J.B. INSTITUTE OF ENGINEERING AND TECHNOLOGY
(UGC AUTONOMOUS)
Bhaskar Nagar, Moinabad Mandal, R.R. District, Hyderabad -500075

COMPUTER VISION

Image Processing Toolkit

Assignment Report

Evaluator & Reviewer:
Mr. Purushotham
Professor AIML
JBIET

Submitter & Assignee:
L.Sumana,
22671A7385, AIML-B,
4th year, JBIET

Introduction

The objective of this project is to design and implement an interactive Image Processing Toolkit using a Streamlit-based GUI. The toolkit enables users to upload, process, and visualize images in real-time.

Tools & Technologies

Library	Purpose
Streamlit	Graphical User Interface (GUI)
OpenCV	Image Processing Operations
NumPy	Numerical Computations
Pillow	Image Handling & Manipulation

Implementation

The project consists of three main components:
• app.py: Streamlit-based GUI for image upload and processing.
• Jupyter Notebook: Contains function implementation details and explanations.
• Report: Explains workflow, results, and features.

Notes on CMOS vs CCD

1. CCD (Charge-Coupled Device)

- Captures light as charge in pixel wells and transfers it sequentially to a single output node.
- Produces **high image quality** with low noise and excellent sensitivity.
- Requires more power and external circuitry.
- Expensive and slower in readout.
- Used in **scientific, medical, and astronomy imaging**.

2. CMOS (Complementary Metal-Oxide Semiconductor)

- Each pixel has its own charge-to-voltage conversion with integrated amplification.
- Allows **on-chip integration** of ADCs, amplifiers, and processing.
- Provides **faster readout** and lower power consumption.
- Cheaper to manufacture, widely adopted.
- Used in **smartphones, robotics, automotive vision, surveillance**.

Feature	CCD	CMOS
Image Quality	High, low noise, better dynamic range	Good, but traditionally noisier (improved in modern CMOS)
Light Sensitivity	Better in low light	Lower (but improving with BSI – backside illumination)
Speed	Slower readout	Faster readout (supports high FPS)
Power Consumption	High	Low (more efficient)
Integration	Needs external components	Can integrate ADC, amplifiers, processors on-chip
Cost	Expensive	Cheaper, mass-producible
Use Cases	Astronomy, medical, scientific	Consumer devices, robotics, surveillance, automotive vision

Sampling

Sampling is the process of converting a continuous image (in the real world) into a discrete image by selecting pixel values at regular intervals in the spatial domain. The sampling rate determines the resolution of the image. According to the Nyquist sampling theorem, the sampling frequency must be at least twice the highest frequency present in the image to avoid aliasing. If under-sampled, the image may lose detail and appear distorted.

Quantization

Quantization refers to the process of mapping the continuous range of pixel intensity values (grayscale or color) into a finite set of discrete levels. For example, an 8-bit image uses 256 intensity levels (0–255). Higher quantization levels preserve more detail, while lower levels can cause **loss of information and false contours**. Together, **sampling + quantization** convert an analog image into a digital image suitable for computer vision applications.

Point Spread Function (PSF)

The Point Spread Function (PSF) describes how an imaging system responds to a single point source of light. Ideally, a point should appear as a single pixel, but due to imperfections such as lens blur, diffraction, and sensor limitations, it spreads over neighbouring pixels. The PSF characterizes this blur and plays a central role in understanding and modelling image degradation. In image restoration and deblurring, the observed image is modelled as the convolution of the original image with the PSF plus noise.

