ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

MÔN HỌC : Xử lý ảnh số và video số

ĐỒ ÁN THỰC HÀNH 1

BÁO CÁO

Tài liệu này mô tả nội dung đồ án môn học cho môn học Xử lý ảnh số và video số

Giảng viên hướng dẫn

Lý Quốc Ngọc

Nguyễn Mạnh Hùng

Sinh viên thực hiện

Phan Trí Nguyên - 20127578

Thành phố Hồ Chí Minh, ngày 16 tháng 11 năm 2022

MỤC LỤC

1

Tổng quan

1.1 Thông tin sinh viên

MSSV	Họ và tên	Email	Lớp
20127578	Phan Trí Nguyên	20127578@student.hcmus.edu.vn phantringuyen2002@gmail.com	20TGMT01

1.2 Thông tin đồ án

Tên đồ án	Môi trường lập trình
Image Processing	Phần mềm: Jupyther Notebook, Anaconda , Python 3.10, PyCharm Community Edition 2022.1.2.
	Ngôn ngữ: Python

Mô tả bài toán:

- Để thực hiện việc xử lý ảnh cơ bản ứng với nhu cầu người sử dụng từ thư viện OpenCV.
- Cài đặt thuật toán Edge detetion với những thuật toán đã học trên lớp và so sánh với hàm được thư viện hỗ trợ.
- Các thuật toán bao gồm:

✓ Transform color
✓ Transform geometry
✓ Smooth the image (image blurring, image smoothing)
✓ Edge detection algorithm

2

Ý tưởng thực hiện và mô tả chi tiết các hàm

2.1 Ý tưởng thực hiện

2.1.1 Transform Color-RGB image to Grayscale image: [1]

- Chuyển đổi hình ảnh màu thành hình ảnh có thang độ màu xám bằng cách lấy giá trị trung bình của các giá trị RGB (Red, Green, Blue): giảm các điểm ảnh có chứa điểm màu đỏ, điểm màu xanh lá và đặt điểm ảnh có màu xanh dương vào giữa 2 điểm ảnh này, và tăng các điểm ảnh có chứa, được tính theo trọng số ma trận.
- Từ đó, ta sẽ suy ra được công thức sau:

Điểm ảnh xám = ((0.3 * R) + (0.59 * G) + (0.11 * B))

- Chuyển ma trận vừa được xử lí thành ảnh và thực hiện in ra console.

2.1.2 Transform Color-RGB image to HSV image: [2]

- Thông thường, các đối tượng trong hình ảnh có màu sắc và độ sáng riêng biệt, do đó có thể sử dụng các đặc điểm này để phân tách các vùng khác nhau của hình ảnh.
- Trong biểu diễn RGB, màu sắc và độ sáng được biểu thị dưới dạng tổ hợp tuyến tính của các kênh R,G,B, trong khi chúng tương ứng với các kênh đơn lẻ của hình ảnh HSV (kênh Hue và Value).
- Sau đó, một phân đoạn hình ảnh đơn giản có thể được thực hiện một cách hiệu quả chỉ bằng cách tạo ngưỡng cho các kênh HSV.

2.1.3 Transform Geometry-Reduce quality of a RGB image:

- Thực hiện việc đổi kích thước của ảnh gốc bằng cách giảm bớt giá trị width và height.

- Tính tỉ số giữa giá trị width và height ban đầu và so sánh với tỉ số giữa giá trị width và height lúc sau để đảm bảo ảnh không bị tỉ số kích thước (không còn giữ khung ảnh như hiện trạng trước khi giảm chất lượng ảnh.
- Với thư viện, sử dụng thuật toán Linear interpolation **[3]** để giảm (hoặc tăng) kích thước ảnh.

2.1.4 Transform Geometry-Image Rotation: **[4]**

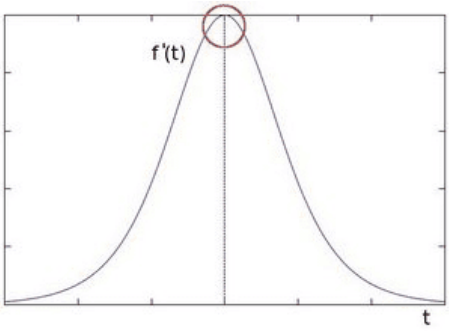
- Đối khi chúng ta xoay một hình ảnh, chúng ta cần chỉ định xung quanh điểm mà chúng ta muốn xoay. Trong hầu hết các trường hợp, bạn sẽ muốn xoay quanh tâm của hình ảnh; tuy nhiên, OpenCV cho phép bạn chỉ định bất kỳ điểm tùy ý nào mà bạn muốn xoay quanh.
- Đưa vào giá trị width và height của bức ảnh và chia đôi để lấy điểm trung tâm bức ảnh.
- Hàm cv2.getRotationMatrix2D nhận ba đối số. Đối số đầu tiên là điểm mà chúng ta xoay hình ảnh (trong trường hợp này là tâm x và y của hình ảnh). Sau đến góc xoay ảnh theo chiều kim đồng hồ và đối số cuối là tinh chỉnh kích thước bức ảnh (VD 0.5 là giảm đi kích thước ảnh phân nửa, 2.0 là nhân đôi kích thước ảnh và 1.0 là kích thước ảnh giữ nguyên).
- Tiếp theo chúng ta có thể áp dụng phép xoay cho hình ảnh của mình bằng phương thức cv2.warpAffine-Affine Transform.

