

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:

Oishi Jyoti
Assistant Professor
Dept of ECE, RUET

Submitted by:

Md. Tajim An Noor
Roll: 2010025
Semester: 4th Year Even

Submission Date: January 20, 2026

Contents

1 Theory and Introduction	1
2 Methodology	1
2.1 Basic Operations and Manipulations	1
2.1.1 MATLAB Part 1: Basic Operations	1
2.1.2 MATLAB Part 2: Image Manipulation	2
2.1.3 Python Implementation	3
2.2 Conceptual Task (Masking & Reconstruction)	5
3 Discussion	7
4 Conclusion	7
References	7

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.

```
1 clc; clear all; close all;
2
3 % 1. Read Image
4 a = imread('Lenna.png');
5 figure;
6 imshow(a);
7 title('Original Image');
8
9 % 2. Resize
```

```

10 % Resize to 50x50 pixels
11 b = imresize(a, [50, 50]);
12 figure;
13 imshow(b);
14 title('Resized Image');
15
16 % 3. Image Info
17 info = imfinfo('Lenna.png');
18 disp(info);
19
20 % 4. Convert to Grayscale
21 c = rgb2gray(a);
22 figure;
23 imshow(c);
24 title('Grayscale');
25
26 % 5. Convert to Binary
27 % Note: im2bw is older; imbinarize is recommended for newer MATLAB versions
28 d = im2bw(a);
29 figure;
30 imshow(d);
31 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.

```

1 %% 1. Merge Image
2 % Concatenate Grayscale (c) and Binary (d)
3 % We multiply logical 'd' by 255 and convert to uint8 to match 'c'
4 m = horzcat(c, uint8(d) * 255);
5 figure;
6 imshow(m);
7 title('Merged Image (Grayscale + Binary)');
8
9 % 2. Channel Manipulation
10 % Create a copy to avoid modifying original 'a'
11 a_mod = a;
12 a_mod(:, :, 1) = 0; % Set Red channel to 0
13 a_mod(:, :, 2) = 0; % Set Green channel to 0
14 figure;
15 imshow(a_mod);
16 title('Blue Channel Only');
17
18 % 3. Flip and Rotate
19 o = flip(a, 2); % Horizontal flip
20 p = imrotate(a, 90); % 90 degree clockwise rotation
21 figure; imshow(o); title('Flipped Horizontally');
22 figure; imshow(p); title('Rotated 90 Degrees');
23
24 % 4. Change Color Intensity
25 red_factor = 0.2;
26 green_factor = 1.8;
27 blue_factor = 0.5;
28
29 % Scale individual channels (using 'a' as source)
30 red_channel = uint8(a(:, :, 1) * red_factor);

```

