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 Odd

Submission Date: January 20, 2026

Contents

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.

```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os

os.makedirs("./images/output", exist_ok=True)

# Path to image
img_path = "./images/Lenna.png"


# Task 1: White Cross
# Load Lenna as RGB and convert to numpy array
img = Image.open(img_path).convert("RGB")
arr = np.array(img)

# Compute image center (row = cy, column = cx)
h, w, _ = arr.shape
cy, cx = h // 2, w // 2

# Draw a white horizontal line through the center row
arr[cy, :, :] = 255
# Draw a white vertical line through the center column
arr[:, cx, :] = 255

# Convert back to PIL image, display, and save
modified_img = Image.fromarray(arr)
# modified_img.show()
modified_img.save("./images/output/task1_cross.png")
print("Task 1: Saved './images/output/task1_cross.png'")

# Task 2: Color Bands (Red Vertical, Blue Horizontal)
# Reload Lenna image and convert to array
img_band = Image.open(img_path).convert("RGB")
arr_band = np.array(img_band)

# Define band thickness
band_half = 50  # total width/height = 100 pixels

# Apply vertical red-only band (centered)
# Logic: Keep Red channel, set Green (index 1) and Blue (index 2) to 0
arr_band[:, cx - band_half : cx + band_half, 1] = 0  # zero green
arr_band[:, cx - band_half : cx + band_half, 2] = 0  # zero blue

# Apply horizontal blue-only band (centered)
# Logic: Keep Blue channel, set Red (index 0) and Green (index 1) to 0
arr_band[cy - band_half : cy + band_half, :, 0] = 0  # zero red
arr_band[cy - band_half : cy + band_half, :, 1] = 0  # zero green

# Convert back to image, display, and save
band_img = Image.fromarray(arr_band)
# band_img.show()
band_img.save("./images/output/task2_bands.png")
print("Task 2: Saved './images/output/task2_bands.png'")

# Task 3: Manual Histogram Loop
# Load Lenna image and convert to grayscale
gray_img_manual = Image.open(img_path).convert("L")
gray_arr_manual = np.array(gray_img_manual)

# Manually compute histogram using a loop
hist_manual = np.zeros(256, dtype=int)
for pixel_value in gray_arr_manual.ravel():
    hist_manual[pixel_value] += 1

# Plot the histogram
plt.figure(figsize=(6, 4))
plt.bar(range(256), hist_manual, width=1.0, color="gray")
plt.xlabel("Pixel value")
plt.ylabel("Frequency")
plt.title("Lenna grayscale histogram (manual loop)")
plt.tight_layout()

# Save the plot
plt.savefig("./images/output/task3_histogram.png")
print("Task 3: Saved './images/output/task3_histogram.png'")
# plt.show()
```

3 Results

(a) Task 1: White Cross

(b) Task 2: Red & Blue Channel Bands

Figure 1: Pixel Manipulation Outputs

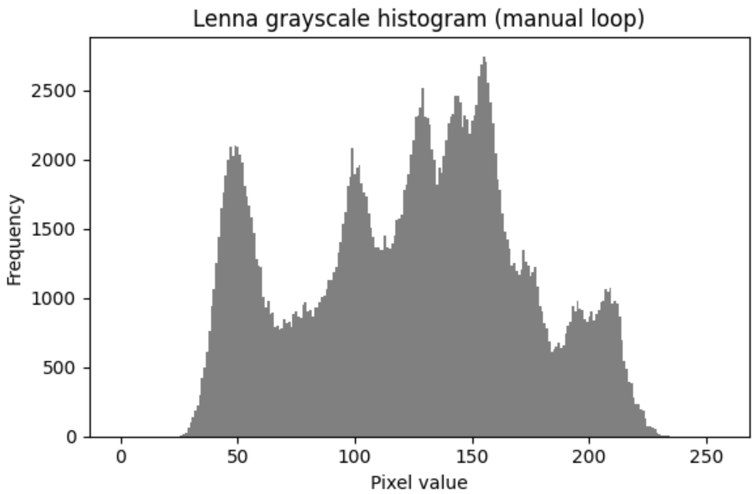

Figure 2: Task 3: Manual Histogram Analysis (Grayscale Frequency Distribution)

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.