

*Heaven's Light is Our Guide*

**Rajshahi University of Engineering & Technology**



**Department of  
Electrical & Computer Engineering**

## **Lab Report 2**

Pixel-Level Manipulation and Histogram Analysis

**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

<b>1</b>	<b>Theory and Introduction</b>	<b>1</b>
<b>2</b>	<b>Methodology</b>	<b>1</b>
2.1	Pixel Manipulation: Crosshair and Color Bands . . . . .	1
2.2	Histogram Computation (Manual Approach) . . . . .	1
2.3	Python Implementation . . . . .	2
<b>3</b>	<b>Results</b>	<b>3</b>
<b>4</b>	<b>Discussion</b>	<b>3</b>
<b>5</b>	<b>Conclusion</b>	<b>4</b>
	<b>References</b>	<b>4</b>

# Pixel-Level Manipulation and Histogram Analysis

## 1 Theory and Introduction

At the fundamental level, a digital image is a matrix of numerical values. Manipulating an image involves accessing and modifying these values at specific coordinates, a technique known as spatial domain processing [1].

This experiment focuses on two core concepts:

- **Direct Pixel Access:** Modifying specific rows, columns, or regions of the image matrix to draw shapes or alter colors. In this experiment, we utilize the Python Imaging Library (PIL) to load images and NumPy to perform efficient matrix operations.
- **Histogram Analysis:** A histogram is a graphical representation of the tonal distribution in a digital image [1]. It plots the number of pixels for each tonal value (0–255). While modern libraries provide optimized functions for this, computing it manually via loops reinforces the understanding of the underlying data distribution algorithm.

## 2 Methodology

### 2.1 Pixel Manipulation: Crosshair and Color Bands

We manipulated the pixel arrays directly using NumPy slicing to alter specific spatial regions.

1. **White Cross:** We calculated the center coordinates ( $c_x, c_y$ ) and assigned the value 255 to all channels in the central row and column, resulting in a white crosshair.
2. **Color Bands:** We defined a 100-pixel wide region around the center. Instead of painting a solid color, we isolated specific channels:
  - **Vertical Red Band:** We set the Green and Blue channels to 0, leaving only the Red component of the original pixels visible.
  - **Horizontal Blue Band:** We set the Red and Green channels to 0, leaving only the Blue component visible.

### 2.2 Histogram Computation (Manual Approach)

For the histogram, we converted the image to grayscale (Luminance mode) and implemented the counting logic from scratch:

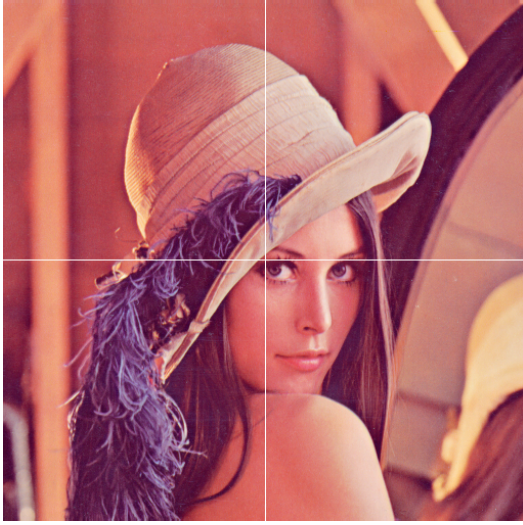
1. Initialize an array of 256 zeros (bins).
2. Flatten the 2D grayscale image into a 1D array.
3. Iterate through every pixel value; use the pixel value as the index to increment the corresponding bin count.

