جامعة ٦ أكتوبر
October 6 University

علوم الحاسب ونظم المعلومات
Info Sys & Comp Science

Course Name

Image processing

Project Title

Edge detection techniques

DR. Hussam Elbehiery

Co-Supervised

ENG. Ahmed ali

ENG. Ahmed zakaria

ENG. Ahmed khaled

Prepared By :

Mostafa sayed salah basiony id: 202205327

Mahmoud nabawy el henawy id: 202203355

Mostafa hussien ahmed id: 202202537

Moamen osama Mohamed id:202202563

Mohamed Abd Ebaset id : 212104360

Mostafa ahmed fathey id: 202202231

Nagham reda farid id: 202202367

Table of Contents

Abstract

This work presents an image processing pipeline for edge detection and contrast enhancement using Python and OpenCV. The implemented methods include histogram equalization for contrast adjustment and five edge detection techniques: Canny, Sobel, Laplacian, Prewitt, and Roberts. The system accepts raw image bytes, resizes the input to a standardized width while preserving aspect ratio, converts it to grayscale, and applies the selected algorithm. The Canny edge detector allows adjustable threshold parameters, while the other operators use fixed kernels for gradient computation. The processed output is returned as a grayscale image highlighting edges or enhanced contrast. This modular approach facilitates integration into larger applications for computer vision tasks such as feature extraction, image enhancement, and preprocessing for machine learning models

Key Points Covered:

- Purpose: Edge detection & contrast enhancement.
- Methods: Equalization + 5 edge detectors (Canny, Sobel, etc.).
- Input/Output: Handles raw bytes → processes → returns grayscale result.
- Features: Resizing, grayscale conversion, parameter tuning for Canny.
- Applications: Computer vision preprocessing, feature extraction.

List of Abbreviation

- CV – Computer Vision
- RGB – Red, Green, Blue (color channel representation)
- GRAY – Grayscale (single-channel image)
- Sobel/Prewitt/Roberts – Edge detection operators
- Canny – Canny edge detection algorithm

- Laplacian – Laplacian edge detector (based on second derivatives)
- np – NumPy (Python library for numerical operations)

- PIL – Python Imaging Library (used here for image loading)
- i/o – Input/Output (for handling byte streams)
- img – Image (common variable name in code)