2.1.5 Algorithm to smooth the image-Image blurring (Smoothing Image):

- Tạo một ma trận gốc Gaussian 3x3 **[5]**, tạo một ma trận phụ với heigth và width bằng với ma trận ảnh gốc với height từ giá trị thứ nhất (từ giá trị 0) đến giá trị n – 1 và step nhảy là 1.
- Thực hiện việc tính toán điểm ảnh theo dòng, cột; với các điểm ảnh theo khung 3x3 xung quanh 1 điểm ảnh nhất định trong ma trận phụ, thực hiện tính trung bình cộng và tổng hợp tất cả các số là chỉ số thứ 0 của ma trận Gauss hình dạng đề cập đến.
- Chuyển ma trận vừa được xử lí thành ảnh và thực hiện in ra console.

2.1.6 Edge detection-Sobel Edge Detection: **[6]**

- Toán tử Sobel phát hiện các rìa được đánh dấu bằng những thay đổi về cường độ pixel và sự gia tăng cường độ thậm chí còn rõ ràng hơn khi vẽ đạo hàm bậc nhất của hàm.



- Đầu tiên ta cần phát hiện được các "rìa" ở những khu vực có độ dốc cao hơn một giá trị ngưỡng cụ thể, bằng một ma trận kernel 3x3. Sử dụng một kernel để xác định những thay đổi về cường độ pixel theo hướng X, một kernel theo hướng Y.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

X-Direction Kernel Y-Direction Kernel

- Và một khi các kernel này được kết hợp với hình ảnh ban đầu (origin image), kết quả nhận được là một hình ảnh rìa Sobel.
- Đặt G_x và G_y là đại diện cho "intensity gradient" theo hướng x và y tương ứng. Với A và B biểu thị cho các kernel X và kernel Y bên trên, và ma trận gốc origin_Image:

$$G_x = A * origin\_Image$$

$$G_y = B * origin\_Image$$

- Sau đấy thực hiện tính toán xấp xỉ của "gradient magnitude" với:

$$G = \sqrt{Gx^2 * Gy^2}$$

- Và hướng của gradient sẽ được tính với giá trị gần đúng như sau:

$$\Theta = arctan(G_x / G_y)$$

2.1.7 Edge detection-Canny Edge Detection: [7]

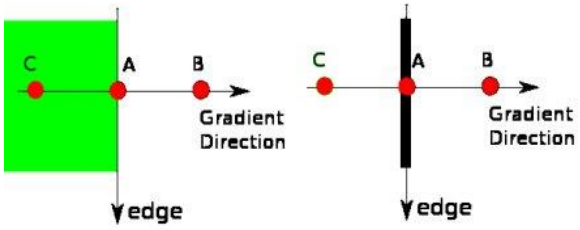
- "A multi-stage algorithm"
- Noise Reduction: Do tính năng phát hiện rìa dễ bị nhiễu trong ảnh nên bước đầu tiên là loại bỏ nhiễu trong ảnh bằng bộ lọc Sobel 3x3.

- Tính toán "the intensity gradient" từ bức ảnh: sử dụng công thức sau:

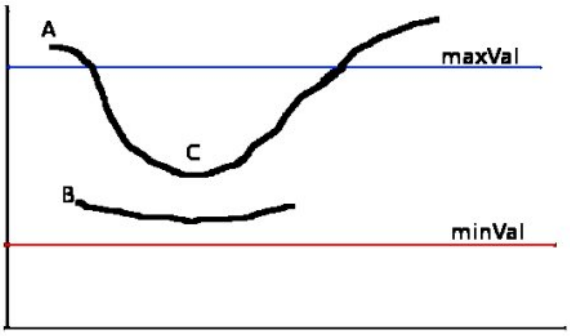
$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

- Hướng dốc luôn vuông góc với các direction
- Non-maximum Suppression: sau khi tính toán được gradient magnitude và direction. Thực hiện việc quét toàn bộ những pixel không cần thiết (những pixel không thể tạo thành các rìa)



- Điểm A đang nằm trên rìa (theo hướng Oy).
- Gradient direction thông thường với các rìa nên xét với điểm B và C, điểm A được kiểm định so với B và C nếu chúng tạo thành một cực đại cục bộ (local maximum). Nếu chúng tạo thành local maximum, đến với bước tiếp theo, ngược lại, triệt tiêu điểm thừa bằng cách cho giá trị pixel tại đấy là 0.
- Bước chuyển đổi cuối là Hysteresis Thresholding:
- Bước này quyết định những rìa nào là rìa cần thiết cho vật chính. Vì điều này, thuật toán sẽ cần truyền vào hai giá trị, bao gồm minVal và maxVal.
- Bất kì rìa nào với intensity gradient lớn hơn maxVal sẽ trở thành rìa cho vật chính và rìa nào nhỏ hơn minVal sẽ loại bỏ.
- Những rìa nằm giữa 2 giá trị này (gọi là thresholds) sẽ được phân loại có trở thành rìa trong ảnh chính hoặc loại bỏ, tùy thuộc vào điểm kết nối giữa chúng. Nếu những rìa này kết nối với những pixel "sure-edge" thì sẽ là một trong những rìa vật chính.



