

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÁO CÁO MÔN HỌC
THỊ GIÁC MÁY TÍNH



**Lab 1: Xử lý ảnh cơ bản với OpenCV trên
WSL**

Sinh viên thực hiện:

Họ và tên: Lê Anh Duy
MSSV: 23127011

Giảng viên hướng dẫn:

Phạm Minh Hoàng
Phạm Thanh Tùng
Võ Hoài Việt

Thành phố Hồ Chí Minh, Ngày 14 tháng 2 năm 2026

Mục lục

1 Giới thiệu	2
2 Môi trường và cách chạy chương trình	2
2.1 Môi trường WSL và các gói phụ thuộc	2
2.2 Cách build	3
2.3 Cách chạy chương trình	3
3 Thiết kế chương trình	4
3.1 Cấu trúc thư mục	4
3.2 Cơ chế chọn thuật toán	4
4 Các thuật toán xử lý ảnh	4
4.1 Chuyển ảnh màu sang xám (rgb2gray)	5
4.2 Chính độ sáng (brightness)	5
4.3 Chính độ tương phản (contrast)	5
4.4 Lọc trung bình (avg)	5
4.5 Lọc trung vị (med)	6
4.6 Lọc Gauss (gau)	6
4.7 Phát hiện biên Sobel (sobel)	6
4.8 Phát hiện biên Laplace (laplace)	7
4.9 Phát hiện biên Laplace (laplace)	7
5 Kết quả thực nghiệm	7
5.1 Chuyển xám và làm mờ	8
5.2 Ảnh hưởng của độ sáng và tương phản	8
5.3 Lọc trung bình, trung vị và Gauss	9
5.4 Phát hiện biên Sobel và Laplace	9
6 Kết luận	9

1 Giới thiệu

Mục tiêu của bài Lab 1 là làm quen với các phép xử lý ảnh cơ bản sử dụng thư viện OpenCV mà không sử dụng các thư viện có sẵn, chỉ sử dụng các hàm đọc và ghi được cung cấp. Chương trình được cài đặt bằng C++17, biên dịch bằng g++ và sử dụng OpenCV để đọc, ghi và thao tác trên ảnh số. Các phép biến đổi được hiện thực bao gồm:

- Chuyển ảnh màu sang ảnh xám (rgb2gray).
- Chính độ sáng (brightness).
- Chính độ tương phản (contrast).
- Lọc trung bình (avg).
- Lọc trung vị (med).
- Lọc Gauss (gau).
- Phát hiện biên Sobel (sobel).
- Phát hiện biên Laplace (laplace).

Github: <https://github.com/Le-Anh-Duy/Computer-Vision-Lab-Lab1>

2 Môi trường và cách chạy chương trình

2.1 Môi trường WSL và các gói phụ thuộc

Chương trình được thiết kế để chạy trên WSL (Ubuntu). Các gói cần cài đặt:

- build-essential: cung cấp g++, make và các công cụ build cơ bản.
- pkg-config: dùng để truy vấn cflags/libs của OpenCV.
- libopencv-dev: thư viện và header OpenCV (opencv2/...).

Lệnh cài đặt trong Ubuntu (WSL):

```
sudo apt update  
sudo apt install build-essential pkg-config libopencv-dev
```

2.2 Cách build

Từ WSL, di chuyển đến thư mục project (ví dụ):

```
cd /mnt/d/Coding/School/Y3-K2/TGMT/Cpp/Lab1  
make
```

Sau khi build thành công, file thực thi sẽ nằm tại:

```
bin/main
```

Để dọn các file sinh ra trong quá trình biên dịch:

```
make clean
```

2.3 Cách chạy chương trình

Chương trình nhận tham số dòng lệnh theo cú pháp:

```
./bin/main <command> <input_path> <output_path> [param]
```

Trong đó:

- <command>: chuỗi lệnh chọn thuật toán (ví dụ -rgb2gray).
- <input_path>: đường dẫn ảnh đầu vào.
- <output_path>: đường dẫn lưu ảnh kết quả.
- [param]: tham số phụ tùy thuật toán (kích thước kernel, hệ số độ sáng, độ tương phản, ...).

