TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO LAB 02

XỬ LÝ ẢNH SỐ VÀ VIDEO SỐ

< EDGE DETECTION - PHÁT HIỆN CẠNH>

Lâm Thanh Ngọc - 21127118
Lớp: 21TGMT

Giảng viên:

Phạm Minh Hoàng

Nguyễn Mạnh Hùng

Lý Quốc Ngọc

Ngày 25 tháng 11 năm 2023

Mục lục

Đánh giá mức độ hoàn thành các chức năng

Chức năng	Mức độ hoàn thành	Đánh giá
Phát hiện cạnh sử dụng Gradient	100%	Hoàn thành
Phát hiện cạnh sử dụng Laplace	100%	Hoàn thành
Phát hiện cạnh sử dụng Laplace với Gaussian	100%	Hoàn thành
Phát hiện cạnh sử dụng Laplace bằng phương pháp Canny	100%	Hoàn thành

Mô tả các hàm

1 Các thư viện và hàm hỗ trợ

1.1 Thư viện

- `NumPy`: xử lý mảng và ma trận
- `SciPy` (sử dụng submodule signal): xử lý tín hiệu trong mảng
- `Matplotlib` (sử dụng module Pyplot): hiển thị ảnh
- `OpenCV`: gọi các hàm xử lý ảnh

1.2 Hàm hỗ trợ

`print_img(imgs)`

- Ý tưởng: Hiển thị các hình ảnh cùng lúc với mảng các ảnh cần in.
- Mô tả: Hàm print_img(imgs) hiển thị các ảnh trong mảng imgs theo dạng lưới với kích thước 1 x 3. Sau khi điều chỉnh kích thước mỗi ảnh hiển thị là 20x15 bằng hàm plt.figure(), từng hình ảnh trong mảng danh sách các hình imgs được vẽ vào 1 subplot với chỉ số tương ứng bằng 2 hàm plt.subplot() và plt.imshow() và lưới hình ảnh được hiển thị bằng hàm plt.show().
- Input:
  + imgs: mảng chứa các hình ảnh cần in.
- Output: Các hình ảnh từ mảng imgs được in ra ở dạng lưới (1x3)

2 Các thuật toán phát hiện cạnh

Để đảm bảo rằng các giá trị điểm ảnh không vượt quá phạm vi cho phép của kiểu dữ liệu uint8, các hàm sẽ sử dụng hàm np.clip để giới hạn các giá trị trong khoảng từ 0 đến 255, và hàm astype được dùng để chuyển đổi kiểu dữ liệu của điểm ảnh về uint8 để đảm bảo ảnh trả về có cùng kiểu dữ liệu với ảnh gốc.

2.1 Phát hiện cạnh sử dụng Gradient

`Sobel_ope(img_d)`

- Ý tưởng: Tính tích chập ma trận ảnh với kernel Sobel ứng với hai trục x, y và tìm độ lớn của Gradient.
- Mô tả: Hàm Sobel_ope(img_d) nhận vào một ảnh img_d dưới dạng mảng 2 chiều numpy. Ma trận ảnh được tính tích chập bằng kỹ thuật convolution với kernel Sobel được xác định cho x, y tương ứng như sau:

$$sobel\_kernel\_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$sobel\_kernel\_y = \begin{pmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

  Kết quả được xác định bởi mảng các giá trị độ lớn của Gradient với công thức:

$$\sqrt{sobelx^2 + sobely^2}$$

  Với sobelx, sobely lần lượt là kết quả tích chập của ma trận ảnh với các kernel Sobel.

- Input: img_d: ma trận numpy chứa các giá trị pixels của hình ảnh.
- Output: Ma trận các giá trị độ lớn của Gradient.

2.2 Phát hiện cạnh sử dụng Laplace

`Laplace(img_d)`

- Ý tưởng: Tính tích chập ma trận ảnh với kernel Laplace.
- Mô tả: Hàm Laplace(img_d) nhận vào một ảnh img_d dưới dạng mảng 2 chiều numpy. Ma trận ảnh được tính tích chập bằng kỹ thuật convolution với

kernel Laplace được xác định như sau:

$$laplace\_kernel = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

- Input: img_d: ma trận numpy chứa các giá trị pixels của hình ảnh.
- Output: Ma trận kết quả tích chập của ma trận ảnh với kernel Laplace.

