

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
KHOA CÔNG NGHỆ THÔNG TIN  
BỘ MÔN THI GIÁC MÁY TÍNH VÀ ĐIỀU KHIỂN  
HỌC THÔNG MINH



---

XỬ LÝ ẢNH SỐ VÀ VIDEO SỐ  
BÁO CÁO THỰC HÀNH GIỮA KỲ  
PHÉP BIẾN ĐỔI HÌNH HỌC

---

## Phần 1. Thông tin của nhóm

Mã số sinh viên	Họ và tên	Chức vụ
18120018	Nguyễn Hoàng Đức	Nhóm trưởng
18120061	Lê Nhựt Nam	Thành viên
18120062	Nguyễn Hoàng Nam	Thành viên
18120533	Dương Đoàn Bảo Sơn	Thành viên
18120543	Trần Đại Tài	Thành viên

## Phần 2. Phân công

Phân phân công cho thực hành lab 02 – Phép biến đổi hình học

Họ tên	Nhiệm vụ	Số điểm
Nguyễn Hoàng Đức	Đổi xứng ảnh qua trực đứng (Oy) hay trực ngang (Ox), Thay đổi kích thước ảnh	4
Lê Nhựt Nam	Cài đặt hàm Transform trong lớp GeometricTransformer	3
Nguyễn Hoàng Nam	Cài đặt lớp nội suy song tuyến tính, Phóng to, thu nhỏ ảnh	4
Dương Đoàn Bảo Sơn	Xoay ảnh quanh tâm (không bảo toàn nội dung ảnh), Xoay ảnh quanh tâm (không bảo toàn nội dung ảnh)	4
Trần Đại Tài	Cài đặt lớp AffineTransform, Cài đặt lớp nội suy láng giềng gần nhất	3

Trình bày báo cáo: Lê Nhựt Nam

## Phần 3. Nội dung thực hiện

A. Nội dung của bài thực hành 02 – Phép biến đổi hình học

Câu	Yêu cầu	Tên câu lệnh	Tham số câu lệnh	Điểm	Mức độ hoàn thành

1	Cài đặt lớp AffineTransform			2	100%
2	Cài đặt lớp nội suy lóng gièng gần n hất			1	100%
3	Cài đặt lớp nội suy song tuyến tính			2	100%
4	Cài đặt hàm Transform trong lớp GeometricTransformer			3	100%
5	Phóng to, thu nhỏ ảnh	--zoom	Hệ số zoom	2	100%
6	Thay đổi kích thước ảnh	--resize	Kích cỡ mới	2	100%
7	Xoay ảnh quanh tâm (bảo toàn nội dung ảnh)	--rotate	Góc xoay	2	100%
8	Xoay ảnh quanh tâm (không bảo toàn nội dung ảnh)	--rotateN	Góc xoay	2	100%
9	Đổi xứng ảnh qua trực đứng (Oy) hay trục ngang (Ox)	--flip	Oy hay Ox	2	100%
Tổng				18	100%

## B. Giải quyết vấn đề

## Câu 1: Cài đặt lớp AffineTransform

```
/*
Lớp biến diển pháp biến đổi affine
*/
class AffineTransform
{
    cv::Mat _matrixTransform; //ma trận 3x3 biến diển phép biến đổi affine

public:
    // Phương thức getter: Lấy matrix transform
    cv::Mat getMatrixTransform() {
        return (this->_matrixTransform);
    }

    // Phương thức setter: Đưa vào matrix transform
    void setMatrixTransform(cv::Mat matrixTransform) {
        this->_matrixTransform = matrixTransform;
    }

    // Xây dựng matrix transform cho phép tịnh tiến theo vector (dx,dy)
    void Translate(float dx, float dy) {
        float TranslateMatrix[3][3] = { {1, 0, dx}, {0, 1, dy}, {0, 0, 1} };
        cv::Mat translateMat = cv::Mat(3, 3, CV_32FC1, TranslateMatrix);
        this->_matrixTransform = translateMat * this->_matrixTransform;
    }

    // Xây dựng matrix transform cho phép xoay 1 góc angle
    void Rotate(float angle){
        float sin_alpha = sin(angle * M_PI / 180);
        float cos_alpha = cos(angle * M_PI / 180);

        float RotateMatrix[3][3] = { {cos_alpha, (-sin_alpha), 0}, {sin_alpha, cos_alpha, 0}, {0, 0, 1} };
    };
    cv::Mat rotateMat = cv::Mat(3, 3, CV_32FC1, RotateMatrix);
    this->_matrixTransform = rotateMat * this->_matrixTransform;
}

    // Xây dựng matrix transform cho phép tỉ lệ theo hệ số
    void Scale(float sx, float sy) {
        float ScaleMatrix[3][3] = { {sx, 0, 0}, {0, sy, 0}, {0, 0, 1} };
        cv::Mat scaleMat = cv::Mat(3, 3, CV_32FC1, ScaleMatrix);
        this->_matrixTransform = scaleMat * this->_matrixTransform;
    }

    // Transform 1 điểm (x,y) theo matrix transform đã có
    void TransformPoint(float& x, float& y) {
        float oldPointMatrix[] = { x, y, 1 };
        cv::Mat newPoint = this->_matrixTransform * cv::Mat(3, 3, CV_32FC1, oldPointMatrix);
        x = newPoint.ptr<float>(0)[0];
        y = newPoint.ptr<float>(0)[1];
    }

    // Phương thức khởi tạo - Constructor
    AffineTransform() {
        float matrix[3][3] = { {1, 0, 0}
                            ,{0, 1, 0}
                            ,{0, 0, 1} };
        this->_matrixTransform = cv::Mat(3, 3, CV_32FC1, matrix).clone();
    }

    // Phương thức hủy - Destructor
    ~AffineTransform() {
        this->_matrixTransform.release();
    }
};
```

## Câu 2: Cài đặt lớp nội suy láng giềng gần nhất

```
● ● ●
class NearestNeighborInterpolate : public PixelInterpolate
{
public:
    uchar Interpolate(float tx, float ty, uchar* pSrc, int srcWidthStep, int nChannels){
        // Tính l = round(x)
        int l = (int)round(tx);

        // Tính k = round(y)
        int k = (int)round(ty);

        // f'(x', y') = f(l, k) = f(round(x), round(y))
        return cv::saturate_cast<uchar>((pSrc + (long long)srcWidthStep * l + (long long)nChannels
* k)[0]);
    }
    NearestNeighborInterpolate();
    ~NearestNeighborInterpolate();
};

};
```

## Câu 3: Cài đặt lớp nội suy song tuyến tính

```
● ● ●
class BilinearInterpolate : public PixelInterpolate
{
public:
    uchar Interpolate(float tx, float ty, uchar* pSrc, int srcWidthStep, int nChannels){
        // f'(x', y') = (1 - a)(1 - b)f(l, k) + a(1 - b)f(l+1, k) + b(1 - a)f(l, k + 1) + abf(l + 1, k +
1)

        // l = round(x)
        int l = (int)(round(tx));

        // k = round(y)
        int k = (int)(round(ty));

        // a = x - l
        float a = tx - l;

        // b = y - k
        float b = ty - k;

        // Tính f(l, k)
        int flk = (pSrc + (long long)srcWidthStep * l + (long long)nChannels * k)[0];

        // Tính f(l + 1, k)
        int fl1k = (pSrc + (long long)srcWidthStep * ((long long)l + 1) + (long long)nChannels * k)[0];

        // Tính f(l, k + 1)
        int flk1 = (pSrc + (long long)srcWidthStep * l + (long long)nChannels * ((long long)k + 1))[0];

        // Tính f(l + 1, k + 1)
        int fl1k1 = (pSrc + (long long)srcWidthStep * ((long long)l + 1) + (long long)nChannels *
((long long)k + 1))[0];

        return cv::saturate_cast<uchar>(round((1 - a) * (1 - b) * flk + a * (1 - b) * fl1k + (1 - a) *
b * flk1 + a * b * fl1k1));
    }
    BilinearInterpolate();
    ~BilinearInterpolate();
};

};
```

#### Câu 4: Cài đặt hàm Transform trong lớp GeometricTransformer

Hàm biến đổi ảnh theo 1 phép biến đổi affine đã có

```
int GeometricTransformer::Transform(
    const cv::Mat & beforeImage,
    cv::Mat & afterImage,
    AffineTransform * transformer,
    PixelInterpolate * interpolator) {
    // Kiểm tra ảnh đầu vào
    if (!beforeImage.data) {
        // Phát hiện lỗi: ảnh input ko tồn tại
        std::cout << "[EXCEPTION] Error with input image.\n";
        return 0; // Trả về 0
    }
    // Chiều rộng của ảnh source
    int widthBeforeImage = beforeImage.cols;
    // Chiều cao của ảnh source
    int heightBeforeImage = beforeImage.rows;
    // Số channels của ảnh source
    int sourceChannels = beforeImage.channels();
    // Width step của ảnh source
    size_t sourceWidthStep = beforeImage.step[0];
    // Lấy ma trận affine
    cv::Mat matrixTransform = transformer->getMatrixTransform();
    float B[] =
    {
        0, 0, 1.0,
    };
    cv::Mat P = cv::Mat(3, 1, CV_32FC1, B);
    // Con trỏ ảnh gốc
    uchar* pSrc = beforeImage.data;
    // Chiều cao của ảnh destination
    int heightAfterImage = afterImage.rows;
    // Chiều rộng của ảnh destination
    int widthAfterImage = afterImage.cols;
    for (int i = 0; i < heightAfterImage; i++)
    {
        uchar* pData = afterImage.ptr<uchar>(i);
        for (int j = 0; j < widthAfterImage; j++)
        {
            // Đặt Px = x, Py = y với x, y là tọa độ đối với tâm (0,0) là gốc trên cùng bên trái
            P.ptr<float>(0)[0] = i * 1.0;
            P.ptr<float>(1)[0] = j * 1.0;
            // Tìm tọa độ thực trên ảnh gốc
            cv::Mat srcP = matrixTransform * P;
            // tx, ty là index thực của điểm ảnh trên ma trận ảnh gốc
            float tx = srcP.ptr<float>(0)[0];
            float ty = srcP.ptr<float>(1)[0];
            // Chỉ xét tx, ty nằm trong ảnh gốc
            if (tx >= 0 && ty >= 0 && tx < heightBeforeImage && ty < widthBeforeImage)
            {
                for (int c = 0; c < sourceChannels; c++) {
                    // Áp dụng Interpolate cho từng channel
                    pData[j * sourceChannels + c] = interpolator->Interpolate(tx, ty, pSrc + c,
sourceWidthStep, sourceChannels);
                }
            }
        }
    }
    // Giải phóng ma trận P
    P.release();
    // Giải phóng ma trận biến đổi
    matrixTransform.release();
    return 1;
}
```

## Hàm xoay bảo toàn nội dung ảnh theo góc xoay cho trước

```
int GeometricTransformer::RotateKeepImage(
    const cv::Mat & srcImage, cv::Mat & dstImage, float angle, PixelInterpolate * interpolator) {
    // Kiểm tra ảnh đầu vào
    if (!srcImage.data) {
        // Phát hiện lỗi: ảnh input ko tồn tại
        std::cout << "[EXCEPTION] Error with input image.\n";
        return 0; // Trả về 0
    }
    // Chiều rộng của ảnh source
    int widthSourceImage = srcImage.cols;
    // Chiều cao của ảnh source
    int heightSourceImage = srcImage.rows;
    // Số channels của ảnh source
    int sourceChannels = srcImage.channels();
    // Width step của ảnh source
    size_t sourceWidthStep = srcImage.step[0];
    // Tính chiều dài của ảnh đích
    int dstWidth = (int)(widthSourceImage * cos(angle * M_PI / 180) + heightSourceImage * sin(angle * M_PI / 180));
    // Tính chiều rộng của ảnh đích
    int dstHeight = (int)(widthSourceImage * sin(angle * M_PI / 180) + heightSourceImage * cos(angle * M_PI / 180));
    // Khởi tạo affine transform
    AffineTransform affineTransform;
    // Tính tiền điểm về góc tọa độ
    affineTransform.Translate(dstHeight / 2 - heightSourceImage / 2, dstWidth / 2 - widthSourceImage / 2);
    affineTransform.Translate(-dstHeight / 2, -dstWidth / 2);
    // Quay ảnh một góc angle
    affineTransform.Rotate(angle);
    // Tính tiền ảnh về chính giữa
    affineTransform.Translate(dstHeight / 2, dstWidth / 2);
    // Khởi tạo ma trận nghịch đảo
    cv::Mat inverseMatrix = affineTransform.getMatrixTransform().inv();
    // Set ma trận nghịch đảo vào affine transform
    affineTransform.setMatrixTransform(inverseMatrix);
    if (sourceChannels == 1) {
        dstImage = cv::Mat::zeros(dstHeight, dstWidth, CV_8UC1);
    }
    else if (sourceChannels == 3) {
        dstImage = cv::Mat::zeros(dstHeight, dstWidth, CV_8UC3);
    }
    else {
        std::cout << "[EXCEPTION]: Too many channels or unsupported this number of channels.\n";
        return 0;
    }
    // Gọi hàm Transform
    GeometricTransformer::Transform(srcImage, dstImage, &affineTransform, interpolator);
    // Giải phóng ma trận nghịch đảo
    inverseMatrix.release();
    // Giải phóng Affine Transform
    affineTransform.~AffineTransform();
    return 1; // Trả về 1
}
```

