TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TPHCM
KHOA CÔNG NGHỆ THÔNG TIN
HỆ: CHÍNH QUI
MÔN: XỬ LÝ ẢNH VÀ VIDEO SỐ

BÁO CÁO

ĐỒ ÁN CUỐI KỲ

(Lab 03)

TP.HCM, ngày 28 tháng 12 năm 2020

MỤC LỤC

I. Thông tin nhóm:

STT	MSSV	Họ tên	Vai trò	Nhiệm vụ
1	1712732	Thái Bá Sơn	Nhóm trưởng	Thực hiện câu 2, 3. Tổng hợp báo cáo.
2	1712724	Huỳnh Công Sinh	Thành viên	Thực hiện câu 1.
3	18120363	Đặng Văn Hiển	Thành viên	Thực hiện câu 4, 5.
4	18120647	Lê Thanh Viễn	Thành viên	Thực hiện câu 6, 7.

I. Đánh giá:

Câu	Yêu cầu	Điểm	Mức độ hoàn thành
1	Hàm tính tích chập hai ảnh	3	100%
2	Lọc trung bình	1	100%
3	Lọc trung vị	2	100%
4	Làm trơn ảnh bằng Gauss	2	100%
5	Phát hiện biên cạnh bằng Sobel	2	100%
6	Phát hiện biên cạnh bằng Prewitt	2	100%
7	Phát hiện biên cạnh bằng Laplace	2	100%

II. Nội dung báo cáo:

a. Cấu trúc chương trình:

- **Convolution.h**: Thư viện tính convolution của ảnh.
- **Convolution.cpp**: Chứa hàm tính convolution của ảnh.

  Hàm **GetKernel**:

  - Tham số: không có
  - Công dụng: lấy giá trị của kernel trong class Convolution.
  - Output: một vector số thực chứa giá trị của kernel.

  Hàm **SetKernel**:

  - Tham số: Vector số thực, 2 số nguyên là width và height.
  - Công dụng: tạo một kernel với thông số truyền vào.
  - Output: không có.

  Hàm **DoConvolution**:

  - Tham số: 2 ảnh gồm ảnh nguồn và ảnh đích.
  - Công dụng: áp dụng phép toán Convolution vào ảnh nguồn và lưu kết quả vào ảnh đích.
  - Output: 1 nếu thất bại, 0 nếu tính thành công.
  - Cách thực hiện:
    - Xét từng pixel của ảnh nguồn (bỏ đi một vài pixel biên để thuận lợi hơn cho việc tính toán) và áp dụng phép tính Convolution với kernel đã được thiết lập từ trước.
    - Nếu giá trị của phép tính lớn hơn 255 thì giới hạn giá trị tại 255.
    - Việc xử lý ở một kênh màu tương tự như ở 3 kênh màu.

  .

- **Blur.h**: Thư viện làm trơn ảnh.
- **Blur.cpp**: Chứa hàm làm trơn ảnh (lọc trung bình, lọc trung vị, Gauss).

- *Làm trơn ảnh bằng tính trung bình:* Đưa vào kernel tham số mean_value

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \cdot & \cdot & \cdot & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

có công thức

```
int mean_blurring(const Mat& sourceImage, Mat& destinationImage, int kWidth, int kHeight)
{
    if (!sourceImage.data)
    {
        return 1;
    }
    Convolution kernel;
    vector<float> mean_value(kWidth * kHeight, float(1) / float(kWidth * kHeight));
    kernel.SetKernel(mean_value, kWidth, kHeight);
    return kernel.DoConvolution(sourceImage, destinationImage);
}
```

