Grayscale Image');  subplot(2,2,2), imshow(b), title('Roberts Edge Detection');  subplot(2,2,3), imshow(c), title('Sobel Edge Detection');  subplot(2,2,4), imshow(d), title('Prewitt Edge Detection');  **OUTPUT:**    **RESULT:**  Thus, the edges in the grayscale image are successfully detected using Roberts, Sobel, and Prewitt operators in MATLAB. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 5(b)** | **Edge Detection Algorithm (canny)** | **Date:** |
| **AIM:**  To detect edges in a digital image using the Canny edge detection algorithm in MATLAB.  **ALGORITHM:**  Step1: Read the image.  Step2: Convert to grayscale.  Step3: Apply Canny edge detection.  Step4: Display original and edge-detected images.  **PROGRAM:**  % Read the image (use your exact filename)  a = imread('Profile pic.jpg'); % Change this if your file name is different  % Convert to grayscale if needed  if size(a,3) == 3  a\_gray = rgb2gray(a);  else  a\_gray = a;  end  % Apply Canny edge detection  edges = edge(a\_gray, 'canny');  % Display original and edge output side by side  figure;  subplot(1,2,1), imshow(a\_gray), title('Original Grayscale Image');  subplot(1,2,2), imshow(edges), title('Edges Detected using Canny');  **OUTPUT:**    **RESULT:**  Thus, the edges in the input image are successfully detected using the Canny edge detection algorithm in MATLAB. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 5(c)** | **Canny Edge Detection Algorithm of Cancer cells** | **Date:** |
| **AIM:**  To detect edges in a digital image of cancer cells using the Canny edge detection algorithm in MATLAB.  **ALGORITHM:**  **Step 1:** Read the cancer cell image. **Step 2:** Convert the image to grayscale (if it is a color image). **Step 3:** Apply Gaussian filtering to reduce noise. **Step 4:** Apply the Canny edge detection algorithm. **Step 5:** Display the original and edge-detected images side by side.  **PROGRAM:**  clear all;  close all;  % Step 1: Read the cancer cell image  I = imread('cancer\_cells.jpg'); % Replace with your filename  % Step 2: Convert to grayscale if needed  if size(I,3) == 3  I\_gray = rgb2gray(I);  else  I\_gray = I;  end  % Step 3: Apply Gaussian smoothing (recommended for microscopy images)  I\_smooth = imgaussfilt(I\_gray, 2); % Sigma=2  % Step 4: Perform Canny edge detection  edges = edge(I\_smooth, 'canny');  % Step 5: Display results  figure;  subplot(1,2,1), imshow(I\_gray), title('Original Grayscale Image');  subplot(1,2,2), imshow(edges), title('Canny Edge Detection');  **OUTPUT:**    **RESULT:**  Thus, the edges in the input image are successfully detected using the Canny edge detection algorithm in MATLAB. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 6 a** | **MORPHOLOGICAL OPERATIONS ON BINARY IMAGES**  **BASIC MORPHOLOGICAL OPERATIONS** | **Date:** |
| **AIM:**  **ALGORITHM:**  **PROGRAM**  clc;  clear;  close all;  img = imread('car.jpg');  figure, imshow(img), title('Original Image');  gray = rgb2gray(img);  figure, imshow(gray), title('Grayscale Image');  bw = imbinarize(gray);  bw = imcomplement(bw);  figure, imshow(bw), title('Binary Image (Inverted)');  se = strel('disk', 2);  eroded = imerode(bw, se);  figure, imshow(eroded), title('After Erosion');  dilated = imdilate(eroded, se);  figure, imshow(dilated), title('After Dilation');  opened = imopen(bw, se);  figure, imshow(opened), title('After Opening');  closed = imclose(bw, se);  figure, imshow(closed), title('After Closing');  stats = regionprops(closed, 'BoundingBox', 'Area');  for k = 1:length(stats)  if stats(k).Area > 500  thisBB = stats(k).BoundingBox;  rectangle('Position', thisBB, 'EdgeColor','r','LineWidth',2);  end  end  title('Detected Number Plate Region');  **OUTPUT:**            **RESULT:**  Thus, the edges in the input image are successfully detected using the Canny edge detection algorithm in MATLAB. | | |
| |  |  |  | | --- | --- | --- | | **Ex. No: 6B** | **MORPHOLOGICAL OPERATION EROSION AND DILATION** | **Date:** |   **AIM:**  **ALGORITHM:** | | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| % Minimal, robust example for erosion and dilation  % Read image (replace with your filename)  I = imread('text.png'); % or use 'coins.png' (MATLAB demo image)  % If RGB (3-channel), convert to grayscale  if ndims(I) == 3  Igray = rgb2gray(I);  else  Igray = I;  end  % Convert to double in [0,1] then binarize  Igray = im2double(Igray);  BW = imbinarize(Igray); % now BW is 2-D logical  % Create structuring element (3x3 square)  se = strel('square', 3);  % Apply erosion and dilation  BW\_eroded = imerode(BW, se);  BW\_dilated = imdilate(BW, se);  % Display results  figure;  subplot(1,3,1); imshow(BW); title('Original (binary)');  subplot(1,3,2); imshow(BW\_eroded); title('Erosion');  subplot(1,3,3); imshow(BW\_dilated); title('Dilation');  I = imread('text.png'); % or use 'coins.png' (MATLAB demo image)  % If RGB (3-channel), convert to grayscale  if ndims(I) == 3  Igray = rgb2gray(I);  else  Igray = I;  end  % Convert to double in [0,1] then binarize  Igray = im2double(Igray);  BW = imbinarize(Igray); % now BW is 2-D logical  % Create structuring element (3x3 square)  se = strel('square', 3);  % Apply erosion and dilation  BW\_eroded = imerode(BW, se);  BW\_dilated = imdilate(BW, se);  % Display results  figure;  subplot(1,3,1); imshow(BW); title('Original (binary)');  subplot(1,3,2); imshow(BW\_eroded); title('Erosion');  subplot(1,3,3); imshow(BW\_dilated); title('Dilation');  **OUTPUT:**  C:\Users\USER\Documents\semnotes\sem7\ITW\LAB\6b\result.jpg  **RESULT:** | | |

