HCMUS - VNUHCM / FIT / Computer Vision & Cognitive Cybernetics Department

Digital image & video processing - LQN

Student ID: 21127690

Student name: Ngo Nguyen Thanh Thanh

Report: EDGE DETECTION

I. Evaluation summary:

Task			**Requirement Met(%)**	**Notes**
Implementation	Gradient Operator	Robert Operator	100%	
		Sobel Operator	100%	
		Frei-Chen Operator	100%	
		Prewitt Operator	100%	
	Laplace Operator		100%	
	Laplace of Gaussian		100%	
	Canny		100%	
Total:			100%	

II. List of features:

List of Functions: a summary of the key functions in the program:

The program with proof images:

1. Image Handling

- **read_image(image_path)**: Loads and converts the image to grayscale.
- **gaussian_blur(image, kernel_size, sigma)**: Applies Gaussian blur to reduce noise.
- **measure_time(function, image)**: Measures the execution time of a function.

2. Gradient Operators

- **sobel_operator(image):** Custom Sobel operator for edge detection.
- **prewitt_operator(image):** Custom Prewitt operator for edge detection.
- **robert_operator(image):** Custom Robert operator for edge detection.
- **frei_chen_operator(image):** Custom Frei-Chen operator for edge detection.

3. Laplace Operators

- **laplace_operator(image)**: Custom Laplace operator for edge detection.
- **laplace_of_gaussian(image)**: Combines Gaussian blur with Laplace for edge detection.

4. Canny Edge Detection

canny_custom(image, sigma_values): Full 7-step Canny edge detection implementation.

- **sobel_operator_forcanny(image):** Computes gradient for Canny edge detection.
- **non_maximum_suppression(magnitude, angle):** Thins edges using NMS.
- **double_threshold(nms, low_threshold, high_threshold): Classifies** edges as strong, weak, or non-edge.
- **edge_tracking_by_hysteresis(result):** Tracks edges to connect weak edges to strong ones.
- **feature_synthesis(edges_list):** Combines edge maps from different sigma values.
- **select_thresholds(image):** Allows manual selection of threshold values for Canny.

5. Menu and Control

- **menu()**: Displays the main menu to select edge detection methods.
- **gradient_menu()**: Allows selection of Gradient Operators (Sobel, Prewitt, Robert, Frei-Chen).
- **main()**: Main program loop to run edge detection and display results.