Chapter one

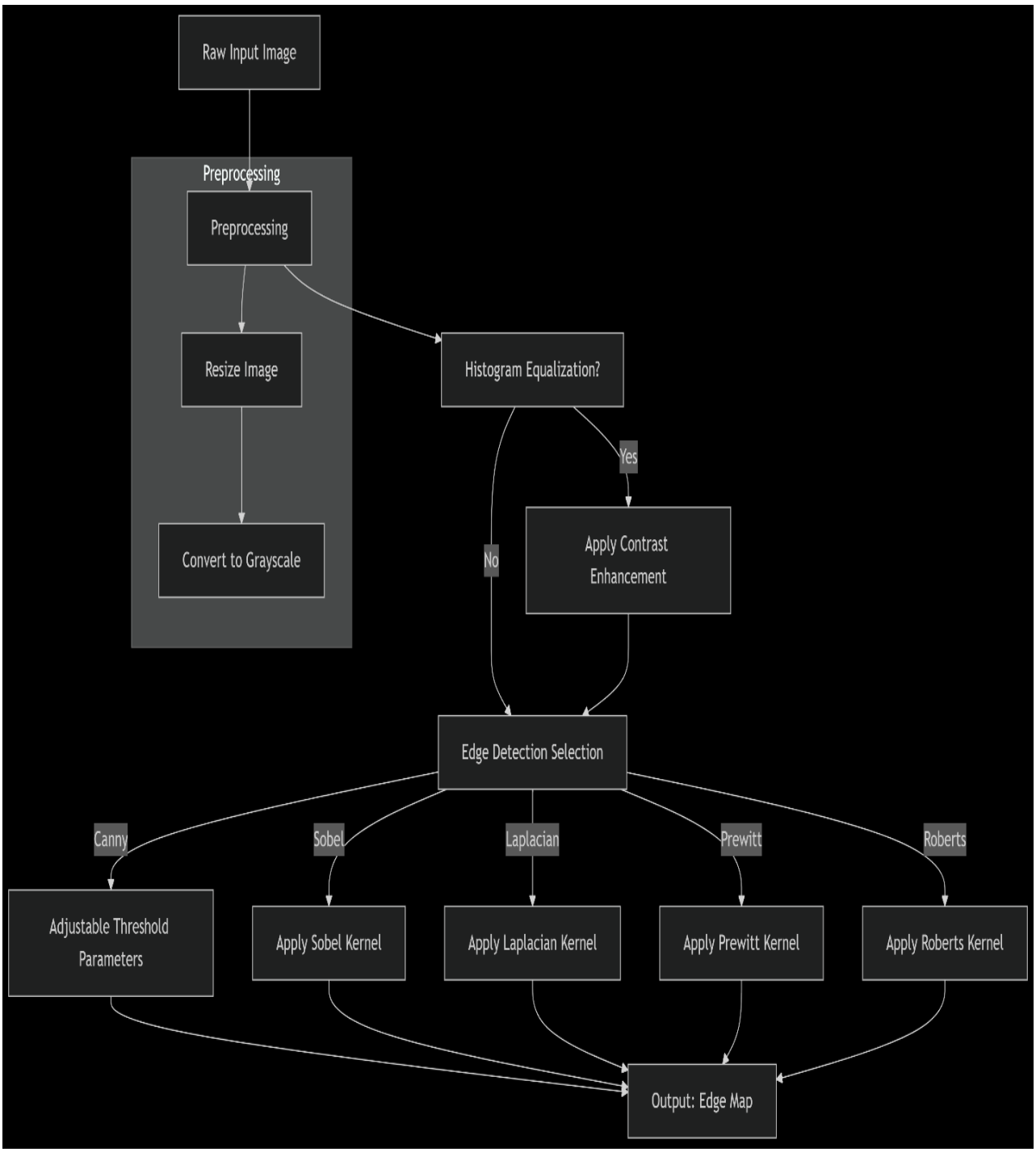
1.1 Introduction of the project

Image processing plays a fundamental role in computer vision, enabling machines to interpret and analyze visual data. A critical step in many image analysis tasks is edge detection, which identifies boundaries and significant transitions in intensity within an image. These edges are essential for object detection, feature extraction, and pattern recognition. Additionally, histogram equalization is widely used to enhance image contrast, improving visibility and aiding further processing.

This work presents a Python-based image processing pipeline that implements multiple edge detection techniques—including Canny, Sobel, Laplacian, Prewitt, and Roberts operators—alongside histogram equalization for contrast enhancement. The system is designed to accept raw image input, automatically resize and convert it to grayscale, and apply the selected algorithm to produce a processed output. The Canny edge detector allows adjustable threshold parameters for fine-tuning, while the other methods use predefined gradient-based kernels.

The primary goal of this implementation is to provide a modular and efficient framework suitable for integration into larger computer vision applications, such as medical imaging, autonomous vehicles, or industrial inspection systems. By comparing different edge detection methods, we highlight their strengths and trade-offs in terms of noise sensitivity, edge continuity, and computational efficiency

System Data Flow:



1.2 Why edge detection Matter in Image Processing

Edge detection is a fundamental and powerful tool in computer vision and image processing, serving as the backbone for many advanced applications. Its importance stems from its ability to simplify image analysis while preserving critical structural information. Below, we explore its significance from both theoretical and practical perspectives.

