

Hoàng Minh Huy – 19120241 – [19120241@student.hcmus.edu.vn](mailto:19120241@student.hcmus.edu.vn) - 0902552446

Bạch Ngọc Minh Tâm – 19120034 – [19120034@student.hcmus.edu.vn](mailto:19120034@student.hcmus.edu.vn) - 0385606705

## BÁO CÁO BT01 – EDGE DETECTION

### A. Các thuật toán nhận diện cạnh đã cài đặt

#### I. Thuật toán Sobel:

##### 1. Ý tưởng:

Một điểm ảnh được gọi là “thuộc một cạnh” khi nó có sự thay đổi màu sắc rõ rệt so với các điểm ảnh xung quanh. Ngược lại, những điểm ảnh có cùng màu với các điểm ảnh xung quanh thì không được gọi là “điểm thuộc cạnh”. Điều này đồng nghĩa, độ lớn của giá trị đạo hàm cấp 1 của điểm ảnh thuộc cạnh sẽ lớn, trong khi một điểm không thuộc cạnh sẽ bằng 0. Do đó, ta sẽ tính độ lớn vector gradient của từng điểm ảnh và lưu lại chúng để xác định các cạnh.

Do ảnh là mảng 2 chiều, ta sẽ tính độ lớn vector gradient của từng điểm ảnh bằng cách tính độ lớn đạo hàm của chúng theo từng phương riêng (phương Ox và phương Oy) rồi tính độ lớn của vector gradient thông qua công thức  $G = \sqrt{G_x^2 + G_y^2}$ , với  $G_x, G_y$  lần lượt là độ lớn đạo hàm của điểm ảnh theo phương Ox và phương Oy.

Để tính đạo hàm, thuật toán Sobel thực hiện tích chập điểm ảnh cần tính đạo hàm và các điểm xung quanh với một kernel (hay filter) có kích thước 3 x3. Với thuật toán Sobel, kernel đạo hàm

theo phương Ox là  $D_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$  và theo phương Oy là  $D_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

##### 2. Cài đặt: Thuật toán Sobel được cài đặt như sau

```
int Detect_by_Sobel(cv::Mat &src, cv::Mat &dest)
{
    if (src.data == NULL)
        return 0;

    int c = src.cols, r = src.rows;
    dest = Mat(c, r, CV_8UC1);

    if (c < 3 || r < 3)
    {
        cout << "Kích thước ảnh không phù hợp !";
        return 0;
    }
    else
    {

```

```

        vector<vector<int>> x_filter = { { -1, 0, 1 }, { -2, 0, 2 }, { -1, 0, 1 }
};
        vector<vector<int>> y_filter = { { 1, 2, 1 }, { 0, 0, 0 }, { -1, -2, -1 }
};

        Mat x_grad = Mat(r, c, CV_8UC1);
        Mat y_grad = Mat(r, c, CV_8UC1);

        for (int i = 1; i < r - 1; i++)
        {
            for (int j = 1; j < c - 1; j++)
            {
                int dx = (int)(Grad_Operation(src, x_filter, i, j));
                int dy = (int)(Grad_Operation(src, y_filter, i, j));
                x_grad.at<uchar>(i, j) = dx;
                y_grad.at<uchar>(i, j) = dy;
                dest.at<uchar>(i, j) = (int)(sqrt(dx*dx + dy*dy));
            }
        }

        Gaussian_blur(dest);

        namedWindow("Display x-grad", WINDOW_AUTOSIZE); // (3)
        imshow("Display x-grad", x_grad); // (4)

        namedWindow("Display y-grad", WINDOW_AUTOSIZE); // (3)
        imshow("Display y-grad", y_grad); // (4)

        namedWindow("Display grad", WINDOW_AUTOSIZE); // (3)
        imshow("Display grad", dest); // (4)

        return 1;
    }
}

```

Hàm `int Detect_by_Sobel(cv::Mat &src, cv::Mat &dest)` thực thi thuật toán này theo trình tự như sau:

- Tham số đầu vào: Hàm nhận vào ảnh src cần nhận diện cạnh, thực hiện thuật toán Sobel và lưu lại kết quả trên ảnh dest. Ảnh src nhận vào cần phải là ảnh grayscale.
- Đầu tiên, hàm kiểm tra điều kiện của ảnh. Nếu ảnh không có dữ liệu hoặc một trong hai chiều của ảnh có kích thước bé hơn 3 thì thuật toán không thể thực hiện được (do kích thước kernel là 3 x 3 nên kích thước tối thiểu của ảnh phải là 3 x 3). Hàm xuất thông báo và kết thúc.
- Nếu kích thước của ảnh lớn hơn kernel, ta lưu lại giá trị của  $D_x$  và  $D_y$  vào 2 biến `x_filter` và `y_filter` và tạo ra 2 ảnh `x_grad`, `y_grad` để lưu lại giá trị đạo hàm theo phương  $O_x$  và  $O_y$  của ảnh.

- Lần lượt duyệt qua tất cả các điểm ảnh khác các điểm ảnh nằm ngoài rìa, tính giá trị đạo hàm theo phương x (lưu lại vào biến tạm dx và ảnh x\_grad) và phương y (lưu lại vào biến tạm dy và ảnh y\_grad) thông qua hàm Grad\_Operation (sẽ giải thích ở phần sau) rồi tính độ lớn vector gradient của điểm ảnh và lưu lại vào dest.
- Sau khi tính, hàm thực hiện làm mờ ảnh để khử bớt đi những “cạnh giả” (có được do nhiễu, không thật sự là cạnh) thông qua hàm Gaussian\_blur (sẽ được giải thích ở phần sau) và xuất ra 3 ảnh x\_grad, y\_grad, dest.
- Kiểu trả về: hàm trả về 0 nếu thực hiện thất bại và trả về 1 nếu thực hiện thành công.

## II. Thuật toán Prewitt:

- Ý tưởng:** tương tự Sobel, tuy nhiên Prewitt sử dụng một kernel khác với Sobel. Trong thuật toán Prewitt, kernel đạo hàm kích thước 3 x 3 theo phương Ox là  $D_x =$

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ và theo phương Oy là } D_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

- Cài đặt:** Thuật toán Prewitt được cài đặt như sau

```
int Detect_by_Prewitt(cv::Mat &src, cv::Mat &dest)
{
    if (src.data == NULL)
        return 0;

    int c = src.cols, r = src.rows;
    dest = Mat(c, r, CV_8UC1);

    if (c < 3 || r < 3)
    {
        cout << "Kích thước ảnh không phù hợp !";
        return 0;
    }
    else
    {
        vector<vector<int>> x_filter = { { -1, 0, 1 }, { -1, 0, 1 }, { -1, 0, 1 } };
        vector<vector<int>> y_filter = { { 1, 1, 1 }, { 0, 0, 0 }, { -1, -1, -1 } };

        Mat x_grad = Mat(r, c, CV_8UC1);
        Mat y_grad = Mat(r, c, CV_8UC1);

        for (int i = 1; i < r - 1; i++)
        {
            for (int j = 1; j < c - 1; j++)
            {
                x_grad.at<uchar>(i, j) = Grad_Operation(src, x_filter, i, j);
            }
        }
    }
}
```

```

        y_grad.at<uchar>(i, j) = Grad_Operation(src, y_filter, i, j);
        dest.at<uchar>(i, j) = (int)(sqrt(
            Grad_Operation(src, x_filter, i, j) * Grad_Operation(src,
x_filter, i, j)
            + Grad_Operation(src, y_filter, i, j)* Grad_Operation(src,
y_filter, i, j)));
    }
}

Gaussian_blur(dest);

namedWindow("Display x-grad", WINDOW_AUTOSIZE); // (3)
imshow("Display x-grad", x_grad); // (4)

namedWindow("Display y-grad", WINDOW_AUTOSIZE); // (3)
imshow("Display y-grad", y_grad); // (4)

namedWindow("Display grad", WINDOW_AUTOSIZE); // (3)
imshow("Display grad", dest); // (4)

return 1;
}
}

```

Về cơ bản, các bước thực hiện của hàm `int Detect_by_Prewitt(cv::Mat &src, cv::Mat &dest)` giống hệt với hàm `int Detect_by_Sobel(cv::Mat &src, cv::Mat &dest)`, chỉ khác mỗi giá trị của `x_filter` và `y_filter`.