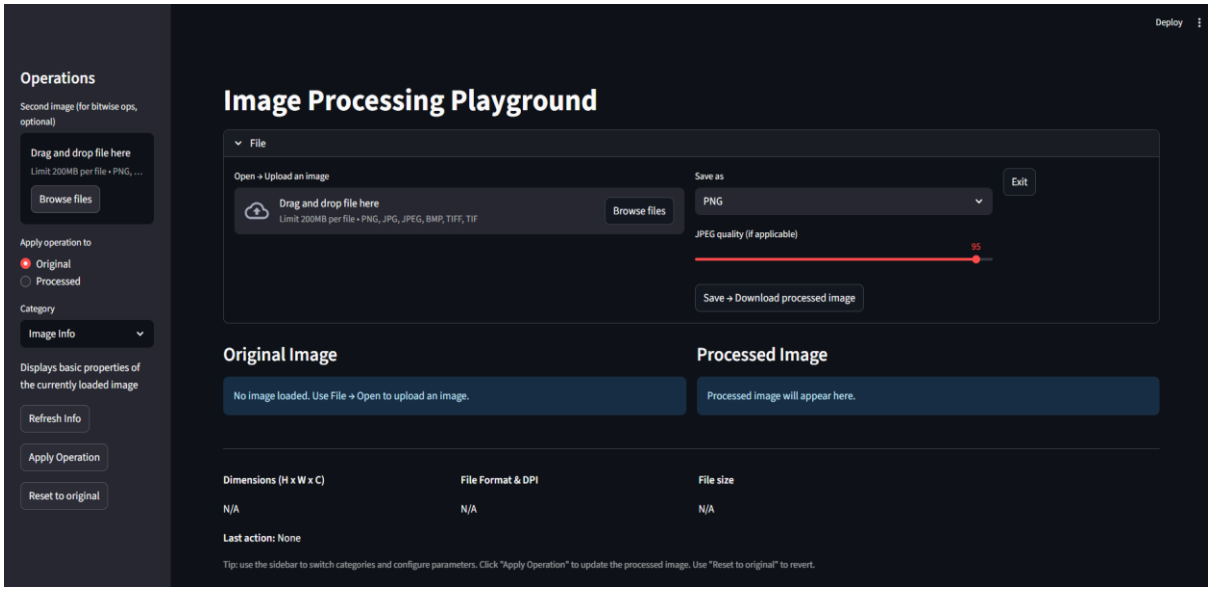
Key Takeaway:

- **Sampling** → determines spatial resolution (pixels).
- **Quantization** → determines intensity resolution (gray levels).
- **PSF** → models the blur and distortion introduced by the imaging system.

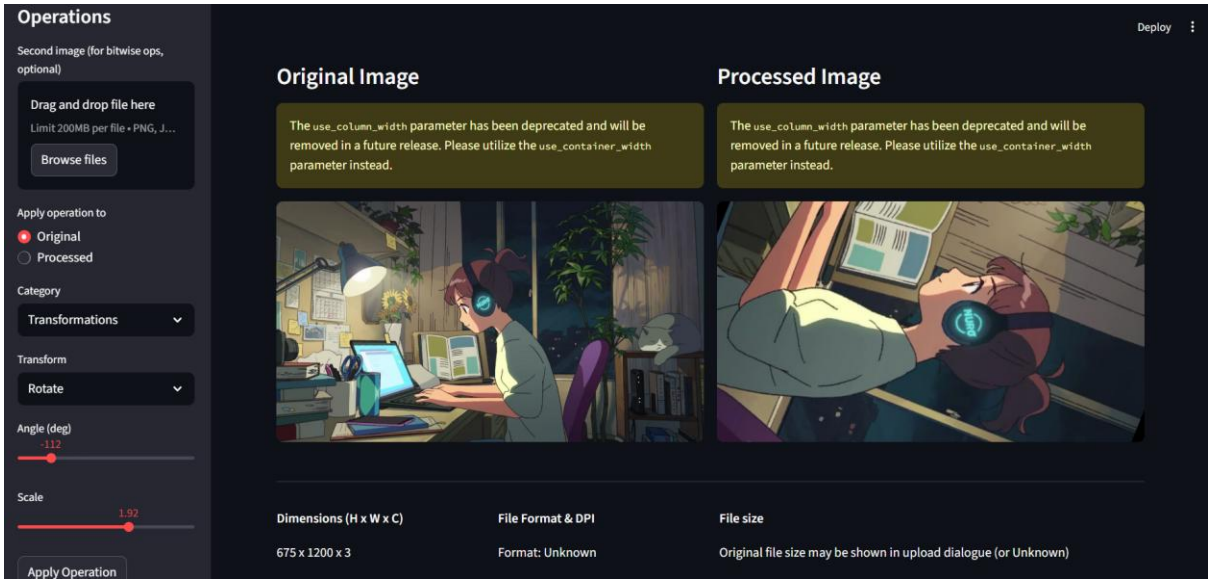
GUI Implementation and results

```
C:\Users\sumana lakka>pip install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.12.0.88-cp37-abi3-win_amd64.whl.metadata (19 kB)
Collecting numpy<2.3.0,>=2 (from opencv-python)
  Downloading numpy-2.2.6-cp313-cp313-win_amd64.whl.metadata (60 kB)
Downloading opencv_python-4.12.0.88-cp37-abi3-win_amd64.whl (39.0 MB)
   ---------------------------------------- 39.0/39.0 MB 8.7 MB/s  0:00:04
Downloading numpy-2.2.6-cp313-cp313-win_amd64.whl (12.6 MB)
   ---------------------------------------- 12.6/12.6 MB 9.8 MB/s  0:00:01
Installing collected packages: numpy, opencv-python
Successfully installed numpy-2.2.6 opencv-python-4.12.0.88

C:\Users\sumana lakka>pip install pillow
Collecting pillow
  Downloading pillow-11.3.0-cp313-cp313-win_amd64.whl.metadata (9.2 kB)
Downloading pillow-11.3.0-cp313-cp313-win_amd64.whl (7.0 MB)
   ---------------------------------------- 7.0/7.0 MB 11.0 MB/s  0:00:00
Installing collected packages: pillow
Successfully installed pillow-11.3.0

C:\Users\sumana lakka>pip install streamlit
Collecting streamlit
  Downloading streamlit-1.49.1-py3-none-any.whl.metadata (9.5 kB)
Collecting altair!=5.4.0,!=5.4.1,<6,>=4.0 (from streamlit)
  Downloading altair-5.5.0-py3-none-any.whl.metadata (11 kB)
Collecting blinker<2,>=1.5.0 (from streamlit)
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting cachetools<7,>=4.0 (from streamlit)
  Downloading cachetools-6.2.0-py3-none-any.whl.metadata (5.4 kB)
Collecting click<9,>=7.0 (from streamlit)
```