2.3 Phát hiện cạnh sử dụng Laplace với Gaussian

`Laplace_Gaussian(img_d)`

- Ý tưởng: Tính tích chập ma trận ảnh đã được xử lý bởi hàm Gaussian_blur(img_d) với kernel Laplace.
- Mô tả:
  + Hàm Gaussian_blur(img_d) nhận vào một ảnh img_d dưới dạng mảng 2 chiều numpy. Ma trận ảnh được tính tích chập bằng kỹ thuật convolution với kernel Gaussian được xác định như sau:

$$gaussian\_kernel = \frac{1}{16} * \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

  + Hàm Laplace_Gaussian(img_d) nhận vào một ảnh img_d dưới dạng mảng 2 chiều numpy. Ma trận ảnh sau khi được xử lý bởi hàm Gaussian_blur(img_d) sẽ được tính tích chập với kernel Laplace như được xác định ở mục 2.2.
- Input: img_d: ma trận numpy chứa các giá trị pixels của hình ảnh.
- Output: Ma trận kết quả tích chập của ma trận ảnh đã được xử lý bởi hàm Gaussian_blur(img_d) với kernel Laplace.

2.4 Phát hiện cạnh sử dụng Laplace bằng phương pháp Canny

`Canny_Method(img_d)`

- Ý tưởng: Sử dụng thuật toán với 5 quy trình để phát hiện các cạnh của ảnh [2]:
  + Giảm nhiễu bằng Gaussian
  + Tính độ lớn Gradient bằng Sobel

+ Tìm giá trị lớn nhất cục bộ bằng Non-maximum supression
+ Phân ngưỡng kép (Double threshold)
+ Phát hiện cạnh bằng độ trễ (Hysteresis)

- Mô tả:
  + Giảm nhiễu bằng Gaussian: Sử dụng kernel Gaussian được xác định ở mục 2.3 để làm mịn ảnh và khử nhiễu[1].
  + Tính độ lớn Gradient bằng Sobel: Thực hiện tương tự hàm Sobel_ope(img_d) ở mục 2.1 và bổ sung hướng của Gradient bằng công thức: $arctan2(sobely, sobelx)$ [1].
  + Tìm giá trị lớn nhất cục bộ bằng Non-maximum supression: Sau khi chuyển đổi hướng của Gradient từ đơn vị radian sang đơn vị độ (degree) và giới hạn các góc lớn hơn 0, xét từng pixel theo chiều kim đồng hồ từ cột dọc và tìm giá trị lớn nhất trong các giá trị cùng hướng với pixel đang xét [1]. Giá trị 22.5 được sử dụng như sai số độ dày của cạnh.
  + Phân ngưỡng kép (Double threshold): Xét 2 giá trị threshold:

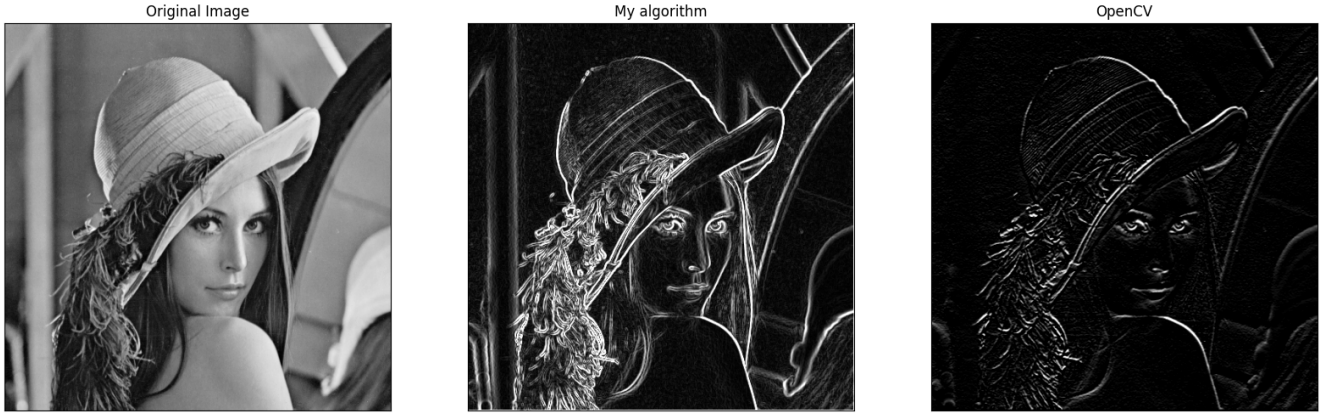
$$high\_threshold = 200$$

$$low\_threshold = 100$$

    Phân loại pixel đang xét trong mảng giá trị cục bộ lớn nhất thành 2 loại: nếu giá trị pixel đang xét lớn hơn high_threshold, giá trị pixel ảnh kết quả được gán giá trị "strong" bằng 255, ngược lại, giá trị pixel ảnh kết quả được gán giá trị "weak" bằng 25 [1].
  + Phát hiện cạnh bằng độ trễ (Hysteresis): Dựa vào kết quả phân ngưỡng kép xét các pixel mang giá trị "weak", nếu tồn tại 1 pixel trong lân cận 8 mang giá trị "strong" thì gán giá trị "strong" cho pixel đang xét, ngược lại thì gán pixel đang xét bằng 0 [1].
- Input: img_d: ma trận numpy chứa các giá trị pixels của hình ảnh.
- Output: Ma trận ảnh kết quả sau khi được xử lý bởi bước "Hysteresis".

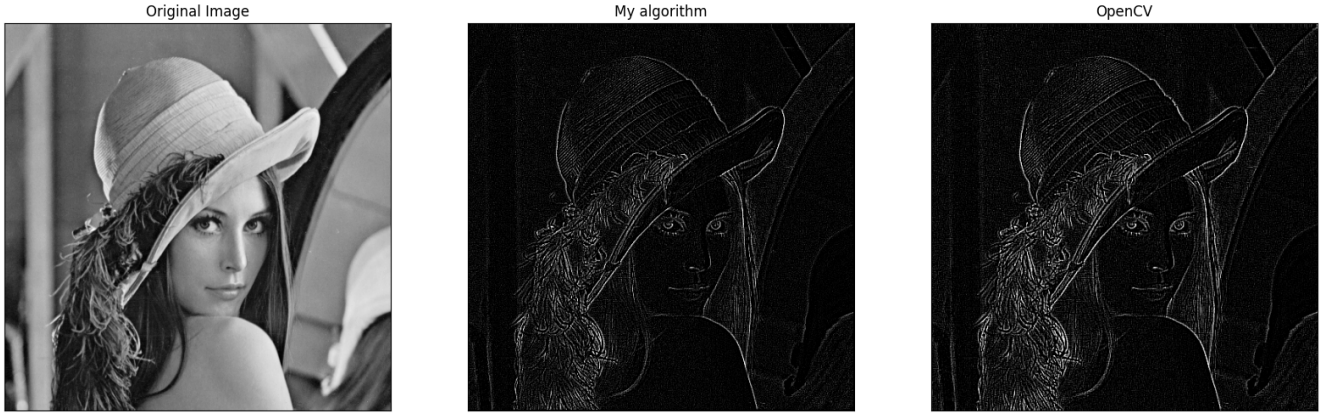
3 So sánh kết quả với các hàm của thư viện OpenCV

3.1 Phát hiện cạnh sử dụng Gradient



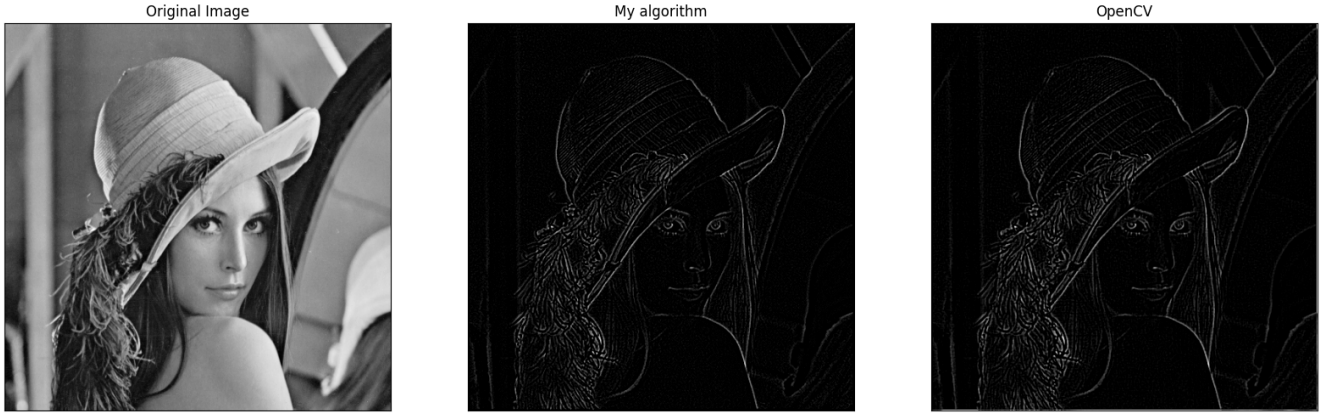
Hàm Sobel_ope(img_d) cho ảnh đầu ra có số cạnh được phát hiện nhiều và rõ nét hơn so với khi sử dụng hàm có sẵn của thư viện OpenCV với các tham số truyền vào: `cv2.Sobel(img_d, cv2.CV_8U, 0, 1, ksize=3)`.