```

31 green_channel = uint8(a(:, :, 2) * green_factor);
32 blue_channel = uint8(a(:, :, 3) * blue_factor);
33
34 % Recombine channels
35 modified_img = cat(3, red_channel, green_channel, blue_channel);
36
37 figure;
38 imshow(modified_img);
39 title('Color Intensity Modified');
40
41 %% --- SUMMARY DISPLAY (All 10 Results) ---
42 figure('Name', 'Basic Operations Complete Summary', 'NumberTitle', 'off', 'Position', [50, 50,
43 → 1400, 600]);
44
45 % Row 1: Basic Conversions
46 subplot(2, 5, 1); imshow(a); title('1. Original');
47 subplot(2, 5, 2); imshow(b); title('2. Resized (50x50)');
48 subplot(2, 5, 3); imshow(c); title('3. Grayscale');
49 subplot(2, 5, 4); imshow(d); title('4. Binary');
50 subplot(2, 5, 5); imshow(m); title('5. Merged (Gray+Bin)');
51
52 % Row 2: Manipulations
53 subplot(2, 5, 6); imshow(a_mod); title('6. Blue Channel Only');
54 subplot(2, 5, 7); imshow(n); title('7. Flip Vertical');
55 subplot(2, 5, 8); imshow(o); title('8. Flip Horizontal');
56 subplot(2, 5, 9); imshow(p); title('9. Rotated 90\circ');
57 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.

```

1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 plt.close("all")
6
7 # --- PART 1: Basic Operations ---
8 # 1. Read Image
9 a_bgr = cv2.imread("./images/Lenna.png")
10 if a_bgr is None:
11     raise FileNotFoundError(
12         "Image not found. Please ensure
13         'Lenna.png' is in the working
14         directory.")
15
16 a = cv2.cvtColor(a_bgr, cv2.COLOR_BGR2RGB)
17
18 # 2. Resize # Resize to 50x50 pixels
19 b = cv2.resize(a, (50, 50))
20
21 # 3. Image Info
22 print("Image Info:")
23 print(f"Shape: {a.shape}")
24 print(f"Data Type: {a.dtype}")
25
26 # 4. Convert to Grayscale
27 c = cv2.cvtColor(a, cv2.COLOR_RGB2GRAY)
28
29 # 5. Convert to Binary
30 _, d = cv2.threshold(c, 127, 255,
31 → cv2.THRESH_BINARY)
32
33 # --- PART 2: Manipulations ---
34 # 1. Merge Image (Grayscale + Binary)
35 m = np.hstack((c, d))
36
37 # 2. Channel Manipulation (Blue Channel Only)
38 a_mod = a.copy()
39 a_mod[:, :, 0] = 0 # Set Red channel to 0
40 a_mod[:, :, 1] = 0 # Set Green channel to 0
41
42 # 3. Flip and Rotate
43 n = cv2.flip(a, 0) # Vertical
44 o = cv2.flip(a, 1) # Horizontal
45 p = cv2.rotate(a, cv2.ROTATE_90_CLOCKWISE)
46
47 # 4. Change Color Intensity
48 red_factor = 0.2
49 green_factor = 1.8

```

```

48 blue_factor = 0.5
49
50 # Scale individual channels
51 red_channel = np.clip(a[:, :, 0] * red_factor,
52 → 0, 255).astype(np.uint8)
52 green_channel = np.clip(a[:, :, 1] *
53 → green_factor, 0, 255).astype(np.uint8)
53 blue_channel = np.clip(a[:, :, 2] *
54 → blue_factor, 0, 255).astype(np.uint8)
55
56 # Recombine channels
57 modified_img = cv2.merge((red_channel,
58 → green_channel, blue_channel))
59
60 # --- SUMMARY DISPLAY (All 10 Results) ---
61 fig, axes = plt.subplots(2, 5, figsize=(20, 8))
62 fig.suptitle("Basic Operations Complete"
63 → Summary", fontsize=16)
64
65 # Row 1: Basic Conversions
66 # 1. Original
67 axes[0, 0].imshow(a)
68 axes[0, 0].set_title("1. Original")
69 axes[0, 0].axis("off")
70
71 # 2. Resized
72 axes[0, 1].imshow(b)
73 axes[0, 1].set_title("2. Resized (50x50)")
74 axes[0, 1].axis("off")
75
76 # 3. Grayscale (needs cmap='gray')
77 axes[0, 2].imshow(c, cmap="gray")
78 axes[0, 2].set_title("3. Grayscale")
79 axes[0, 2].axis("off")
80
81 # 4. Binary (needs cmap='gray')
82 axes[0, 3].imshow(d, cmap="gray")
83 axes[0, 3].set_title("4. Binary")
84 axes[0, 3].axis("off")
85
86 # 5. Merged (needs cmap='gray')
87 axes[0, 4].imshow(m, cmap="gray")
88 axes[0, 4].set_title("5. Merged (Gray+Bin)")
89 axes[0, 4].axis("off")
90
91 # Row 2: Manipulations
92 # 6. Blue Channel Only
93 axes[1, 0].imshow(a_mod)
94 axes[1, 0].set_title("6. Blue Channel Only")
95 axes[1, 0].axis("off")
96
97 # 7. Flip Vertical
98 axes[1, 1].imshow(n)
99 axes[1, 1].set_title("7. Flip Vertical")
100 axes[1, 1].axis("off")
101
102 # 8. Flip Horizontal
103 axes[1, 2].imshow(o)
104 axes[1, 2].set_title("8. Flip Horizontal")
105 axes[1, 2].axis("off")
106
107 # 9. Rotated
108 axes[1, 3].imshow(p)
109 axes[1, 3].set_title("9. Rotated 90°")
110 axes[1, 3].axis("off")
111
112 # 10. Color Modified
113 axes[1, 4].imshow(modified_img)
114 axes[1, 4].set_title("10. Color Modified")
115 axes[1, 4].axis("off")
116
117 plt.tight_layout()
118 plt.show()