- Rìa A nằm trên đường maxVal nên được xét là "sure-edge"
- Rìa C nằm giữa 2 thresholds nhưng được kết nối với rìa A nên được xét là rìa hợp lệ cho vật chính nên ta xét cả đường cung AC.
- Mặc dù rìa B nằm giữa 2 thresholds như rìa C nhưng lại không có đường kết nối với sure-edge trong hình là rìa A

2.2 Mô tả các hàm

Tên hàm	Mô tả

`import numpy as np`
`import cv2`
`import matplotlib.pyplot as plt`
`import imageio.v2 as imageio`
`from PIL import Image`
`from scipy import ndimage`
`%matplotlib inline`
	Import các thư viện cần thiết trong đó:
	- matplotlib.pyplot: hiển thị ảnh.
	- numpy: dùng cho tính toán trên ma trận.
	- PIL: đọc ảnh.
	- cv2: opencv-python
	- imageio.v2: cung cấp một giao diện dễ dàng để đọc và ghi nhiều loại dữ liệu hình ảnh, bao gồm hình ảnh động, dữ liệu thể tích,…

`def RGB2Gray_opencv(img_path):`	- Chuyển đổi từ ảnh RGB sang ảnh Grayscale với hàm được xây dựng sẵn trong thư viện OpenCV - **Input**: tên file ảnh. - **Output**: ảnh sau khi được Greyscale.
`def to_grayscale(image):`	- Thực hiện việc chuyển đổi từ ảnh RGB thành ảnh xám Greyscale. - Với công thức các điểm ảnh xám 0.3 x red, 0.59 x green và 0.11 x blue. Lặp lại số lượng phần tử có trong một mảng với hàm np.tile(). Khi in ảnh ra, sử dụng cmap = 'Greys' để thực hiện in ảnh xám. - **Input:** mảng 3 chiều ma trận ảnh gốc (RGB image). - **Output:** mảng 3 chiều ma trận với các điểm ảnh xám tương ứng.
`def RGB2Greyscale(image):`	- Thực hiện việc chạy trung gian giữa hàm main và hàm to_grayscale(image). - Chuyển đổi ảnh gốc thành ma trận. - Đưa ma trận 3 chiều vào hàm to_grayscale(image) xử lí và chuyển ma trận 3 chiều Greyscale thành ảnh, in ra console và lưu ảnh. - **Input:** ảnh gốc - **Output:** ảnh sau khi được Greyscale.

`def RGB2HSV_opencv(image):`	- Thực hiện việc chuyển đổi giữa ảnh RGB sang ảnh HSV bằng hàm thư viện OpenCV - **Input:** ảnh gốc được giải mã bởi hàm thư viện OpenCV và chuyển lại theo thứ tự RGB trước khi chuyển sang HSV - **Output:** ảnh sau khi được chuyển đổi thành HSV thành công
`def reduceImageSize_opencv(image):`	- Giảm đi kích thước của ảnh gốc bằng cách giảm bớt giá trị width và height - **Input:** ảnh gốc được giải mã bởi hàm thư viện OpenCV và chuyển lại theo thứ tự RGB trước khi chuyển sang HSV - **Output:** ảnh sau khi được giảm kích thước đi phân nửa.
`def reduceImageSize(image, new_width, new_height):`	- Thực hiện việc giảm kích thước của ảnh gốc bằng cách giảm bớt giá trị width và height bằng thuật toán tự xây dựng - **Input:** ảnh được đọc bởi thư viện numpy (RGB image), kích thước chiều dài mới, chiều cao mới của ảnh. - **Output:** ảnh sau khi được giảm kích thước thành công.
`def imageRotation_opencv(image, degrees):`	- Thực hiện việc xoay ảnh với một góc degree so với góc ban đầu. - **Input:** ảnh gốc được giải mã bởi hàm thư viện OpenCV và chuyển lại theo thứ tự RGB trước khi chuyển sang HSV - **Output:** ảnh sau khi được xoay theo một góc degrees theo chiều kim đồng hồ

`def GaussianBlur_opencv(image, sigmaX, sigmaY):`	- Thực hiện việc làm mờ ảnh thông qua Gaussian kernel. - Chỉ định chiều rộng và chiều cao của kernel là số dương và số lẻ, đồng thời độ lệch chuẩn theo hướng X và Y, sigmaX và sigmaY tương ứng. - Nếu chỉ sigmaX được chỉ định, sigmaY được lấy giống như sigmaX. Nếu cả hai đều là số không, thì chúng được tính từ kích thước hạt nhân. Làm mờ Gaussian có hiệu quả cao trong việc loại bỏ nhiễu Gaussian khỏi hình ảnh. - **Input:** ảnh gốc được giải mã bởi hàm thư viện OpenCV và chuyển lại theo thứ tự RGB trước khi chuyển sang HSV, sigmaX và sigmaY - **Output:** ảnh kết quả sau khi được làm mờ.
`def blurImage(image_path):`	- Thực hiện việc làm mờ bức ảnh. - Lấy ra chiều dài, chiều rộng của ảnh. - Tạo một ma trận Gaussian - Thực hiện việc tính toán điểm ảnh theo dòng, cột; với các điểm ảnh theo khung 3x3 xung quanh 1 điểm ảnh nhất định trong ma trận phụ, thực hiện tính trung bình cộng và tổng hợp tất cả các số là chỉ số thứ 0 của ma trận Gauss hình dạng đề cập đến. - Chuyển ma trận về dạng ảnh, in ảnh ra console và lưu ảnh. - **Input:** ảnh gốc được đọc bởi thư viện numpy - **Output:** ảnh kết quả sau khi được làm mờ.
`def SobelDetection(dx, dy):`	- Thực hiện việc tìm rìa ảnh để phát hiện rìa/cạnh trong một bức ảnh - Ghi chú: ảnh trước khi xử lí thuật toán Sobel tìm cạnh thì nên chuuyển về dạng ảnh RGB và làm mờ ảnh để dễ tìm được các rìa dễ hơn. - **Input:** ảnh sau khi được blur và grayscale, giá trị đạo hàm của theo x và theo y - **Output:** ảnh sau khi tìm được rìa