Ví dụ chạy trên WSL với đường dẫn ảnh dạng /mnt/d/...:

```
./bin/main -rgb2gray lena.png output/lena_gray.png  
./bin/main -brightness lena.png output/lena_brightness_50.png 50  
./bin/main -contrast lena.png output/lena_contrast_1.5.png 1.5  
./bin/main -avg lena.png output/lena_avg_5.png 5
```

3 Thiết kế chương trình

3.1 Cấu trúc thư mục

Project được tổ chức với các thư mục và file chính như sau:

- **Makefile**: file build chính, dùng pkg-config để link OpenCV.
- **src/**: mã nguồn C++
 - **main.cpp**: điểm vào chương trình, đọc tham số dòng lệnh, gọi hàm xử lý ảnh tương ứng.
 - **functions.cpp**: ảnh xạ chuỗi lệnh (ví dụ `-rgb2gray`) sang con trả hàm xử lý.
 - **rgb2gray.cpp, brightness.cpp, contrast.cpp, avg.cpp, med.cpp, gau.cpp, sobel.cpp, laplace.cpp**: hiện thực các thuật toán xử lý ảnh cụ thể.
 - **convolution.cpp**: hiện thực hàm tích chập tổng quát `apply_filter`.
- **include/**: các file header tương ứng, khai báo hàm và kiểu dữ liệu.
- **bin/**: chứa file thực thi sau khi build.
- **output/**: chứa các ảnh kết quả sau khi chạy chương trình.

3.2 Cơ chế chọn thuật toán

File `functions.cpp` định nghĩa hai vector:

- **commands_list**: danh sách các chuỗi lệnh như `-rgb2gray`, `-contrast`, ...
- **functions_list**: danh sách các con trả hàm tương ứng kiểu `ImageFunc`.

Hàm `get_cv_func` duyệt qua danh sách này và trả về con trả hàm ứng với cmd. Trong `main.cpp`, chương trình đọc `argv[1]` làm lệnh, `argv[2]` là ảnh đầu vào, `argv[3]` là ảnh đầu ra và `argv[4]` (nếu có) làm tham số chuỗi cho thuật toán.

4 Các thuật toán xử lý ảnh

Lưu ý chung: Trong tất cả các thuật toán sử dụng cửa sổ trượt (sliding window) hay kernel dưới đây, kích thước kernel $k \times k$ **bắt buộc phải là số lẻ** (ví dụ: $3 \times 3, 5 \times 5$) để xác định chính xác điểm tâm (anchor point) của mặt nạ. Ngoài ra, để giữ nguyên kích thước ảnh sau khi tích chập, ảnh đầu vào được xử lý biên (padding) bằng phương pháp điền số 0 (Zero Padding).

4.1 Chuyển ảnh màu sang xám (rgb2gray)

Mục đích: Chuyển ảnh màu BGR sang ảnh xám một kênh (grayscale) để phục vụ các bước xử lý tiếp theo.

Ý tưởng chính: Với mỗi pixel, sử dụng công thức độ chói (Luminance) theo chuẩn Rec. 601¹:

$$Y = 0,299R + 0,587G + 0,114B \quad (1)$$

Giá trị Y được làm tròn và gán cho ảnh kết quả thuộc kiểu CV_8UC1 (ảnh đơn kênh, 8-bit không dấu).

4.2 Chỉnh độ sáng (brightness)

Mục đích: Tăng hoặc giảm độ sáng toàn ảnh.

Ý tưởng chính: Cộng một hằng số $bias$ vào giá trị pixel. Để tối ưu tốc độ, thuật toán sử dụng bảng tra (Look-up Table - LUT).

$$I_{out} = \text{clamp}(I_{in} + bias, 0, 255) \quad (2)$$

Chi tiết về thay đổi độ sáng có thể tham khảo tại OpenCV Documentation².

4.3 Chỉnh độ tương phản (contrast)

Mục đích: Thay đổi độ tương phản giúp ảnh sắc nét hơn hoặc dịu đi.

Ý tưởng chính: Sử dụng công thức tuyến tính quanh mức xám trung bình 128:

$$I_{out} = \text{clamp}(\alpha(I_{in} - 128) + 128, 0, 255) \quad (3)$$

Trong đó α là hệ số tương phản ($\alpha > 1$: tăng tương phản).

4.4 Lọc trung bình (avg)

Mục đích: Làm mờ ảnh, giảm nhiễu hạt.

Ý tưởng chính: Thay thế giá trị pixel trung tâm bằng trung bình cộng của các pixel trong cửa sổ $k \times k$ (với k là số lẻ). Kernel tích chập H có dạng:

$$H = \frac{1}{k^2} \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix} \quad (4)$$

¹Tham khảo công thức Grayscale: <https://en.wikipedia.org/wiki/Grayscale>

²https://docs.opencv.org/4.x/d3/dc1/tutorial_basic_linear_transform.html

4.5 Lọc trung vị (med)

Mục đích: Khử nhiễu muối tiêu (salt-and-pepper noise) hiệu quả mà không làm mờ biên quá mức.