## Hàm xoay không bảo toàn nội dung ảnh theo góc xoay cho trước



```
int GeometricTransformer::RotateUnkeepImage(
    const cv::Mat & srcImage,
    cv::Mat & dstImage,
    float angle,
    PixelInterpolate * interpolator) {
    // Kiểm tra ảnh đầu vào
    if (!srcImage.data) {
        // Phát hiện lỗi: ảnh input ko tồn tại
        std::cout << "[EXCEPTION] Error with input image.\n";
        return 0; // Trả về 0
    }
    // Chiều rộng của ảnh source
    int widthSourceImage = srcImage.cols;
    // Chiều cao của ảnh source
    int heightSourceImage = srcImage.rows;
    // Số channels của ảnh source
    int sourceChannels = srcImage.channels();
    // Width step của ảnh source
    size_t sourceWidthStep = srcImage.step[0];
    // Khởi tạo affine transform
    AffineTransform affineTransform;
    // Tịnh tiến ảnh về góc tọa độ
    affineTransform.Translate(-heightSourceImage / 2, -widthSourceImage / 2);
    // Quay một góc angle
    affineTransform.Rotate(angle);
    // Tịnh tiến ảnh trở về chính giữa
    affineTransform.Translate(heightSourceImage / 2, widthSourceImage / 2);
    // Khởi tạo ma trận nghịch đảo
    cv::Mat inverseMatrix = affineTransform.getMatrixTransform().inv();
    // Set ma trận nghịch đảo vào affine transform
    affineTransform.setMatrixTransform(inverseMatrix);
    if (sourceChannels == 1) {
        dstImage = cv::Mat::zeros(heightSourceImage, widthSourceImage, CV_8UC1);
    }
    else if (sourceChannels == 3) {
        dstImage = cv::Mat::zeros(heightSourceImage, widthSourceImage, CV_8UC3);
    }
    else {
        std::cout << "[EXCEPTION]: Too many channels or unsupported this number of channels.\n";
        return 0;
    }
    // Gọi hàm Transform
    GeometricTransformer::Transform(srcImage, dstImage, &affineTransform, interpolator);
    // Giải phóng ma trận nghịch đảo
    inverseMatrix.release();
    // Giải phóng Affine Transform
    affineTransform.~AffineTransform();
    return 1; // Trả về 1
}
```

## Hàm phóng to, thu nhỏ ảnh theo tỉ lệ cho trước

```
int GeometricTransformer::Scale(
    const cv::Mat & srcImage,
    cv::Mat & dstImage,
    float sx, float sy,
    PixelInterpolate * interpolator) {
    // Kiểm tra ảnh đầu vào
    if (!srcImage.data) {
        // Phát hiện lỗi: ảnh input ko tồn tại
        std::cout << "[EXCEPTION] Error with input image.\n";
        return 0; // Trả về 0
    }
    // Chiều rộng của ảnh source
    int widthBeforeImage = srcImage.cols;
    // Chiều cao của ảnh source
    int heightBeforeImage = srcImage.rows;
    // Số channels của ảnh source
    int sourceChannels = srcImage.channels();
    // Width step của ảnh source
    size_t sourceWidthStep = srcImage.step[0];
    // Khởi tạo Affine Transform
    AffineTransform affineTransform;
    // Tịnh tiến ảnh về gốc tọa độ
    affineTransform.Translate(-heightBeforeImage / 2, -widthBeforeImage / 2);
    // Scale theo tỉ lệ
    affineTransform.Scale(sx, sy);
    // Tịnh tiến ảnh trở về chính giữa
    affineTransform.Translate(heightBeforeImage / 2, widthBeforeImage / 2);
    // Khởi tạo ma trận nghịch đảo
    cv::Mat inverseMatrix = affineTransform.getMatrixTransform().inv();
    // Set ma trận nghịch đảo vào affine transform
    affineTransform.setMatrixTransform(inverseMatrix);
    // Khởi tạo ảnh kết quả
    if (sourceChannels == 1) {
        dstImage = cv::Mat::zeros(heightBeforeImage, widthBeforeImage, CV_8UC1);
    }
    else if (sourceChannels == 3) {
        dstImage = cv::Mat::zeros(heightBeforeImage, widthBeforeImage, CV_8UC3);
    }
    else {
        std::cout << "[EXCEPTION]: Too many channels or unsupported this number of channels.\n";
        return 0;
    }
    // Gọi hàm Transform
    GeometricTransformer::Transform(srcImage, dstImage, &affineTransform, interpolator);
    // Giải phóng ma trận nghịch đảo
    inverseMatrix.release();
    // Giải phóng Affine Transform
    affineTransform.~AffineTransform();
    return 1; // Nếu biến đổi thành công
}
```

## Hàm thay đổi kích thước ảnh

```
● ● ●

int GeometricTransformer::Resize(
    const cv::Mat & srcImage,
    cv::Mat & dstImage,
    int newWidth, int newHeight,
    PixelInterpolate * interpolator) {
    // Kiểm tra ảnh đầu vào
    if (!srcImage.data) {
        // Phát hiện lỗi: ảnh input ko tồn tại
        std::cout << "[EXCEPTION] Error with input image.\n";
        return 0; // Trả về 0
    }
    // Chiều rộng của ảnh source
    int widthBeforeImage = srcImage.cols;
    // Chiều cao của ảnh source
    int heightBeforeImage = srcImage.rows;
    // Số channels của ảnh source
    int sourceChannels = srcImage.channels();
    // Width step của ảnh source
    size_t sourceWidthStep = srcImage.step[0];
    // Khởi tạo ảnh kết quả
    if (sourceChannels == 1) { // Nếu là ảnh grayscale - 1 kênh
        dstImage = cv::Mat::zeros(newHeight, newWidth, CV_8UC1);
    }
    else if (sourceChannels == 3) { // Nếu là ảnh màu - 3 kênh
        dstImage = cv::Mat::zeros(newHeight, newWidth, CV_8UC3);
    }
    else { // Khác
        std::cout << "[EXCEPTION]: Too many channels or unsupported this number of channels.\n";
        return 0;
    }
    // Tính toán hệ số a
    float a = 1.0 * heightBeforeImage / newHeight;
    // Tính toán hệ số b
    float b = 1.0 * widthBeforeImage / newWidth;
    // Khởi tạo affine transform
    AffineTransform affineTransform;
    // Khởi tạo ma trận resize
    float matrix[3][3] = { {a, 0, 0}
                           ,{0, b, 0}
                           ,{0, 0, 1} };
    // Đưa ma trận resize vào affine transform
    affineTransform.setMatrixTransform(cv::Mat(3, 3, CV_32FC1, matrix).clone());
    // Gọi hàm Transform
    GeometricTransformer::Transform(srcImage, dstImage, &affineTransform, interpolator);
    // Giải phóng affine
    affineTransform.~AffineTransform();
    return 1;
}
```

## Hàm lấy đối xứng ảnh

```

int GeometricTransformer::Flip(
    const cv::Mat & srcImage,
    cv::Mat & dstImage,
    bool direction,
    PixelInterpolate * interpolator) {
    // Kiểm tra ảnh đầu vào
    if (!srcImage.data) {
        // Phát hiện lỗi: ảnh input ko tồn tại
        std::cout << "[EXCEPTION] Error with input image.\n";
        return 0; // Trả về 0
    }
    // Chiều rộng của ảnh source
    int widthBeforeImage = srcImage.cols;
    // Chiều cao của ảnh source
    int heightBeforeImage = srcImage.rows;
    // Số channels của ảnh source
    int sourceChannels = srcImage.channels();
    size_t sourceWidthStep = srcImage.step[0];
    // Nếu là lật dọc xứng theo trục ngang Ox
    if (direction == 1) {
        // Khởi tạo Affine Transform
        AffineTransform affineTransform;
        // Khởi tạo ma trận cho affine: Ma trận cho phép flip theo trục Ox
        float matrix[3][3] = { {-1, 0, heightBeforeImage - 1},
                               {0, 1, 0},
                               {0, 0, 1} };
        // Đưa ma trận vừa khởi tạo vào affine
        affineTransform.setMatrixTransform(cv::Mat(3, 3, CV_32FC1, matrix).clone());
        // Gọi hàm Transform
        GeometricTransformer::Transform(srcImage, dstImage, &affineTransform, interpolator);
        affineTransform.~AffineTransform();
        return 1; // Trả về 1
    }
    // Nếu là lật dọc xứng theo trục đứng Oy
    else if (direction == 0) {
        // Khởi tạo Affine Transform
        AffineTransform affineTransform;
        // Khởi tạo ma trận cho affine: Ma trận cho phép flip theo trục Oy
        float matrix[3][3] = { {1, 0, 0},
                               {0, -1, widthBeforeImage - 1},
                               {0, 0, 1} };
        // Đưa ma trận vừa khởi tạo vào affine
        affineTransform.setMatrixTransform(cv::Mat(3, 3, CV_32FC1, matrix).clone());
        // Gọi hàm Transform
        GeometricTransformer::Transform(srcImage, dstImage, &affineTransform, interpolator);
        // Giải phóng affine
        affineTransform.~AffineTransform();
        return 1; // Trả về 1
    }
    else {
        return 0;
    }
}

```

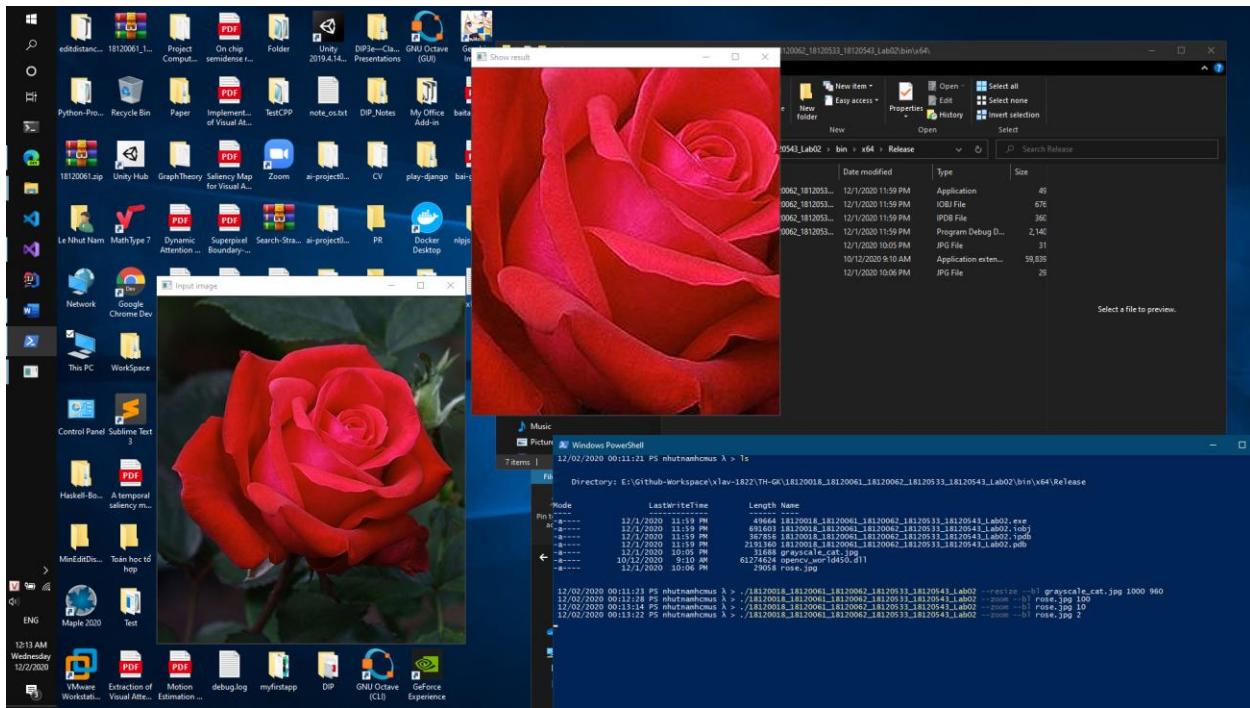
### C. Demo/ Kiểm thử các yêu cầu

#### 1. Câu 5: Phóng to, thu nhỏ ảnh

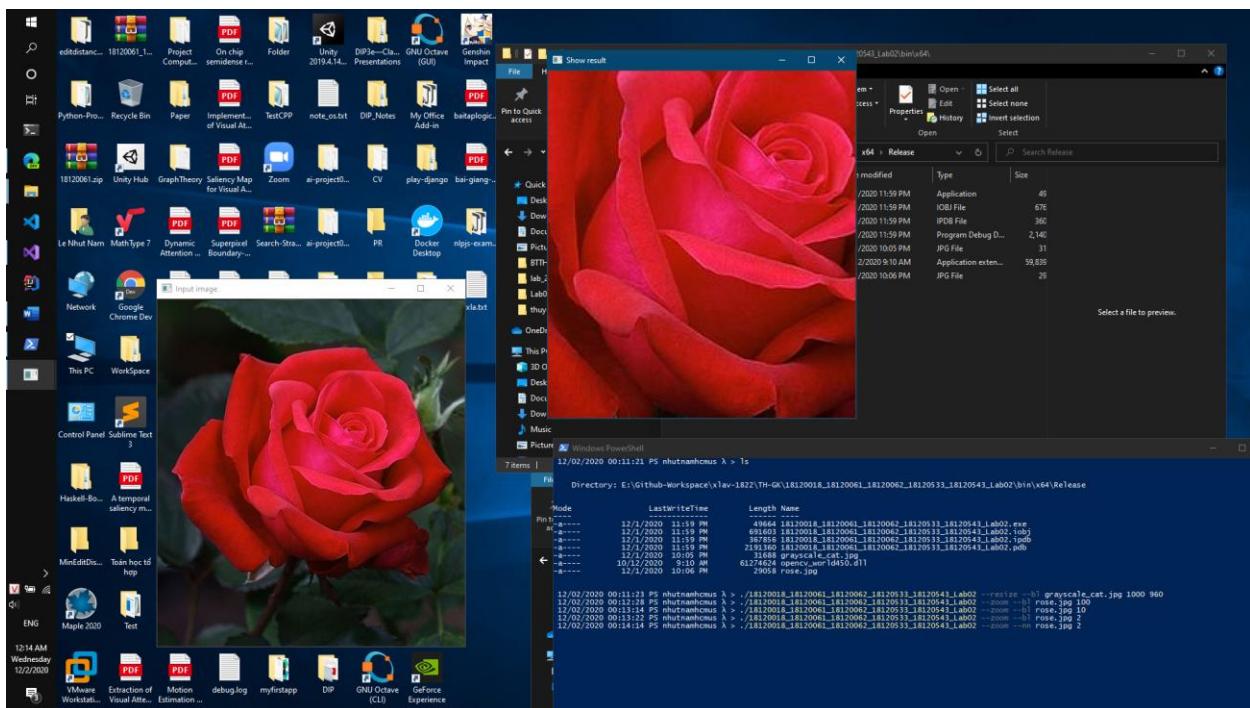
Phóng to:

Ảnh màu

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --zoom --bl rose.jpg 2

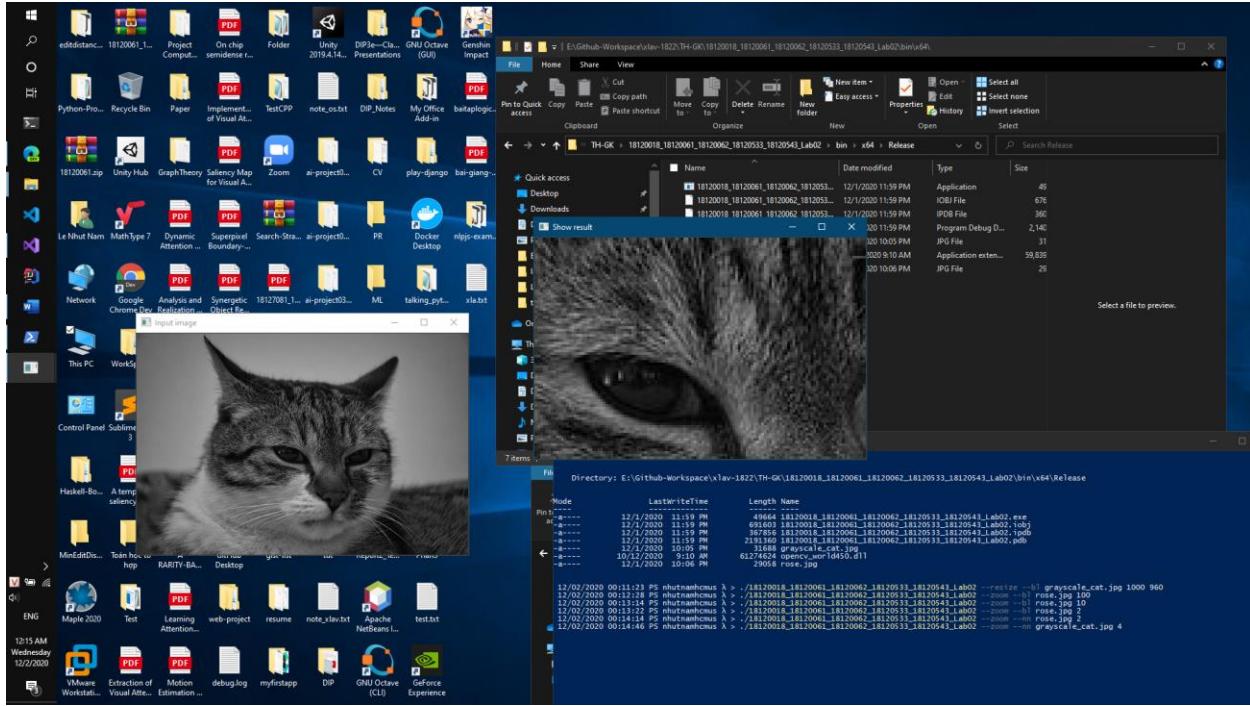


./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --zoom --nn rose.jpg 2

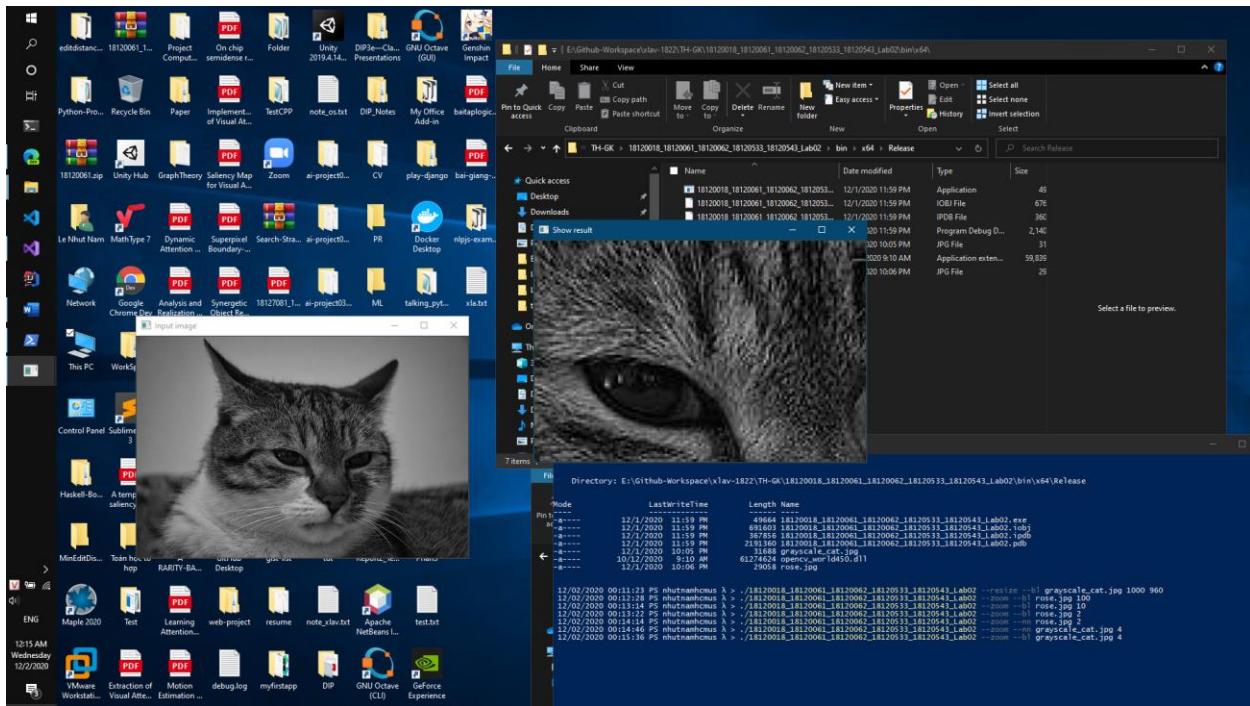


Ảnh xám

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --zoom --nn  
grayscale\_cat.jpg 4



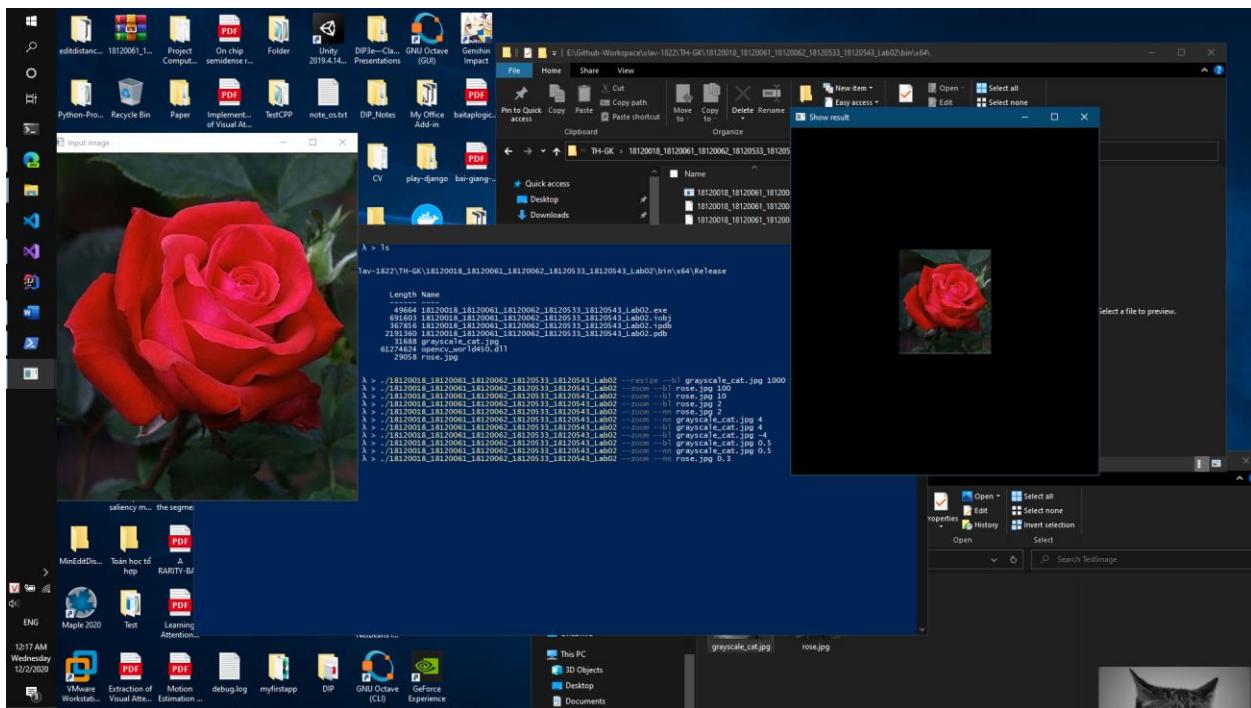
./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --zoom --bl  
grayscale\_cat.jpg 4



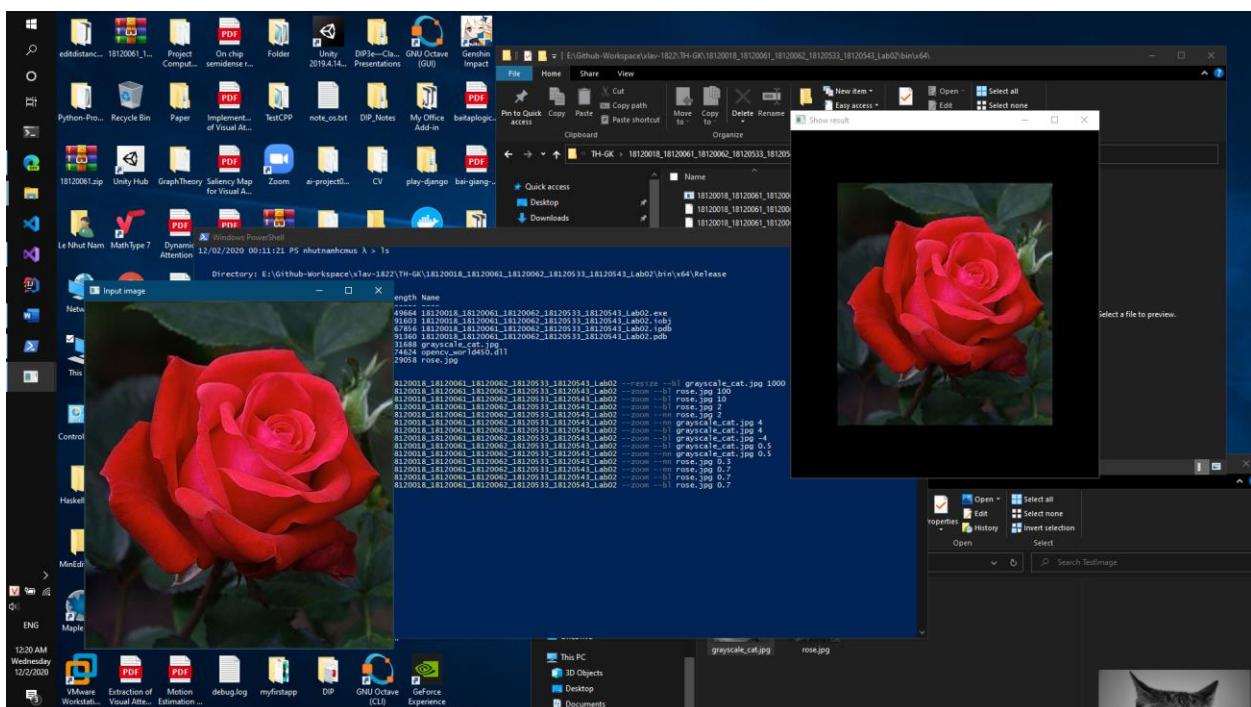
Thu nhỏ:

## Ảnh màu

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --zoom --nn rose.jpg 0.3

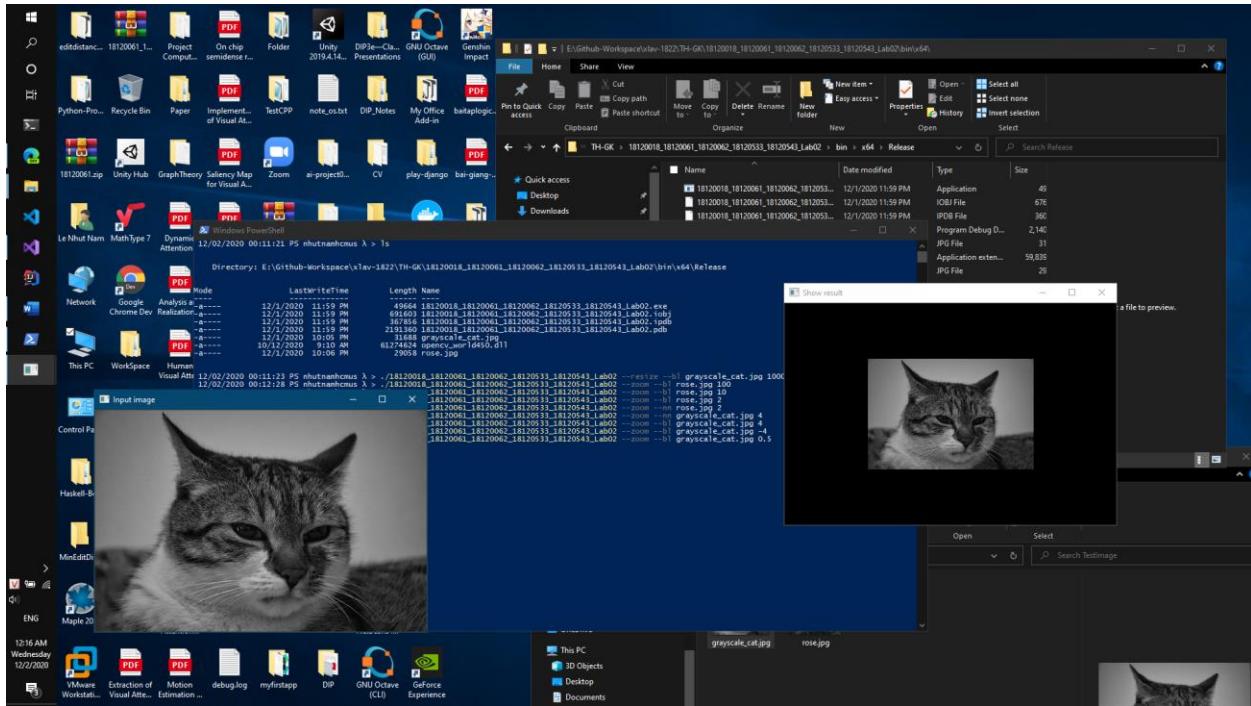


./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --zoom --bl rose.jpg 0.7

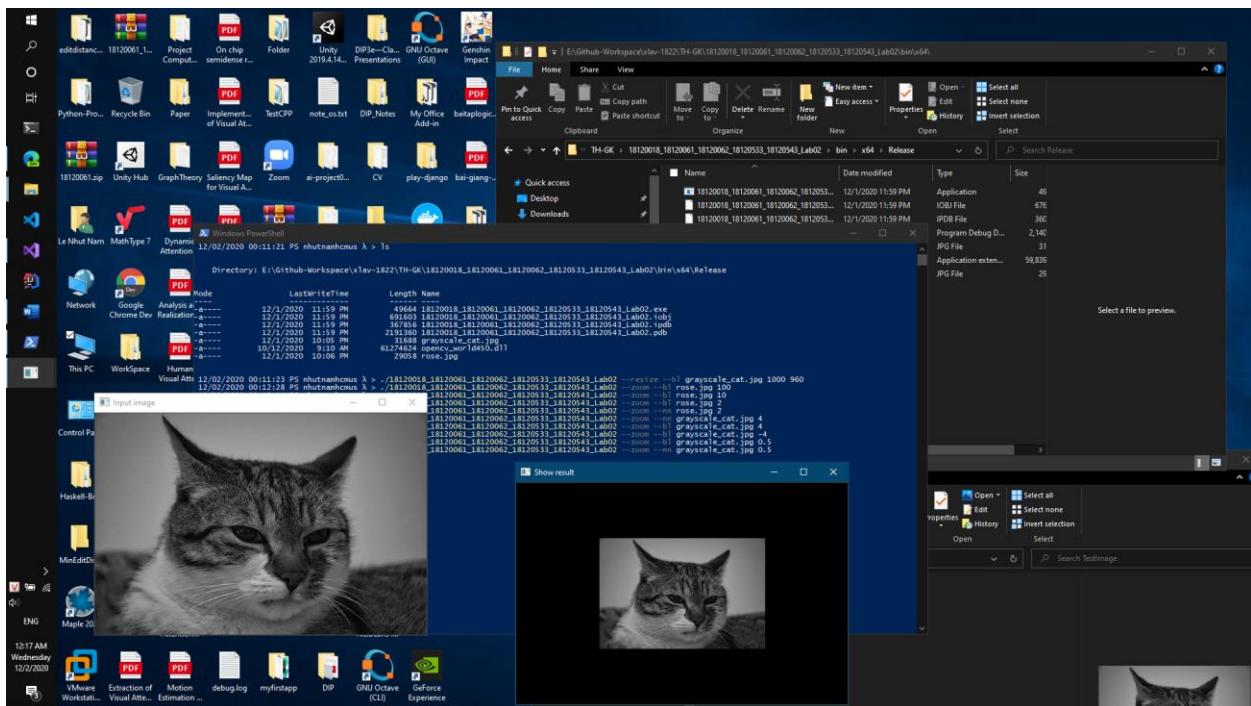


## Ảnh xám

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --zoom --bl  
grayscale\_cat.jpg 0.5



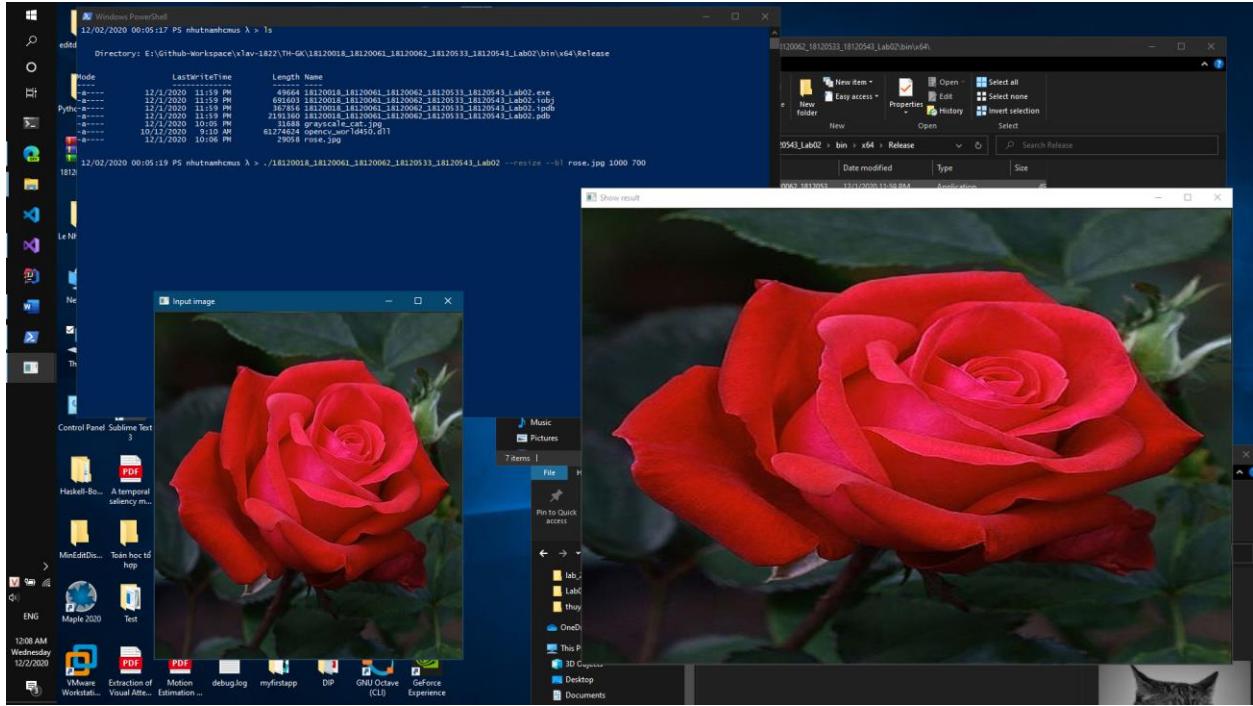
./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --zoom --nn  
grayscale\_cat.jpg 0.5



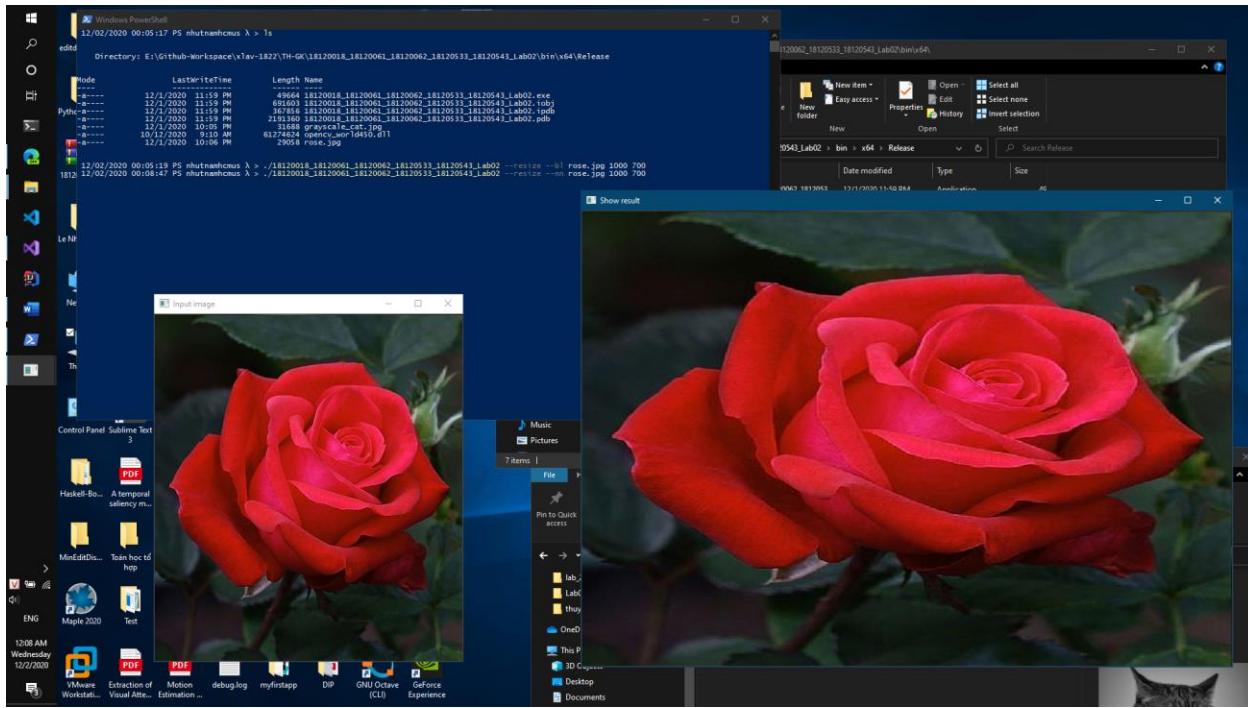
## 2. Câu 6: Thay đổi kích thước ảnh

Ảnh màu

```
./18120018_18120061_18120062_18120533_18120543_Lab02 --resize --bl rose.jpg  
1000 700
```