## 2.3 Python Implementation

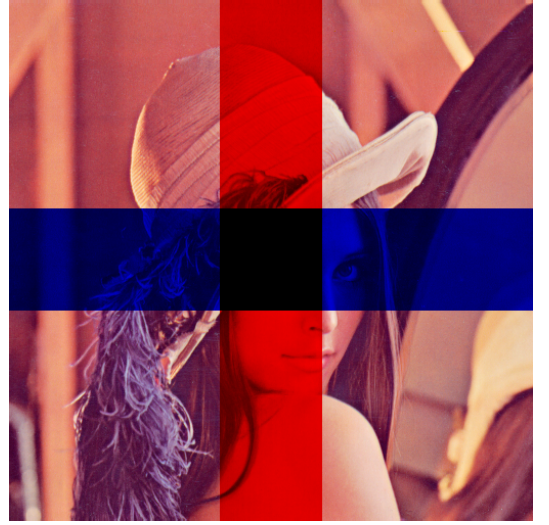
The following single script executes all three tasks and saves the results as PNG images.

```
1  from PIL import Image
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import os
5
6  os.makedirs("./images/output", exist_ok=True)
7
8  # Path to image
9  img_path = "./images/Lenna.png"
10
11
12  # Task 1: White Cross
13  # Load Lenna as RGB and convert to numpy array
14  img = Image.open(img_path).convert("RGB")
15  arr = np.array(img)
16
17  # Compute image center (row = cy, column = cx)
18  h, w, _ = arr.shape
19  cy, cx = h // 2, w // 2
20
21  # Draw a white horizontal line through the
22  ↪ center row
23  arr[cy, :, :] = 255
24  # Draw a white vertical line through the center
25  ↪ column
26  arr[:, cx, :] = 255
27
28  # Convert back to PIL image, display, and save
29  modified_img = Image.fromarray(arr)
30  # modified_img.show()
31  modified_img.save("./images/output/task1_cross_
32  ↪ .png")
33  print("Task 1: Saved
34  ↪ './images/output/task1_cross.png'")
35
36  # Task 2: Color Bands (Red Vertical, Blue
37  ↪ Horizontal)
38  # Reload Lenna image and convert to array
39  img_band = Image.open(img_path).convert("RGB")
40  arr_band = np.array(img_band)
41
42  # Define band thickness
43  band_half = 50 # total width/height = 100
44  ↪ pixels
45
46  # Apply vertical red-only band (centered)
47  # Logic: Keep Red channel, set Green (index 1)
48  ↪ and Blue (index 2) to 0
49
50  arr_band[:, cx - band_half : cx + band_half, 1]
51  ↪ = 0 # zero green
52  arr_band[:, cx - band_half : cx + band_half, 2]
53  ↪ = 0 # zero blue
54
55  # Apply horizontal blue-only band (centered)
56  # Logic: Keep Blue channel, set Red (index 0)
57  ↪ and Green (index 1) to 0
58  arr_band[cy - band_half : cy + band_half, :, 0]
59  ↪ = 0 # zero red
60  arr_band[cy - band_half : cy + band_half, :, 1]
61  ↪ = 0 # zero green
62
63  # Convert back to image, display, and save
64  band_img = Image.fromarray(arr_band)
65  # band_img.show()
66  band_img.save("./images/output/task2_bands.png_
67  ↪ ")
68  print("Task 2: Saved
69  ↪ './images/output/task2_bands.png'")
70
71  # Task 3: Manual Histogram Loop
72  # Load Lenna image and convert to grayscale
73  gray_img_manual =
74  ↪ Image.open(img_path).convert("L")
75  gray_arr_manual = np.array(gray_img_manual)
76
77  # Manually compute histogram using a loop
78  hist_manual = np.zeros(256, dtype=int)
79  for pixel_value in gray_arr_manual.ravel():
80  hist_manual[pixel_value] += 1
81
82  # Plot the histogram
83  plt.figure(figsize=(6, 4))
84  plt.bar(range(256), hist_manual, width=1.0,
85  ↪ color="gray")
86  plt.xlabel("Pixel value")
87  plt.ylabel("Frequency")
88  plt.title("Lenna grayscale histogram (manual
89  ↪ loop)")
90  plt.tight_layout()
91
92  # Save the plot
93  plt.savefig("./images/output/task3_histogram.p_
94  ↪ ng")
95  print("Task 3: Saved
96  ↪ './images/output/task3_histogram.png'")
97  # plt.show()
```

### 3 Results



(a) Task 1: White Cross



(b) Task 2: Red & Blue Channel Bands

Figure 1: Pixel Manipulation Outputs

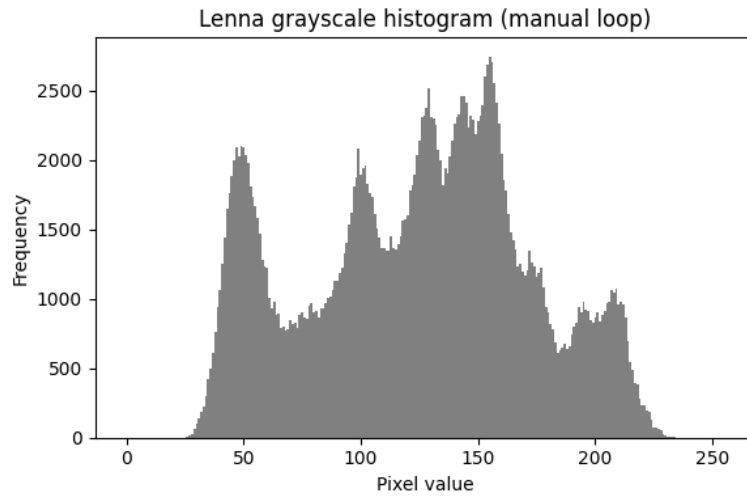


Figure 2: Task 3: Manual Histogram Analysis (Grayscale Frequency Distribution)

### 4 Discussion

The experiment demonstrated treating images as NumPy arrays. In Task 2, channel masking (setting specific channels to zero) retained luminosity while filtering colors—a technique fundamental to color processing. The manual histogram loop revealed the bimodal distribution in the Lenna image (light and dark regions). While educational, manual loops are slower than vectorized NumPy functions like `np.histogram` for large datasets.

## 5 Conclusion

This lab successfully demonstrated direct matrix manipulation and statistical analysis of digital images. We learned how to modify specific spatial regions using coordinate math and how to extract global image statistics through histogram computation. The use of NumPy slicing proved to be an efficient method for defining Regions of Interest (ROI) without the need for complex iterative loops.

## References

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