Ý tưởng chính: Với mỗi cửa sổ $k \times k$ (k lẻ):

1. Thu thập tất cả giá trị pixel trong cửa sổ vào một mảng.
2. Sắp xếp mảng theo thứ tự tăng dần.
3. Chọn giá trị ở giữa (median) làm giá trị mới cho pixel trung tâm³.

4.6 Lọc Gauss (gau)

Mục đích: Làm mờ ảnh một cách tự nhiên hơn lọc trung bình, dùng để giảm nhiễu Gaussian.

Ý tưởng chính: Sử dụng kernel $k \times k$ (k lẻ) với các trọng số tuân theo phân phối Gauss 2D:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (5)$$

Sau khi tính toán, kernel được chuẩn hóa (normalize) sao cho tổng các phần tử bằng 1 để không làm thay đổi độ sáng trung bình của ảnh⁴.

4.7 Phát hiện biên Sobel (sobel)

Mục đích: Tìm biên của vật thể dựa trên sự thay đổi cường độ sáng. Kết quả là một ảnh xám thể hiện độ lớn của gradient.

Ý tưởng chính:

- Dùng 2 kernel 3×3 (số lẻ) để tính đạo hàm theo phương x (G_x) và phương y (G_y).
- **Lưu ý cài đặt:** Các giá trị đạo hàm có thể âm, do đó cần tính toán trên kiểu dữ liệu có dấu (VD: float hoặc short) sau đó lấy giá trị tuyệt đối.
- Độ lớn Gradient tổng hợp tại mỗi điểm⁵:

$$G = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y| \quad (6)$$

- Kết quả cuối cùng được chuyển về đoạn $[0, 255]$ (ảnh grayscale).

³Median Filter: https://en.wikipedia.org/wiki/Median_filter

⁴Gaussian Blur: https://en.wikipedia.org/wiki/Gaussian_blur

⁵Sobel Operator: https://en.wikipedia.org/wiki/Sobel_operator

4.8 Phát hiện biên Laplace (`laplace`)

Mục đích: Phát hiện biên dựa trên đạo hàm bậc hai, nhạy cảm với các thay đổi đột ngột. Output là ảnh đen trắng.

Ý tưởng chính:

- Áp dụng kernel Laplace 3×3 (thường có giá trị tâm là 4 hoặc 8, xung quanh là -1).
- Tương tự Sobel, kết quả tích chập có thể âm. Cần lấy **giá trị tuyệt đối** hoặc chuẩn hoá để hiển thị được trên ảnh xám 8-bit.

Tham khảo Laplace Operator tại OpenCV Docs⁶.

4.9 Phát hiện biên Laplace (`laplace`)

Mục đích: làm nổi bật các vùng có biến thiên cường độ theo mọi hướng (biên).

Ý tưởng chính:

- Chuyển ảnh sang xám và làm mờ nhẹ bằng lọc Gauss 3×3 .
- Áp dụng kernel Laplace 3×3 với trọng số trung tâm âm lớn và các ô xung quanh dương.
- Sử dụng `apply_filter` để tính tích chập và thu được ảnh biên.

5 Kết quả thực nghiệm

Trong phần này, ta sử dụng ảnh chuẩn `lena.png` làm ảnh đầu vào, sau đó áp dụng các thuật toán để quan sát trực quan hiệu ứng.

⁶https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html

5.1 Chuyển xám và làm mờ



Hình 1: Kết quả chuyển xám và làm mờ bằng lọc Gauss.

5.2 Ảnh hưởng của độ sáng và tương phản



Hình 2: So sánh các mức độ sáng và tương phản khác nhau.

5.3 Lọc trung bình, trung vị và Gauss



Hình 3: So sánh các bộ lọc làm mờ/khử nhiễu với kernel 5×5 .

5.4 Phát hiện biên Sobel và Laplace



Hình 4: Kết quả phát hiện biên với Sobel và Laplace.

6 Kết luận

Trong bài Lab này, em đã:

- Thiết lập môi trường lập trình C++ với OpenCV trên WSL.

- Xây dựng chương trình dòng lệnh có khả năng áp dụng nhiều phép biến đổi ảnh khác nhau thông qua tham số command.
- Hiện thực và thử nghiệm các thuật toán xử lý ảnh cơ bản: chuyển xám, chỉnh độ sáng/tương phản, các bộ lọc trung bình, trung vị, Gauss và các toán tử phát hiện biên Sobel, Laplace.
- Quan sát, so sánh trực quan tác động của từng thuật toán lên ảnh chuẩnlena.png.

Qua đó, em hiểu rõ hơn về cách ánh xạ các công thức xử lý tín hiệu rời rạc vào cài đặt C++, cũng như cách tổ chức project, viết Makefile và sử dụng OpenCV trong thực tế.