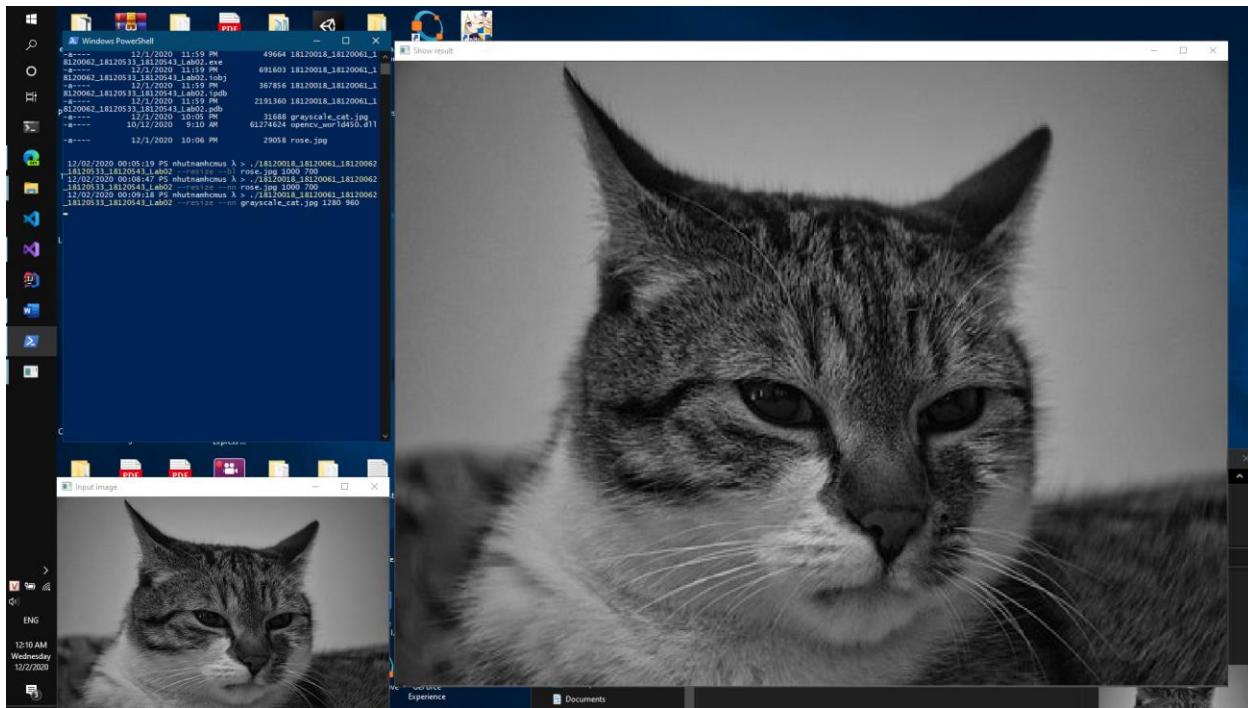


```
./18120018_18120061_18120062_18120533_18120543_Lab02 --resize --nn rose.jpg  
1000 700
```

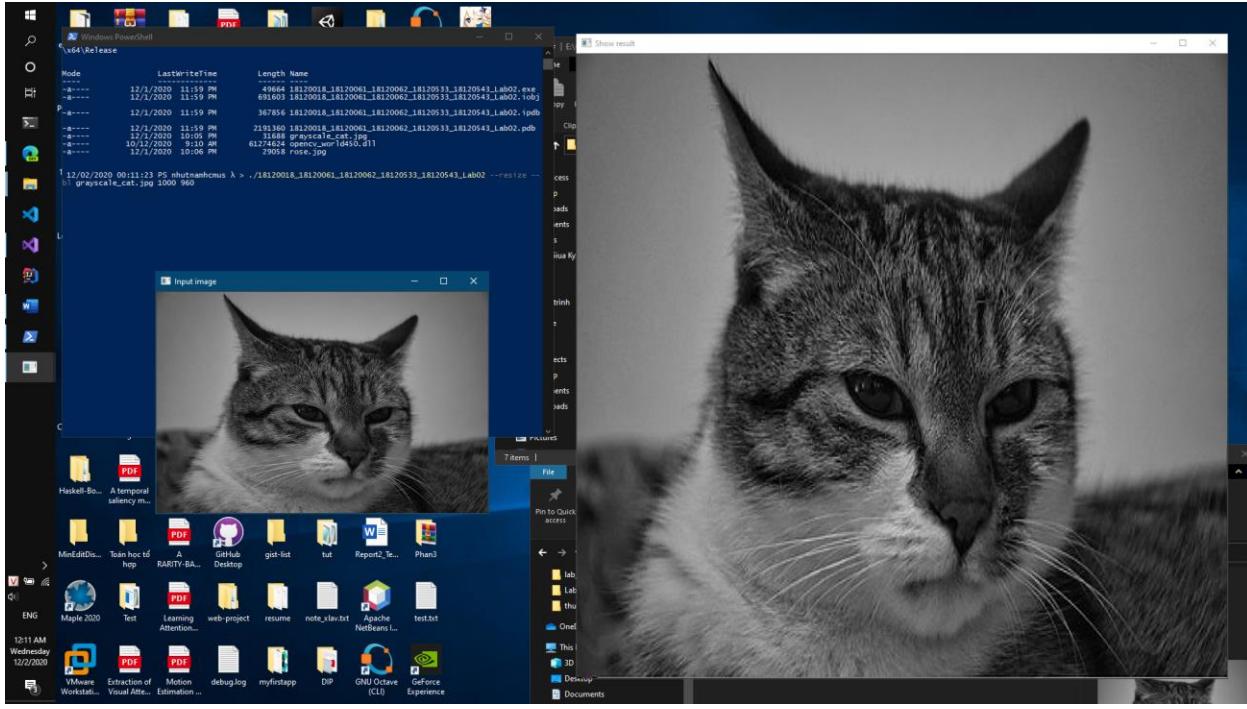


## Ảnh xám

/18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --resize --nn  
grayscale\_cat.jpg 1280 960



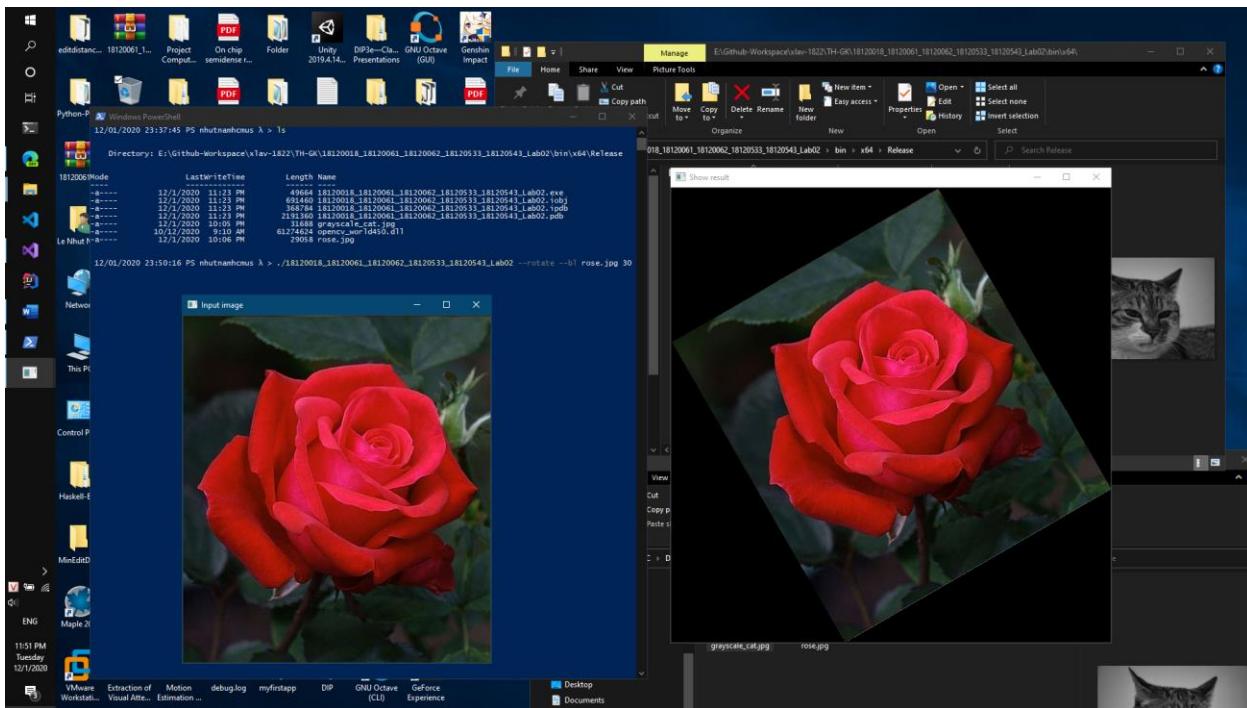
/18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --resize --bl  
grayscale\_cat.jpg 1000 960



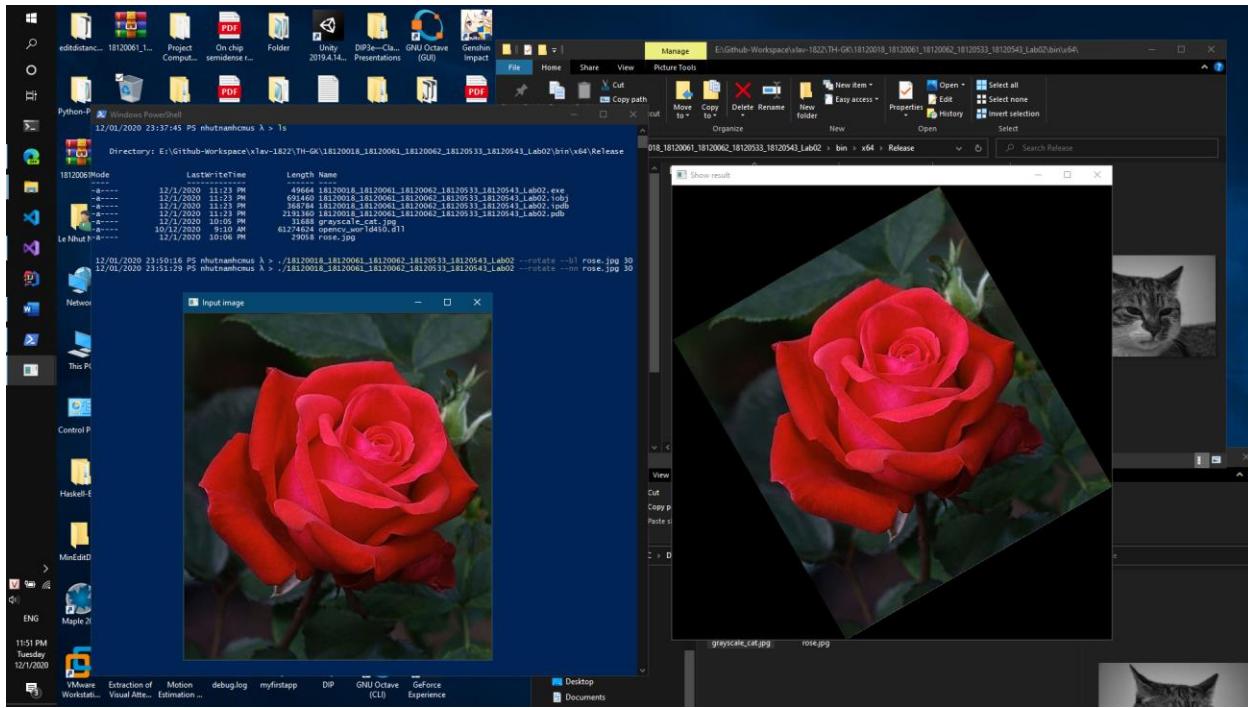
### 3. Câu 7: Xoay ảnh quanh tâm (bảo toàn nội dung ảnh)

Ảnh màu:

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --rotate --bl rose.jpg 30

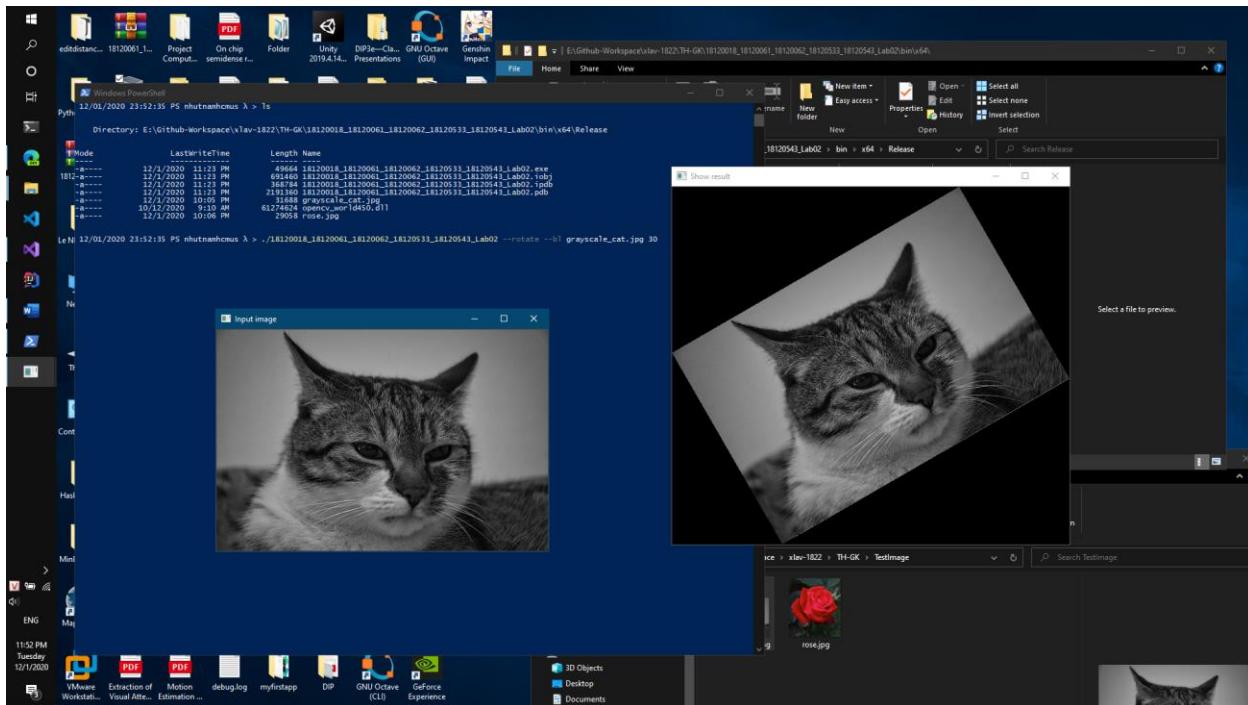


./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --rotate --nn rose.jpg 30

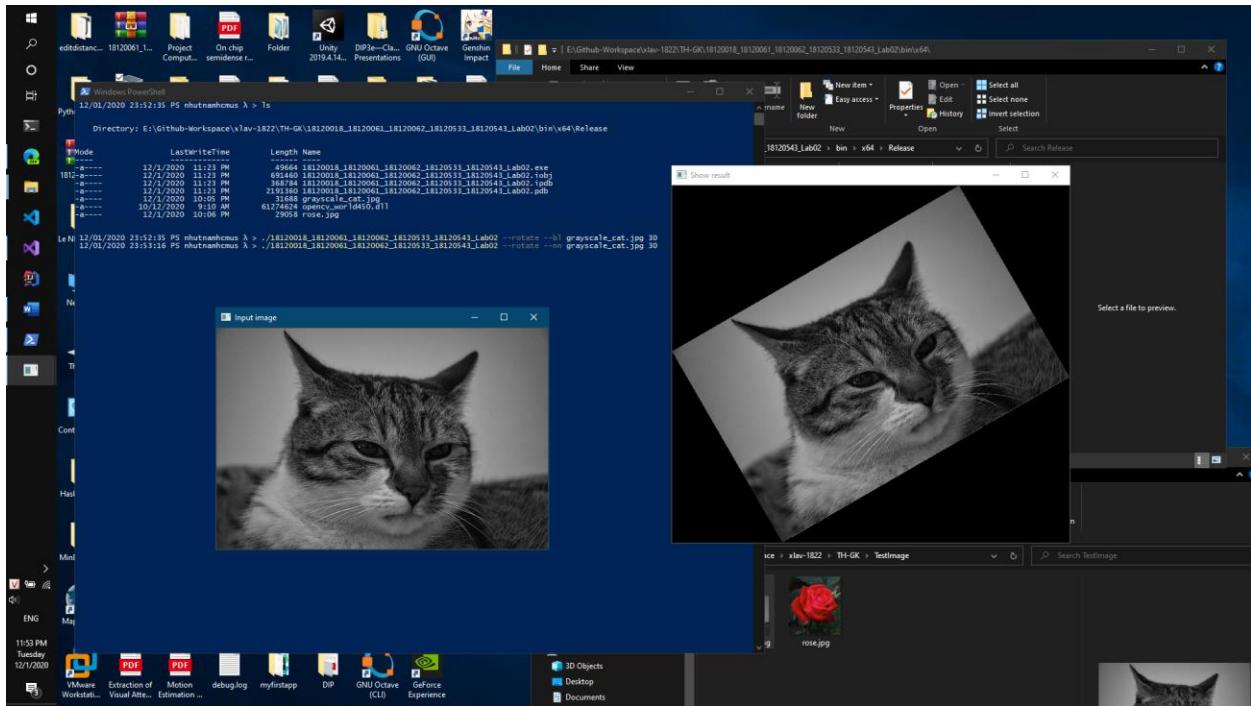


## Ảnh xám

`./18120018_18120061_18120062_18120533_18120543_Lab02 --rotate --bl  
grayscale_cat.jpg 30`



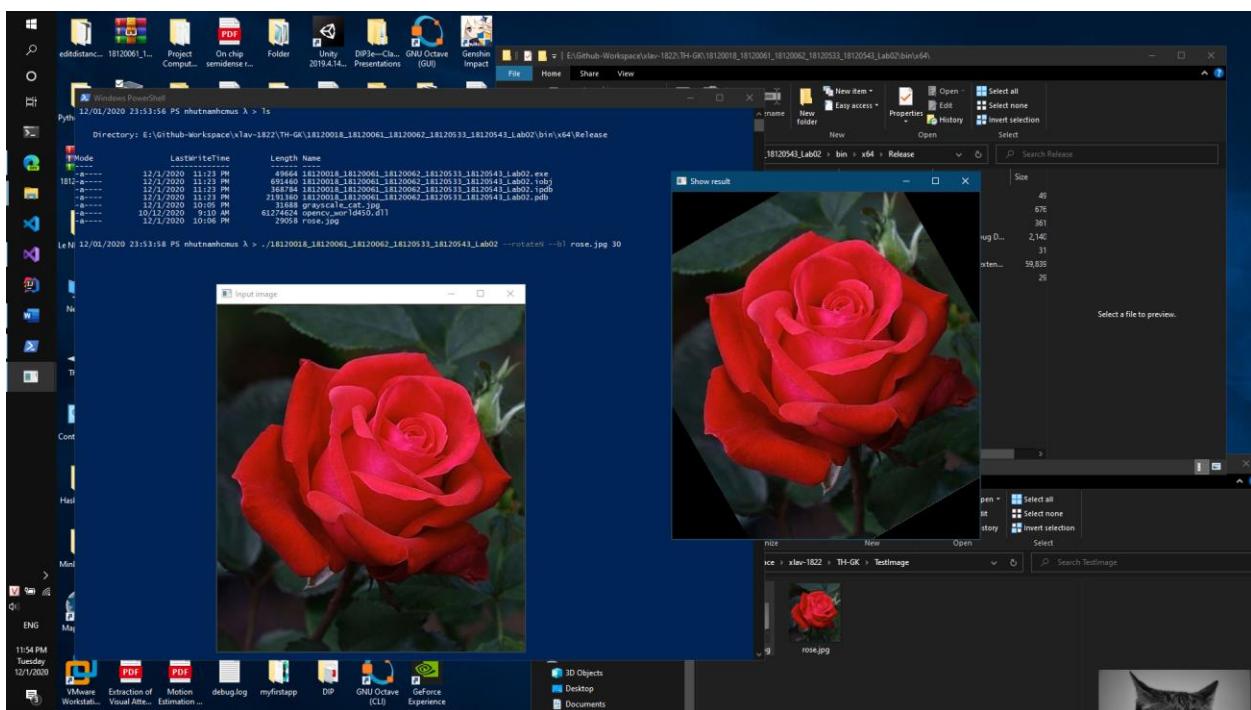
`./18120018_18120061_18120062_18120533_18120543_Lab02 --rotate --nn  
grayscale_cat.jpg 30`



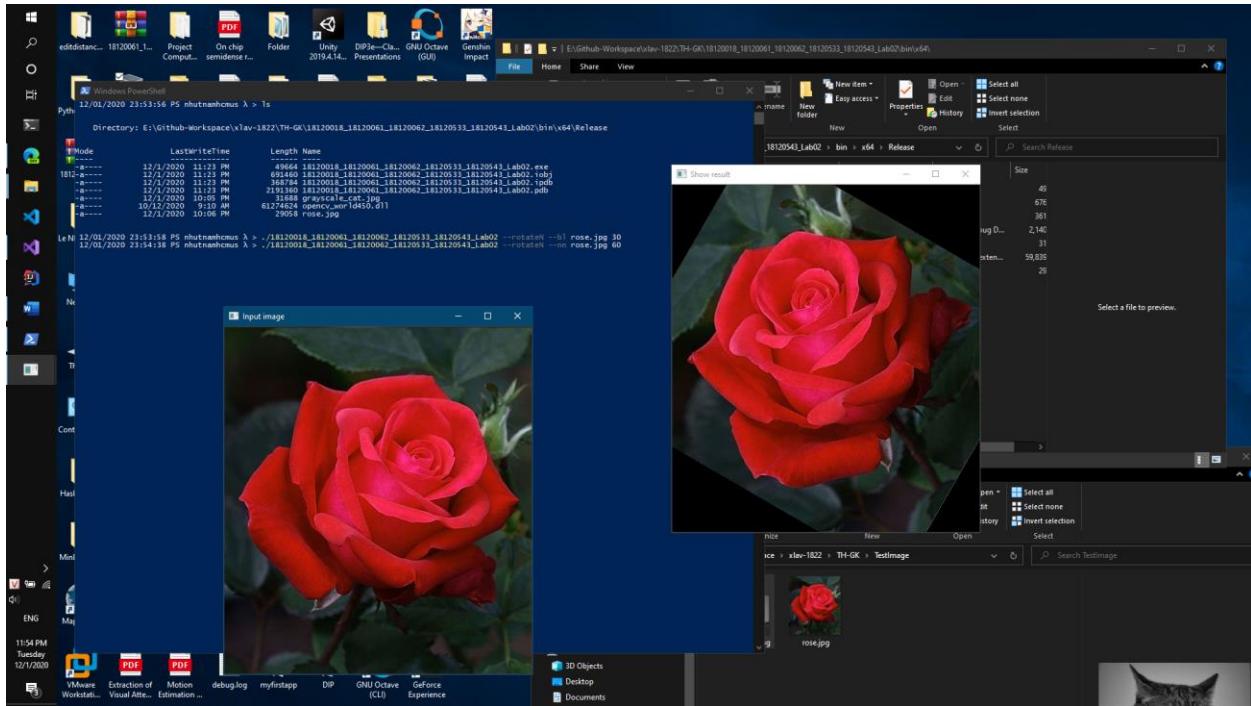
#### 4. Câu 8: Xoay ảnh quanh tâm (không bảo toàn nội dung ảnh)

Ảnh màu

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --rotateN --bl rose.jpg 30

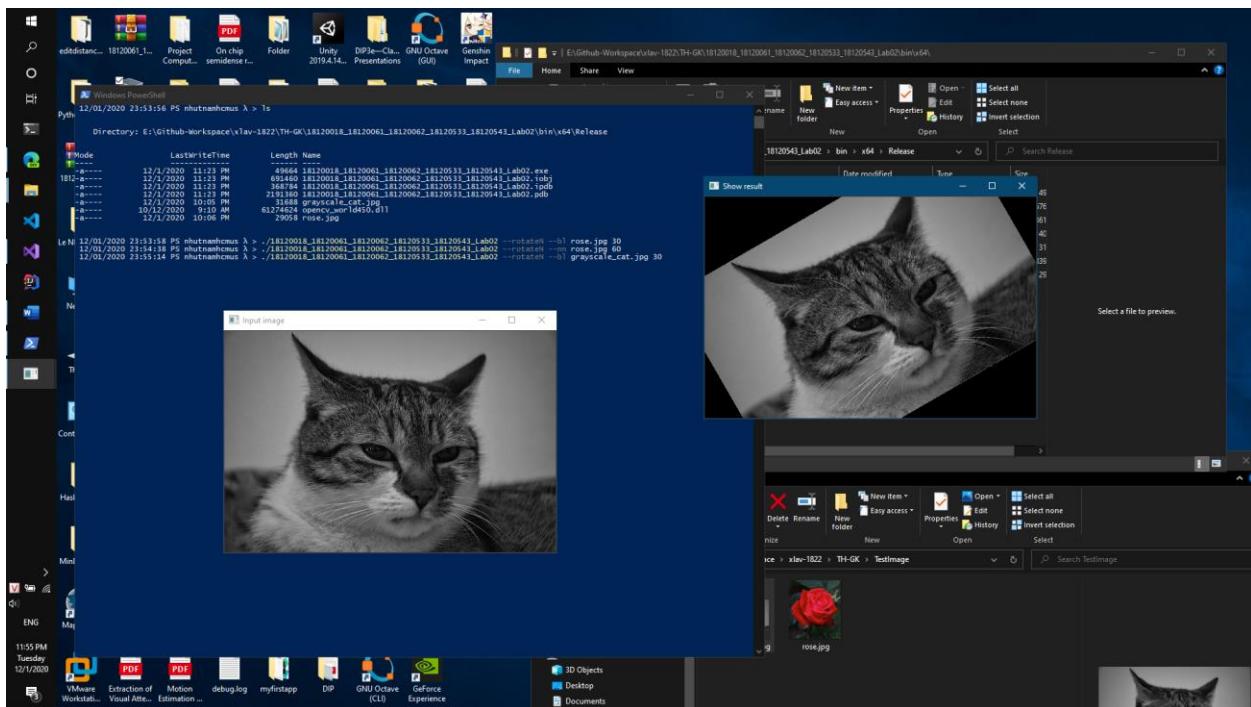


./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --rotateN --nn rose.jpg  
60