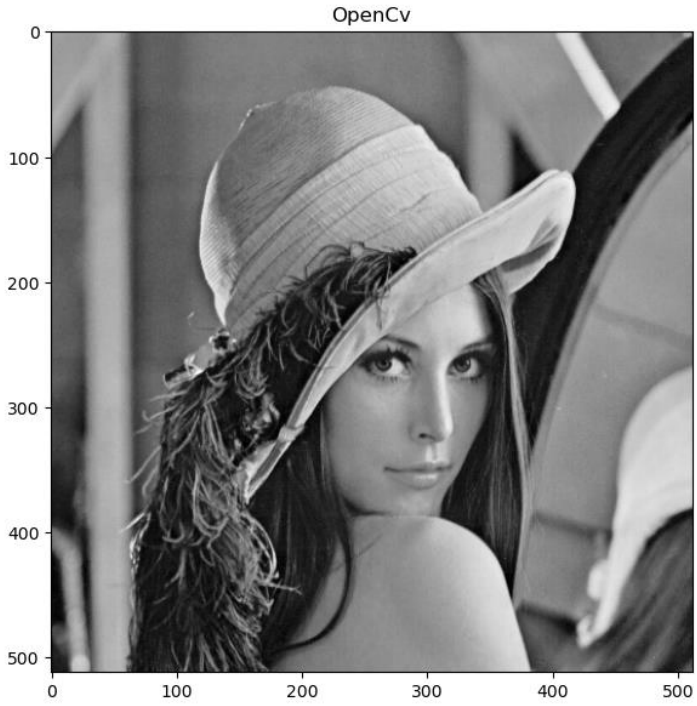
`def CannyEdgeDetect(threshold1, threshold2):`	- Thực hiện việc tìm ra rìa biên bằng thuật toán Canny Edge Detection với hàm thư viện OpenCV. - **Input:** threshold1, threshold2 - **Output:** các rìa biên được phân biệt so với bìa.
`def Sobel_filter(img, dimension):`	- Thực hiện việc tính toán gradient cho G_x và G_y để sử dụng Sobel filter - **Input:** ảnh sau khi được blur và grayscale, dimension để tính - **Output:** ma trận
`def Normalize(Gx, Gy):`	- Thực hiện việc chuyển hóa mảng pixel, sao cho các giá trị <= 1 - **Input:** 2 ma trận Gx và Gy vừa tính được từ hàm Sobel filter trên - **Output:** in
`def closest_dir_function(grad_dir) :`	- Thực hiện việc tìm ra gradient direction gần nhau nhất để giảm bớt độ nhiễu của ảnh - **Input:** ma trận điểm ảnh với gradient direction sau khi thực hiện tính góc lệc. - **Output:** ma trận điểm ảnh sau khi chuyển đổi các giá trị có góc lệch 0, 45, 90 và 135 độ. Điều này sẽ bắt được cạnh trái và phải của chân ghế vì điều này sẽ thấy sự khác biệt về cường độ của các đối tượng được căn chỉnh theo chiều dọc trên hình ảnh.
`def non_maximal_suppressor(grad_mag, closest_dir) :`	- Thực hiện việc chuyển đổi cho rìa trở nên mỏng hơn - **Input:** ma trận ảnh sau khi chuyển đổi thành công các góc lệch - **Output:** ma trận ảnh được loại bỏ các pixel không phải là cạnh của ảnh vật chính - một hình ảnh nhị phân với "thin edges".

`def DFS(img) :`	- Chức năng bao gồm các pixel yếu được kết nối với chuỗi pixel mạnh - **Input:** ma trận ảnh bao gồm cả chuỗi pixel yếu mà pixel mạnh - **Output:** in ra ma trận ảnh chỉ bao gồm pixel mạnh
`def hysteresis_thresholding(img) :`	- Sử dụng threshold để tạo loại bỏ cạnh giả, xác định cạnh thực sự - **Input:** ma trận ảnh bao gồm cả cạnh thật và cạnh giả - **Output:** ma trận ảnh khi lấy trung bình các thành phần, chúng ta giảm nhiễu vì chúng ta loại bỏ các thành phần "not sure-edge" trong ma trận ảnh.