```

info x			
1x1 struct with 40 fields			
Field	Value	Size	Class
Filename	'/MATLAB Drive/DIP/LR1/Len...	1x31	char
FileModDate	'19-Jan-2026 20:46:48'	1x20	char
FileSize	473831	1x1	double
Format	'png'	1x3	char
FormatVersion	[]	0x0	double
Width	512	1x1	double
Height	512	1x1	double
BitDepth	24	1x1	double
ColorType	'truecolor'	1x9	char
FormatSignature	[137,80,78,71,13,10,26,10]	1x8	double
Colormap	[]	0x0	double
Histogram	[]	0x0	double
InterlaceType	'none'	1x4	char
Transparency	'none'	1x4	char
SimpleTransparencyData	[]	0x0	double
BackgroundColor	[]	0x0	double
RenderingIntent	'perceptual'	1x10	char
Chromaticities	[0.3127,0.3290,0.6400,0.330...	1x8	double
Gamma	0.4546	1x1	double

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

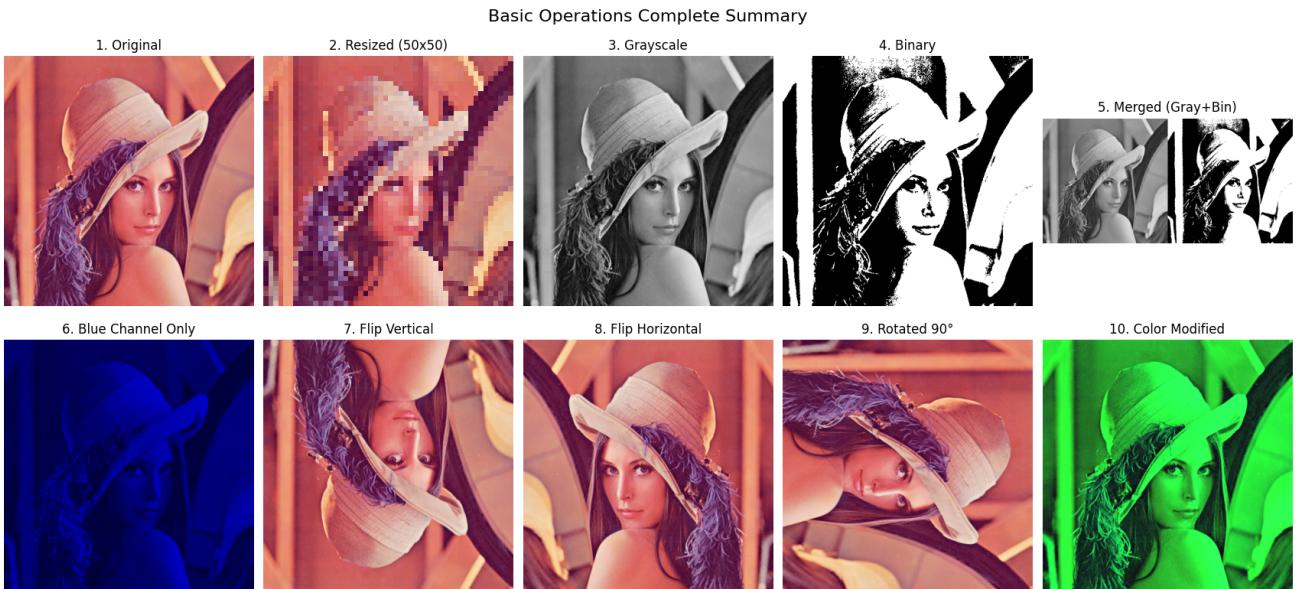


Figure 2: Output of Basic Operations and Manipulations Using MATLAB & Python. Top row: Original, Grayscale, Binary. Bottom row: Merged, Rotated, and Color Channel Modified images.

