ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO LAB 01

Môn: Thị giác máy tính - CSC16004

Lớp: 21_22

Tên: Lê Nguyễn

Mã số sinh viên: 21120511

Thành phố Hồ Chí Minh - 2024

Mục lục

1 Hướng dẫn sử dụng

Hướng dẫn được thực hiện trên hệ điều hành **Ubuntu 22.04.4 LTS** [1].

1.1 Build source code

- Đầu tiên cài đặt những package quan trọng (gcc, g++, gdb, ...) và cmake (công cụ build chính):

```
$ sudo apt install build-essential gdb cmake
```

- Tiếp theo cài đặt thư viện OpenCV:

```
$ sudo apt install libopencv-dev
```

- Trong thư mục Sources (gồm source code chính và file CMakeLists.txt), tạo thư mục build và đi vào thư mục build [2]:

```
$ mkdir build
$ cd build
```

- Sau đó thực hiện build source code bằng:

```
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
$ make
```

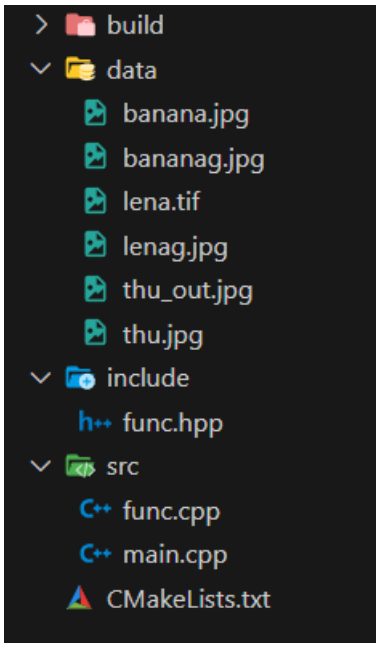
- Khi đã build source code thành công, thì có thể chạy bằng command sau:

```
$ ./21120511 -<command> <InputPath> <OutputPath> <Value (optional)>
```

Lưu ý: đường dẫn tới ảnh (hoặc bất cứ file nào trong thư mục Sources) được hiểu là relative path. Giả sử muốn dùng ảnh thu.jpg trong thư mục data làm ảnh input và ảnh thu_out.jpg làm ảnh output, ta có thể dùng command như sau:

```
$ ./21120511 -<command> data/thu.jpg
                        data/thu_out.jpg
                        <Value>

// Do giới hạn chỗ viết nên em xuống dòng
// nhưng khi gõ command thì ta gõ
// luôn một dòng
```



Hình 1: Cấu trúc của thư mục Sources

[1] Ubuntu này được em chạy trên WSL2 nhưng em nghĩ vẫn có thể áp dụng lên Ubuntu chạy trên máy thật.
[2] Mỗi một dấu `$` được hiểu là một command, hai dấu tức là thực hiện command phía trên xong rồi đến command phía dưới.

1.2 Chạy bằng file executable

Trong folder nộp bài sẽ gồm 4 folder nhỏ:

- **Document**: chứa báo cáo.
- **Sources**: chứa source code, có thể build và chạy như phía trên.
- **Data**: bao gồm ảnh input và các ảnh output (nếu muốn chạy bằng cách trên, hãy copy các ảnh này vào thư mục Sources).
- **Executable**: bao gồm file executable có tên là 21120511.

Để chạy bằng file executable, đầu tiên cho phép quyền chạy file bằng:

```
$ cd Executable
$ chmod +x 21120511
```

Sau đó có thể thực hiện chạy file bằng command như phần trên. Lưu ý, vị trí đường dẫn ảnh lúc này không còn là relative path. Giả sử muốn input ảnh sekiro.jpg nằm trong folder **Data** và chuyển thành ảnh xám với output là ảnh sekiro_out.jpg nằm trong folder hiện tại thì ta nhập như sau:

```
$ ./21120511 -rgb2gray ../Data/sekiro.jpg sekiro_out.jpg
```

2 Giải thích chi tiết

Hình 2: Ảnh được dùng để chạy code

Quy ước:

