Image Processing Application (Canny, Blur, Sobel, Frequency Filters)

Overview

This program uses OpenCV to capture live video from a webcam and apply different image processing techniques in real-time. The user can switch between spatial domain (e.g., blur, Sobel, Canny edge detection) and frequency domain (e.g., low-pass, high-pass, bandpass, bandreject) filters interactively via keyboard controls.

1. Spatial Domain Functions

- **NoisRedeuction(img, Kernalsize)** Applies a Gaussian blur to an image to reduce noise before edge detection. *Parameters:*
    1. img: Input grayscale or color image.
    2. Kernalsize: Tuple (e.g., (5,5)) defining the blur size. *Returns:* Blurred image. *Explanation:* Gaussian blur smooths variations and suppresses small unwanted edges.
- **Gradient(img)** Computes image gradients using the Sobel operator to measure intensity change in the x and y directions. *Returns:*
    1. grad_magnitude: Strength of edges.
    2. grad_direction: Direction (orientation) of the edge.
    3. grad_magnitude_disp: Normalized magnitude for display. *Process:*
    4. Calculate horizontal and vertical derivatives using cv2.Sobel.
    5. Compute magnitude and direction using: magnitude = $\sqrt{(Gx^2 + Gy^2)}$, direction = arctan2(Gy, Gx).
    6. Normalize both for visualization.
- **non_maximum_suppression(grad_mag, grad_angle)** Performs edge thinning by keeping only local maxima along gradient direction — an essential step in Canny edge detection. *Inputs:*
    1. Gradient magnitude and direction (in radians). *Returns:* Suppressed gradient matrix where only pixel-local maxima are retained.
- **double_threshold(image, low_threshold, high_threshold)** Classifies pixels into strong, weak, or non-edge based on thresholds. *Returns:* Two boolean masks:
    1. strong_edges: Pixels above high_threshold.
    2. weak_edges: Pixels between thresholds.
- **edge_tracking_by_hysteresis(strong_edges, weak_edges)** Connects weak edges to strong ones if they are adjacent, forming continuous edges. *Returns:* Binary edge image (strong edges kept, isolated weak edges discarded).

- **auto_threshold(image)** Automatically determines Canny thresholds based on image brightness and contrast. *Method:*
    1. Computes mean and standard deviation.
    2. Sets thresholds using: L = mean - 0.5σ, H = mean + 1.5σ.
    3. Clamps them to safe ranges for robustness.
- **Canny(image, kernel_size, low_thresh=None, high_thresh=None, auto_thresh=True)** Full Canny edge detector pipeline built from scratch. *Steps:*
    1. Noise reduction (Gaussian blur).
    2. Gradient computation.
    3. Non-maximum suppression.
    4. Thresholding (manual or auto).
    5. Hysteresis tracking. *Returns:* (edges, low_threshold, high_threshold)

2. Frequency Domain Functions

These functions perform filtering in the Fourier domain, manipulating spatial frequencies.

- **to_freq(img)** Converts a grayscale image to its frequency representation using FFT. *Process:* Applies np.fft.fft2, then shifts the zero frequency to the image center with np.fft.fftshift.
- **from_freq(f_shift)** Converts a frequency-domain image back to the spatial domain. *Steps:*
    1. Undo shift (ifftshift).
    2. Perform inverse FFT.
    3. Take magnitude and normalize to 0–255.
- **make_filter_mask(shape, ftype='ideal', mode='low', D0=30, n=2, W=20)** Creates 2D frequency filter masks for ideal, Gaussian, and Butterworth filters. *Modes:*
    1. 'low' → Low-pass filter (keeps smooth areas).
    2. 'high' → High-pass filter (keeps edges/details).
    3. 'bandreject' → Rejects a band of frequencies.
    4. 'bandpass' → Keeps only a specific band. *Parameters:*
    5. D0: Cutoff frequency or band center radius.
    6. n: Filter order (for Butterworth).
    7. W: Width for band filters.
- **freq_filter(img, ftype='ideal', mode='low', D0=30, n=2, W=20)** Applies the chosen frequency-domain filter to an image. *Process:*
    1. Convert image to frequency domain (to_freq).
    2. Multiply with the filter mask.
    3. Convert back using from_freq. *Returns:* Filtered image.

