Assignment 3 – A Practical Application with a Graphical User Interface (GUI)

NAME: S. Madhusmitha Reddy

Roll Number: 22671A73A6

Date: 07 September 2025

Abstract

This report presents the design and implementation of an Image Processing Toolkit with a Graphical User Interface (GUI), developed using Python, OpenCV, Pillow (PIL), and Streamlit. The toolkit allows users to upload images, perform basic computer vision operations (color conversions, geometric transformations, resizing, rotation, translation), analyze image properties, and download processed results.

The application is implemented in a single Python file (app.py) and provides a user-friendly interface for performing image processing tasks without requiring prior coding knowledge. The report explains the system architecture, helper functions, and GUI design, including mathematical formulations behind operations. Source code snippets are included, along with placeholders for screenshots demonstrating the GUI outputs.

Contents

Chapter 1 – Introduction

The objective of this assignment is to develop a practical GUI-based image processing application using Python libraries. The application provides:

- File upload and display of original/processed images.
- Color conversions (RGB→GRAY, HSV, YCbCr, etc.).
- Geometric transformations (rotate, resize, translate).
- Image property extraction (resolution, format, size).
- Download functionality for processed images.

Motivation: While OpenCV and Pillow provide strong APIs for computer vision, they are primarily code-based. Streamlit bridges this gap by offering an intuitive GUI, making the toolkit accessible to non-programmers.

Chapter 2 – System Overview

System Components

- **Frontend (Streamlit GUI):** User interface for image upload, selection of operations, display of results, and download.
- **Backend (OpenCV + Pillow):** Image processing logic for conversions and transformations.

Workflow

1. User uploads an image (JPG, PNG, BMP).
2. Application displays the original image.
3. User selects an operation from the sidebar.
4. Processed image is displayed alongside the original.
5. Image information is shown (resolution, channels, format, size).
6. User downloads processed image.

Figure 2.1 – System Workflow Diagram
(Insert workflow diagram here: Upload → Process → Display → Save)

Chapter 3 – Helper Functions

3.1 Image Loading

```
def load_image(image_file):
    image = Image.open(image_file)
    return cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
```

Explanation:

- Uses Pillow to open uploaded files.
- Converts to NumPy array and then to OpenCV BGR format for further processing.

3.2 Image Information Retrieval

```
def image_info(image):
    h, w = image.shape[:2]
    channels = image.shape[2] if len(image.shape) == 3 else 1
    size_kb = len(cv2.imencode('.jpg', image)[1]) / 1024
    return {
        "Resolution": f"{w} x {h}",
        "Channels": channels,
        "Size (KB)": f"{size_kb:.2f}",
        "Format": uploaded_file.type if uploaded_file else "N/A"
    }
```

Explanation:

- Extracts resolution, number of channels, file size, and format.
- Displayed in the status bar.

3.3 Color Conversions

```
def convert_color(image, conversion):
    if conversion == "RGB to GRAY":
```

```
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  elif conversion == "RGB to HSV":
    return cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
  elif conversion == "RGB to YCbCr":
    return cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
  elif conversion == "BGR to RGB":
    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  return image
```

Explanation:

- Supports multiple color models.
- Example formulas:
    - Gray conversion:

$$Y=0.299R+0.587G+0.114BY = 0.299R + 0.587G + 0.114B$$

3.4 Geometric Transformations

Rotation

```
def rotate_image(image, angle):
  h, w = image.shape[:2]
  matrix = cv2.getRotationMatrix2D((w//2, h//2), angle, 1.0)
  return cv2.warpAffine(image, matrix, (w, h))
```

Formula:

$$[x′y′]=[cos⁡θ−sin⁡θsin⁡θcos⁡θ][xy]\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Resize

```
def resize_image(image, scale):
  return cv2.resize(image, None, fx=scale, fy=scale)
```

Translation

```
def translate_image(image, tx, ty):
    matrix = np.float32([[1, 0, tx], [0, 1, ty]])
    return cv2.warpAffine(image, matrix, (image.shape[1], image.shape[0]))
```

Formula:

[x′y′]=[10tx01ty][xy1]\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}

Chapter 4 – Streamlit Application Structure

- **Sidebar:** Upload + operations.
- **Main panel:** Two columns (original vs processed).
- **Status bar:** Displays image info + save option.