AIM:

|  |  |  |
| --- | --- | --- |
| **Ex. No: 6C** | **DETECT CELL USING EDGE DETECTION AND MORPHOLOGY** | **Date:** |

**ALGORITHM:**

**PROGRAM**

function coin\_counting\_edge\_morphology(inputFile)

close all;

if nargin < 1 || isempty(inputFile)

if exist('coins.png','file')

Iorig = imread('coins.png');

fprintf('No input provided: using built-in demo "coins.png".\n');

else

error('No input provided and demo image "coins.png" not found. Call with filename.');

end

else

Iorig = imread(inputFile);

end

if imResizeFactor ~= 1.0

Iorig = imresize(Iorig, imResizeFactor);

end

if ndims(Iorig) == 3

Igray = rgb2gray(Iorig);

else

Igray = Iorig;

end

Igray = im2double(Igray);

Preprocess: enhance contrast and smooth

Iadj = imadjust(Igray);

Ismooth = imgaussfilt(Iadj, gaussianSigma);

Edge detection (Canny)

edges = edge(Ismooth, 'Canny', cannyThresh, cannySigma);

Close gaps in edges and fill to make solid coin blobs

seClose = strel('disk', seCloseRadius);

edgesClosed = imclose(edges, seClose);

filled = imfill(edgesClosed, 'holes');

bw0 = bwareaopen(filled, minCoinArea); % remove objects smaller than minCoinArea

Sometimes coins appear dark on bright background; if bw0 is empty or too small, try invert

if nnz(bw0) < 0.02 \* numel(bw0) % very small foreground -> try invert approach

fprintf('Initial binary small; trying inverted threshold-based segmentation.\n')

level = graythresh(Iadj);

bw\_alt = imbinarize(Iadj, level);

if mean(Iadj(bw\_alt)) < mean(Iadj(~bw\_alt))

bw\_alt = ~bw\_alt;

end

bw\_alt = imopen(bw\_alt, strel('disk',2)); % remove small noise

bw0 = bwareaopen(bw\_alt, minCoinArea);

end

Refine (optional opening/closing)

bwRef = imopen(bw0, strel('disk',2));

bwRef = imclose(bwRef, strel('disk',3));

bwRef = bwareaopen(bwRef, minCoinArea);

