**# Image Processing & Analysis Toolkit — Report**

**\*\*Student Roll:\*\*** 22671A7331

**\*\*Generated:\*\*** 2025-09-08 16:04:04

---

**## 1. Introduction**

Image processing is a foundational discipline in computer vision that transforms raw image data into information that is more useful for a particular task. This toolkit demonstrates fundamental operations — from color-space transforms and filters to geometric transforms, morphology, and edge detection — all accessible through a GUI and reproducible in notebook form.

This report explains the underlying theory as well as practical implementation notes for each module implemented in the delivered Streamlit app (`app.py`).

---

**## 2. Image Acquisition: CMOS vs CCD**

**\*\*CCD (Charge-Coupled Device)\*\*** and **\*\*CMOS (Complementary Metal–Oxide–Semiconductor)\*\*** are two types of image sensor technologies used in cameras:

- **\*\*CCD\*\*** sensors historically produced lower noise and higher image quality for still imaging, but were more power-hungry and expensive.

- **\*\*CMOS\*\*** sensors are cheaper, consume less power, and integrate more control circuitry on-chip; modern CMOS sensors have closed the quality gap and are dominant in consumer devices.

- **\*\*Practical implication:\*\*** Modern webcam/mobile sensors are CMOS; they may introduce pattern noise, hot pixels, and rolling-shutter effects. Some processing steps (demosaicing, noise reduction) are applied in-camera or in software pipelines.

---

**## 3. Sampling & Quantization**

- **\*\*Sampling\*\*** refers to how a continuous scene is discretized spatially (pixels). The sampling frequency (pixels per unit length) must be high enough to avoid aliasing — typically guided by the Nyquist theorem.

- **\*\*Quantization\*\*** refers to mapping sensor analog intensity values to discrete numerical levels (e.g., 8-bit, 0–255). Lower bit-depth increases quantization error and banding.

- **\*\*Practical step:\*\*** When reading images with OpenCV (`cv2.imread`) you obtain a sampled, quantized digital image. Consider converting to floating point for numeric processing and normalizing where necessary.

---

**## 4. Point Spread Function (PSF)**

- The PSF describes how a point source of light is blurred by the imaging system. Optical aberrations, focus, and motion produce PSFs that smear points into small disks or elongated shapes.

- Many restoration techniques (deconvolution) model the PSF to recover sharper images.

---

**## 5. Color Spaces & Conversions**

**\*\*Why multiple color spaces?\*\*** Some operations (thresholding, segmentation) are easier in one color space than another. For example, HSV separates chromatic content (hue) from intensity (value), making color segmentation simpler.

Common conversions used:

- RGB ↔ BGR (OpenCV uses BGR ordering)

- BGR ↔ HSV

- BGR ↔ YCrCb (useful for histogram equalization on luminance channel)

- BGR ↔ Grayscale (luminance extraction)

**\*\*Implementation note:\*\*** In the app code we use `cv2.cvtColor` for conversions, and demonstrate manual formulas inside the notebook for educational purposes.

---

**## 6. Geometric Transformations**

- **\*\*Translation\*\*** shifts pixel coordinates.

- **\*\*Scaling\*\*** resizes the image via interpolation (nearest, bilinear, bicubic).

- **\*\*Rotation\*\*** rotates around a center, requiring appropriate interpolation and handling of borders.

- **\*\*Affine transform\*\*** preserves lines and parallelism (but not necessarily angles / lengths). Parameterized by a 2×3 matrix computed from 3 point correspondences.

- **\*\*Perspective transform\*\*** maps quadrilaterals to quadrilaterals using a 3×3 homography; used for document rectification.

Implementation uses OpenCV functions (`warpAffine`, `warpPerspective`) and computed matrices (`getRotationMatrix2D`, `getAffineTransform`, `getPerspectiveTransform`).

---

**## 7. Filtering & Morphology**

**\*\*Smoothing / denoising:\*\***

- **\*\*Mean (box) filter:\*\*** simple averaging over kernel; reduces random noise but blurs edges.

- **\*\*Gaussian filter:\*\*** weighted averaging using Gaussian kernel; better preserves smooth transitions.

- **\*\*Median filter:\*\*** replaces each pixel by median in neighborhood; excellent for salt-and-pepper noise.

**\*\*Morphological operations:\*\***

- **\*\*Erosion:\*\*** shrinks bright regions; removes small bright artifacts.

- **\*\*Dilation:\*\*** grows bright regions; fills small holes.

- **\*\*Opening:\*\*** erosion followed by dilation — useful for removing small objects.

- **\*\*Closing:\*\*** dilation followed by erosion — fill small holes in objects.

Mathematical definitions and examples are included in the notebook.

---

**## 8. Edge Detection & Enhancement**

**\*\*Sobel operator:\*\*** computes image gradients in x and y; gradient magnitude approximates edge strength.

**\*\*Laplacian operator:\*\*** second-order derivative operator sensitive to rapid changes; can produce double edges.

**\*\*Canny edge detector:\*\*** multi-step detector (Gaussian smoothing → gradient computation → non-maximum suppression → hysteresis thresholding).

**\*\*Histogram equalization:\*\*** redistributes intensities to enhance contrast (global equalization).

**\*\*Contrast stretching:\*\*** linear remapping between percentiles to expand dynamic range.

**\*\*Sharpening:\*\*** using unsharp masking or a high-boost kernel (e.g., [[0,-1,0],[-1,5,-1],[0,-1,0]]).

---

**## 9. Compression & File Formats**

- **\*\*PNG:\*\*** lossless compression, larger file size for natural images but preserves detail.

- **\*\*JPEG:\*\*** lossy, adjustable quality parameter; much smaller file sizes with perceptual loss.

- **\*\*BMP:\*\*** uncompressed (or simple header-based), large files.

The app allows saving in multiple formats and compares sizes.

---

**## 10. Realtime Webcam Processing (Notes)**

- Processing every webcam frame needs careful optimization; use smaller frame sizes or GPU acceleration for heavy operations.

- For interactive sliders that modify processing, prefer lightweight operations (grayscale, small kernels) to keep frame rate high.

---

**## 11. How to use the delivered files**

**\*\*Files generated:\*\***

- `app.py` (Streamlit app)

- `ImageToolkit\_22671A7331.ipynb` (this interactive notebook)

- `report\_22671A7331.pdf` (this report)

**\*\*To run the notebook locally:\*\***

1. Install dependencies: `pip install notebook opencv-python numpy pillow matplotlib`

2. Launch: `jupyter notebook ImageToolkit\_22671A7331.ipynb`

**\*\*To run the GUI:\*\***

1. `pip install streamlit opencv-python numpy pillow`

2. `streamlit run app.py`