### III. Thuật toán Laplace:

#### 1. Ý tưởng:

Về cơ bản, thuật toán Laplace cũng nhận diện cạnh dựa trên mức độ biến thiên của màu sắc điểm ảnh như Sobel, nhưng thay vì chỉ tính đạo hàm bậc 1, thuật toán Laplace tính đạo hàm bậc 2 theo cả 2 phương rồi mới tính vector gradient.

Với thuật toán Laplace, kernel đạo hàm theo phương Ox và theo phương Oy là như nhau. Có 2 kernel kích thước 3 x 3 thường được dùng là  $K_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ ,  $K_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ .

Trong bài tập này, nhóm sử dụng kernel  $K_2$

Do tính đạo hàm bậc 2 nên thuật toán Laplace rất nhạy cảm với nhiễu. Do đó, trước khi thực hiện thuật toán ta cần khử nhiễu ảnh.

#### 2. Cài đặt: Thuật toán Laplace được cài đặt như sau

```

int Detect_by_Laplace(cv::Mat &src, cv::Mat &dest)
{
    if (src.data == NULL)
        return 0;

    int c = src.cols, r = src.rows;
    dest = Mat(c, r, CV_8UC1);

    if (c < 3 || r < 3)
    {
        cout << "Kich thuoc anh khong phu hop !";
        return 0;
    }
    else
    {
        Gaussian_blur(src);
        vector<vector<int>> x_filter = { { -1, -1, -1 }, { -1, 8, -1 }, { -1, -1, -
1 } } };
        vector<vector<int>> y_filter = { { -1, -1, -1 }, { -1, 8, -1 }, { -1, -1, -
1 } } };
        Mat x_grad = Mat(r, c, CV_8UC1);
        Mat y_grad = Mat(r, c, CV_8UC1);

        for (int i = 1; i < r - 1; i++)
        {
            for (int j = 1; j < c - 1; j++)
            {
                x_grad.at<uchar>(i, j) = Grad_Operation(src, x_filter, i, j);
                y_grad.at<uchar>(i, j) = Grad_Operation(src, y_filter, i, j);
                dest.at<uchar>(i, j) = (int)(sqrt(
                    Grad_Operation(src, x_filter, i, j) * Grad_Operation(src,
x_filter, i, j)
                    + Grad_Operation(src, y_filter, i, j)* Grad_Operation(src,
y_filter, i, j)));
            }
        }

        namedWindow("Display x-grad", WINDOW_AUTOSIZE); // (3)
        imshow("Display x-grad", x_grad); // (4)

        namedWindow("Display y-grad", WINDOW_AUTOSIZE); // (3)
        imshow("Display y-grad", y_grad); // (4)

        namedWindow("Display grad", WINDOW_AUTOSIZE); // (3)
    }
}

```

```

    imshow("Display grad", dest); // (4)

    return 1;
}

```

Về cơ bản, Về cơ bản, các bước thực hiện của hàm `int Detect_by_Laplace(cv::Mat &src, cv::Mat &dest)` khá giống với hàm `int Detect_by_Sobel(cv::Mat &src, cv::Mat &dest)`. Khác biệt của hai hàm nằm ở giá trị của `x_filter` và `y_filter` và bước xử lý nhiễu diễn ra ở ảnh `src` ngay trước khi thực hiện thuật toán, thay vì diễn ra ở ảnh `dest` sau khi thuật toán đã được thực hiện.

## IV. Thuật toán Canny:

### 1. Ý tưởng:

Các thuật toán nhận diện cạnh ở 3 phần trước đều có những điểm yếu: không thể làm mỏng cạnh, khiến kết quả đầu ra không rõ; không xác định được điểm ảnh thật sự thuộc về cạnh nào.

Do đó, thuật toán Canny sẽ xử lý các nhược điểm trên bằng cách thực hiện thêm 2 bước hậu xử lý để kết quả đầu ra chất lượng hơn. Cụ thể, thuật toán gồm các bước sau:

- Làm mờ ảnh đầu vào để khử bớt nhiễu.
- Tính độ lớn vector gradient của các điểm ảnh, đồng thời xác định phương của vector gradient để xác định hướng của cạnh mà điểm ảnh thuộc về theo công thức  $\tan \alpha = \frac{G_y}{G_x}$ . Để thuận tiện cho việc xử lý, ta chỉ xác định 4 phương chính, mỗi phương được xác định bằng góc lệch của phương đó với phương thẳng đứng
  - Phương thẳng đứng: thể hiện bằng góc  $0^\circ$
  - Phương ngang: thể hiện bằng góc  $90^\circ$
  - Phương Tây Bắc – Đông Nam: thể hiện bằng góc  $45^\circ$
  - Phương Đông Bắc – Tây Nam: thể hiện bằng góc  $135^\circ$
- Thực hiện non-maximum supression: nếu một điểm thật sự thuộc một cạnh có phương  $\vec{u}$  thì nó sẽ có độ lớn gradient lớn nhất so với các điểm ảnh xung quanh theo phương  $\vec{v}$ , với  $\vec{u} \perp \vec{v}$ . Dựa trên phương của mỗi điểm ảnh đã xác định ở trên, ta sẽ kiểm tra 2 điểm kề nhau theo phương  $\vec{v}$  (về 2 phía của điểm ảnh). Nếu độ lớn gradient của điểm ảnh không phải lớn nhất, điều này chứng tỏ điểm ảnh đang xét không thật sự thuộc về một cạnh. Do đó, giá trị gradient của nó sẽ bị “đè” xuống (xóa về 0). Bước này giúp các cạnh trở nên mỏng và dễ nhìn hơn.
- Double threshold: sau khi thực hiện non-maximum supression, ta vẫn chưa thể thực sự nhận định được cạnh nào thật sự là cạnh và cạnh nào được sinh ra do nhiễu. Do đó, ta sẽ sử dụng hai ngưỡng chặn trên  $v_{max}$  và chặn dưới  $v_{min}$  để kiểm tra:
  - Nếu một điểm ảnh có độ lớn gradient  $> v_{max}$  thì điểm ảnh đó được đánh dấu là “cạnh mạnh” và chắc chắn sẽ được giữ lại.
  - Nếu một điểm ảnh có độ lớn gradient  $< v_{min}$  thì điểm ảnh đó sẽ bị “đè” xuống

- Nếu một điểm ảnh có độ lớn gradient nằm trong khoảng  $[v_{min}; v_{max}]$  thì ta sẽ đánh dấu nó là “cạnh yếu”. Khi đó, nếu điểm ảnh nằm liền kề với cạnh mạnh thì điểm ảnh được giữ lại. Ngược lại, điểm ảnh sẽ bị xóa.