Code Snippet (Main Structure):

```
uploaded_file = st.sidebar.file_uploader("Upload an Image", type=["jpg", "jpeg", "png", "bmp"])

if uploaded_file:
    original_image = load_image(uploaded_file)
    processed_image = original_image.copy()
```

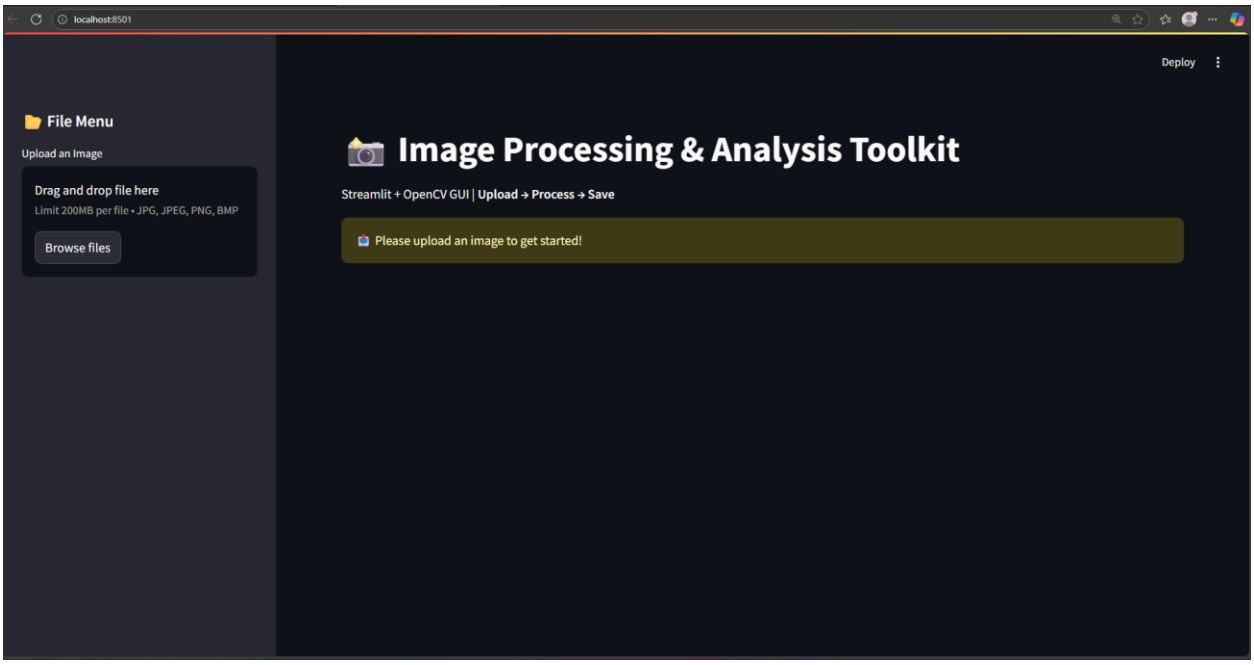

Figure 4.1 – GUI Layout

Chapter 5 – Detailed Operations

5.1 Color Conversions

- Modes: GRAY, HSV, YCbCr, RGB.

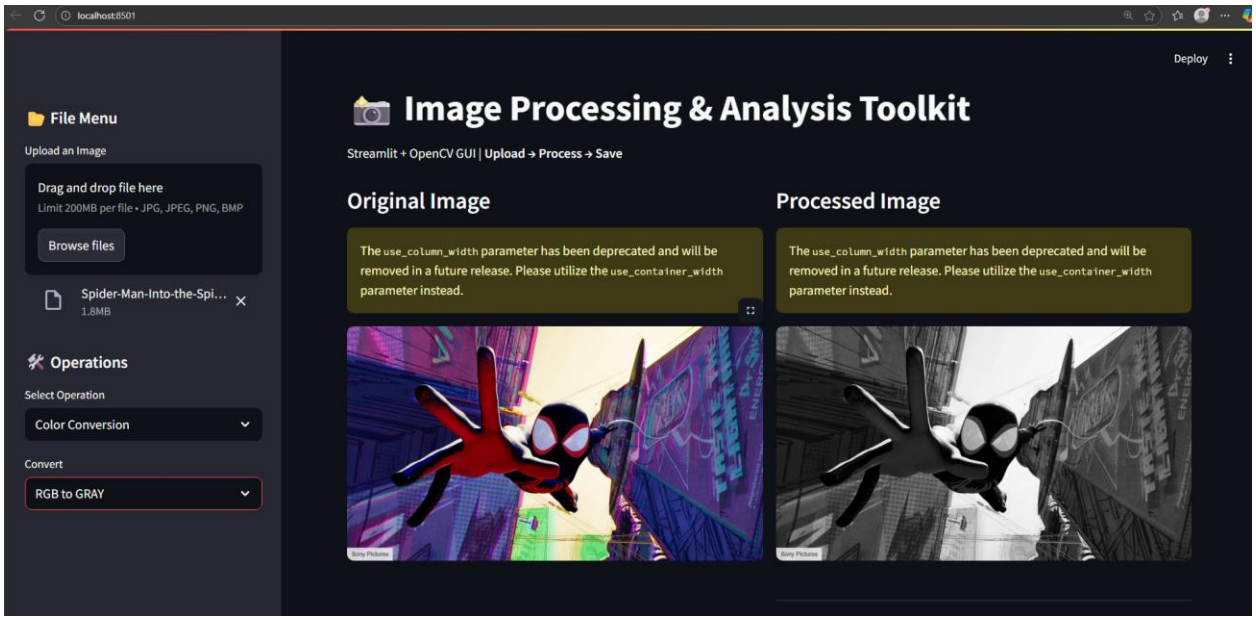

Figure 5.1: RGB to Grayscale.