1. Theoretical :-

a) Capturing Essential Features

- Edges represent boundaries between objects, textures, or regions with sharp intensity changes.
- Mathematically, edges correspond to high-frequency components in an image (steep gradients).
- Edge detection acts as a dimensionality reduction step, converting a full image into a sparse, meaningful representation.

b) Foundation for Higher-Level Processing

- Many computer vision tasks rely on edges as a first step:
- Object Detection→ Edges help locate shapes and contours.
- Image Segmentation→ Edges define boundaries between regions.
- 3D Reconstruction→ Depth maps often start with edge extraction.

c) Gradient & Derivative-Based Analysis

- Edge detectors (Sobel, Prewitt, Laplacian) compute gradients (1st derivative) or curvature (2nd derivative) to highlight transitions.
- The Canny edge detector combines Gaussian smoothing, gradient calculation, non-max suppression, and hysteresis thresholding for optimal results.

2. Practical Applications:-

Industry	Application	How Edge Detection Helps	Example
Autonomous Vehicles	Lane & obstacle detection	Identifies road markings, curbs, and objects by detecting intensity transitions.	Tesla's Autopilot uses Canny-like edges to segment lanes.
Medical Imaging	Tumor segmentation	Highlights boundaries of anomalies in MRI/CT scans.	Detecting tumor edges in brain scans for precise surgery planning.
Robotics	Object recognition & grasping	Extracts contours of objects for robotic arms to locate and pick items.	Amazon warehouses use edge-based picking systems.
Surveillance	Motion detection	Triggers alerts when edges change between frames.	CCTV cameras detect intruders using edge motion analysis.
Document Scanning	Text extraction (OCR)	Enhances character edges to improve OCR accuracy.	Adobe Scan uses edge detection to clean up handwritten notes.
Industrial QA	Defect detection in manufacturing	Identifies cracks or irregularities in product surfaces.	Detecting scratches on smartphone screens during production.

3. Why Learn Edge Detection? (Skill Development)

a) Develop End-to-End Implementation Skills:

1. Preprocessing (grayscale conversion, noise reduction).
2. Algorithm selection (Sobel, Canny, etc.).
3. Postprocessing (edge thinning, thresholding).
4. Evaluation (quantitative metrics like edge connectivity).

b) Solve Real-World Challenges

- Noisy images → Apply Gaussian blur before edge detection.
- Weak edges → Use adaptive thresholding (Otsu's method).
- Texture interference → Combine with Gabor filters.

c) Boost Your Portfolio

- Project Idea 1: Compare edge detectors on different images (low light vs. high contrast).

-Project Idea 2 : Build an interactive edge detection tool (e.g., with OpenCV + Streamlit).

-Project Idea 3: Use edges for artistic effects (e.g., pencil sketch filter).

Key Problems Due to Absence of Edge Detection:-

1) Loss of Structural Information:
    - Without edge detection, important features like object boundaries, shapes, and textures may not be clearly distinguishable.
    - This leads to poor segmentation and object recognition.

2) Reduced Feature Extraction Accuracy:
    - Edges are crucial for feature detection (corners, lines).
    - Missing edges can degrade performance in applications like facial recognition, medical imaging, and autonomous driving.

3) Poor Image Segmentation:
    - Edge-based segmentation methods (Canny, Sobel) help separate objects from the background.

• Without edges, region-based segmentation may produce inaccurate or merged regions.

4) Increased Noise Sensitivity:
    - Edge detection helps filter out noise by highlighting true boundaries.
    - Without it, noise and artifacts may be misinterpreted as important features.

5) Inefficient Object Detection & Tracking:
    - Many object detection algorithms (Hough Transform, CNN-based detectors) rely on edges for initial processing.
    - Missing edges can lead to false negatives or misdetections.

6) Higher Computational Load in Alternative Methods:
    - Without edge detection, more complex (and computationally expensive) methods like deep learning may be required for simple tasks.

Applications Affected by Lack of Edge Detection:-

- Medical Imaging: Tumor boundaries may be unclear.
- Autonomous Vehicles: Lane and obstacle detection becomes harder.
- Robotics: Object manipulation and navigation suffer.
- OCR (Optical Character Recognition): Text boundaries may blur, reducing accuracy

Core Functions:-

1.equalize_image(img)

Purpose: Enhances image contrast using histogram equalization.