**2. Cài đặt:** Thuật toán Canny được cài đặt như sau:

```
int Detect_by_Canny(cv::Mat &src, cv::Mat &dest, int upper_threshold, int
lower_threshold)
{
    if (src.data == NULL)
        return 0;

    int c = src.cols, r = src.rows;
    dest = Mat(c, r, CV_8UC1);

    if (c < 3 || r < 3)
    {
        cout << "Kich thuoc anh khong phu hop !";
        return 0;
    }
    else
    {
        // noise processing
        Gaussian_blur(src);
        vector<vector<int>> x_filter = { { -1, 0, 1 }, { -1, 0, 1 }, { -1, 0, 1 }
};
        vector<vector<int>> y_filter = { { 1, 1, 1 }, { 0, 0, 0 }, { -1, -1, -1 }
};

        // gradient matrix and direction detecting
        Mat x_grad = Mat(r, c, CV_8UC1);
        Mat y_grad = Mat(r, c, CV_8UC1);
        vector<vector<int>> angle;

        for (int i = 1; i < r - 1; i++)
        {
            vector<int> tmp;
            for (int j = 1; j < c - 1; j++)
            {
                // counting gradient using Prewitt kernel
                int dx = Grad_Operation(src, x_filter, i, j);
                int dy = Grad_Operation(src, y_filter, i, j);
                x_grad.at<uchar>(i, j) = (uchar)(dx);
                y_grad.at<uchar>(i, j) = (uchar)(dy);
                dest.at<uchar>(i, j) = (int)(sqrt(dx*dx + dy*dy));
            }
        }
    }
}
```

```

        // detecting angle
        if (dx == 0)
            dx = 0.000001; // avoid dividing zero
        double a = 180.0 * atan2(1.0 * dy, 1.0 * dx) / pi;

        // round up the angle
        if (a < 0) a += 180;
        if (a < 22.5) a = 0;
        else if (a < 67.5) a = 45;
        else if (a < 112.5) a = 90;
        else if (a < 157.5) a = 135;
        else a = 180.0;
        tmp.push_back((int)(a));
    }
    angle.push_back(tmp);
}

// lower bound thresholding
for (int i = 1; i < c - 1; i++)
{
    for (int j = 1; j < r - 1; j++)
    {
        int a = angle[i - 1][j - 1];
        Non_maximum_supression(dest, i, j, a);
    }
}

// double thresholding
vector<vector<bool>> edge_or_noise = keep_or_supress(dest,
upper_threshold, lower_threshold);
for (int i = 1; i < c - 1; i++)
{
    for (int j = 1; j < r - 1; j++)
    {
        if (edge_or_noise[i - 1][j - 1] == false)
            dest.at<uchar>(i, j) = (uchar)(0);
    }
}

namedWindow("Display x-grad", WINDOW_AUTOSIZE); // (3)
imshow("Display x-grad", x_grad); // (4)

namedWindow("Display y-grad", WINDOW_AUTOSIZE); // (3)
imshow("Display y-grad", y_grad); // (4)

```



```

        namedWindow("Display grad", WINDOW_AUTOSIZE); // (3)
        imshow("Display grad", dest); // (4)

        return 1;
    }
}

```

Hàm thực hiện thuật toán có thể chia thành các pha sau:

- Đầu vào: ảnh nguồn src (ảnh grayscale) cần xác định cạnh, ảnh đích dest chứa kết quả và 2 ngưỡng chặn upper\_bound (chặn trên) và lower\_bound (chặn dưới)
- Kiểm tra điều kiện thực thi thuật toán: hết như 3 thuật toán trước đó.
- Khử nhiễu ảnh đầu vào: Gọi hàm Gaussian\_blur
- Tính gradient của các điểm ảnh: sử dụng thuật toán Prewitt để tính
- Xác định phương của các điểm ảnh dựa trên góc (đã trình bày ở trên), chuyển đổi sang đơn vị độ rồi thực hiện làm tròn để các điểm luôn thuộc về 1 trong 4 phương đã quy định.
- Non-maximum supression: ta duyệt qua tất cả các điểm ảnh. Thao tác Non-maximum supression ở mỗi điểm ảnh được thực hiện bởi hàm Non\_maximum\_supression (sẽ được trình bày ở phần sau).
- Double thresholding: ta xây dựng một ma trận 2 chiều kiểu bool edge\_or\_noise để xác định xem điểm nào sẽ được giữ lại (mang giá trị true) và điểm nào là điểm nhiễu sẽ bị “đè” (mang giá trị false). Điều này được thực hiện bởi hàm keep\_or\_supress (sẽ được trình bày ở phần sau). Sau đó, ta duyệt lại ma trận một lần nữa để “đè” những điểm đã được đánh dấu.

## V. Một số hàm hỗ trợ:

Bên cạnh 4 hàm chính thực hiện 4 thuật toán nhận diện cạnh, nhóm có xây dựng một số hàm hỗ trợ cho các công đoạn thực hiện của 4 thuật toán để mã nguồn được gọn gàng hơn

- Hàm `int RGB_to_Gray(cv::Mat &src, cv::Mat &dest)`: thực hiện chuyển đổi một ảnh RGB thành ảnh Grayscale. Hàm nhận vào 2 tham số đầu vào là ảnh RGB src và ảnh mục tiêu dest cần được chuyển thành grayscale được cài đặt như sau:

```

int RGB_to_Gray(cv::Mat &src, cv::Mat &dest)
{
    if (src.data == NULL)
        return 0;

    int c = src.cols, r = src.rows;
    int src_channel = src.channels();
    int dest_channel = dest.channels();

    for (int i = 0; i < c; i++)
    {

```

```

    uchar* src_pRow = src.ptr<uchar>(i);
    uchar* dest_pRow = dest.ptr<uchar>(i);
    for (int j = 0; j < r; j++, src_pRow += src_channel, dest_pRow +=
dest_channel)
    {
        uchar B = src_pRow[0];
        uchar G = src_pRow[1];
        uchar R = src_pRow[2];
        uchar gray = (uchar)(0.3*R + 0.59*G + 0.11*B);
        dest_pRow[0] = gray;
    }
}
return 1;
}

```

Hàm có kiểu trả về là int: trả về 0 nếu thực hiện thất bại và trả về 1 nếu thực hiện thành công. Ý tưởng cơ bản là hàm sẽ duyệt qua tất cả các điểm ảnh của src, lấy 3 giá trị màu R, G, B ở mỗi điểm ảnh, tính toán giá trị grayscale theo công thức  $\text{Grayscale} = 0.3R + 0.59G + 0.11B$  rồi gán giá trị grayscale vào pixel tương ứng ở ảnh dest.