- *Làm trơn ảnh bằng tính trung vị:*

```
// Tạo ảnh đích với kích cỡ cũ
destinationImage.create(sourceImage.rows, sourceImage.cols, sourceImage.type());

//Tạo bảng offsets trỏ tới các phần tử pixel cần tính trung vị
vector<int> offsets;
for (int y = -kHeight / 2; y <= kHeight / 2; y++)
    for (int x = -kWidth / 2; x <= kWidth / 2; x++)
        offsets.push_back(y * sourceImage.step[0] + x * nchannel);

// Matran trung vi
vector<uchar> h;
uchar* pDes, * pSrc;
for (int i = kHeight / 2; i < sourceImage.rows - kHeight / 2; i++)
{
    pDes = (uchar*)destinationImage.ptr<uchar>(i);
    pSrc = (uchar*)sourceImage.ptr<uchar>(i);
    for (int j = kWidth / 2 * nchannel; j < (sourceImage.cols - kWidth / 2) * nchannel; j++, pSrc += 1, pDes += 1)
    {
        // Tạo mảng cho matran trung vị
        for (int offset : offsets)
            h.push_back(pSrc[offset]);

        // Sắp xếp mảng
        InsertionSort(h);

        // Gán giá trị pixel với trung vị của mảng
        *pDes = h[h.size() / 2];
        h.clear();
    }
}
```

- *Làm trơn ảnh bằng Gauss*: làm trơn ảnh giảm nhiễu giúp cải thiện phát hiện biên cạnh nhưng đồng thời cũng làm mờ nó, với toán tử Gauss nó giữ lại biên cạnh tốt hơn toán tử trung bình.

Code:

<u>Bước 1:</u> Ta tạo kernel với $h(i,j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2+j^2}{2\sigma^2}}$ giá trị phần tử (với sigma được chọn là 1)

```
//Tính kernel
for (int y = -kHeight / 2; y <= kHeight / 2; y++)
{
    for (int x = -kWidth / 2; x <= kWidth / 2; x++)
    {
        h = expf(-(y * y + x * x) / sigma2) / k;
        kernel.push_back(h);
        sum += h;
    }
}
```

Bước 2: Chuẩn hóa kernel - tổng các phần tử kernel = 1

```
//Chuẩn hóa kernel
for (float& x : kernel) {
    x /= sum;
}
```

Bước 3: Tính tích chập ảnh input với kernel trên ghi lên ảnh output, dựa trên class Convolution đã được cung cấp.

```
// Set kernel
conv.SetKernel(kernel, kWidth, kHeight);
// Tính tích chập với kernel trên
return conv.DoConvolution(sourceImage, destinationImage);
```

- **EdgeDetector.h**: Chứa hàm phát hiện biên cạnh (Sobel, Prewitt, Laplace).

  Khởi tạo ma trận của các toán tử:

```
// Khoi tao ma tran Wx cua toan tu Sobel
vector<float> tempSobelKernel1 = { -1, 0, 1,
                                   -2, 0, 2,
                                   -1, 0, 1 };

// Khoi tao ma tran Wy cua toan tu Sobel
vector<float> tempSobelKernel2 = { -1,-2,-1,
                                    0, 0, 0,
                                    1, 2, 1 };

// Khoi tao ma tran Wx cua toan tu Prewitt
vector<float> tempPrewittKernel1 = { 5, 5, 5,
                                    -3, 0,-3,
                                    -3,-3,-3 };

// Khoi tao ma tran Wy cua toan tu Prewitt
vector<float> tempPrewittKernel2 = { 5, -3, -3,
                                     5, 0, -3,
                                     5, -3, -3 };

// Khoi tao ma tran cua toan tu Laplace
vector<float> tempLaplaceKernel = { 0, 1, 0,
                                    1, -4, 1,
                                    0, 1, 0 };
```

Hàm phá hiện biên cạnh bằng Sobel

```
// Ham phat hien bien canh bang Sobel
int Sobel(const Mat& sourceImage, Mat& destinationImage)
{
    int nRow = sourceImage.rows;
    int nCol = sourceImage.cols;
    Convolution SobelKernel;
    Mat Img1, Img2;

    // Ma tran Sobel 1
    SobelKernel.SetKernel(tempSobelKernel1, 3, 3);

    SobelKernel.DoConvolution(sourceImage, Img1);
    // Ma tran Sobel 2
    SobelKernel.SetKernel(tempSobelKernel2, 3, 3);

    SobelKernel.DoConvolution(sourceImage, Img2);

    // Tạo mma tran de luu gi tri pixel
    destinationImage.create(Size(nCol, nRow), CV_8UC1);
```

```
    for (int i = 0; i < nRow; i++)
    {
        // Lay dia chi dong cua anh dich de luu ket qua vao
        uchar* data = destinationImage.ptr<uchar>(i);

        // Lay dia chi dong cua Image1
        uchar* data1 = Img1.ptr<uchar>(i);

        //Lay dia chi dong cua Image2
        uchar* data2 = Img2.ptr<uchar>(i);

        // Gan tong gia tri tuyet doi cua hai mang vua tich chap vao anh dich
        for (int j = 0; j < nCol; j++)
        {
            int val = 0;

            if (data1[j] >= 0) val += data1[j];
            else val += data1[j];

            if (data2[j] >= 0) val += data2[j];
            else val += data2[j];

            // Gan gia tri cho ma tran dich
            data[j] = val;
        }
    }

    return 0;
}
```