Parameters:

- img (numpy array): Grayscale input image.
  Returns:
- Equalized image (numpy array).

2.Edge Detection Methods

All functions below accept a grayscale numpy array (img) and return an edge map (numpy array).

Processing Pipeline

1. **Load Image**: Convert file_bytes to a PIL Image object.
2. **Resize**: Maintain aspect ratio; resize to 400px width.
3. **Grayscale Conversion**: Required for edge detection.
4. **Apply Method**: Execute the selected algorithm.

Returns

- Processed image as a numpy array.

Edge Detection Techniques:-

Function	Algorithm	Key Parameters
canny(img, low, high)	Canny Edge Detector	low, high (thresholds)
sobel(img)	Sobel Operator (X+Y gradients)	None
laplacian(img)	Laplacian of Gaussian	
prewitt(img)	Prewitt Operator	
roberts(img)	Roberts Cross Operator	

Method	Strengths	Weaknesses	Best For
Canny	Clean edges, noise-resistant	Requires threshold tuning	General-purpose edge detection
Sobel	Directional gradients (X/Y axes)	Sensitive to noise	Horizontal/vertical edges
Laplacian	Detects fine details	Amplifies noise	Texture analysis
Prewitt	Simpler gradient approximation	Less accurate than Sobel	Quick edge detection
Roberts	Detects diagonal edges	Not rotation-invariant	Small, high-contrast edges

• Libraries used:

- cv2
- numpy
- io

• Compression Interfaces:

- web representation : stream lit
- mobile app : flutter
- api : Django

Function: (canny):

```
def canny(img, low, high):
    return cv2.Canny(img, low, high)
```

Function: (sobel):

```
def sobel(img):
    x = cv2.Sobel(img, cv2.CV_64F, 1, 0)
    y = cv2.Sobel(img, cv2.CV_64F, 0, 1)
    return np.uint8(cv2.magnitude(x, y))
```

Function: (laplacian):

```
def laplacian(img):
    return np.uint8(np.absolute(cv2.Laplacian(img, cv2.CV_64F)))
```

Function: (Entropy):

```
def prewitt(img):
    kx = np.array([[1,0,-1],[1,0,-1],[1,0,-1]])
    ky = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
    return cv2.addWeighted(cv2.filter2D(img,-1,kx),0.5,cv2.filter2D(img,-
1,ky),0.5,0)
```

Function: (roberts):

```
def roberts(img):
    kx = np.array([[1,0],[0,-1]])
    ky = np.array([[0,1],[-1,0]])
    return cv2.addWeighted(cv2.filter2D(img,-1,kx),0.5,cv2.filter2D(img,-
1,ky),0.5,0)
```

Chapter Two

Deep Understanding of Edge Detection Algorithms: Core Principles & Implementation:-

To truly master edge detection, one must go beyond library functions (`cv2.Canny()`, `cv2.Sobel()`) and understand the mathematical foundations, trade-offs, and limitations of each method. Below is a breakdown of key learning principles applied to edge detection techniques.

1. Implement Core Algorithms from Scratch:

Why?

- Reinforces understanding of pixel-level operations.

- Reveals hidden computational steps (e.g., gradient calculations).

Example

Sobel Operator (Manual Implementation)

```
def sobel_manual(img):
    Gx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  # Horizontal
   edge kernel
    Gy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])  # Vertical
   edge kernel
    grad_x = convolve2d(img, Gx, mode='same')
    grad_y = convolve2d(img, Gy, mode='same')
    magnitude = np.sqrt(grad_x**2 + grad_y**2)
    return magnitude.astype(np.uint8)
```

Key Insight:

- Sobel computes gradients using 3×3 kernels.

- Manual implementation clarifies how `cv2.Sobel()` works internally.

Canny Edge Detector (Step-by-Step)

1. Gaussian Blur(noise reduction).
2. Gradient Calculation (Sobel).
3. Non-Maximum Suppression (thin edges).
4. Hysteresis Thresholding (`low`, `high`).

