1. **Please design a highboost method including the Sobel and Laplacian filter in pp.183-195 to enhance the image,‘bodybone.bmp’ as Fig. 3.49 (e). Please describe the your highboost filter, procedures, final enhanced image and print out the source code? (40)**

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Define Sobel filter
def sobel_filter(image):
    # Define Sobel kernels (standard Sobel kernels)
    sobel_x = np.array([[-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]], dtype=np.float32)  # Horizontal direction

    sobel_y = np.array([[1,  2,  1],
                        [0,  0,  0],
                        [-1, -2, -1]], dtype=np.float32)  # Vertical direction

    # Apply the filter
    grad_x = cv2.filter2D(image, cv2.CV_64F, sobel_x)
    grad_y = cv2.filter2D(image, cv2.CV_64F, sobel_y)

    # Compute the gradient magnitude and normalize
    magnitude = np.sqrt(grad_x**2 + grad_y**2)
    magnitude = np.clip(magnitude, 0, 255).astype(np.uint8)  # Ensure result is within range
    return magnitude

# Define Laplacian filter (5x5 kernel)
def laplacian_filter(image):
    # Define Laplacian kernel (5x5kernel)
    laplacian = np.array([[0,  1, 0,  1, 0],
                          [1, -4, 1, -4, 1],
                          [0,  1, 0,  1, 0],
                          [1, -4, 1, -4, 1],
                          [0,  1, 0,  1, 0]], dtype=np.float32)

    # Apply the filter
    laplacian_result = cv2.filter2D(image, cv2.CV_64F, laplacian)

    return laplacian_result
```

```python
# Define high-boost method
def high_boost(image, alpha=1.0):

    # Use Gaussian blur for noise reduction
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
    # Convert to grayscale
    gray_image = cv2.cvtColor(blurred_image, cv2.COLOR_BGR2GRAY)

    # Apply Sobel and Laplacian filters
    sobel_edges = sobel_filter(gray_image)
    laplacian_edges = laplacian_filter(gray_image)

    # Combine Sobel and Laplacian edges
    edges = np.clip(sobel_edges + laplacian_edges, 0, 255).astype(np.uint8)

    # High-boost image
    high_boost_image = cv2.addWeighted(gray_image, 1 + alpha, edges, -alpha, 0)
    return gray_image, sobel_edges, laplacian_edges, high_boost_image

# Main function
if __name__ == "__main__":
    # Load the image
    image = cv2.imread('Bodybone.bmp')  # Replace with your image path

    if image is None:
        print("Unable to read the image, please check the file path.")
    else:
        # Apply high-boost method
        gray_image, sobel_edges, laplacian_edges, high_boost_image = high_boost(image)

        # Display the images
        plt.figure(figsize=(12, 8))

        # Original image
        plt.subplot(2, 2, 1)
        plt.title('Original Image')
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert color channels
        plt.axis('off')
```

```python
# Sobel filter result
plt.subplot(2, 2, 2)
plt.title('Sobel Filter Result')
plt.imshow(sobel_edges, cmap='gray')
plt.axis('off')

# Laplacian filter result
plt.subplot(2, 2, 3)
plt.title('Laplacian Filter Result')
plt.imshow(laplacian_edges, cmap='gray')
plt.axis('off')

# High-boost enhanced image
plt.subplot(2, 2, 4)
plt.title('High Boost Enhanced Image')
plt.imshow(high_boost_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Description of the High-Boost Filter:

In this program, a **high-boost filter** is implemented by combining Sobel and Laplacian filters to enhance details and edges in the image. The main purpose of the high-boost filter is to amplify fine details and edges, which is especially useful for highlighting key features in the image, such as bone structures.

Procedure:

1. **Gaussian Denoising:** First, the input image is processed with a Gaussian filter to reduce noise. This step produces a blurred image, which helps reduce false edges introduced during the high-boost process.
2. **Conversion to Grayscale:** The blurred image is converted to grayscale in preparation for edge detection, as edge detection is typically performed on grayscale images.
3. **Sobel Filter:** The Sobel filter is applied to detect horizontal and vertical edges in the image. It calculates gradients in both directions to highlight edges, generating an edge intensity map.
4. **Laplacian Filter:** The Laplacian filter is then used for edge detection. Based on the second derivative, it detects areas of rapid change in the image, particularly around edges.
5. **Edge Combination:** The results from the Sobel and Laplacian filters are combined to capture edges detected by both methods. This produces a more comprehensive edge-enhanced image.
6. **High-Boost Image:** Finally, the original grayscale image is combined with the enhanced edge image, with the parameter alpha controlling the strength of the enhancement. A higher alpha value results in stronger edge emphasis.

Final Enhanced Image:

The high-boosted image has sharper edges and more pronounced details, especially in areas that may have been less noticeable in the original image. This kind of enhancement is suitable for applications that require accentuation of edges or fine details, such as analyzing bone structures in medical imaging.

The program uses the following mathematical formulas for image processing:

1. Sobel Filter Formula:

The Sobel filter is a first-order derivative-based edge detection technique that computes the gradient in both horizontal and vertical directions. The corresponding convolution kernels are:

- Horizontal direction (Sobel-X):

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

- Vertical direction (Sobel-Y):

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

After applying these filters, the horizontal and vertical gradients are calculated, and the gradient magnitude is computed using the following formula:

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

where M represents the edge intensity of the image, and G_x and G_y are the horizontal and vertical gradients, respectively.

2. Laplacian Filter Formula:

The Laplacian filter is based on the second derivative and is used to detect areas of rapid intensity changes (usually edges) in an image. The formula for a 5x5Laplacian kernel is:

$$L = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & -4 & 1 & -4 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & -4 & 1 & -4 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

This filter calculates intensity changes around a pixel, emphasizing the edges in the image.

3. High-Boost Filter Formula:

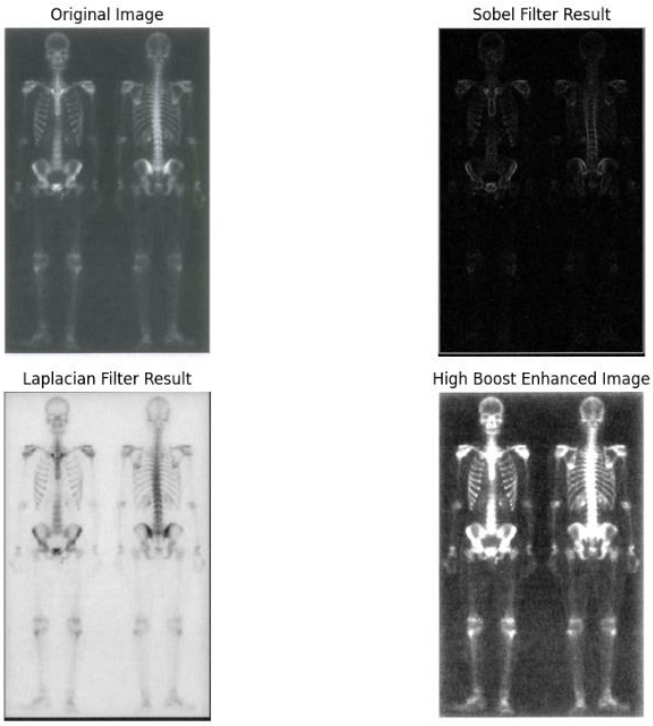
The high-boost filter is based on a linear combination of the original image and the edge image. The formula is as follows:

$$I_{\text{high-boost}} = (1 + \alpha) \cdot I_{\text{original}} - \alpha \cdot I_{\text{edges}}$$

where:

- $I_{\text{high-boost}}$ is the high-boosted image.
- I_{original} is the original image.
- I_{edges} is the edge image obtained from Sobel and Laplacian filters.
- α is the enhancement factor, controlling the degree of edge amplification.

This formula aims to add edge details back to the original image, where the value of α determines how much the details are emphasized or suppressed.



2. Repeat (1) steps in the image ‘fish.jpg’? (40)

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Define Sobel filter
def sobel_filter(image):
    # Define Sobel kernels (standard Sobel kernels)
    sobel_x = np.array([[-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]], dtype=np.float32)  # Horizontal direction

    sobel_y = np.array([[1,  2,  1],
                        [0,  0,  0],
                        [-1, -2, -1]], dtype=np.float32)  # Vertical direction

    # Apply the filter
    grad_x = cv2.filter2D(image, cv2.CV_64F, sobel_x)
    grad_y = cv2.filter2D(image, cv2.CV_64F, sobel_y)

    # Compute the gradient magnitude and normalize
    magnitude = np.sqrt(grad_x**2 + grad_y**2)
    magnitude = np.clip(magnitude, 0, 255).astype(np.uint8)  # Ensure result is within range
    return magnitude

# Define Laplacian filter (3x3 kernel)
def laplacian_filter(image):
    # Define Laplacian kernel (3x3 kernel)
    laplacian = np.array([[0,  1, 0],
                          [1, -4, 1],
                          [0,  1, 0]], dtype=np.float32)

    # Apply the filter
    laplacian_result = cv2.filter2D(image, cv2.CV_64F, laplacian)
    laplacian_result = np.clip(laplacian_result, 0, 255).astype(np.uint8)  # Ensure result is within range
    return laplacian_result
```

```python
# Define high-boost method
def high_boost(image, alpha=1.0):
    # Use Gaussian blur for noise reduction
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # Convert to grayscale
    gray_image = cv2.cvtColor(blurred_image, cv2.COLOR_BGR2GRAY)

    # Apply Sobel and Laplacian filters
    sobel_edges = sobel_filter(gray_image)
    laplacian_edges = laplacian_filter(gray_image)

    # Combine Sobel and Laplacian edges
    edges = np.clip(sobel_edges + 0.5 * laplacian_edges, 0, 255).astype(np.uint8)

    # Convert edges to color
    edges_colored = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

    # High-boost image, using the original color image as the base
    high_boost_image = cv2.addWeighted(image, 1 + alpha, edges_colored, -alpha, 0)
    return gray_image, sobel_edges, laplacian_edges, high_boost_image

# Main function
if __name__ == "__main__":
    # Load the image
    image = cv2.imread('fish.jpg')  # Replace with your image path

    if image is None:
        print("Unable to read the image, please check the file path.")
    else:
        # Apply high-boost method
        gray_image, sobel_edges, laplacian_edges, high_boost_image = high_boost(image)

        # Display the images
        plt.figure(figsize=(12, 8))
```

```python
# Original image
plt.subplot(2, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert color channels
plt.axis('off')

# Sobel filter result
plt.subplot(2, 2, 2)
plt.title('Sobel Filter Result')
plt.imshow(sobel_edges, cmap='gray')
plt.axis('off')

# Laplacian filter result
plt.subplot(2, 2, 3)
plt.title('Laplacian Filter Result')
plt.imshow(laplacian_edges, cmap='gray')
plt.axis('off')

# High boost enhanced image
plt.subplot(2, 2, 4)
plt.title('High Boost Enhanced Image')
plt.imshow(cv2.cvtColor(high_boost_image, cv2.COLOR_BGR2RGB))  # Ensure color channel conversion
plt.axis('off')
plt.tight_layout()
plt.show()
```

High Boost Method Design

The purpose of the high boost method is to emphasize the edges and details in an image using the Sobel and Laplacian filters, thereby improving the contrast and visibility of the image. The specific steps are as follows:

Design of the High Boost Filter:

1. **Sobel Filter**:
    - The Sobel filter uses two directional convolution kernels to calculate the gradients in the horizontal and vertical directions. These two convolution kernels are defined as follows:
        - Horizontal gradient:

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

        - Vertical gradient:

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

    - The calculation of the **gradient magnitude** is expressed as:

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

    - This allows for clear extraction of edges in the image.

Laplacian Filter:

- The Laplacian filter is a second-order derivative filter that uses a 3x3 kernel to detect rapid changes in brightness in the image, thereby enhancing boundaries. Its 3x3 kernel is defined as:

$$\text{Laplacian Kernel} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Steps of the High Boost Process:

1. **Noise Removal**: Use a Gaussian filter to denoise the original image, reducing random noise while preserving the main structural information.
2. **Grayscale Conversion**: Convert the denoised image to grayscale, which facilitates better application of the Sobel and Laplacian filters.
3. **Application of Sobel and Laplacian Filters**:
    - Apply the Sobel filter to extract edge information in the horizontal and vertical directions, and apply the Laplacian filter to enhance the edges in the image.
    - The results of both filters are combined to obtain more complete edge information.
4. **Edge Enhancement and Fusion**:
    - Fuse the edge results from the Sobel and Laplacian filters, and convert them back to a color image.
    - This step enhances the edge features and combines them with the original image, making the details clearer.
5. **Generation of High Boost Image**: Finally, perform a linear addition of the edge image with the original image, represented mathematically as:

$$I_{\text{high-boost}} = (1 + \alpha) \cdot I_{\text{original}} - \alpha \cdot I_{\text{edges}}$$

Where:

- $I_{\text{high-boost}}$ is the enhanced image.
- I_{original} is the original image.
- I_{edges} is the edge detection result.
- α is the boost factor that controls the enhancement effect.

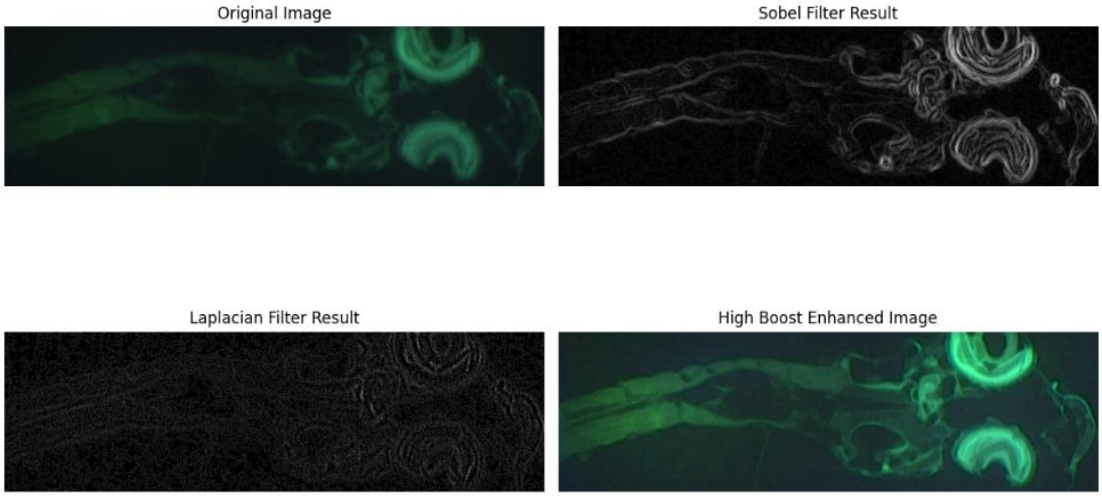
Explanation of the Final Enhanced Image:

After applying this high boost method, the edges become clearer, and the details of the entire image are effectively improved. The Sobel filter strengthens horizontal and vertical edges, while the Laplacian further enhances edge sharpness. Finally, through the high boost method, both the contrast and details of the entire image are significantly improved.

The specific output images of this program include:

- Original Image
- Sobel Filter Result
- Laplacian Filter Result
- High Boost Image (combining Sobel and Laplacian edges)

This method effectively extracts edges and enhances important details in the image, making it suitable for applications that require enhanced image details.



3. Please comment and compare your two designed filters and results ?

(20)

1.Sobel Filter Both codes use the Sobel filter with the same horizontal and vertical kernels (sobel_x and sobel_y) to detect edges in the image. During the gradient calculation process, the results are normalized to the range of 0 to 255 to facilitate easier display and further processing.

Comparison:

- In the Sobel filter part, the implementation in both codes is almost identical, so the results in this section are visually the same.

2.Laplacian Filter The main difference between the two codes lies in the design of the Laplacian filter. The first code uses a larger 5x5 kernel, while the second code uses a standard 3x3 kernel.

Comparison:

- **5x5 kernel (first code):** This can capture more details but may also introduce more noise and false edges. The larger kernel provides finer edge detection.
- **3x3 kernel (second code):** This is a more common Laplacian kernel, offering faster computation with less noise, but it might miss some subtle details. It is suitable for scenarios where noise reduction is essential.

3. **High-Boost Filtering** There are also some differences in the implementation of the high-boost filter between the two codes:

- The first code combines the results of the Sobel and Laplacian filters directly and enhances them based on the grayscale image.
- The second code adds the Laplacian filter result with a weight of 0.5 before combining it and then converts the edges to a color image, using the color image as the base for enhancement.

Comparison:

- **First code:** Uses the grayscale image for high-boost filtering, making it more suitable for emphasizing edge details.

- **Second code:** Uses the original color image as the base for high-boost filtering, providing a more natural visual effect and preserving the color information of the image.

Summary

1. **Filter design:** The first code's 5x5 Laplacian kernel is more suitable for cases requiring fine edge detection, while the second code's 3x3 kernel is better for balancing noise and edge detection.
2. **High-boost filter design:** The second code retains the color information of the image during high-boost filtering, making it more visually appealing, while the first code focuses more on edge enhancement.