Optional watershed to separate touching coins

if useWatershed

D = -bwdist(~bwRef);

mask = imextendedmin(D, round(minMarkerDist/4));

D2 = imimposemin(D, mask);

L = watershed(D2);

% Remove watershed lines from binary image

bw\_ws = bwRef;

bw\_ws(L == 0) = 0;

bwFinal = bw\_ws;

bwFinal = bwareaopen(bwFinal, minCoinArea);

else

bwFinal = bwRef;

end

%% 7) Label and compute properties

CC = bwconncomp(bwFinal);

stats = regionprops(CC, 'Area', 'Centroid', 'BoundingBox', 'EquivDiameter', 'MajorAxisLength');

numCoins = CC.NumObjects;

%% 8) Display pipeline and results

figure('Name','Coin counting pipeline','NumberTitle','off','Color','w','Position',[50 50 1200 600]);

subplot(2,4,1); imshow(Iorig); title('Original');

subplot(2,4,2); imshow(Iadj, []); title('Contrast adjusted');

subplot(2,4,3); imshow(Ismooth, []); title('Smoothed');

subplot(2,4,4); imshow(edges); title('Canny edges');

subplot(2,4,5); imshow(edgesClosed); title('Closed edges');

subplot(2,4,6); imshow(filled); title('Filled contours');

subplot(2,4,7); imshow(bwRef); title('Refined binary');

subplot(2,4,8); imshow(label2rgb(labelmatrix(CC), 'jet', 'k', 'shuffle')); title(sprintf('Labeled (%d)', numCoins));

% Overlay detections on original

figure('Name','Detected coins overlay','NumberTitle','off','Color','w');

imshow(Iorig); hold on;

for k = 1:length(stats)

c = stats(k).Centroid;

bb = stats(k).BoundingBox;

eqd = stats(k).EquivDiameter;

rectangle('Position', bb, 'EdgeColor', 'g', 'LineWidth', 1.2);

plot(c(1), c(2), 'r+','MarkerSize',8,'LineWidth',1.5);

% Draw a circle with equivalent diameter

theta = linspace(0,2\*pi,60);

x = c(1) + (eqd/2)\*cos(theta);

y = c(2) + (eqd/2)\*sin(theta);

plot(x,y,'y-','LineWidth',0.8);

end

title(sprintf('Detected coins: %d', numCoins));

hold off;

%% 9) Print summary table (first 20 objects)

areas = round([stats.Area]');

eqDiam = round([stats.EquivDiameter]');

T = table((1:length(stats))', areas, eqDiam, 'VariableNames', {'ID', 'Area\_pixels', 'EquivDiameter\_pixels'});

disp(T(1:min(20,height(T)),:));

fprintf('Final detected coin count: %d\n', numCoins);