3. Real-Time Processing Loop

The main loop:

1. Captures frames from webcam.

2. Converts them to grayscale.
3. Applies one of the processing modes:
    - normal: raw camera view.
    - blur: Gaussian blur.
    - sobel: gradient visualization.
    - canny: full Canny edge detection.
    - lpf, hpf, brf, bpf: frequency filters (low-pass, high-pass, band-reject, band-pass).

On-Screen Info

Text overlays show:

- Current kernel size.
- Thresholds (in Canny mode).
- Filter parameters (in frequency mode).

4. Interactive Keyboard Controls

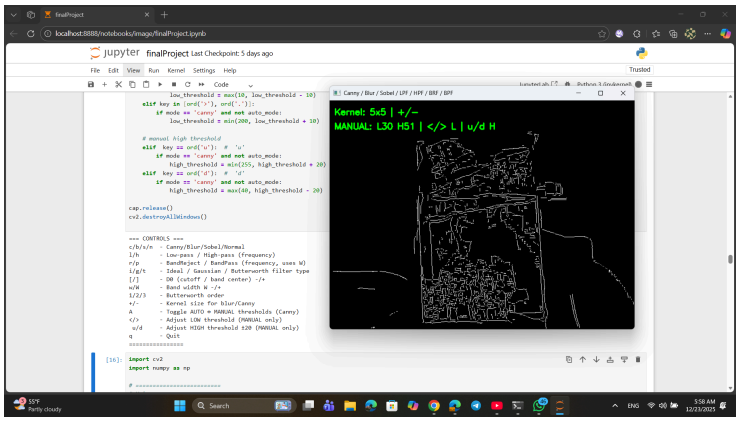
Key	Function
q	Quit the program
c / b / s / n	Switch Canny / Blur / Sobel / Normal
l / h / r / p	Low-pass / High-pass / Band-reject / Band-pass
i / g / t	Ideal / Gaussian / Butterworth filter type
[/]	Decrease / Increase cutoff frequency D0
w / W	Decrease / Increase bandwidth W
1 / 2 / 3	Set Butterworth order
+ / -	Change kernel size (for blur & Canny)
A	Toggle between AUTO and MANUAL thresholds (Canny)
< > / , .	Adjust low threshold (Manual mode only)
u / d	Adjust high threshold (Manual mode only)