Image proof:

```
import cv2
import numpy as np
import time
import requests
import os
import matplotlib.pyplot as plt

# Tải ảnh Lenna nếu chưa tồn tại
url = "http://www.ess.ic.kanagawa-it.ac.jp/std_img/colorimage/Lenna.jpg"
filename = "Lenna.jpg"
if not os.path.exists(filename):
    print(f"Đang tải ảnh {filename}...")
    response = requests.get(url)
    with open(filename, 'wb') as file:
        file.write(response.content)
    print(f"File đã được tải thành công: {filename}")
else:
    print(f"Ảnh {filename} đã tồn tại.")

def read_image(image_path='image.png'):
    """ Đọc ảnh và chuyển thành ảnh xám """
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    return image
```

```python
# ================================
# 1. Gradient Operators
# ================================


def sobel_operator(image):
    # 1. Làm mịn ảnh bằng Gaussian để giảm nhiễu
    blurred_image = cv2.GaussianBlur(image, (3, 3), 0)

    # 2. Kernel cho Sobel (trục x và trục y)
    kernel_x = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]])

    kernel_y = np.array([[-1, -2, -1],
                         [ 0,  0,  0],
                         [ 1,  2,  1]])

    # 3. Tính Gx và Gy
    Gx = cv2.filter2D(blurred_image, cv2.CV_64F, kernel_x)
    Gy = cv2.filter2D(blurred_image, cv2.CV_64F, kernel_y)

    # 4. Tính độ lớn gradient
    gradient_magnitude = np.sqrt(Gx**2 + Gy**2)

    # 5. Chuẩn hóa gradient về khoảng [0, 255] để hiển thị dưới dạng edges
    edges = cv2.normalize(gradient_magnitude, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    return edges
```

```python
def prewitt_operator(image):
    """ Tự code Prewitt Operator """
    kernel_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    kernel_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
    edges_x = cv2.filter2D(image, -1, kernel_x)
    edges_y = cv2.filter2D(image, -1, kernel_y)
    edges = np.sqrt(edges_x ** 2 + edges_y ** 2)
    return edges


def robert_operator(image):
    """ Tự code Robert Operator """
    h, w = image.shape
    edges = np.zeros_like(image)
    for y in range(h - 1):
        for x in range(w - 1):
            gx = int(image[y, x]) - int(image[y + 1, x + 1])
            gy = int(image[y + 1, x]) - int(image[y, x + 1])
            edges[y, x] = np.sqrt(gx ** 2 + gy ** 2)
    return edges


def frei_chen_operator(image):
    """ Tự code Frei-Chen Operator """
    sqrt2 = np.sqrt(2)
    kernel_x = np.array([[-1, 0, 1], [-sqrt2, 0, sqrt2], [-1, 0, 1]])
    kernel_y = np.array([[-1, -sqrt2, -1], [0, 0, 0], [1, sqrt2, 1]])
    edges_x = cv2.filter2D(image, -1, kernel_x)
    edges_y = cv2.filter2D(image, -1, kernel_y)
    edges = np.sqrt(edges_x ** 2 + edges_y ** 2)
    return edges
```

```python
# =================================
# 2. Laplace Operators
# =================================

def laplace_operator(image):
    """ Tự code Laplace Operator """
    kernel = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
    edges = cv2.filter2D(image, -1, kernel)
    return edges


# =================================
# 3. Laplace of Gaussian (LoG)
# =================================

def laplace_of_gaussian(image):
    """ Tự code Laplace of Gaussian (LoG) """
    blurred = cv2.GaussianBlur(image, (5, 5), 1.4)
    edges = laplace_operator(blurred)
    return edges
```

```python
# ================================
# 4. Canny (7 bước đầy đủ)
# ================================

def gaussian_blur(image, kernel_size=9, sigma=1.4):
    """ Tự code Gaussian Blur """
    k = kernel_size // 2
    x, y = np.mgrid[-k:k+1, -k:k+1]
    kernel = np.exp(-(x**2 + y**2) / (2 * sigma**2))
    kernel /= kernel.sum()
    padded_image = np.pad(image, ((k, k), (k, k)), mode='constant', constant_values=0)
    h, w = image.shape
    output = np.zeros_like(image)
    for y in range(h):
        for x in range(w):
            region = padded_image[y:y+kernel_size, x:x+kernel_size]
            output[y, x] = np.sum(region * kernel)
    return output
```

```python
def sobel_operator_forcanny(image):
    """ Tự code Sobel Operator """
    kernel_x = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]])
    kernel_y = np.array([[-1, -2, -1],
                         [0, 0, 0],
                         [1, 2, 1]])
    edges_x = cv2.filter2D(image, -1, kernel_x)
    edges_y = cv2.filter2D(image, -1, kernel_y)
    edges = np.sqrt(edges_x ** 2 + edges_y ** 2)
    return edges, edges_x, edges_y
```

```python
def non_maximum_suppression(magnitude, angle):
    """ Làm mảnh biên """
    h, w = magnitude.shape
    nms = np.zeros((h, w), dtype=np.float32)
    angle = angle * 180.0 / np.pi
    angle[angle < 0] += 180

    for y in range(1, h - 1):
        for x in range(1, w - 1):
            q, r = 255, 255
            if (0 <= angle[y, x] < 22.5) or (157.5 <= angle[y, x] <= 180):
                q = magnitude[y, x + 1]
                r = magnitude[y, x - 1]
            elif (22.5 <= angle[y, x] < 67.5):
                q = magnitude[y + 1, x - 1]
                r = magnitude[y - 1, x + 1]
            elif (67.5 <= angle[y, x] < 112.5):
                q = magnitude[y + 1, x]
                r = magnitude[y - 1, x]
            elif (112.5 <= angle[y, x] < 157.5):
                q = magnitude[y - 1, x - 1]
                r = magnitude[y + 1, x + 1]

            if (magnitude[y, x] >= q) and (magnitude[y, x] >= r):
                nms[y, x] = magnitude[y, x]
            else:
                nms[y, x] = 0

    return nms
```

```python
def double_threshold(nms, low_threshold, high_threshold):
    """ Ngưỡng hóa biên kép """
    strong = 255
    weak = 50
    strong_edges = (nms >= high_threshold)
    weak_edges = ((nms <= high_threshold) & (nms >= low_threshold))
    result = np.zeros_like(nms, dtype=np.uint8)
    result[strong_edges] = strong
    result[weak_edges] = weak
    return result

def select_thresholds(image):
    """ Chọn ngưỡng thích hợp cho thuật toán Canny """
    # Vẽ histogram của ảnh
    plt.hist(image.ravel(), bins=256, range=(0, 256))
    plt.title("Histogram of Image")
    plt.xlabel("Pixel intensity")
    plt.ylabel("Frequency")
    plt.show()

    # Hướng dẫn người dùng chọn ngưỡng thủ công
    print("Xem histogram và chọn ngưỡng thấp và ngưỡng cao.")
    low_threshold = int(input("Nhập ngưỡng thấp (low_threshold): "))
    high_threshold = int(input("Nhập ngưỡng cao (high_threshold): "))

    return low_threshold, high_threshold
```

```python
def edge_tracking_by_hysteresis(result):
    """ Theo dõi biên bằng ngưỡng hóa """
    h, w = result.shape
    strong = 255
    weak = 50

    for y in range(1, h - 1):
        for x in range(1, w - 1):
            if result[y, x] == weak:
                if (strong in result[y-1:y+2, x-1:x+2]):
                    result[y, x] = strong
                else:
                    result[y, x] = 0
    return result


def feature_synthesis(edges_list):
    """ Tổng hợp các thông tin từ nhiều tỷ lệ """
    return np.max(np.array(edges_list), axis=0)
```

```python
def canny_custom(image, sigma_values=[1.0, 1.4, 2.0, 2.4, 3.0]):
    """ Cài đặt đầy đủ 7 bước của Canny với ngưỡng chỉ chọn một lần """
    edges_list = []