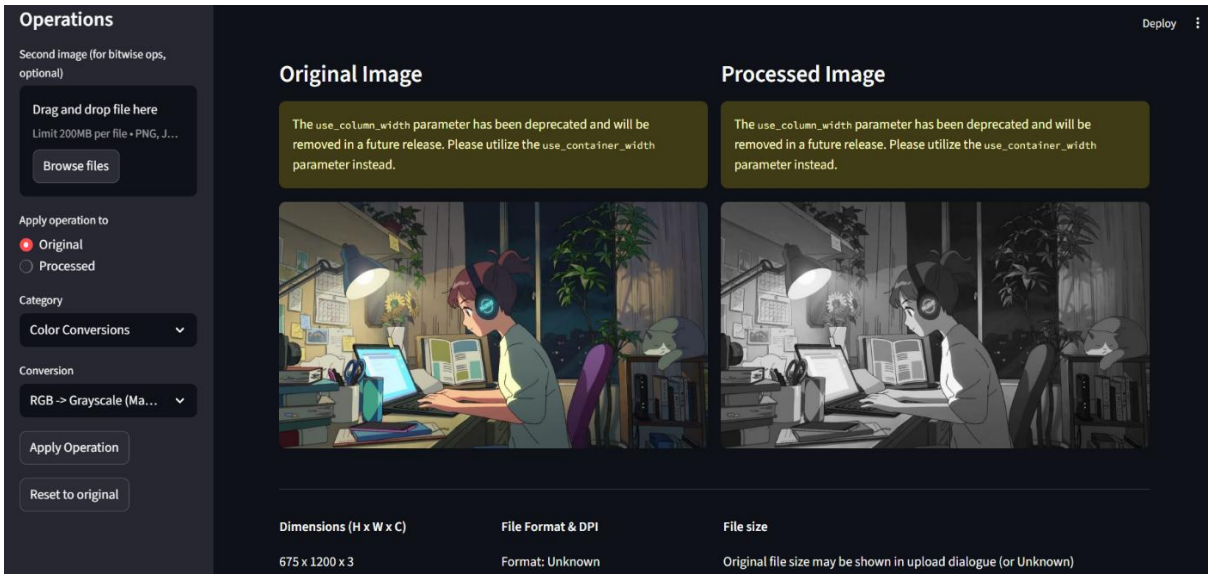
Fig(i). Package installation



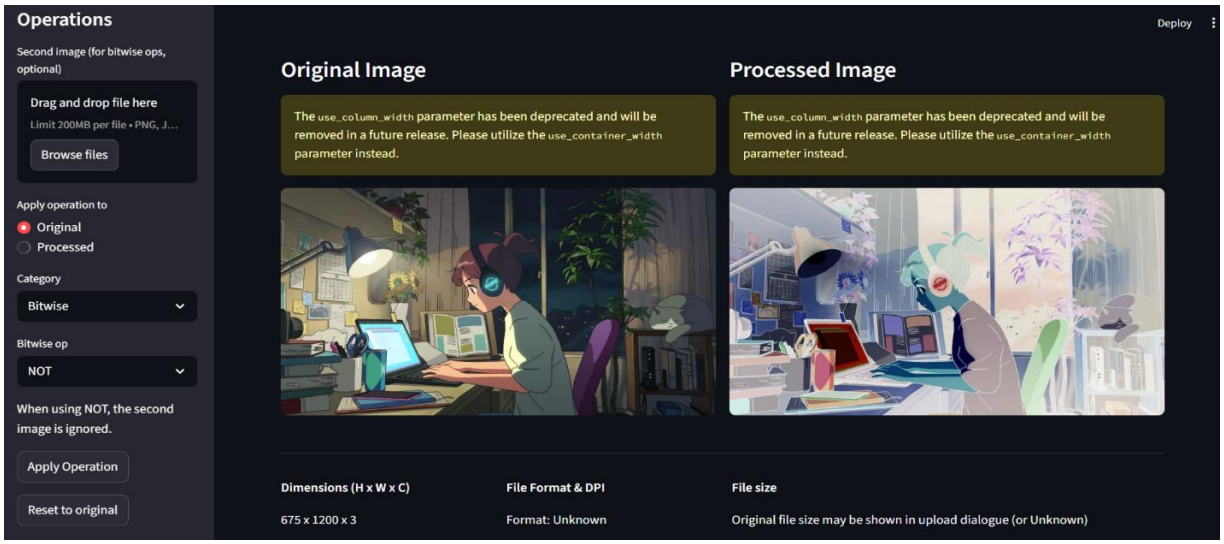
Fig(ii). Interactive GUI

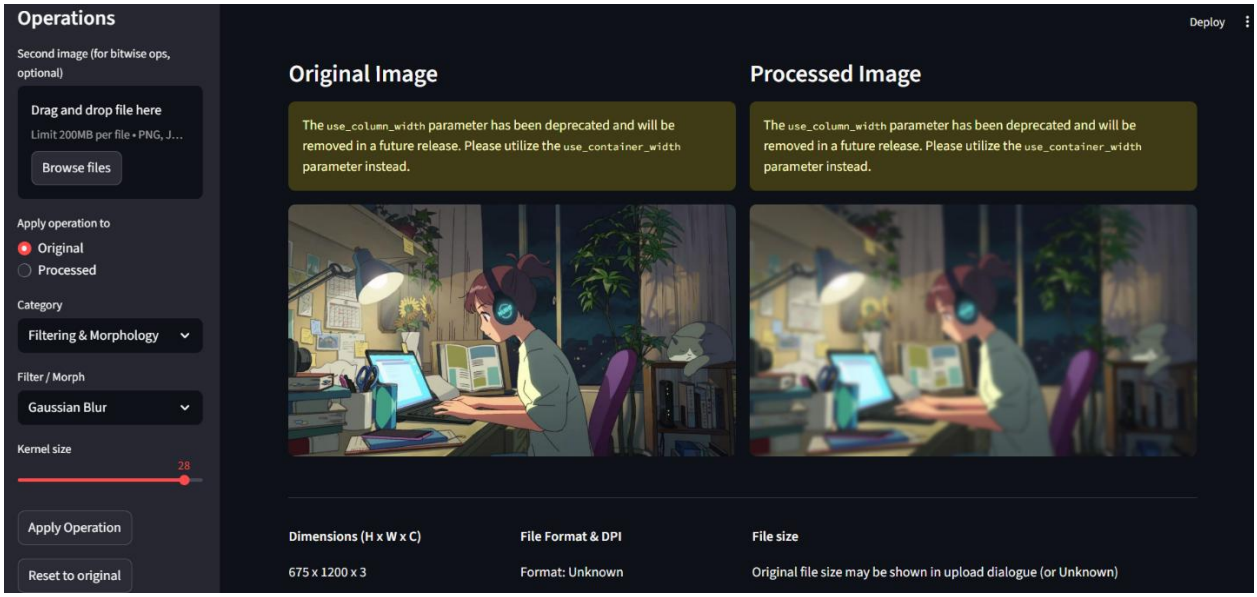


Fig(iii). Rotation & scaling – transformation

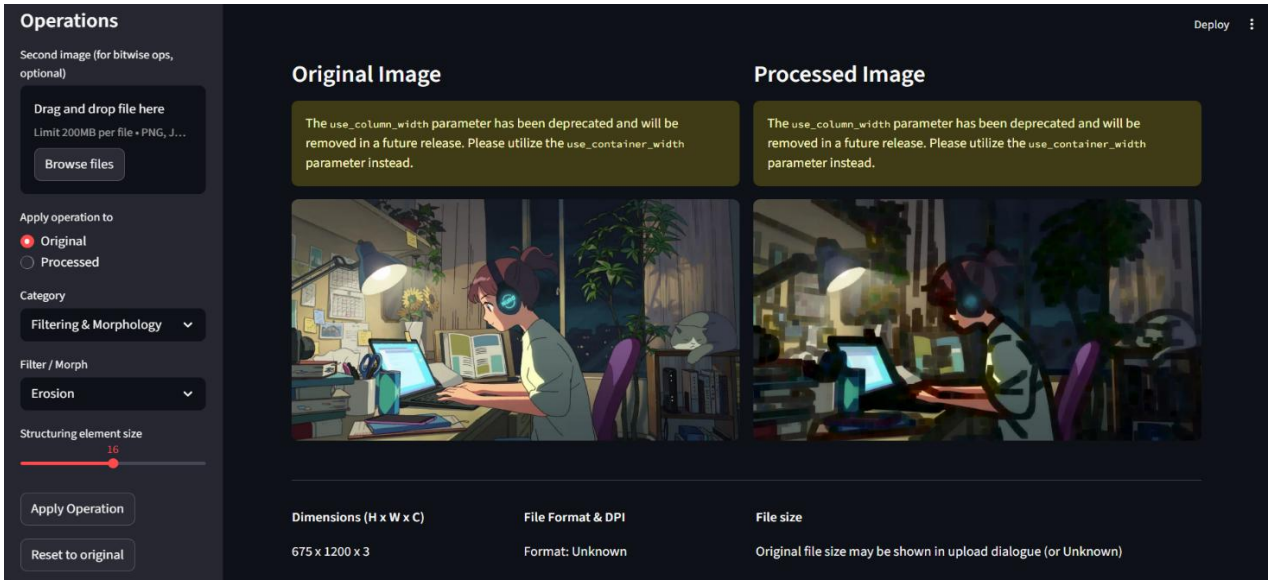


Fig(iv).RGB to Grayscale

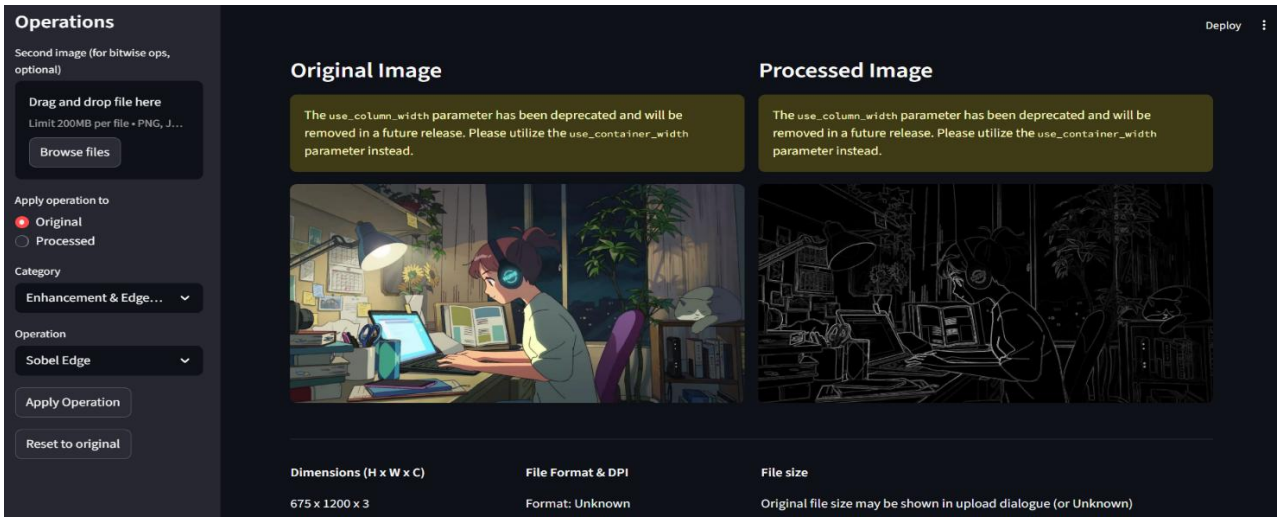


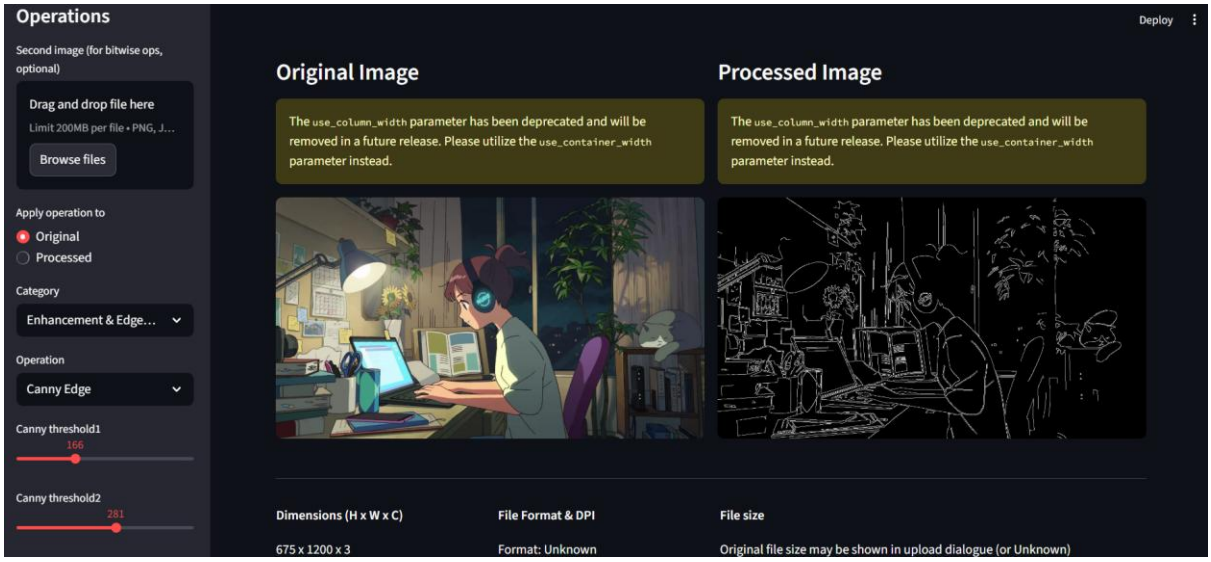


Fig(v). Gaussian Blur



Fig(vi). Erosion





Fig(vii). Canny Edge – Enhancement & Edge Detection

This interface allows interactive experimentation with image processing techniques. The user can upload an image, apply transformations such as rotation and scaling, and instantly visualize the processed result alongside the original, making it a practical tool for learning and demonstrating computer vision concepts.

Categories in Image Processing

1. Image Info

- Extracts basic details of an image.
- Examples:
  - Image size (width, height)
  - Format (PNG, JPEG, etc.)
  - Number of channels (grayscale = 1, color = 3)
  - Histogram of pixel intensities
- **Use case:** Helps understand image properties before processing.

---

2. Color Conversions

- Changing image color representations.
- Examples:
  - RGB ↔ Grayscale
  - RGB ↔ HSV (Hue, Saturation, Value)
  - RGB ↔ LAB
- **Why:** Some algorithms work better in certain color spaces (e.g., HSV is good for object tracking, LAB for color corrections).