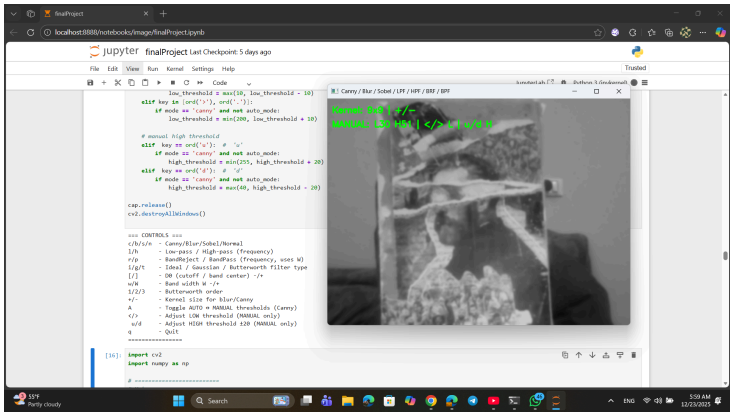
5. Output Display

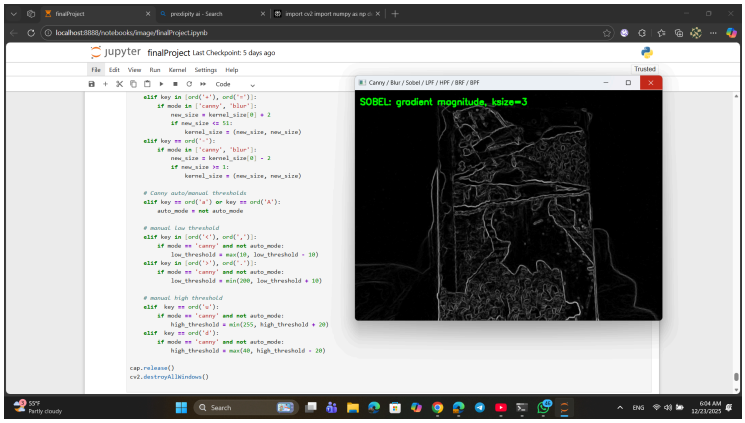
The window title "Canny / Blur / Sobel / LPF / HPF / BRF / BPF" shows live processed video.

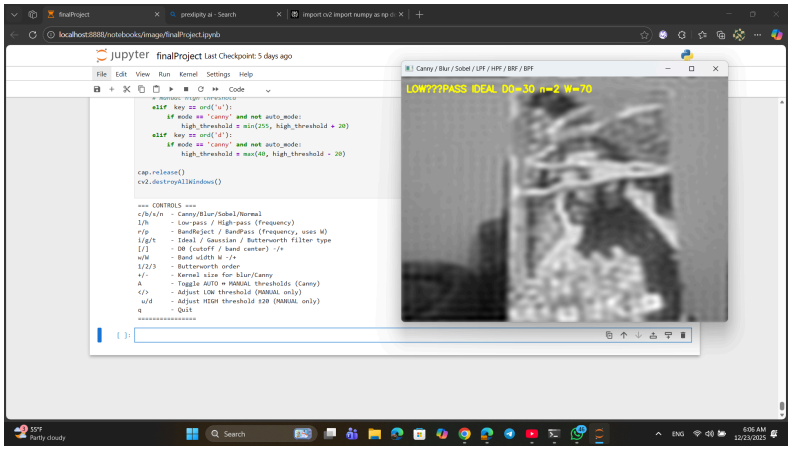
Information overlay displays parameters dynamically as you modify settings through keyboard