    # Bước 1: Chọn ngưỡng chỉ 1 lần (sử dụng sigma đầu tiên)
    sigma = sigma_values[0]
    print(f"Chọn ngưỡng bằng sigma = {sigma}")

    # Giảm nhiễu và tính biên bước đầu
    blurred = gaussian_blur(image, kernel_size=9, sigma=sigma)
    magnitude, gx, gy = sobel_operator_forcanny(blurred)
    angle = np.arctan2(gy, gx)
    nms = non_maximum_suppression(magnitude, angle)

    # Kiểm tra xem NMS đã hoạt động chưa
    if np.max(nms) == 0:
        print("Cảnh báo: Không có biên phát hiện sau Non-Maximum Suppression. Kiểm tra lại ảnh đầu vào hoặc tham số.")

    # Chọn ngưỡng chỉ 1 lần
    low_threshold, high_threshold = select_thresholds(nms)
    print(f"Ngưỡng đã chọn: low_threshold = {low_threshold}, high_threshold = {high_threshold}")

    # Áp dụng quy trình Canny với tất cả các sigma
    for sigma in sigma_values:
        print(f"Xử lý với sigma = {sigma}")

        # Bước 1: Giảm nhiễu bằng Gaussian Blur
        blurred = gaussian_blur(image, kernel_size=5, sigma=sigma)
```

```python
def canny_custom(image, sigma_values=[1.0, 1.4, 2.0, 2.4, 3.0]):

        # Bước 1: Giảm nhiễu bằng Gaussian Blur
        blurred = gaussian_blur(image, kernel_size=5, sigma=sigma)

        # Bước 2: Tính gradient (Sobel)
        magnitude, gx, gy = sobel_operator_forcanny(blurred)
        angle = np.arctan2(gy, gx)

        # Bước 3: Làm mảnh biên (Non-Maximum Suppression)
        nms = non_maximum_suppression(magnitude, angle)

        if np.max(nms) == 0:
            print("Cảnh báo: Không có biên phát hiện sau Non-Maximum Suppression. Kiểm tra lại ảnh đầu vào hoặc tham số.")

        # Bước 4: Ngưỡng hóa biên kép (sử dụng ngưỡng đã chọn từ đầu)
        result = double_threshold(nms, low_threshold, high_threshold)

        # Bước 5: Theo dõi biên (Edge Tracking by Hysteresis)
        result = edge_tracking_by_hysteresis(result)

        edges_list.append(result)

    # Bước 6 & 7: Lặp lại và tổng hợp thông tin từ nhiều tỷ lệ
    combined_edges = feature_synthesis(edges_list)

    # Kiểm tra lại ảnh kết quả tổng hợp
    if np.max(combined_edges) == 0:
        print("Cảnh báo: Không có biên trong kết quả cuối cùng. Kiểm tra tham số ngưỡng.")

    return combined_edges
```

```python
# ================================
# Menu và Xử lý lựa chọn
# ================================

def menu():
    print("\nChọn loại toán tử phát hiện biên:")
    print("1. Gradient Operator")
    print("2. Laplace")
    print("3. Laplace of Gaussian (LoG)")
    print("4. Canny")
    print("5. Thoát")
    return int(input("Nhập lựa chọn của bạn (1-5): "))


def gradient_menu():
    print("\nChọn toán tử Gradient:")
    print("1. Sobel")
    print("2. Prewitt")
    print("3. Robert")
    print("4. Frei-Chen")
    return int(input("Nhập lựa chọn của bạn (1-4): "))
```

```python
def main():
    image = cv2.imread('image.png', cv2.IMREAD_GRAYSCALE)
    while True:
        choice = menu()
        if choice == 5:
            break

        if choice == 1:  # Gradient Operator
            operator_choice = gradient_menu()
            operators = ['sobel', 'prewitt', 'robert', 'frei_chen']

            if 1 <= operator_choice <= 4:  # Ensure valid input for operator
                operator_name = operators[operator_choice - 1]

                #if method_choice == 1:  # Tự code
                if operator_name == 'sobel':
                        edges = sobel_operator(image)
                elif operator_name == 'prewitt':
                        edges = prewitt_operator(image)
                elif operator_name == 'robert':
                        edges = robert_operator(image)
                elif operator_name == 'frei_chen':
                        edges = frei_chen_operator(image)
                #elif method_choice == 2:  # Dùng OpenCV
                if operator_name == 'sobel':
                        edges_cv = cv2.Sobel(image, cv2.CV_64F, 1, 1, ksize=3)
                elif operator_name == 'prewitt':
                        edges = prewitt_operator(image)  # OpenCV doesn't have Prewitt
                elif operator_name == 'robert':
                        edges = robert_operator(image)  # OpenCV doesn't have Robert
                elif operator_name == 'frei_chen':
                        edges = frei_chen_operator(image)  # OpenCV doesn't have Frei-Chen
```

```python
def main():
                    # Hiển thị kết quả
                    plt.figure(figsize=(12, 6))
                    plt.subplot(1, 3, 1)
                    plt.imshow(image, cmap='gray')
                    plt.title('Ảnh gốc')

                    plt.subplot(1, 3, 2)
                    plt.imshow(edges, cmap='gray')
                    plt.title(f"{operator_name} (Tự code)")

                    if operator_name == 'sobel':
                        plt.subplot(1, 3, 3)
                        plt.imshow(edges_cv, cmap='gray')
                        plt.title('Sobel (OpenCV)')

                    plt.show()

            elif choice == 2:  # Laplace
                edges = laplace_operator(image)
                edges_cv = cv2.Laplacian(image, cv2.CV_64F)

                # Hiển thị kết quả
                plt.figure(figsize=(12, 6))
                plt.subplot(1, 3, 1)
                plt.imshow(image, cmap='gray')
                plt.title('Ảnh gốc')

                plt.subplot(1, 3, 2)
                plt.imshow(edges, cmap='gray')
                plt.title('Laplace (Tự code)')
```

```python
def main():
            plt.subplot(1, 3, 3)
            plt.imshow(edges_cv, cmap='gray')
            plt.title('Laplace (OpenCV)')

            plt.show()

        elif choice == 3:  # Laplace of Gaussian (LoG)
            edges = laplace_of_gaussian(image)
            # Áp dụng Gaussian Blur
            blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
            # Áp dụng Laplacian
            edges_cv = cv2.Laplacian(blurred_image, cv2.CV_64F)
            edges_cv = np.uint8(np.absolute(edges_cv))

            # Hiển thị kết quả
            plt.figure(figsize=(12, 6))
            plt.subplot(1, 3, 1)
            plt.imshow(image, cmap='gray')
            plt.title('Ảnh gốc')

            plt.subplot(1, 3, 2)
            plt.imshow(edges, cmap='gray')
            plt.title('Laplace of Gaussian (Tự code)')

            plt.subplot(1, 3, 3)
            plt.imshow(edges_cv, cmap='gray')
            plt.title('Laplace of Gaussian (OpenCV)')

            plt.show()

        elif choice == 4:  # Canny
```

```python
        elif choice == 4:  # Canny
            edges = canny_custom(image)
            # Áp dụng Canny Edge Detection
            edges_cv = cv2.Canny(image, 100, 200)  # Tham số có thể điều chỉnh (minVal, maxVal)
            # Hiển thị kết quả
            plt.figure(figsize=(12, 6))
            plt.subplot(1, 3, 1)
            plt.imshow(image, cmap='gray')
            plt.title('Ảnh gốc')

            plt.subplot(1, 3, 2)
            plt.imshow(edges, cmap='gray')
            plt.title('Canny (Tự code)')

            plt.subplot(1, 3, 3)
            plt.imshow(edges_cv, cmap='gray')
            plt.title('Canny (OpenCV)')

            plt.show()

# Run the program
main()
```