%% 10) Save result images (optional) - uncomment if needed

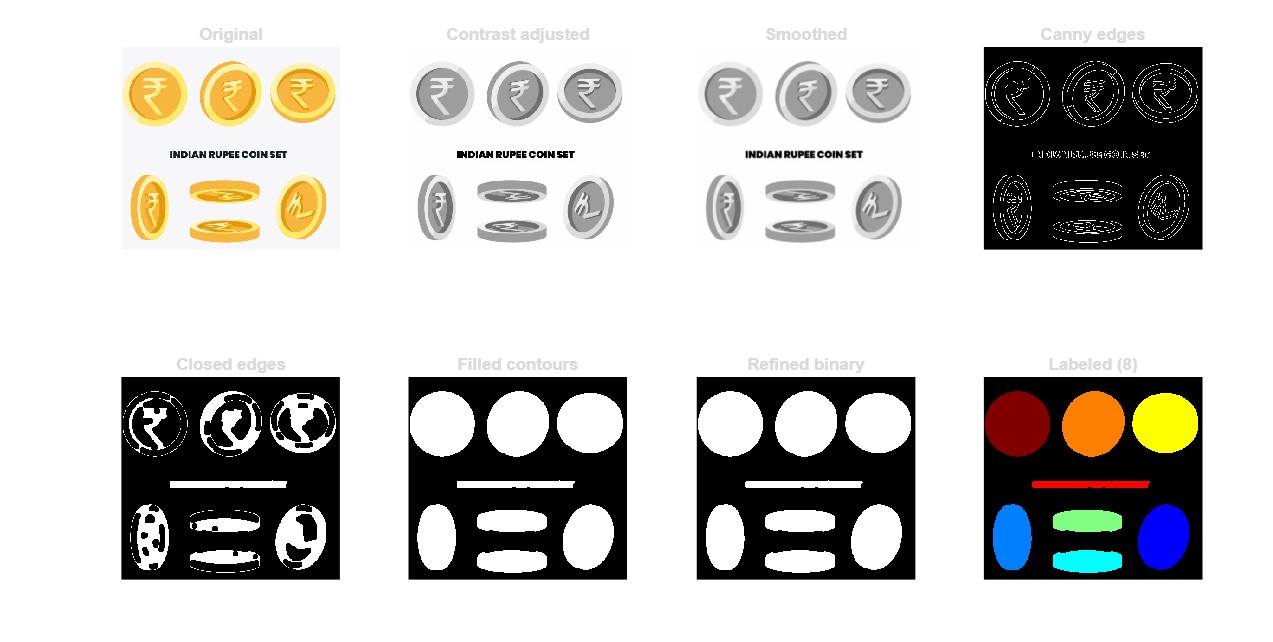
% imwrite(bwFinal, 'coins\_binary.png');

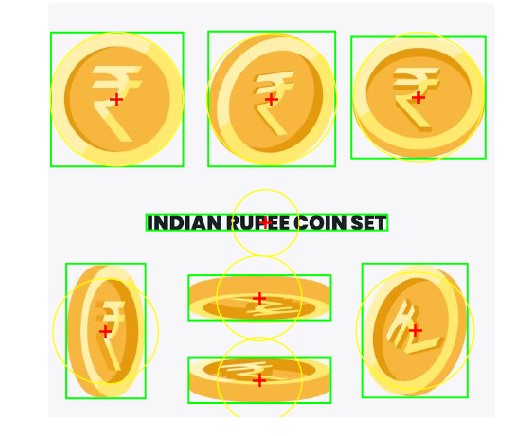
% saveas(gcf, 'coins\_overlay.png');