- Ảnh đầu vào được kí hiệu là f, mỗi điểm ảnh tại (x, y) được kí hiệu là $f(x, y)$.
- Nếu f là một ảnh xám, thì ta hiểu mỗi điểm ảnh $f(x, y)$ là một giá trị thuộc $\mathbb{R}$.
- Nếu f là một ảnh màu, thì ta hiểu mỗi điểm ảnh $f(x, y)$ là một vector:

$$f(x,y) = \begin{bmatrix} R(x,y) & B(x,y) & G(x,y) \end{bmatrix}^T$$

  với $R(x, y), B(x, y), G(x, y) \in \mathbb{R}$ lần lượt là các giá trị màu đỏ, xanh dương và xanh lá của điểm ảnh $f(x, y)$.

- Ảnh đầu ra được kí hiệu là g và được ghi rõ là loại ảnh gì. Nếu không ghi rõ loại ảnh thì dựa vào f, f là ảnh màu thì g cũng là ảnh màu, tương tự với f là ảnh xám.
- Các phép toán $+, \cdot$ trên vector $f(x, y)$ sẽ được hiểu là các phép toán trên từng phần tử. Ví dụ:

$$f(x,y) + 3 = \begin{bmatrix} R(x,y) + 3 \\ B(x,y) + 3 \\ G(x,y) + 3 \end{bmatrix}$$

2.1 Chuyển ảnh về ảnh xám

Hình 3: Ảnh 2 sau khi được chuyển thành ảnh xám.

Command:

```
$ ./21120511 -rgb2gray data/sekiro.jpg data/sekiro_gray.jpg
```

Để tính ảnh xám của ảnh đầu vào f. Ta duyệt từng điểm ảnh $f(x, y)$ và gán vào ảnh đầu ra g với g là ảnh xám bằng công thức sau:

$$g(x, y) = 0.11 \cdot R(x, y) + 0.59 \cdot B(x, y) + 0.3 \cdot G(x, y)$$

2.2 Điều chỉnh độ sáng

Hình 4: Ảnh 2 khi tăng giá trị độ sáng thêm 80

Command:

```
$ ./21120511 -brightness data/sekiro.jpg data/sekiro_brightness_80.jpg 80
```

Để điều chỉnh độ sáng của ảnh đầu vào f theo một giá trị $\beta \in \mathbb{Z}$. Ta duyệt từng điểm $f(x, y)$ và gán vào ảnh đầu ra g bằng công thức sau:

$$g(x, y) = f(x, y) + \beta$$

2.3 Điều chỉnh tương phản

Hình 5: Ảnh 2 khi tăng độ tương phản lên 1.5

Command:

```
$ ./21120511 -constrast data/sekiro.jpg data/sekiro_contrast_1_5.jpg 1.5
```

Để điều chỉnh độ tương phản của ảnh đầu vào f theo một giá trị $\alpha \in \mathbb{R}^+$. Ta duyệt từng điểm $f(x, y)$ và gán vào ảnh đầu ra g bằng công thức sau:

$$g(x, y) = \alpha \cdot f(x, y)$$

2.4 Lọc trung bình (Average Filter)

Hình 6: Ảnh 2 khi lọc trung bình bằng kernel kích thước 3×3

Command:

```
$ ./21120511 -avg data/sekiro.jpg data/sekiro_avg_3.jpg 3
```

Hình 7: Ảnh 2 khi lọc trung bình bằng kernel kích thước 5×5

Command:

```
$ ./21120511 -avg data/sekiro.jpg data/sekiro_avg_5.jpg 5
```

- Đầu tiên ta tạo kernel có kích thước $k \times k$ cho việc lọc trung bình, gọi là AVG:

$$\text{AVG} = \frac{1}{k} \begin{bmatrix} 1 & ... & 1 \\ \vdots & \ddots & \vdots \\ 1 & ... & 1 \end{bmatrix}_{k \times k}$$

- *Lưu ý*: khác với ma trận thông thường, một kernel có kích thước $k \times k$ sẽ có index từ $\lfloor -k/2 \rfloor, ..., 0, ..., \lfloor k/2 \rfloor$ thay vì $0, ..., k-1$.