3.2 Phát hiện cạnh sử dụng Laplace



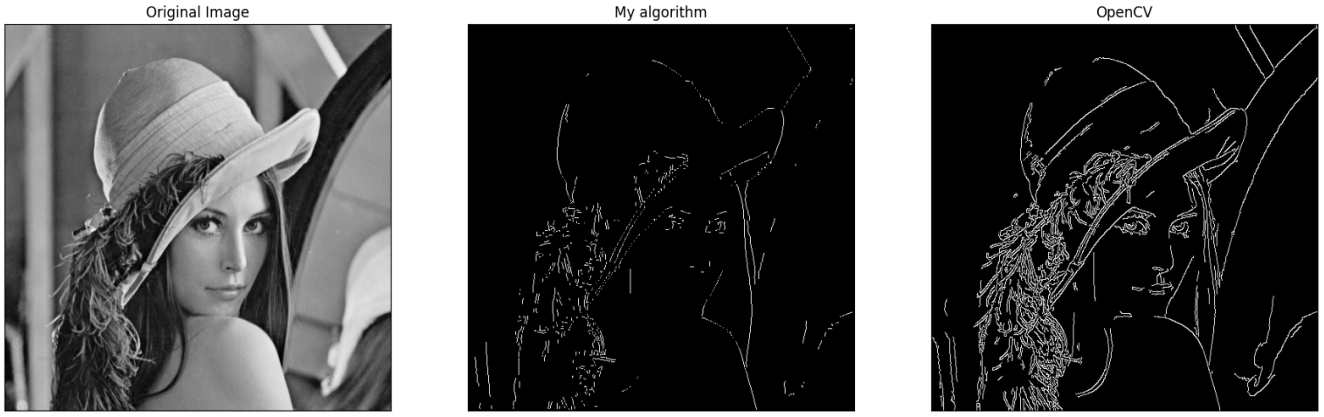
Hàm Laplace(img_d) cho ảnh đầu ra gần như tương đồng với hàm có sẵn của thư viện OpenCV với các tham số truyền vào: `cv2.Laplacian(img_d, cv2.CV_8U, ksize=3)`

3.3 Phát hiện cạnh sử dụng Laplace với Gaussian



Hàm Laplace_Gaussian(img_d) cho ảnh đầu ra gần như tương đồng với hàm có sẵn của thư viện OpenCV với các tham số truyền vào: `cv2.Laplacian(Gaussian_blur (img_d), cv2.CV_8U, ksize=3)`

3.4 Phát hiện cạnh sử dụng Laplace bằng phương pháp Canny



Hàm Canny_Method(img_d) cho ảnh đầu ra tuy vẫn phác hoạ được dáng tổng quan của ảnh gốc nhưng bị mất nhiều chi tiết. So với hàm Canny_Method(img_d), hàm có sẵn của thư viện OpenCV với các tham số truyền vào: `cv2.Canny(img_d, 100, 200)` cho ảnh đầu ra có nhiều chi tiết cụ thể với các đường nét rõ hơn.

Tài liệu tham khảo

[1] *Canny Method Steps*. Accessed: Nov. 25, 2023. URL: https://en.wikipedia.org/wiki/Canny_edge_detector.

[2] *Step Description*. Accessed: Nov. 25, 2023. URL: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123.