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.

```

1 clc;
2 clear;
3 close all;
4
5 %% 1. Read Image and Convert
6 % Read the image
7 rgb_img = imread('Lenna.png');
8
9 % Convert to Grayscale
10 gray_img = rgb2gray(rgb_img);
11
12 % Display Step 1
13 figure; imshow(rgb_img); title('Step 1:
14 %> Original RGB Image');
15 figure; imshow(gray_img); title('Step 1:
16 %> Grayscale Image');
17
18 %% 2. Identify Objects (Red Channel > 150)

```

```

17 % Extract Red Channel
18 R = rgb_img(:,:,1);
19
20 % Create logical mask where Red intensity > 150
21 mask = R > 150;
22
23 % Display Step 2
24 figure; imshow(mask); title('Step 2: Binary
→ Mask (Red > 150)');
25
26 %% 3. Create Masked Image (Isolate Object)
27 % Initialize masked image with zeros (black)
28 masked_img = zeros(size(rgb_img), 'uint8');
29
30 % Apply mask to all three channels
31 masked_img(:,:1) = rgb_img(:,:1) .* 
→ uint8(mask);
32 masked_img(:,:2) = rgb_img(:,:2) .* 
→ uint8(mask);
33 masked_img(:,:3) = rgb_img(:,:3) .* 
→ uint8(mask);
34
35 % Display Step 3
36 figure; imshow(masked_img); title('Step 3:
→ Isolated Masked Object');
37
38 %% 4. Apply Transformations on Masked Object
39 % i. Flip Horizontally
40 masked_flip = fliplr(masked_img);
41
42 % ii. Rotate 90 degrees clockwise
43 % Note: rot90 rotates counter-clockwise by
→ default, so we use -1 for clockwise
44 masked_rot_raw = rot90(masked_flip, -1);
45
46 % Resize back to original image dimensions to
→ allow merging
47 % (Rotation changes dimensions, so we force it
→ back to fit the frame)
48 masked_rot = imresize(masked_rot_raw,
→ [size(rgb_img,1), size(rgb_img,2)]);
49
50 % iii. Increase intensity of dominant channel
→ (Red) by 50%
51 % We convert to double for calculation,
→ multiply by 1.5, then clip at 255
52 boosted_red = double(masked_rot(:,:1)) * 1.5;
53 masked_rot(:,:1) = uint8(min(boosted_red,
→ 255));
54
55 % Display Step 4
56 figure; imshow(masked_rot); title('Step 4:
→ Transformed Object (Flipped, Rotated,
→ Boosted)');
57
58 %% 5. Merge Transformed Object Back to Original
59 % Logic: Keep the background where the mask is
→ NOT present (~mask),
60 % then add the new transformed object
→ (masked_rot).
61 final_img = rgb_img;
62
63 % Clear the area where the original object was
→ (create a "hole")
64 background_only = rgb_img .* 
→ uint8(repmat(~mask, [1, 1, 3]));
65
66 % Add the transformed object to the background
67 final_img = background_only + masked_rot;
68
69 % Display Step 5
70 figure; imshow(final_img); title('Step 5: Final
→ Reconstructed Image');
71
72 %% 6. Compute Area
73 % Sum of all white pixels in the binary mask
74 object_area = sum(mask(:));
75
76 % Display Area in Command Window
77 fprintf('-----\n');
78 fprintf('Object Detection Analysis:\n');
79 fprintf('Total Area of Detected Objects: %d
→ pixels\n', object_area);
80 fprintf('-----\n');
81
82 %% 7. Summary Display (All Results)
83 % Create a single figure with subplots for the
→ report
84 figure('Name', 'Conceptual Task Results',
→ 'NumberTitle', 'off');
85
86 subplot(2, 3, 1);
87 imshow(rgb_img);
88 title('Original Image');
89
90 subplot(2, 3, 2);
91 imshow(mask);
92 title('Mask (Red > 150)');
93
94 subplot(2, 3, 3);
95 imshow(masked_img);
96 title('Isolated Object');
97
98 subplot(2, 3, 4);
99 imshow(masked_rot);
100 title('Transformed Object');
101
102 subplot(2, 3, 5);
103 imshow(final_img);
104 title('Final Reconstructed');
105
106 subplot(2, 3, 6);
107 % Display text for the area
108 axis off;
109 text(0.1, 0.5, sprintf('Detected Area:\n%d
→ pixels', object_area), ...
110 'FontSize', 12, 'FontWeight', 'bold');
111 title('Area Calculation');