5.2 Rotate

- Slider control: -180° to 180°.

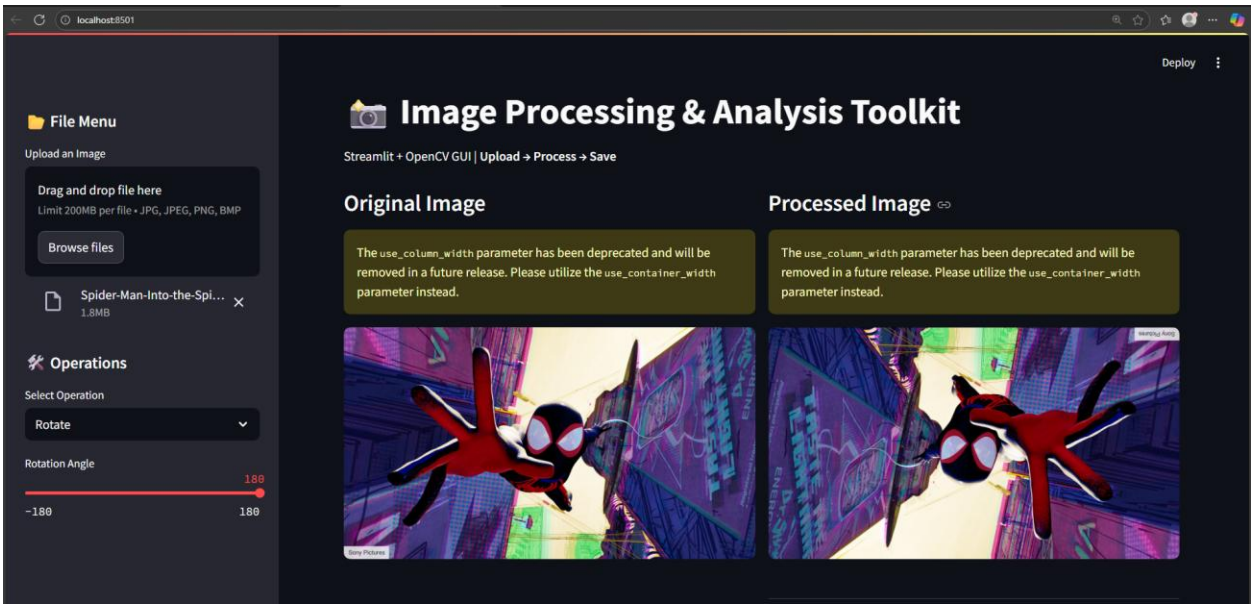

Figure 5.2 – Example: Image rotated by 90°

5.3 Resize

- Slider: scale percentage (10% – 300%).

