Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of

Electrical & Computer Engineering

Lab Report 1

Basics of Digital Image Processing

Course Code: ECE 4224

Course Title: Digital Image Processing Sessional

Submitted to:	**Submitted by:**
Oishi Jyoti	Md. Tajim An Noor
Assistant Professor	Roll: 2010025
Dept of ECE, RUET	Semester: 4th Year Odd

Submission Date: January 20, 2026

Contents

Basics of Digital Image Processing

1 Theory and Introduction

Digital Image Processing (DIP) involves computational techniques to enhance, analyze, and manipulate digital images, represented as 2D arrays of pixel intensities [1]. It serves as a bridge between classical signal processing and computer vision, facilitating applications ranging from medical imaging to autonomous systems.

Key fundamental concepts explored in this experiment include:

- **Enhancement:** Improving visual quality via contrast stretching and histogram equalization.
- **Restoration:** Removing noise and artifacts.
- **Segmentation:** Partitioning images into meaningful regions.
- **Feature Extraction:** Identifying edges and characteristics for analysis.

In this experiment, we implement basic image operations (resizing, color conversion, binary thresholding) and advanced manipulations (geometric transforms, channel isolation) using both MATLAB and Python environments [2].

2 Methodology

2.1 Basic Operations and Manipulations

In this section, we implement fundamental image processing techniques including reading, resizing, converting formats, and geometric manipulations using both MATLAB and Python.

2.1.1 MATLAB Part 1: Basic Operations

We began by utilizing the MATLAB Image Processing Toolbox to read the standard "Lenna.png" image. The image was resized to a lower resolution (50×50) and converted into grayscale (luminance) and binary (black and white) formats to understand matrix representations of different image modes.

```matlab
clc; clear all; close all;

% 1. Read Image
a = imread('Lenna.png');
figure;
imshow(a);
title('Original Image');

% 2. Resize
% Resize to 50x50 pixels
b = imresize(a, [50, 50]);
figure;
imshow(b);
title('Resized Image');

% 3. Image Info
info = imfinfo('Lenna.png');
disp(info);

% 4. Convert to Grayscale
c = rgb2gray(a);
figure;
imshow(c);
title('Grayscale');

% 5. Convert to Binary
% Note: im2bw is older; imbinarize is recommended for newer MATLAB versions
d = im2bw(a);
figure;
imshow(d);
title('Binary Image');
```

2.1.2 MATLAB Part 2: Image Manipulation

We manipulated the spatial and color properties of the image. This involved separating color channels (isolating blue), merging distinct image types (grayscale and binary side-by-side), and applying geometric transformations such as horizontal flipping and 90-degree clockwise rotation.

```matlab
%% 1. Merge Image
% Concatenate Grayscale (c) and Binary (d)
% We multiply logical 'd' by 255 and convert to uint8 to match 'c'
m = horzcat(c, uint8(d) * 255);
figure;
imshow(m);
title('Merged Image (Grayscale + Binary)');

% 2. Channel Manipulation
% Create a copy to avoid modifying original 'a'
a_mod = a;
a_mod(:, :, 1) = 0; % Set Red channel to 0
a_mod(:, :, 2) = 0; % Set Green channel to 0
figure;
imshow(a_mod);
title('Blue Channel Only');

% 3. Flip and Rotate
o = flip(a, 2);      % Horizontal flip
p = imrotate(a, 90); % 90 degree clockwise rotation
figure; imshow(o); title('Flipped Horizontally');
figure; imshow(p); title('Rotated 90 Degrees');

% 4. Change Color Intensity
red_factor = 0.2;
green_factor = 1.8;
blue_factor = 0.5;

% Scale individual channels (using 'a' as source)
red_channel   = uint8(a(:, :, 1) * red_factor);
```