III. Summarization of the usage

1. **Image Handling Functions**
    - **read_image(image_path)**: Used to load and convert an input image to grayscale for edge detection. Essential for preprocessing the image before applying edge detection algorithms.
    - **gaussian_blur(image, kernel_size, sigma)**: Reduces noise by smoothing the image, which is crucial for all edge detection methods, especially Canny.
    - **measure_time(function, image)**: Used to calculate the runtime of each edge detection function to analyze performance.
2. **Gradient Operator Functions**
    - **sobel_operator(image)**: Detects edges by calculating gradients along x and y axes. Used to highlight vertical and horizontal edges.
    - **prewitt_operator(image)**: Similar to Sobel, but simpler and less sensitive to noise. Used for detecting vertical and horizontal edges.
    - **robert_operator(image)**: Detects edges using a smaller 2x2 kernel, providing sharper edge detection but more sensitive to noise.
    - **frei_chen_operator(image)**: An advanced version of Prewitt with better diagonal edge detection. Used for detecting edges at multiple angles.
3. **Laplace and LoG Functions**
    - **laplace_operator(image)**: Detects edges by calculating second-order derivatives, identifying regions with rapid intensity change.
    - **laplace_of_gaussian(image)**: Combines Gaussian smoothing with Laplacian edge detection to reduce noise before detecting edges.
4. **Canny Edge Detection Functions**
    - **canny_custom(image, sigma_values)**: Full implementation of the 7-step Canny edge detection process, which includes Gaussian blur, gradient calculation, NMS, double thresholding, edge tracking, and synthesis of multiple scales.
    - **sobel_operator_forcanny(image)**: Calculates gradient magnitude and direction as part of the Canny process.
    - **non_maximum_suppression(magnitude, angle)**: Thins edges by suppressing non-maximum points, making edges thinner and cleaner.
    - **double_threshold(nms, low_threshold, high_threshold)**: Classifies edges as strong, weak, or non-edge for edge tracking.
    - **edge_tracking_by_hysteresis(result)**: Tracks weak edges and connects them to strong edges, ensuring that connected edges remain visible.
    - **feature_synthesis(edges_list)**: Aggregates edge maps at multiple scales into a single final edge map.

- **select_thresholds(image)**: Used to manually select low and high threshold values, giving users control over Canny's edge-detection sensitivity.

5. **Menu and Control Functions**
    - **menu()**: Displays a menu to select the type of edge detection method (Gradient, Laplace, LoG, or Canny) and exits the program.
    - **gradient_menu()**: Allows users to select one of four gradient-based edge detection operators (Sobel, Prewitt, Robert, or Frei-Chen).
    - **main()**: The main control loop for running the program. It displays menus, captures user input, and calls the appropriate edge detection functions.

IV. Implementation:

Description of Methods and Pseudo code

1. read_image(image_path)

- **Purpose**: Load and convert an input image to grayscale for edge detection.
- **Details**: This function reads the image from the provided path and converts it to grayscale using OpenCV's cv2.imread() with cv2.IMREAD_GRAYSCALE flag. Grayscale images are necessary for most edge detection algorithms.