- Hàm `int Grad_Operation(cv::Mat &img, vector<vector<int>> filter, int x, int y)`: đây là hàm thực hiện tích chập mỗi điểm ảnh với một ma trận kernel kích thước 3 x 3. Hàm nhận vào 4 tham số: ảnh cần xử lý img, ma trận kernel filter và 2 tọa độ x, y của điểm ảnh.

```

int Grad_Operation(cv::Mat &img, vector<vector<int>> filter, int x, int y)
{
    return (int)(img.at<uchar>(x - 1, y - 1)) * filter[0][0]
        + (int)(img.at<uchar>(x - 1, y)) * filter[0][1]
        + (int)(img.at<uchar>(x - 1, y + 1)) * filter[0][2]
        + (int)(img.at<uchar>(x, y - 1)) * filter[1][0]
        + (int)(img.at<uchar>(x, y)) * filter[1][1]
        + (int)(img.at<uchar>(x, y + 1)) * filter[1][2]
        + (int)(img.at<uchar>(x + 1, y - 1)) * filter[2][0]
        + (int)(img.at<uchar>(x + 1, y)) * filter[2][1]
        + (int)(img.at<uchar>(x + 1, y + 1)) * filter[2][2];
}

```

Về cơ bản, hàm chỉ trả về giá trị tích chập sau khi tính tích chập theo công thức  $A * B = \sum_{i=1}^n \sum_{j=1}^n a_{ij} * b_{ij}$ , với A, B là 2 ma trận kích thước n x n. Ở đây, A là ma trận 3 x 3 tạo bởi điểm ảnh cần tính tích chập và 8 điểm xung quanh điểm ảnh đó; B là kernel 3 x 3 tương ứng.

- Hàm `void Gaussian_blur(cv::Mat &img)`: đây là hàm khử nhiễu bằng cách nhân ma trận ảnh với kernel Gauss kích thước 3 x 3  $\left(\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}\right)$ . Hàm có 1 tham số đầu vào là ảnh img cần xử lý và được cài đặt như sau:

```

void Gaussian_blur(cv::Mat &img)

```

```

{
    vector<vector<int>> gauss_filter = { { 1, 2, 1 }, // Gauss kernel
                                         { 2, 4, 2 },
                                         { 1, 2, 1 } };
    int c = img.cols, r = img.rows;
    for (int i = 1; i < c - 1; i++)
    {
        for (int j = 1; j < r - 1; j++)
            img.at<uchar>(i, j) = Grad_Operation(img, gauss_filter, i, j) / 16;
    }
}

```

Về cơ bản, hàm cho kernel Gauss chạy qua tất cả các điểm của hình ảnh (trừ các điểm trên 4 cạnh ngoài cùng) và thực hiện tính tích chập các điểm ảnh đó. Để tương thích với hàm Grad\_Operation đã cài đặt ở trên, thay vì lưu kernel Gauss dưới kiểu dữ liệu số thực thì nhóm lưu dưới kiểu số nguyên rồi sau đó thực hiện phép chia lấy phần nguyên cho 16.

- Hàm `void Non_maximum_supression(cv::Mat &img, int x, int y, int angle)`: dùng để hàm mỏng cạnh trong pha xử lý non-maximum supression của thuật toán Canny. Hàm nhận 4 tham số: ảnh img cần xử lý, tọa độ x, y của điểm ảnh cần được xử lý và tham số góc (hướng của cạnh mà điểm ảnh thuộc về), xử lý trên từng pixel và được cài đặt như sau:

```

void Non_maximum_supression(cv::Mat &img, int x, int y, int angle)
{
    // north - south edge -> check east - west direction
    if (angle == 0 || angle == 180)
    {
        int t1 = int(img.at<uchar>(x, y));
        int t2 = int(img.at<uchar>(x, y - 1));
        int t3 = int(img.at<uchar>(x, y + 1));
        if (!is_largest(t1, t2, t3))
            img.at<uchar>(x, y) = (uchar)(0);
    }

    // north-west - south-east edge -> check north-east - south-west direction
    if (angle == 45)
    {
        int t1 = int(img.at<uchar>(x, y));
        int t2 = int(img.at<uchar>(x + 1, y - 1));
        int t3 = int(img.at<uchar>(x - 1, y + 1));
        if (!is_largest(t1, t2, t3))
            img.at<uchar>(x, y) = (uchar)(0);
    }

    // east - west edge -> check north - south direction
}

```

```

if (angle == 90)
{
    int t1 = int(img.at<uchar>(x, y));
    int t2 = int(img.at<uchar>(x - 1, y));
    int t3 = int(img.at<uchar>(x + 1, y));
    if (!is_largest(t1, t2, t3))
        img.at<uchar>(x, y) = (uchar)(0);
}

// north-east - south-west edge -> check north-west - south-east direction
if (angle == 135)
{
    int t1 = int(img.at<uchar>(x, y));
    int t2 = int(img.at<uchar>(x - 1, y - 1));
    int t3 = int(img.at<uchar>(x + 1, y + 1));
    if (!is_largest(t1, t2, t3))
        img.at<uchar>(x, y) = (uchar)(0);
}
}

```

Về cơ bản, hàm sẽ kiểm tra xem điểm ảnh đang xét có thật sự là một phần của cạnh hay không, bằng cách kiểm tra xem điểm ảnh đó có phải là điểm ảnh có sự thay đổi màu mạnh nhất trên phương vuông góc với phương của cạnh (mà điểm ảnh đó đang nằm trên).

Hàm sẽ so sánh giá trị gradient của điểm ảnh đang xét với 2 điểm ảnh lân cận (theo phương vuông góc với cạnh và về 2 hướng ngược nhau) bằng hàm (sẽ được trình bày ở phía dưới). Nếu điểm ảnh đang xét không phải là điểm ảnh lớn nhất trong 3 điểm ảnh, nó sẽ bị xóa (được gán bằng 0). Ngược lại, điểm ảnh sẽ được giữ nguyên giá trị.

- Hàm `bool is_largest(int a, int b, int c)`: dùng để kiểm tra xem số nguyên a có là số lớn nhất trong 3 số nguyên a, b, c hay không. Hàm được cài đặt như sau:

```

bool is_largest(int a, int b, int c) //check if a is the largest of all 3
{
    int tmp = max(a, b);
    if (tmp == b)
        return false;
    else
    {
        tmp = max(a, c);
        if (tmp == c)
            return false;
        return true;
    }
}

```

- Hàm `vector<vector<bool>> keep_or_supress(cv::Mat &img, int upper_bound, int lower_bound)`: quyết định xem điểm ảnh nào sẽ được giữ lại và điểm ảnh nào sẽ bị xóa. Hàm được cài đặt như sau:

```
vector<vector<bool>> keep_or_supress(cv::Mat &img, int upper_bound, int
lower_bound)
{
    // true -> strong edge -> keep that pixel
    // false -> not and edge -> supress
    // Note: In this function, the weak edge will be categorized as noise or
strong edge
    vector<vector<bool>> res;
    int c = img.cols, r = img.rows;
    for (int i = 1; i < r - 1; i++)
    {
        vector<bool> tmp;
        for (int j = 1; j < c - 1; j++)
        {
            int v = (int)(img.at<uchar>(i, j));
            if (v > upper_bound)
                tmp.push_back(true);
            else
            {
                if (v < lower_bound)
                    tmp.push_back(false);
                else
                {
                    bool noise_or_edge = is_connect_to_strong_edge(img, i, j,
upper_bound);
                    if (noise_or_edge == true)
                        tmp.push_back(true);
                    else
                        tmp.push_back(false);
                }
            }
        }
        res.push_back(tmp);
    }
    return res;
}
```

Về cơ bản, hàm so sánh giá trị gradient của điểm ảnh với 2 ngưỡng chặn:

- Nếu điểm ảnh lớn hơn chặn trên `upper_bound` => giữ lại
- Nếu điểm ảnh bé hơn chặn dưới `lower_bound` => xóa
- Nếu điểm ảnh nằm giữa 2 ngưỡng chặn: kiểm tra xem điểm ảnh có nằm kề với cạnh mạnh (điểm ảnh lớn hơn `upper_bound`) hay không. Nếu có thì giữ, nếu không thì xóa.