- Sau đó duyệt từng điểm ảnh $f(x, y)$, thực hiện tích chập AVG với một cửa sổ có trung tâm là (x, y) và có kích thước là $k \times k$.

$$g(x, y) = (f * AVG)(x, y)$$

  với $*$ được định nghĩa là phép tích chập, đồng thời $*$ là một phép toán có thể áp dụng lên từng phần tử của vector.

- *Lưu ý*: để dễ hơn, em bỏ đi các phần tử nằm tại biên, túc là tại các phần tử đó, không tồn tại cửa sổ mà phần tử đó nằm ở trung tâm và có kích thước $k \times k$. Vì vậy ảnh đầu ra có viền đen xung quanh.

- Dưới đây là ví dụ của phép tích chập tại điểm ảnh $g(2, 2)$ với kernel h, kết quả đưa vào điểm ảnh $f(2, 2)$.

$f_{1,1}$	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{1,5}$
$f_{2,1}$	$f_{2,2}$	$f_{2,3}$	$f_{2,4}$	$f_{2,5}$
$f_{3,1}$	$f_{3,2}$	$f_{3,3}$	$f_{3,4}$	$f_{3,5}$
$f_{4,1}$	$f_{4,2}$	$f_{4,3}$	$f_{4,4}$	$f_{4,5}$
$f_{5,1}$	$f_{5,2}$	$f_{5,3}$	$f_{5,4}$	$f_{5,5}$

$=$

$g_{1,1}$	$g_{1,2}$	$g_{1,3}$	$g_{1,4}$	$g_{1,5}$
$g_{2,1}$	$g_{2,2}$	$g_{2,3}$	$g_{2,4}$	$g_{2,5}$
$g_{3,1}$	$g_{3,2}$	$g_{3,3}$	$g_{3,4}$	$g_{3,5}$
$g_{4,1}$	$g_{4,2}$	$g_{4,3}$	$g_{4,4}$	$g_{4,5}$
$g_{5,1}$	$g_{5,2}$	$g_{5,3}$	$g_{5,4}$	$g_{5,5}$

$*$

-1	0	$+1$	
$h_{-,-}$	$h_{-,0}$	$h_{-,+}$	-1
$h_{0,-}$	$h_{0,0}$	$h_{0,+}$	0
$h_{+,-}$	$h_{+,0}$	$h_{+,+}$	$+1$

$$f_{2,2} = g_{3,3}h_{-,-} + g_{3,2}h_{-,0} + g_{3,1}h_{-,+} + g_{2,3}h_{0,-} + g_{2,2}h_{0,0} + g_{2,1}h_{0,+} + g_{1,3}h_{+,-} + g_{1,2}h_{+,0} + g_{1,1}h_{+,+}$$

Hình 8: Ví dụ phép tích chập (nguồn: Convolution).

2.5 Lọc Gaussian (Gaussian Filter)

Hình 9: Ảnh 2 khi lọc gaussian bằng kernel kích thước 3×3

Command:

```
$ ./21120511 -gau data/sekiro.jpg data/sekiro_gau_3.jpg 3
```

Hình 10: Ảnh 2 khi lọc gaussian bằng kernel kích thước 5×5

Command:

```
$ ./21120511 -gau data/sekiro.jpg data/sekiro_gau_5.jpg 5
```

- Tương tự như lọc trung bình. Đầu tiên tạo kernel có kích thước $k \times k$ cho việc lọc Gaussian, gọi là GAU:

$$[\text{GAU}]_{xy} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

  trong đó $[\text{GAU}]_{xy}$ là giá trị của kernel tại vị trí (x, y) và σ là một tham số mà ta tự điều chỉnh để cho ra kết quả tốt nhất (trong code dùng $sigma = 1$).