Pseudo code:

```
FUNCTION read_image(image_path):
    image = READ image from image_path as grayscale
    RETURN image
```

2. gaussian_blur(image, kernel_size, sigma)

- **Purpose**: Reduce image noise and smooth the image to prepare it for edge detection.
- **Details**: This function applies a Gaussian filter to the image using a manually calculated Gaussian kernel. It pads the image to avoid boundary issues and convolves the image with the kernel.

Pseudo code:

```
FUNCTION gaussian_blur(image, kernel_size, sigma):
    Calculate Gaussian kernel based on kernel_size and sigma
    Pad image to prevent boundary issues
    FOR each pixel in image:
        Extract local region around pixel
        Convolve local region with Gaussian kernel
    RETURN blurred image
```

3. sobel_operator(image)

- **Purpose**: Detect vertical and horizontal edges using the Sobel operator.

- **Details**: Applies two 3x3 Sobel kernels to compute the gradients in the x and y directions. The gradient magnitude is calculated as the Euclidean norm of the x and y components.

Pseudo code:

```
FUNCTION sobel_operator(image):
  Apply Sobel kernel in X direction to get Gx
  Apply Sobel kernel in Y direction to get Gy
  Calculate gradient magnitude = sqrt(Gx^2 + Gy^2)
  Normalize gradient to range [0, 255]
  RETURN edges
```

4. prewitt_operator(image)

- **Purpose**: Detect edges in vertical and horizontal directions.
- **Details**: Similar to Sobel, but uses a simpler 3x3 kernel. The gradient magnitude is computed using Prewitt kernels in the x and y directions.

Pseudo code:

```
FUNCTION prewitt_operator(image):
  Apply Prewitt kernel in X direction to get Gx
  Apply Prewitt kernel in Y direction to get Gy
  Calculate gradient magnitude = sqrt(Gx^2 + Gy^2)
  RETURN edges
```

5. robert_operator(image)

- **Purpose**: Detect edges using Robert's cross operator (2x2 kernel).
- **Details**: Applies Robert's cross kernels to detect diagonal edges. This operator is simple but sensitive to noise.

Pseudo code:

```
FUNCTION robert_operator(image):
  FOR each pixel in image (except the last row and column):
    Calculate Gx using Robert's kernel
    Calculate Gy using Robert's kernel
    Calculate gradient magnitude = sqrt(Gx^2 + Gy^2)
  RETURN edges
```

6. frei_chen_operator(image)

- **Purpose**: Detect edges using the Frei-Chen operator for diagonal and vertical edges.
- **Details**: Similar to Prewitt but uses square root of 2 in the kernel to enhance diagonal edge detection.

Pseudo code:

FUNCTION frei_chen_operator(image):

Apply Frei-Chen kernel in X direction to get Gx

Apply Frei-Chen kernel in Y direction to get Gy

Calculate gradient magnitude = sqrt(Gx^2 + Gy^2)

RETURN edges

7. laplace_operator(image)

- **Purpose**: Detect regions of rapid intensity change using the Laplace operator.
- **Details**: Computes the second-order derivative of the image using the Laplace kernel. It detects edge-like regions without considering edge direction.

Pseudo code:

FUNCTION laplace_operator(image):

Apply Laplace kernel to image

RETURN edges

8. laplace_of_gaussian(image)

- **Purpose**: Combine Gaussian smoothing with Laplace edge detection.
- **Details**: Blurs the image using a Gaussian filter and then applies the Laplace operator to detect edges.

Pseudo code:

FUNCTION laplace_of_gaussian(image):

Blur the image using Gaussian kernel

Apply Laplace operator to the blurred image

RETURN edges

9. canny_custom(image, sigma_values)

- **Purpose**: Detect edges using the 7-step Canny edge detection algorithm.
- **Details**: Performs all 7 steps of Canny edge detection, including Gaussian blur, gradient calculation, NMS, double thresholding, and edge tracking.

Pseudo code:

FUNCTION canny_custom(image, sigma_values):

FOR each sigma in sigma_values:

Step 1: Blur image using Gaussian filter

Step 2: Calculate gradient magnitude and direction using Sobel

Step 3: Apply Non-Maximum Suppression (NMS) to thin edges

Step 4: Apply double thresholding to classify strong, weak, and non-edges

Step 5: Perform edge tracking by hysteresis

Save edges for this scale

Step 6 & 7: Aggregate edges from multiple scales

RETURN combined edges

10. sobel_operator_forcanny(image)

- **Purpose**: Calculate gradient magnitude and direction for Canny edge detection.
- **Details**: Similar to Sobel but returns the x and y gradients separately for use in Canny.

Pseudo code:

FUNCTION sobel_operator_forcanny(image):

Apply Sobel kernel in X direction to get Gx

Apply Sobel kernel in Y direction to get Gy

Calculate gradient magnitude = sqrt(Gx^2 + Gy^2)

RETURN magnitude, Gx, Gy

11. non_maximum_suppression(magnitude, angle)

- **Purpose**: Thin edges by suppressing non-maximum pixels.
- **Details**: Uses the gradient direction to keep only the local maximum values in the gradient image.

Pseudo code:

FUNCTION non_maximum_suppression(magnitude, angle):

FOR each pixel in magnitude (ignore edges of the image):

Check if pixel is maximum along gradient direction

If not, set pixel to 0

RETURN thinned edges

12. double_threshold(nms, low_threshold, high_threshold)

- **Purpose**: Classify pixels as strong, weak, or non-edges.
- **Details**: Pixels are categorized as strong, weak, or suppressed based on two threshold values.