Hàm `bool is_connect_to_strong_edge(cv::Mat &img, int x, int y, int upper_bound)`: kiểm tra xem một điểm ảnh có kết nối với các “cạnh mạnh” hay không

```
bool is_connect_to_strong_edge(cv::Mat &img, int x, int y, int upper_bound)
{
    int c = img.cols, r = img.rows;
    if (x == 0 || x == c - 1 || y == 0 || y == c - 1)
        return false;
    else
    {
        int count = 0; //count the number of adjacent strong edge pixel
        if ((int)(img.at<uchar>(x - 1, y - 1)) > upper_bound) count++;
        if ((int)(img.at<uchar>(x - 1, y)) > upper_bound) count++;
        if ((int)(img.at<uchar>(x - 1, y + 1)) > upper_bound) count++;
        if ((int)(img.at<uchar>(x, y - 1)) > upper_bound) count++;
        if ((int)(img.at<uchar>(x - 1, y + 1)) > upper_bound) count++;
        if ((int)(img.at<uchar>(x + 1, y - 1)) > upper_bound) count++;
        if ((int)(img.at<uchar>(x + 1, y)) > upper_bound) count++;
        if ((int)(img.at<uchar>(x + 1, y + 1)) > upper_bound) count++;
        if (count > 0) return true;
        return false;
    }
}
```

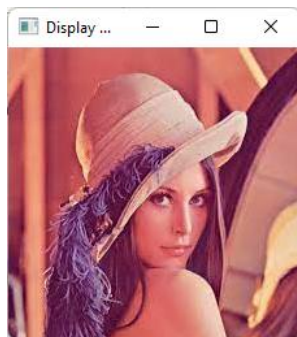
Hàm so sánh các điểm xung quanh điểm ảnh đang xét với ngưỡng chặn trên và đếm số điểm thỏa mãn yêu cầu “cạnh mạnh”

## B. Kết quả thực nghiệm:

Nhóm sử dụng 5 bức ảnh khác nhau với kích thước 225 x 225 pixel (do trong cách cài đặt, nhóm chỉ cho phép ảnh được xử lý dưới kích thước WINDOW\_AUTOSIZE là 225 x 225). Mỗi bức ảnh sẽ được thực hiện bởi cả 4 thuật toán, riêng với thuật toán Canny, 2 ngưỡng chặn là 50 và 150. Kết quả xuất ra bao gồm ảnh gốc, ảnh gradient theo phương Ox, ảnh gradient theo phương Oy và ảnh đầu ra (edge detection).

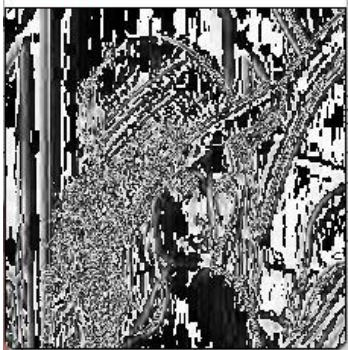
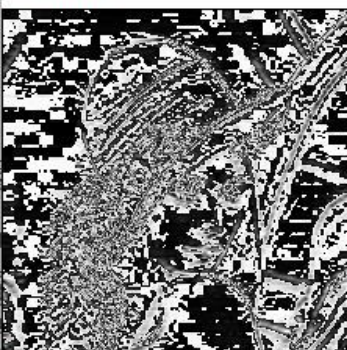


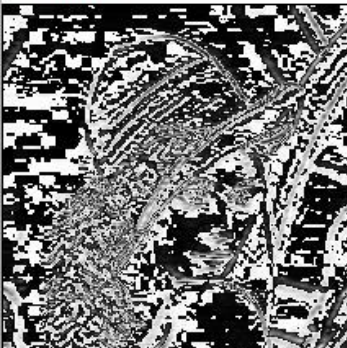



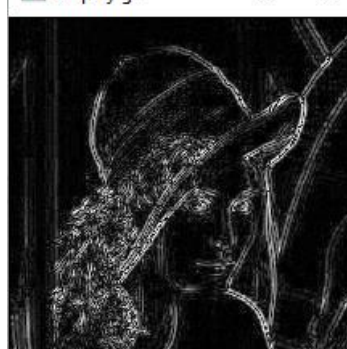
### Ảnh 1: Lena.


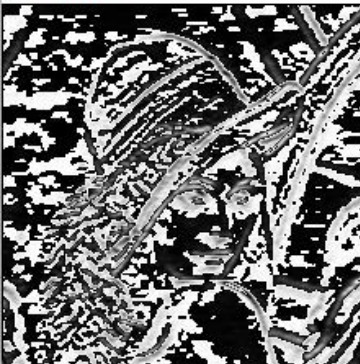

Ảnh gốc:





Kết quả thực thi:

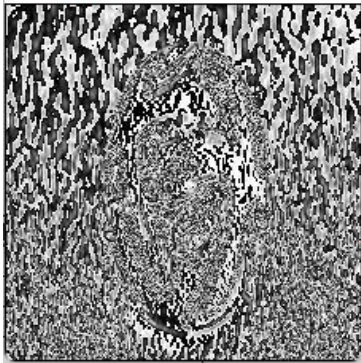
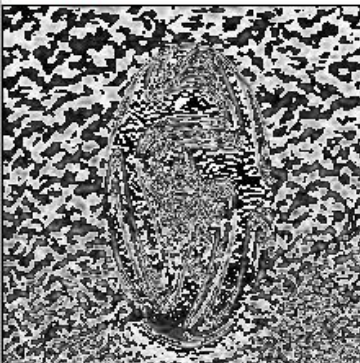

Thuật toán	Gradient theo x	Gradient theo y	Kết quả đầu ra
Sobel	 A window titled 'Display x...' showing the Sobel gradient in the x direction. The image highlights horizontal edges with white lines on a black background.	 A window titled 'Display y...' showing the Sobel gradient in the y direction. The image highlights vertical edges with white lines on a black background.	 A window titled 'Display g...' showing the combined Sobel gradient. The image shows both horizontal and vertical edges in white on a black background.
Prewitt	 A window titled 'Display x...' showing the Prewitt gradient in the x direction. The image highlights horizontal edges with white lines on a black background.	 A window titled 'Display y...' showing the Prewitt gradient in the y direction. The image highlights vertical edges with white lines on a black background.	 A window titled 'Display g...' showing the combined Prewitt gradient. The image shows both horizontal and vertical edges in white on a black background.
Laplace	 A window titled 'Display x...' showing the Laplace gradient in the x direction. The image highlights horizontal edges with white lines on a black background.	 A window titled 'Display y...' showing the Laplace gradient in the y direction. The image highlights vertical edges with white lines on a black background.	 A window titled 'Display g...' showing the combined Laplace gradient. The image shows both horizontal and vertical edges in white on a black background.