- Ngoài ra, ta chuẩn hoá kernel để cho tổng các phần tử trong kernel bằng 1, do đó:

$$\text{GAU} = \frac{1}{\text{sum}(\text{GAU})}\text{GAU}$$

  với:

$$\text{sum}(\text{GAU}) = \sum_{x}^{k}\sum_{y}^{k}[\text{GAU}]_{xy}$$

- Sau đó, ta chỉ cần duyệt qua từng vị trí (x, y) và tích chập $f(x, y)$ với kernel GAU:

$$g(x, y) = (f * \text{GAU})(x, y)$$

2.6 Lọc trung vị (Median Filter)

Hình 11: Ảnh 2 khi lọc trung vị với $k = 3$

Command:

```
$ ./21120511 -med data/sekiro.jpg data/sekiro_med_3.jpg 3
```

Hình 12: Ảnh 2 khi lọc trung vị với $k = 5$

Command:

```
$ ./21120511 -med data/sekiro.jpg data/sekiro_med_5.jpg 5
```

- Khác với hai phương pháp lọc trên, ta sẽ không dùng kernel.
- Duyệt qua từng vị trí (x, y), ta tìm được một cửa sổ với (x, y) là trung tâm và có kích thước $k \times k$. Giả sử ta đang ở vị trí $(1, 1)$ và $k = 3$, tìm được cửa sổ W sau:

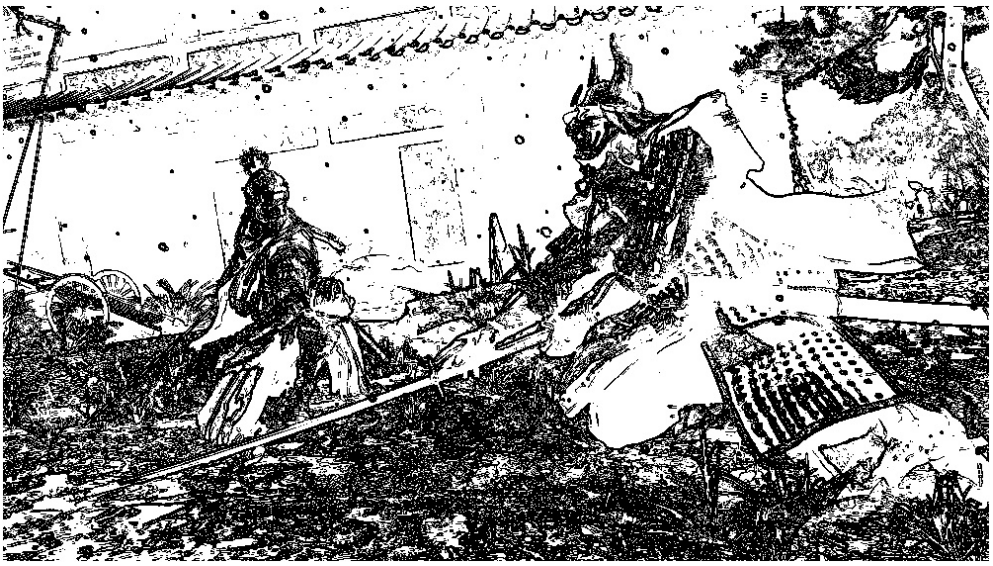
$$W = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) \\ f(1,0) & f(1,1) & f(1,2) \\ f(2,0) & f(2,1) & f(2,2) \end{bmatrix}$$

- Sau đó ta tìm trung vị của cửa sổ này, gọi là median(W) và gán $g(x, y) = \text{median}(W)$.
- Để tìm median(W), ta đưa W thành một vector, gọi là W':

$$W' = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & f(1,0) & f(1,1) & f(1,2) & f(2,0) & f(2,1) & f(2,2) \end{bmatrix}^T$$

- Sắp xếp W' theo thứ tự (tăng dần hoặc giảm dần) rồi sau đó chọn phần tử nằm tại vị trí chính giữa của W'. Lúc này phần tử chính giữa đó chính là median(W).