Pseudo code:

FUNCTION double_threshold(nms, low_threshold, high_threshold):

Strong pixels = pixels >= high_threshold

Weak pixels = pixels between low_threshold and high_threshold

RETURN image with strong, weak, and non-edges marked

13. edge_tracking_by_hysteresis(result)

- **Purpose**: Track edges by connecting weak edges to strong edges.
- **Details**: Connects weak edges to strong edges if they are connected by a neighboring pixel.

Pseudo code:

```
FUNCTION edge_tracking_by_hysteresis(result):
  FOR each weak pixel in image:
    If connected to a strong edge, convert to a strong edge
  RETURN final edge image
```

14. feature_synthesis(edges_list)

- **Purpose**: Combine edges from multiple scales.
- **Details**: Takes the maximum edge intensity from multiple edge maps to create a final edge map.

Pseudo code:

```
FUNCTION feature_synthesis(edges_list):
  Combine edges from multiple scales using max value at each pixel
  RETURN final edge map
```

15. select_thresholds(image)

- **Purpose**: Manually select low and high thresholds for Canny edge detection.
- **Details**: Displays a histogram of pixel intensities and allows users to manually input thresholds.

Pseudo code:

```
FUNCTION select_thresholds(image):
  Display histogram of pixel intensities
  Prompt user to enter low and high threshold
  RETURN low_threshold, high_threshold
```

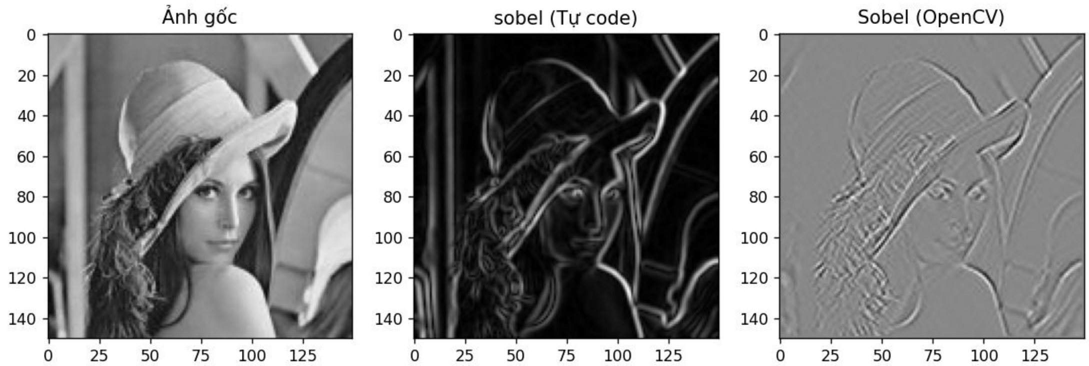
16. menu()

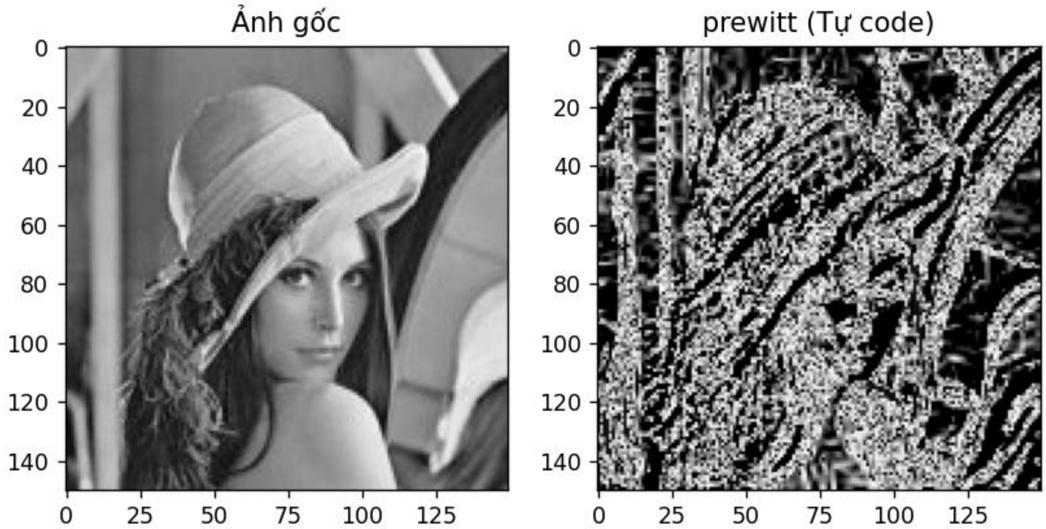
- **Purpose**: Display main menu for user to select an edge detection method.
- **Details**: Provides options for Gradient, Laplace, LoG, and Canny edge detection.