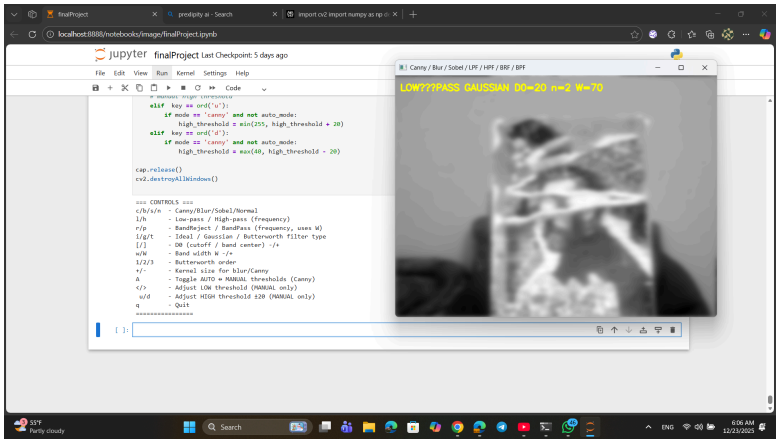
shortcuts

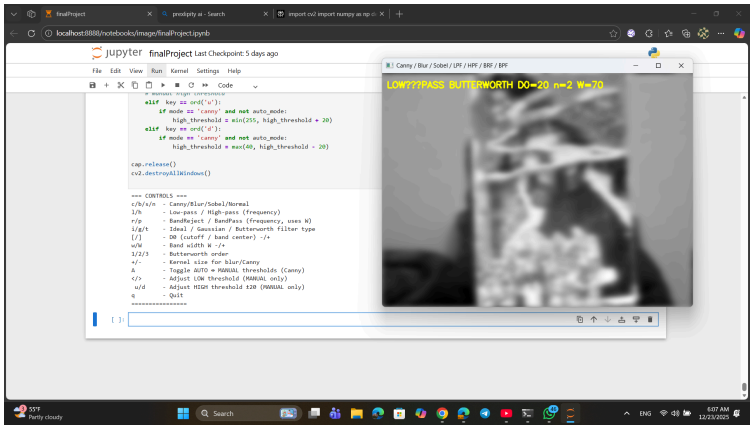

Kernel: 5x5 | +/-
MANUAL: L30 H51 | </> L | u/d H

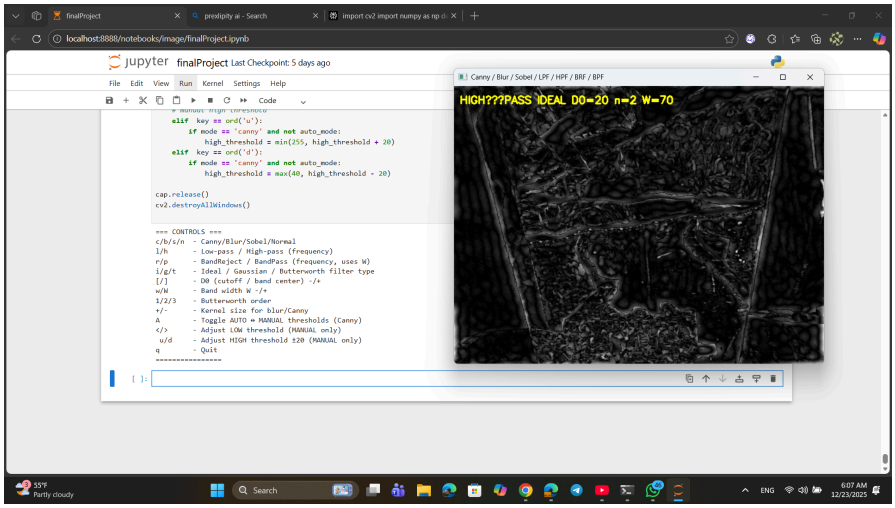

Kernel: 5x5 | +/-
MANUAL: L30 H51 | </> L | u/d H

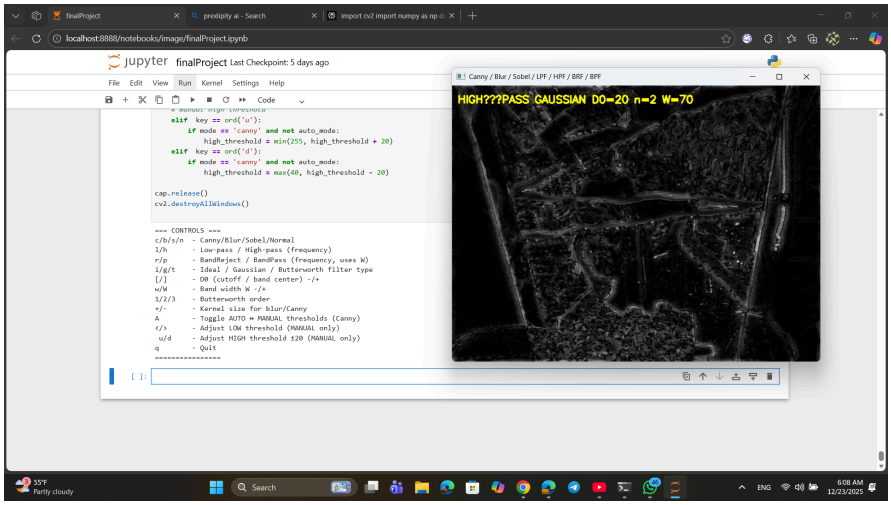

SOBEL: gradient magnitude, ksize=3

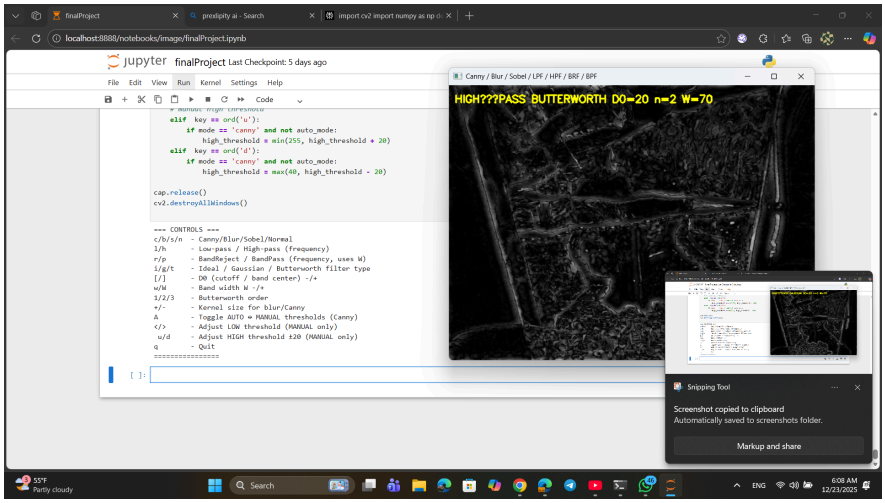

Canny / Blur / Sobel / LPF / HPF / BRF / BPF
LOW???PASS IDEAL D0=30 n=2 W=70

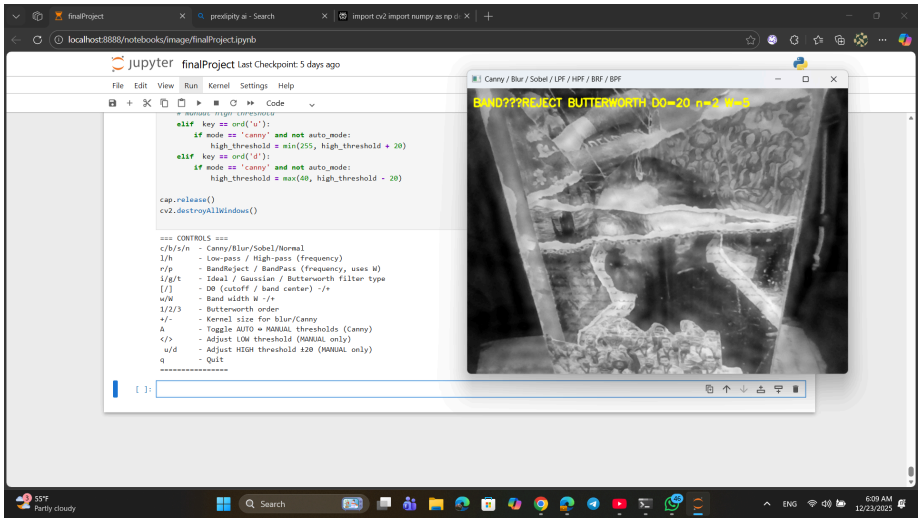

Canny / Blur / Sobel / LPF / HPF / BRF / BPF
LOW???PASS GAUSSIAN D0=20 n=2 W=70

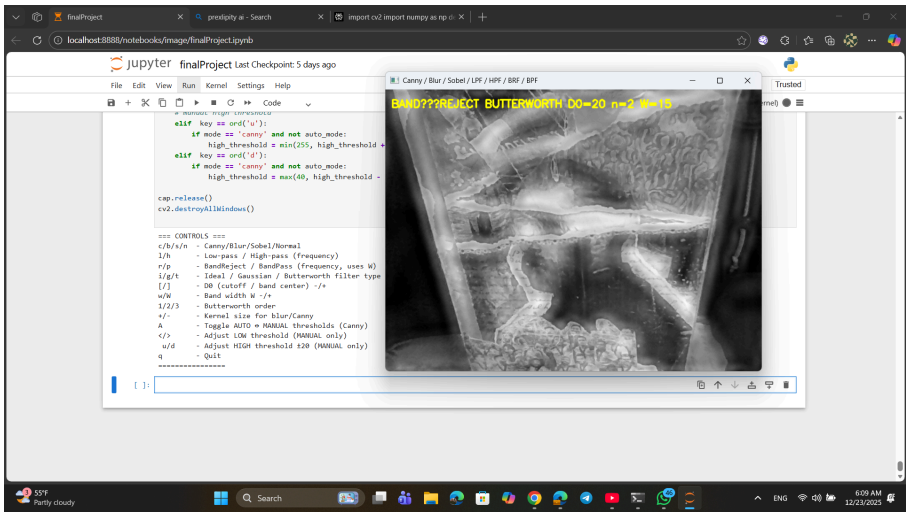

Canny / Blur / Sobel / LPF / HPF / BRF / BPF
LOW???PASS BUTTERWORTH D0=20 n=2 W=70


Canny / Blur / Sobel / LPF / HPF / BRF / BPF
HIGH???PASS IDEAL D0=20 n=2 W=70
jupyter finalProject Last Checkpoint: 5 days ago
=== CONTROLS ===
c/b/s/n - Canny/Blur/Sobel/Normal
l/h - Low-pass / High-pass (frequency)
r/p - BandReject / BandPass (frequency, uses W)
i/g/t - Ideal / Gaussian / Butterworth filter type
[/] - D0 (cutoff / band center) -/+
w/W - Band width W -/+