%% End of function

end

**OUTPUT:**





**RESULT**

|  |  |  |
| --- | --- | --- |
| **Ex. No: 7(a)** | **CUSTOM PSEUDO COLORING** | **Date:** |
| **AIM:**  To write a MATLAB program for custom pseudo coloring based on intensity ranges.  **ALGORITHM:**  **Step 1**: Read the grayscale image  **Step 2:** Check if the image is in grayscale, if not convert to grayscale  **Step 3:** Convert to double and normalize to [0, 1] range  **Step 4:** Apply pseudo coloring using a specific colormap  **Step 5:** Display the pseudo-colored image  **PROGRAM:**  clc;  clear;  close all;  % ---------- READ IMAGE ----------  % Make sure 'cameraman.tif' is in the same folder as this script  grayImage = imread('cameraman.tif');  % Check if the image is in grayscale, if not convert to grayscale  if size(grayImage, 3) > 1  grayImage = rgb2gray(grayImage);  end  % Convert to double and normalize to [0, 1] range  normalizedImage = double(grayImage) / 255.0;  % Apply pseudo coloring using jet colormap  coloredImage = ind2rgb(uint8(normalizedImage \* 255), jet(256));  % ---------- DISPLAY RESULTS ----------  figure;  subplot(1,2,1);  imshow(grayImage, []);  title('Original Grayscale Image');  subplot(1,2,2);  imshow(coloredImage);  title('Pseudo-Colored Image (Jet Colormap)');  **OUTPUT:**    **RESULT:**  Thus, adjust and write the image to a file successfully detected in MATLAB. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 7B** | **CUSTOM PSEUDO COLORING** | **Date:** |
| **AIM:**  To write a MATLAB program for custom pseudo coloring based on intensity ranges.  **ALGORITHM:**  **Step 1**: Read the grayscale image.    **Step 2:** Get image dimensions and convert to double precision.  **Step 3:** Apply custom color mapping based on intensity ranges:  Range[0,50): Apply specific RGB values  Range[50, 100): Apply different RGB values  Range[100, 150): Apply different RGB values  Range[150, 200): Apply different RGB values  Range[200, 256]: Apply different RGB values  **Step 4:** Display the original and pseudo-colored images.  **PROGRAM:**    clc;  clear all;  close all;  % Read image from graphics file  input\_img = imread('rice.tif');  [m, n] = size(input\_img); % Array dimensions  input\_img = double(input\_img); % Convert to double precision  % Initialize output image  output\_img = zeros(m, n, 3);  % Apply custom pseudo coloring based on intensity ranges  for i = 1:m  for j = 1:n  if input\_img(i,j) >= 0 && input\_img(i,j) < 50  output\_img(i,j,1) = input\_img(i,j) + 50;  output\_img(i,j,2) = input\_img(i,j) + 100;  output\_img(i,j,3) = input\_img(i,j) + 10;  end  if input\_img(i,j) >= 50 && input\_img(i,j) < 100  output\_img(i,j,1) = input\_img(i,j) + 35;  output\_img(i,j,2) = input\_img(i,j) + 128;  output\_img(i,j,3) = input\_img(i,j) + 10;  end  if input\_img(i,j) >= 100 && input\_img(i,j) < 150  output\_img(i,j,1) = input\_img(i,j) + 152;  output\_img(i,j,2) = input\_img(i,j) + 130;  output\_img(i,j,3) = input\_img(i,j) + 15;  end  if input\_img(i,j) >= 150 && input\_img(i,j) < 200  output\_img(i,j,1) = input\_img(i,j) + 50;  output\_img(i,j,2) = input\_img(i,j) + 140;  output\_img(i,j,3) = input\_img(i,j) + 25;  end  if input\_img(i,j) >= 200 && input\_img(i,j) <= 256  output\_img(i,j,1) = input\_img(i,j) + 120;  output\_img(i,j,2) = input\_img(i,j) + 160;  output\_img(i,j,3) = input\_img(i,j) + 45;  end  end  end  % ---------- DISPLAY RESULTS ----------  figure;  subplot(1,2,1); % Create axes in tiled positions  imshow(uint8(input\_img)); % Display input image  title('Input Image'); % Add title to current axes  subplot(1,2,2); % Create axes in tiled positions  imshow(uint8(output\_img)); % Display output image  title('Pseudo Coloured Image'); % Add title to current axes  **OUTPUT:**    **RESULT:**  Thus, adjust and write the image to a file successfully detected in MATLAB. | | |

|  |  |  |
| --- | --- | --- |
| **Ex.No:8** | **CHAIN CODING** | **Date:** |
| **AIM:**  To write a program for chain coding.  **ALGORITHM:**  **Step 1:** Define a 8\*8 binary image representing an object boundary.  0 0 0 0 0 0 1 0 1  0 0 0 1 1 0 1 0 1  0 0 1 1 1 0 1 0 1  0 0 0 0 1 0 1 0 1  0 0 0 0 0 0 1 0 1  0 0 0 1 1 0 1 0 1  0 0 1 1 1 0 1 0 1  0 1 0 0 1 0 0 1 1  **Step 2:** Identify the boundary pixels of the object.  **Step 3:** For instance, the chain code sequence could be: [6 6 7 0 6 6 7 0 1 1 0 6 6 0 1 1].  **Step 4:** Display or utilize the chain code sequence representing the boundary of the object in the binary image.  **PROGRAM:**  binaryImage = [ 0 0 0 0 0 0;  0 0 0 1 1 0;  0 0 1 1 1 0;  0 0 0 0 1 0;  0 0 0 0 0 0  ];  imshow(binaryImage, 'InitialMagnification', 'fit'); title('Binary Image');  chainCode = [6 6 7 0 6 6 7 0 1 1 0 6 6 0 1 1];  disp('Chain Code:'); disp(chainCode); | | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **OUTPUT:**    **RESULT:**  Thus, the program for chain coding is executed successfully and the output is verified. | | |