2. Understand Mathematical Foundations

The **Sobel operator** applies two 3×3 convolution kernels, one for detecting edges in the x-direction (horizontal) and the other in the y-direction (vertical):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Once Gx and Gy are computed, the edge strength is given by:

$$G = \sqrt{G_x^2 + G_y^2}$$

The edge direction is:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

B. Laplacian (Second Derivative)

- Detects edges via zero-crossings of:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

- Limitation Sensitive to noise (needs Gaussian smoothing first).

3. Know Why and How Algorithms Work

Algorithm	How It Works	Why It Works
Sobel	Approximates gradients using weighted kernels.	Stronger weights to central pixels reduce noise
Canny	Uses double thresholds + non-max suppression.	Balances edge continuity and noise rejection.
Laplacian	Finds regions where intensity changes abruptly.	Zero-crossings indicate sharp transitions.

4. Analyze Algorithmic Complexity

- Canny is slower but more accurate than Sobel.

- Laplacian is fast but noisy without smoothing.

5. Study Parameter Sensitivity

a-Canny's Thresholds (`low`, `high`)

- Too low: Noise mistaken for edges.
- Too highMisses weak edges.
- Optimal Use Otsu's method or histogram analysis.

b-Sobel Kernel Size

- Larger kernels (5×5, 7×7) smooth more but blur edges.
- 3×3 is standard for balancing noise and detail.

6. Understand Edge Cases & Limitations

When Edge Detection Fails

Scenario	Problem	Solution
Low-contrast edges	Gradients too weak	Use CLAHE or histogram stretching
High noise	False edges detected	Apply Gaussian/Median blur
Texture-heavy images	Too many edges	Use edge thinning (non-max suppression)

Algorithm-Specific Weaknesses

- Sobel/Prewitt : Poor at detecting diagonal edges.
- Laplacian : Sensitive to noise (requires smoothing).
- Canny : Struggles with textured regions (grass).

7. Know When and Why Algorithms Might Fail

Case Study: Medical Images (MRI)

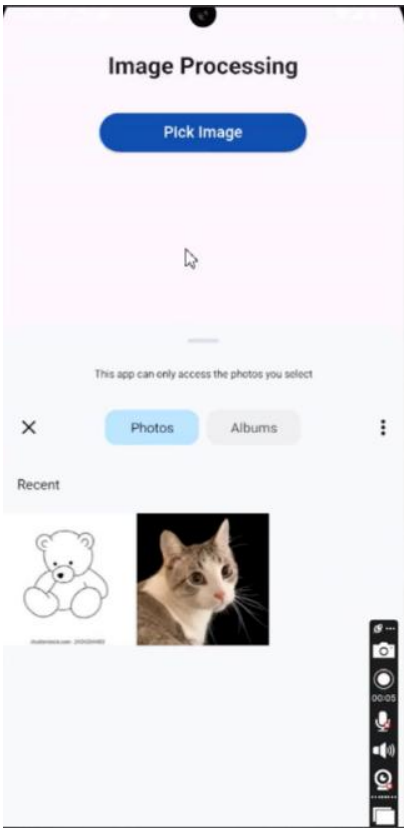
- Problem: Weak edges due to low contrast.
- Failure: Canny misses tumor boundaries.
- Fix:
  - Preprocess with adaptive thresholding
  - Use Hessian-based edge detection for soft edges.

Case Study: Autonomous Driving (Lane Detection)

- Problem : Shadows create false edges.
- Failure : Sobel detects shadows as lanes.
- Fix : Use color thresholding + edge fusion