Canny			
-------	---	--	---

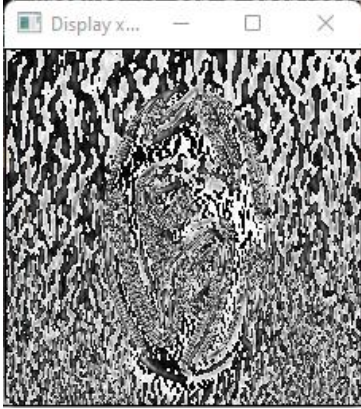

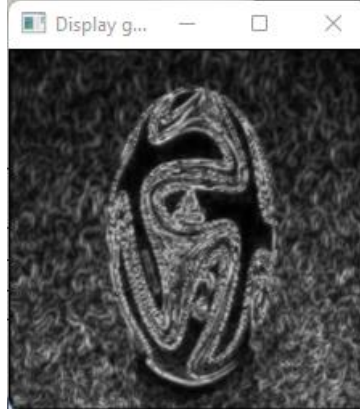
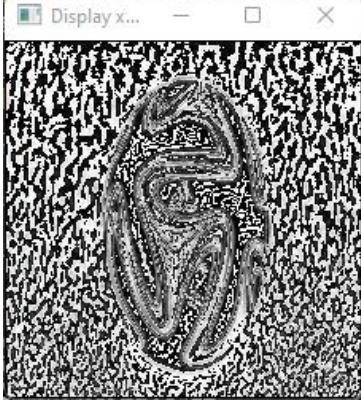

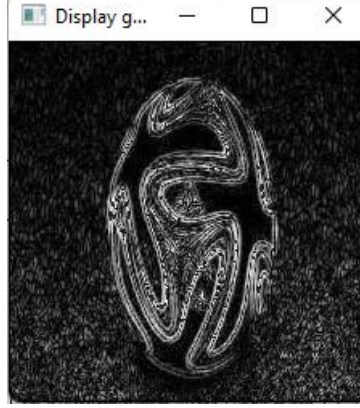

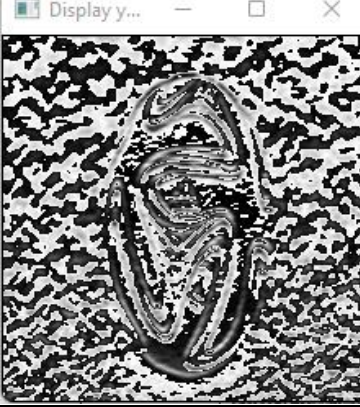

**Ảnh 2: Brazuca**  
Ảnh gốc:



Kết quả thực thi:

Thuật toán	Gradient theo x	Gradient theo y	Kết quả đầu ra
Sobel			



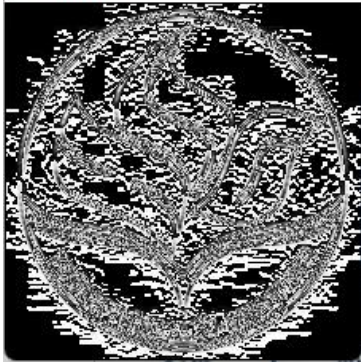
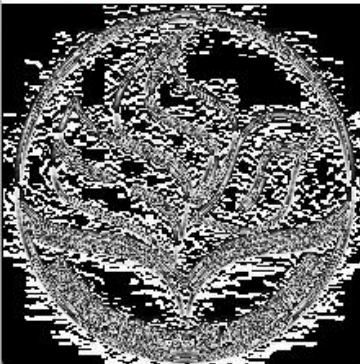

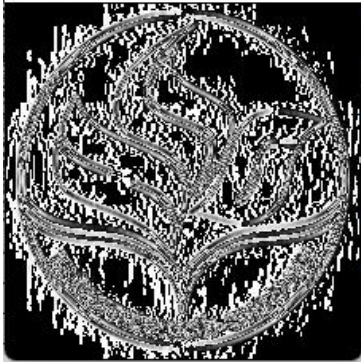
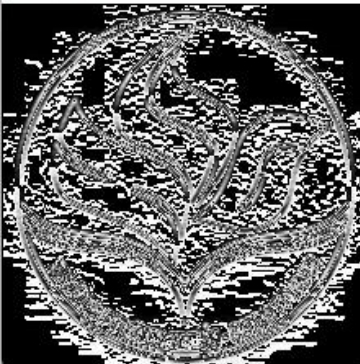

Prewitt	 A window titled 'Display x...' showing the x-axis edge detection result for the Prewitt operator. The image displays horizontal edges as bright lines against a dark background.	 A window titled 'Display y...' showing the y-axis edge detection result for the Prewitt operator. The image displays vertical edges as bright lines against a dark background.	 A window titled 'Display g...' showing the gradient magnitude edge detection result for the Prewitt operator. The image displays the combined edges from both x and y directions.
Laplace	 A window titled 'Display x...' showing the x-axis edge detection result for the Laplace operator. The image displays horizontal edges as bright lines against a dark background.	 A window titled 'Display y...' showing the y-axis edge detection result for the Laplace operator. The image displays vertical edges as bright lines against a dark background.	 A window titled 'Display g...' showing the gradient magnitude edge detection result for the Laplace operator. The image displays the combined edges from both x and y directions.
Canny	 A window titled 'Display x...' showing the x-axis edge detection result for the Canny operator. The image displays horizontal edges as bright lines against a dark background.	 A window titled 'Display y...' showing the y-axis edge detection result for the Canny operator. The image displays vertical edges as bright lines against a dark background.	 A window titled 'Display g...' showing the gradient magnitude edge detection result for the Canny operator. The image displays the combined edges from both x and y directions.

### Ảnh 3: Logo

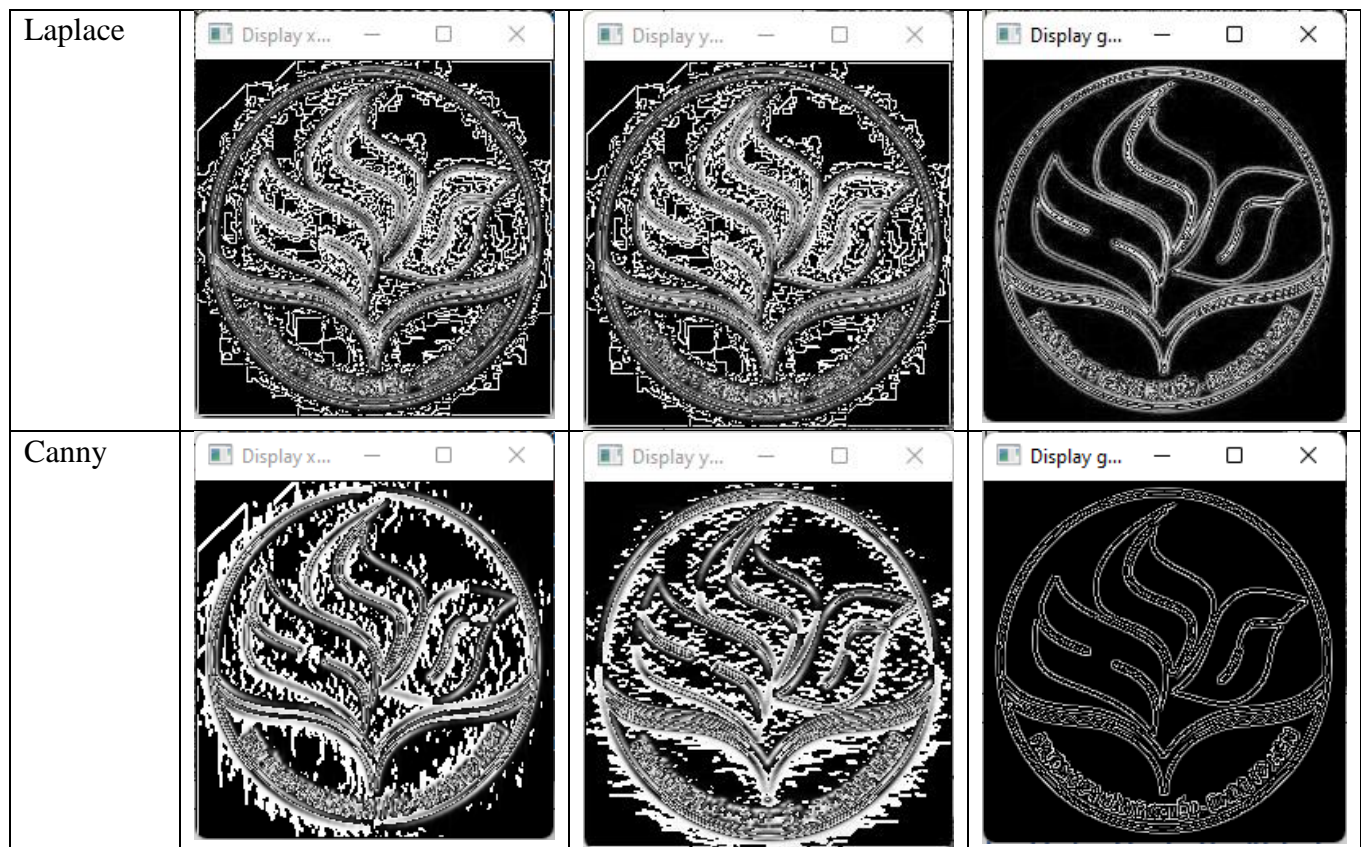
Ảnh gốc:



Kết quả thực thi:

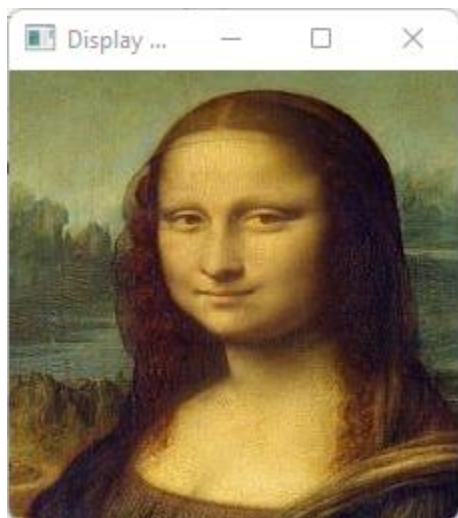
Thuật toán	Gradient theo x	Gradient theo y	Kết quả đầu ra
Sobel			
Prewitt			







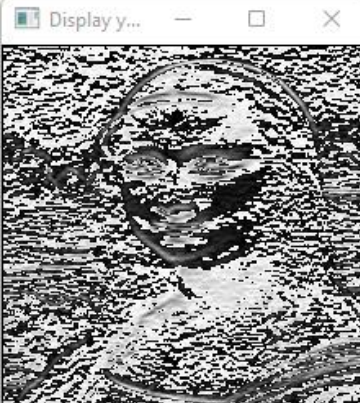



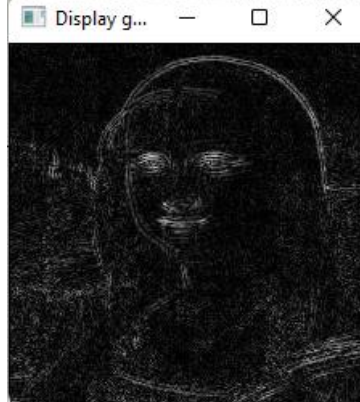


**Ảnh 4: Mona Lisa**

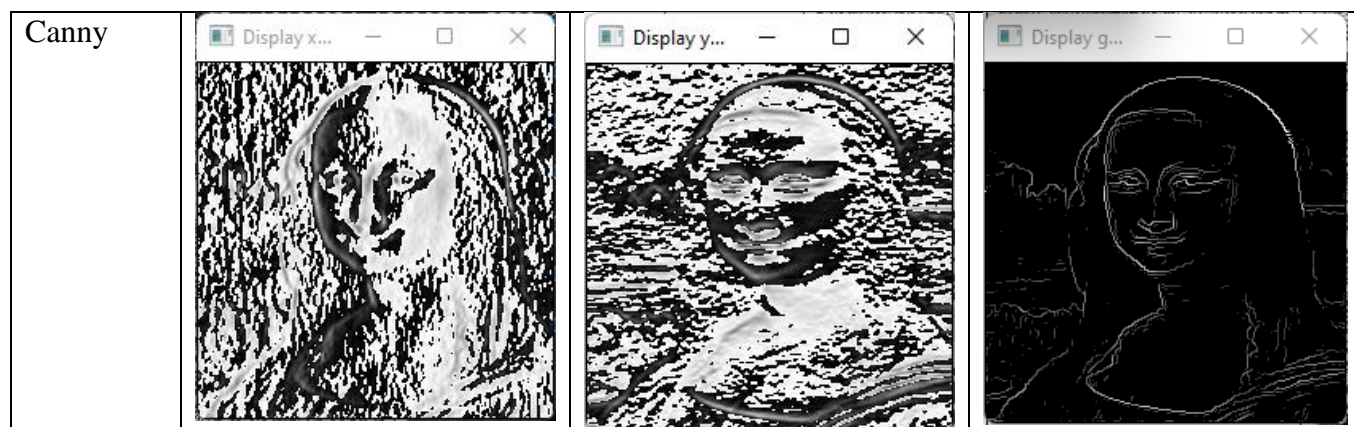
Ảnh gốc:



Kết quả thực thi:

Thuật toán	Gradient theo x	Gradient theo y	Kết quả đầu ra
Sobel			
Prewitt			
Laplace			



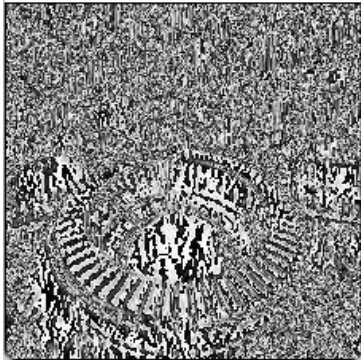
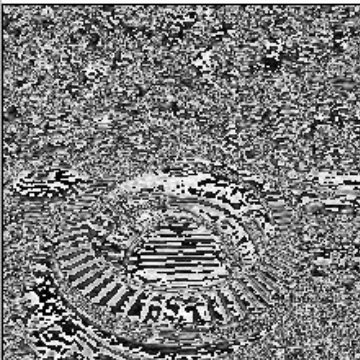
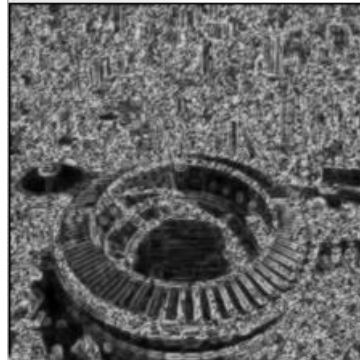