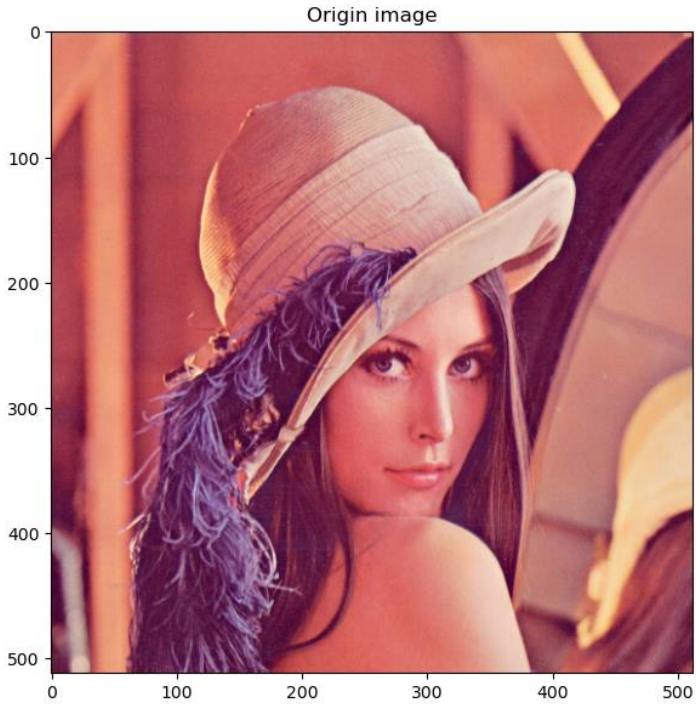
3

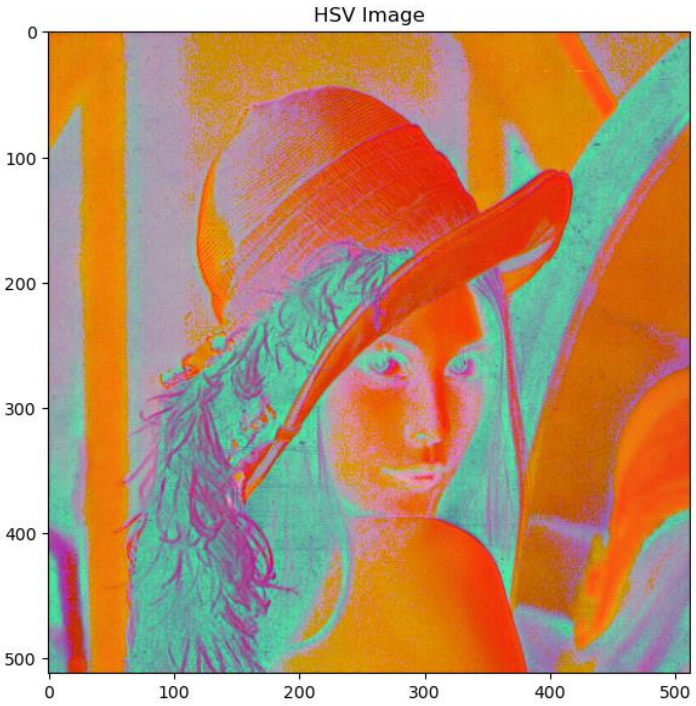
Demo kết quả chạy chương trình

3.1 Tranform RGB image to Grayscale image



3.2 Transform RGB image to HSV image

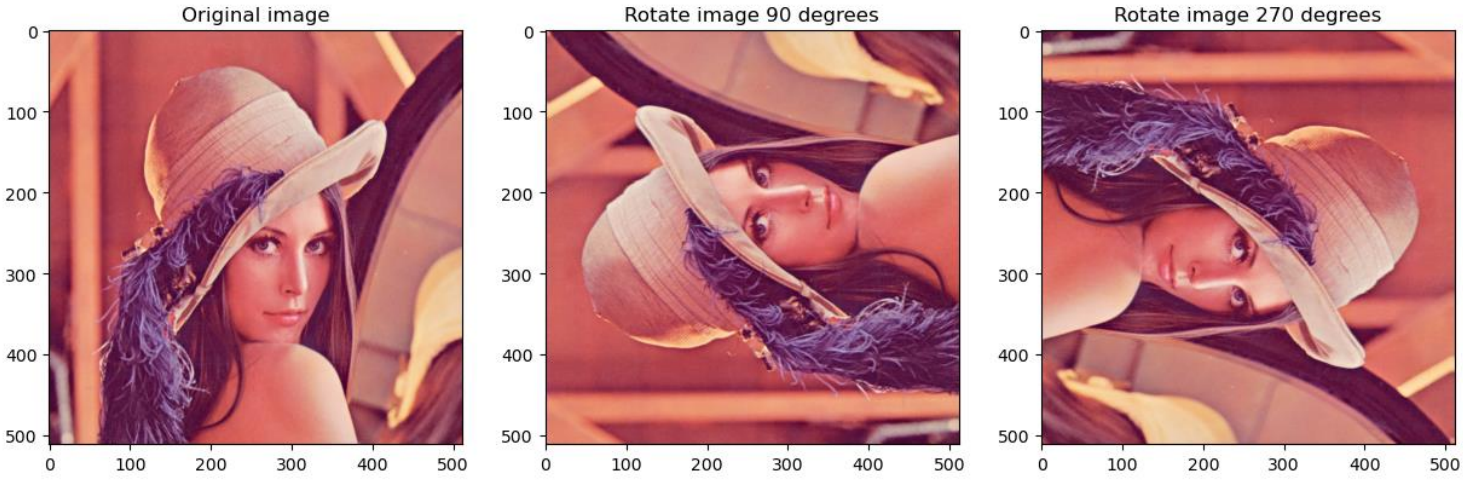




3.2 Reduce quality of a RGB image

Image after desized by half: (256, 256, 3)

