Project Report: Image Processing Toolkit using Streamlit & OpenCV

1. Problem Statement / Objective

The objective of this project is to design and implement an interactive image processing toolkit that allows users to:
- Upload images in multiple formats.
- Apply different image transformations, enhancements, and filters interactively.
- Visualize both original and processed images side-by-side.
- Save or download processed results in different formats.

2. Introduction / Assignment Tasks

Image processing is widely used in fields like computer vision, medical imaging, biometrics, and multimedia applications.
This project integrates Streamlit (for UI) with OpenCV (for core image processing) to build a web-based interactive platform.
Assignment tasks covered include:
1. Image upload, display, and format handling.
2. Basic operations: color space conversions, transformations.
3. Bitwise operations on multiple images.
4. Filtering, morphological operations, and edge detection.
5. Image enhancement techniques (histogram equalization, contrast stretching, sharpening).
6. Exporting processed images in different formats.

3. Requirements Gathering / Libraries Used

Requirements:
- Simple GUI for non-programmers to experiment with images.
- Fast, interactive feedback while applying transformations.
- Support for multiple formats (PNG, JPG, BMP, TIFF).
- Ability to download/save the processed results.

Libraries Used:
- Streamlit: for building the web-based GUI.
- OpenCV (cv2): core image processing functions.
- NumPy: numerical operations and image array handling.
- PIL (Pillow): image conversion and export.
- io, os, tempfile: temporary file handling and downloads.

4. Methodology / Code Explanation

The project is divided into functional blocks:

a. Image Loading & Utilities:
- Handles uploads using st.file_uploader().
- Converts uploaded file bytes into OpenCV-readable NumPy arrays.
- Utilities for PIL ↔ OpenCV conversion, image metadata extraction, and byte export.

b. Image Processing Operations:
1. Color Conversions: BGR ↔ RGB, HSV, YCbCr, Grayscale.
2. Transformations: Rotate, Scale, Translate, Affine, Perspective.
3. Bitwise Operations: AND, OR, XOR, NOT (with optional second image).
4. Filtering & Morphology: Gaussian, Median, Mean, Sobel, Laplacian, Dilation, Erosion, Opening, Closing.
5. Enhancement & Edge Detection: Histogram Equalization, Contrast Stretching, Sharpening, Canny Edge.

c. User Interface (Streamlit):
- Sidebar menus for selecting operations and parameters (sliders, checkboxes, dropdowns).
- Dual display columns: Original vs. Processed image.
- Image info panel (dimensions, channels, format, size).
- Save/export options with adjustable JPEG quality.

5. Results

The system successfully:
- Accepts multiple input formats.
- Displays images before & after processing side-by-side.
- Applies all mentioned operations interactively with real-time updates.
- Saves processed images in PNG, JPG, BMP formats.

(Screenshots of implementation can be inserted here.)

6. Conclusion

Knowledge Gained:
- Practical use of OpenCV functions for image processing.
- Building interactive web apps with Streamlit.
- File handling, format conversion, and real-time UI updates.

Difficulties Overcome:
- Handling images with different channels (RGB, Grayscale, HSV).
- Ensuring consistent BGR/RGB conversions between OpenCV and PIL.
- Making kernel size adjustments for filters (odd sizes only).
- Preserving interactivity without performance lag.

Future Improvements:

- Add advanced filters (bilateral, non-local means denoising).
- Support for video processing in real-time.
- Region-of-interest (ROI) based editing.
- Deep learning–based enhancements (super-resolution, segmentation).

5. Results

The following screenshots showcase the implemented image processing toolkit. The system allows users to upload images, apply transformations, and view the original and processed outputs side-by-side.

```python
import streamlit as st
import cv2
import numpy as np
from PIL import Image
import io
import os
import tempfile

st.set_page_config(page_title="Image Processing Toolkit", layout="wide")

# ---------------------- Utility Functions ----------------------
@st.cache_data(show_spinner=False)
def load_image(uploaded_file):
    if uploaded_file is None:
        return None
    file_bytes = np.asarray(bytearray(uploaded_file.read()), dtype=np.uint8)
    img = cv2.imdecode(file_bytes, cv2.IMREAD_UNCHANGED)
    # Convert 4-channel BGRA to BGR for consistency
    if img is not None and img.shape[-1] == 4:
        img = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)
    return img
```

These screenshots demonstrate different image processing operations such as histogram equalization, color conversions, and geometric transformations applied via the Streamlit-based toolkit.