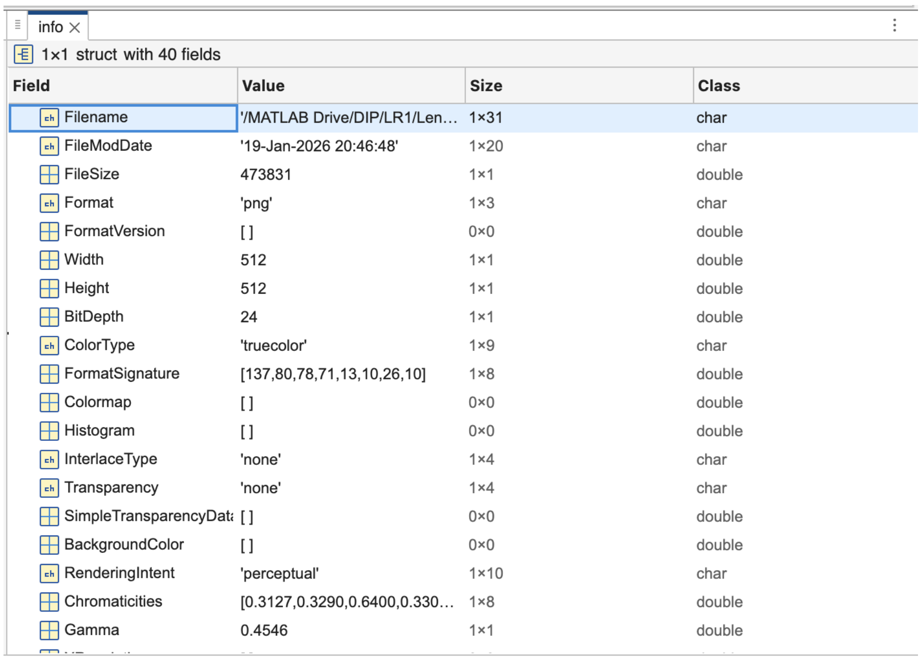
```
green_channel = uint8(a(:, :, 2) * green_factor);
blue_channel  = uint8(a(:, :, 3) * blue_factor);

% Recombine channels
modified_img = cat(3, red_channel, green_channel, blue_channel);

figure;
imshow(modified_img);
title('Color Intensity Modified');

%% --- SUMMARY DISPLAY (All 10 Results) ---
figure('Name', 'Basic Operations Complete Summary', 'NumberTitle', 'off', 'Position', [50, 50,
↪  1400, 600]);

% Row 1: Basic Conversions
subplot(2, 5, 1); imshow(a); title('1. Original');
subplot(2, 5, 2); imshow(b); title('2. Resized (50x50)');
subplot(2, 5, 3); imshow(c); title('3. Grayscale');
subplot(2, 5, 4); imshow(d); title('4. Binary');
subplot(2, 5, 5); imshow(m); title('5. Merged (Gray+Bin)');

% Row 2: Manipulations
subplot(2, 5, 6); imshow(a_mod); title('6. Blue Channel Only');
subplot(2, 5, 7); imshow(n); title('7. Flip Vertical');
subplot(2, 5, 8); imshow(o); title('8. Flip Horizontal');
subplot(2, 5, 9); imshow(p); title('9. Rotated 90\circ');
subplot(2, 5, 10); imshow(modified_img); title('10. Color Modified');
```

2.1.3 Python Implementation

Equivalent operations were performed in Python using the `OpenCV` and `NumPy` libraries. This validates the results across different computing environments, utilizing `cv2.threshold` for segmentation and array slicing for channel manipulation.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

plt.close("all")

# --- PART 1: Basic Operations ---
# 1. Read Image
a_bgr = cv2.imread("./images/Lenna.png")
if a_bgr is None:
    raise FileNotFoundError(
        "Image not found. Please ensure
        ↪  'Lenna.png' is in the working
        ↪  directory."
    )

a = cv2.cvtColor(a_bgr, cv2.COLOR_BGR2RGB)