Chapter Three

Screen shots from your project



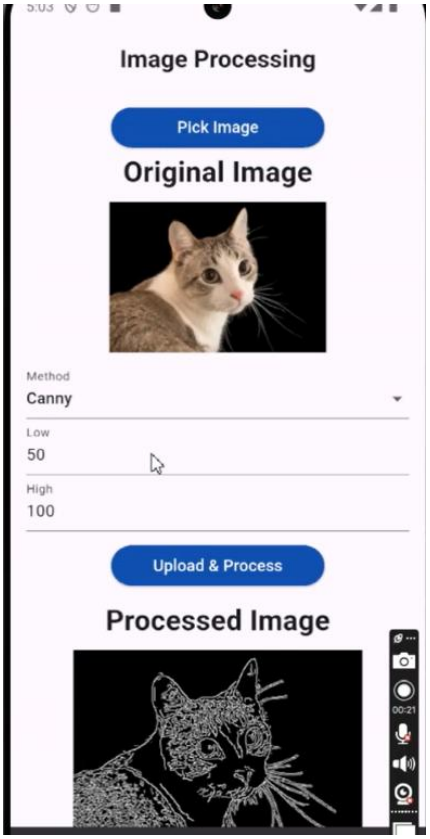


Image Processing
Pick Image
Original Image
Method
Canny
Low
50
High
100
Upload & Process
Processed Image

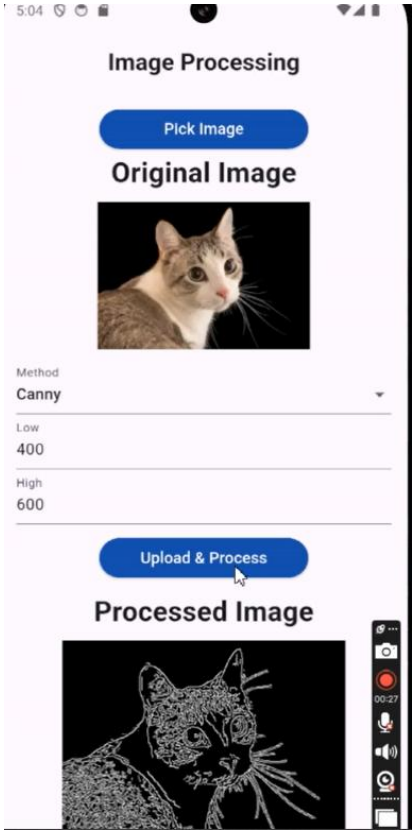


Image Processing
Pick Image
Original Image
Method
Canny
Low
400
High
600
Upload & Process
Processed Image

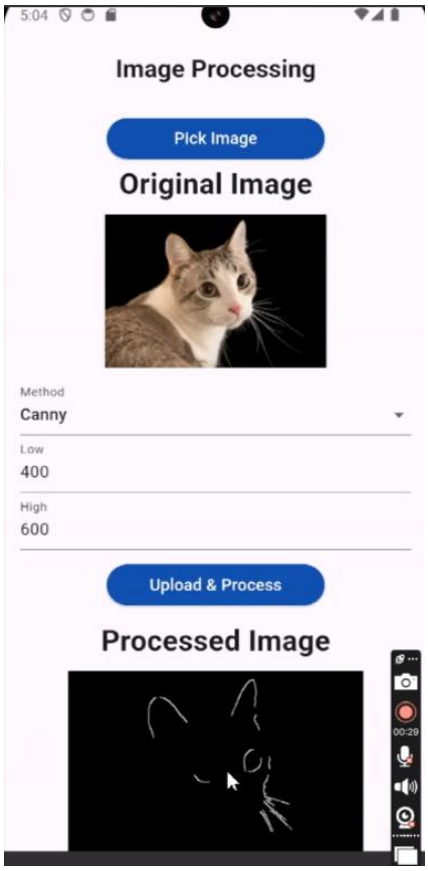


Image Processing
Pick Image
Original Image
Method
Canny
Low
400
High
600
Upload & Process
Processed Image