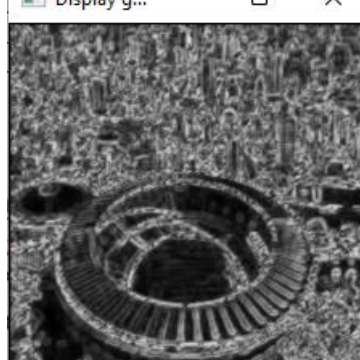

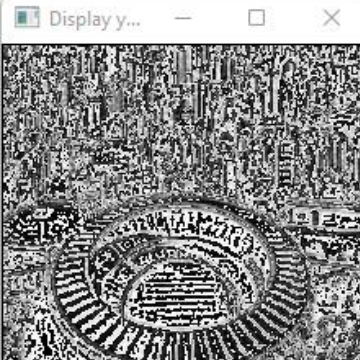



**Ảnh 5: Staium**

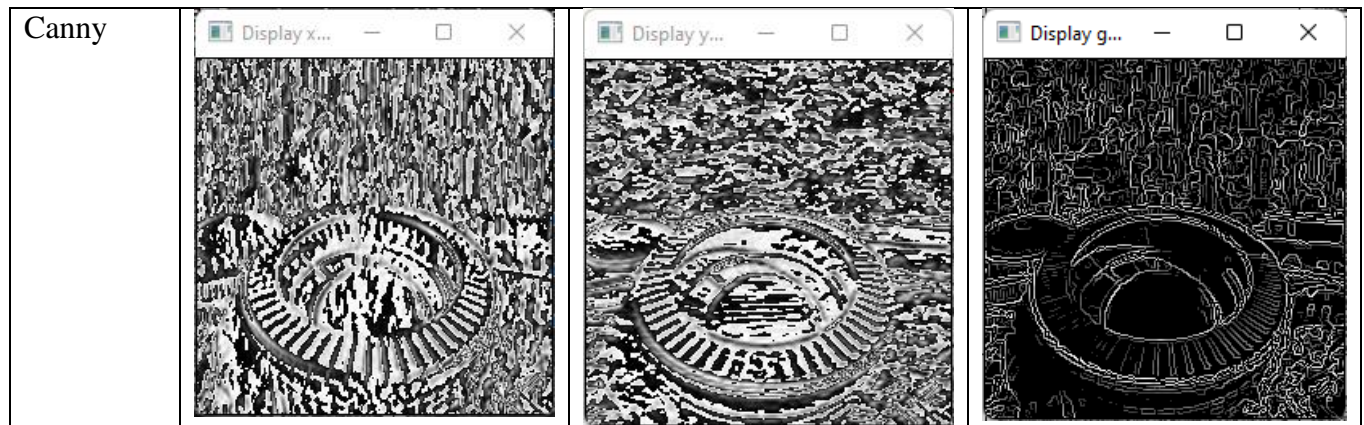
Ảnh gốc:



Kết quả thực thi:

Thuật toán	Gradient theo x	Gradient theo y	Kết quả đầu ra
Sobel	 A window titled 'Display x...' showing the Sobel gradient in the x-direction. The image highlights horizontal edges with a noisy, high-contrast black and white pattern.	 A window titled 'Display y...' showing the Sobel gradient in the y-direction. The image highlights vertical edges with a noisy, high-contrast black and white pattern.	 A window titled 'Display g...' showing the final Sobel edge detection result. The image shows a clear, smooth circular structure with radial lines, representing the combined magnitude of the gradients.
Prewitt	 A window titled 'Display x...' showing the Prewitt gradient in the x-direction. The image highlights horizontal edges with a noisy, high-contrast black and white pattern.	 A window titled 'Display y...' showing the Prewitt gradient in the y-direction. The image highlights vertical edges with a noisy, high-contrast black and white pattern.	 A window titled 'Display g...' showing the final Prewitt edge detection result. The image shows a clear, smooth circular structure with radial lines, representing the combined magnitude of the gradients.
Laplace	 A window titled 'Display x...' showing the Laplace gradient in the x-direction. The image highlights horizontal edges with a noisy, high-contrast black and white pattern.	 A window titled 'Display y...' showing the Laplace gradient in the y-direction. The image highlights vertical edges with a noisy, high-contrast black and white pattern.	 A window titled 'Display g...' showing the final Laplace edge detection result. The image shows a clear, smooth circular structure with radial lines, representing the combined magnitude of the gradients.





Giao diện console và hướng dẫn sử dụng:

```

C:\Windows\System32\cmd.exe
E:\19120034_19120241_BT02\Release>19120034_19120241_BT02 --help
Usage: 19120034_19120241_BT02 [params] image

    --c, --color (value:rgb gray)
        loại màu của ảnh đầu vào (default: color)
    --t, --type (value:sobel prewitt laplace canny)
        thuật toán dùng để xử lý
    -h, --help (value:true)
        in thông tin hướng dẫn

    image (value:/path/to/image)
        đường dẫn đến tập tin

E:\19120034_19120241_BT02\Release>

```

### C. So sánh, đánh giá các thuật toán:

Dựa trên kết quả thực nghiệm và các kết quả lý thuyết, nhóm có một số nhận xét về các thuật toán edge detection như sau:

Về cài đặt: Canny là thuật toán khó cài đặt nhất, do có nhiều pha xử lý. Ba thuật toán Sobel, Prewitt và Laplace dễ tiếp cận về mặt ý tưởng, khá dễ cài đặt và gần như khá tương đồng với nhau trong các pha xử lý.

Về hiệu quả:

- Sobel và Prewitt do thiếu các bước xử lý phía sau nên các cạnh nhận dạng được còn dày, còn nhiều cạnh thừa và chưa thật sự nhấn mạnh được các cạnh, kể cả khi đã có bước khử nhiễu sau khi nhận diện. Nếu bỏ qua bước làm mờ bằng kernel Gauss (khử nhiễu) thì kết quả thu

được còn chứa nhiều cạnh “giả” (xuất hiện do nhiễu). Còn nếu thực hiện làm mờ để khử nhiễu thì ảnh đầu ra lại không còn được rõ nét.

- Kết quả đầu ra của Laplace nhìn chung khá hơn Prewitt và Sobel khi ở ảnh đầu ra, các cạnh đã được định hình rõ nét hơn. Tuy nhiên, thuật toán này cực kỳ nhạy cảm với nhiễu. Nếu ảnh có độ nhiễu lớn, thuật toán sẽ không nhận diện được cạnh. Việc làm mờ để khử nhiễu trước khi nhận diện làm tăng hiệu quả nhận diện của Laplace, nhưng cũng ảnh hưởng đến chất lượng ảnh đầu ra (bị mờ đi)
- Canny là thuật toán cho ra hiệu quả cao nhất, khi nhận diện được cụ thể và rõ ràng các đường nét chính của ảnh. Tuy nhiên, việc lựa chọn hai ngưỡng chặn thích hợp là vô cùng quan trọng. Khoảng của ngưỡng chặn quá lớn sẽ xóa đi gần hết các nét của ảnh đầu ra. Thêm vào đó, pha xử lý cuối cùng (double threshold) cần được thực hiện cẩn thận để tránh việc nhận định sai các điểm ảnh “weak edge”



## D. Nguồn tham khảo:

Slide bài giảng lý thuyết

<https://opencv.org/>

[https://en.wikipedia.org/wiki/Gaussian\\_filter](https://en.wikipedia.org/wiki/Gaussian_filter)

<https://www.youtube.com/watch?v=hUC1uoigH6s>

<https://www.youtube.com/watch?v=IOEBsQodtEQ>

[https://en.wikipedia.org/wiki/Canny\\_edge\\_detector](https://en.wikipedia.org/wiki/Canny_edge_detector)