2.7 Phát hiện biên cạnh Sobel



Hình 13: Ảnh 2 khi được áp dụng phát hiện biên cạnh Sobel

Command:

```
$ ./21120511 -sobel data/sekiro.jpg data/sekiro_sobel.jpg
```

- Đầu tiên, ta phải đưa f về ảnh xám, gọi là f'.
- Tiếp theo, ta có hai kernel cho thuật toán phát hiện biên cạnh Sobel, gọi là sobelX và sobelY, ta có:

$$\text{sobelX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{và} \quad \text{sobelY} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Sau đó tại mỗi vị trí (x, y), lần lượt thực hiện tích chập giữa $f'(x, y)$ với sobelX để cho ra $G_x(x, y)$ và sobelY để cho ra $G_y(x, y)$:

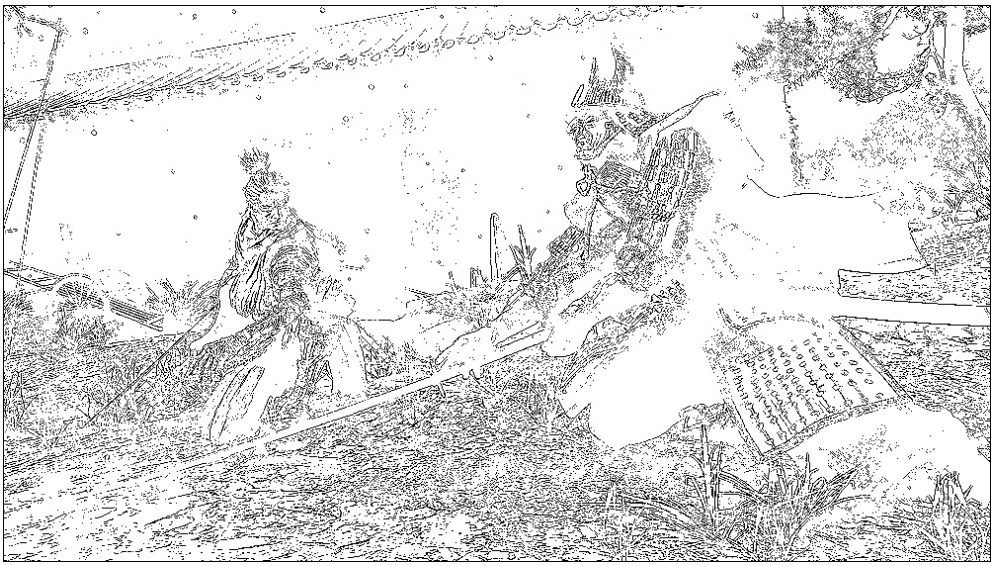
$$G_x(x, y) = (f' * \text{sobelX})(x, y)$$
$$G_y(x, y) = (f' * \text{sobelY})(x, y)$$

- Lúc này ảnh đầu ra g sẽ có giá trị là:

$$g(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$$

- Để có thể nhìn rõ biên cạnh, ta có thể chuyển ảnh về ảnh nhị phân bằng cách dùng một ngưỡng nào đó, gọi là ε (trong code $\varepsilon = 70$), những điểm ảnh $f(x, y)$ sẽ đưa về 0 nếu $f(x, y) > \varepsilon$, ngược lại ta đưa về 255.

2.8 Phát hiện biên cạnh Laplace



Hình 14: Ảnh 2 khi được áp dụng phát hiện biên cạnh Laplace

Command:

```
$ ./21120511 -laplace data/sekiro.jpg data/sekiro_laplace.jpg
```

- Đầu tiên, ta phải đưa f về ảnh xám, gọi là f'.
- Tiếp theo, ta có kernel để phát hiện biên cạnh Laplace, gọi là LAPLACE:

$$\text{LAPLACE} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Sau đó tại mỗi vị trí (x, y), thực hiện tích chập giữa $f'(x, y)$ và kernel LAPLACE để cho ra $g(x, y)$:

$$g(x, y) = (f * \text{LAPLACE})(x, y)$$

- Để có thể nhìn rõ biên cạnh, ta có thể chuyển ảnh về ảnh nhị phân bằng cách dùng một ngưỡng nào đó, gọi là ε (trong code $\varepsilon = 50$), những điểm ảnh $f(x, y)$ sẽ đưa về 0 nếu $f(x, y) > \varepsilon$, ngược lại ta đưa về 255.