Hàm phát hiện biên cạnh bằng Prewitt:

```
// Ham phat hien bien canh bang Prewitt
int Prewitt(const Mat& sourceImage, Mat& destinationImage)
{
    int nRow = sourceImage.rows;
    int nCol = sourceImage.cols;
    Convolution PrewittKernel;
    Mat Img1, Img2;

    // Ma tran Prewitt 1
    PrewittKernel.SetKernel(tempPrewittKernel1, 3, 3);

    PrewittKernel.DoConvolution(sourceImage, Img1);
    // Ma tran Prewitt 2
    PrewittKernel.SetKernel(tempPrewittKernel2, 3, 3);

    PrewittKernel.DoConvolution(sourceImage, Img2);

    // Tao ma tran de luu gia tri pixel
    destinationImage.create(Size(nCol, nRow), CV_8UC1);
```

```
    for (int i = 0; i < nRow; i++)
    {
        // Lay dia chi dong cua anh dich de luu ket qua vao
        uchar* data = destinationImage.ptr<uchar>(i);

        // Lay dia chi dong cua Image1
        uchar* data1 = Img1.ptr<uchar>(i);

        //Lay dia chi dong cua Image2
        uchar* data2 = Img2.ptr<uchar>(i);

        // Gan tong gia tri tuyet doi cua hai mang vua tich chap vao anh dich
        for (int j = 0; j < nCol; j++)
        {
            int val = 0;

            if (data1[j] > 0) val += data1[j];
            else val += -data1[j];

            if (data2[j] > 0) val += data2[j];
            else val += -data2[j];

            // Gan gia tri cho ma tran dich
            data[j] = val;
        }
    }

    return 0;
}
```

Hàm phát hiện biên cạnh bằng Laplace:

```
// Ham phat hien bien canh bang Laplace
int Laplace(const Mat& sourceImage, Mat& destinationImage)
{
    Convolution LaplaceKernel;

    // Ma tran Laplace
    LaplaceKernel.SetKernel(tempLaplaceKernel, 3, 3);

    LaplaceKernel.DoConvolution(sourceImage, destinationImage);
    return 0;
}
```

Hàm gọi các hàm chức năng:

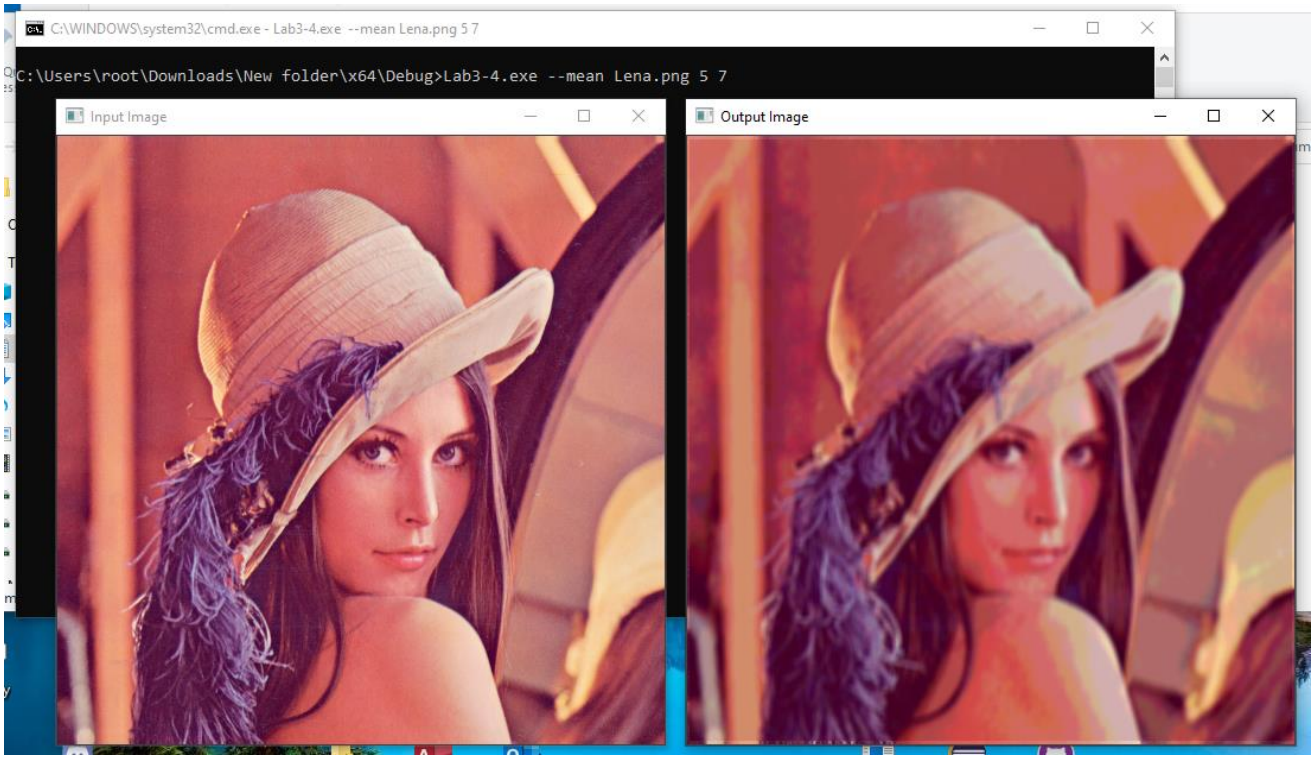
```cpp
// Ham goi cac ham chuc nang
int DetectEdge(const Mat& sourceImage, Mat& destinationImage, int kWidth, int kHeight, int method) {
    if (sourceImage.ptr(0) != NULL)
    {
        EdgeDetector func;
        if (method == 1) func.Sobel(sourceImage, destinationImage);
        else if (method == 2) func.Prewitt(sourceImage, destinationImage);
        else if (method == 3) func.Laplace(sourceImage, destinationImage);
        else return 1;
        return 0;
    }
    return 1;
}
```

- **Lab3-4.cpp**: File thực thi.