```

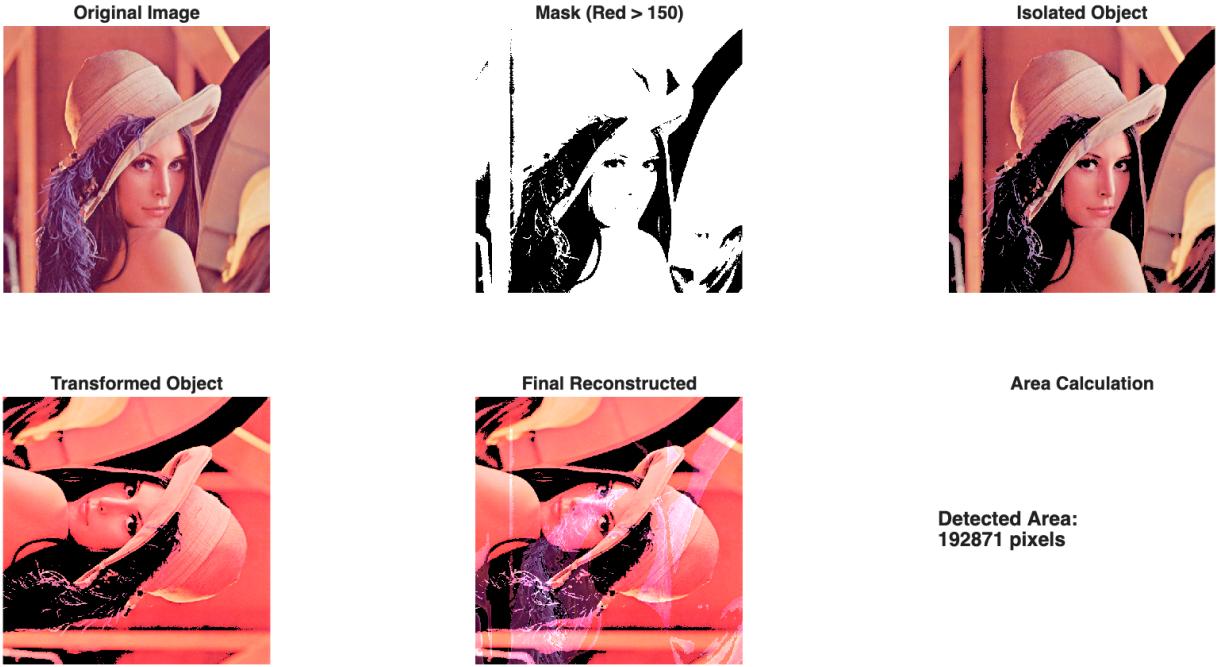


Figure 3: Conceptual Task Results. The figure displays the original image, the generated red-channel mask, the isolated object, and the final reconstructed image where the object is transformed while the background remains static.

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 ($\text{Red} > 150$), 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].