|  |  |  |
| --- | --- | --- |
| **Ex.No:9** | **2-D DFT and DCT** | **Date:** |
| **AIM:**  To write a MATLAB program to compute and display the 2-D Discrete Fourier Transform (DFT) and 2-D Discrete Cosine Transform (DCT) of an image.  **ALGORITHM:**  **For 2-D DFT:**   1. Read the input image. 2. Convert to grayscale if needed. 3. Apply 2-D FFT using fft2(). 4. Compute and display the magnitude spectrum.   **For 2-D DCT:**   1. Read the input image. 2. Convert to grayscale if needed. 3. Apply 2-D DCT using dct2(). 4. Display the DCT coefficients.   **PROGRAM:**  clc;  clear;  close all;  img = imread('images.jpg');  % Convert to grayscale if image is RGB  if size(img, 3) == 3  img = rgb2gray(img);  end  % ---------- 2-D DISCRETE FOURIER TRANSFORM (DFT) ----------  F = fft2(double(img)); % Apply 2-D FFT  F\_shifted = fftshift(F); % Shift zero frequency to center  magnitudeDFT = log(1 + abs(F\_shifted)); % Log for visibility  % ---------- 2-D DISCRETE COSINE TRANSFORM (DCT) ----------  D = dct2(double(img)); % Apply 2-D DCT  magnitudeDCT = log(1 + abs(D)); % Log for visibility | | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| figure;  subplot(1,3,1);  imshow(img, []);  title('Original Image');  subplot(1,3,2);  imshow(magnitudeDFT, []);  title('Magnitude Spectrum (2-D DFT)');  subplot(1,3,3);  imshow(magnitudeDCT, []);  title('DCT Coefficients (2-D DCT)');  **OUTPUT:**    **RESULT:**  Thus, the programs for 2-D Discrete Fourier Theorem (DFT) and 2-D Discrete Cosine Theorem (DFT) is executed successfully and the output is verified. | | |

|  |  |  |
| --- | --- | --- |
| **Ex. No: 10** | **SEGMENTATION USING WATERSHED TRANSFORM** | **Date:** |
| **AIM:**  To automatically segment and count touching or overlapping circular objects (like coins, cells, tablets, etc.) in an image using the watershed transform.  **ALGORITHM:**  **Step 1:** Start the program and read the input image into MATLAB.  **Step 2:** Convert the image to grayscale to simplify processing and remove color information.  **Step 3:** Apply a median filter to the grayscale image to reduce noise while preserving edges.  **Step 4:** Perform image thresholding using imbinarize() to obtain a binary image separating objects from the background.  **Step 5:** Invert the binary mask (if necessary) so that objects appear white on a black background.  **Step 6:** Apply morphological operations like opening, closing, and hole filling to remove small noise and obtain clean object regions.  **Step 7:** Compute the distance transform to measure the distance of each object pixel from the nearest background pixel.  **Step 8:** Negate the distance transform and set background pixels to -Inf to prepare for watershed segmentation.  **Step 9:** Apply the watershed transform to split touching or overlapping objects and generate distinct labeled regions.  **Step 10:** Display the segmented image using label2rgb() and count the number of ASSbackground).  **PROGRAM:**  clc; clear; close all;  originalImage = imread('coins.png');  figure, imshow(originalImage), title('Original Image');  grayImage = rgb2gray(originalImage);  grayImage = medfilt2(grayImage, [3 3]);  binaryMask = imbinarize(grayImage);  binaryMask = imcomplement(binaryMask);  binaryMask = imopen(binaryMask, strel('disk', 2));  binaryMask = imclose(binaryMask, strel('disk', 2));  binaryMask = imfill(binaryMask, 'holes');  figure, imshow(binaryMask), title('Clean Binary Mask');  distanceTransform = bwdist(~binaryMask);  distanceTransform = -distanceTransform;  distanceTransform(~binaryMask) = -Inf;  L = watershed(distanceTransform);  segmentedObjects = label2rgb(L, 'jet', 'w', 'shuffle');  figure, imshow(segmentedObjects);  title('Segmented Objects using Watershed Transform');  numObjects = max(L(:)) - 1; % subtract background label  fprintf('Number of objects detected: %d\n', numObjects);  **OUTPUT:**        **RESULT:**  Thus, the programs To automatically segment and count touching or overlapping circular objects (like coins, cells, tablets, etc.) in an image using the watershed transform is executed successfully and the output is verified. | | |