1/2/3 - Butterworth order
+/- - Kernel size for blur/Canny
A - Toggle AUTO + MANUAL thresholds (Canny)
</> - Adjust LOW threshold (MANUAL only)
u/d - Adjust HIGH threshold ±20 (MANUAL only)
q - Quit


Canny / Blur / Sobel / LPF / HPF / BRF / BPF
HIGH???PASS GAUSSIAN D0=20 n=2 W=70
jupyter finalProject Last Checkpoint: 5 days ago
=== CONTROLS ===
c/b/s/n - Canny/Blur/Sobel/Normal
l/h - Low-pass / High-pass (frequency)
r/p - BandReject / BandPass (frequency, uses W)
i/g/t - Ideal / Gaussian / Butterworth filter type
[/] - D0 (cutoff / band center) -/+
w/W - Band width W -/+
1/2/3 - Butterworth order
+/- - Kernel size for blur/Canny
A - Toggle AUTO + MANUAL thresholds (Canny)
</> - Adjust LOW threshold (MANUAL only)
u/d - Adjust HIGH threshold ±20 (MANUAL only)
q - Quit


Canny / Blur / Sobel / LPF / HPF / BRF / BPF
HIGH???PASS BUTTERWORTH D0=20 n=2 W=70
jupyter finalProject Last Checkpoint: 5 days ago
=== CONTROLS ===
c/b/s/n - Canny/Blur/Sobel/Normal
l/h - Low-pass / High-pass (frequency)
r/p - BandReject / BandPass (frequency, uses W)
i/g/t - Ideal / Gaussian / Butterworth filter type
[/] - D0 (cutoff / band center) -/+
w/W - Band width W -/+
1/2/3 - Butterworth order
+/- - Kernel size for blur/Canny
A - Toggle AUTO + MANUAL thresholds (Canny)
</> - Adjust LOW threshold (MANUAL only)
u/d - Adjust HIGH threshold ±20 (MANUAL only)
q - Quit
Snipping Tool
Screenshot copied to clipboard
Automatically saved to screenshots folder.
Markup and share





.==========

Line-by-Line Code Explanation

Real-Time Image Processing using OpenCV

==========

---

1. Library Imports

---

import cv2

Imports OpenCV, which is used for camera access, image processing,

filtering, drawing text, and displaying images.

import numpy as np

Imports NumPy for numerical operations, matrix handling,

and Fourier Transform calculations.

---

2. Spatial Domain Functions

---

def NoisRedeuction(img, Kernalsize):

Applies Gaussian Blur to reduce noise in the input image.

The kernel size controls the amount of smoothing.

cv2.GaussianBlur automatically selects sigma when sigmaX=0.

Returns the blurred image.