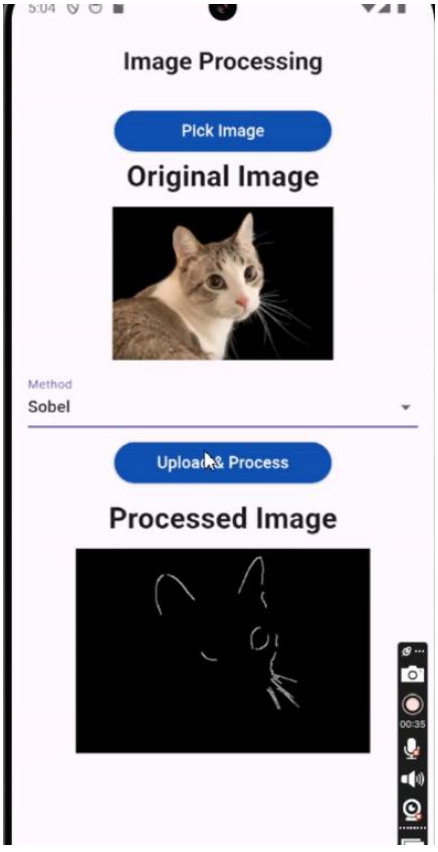
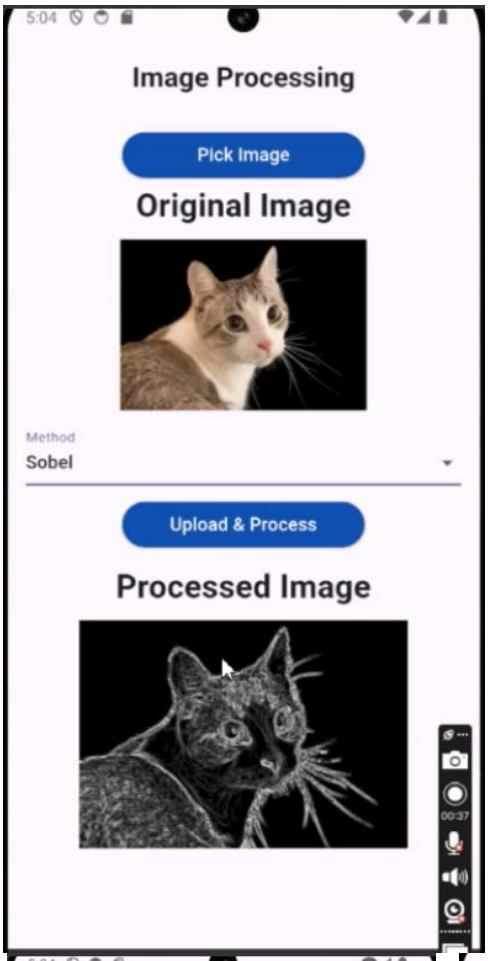
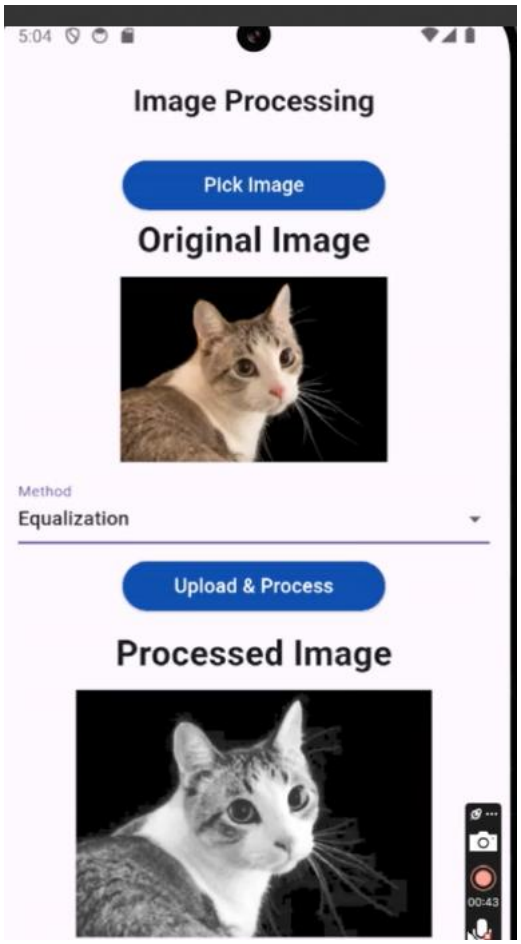