---

3. Transformations

- Operations that change the geometry or position of an image.
- Examples:
  - Scaling (resize)
  - Rotation
  - Translation (shifting image left/right/up/down)
  - Affine Transform (rotation, scaling, skewing)
  - Perspective Transform (warp like a tilted view)
- **Use case:** Useful for alignment, data augmentation, and preprocessing.

---

4. Bitwise Operations

- Logical operations applied pixel by pixel.
- Examples:
  - **AND** → keeps overlapping white parts of two images
  - **OR** → combines two images
  - **XOR** → highlights non-overlapping areas
  - **NOT** → inverts pixel values
- **Use case:** Masking, region-of-interest (ROI) selection.

---

5. Filtering & Morphology

- **Filtering** → Enhances or removes details using convolution filters.
  - Blur, sharpen, edge detection, etc.
- **Morphology** → Shape-based operations (applied on binary images).
  - **Erosion:** shrinks white regions
  - **Dilation:** expands white regions
  - **Opening:** erosion followed by dilation (removes small noise)
  - **Closing:** dilation followed by erosion (fills small holes)
- **Use case:** Noise removal, object separation, shape analysis.

---

6. Enhancement & Edge Detection

- Improves image quality or highlights important structures.
- Examples:
  - Histogram equalization (improves contrast)
  - CLAHE (adaptive contrast enhancement)
  - Sobel/Laplacian edge detection
  - Canny edge detection
- **Use case:** Preprocessing before object detection, OCR, or medical imaging.

---

7. Compression & File Handling

- Reducing image size or changing format.
- Examples:
    - JPEG/PNG compression
    - Encoding/decoding images (base64, etc.)
    - Reading/writing image files
- **Use case:** Reduces storage size, speeds up transmission in applications.

Explanation of Algorithms Used

1. Grayscale Conversion

- **What it does:** Converts a colored image (RGB) into shades of gray.
- **How:** Calculates brightness from Red, Green, and Blue channels using weighted sum:

$$Gray = 0.299R + 0.587G + 0.114B$$

- **Why:** Reduces complexity by removing color information while keeping structure and intensity. Useful for tasks like edge detection.

---

2. Blurring (Gaussian Blur)

- **What it does:** Smooths the image by reducing noise and details.
- **How:** Each pixel is replaced by a **weighted average** of its neighbors, where closer pixels have more weight (Gaussian distribution).
- **Why:** Helps remove high-frequency noise and is often a preprocessing step before edge detection.

---

3. Canny Edge Detection

- **What it does:** Detects edges (boundaries between objects).
- **Steps:**
    1. Apply Gaussian Blur (to reduce noise).
    2. Compute gradients (Sobel operators) to find intensity changes.
    3. Apply **non-maximum suppression** (keep only the sharpest edges).
    4. Use **hysteresis thresholding** with two thresholds to finalize edges.
- **Why:** Very accurate and widely used edge detection algorithm.

---

4. Thresholding

- **What it does:** Converts a grayscale image into **binary black & white**.
- **How:** For a pixel value `p`:

$$p = \begin{cases} 255 & \text{if } p > T \\ 0 & \text{if } p \leq T \end{cases}$$

where `T` = threshold value.

- **Why:** Useful for separating objects from background (e.g., document scanning).

---

5. Rotation

- **What it does:** Rotates the image by a given angle.
- **How:** Uses a **rotation matrix**:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- **Why:** To adjust image orientation or for augmentation in AI datasets.

---

6. Flipping

- **What it does:** Mirrors the image **horizontally**, **vertically**, or both.
- **How:** Changes pixel positions by reversing indices:
  - Flip Code `0` → Vertical
  - Flip Code `1` → Horizontal
  - Flip Code `-1` → Both
- **Why:** Useful for augmentation and correcting mirrored images.

---

7. Brightness & Contrast Adjustment (if included)

- **What it does:** Makes the image brighter/darker or increases/decreases contrast.
- **How:** Applies a linear transformation:

$$new\_pixel = \alpha \times pixel + \beta$$

where **α** = contrast factor, **β** = brightness factor.

- **Why:** Enhances visibility and prepares data for further processing.

Conclusion

This project highlights the effectiveness of Python libraries like Streamlit and OpenCV for building interactive computer vision applications. It provides a user-friendly way to experiment with image processing and serves as a foundation for advanced AI-based vision projects.