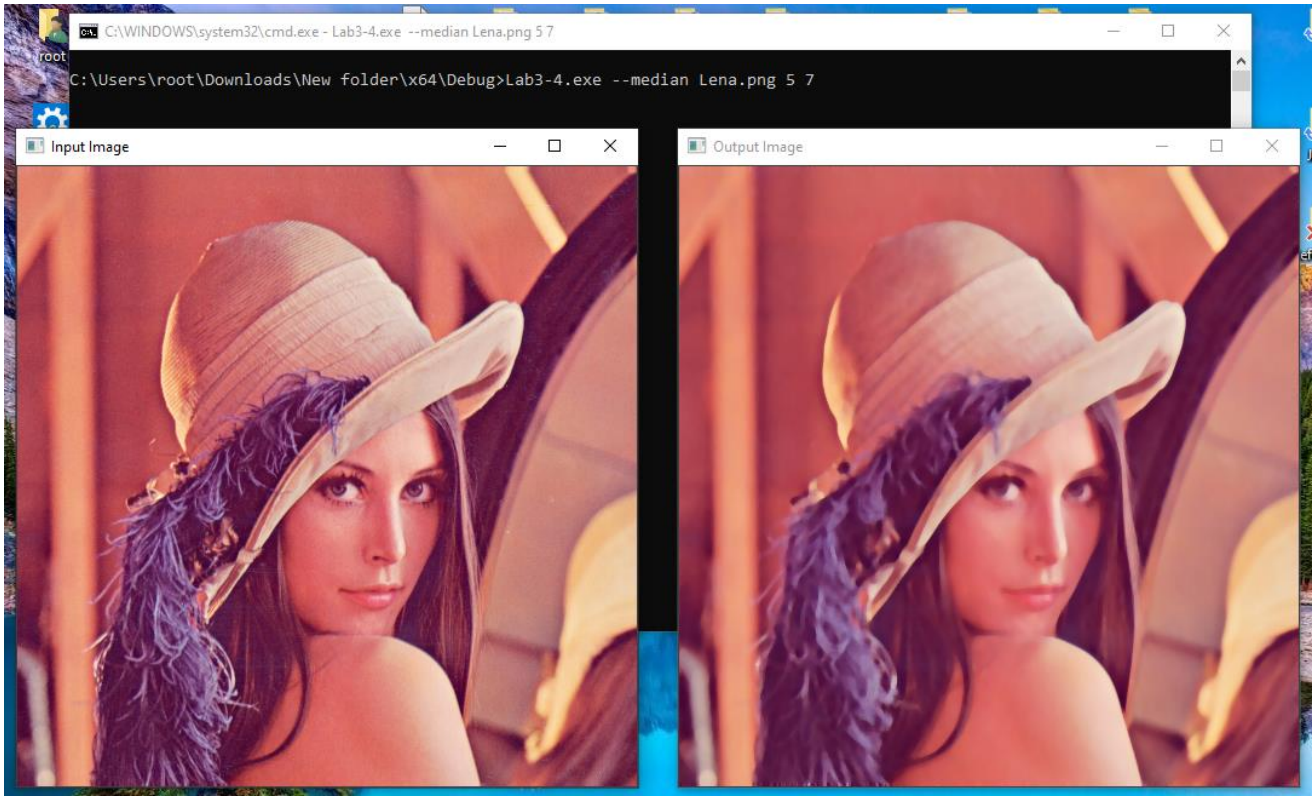
b. Hướng dẫn chạy:

- *Tham số dòng lệch:* <Program.exe> <Command> <Interpolate> <InputPath> <CmdArguments>
- Truy xuất vào file .exe theo đường dẫn với cú pháp: cd + <đường dẫn>.

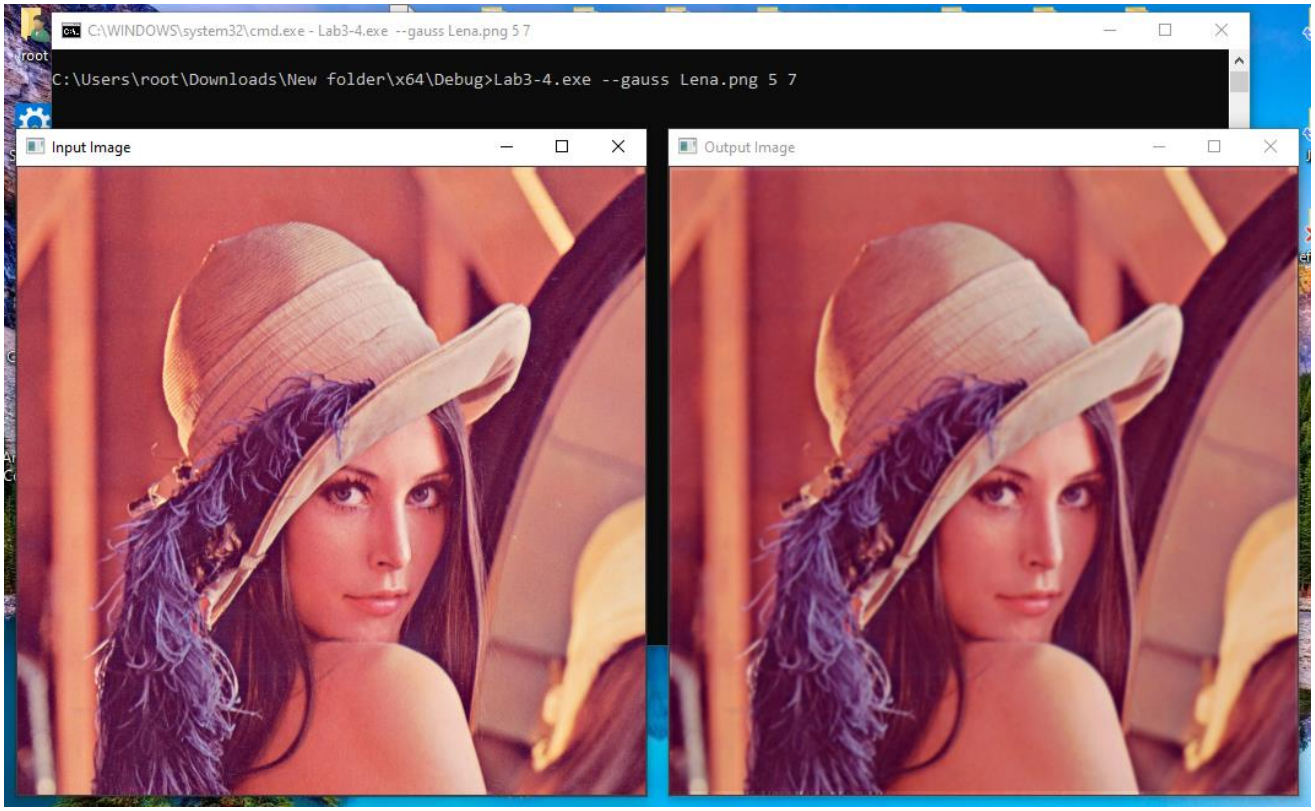
(i) Yêu cầu 02 - Lọc trung bình:



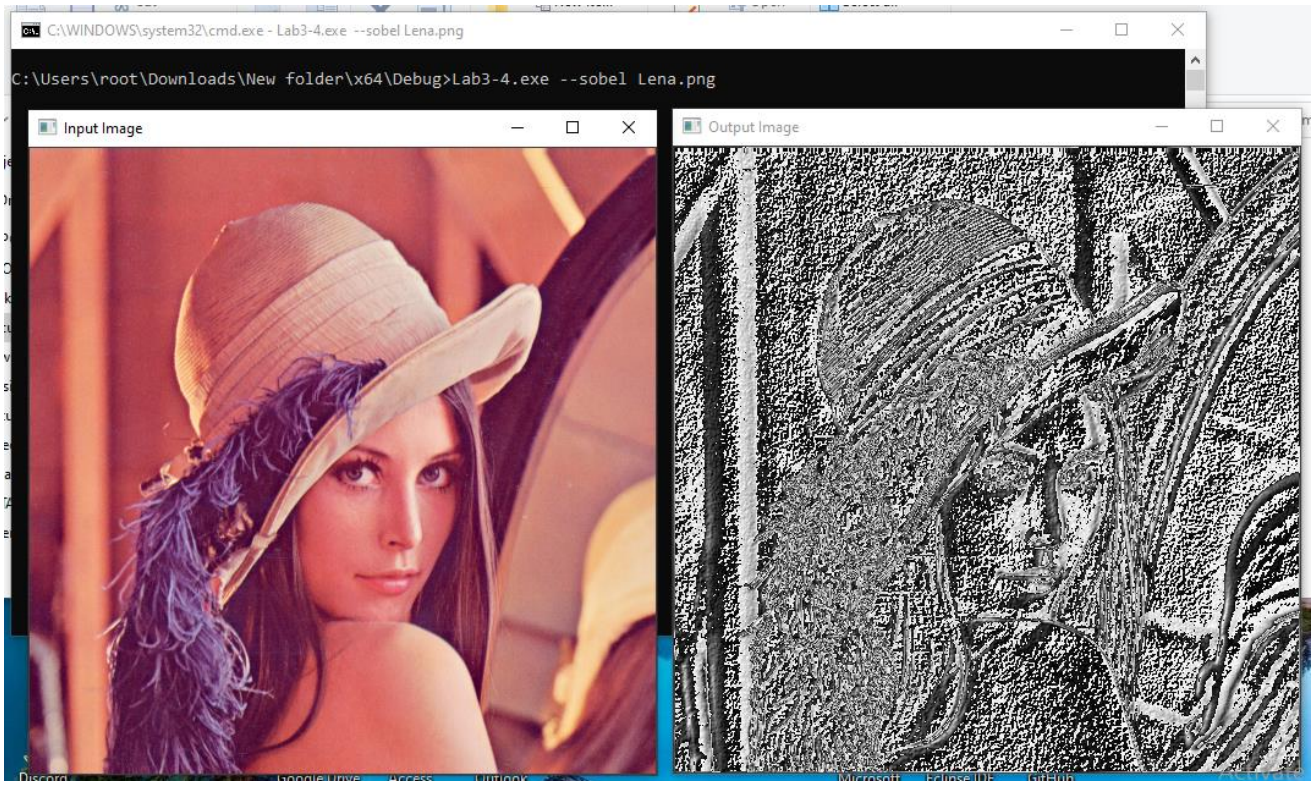
(ii) Yêu cầu 03 - Lọc trung vị:



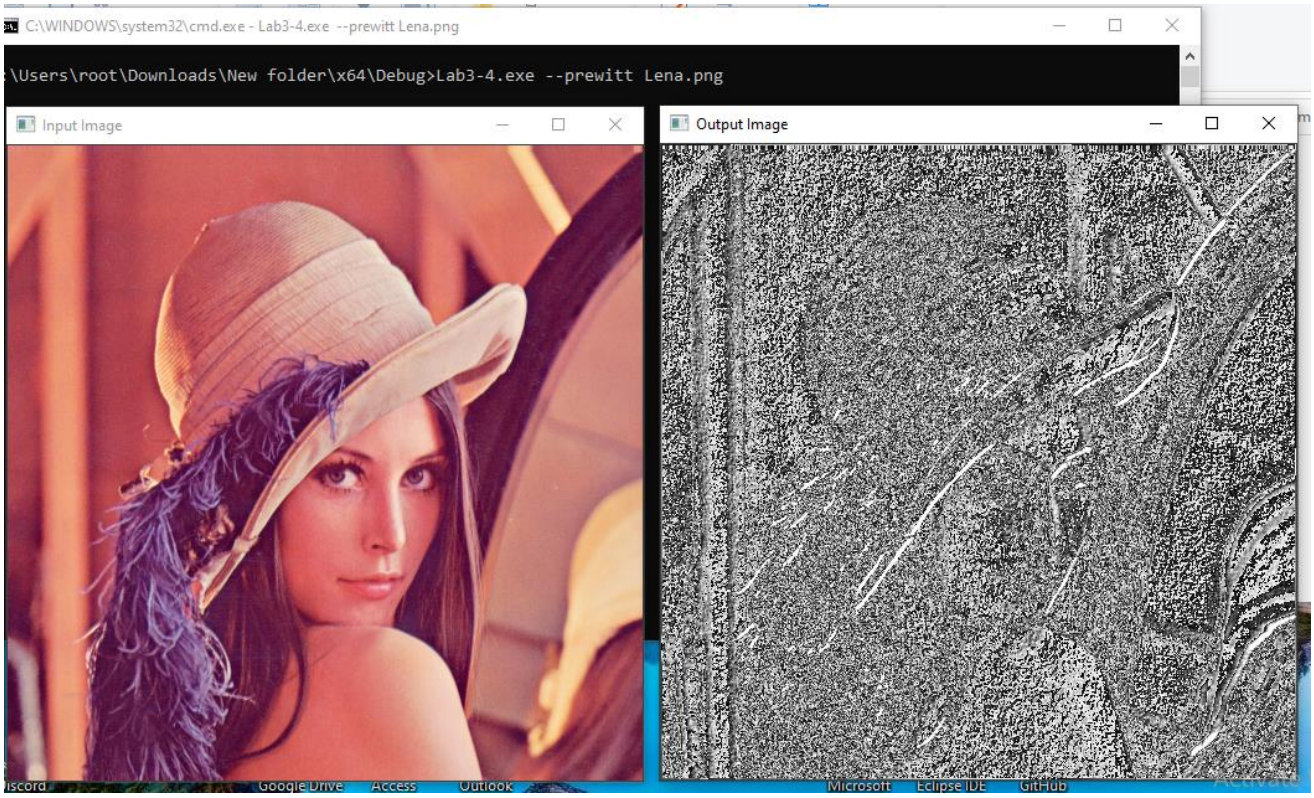
(iii) Yêu cầu 04 – Làm trơn ảnh bằng Gauss:



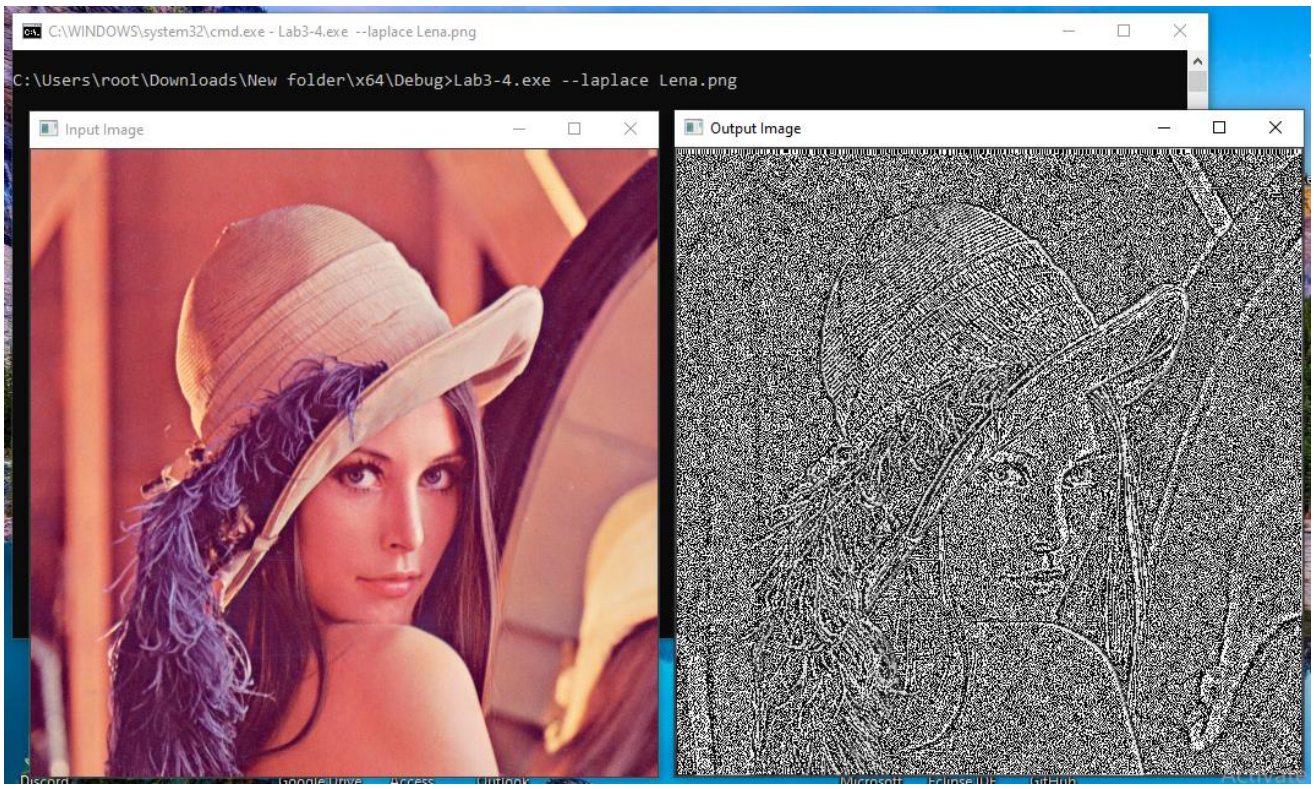
(iv) Yêu cầu 05 – Phát hiện biên cạnh bằng Sobel:



(v) Yêu cầu 06 - Phát hiện biên cạnh bằng Prewitt:



(vi) Yêu cầu 07 - Phát hiện biên cạnh bằng Laplace:



III. Nguồn/Tài liệu tham khảo:

- https://aicurious.io/posts/2018-09-29-loc-anh-image-filtering/