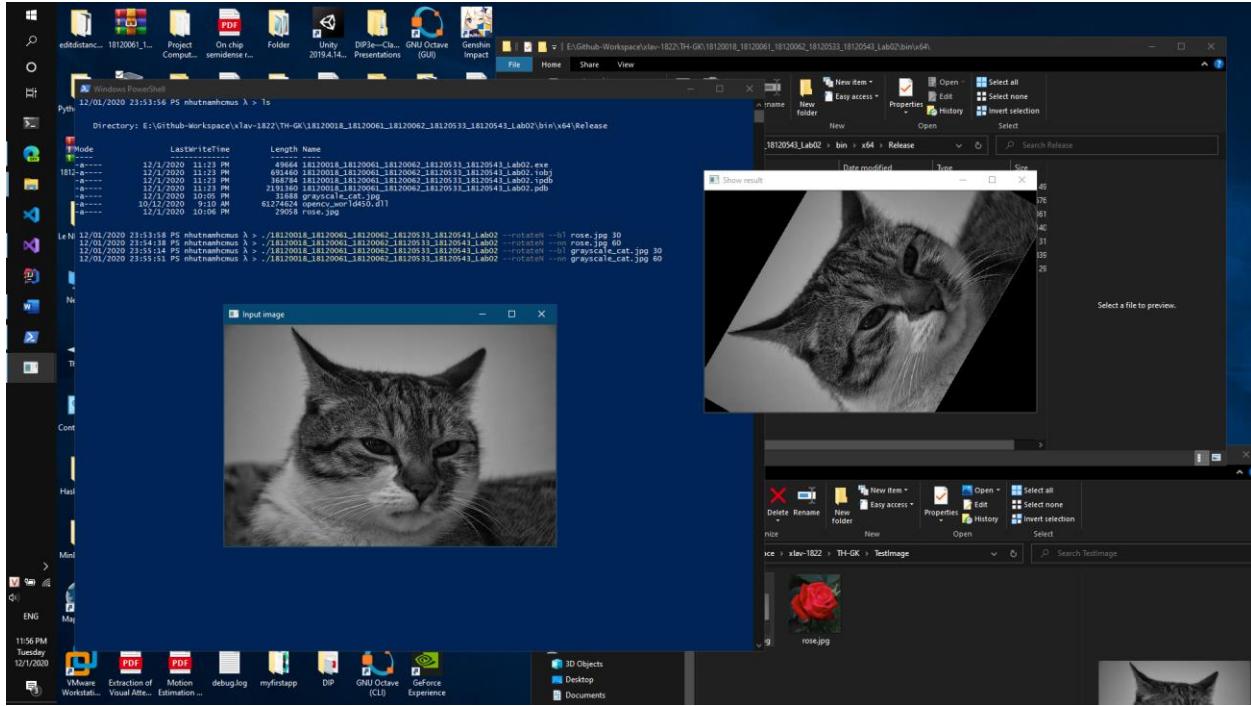


Ảnh xám

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --rotateN --bl  
grayscale\_cat.jpg 30



```
./18120018_18120061_18120062_18120533_18120543_Lab02 --rotateN --nn  
grayscale_cat.jpg 60
```

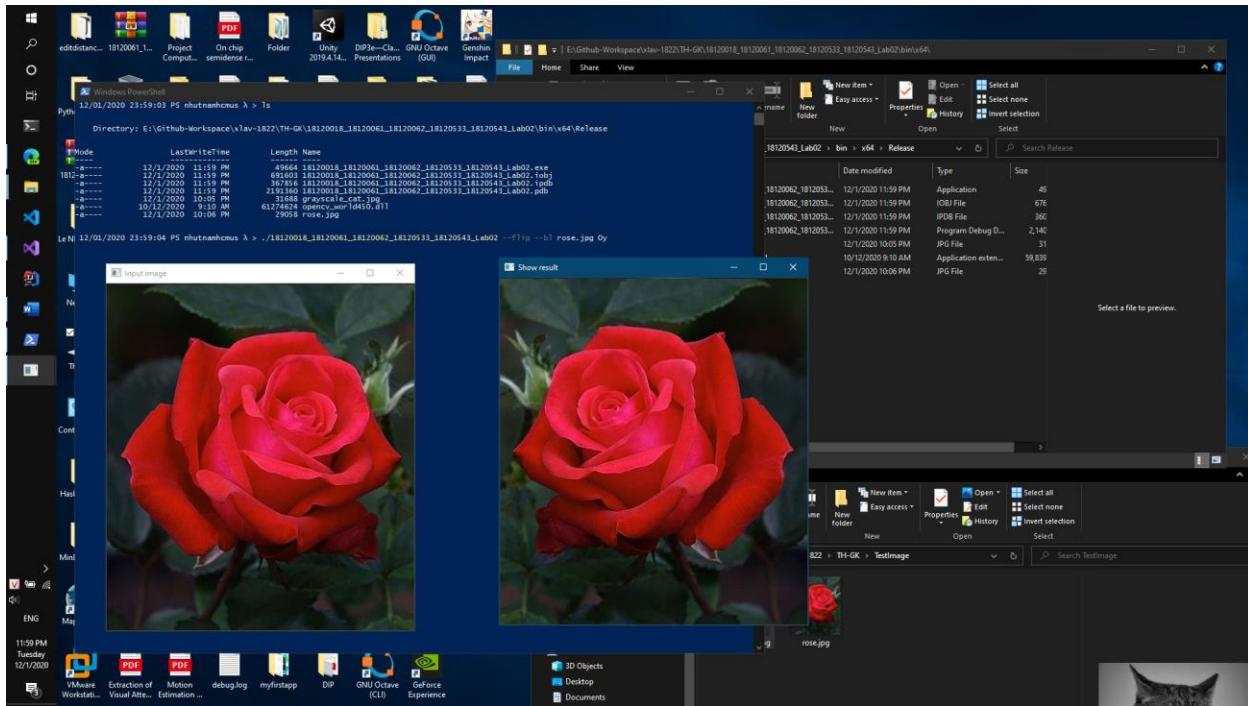


## 5. Câu 9: Đổi xứng ảnh qua trục đứng (Oy) hay trục ngang (Ox)

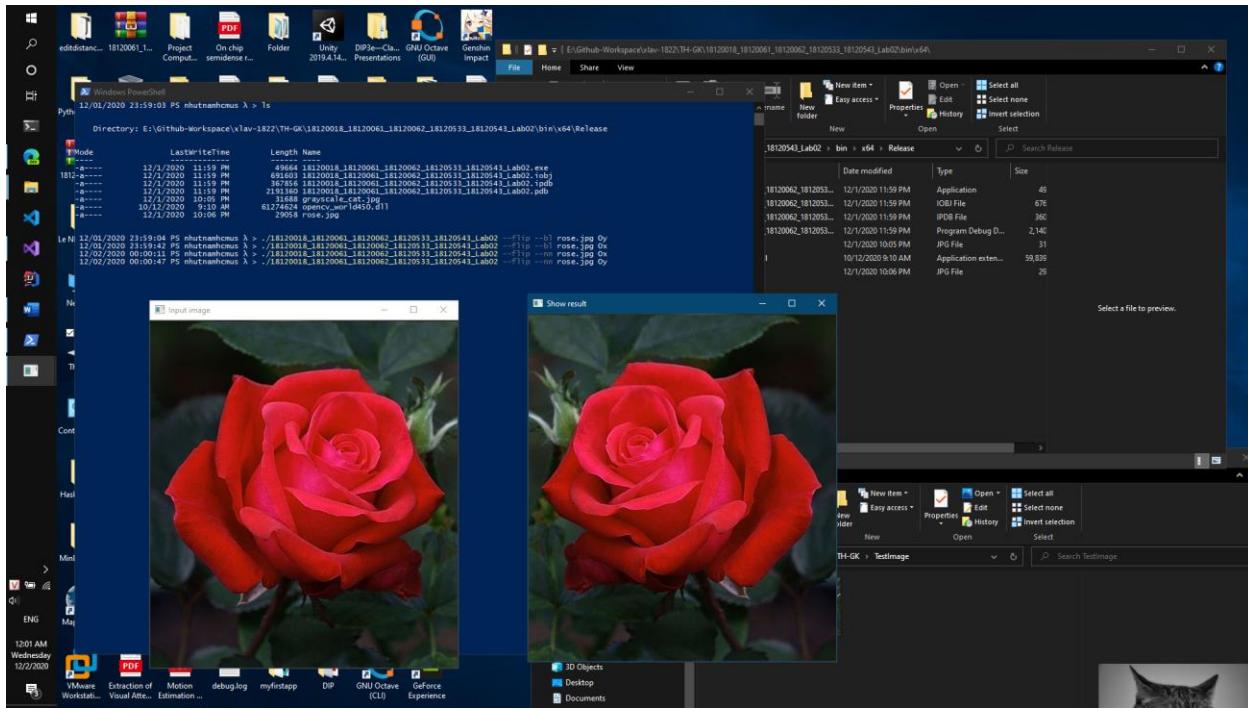
Ảnh màu

Đổi xứng trực đứng Oy

```
./18120018_18120061_18120062_18120533_18120543_Lab02 --flip --bl rose.jpg Oy
```