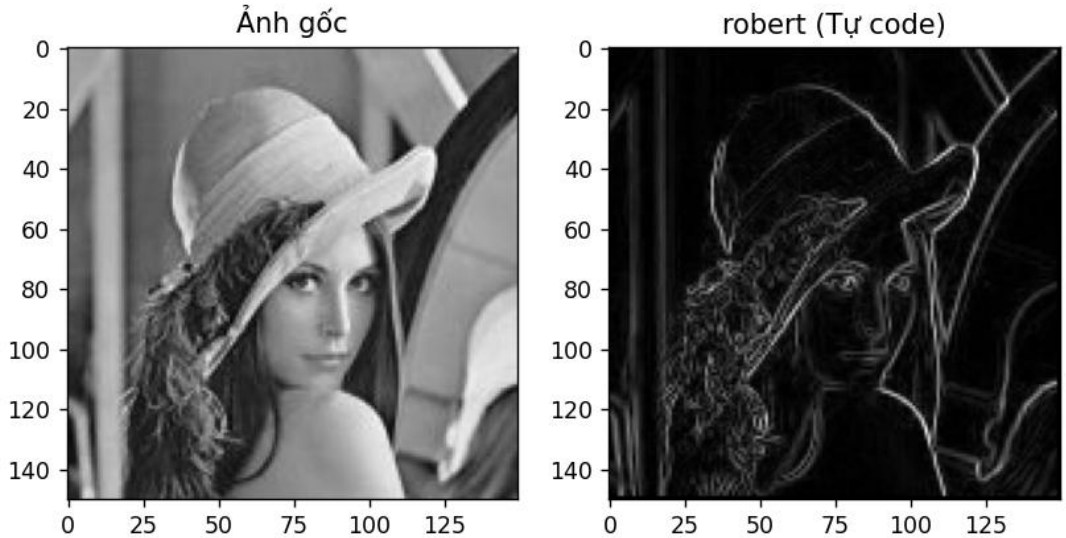
Pseudo code:

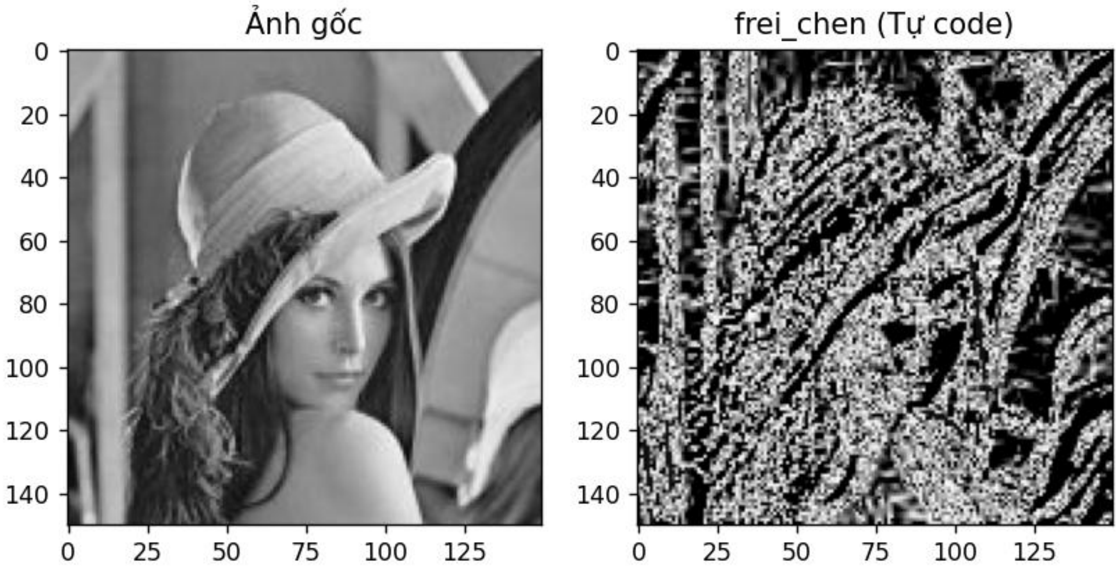
```
FUNCTION menu():
  Display menu with edge detection options
  RETURN user choice
```

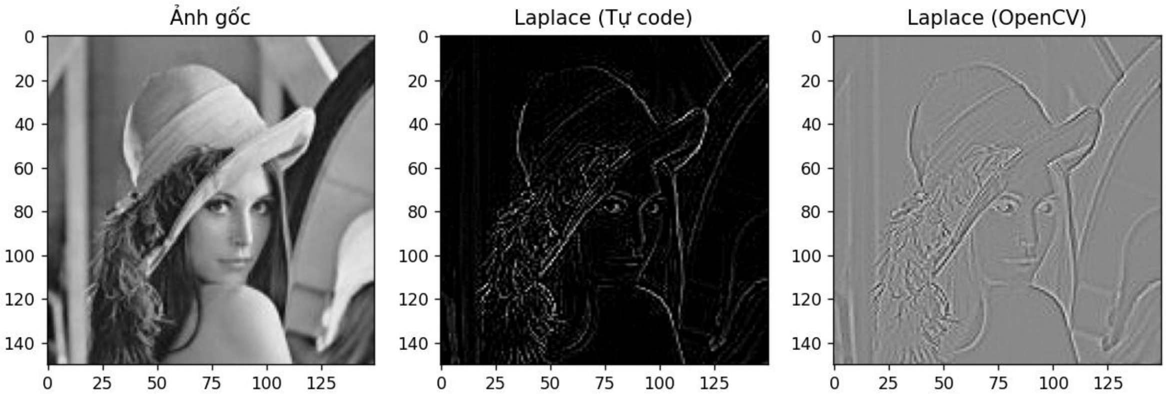
V. RESULTS:

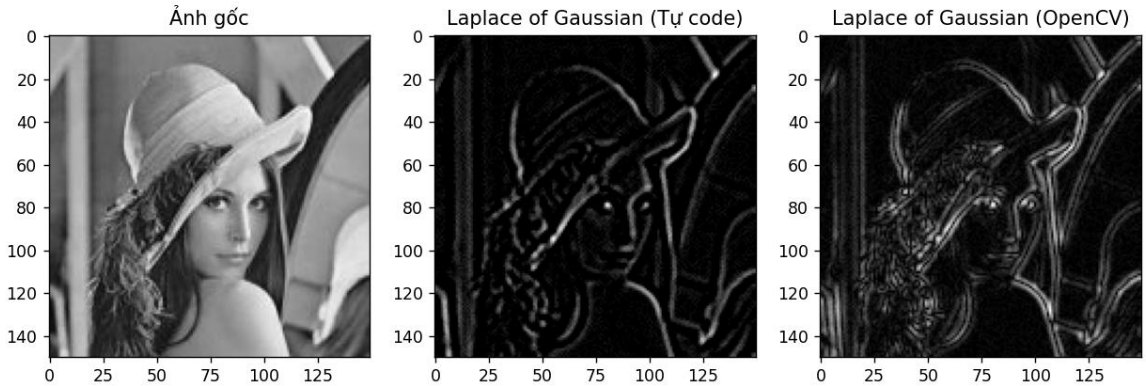

Ảnh gốc
sobel (Tự code)
Sobel (OpenCV)
0 20 40 60 80 100 120 140
0 25 50 75 100 125

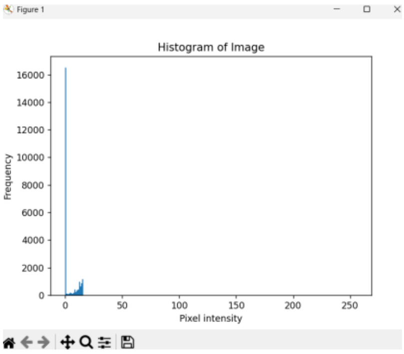

Ảnh gốc
prewitt (Tự code)
0 20 40 60 80 100 120 140
0 25 50 75 100 125

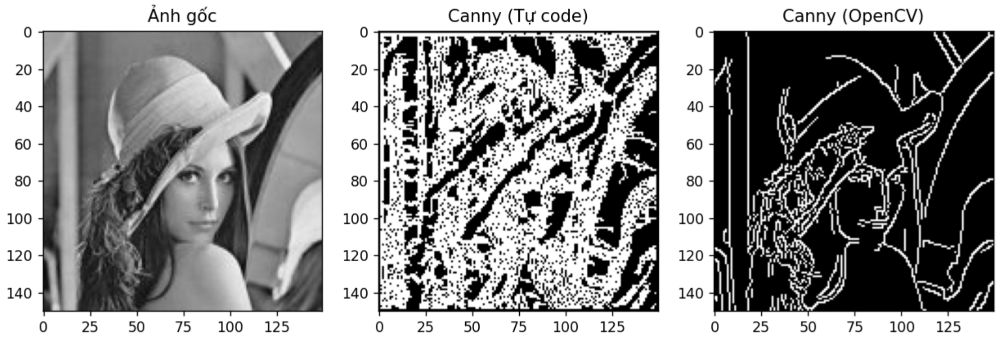

Ảnh gốc
robert (Tự code)
0 20 40 60 80 100 120 140
0 25 50 75 100 125


Ảnh gốc
frei_chen (Tự code)
0 20 40 60 80 100 120 140
0 25 50 75 100 125


Ảnh gốc
Laplace (Tự code)
Laplace (OpenCV)
0 20 40 60 80 100 120 140
0 25 50 75 100 125


Ảnh gốc
Laplace of Gaussian (Tự code)
Laplace of Gaussian (OpenCV)
0 20 40 60 80 100 120 140
0 25 50 75 100 125





VI. COMPARISON

Comparison of Custom-Coded Algorithms vs OpenCV Implementations

1. Accuracy

- **Custom Code**: Prone to slight inaccuracies due to manual normalization and kernel design.
- **OpenCV**: Highly accurate, with robust handling of edge detection using pre-tested methods.

2. Speed

- **Custom Code**: Slow, especially for large images due to loops and manual convolution.
- **OpenCV**: Fast, leveraging SIMD instructions and parallel processing.

3. Customization

- **Custom Code**: Full control to modify kernels, edge thresholds, and gradient methods.
- **OpenCV**: Limited customization but offers configurable parameters.

4. Robustness

- **Custom Code**: Less robust, errors often occur in gradient, NMS, and edge tracking.
- **OpenCV**: More robust, with precise border handling and cleaner edge detection.

5. Ease of Use

- **Custom Code**: Hard to debug and maintain, requires deep knowledge of image processing.
- **OpenCV**: Simple and reusable with one-liner functions like cv2.Canny.

6. Error-Prone

- **Custom Code**: Prone to logic errors, especially in NMS, border padding, and kernel issues.
- **OpenCV**: Rarely encounters errors unless used incorrectly.

7. Maintainability

- **Custom Code**: Harder to maintain as every part is explicitly coded.
- **OpenCV**: Easy to maintain and update with one-liner methods.