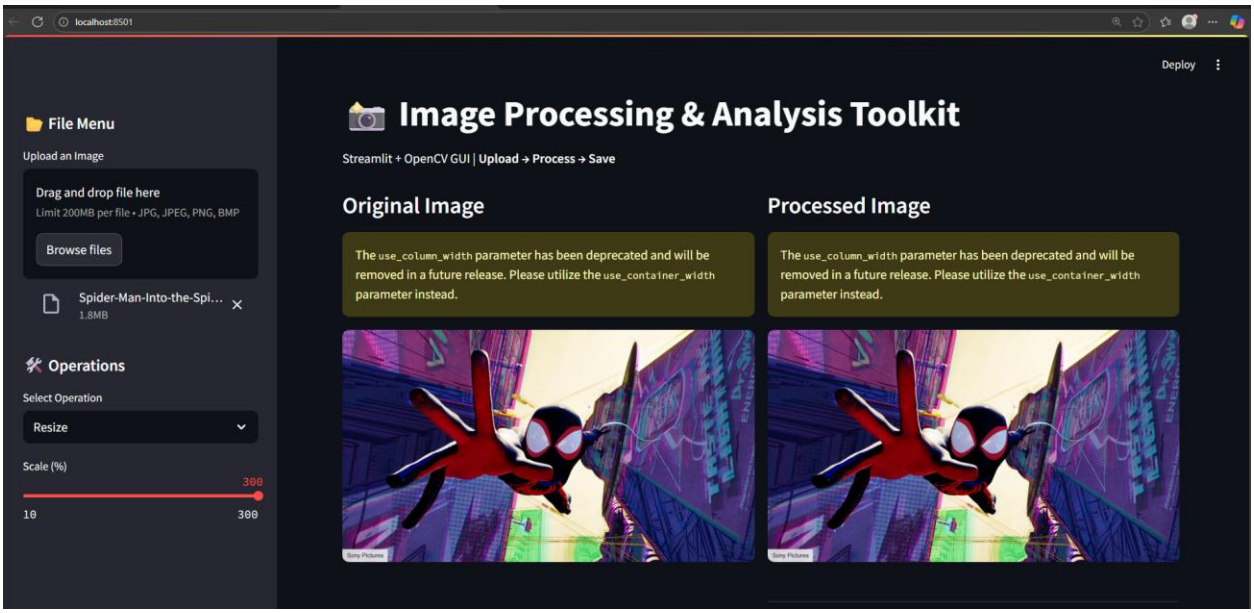

Figure 5.3 – Example: Image scaled to 50%

5.4 Translate

- Sliders for X, Y shift.

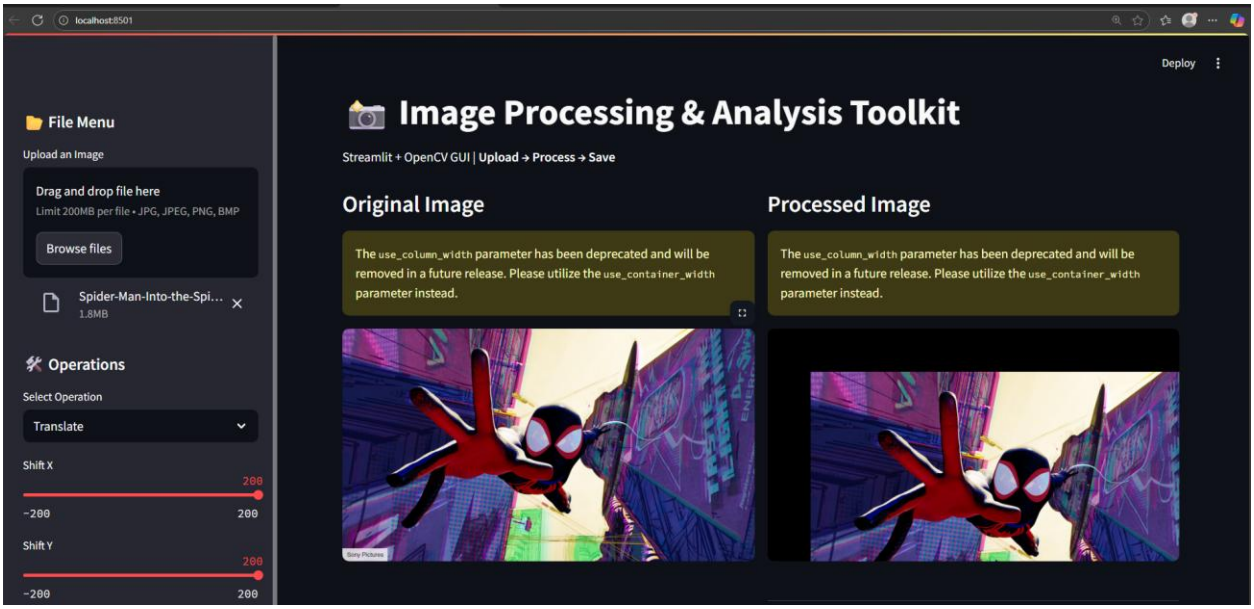

Figure 5.4 – Example: Image shifted 100px right

Chapter 6 – Save and Status Bar

```
st.download_button(
    label="💾 Download Processed Image",
    data=cv2.imencode('.png', processed_image)[1].tobytes(),
    file_name="processed_image.png",
    mime="image/png"
)
```

- Supports PNG by default.
- Future work: JPEG quality sliders.

Chapter 7 – Jupyter Notebook Analysis

For backend testing:

```
import matplotlib.pyplot as plt

plt.subplot(1,2,1)

plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))

plt.subplot(1,2,2)

plt.imshow(cv2.cvtColor(processed_image, cv2.COLOR_BGR2RGB))

plt.show()
```

Chapter 8 – Conclusion

The image processing toolkit successfully integrates OpenCV, Pillow, and Streamlit to deliver an interactive GUI for essential computer vision tasks. It simplifies image processing for beginners while maintaining modular, reusable backend code.

Future Enhancements:

- Filters (Gaussian, Median).
- Edge detection (Sobel, Canny).
- Morphological operations.
- Batch processing.

Chapter 9 – References

- OpenCV Documentation: https://docs.opencv.org/
- Streamlit Documentation: https://docs.streamlit.io/
- Pillow Documentation: https://pillow.readthedocs.io/