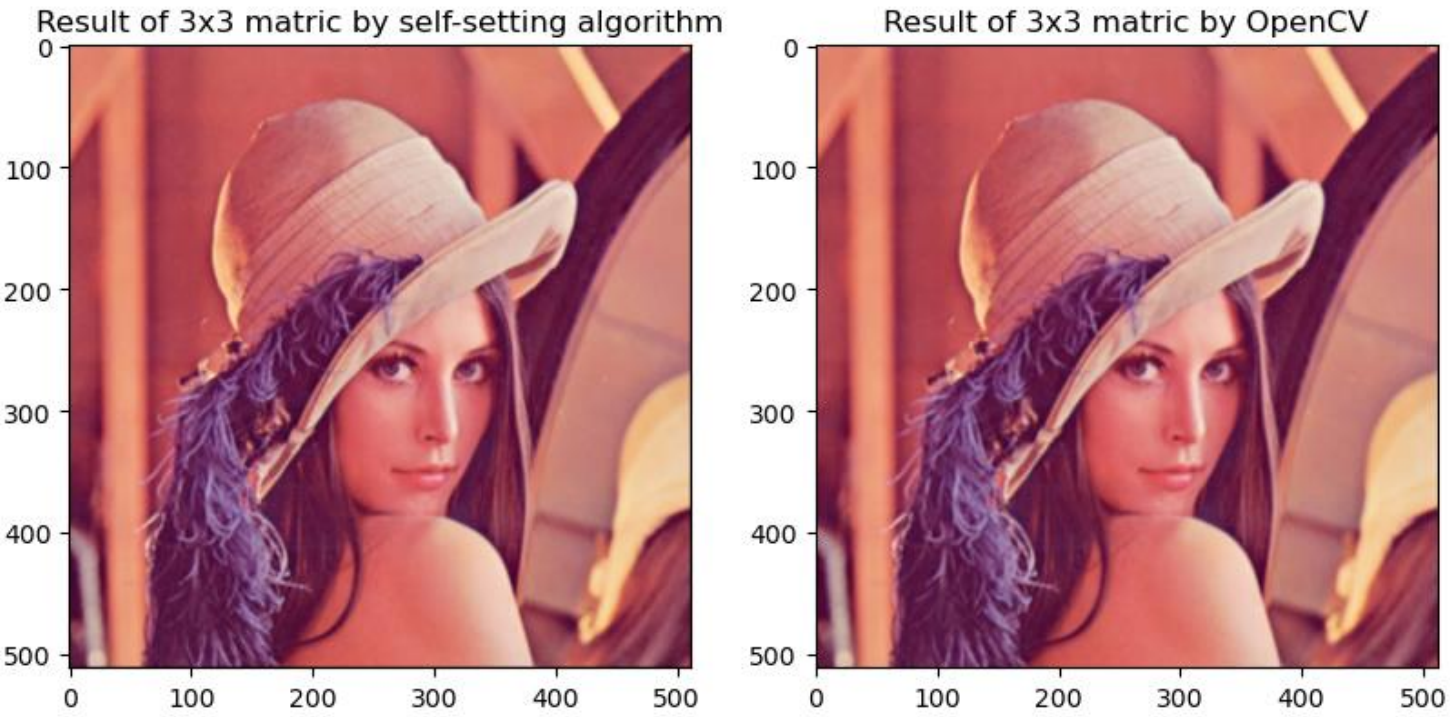
3.4 Image Rotation



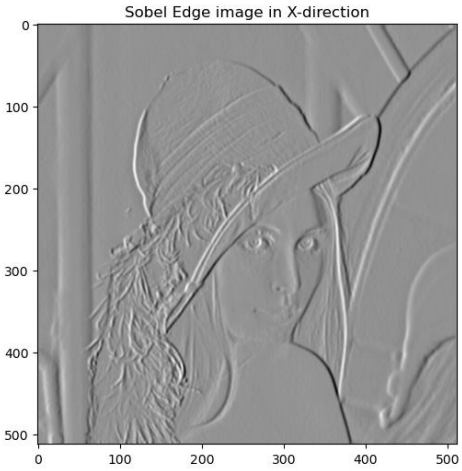
3.5 Algorithm to smooth the image

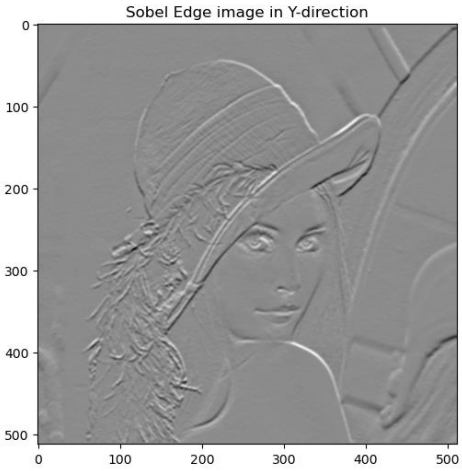
- By OpenCV library

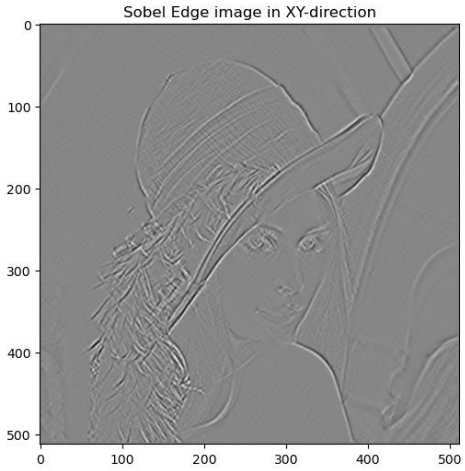