---

def Gradient(img):

Uses Sobel operators to compute image gradients.

grad_x:

Horizontal gradient (detects vertical edges).

grad_y:

Vertical gradient (detects horizontal edges).

grad_magnitude:

Edge strength computed as sqrt(grad_x² + grad_y²).

grad_direction:

Edge orientation computed using arctan2.

The magnitude and direction are normalized for display.

Returns:

Raw gradient magnitude,

Gradient direction,

Display-ready magnitude image.

---

def non_maximum_suppression(grad_mag, grad_angle):

Performs edge thinning by keeping only local maxima.

Image dimensions are extracted.

```
    Gradient angles are converted from radians to degrees
    and mapped to the range [0, 180).

    For each pixel:
        - The gradient direction determines which two neighbors
          should be compared.
        - If the pixel is not greater than its neighbors,
          it is suppressed (set to zero).

    Returns a thinned edge image.

------------------------------------------------------------

def double_threshold(image, low_threshold, high_threshold):
    Separates edges into strong and weak edges.

    Strong edges:
        Pixel intensity >= high_threshold (assigned value 255).

    Weak edges:
        Pixel intensity between low and high thresholds
        (assigned value 75).

    Returns:
```

strong_edges image,

weak_edges image.

---

def edge_tracking_by_hysteresis(strong_edges, weak_edges):

Finalizes edge detection by connecting weak edges to strong ones.

Each weak pixel is examined:

If any of its 8-connected neighbors is strong,

it is promoted to a strong edge.

Weak edges not connected to strong edges are removed.

Returns the final edge map.

---

def auto_threshold(image):

Automatically computes Canny thresholds.

The mean and standard deviation of the image are calculated.

Thresholds are derived from these statistics

and clamped to valid ranges.

Returns:

low_threshold,

high_threshold.

---

def Canny(image, kernel_size, low_thresh, high_thresh, auto_thresh):

Implements a complete Canny edge detector from scratch.

Step 1:

Noise reduction using Gaussian Blur.

Step 2:

Gradient computation using Sobel operators.

Step 3:

Non-maximum suppression to thin edges.

Step 4:

Threshold selection (automatic or manual).

Step 5:

Double thresholding.

Step 6:

Edge tracking by hysteresis.

Returns:

Final edge image,

Selected low and high thresholds.

---

3. Frequency Domain Functions

---

def to_freq(img):

Converts a grayscale image to the frequency domain.

Uses 2D FFT and shifts the zero-frequency component

to the center of the spectrum.

Returns the shifted frequency representation.

---

def from_freq(f_shift):

Converts frequency-domain data back to spatial domain.

Applies inverse FFT and takes the magnitude of the result.

Normalizes values to the range 0–255 for display.

Returns reconstructed grayscale image.

---

def make_filter_mask(shape, ftype, mode, D0, n, W):

Generates a frequency-domain filter mask.

A distance matrix is computed from the center of the spectrum.

Depending on the filter type:

- Ideal: sharp cutoff
- Gaussian: smooth decay
- Butterworth: controlled transition using order n

Filter modes:

- Low-pass
- High-pass
- Band-reject
- Band-pass

Returns the filter mask.

---

def freq_filter(img, ftype, mode, D0, n, W):

Applies frequency-domain filtering.

Steps:

- Convert image to frequency domain.
- Generate filter mask.
- Multiply spectrum by mask.
- Convert back to spatial domain.

Returns filtered image.

---

4. Main Program Logic

---

cap = cv2.VideoCapture(0)

Opens the default webcam.

kernel_size, thresholds, mode variables:

Control blur strength, edge detection thresholds,

and current processing mode.

print("=== CONTROLS ===")

Displays keyboard control instructions in the console.

---

while True:

Continuously captures frames from the camera.

Each frame is:

- Converted to grayscale.
- Processed according to the selected mode
  (Normal, Blur, Sobel, Canny, or Frequency filters).
- Displayed with on-screen parameter information.

cv2.waitKey(1):

Captures keyboard input for real-time control.

Pressing 'q' exits the loop.

---

cap.release()

Releases the camera resource.

cv2.destroyAllWindows()

Closes all OpenCV windows.