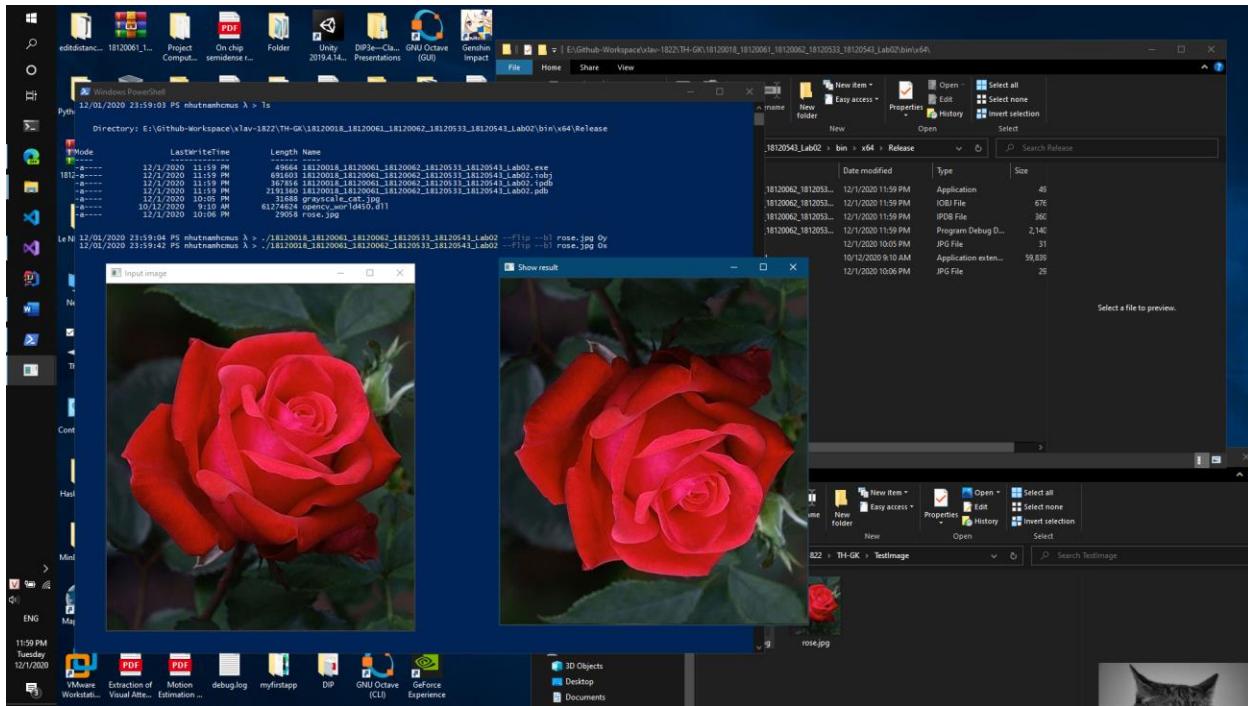


./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --flip --nn rose.jpg Oy

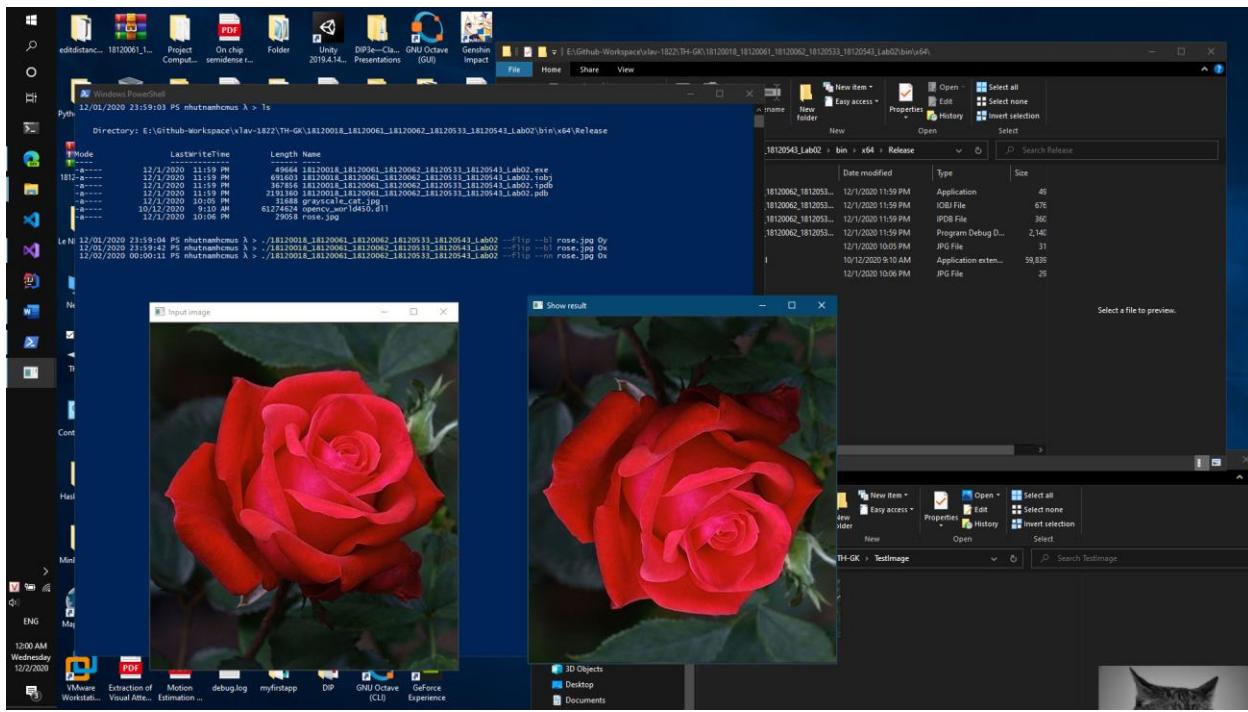


Đổi xứng trực ngang Ox

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --flip --bl rose.jpg Ox



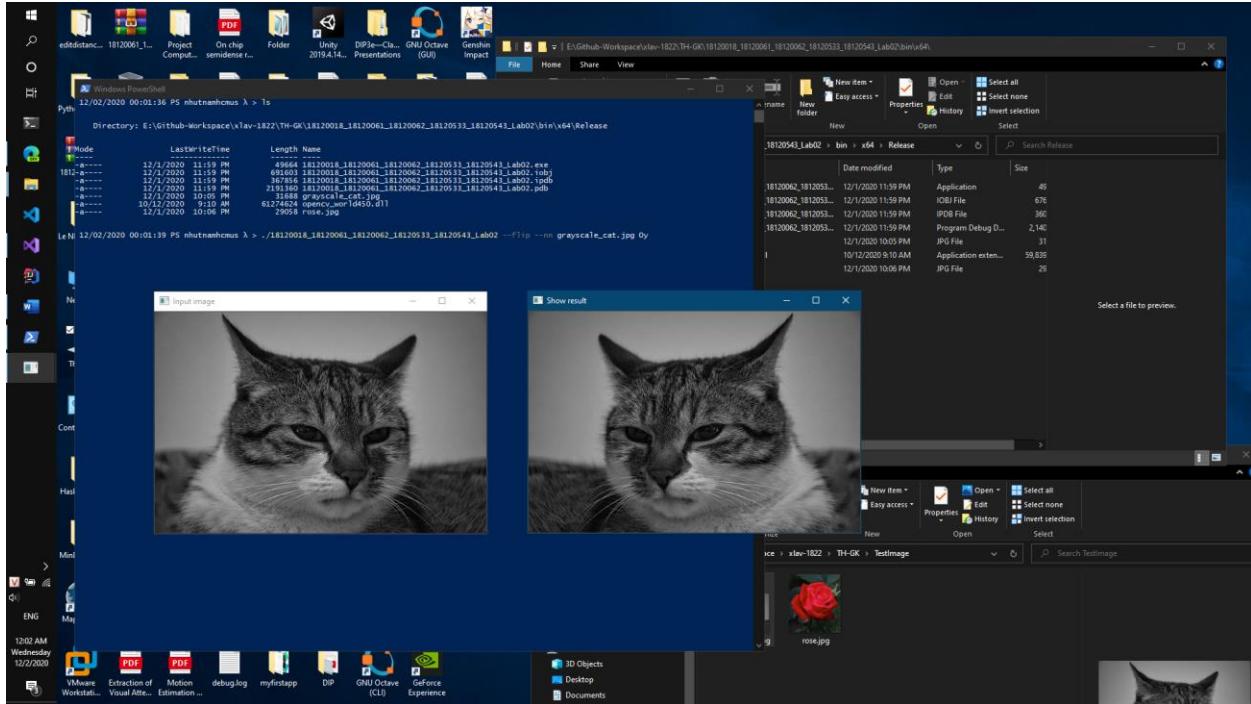
./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --flip --nn rose.jpg Oy



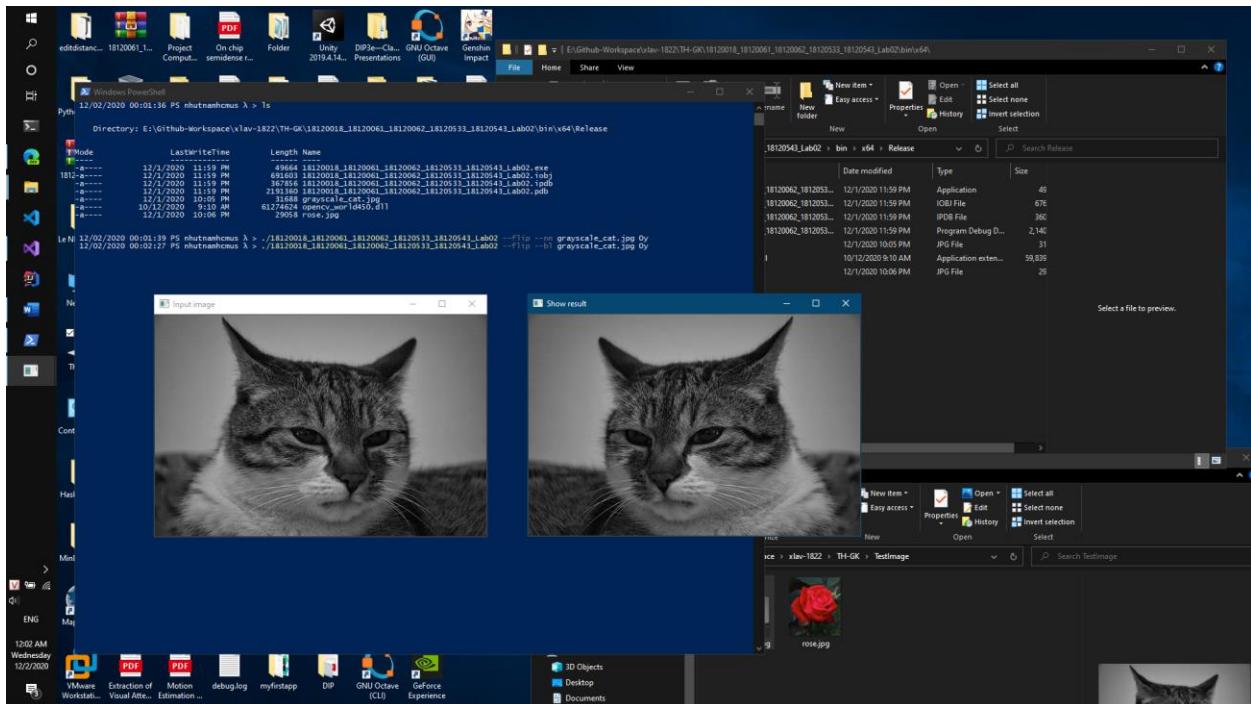
Ảnh xám

Đối xứng trực đứng Oy

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --flip --nn  
grayscale\_cat.jpg Oy

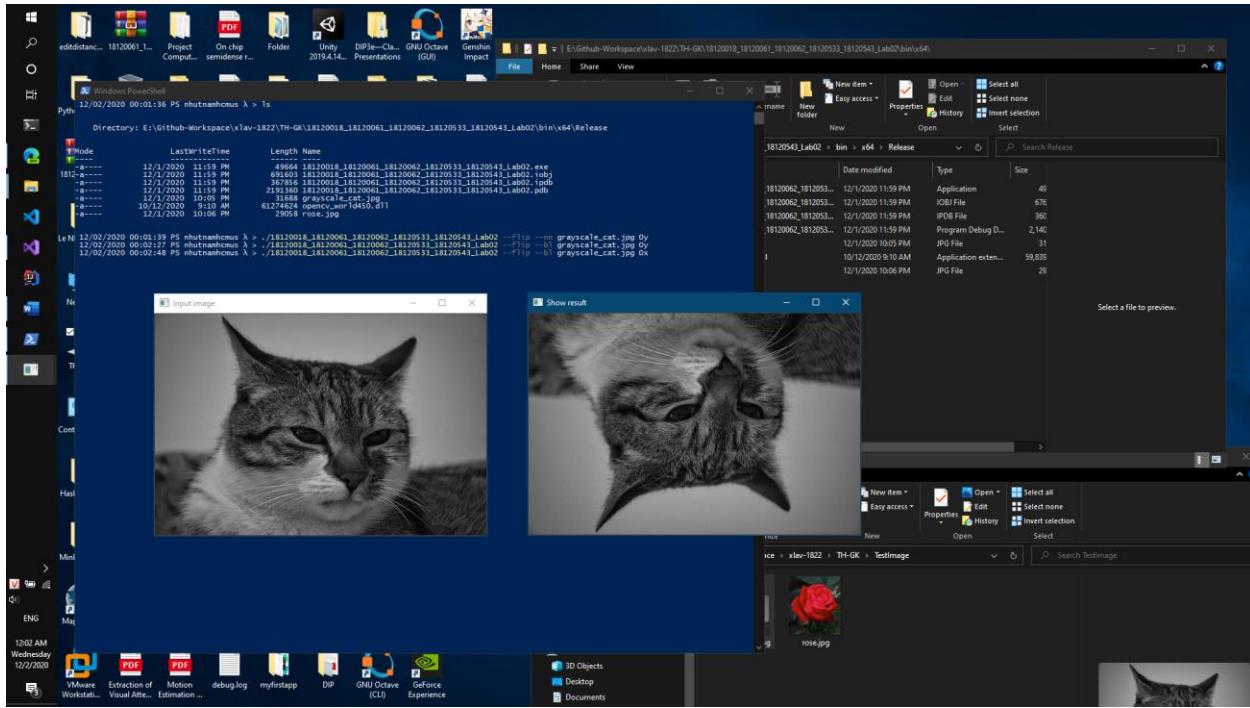


./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --flip --bl  
grayscale\_cat.jpg Oy

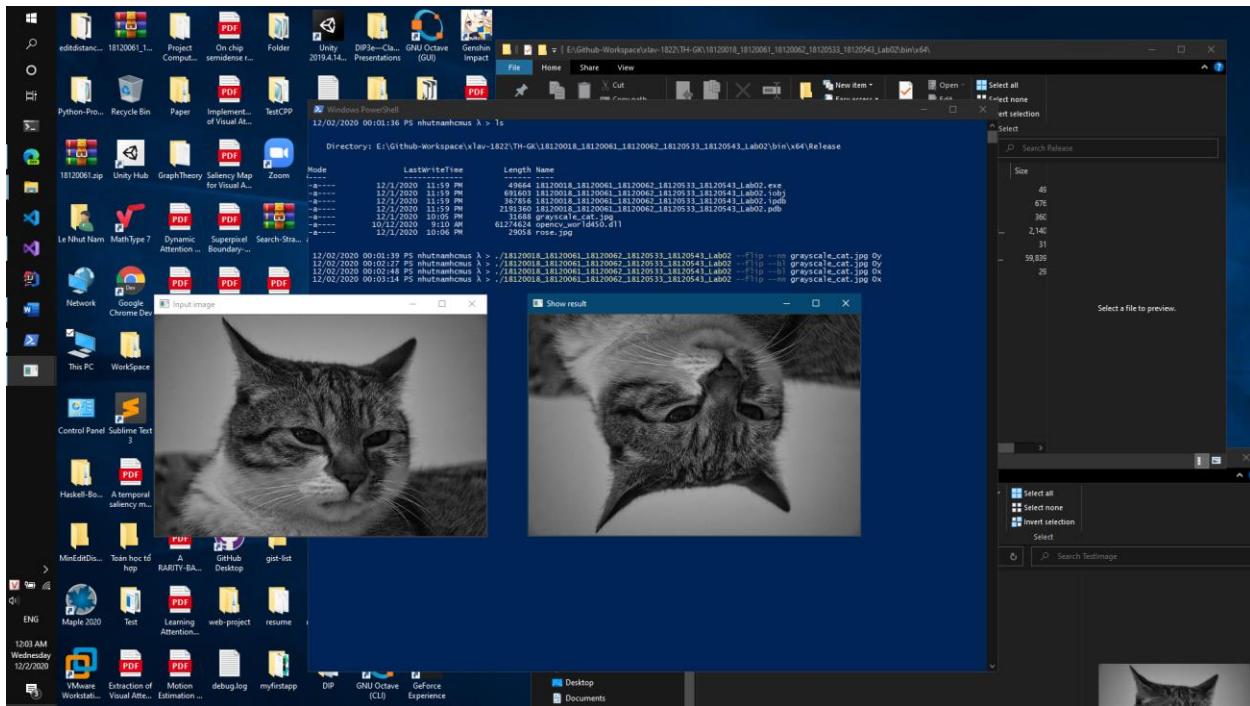


Đối xứng trực ngang Ox

./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --flip --bl  
grayscale\_cat.jpg Ox



./18120018\_18120061\_18120062\_18120533\_18120543\_Lab02 --flip --nn  
grayscale\_cat.jpg Ox



## Phản 4. Tổng kết