- Compare 2 results



3.6 Edge detection by Sobel Edge Detection

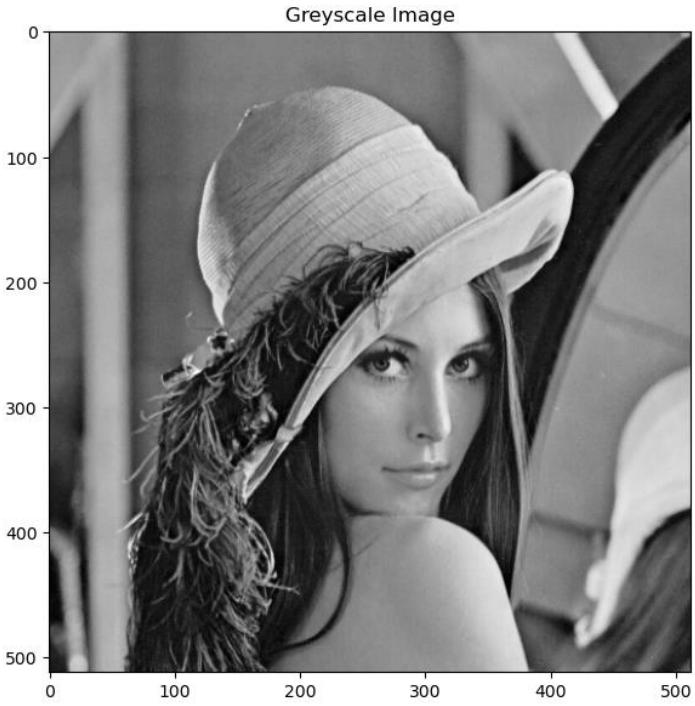


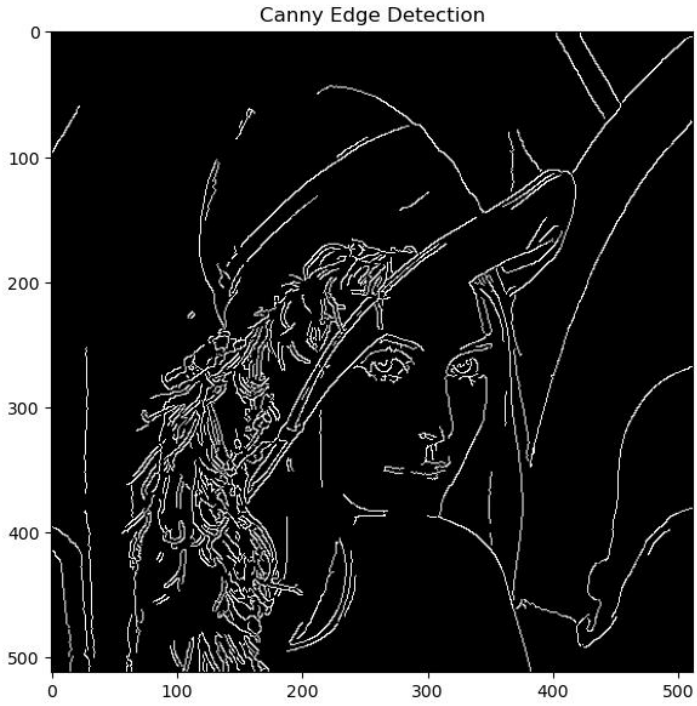




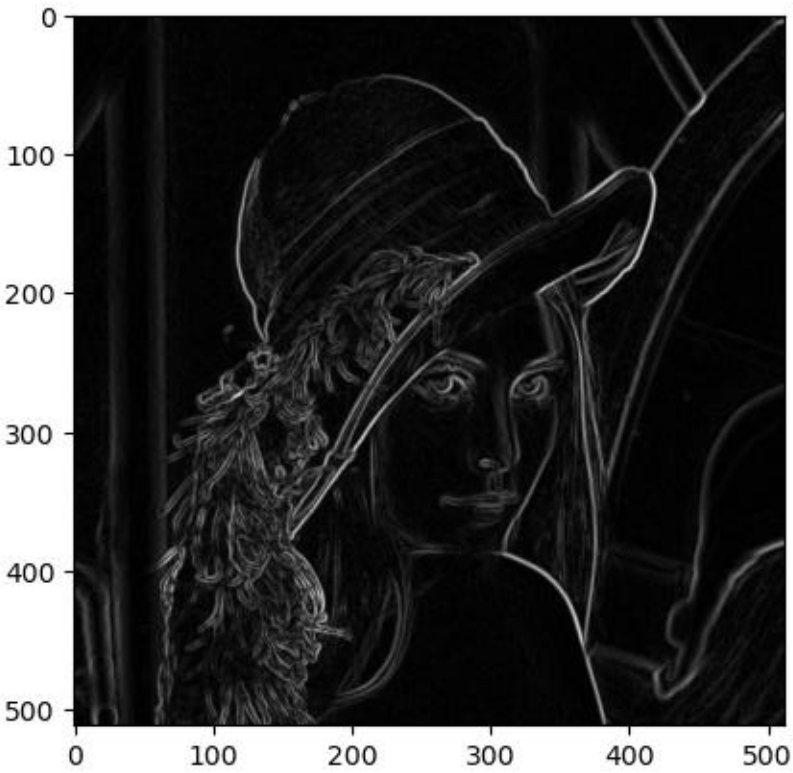
3.7 Edge detection by Canny Edge Detection