# 2. Resize # Resize to 50x50 pixels
b = cv2.resize(a, (50, 50))

# 3. Image Info
print("Image Info:")
print(f"Shape: {a.shape}")
print(f"Data Type: {a.dtype}")

# 4. Convert to Grayscale
c = cv2.cvtColor(a, cv2.COLOR_RGB2GRAY)

# 5. Convert to Binary
_, d = cv2.threshold(c, 127, 255,
↪  cv2.THRESH_BINARY)

# --- PART 2: Manipulations ---
# 1. Merge Image (Grayscale + Binary)
m = np.hstack((c, d))

# 2. Channel Manipulation (Blue Channel Only)
a_mod = a.copy()
a_mod[:, :, 0] = 0  # Set Red channel to 0
a_mod[:, :, 1] = 0  # Set Green channel to 0

# 3. Flip and Rotate
n = cv2.flip(a, 0)  # Vertical
o = cv2.flip(a, 1)  # Horizontal
p = cv2.rotate(a, cv2.ROTATE_90_CLOCKWISE)

# 4. Change Color Intensity
red_factor = 0.2
green_factor = 1.8
```

```python
blue_factor = 0.5

# Scale individual channels
red_channel = np.clip(a[:, :, 0] * red_factor, 0, 255).astype(np.uint8)
green_channel = np.clip(a[:, :, 1] * green_factor, 0, 255).astype(np.uint8)
blue_channel = np.clip(a[:, :, 2] * blue_factor, 0, 255).astype(np.uint8)

# Recombine channels
modified_img = cv2.merge((red_channel, green_channel, blue_channel))

# --- SUMMARY DISPLAY (All 10 Results) ---
fig, axes = plt.subplots(2, 5, figsize=(20, 8))
fig.suptitle("Basic Operations Complete Summary", fontsize=16)

# Row 1: Basic Conversions
# 1. Original
axes[0, 0].imshow(a)
axes[0, 0].set_title("1. Original")
axes[0, 0].axis("off")
# 2. Resized
axes[0, 1].imshow(b)
axes[0, 1].set_title("2. Resized (50x50)")
axes[0, 1].axis("off")
# 3. Grayscale (needs cmap='gray')
axes[0, 2].imshow(c, cmap="gray")
axes[0, 2].set_title("3. Grayscale")
axes[0, 2].axis("off")
# 4. Binary (needs cmap='gray')
axes[0, 3].imshow(d, cmap="gray")
axes[0, 3].set_title("4. Binary")
axes[0, 3].axis("off")
# 5. Merged (needs cmap='gray')
axes[0, 4].imshow(m, cmap="gray")
axes[0, 4].set_title("5. Merged (Gray+Bin)")
axes[0, 4].axis("off")

# Row 2: Manipulations
# 6. Blue Channel Only
axes[1, 0].imshow(a_mod)
axes[1, 0].set_title("6. Blue Channel Only")
axes[1, 0].axis("off")
# 7. Flip Vertical
axes[1, 1].imshow(n)
axes[1, 1].set_title("7. Flip Vertical")
axes[1, 1].axis("off")
# 8. Flip Horizontal
axes[1, 2].imshow(o)
axes[1, 2].set_title("8. Flip Horizontal")
axes[1, 2].axis("off")
# 9. Rotated
axes[1, 3].imshow(p)
axes[1, 3].set_title("9. Rotated 90°")
axes[1, 3].axis("off")
# 10. Color Modified
axes[1, 4].imshow(modified_img)
axes[1, 4].set_title("10. Color Modified")
axes[1, 4].axis("off")

plt.tight_layout()
plt.show()
```



Field	Value	Size	Class
Filename	'/MATLAB Drive/DIP/LR1/Len…	1×31	char
FileModDate	'19-Jan-2026 20:46:48'	1×20	char
FileSize	473831	1×1	double
Format	'png'	1×3	char
FormatVersion	[]	0×0	double
Width	512	1×1	double
Height	512	1×1	double
BitDepth	24	1×1	double
ColorType	'truecolor'	1×9	char
FormatSignature	[137,80,78,71,13,10,26,10]	1×8	double
Colormap	[]	0×0	double
Histogram	[]	0×0	double
InterlaceType	'none'	1×4	char
Transparency	'none'	1×4	char
SimpleTransparencyDat:	[]	0×0	double
BackgroundColor	[]	0×0	double
RenderingIntent	'perceptual'	1×10	char
Chromaticities	[0.3127,0.3290,0.6400,0.330…	1×8	double
Gamma	0.4546	1×1	double

Figure 1: Basic Image Information. Displays original image dimensions and data type, along with resized image dimensions and data type.

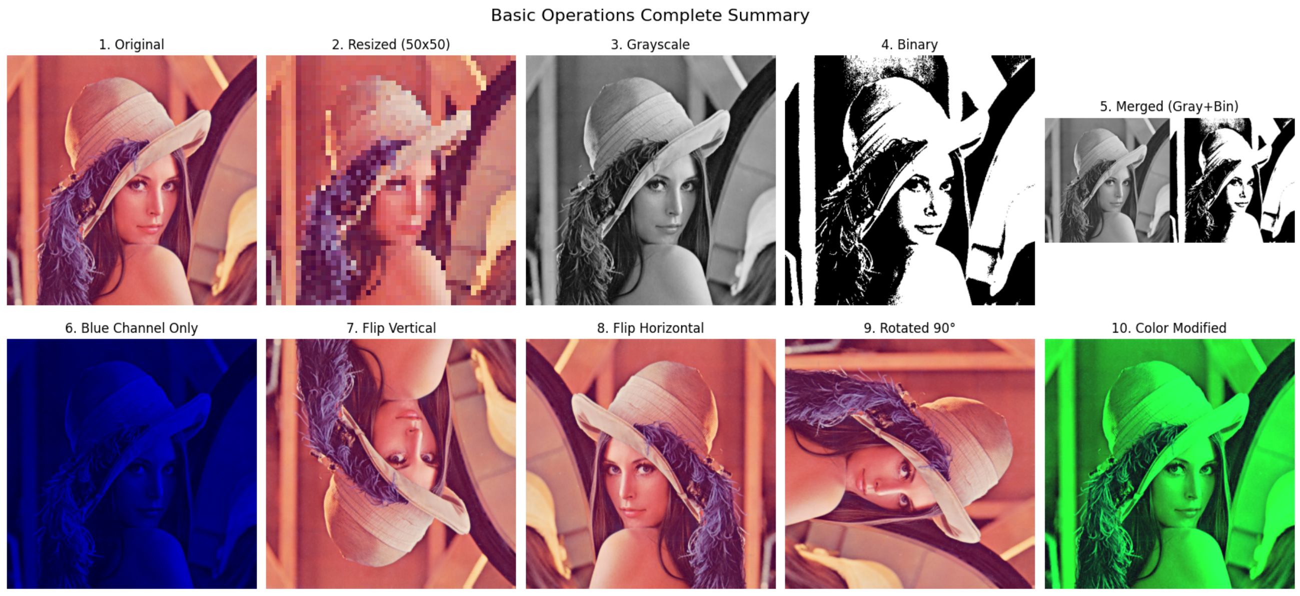


Figure 2: Output of Basic Operations and Manipulations Using MATLAB & Python. Top row: Original, Grayscale, Binary. Bottom row: Merged, Rotated, and Color Channel Modified images.

2.2 Conceptual Task (Masking & Reconstruction)

Task Description

1. Read the image and convert it to RGB and grayscale.
2. Identify objects based on colour intensity (detect all areas where the red channel is > 150).
3. Create a mask of the detected objects.
4. Apply transformation only on the masked objects:
   (a) Flip horizontally.
   (b) Rotate 90 degrees clockwise.
   (c) Increase intensity of the dominant colour channel by 50%.
5. Merge the transformed objects back into the original image, keeping the background unchanged.
6. Compute the area (in pixels) of each detected object.

The following MATLAB code implements this logic, performing detection, isolation, transformation, and reconstruction in a single script.

```matlab
clc;
clear;
close all;

%% 1. Read Image and Convert
% Read the image
rgb_img = imread('Lenna.png');

% Convert to Grayscale
gray_img = rgb2gray(rgb_img);

% Display Step 1
figure; imshow(rgb_img); title('Step 1:
↪  Original RGB Image');
figure; imshow(gray_img); title('Step 1:
↪  Grayscale Image');

%% 2. Identify Objects (Red Channel > 150)
```

```matlab
% Extract Red Channel
R = rgb_img(:,:,1);

% Create logical mask where Red intensity > 150
mask = R > 150;

% Display Step 2
figure; imshow(mask); title('Step 2: Binary Mask (Red > 150)');

%% 3. Create Masked Image (Isolate Object)
% Initialize masked image with zeros (black)
masked_img = zeros(size(rgb_img), 'uint8');

% Apply mask to all three channels
masked_img(:,:,1) = rgb_img(:,:,1) .* uint8(mask);
masked_img(:,:,2) = rgb_img(:,:,2) .* uint8(mask);
masked_img(:,:,3) = rgb_img(:,:,3) .* uint8(mask);

% Display Step 3
figure; imshow(masked_img); title('Step 3: Isolated Masked Object');

%% 4. Apply Transformations on Masked Object
% i. Flip Horizontally
masked_flip = fliplr(masked_img);

% ii. Rotate 90 degrees clockwise
% Note: rot90 rotates counter-clockwise by default, so we use -1 for clockwise
masked_rot_raw = rot90(masked_flip, -1);

% Resize back to original image dimensions to allow merging
% (Rotation changes dimensions, so we force it back to fit the frame)
masked_rot = imresize(masked_rot_raw, [size(rgb_img,1), size(rgb_img,2)]);

% iii. Increase intensity of dominant channel (Red) by 50%
% We convert to double for calculation, multiply by 1.5, then clip at 255
boosted_red = double(masked_rot(:,:,1)) * 1.5;
masked_rot(:,:,1) = uint8(min(boosted_red, 255));

% Display Step 4
figure; imshow(masked_rot); title('Step 4: Transformed Object (Flipped, Rotated, Boosted)');

%% 5. Merge Transformed Object Back to Original
% Logic: Keep the background where the mask is NOT present (~mask),
% then add the new transformed object (masked_rot).
final_img = rgb_img;

% Clear the area where the original object was (create a "hole")
background_only = rgb_img .* uint8(repmat(~mask, [1, 1, 3]));

% Add the transformed object to the background
final_img = background_only + masked_rot;

% Display Step 5
figure; imshow(final_img); title('Step 5: Final Reconstructed Image');

%% 6. Compute Area
% Sum of all white pixels in the binary mask
object_area = sum(mask(:));

% Display Area in Command Window
fprintf('---------------------------------------------------\n');
fprintf('Object Detection Analysis:\n');
fprintf('Total Area of Detected Objects: %d pixels\n', object_area);
fprintf('---------------------------------------------------\n');

%% 7. Summary Display (All Results)
% Create a single figure with subplots for the report
figure('Name', 'Conceptual Task Results', 'NumberTitle', 'off');

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

subplot(2, 3, 2);
imshow(mask);
title('Mask (Red > 150)');

subplot(2, 3, 3);
imshow(masked_img);
title('Isolated Object');

subplot(2, 3, 4);
imshow(masked_rot);
title('Transformed Object');

subplot(2, 3, 5);
imshow(final_img);
title('Final Reconstructed');

subplot(2, 3, 6);
% Display text for the area
axis off;
text(0.1, 0.5, sprintf('Detected Area:\n%d pixels', object_area), ...
    'FontSize', 12, 'FontWeight', 'bold');
title('Area Calculation');
```

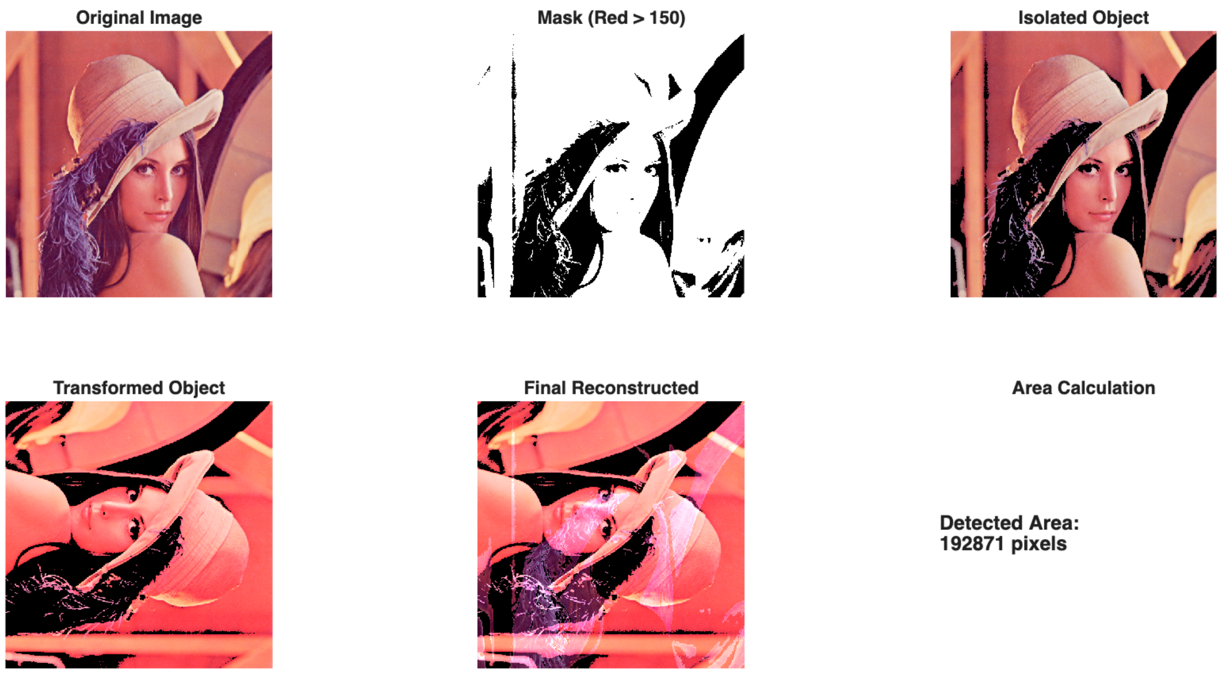


Figure 3: Conceptual Task Results. The figure displays the original image, the generated red-channel mask, the isolated object, and the final reconstructed image where the object is transformed while the background remains static.

3 Discussion

MATLAB's Image Processing Toolbox enabled high-level operations (binary conversion, rotation), while Python's `NumPy` and `OpenCV` required granular array manipulation, reinforcing that images are 2D numerical matrices. Both environments produced identical outputs, validating algorithmic consistency. The conceptual task demonstrated effective object segmentation using color thresholding (Red $>$ 150), isolating Lenna's features without complex edge detection. The reconstruction step illustrated selective region editing via bitwise operations, a cornerstone technique in photo editing and medical imaging applications.

4 Conclusion

In this experiment, we successfully implemented fundamental Digital Image Processing operations in both MATLAB and Python, progressing from basic file handling to geometric transformations and selective region editing via binary masks. These core techniques—segmentation, feature extraction, and computational image manipulation—form the foundation for advanced applications in autonomous systems and medical imaging, while bridging towards modern deep learning approaches.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Boston, MA, USA: Pearson, 2018.

[2] MathWorks, “Image processing toolbox,” [Online]. Available: https://www.mathworks.com/products/image.html, [Accessed: 2024].