Image Processing
Pick Image
Original Image
Method
Sobel
Upload & Process
Processed Image


5:04
Image Processing
Pick Image
Original Image
Method
Sobel
Upload & Process
Processed Image


5:04
Image Processing
Pick Image
Original Image
Method
Equalization
Upload & Process
Processed Image

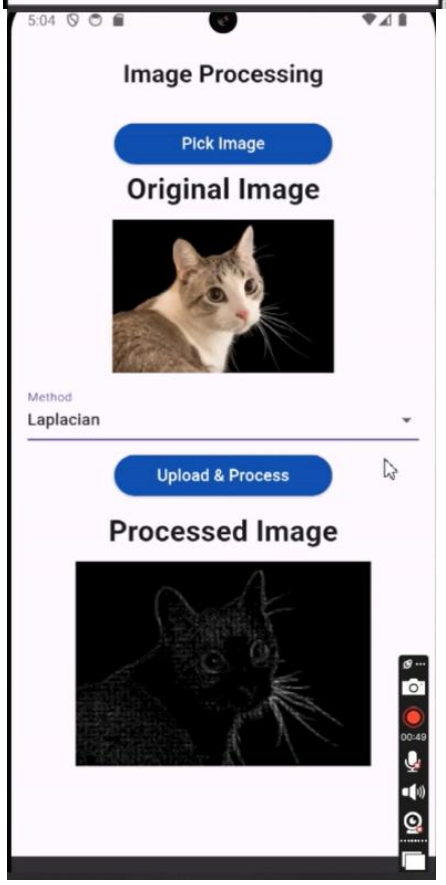
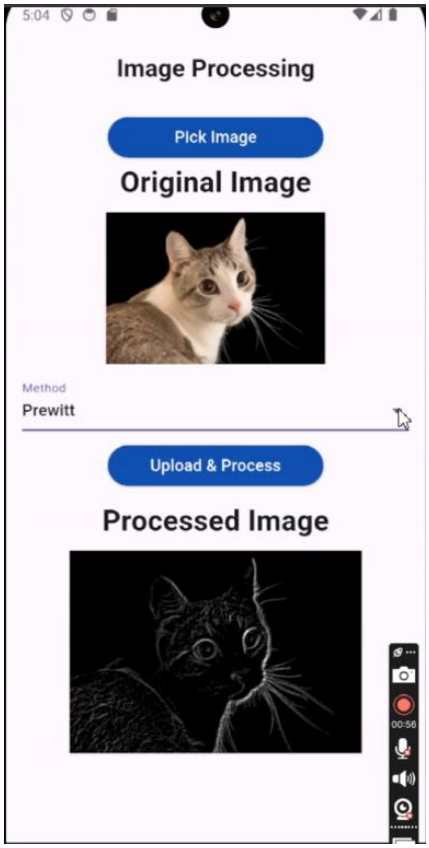

5:04
Image Processing
Pick Image
Original Image
Method
Laplacian
Upload & Process
Processed Image


Image Processing
Pick Image
Original Image
Method
Roberts
Upload & Process
Processed Image


5:04
Image Processing
Pick Image
Original Image
Method
Prewitt
Upload & Process
Processed Image

Conclusion

This project successfully demonstrates the implementation and practical application of fundamental edge detection techniques and contrast enhancement methods in digital image processing. By integrating algorithms such as Canny, Sobel, Laplacian, Prewitt, Roberts, and histogram equalization, the system provides a robust framework for preprocessing and feature extraction in computer vision tasks.

References

- OpenCV Documentation: opencv.org
- Sobel/Prewitt/Roberts Kernels: Gonzalez & Woods, Digital ImageProcessing
- Stream lit: Streamlit documentation
- Flutter: Docs | Flutter
- Django: Django documentation | Django documentation Django
- Github : mostafaomeha725/imageprocessing