- By OpenCV library

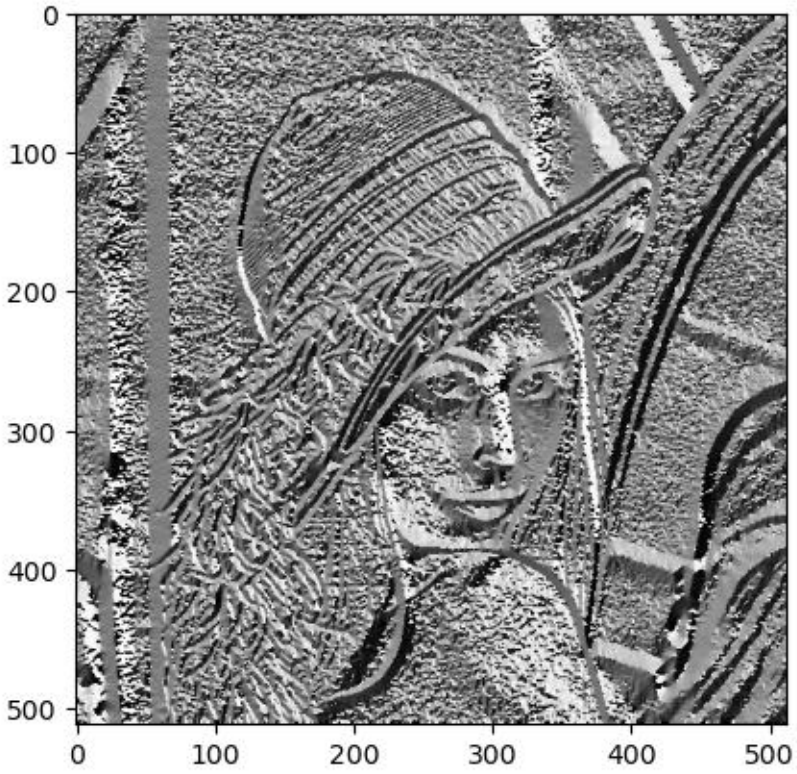




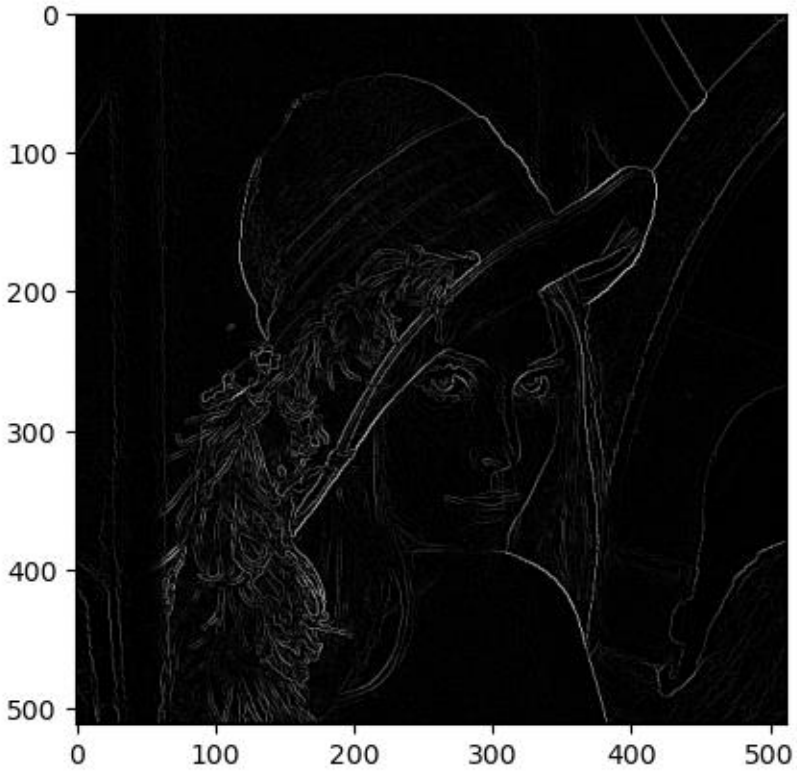
- By self-creating algorithm
- Compute edge strength:



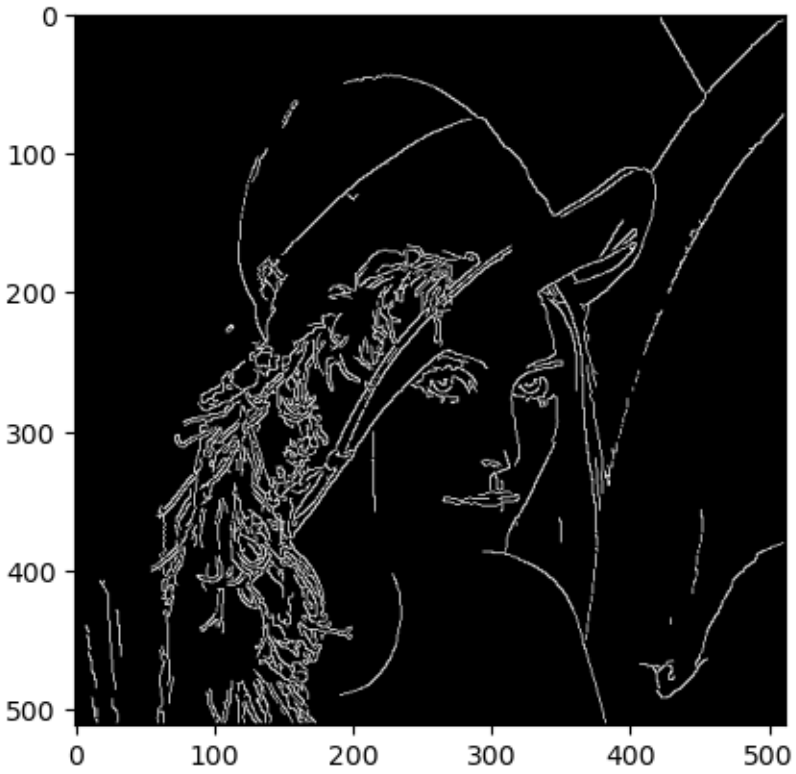
- Compute direction of gradient:



- Non maximal suppression:



- Hysteresis Thresholding:


0
100
200
300
400
500
0
100
200
300
400
500

4

Nhận xét kết quả demo chạy chương trình

4.1 Đánh giá kết quả

STT	Các thao tác xử lý ảnh	Chức năng	Mức độ hoàn thành
1	*Transform color*	RGB to Grayscale	100%
2		RGB to HSV	100%
3	*Transform geometry*	Reduce quality of a RGB image	100%
4		Image Rotation	100%
5	*Image blurring, Image smoothing*	Smoothing Image	100%
6	*Edge detection algorithm*	Sobel Edge Detection	100%
7		Canny Edge Detection	100%

5

Tài liệu tham khảo

1/ https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_rgb_to_hsv.html

2/ https://docs.opencv.org/3.4/d4/d61/tutorial_warp_affine.html

3/ https://viblo.asia/p/part1-edge-detection-voi-opencv-L4x5xLVB5BM

4/ https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html

5/ https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

6/ https://learnopencv.com/edge-detection-using-opencv/

7/ https://github.com/cynicphoenix/Canny-Edge-Detector

8/ https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

9/ https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Gradient_Sobel_Laplacian_Derivatives_Edge_Detection.php

10/ https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

11/ https://github.com/FienSoP/canny_edge_detector

12/ https://en.wikipedia.org/wiki/Canny_edge_detector

13/ https://en.wikipedia.org/wiki/Sobel_operator

14/ https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection