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

BÁO CÁO LAB 03

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

Đánh giá

Yêu cầu	Mức độ hoàn thành
Thực hiện báo cáo	100%
Hướng dẫn chi tiết	100%
Cấu trúc thư mục đúng	100%
Hoàn thành các yêu cầu code	100%

Bảng 1: Bảng đánh giá mức độ hoàn thành

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) [2]:

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

- **Lưu ý**: nếu đang dùng anaconda thì cần tắt anaconda để tránh conflict, tắt anaconda bằng cách:

```
$ conda deactivate
```

- Để dùng được SIFT feature, ta phải build opencv và opencv_contrib (extension cho opencv) từ source, do đó cần install thêm các package sau:

```
$ sudo apt install python3-dev python3-numpy \
                   libavcodec-dev libavformat-dev libswscale-dev \
                   libgstreamer-plugins-base1.0-dev \
                   libgstreamer1.0-dev \
                   libgtk-3-dev \
                   libpng-dev libjpeg-dev libopenexr-dev \
                   libtiff-dev libwebp-dev \
                   libva-dev
```

- Tạo một folder để chứa opencv và opencv_contrib (ở đây em sẽ dùng bản 4.9.0 cả hai) [3]:

```
$ mkdir OpenCV && cd OpenCV
$ wget -O opencv.zip \
    https://github.com/opencv/opencv/archive/refs/tags/4.9.0.zip
$ wget -O opencv_contrib.zip \
    https://github.com/opencv/opencv_contrib/archive/refs/tags/4.9.0.zip
$ unzip opencv.zip
$ unzip opencv_contrib.zip
```

- Xong bước trên ta sẽ được 2 thư mục là **opencv-4.9.0** và **opencv_contrib-4.9.0**. Tiếp theo, tạo thư mục **build** và thực hiện build, ở đây ta chỉ thực hiện build extension xfeatures2d bởi vì ta cần extension này để dùng SIFT [4].

[1] Ubuntu này được em chạy trên WSL2 nhưng em nghĩ vẫn có thể áp dụng lên Ubuntu chạy trên máy thật.

[2] Ở đây em chọn build bằng ninja sẽ nhanh hơn dùng make.

[3] Mỗi một dấu `$` được hiểu là một command, hai dấu tức là thực hiện command phía trên xong rồi đến command phía dưới.

[4] Lưu ý, trong quá trình build có thể gặp lỗi, đừng lo, nếu gặp lỗi dừng thì cứ chạy lệnh `ninja` lần nữa, cho đến khi nào build hết tất cả file, việc chạy lệnh lại lần nữa thì `ninja` sẽ tự động bỏ các file build lỗi.

```
$ mkdir build && cd build
$ cmake -GNinja \
-DCMAKE_BUILD_TYPE=Release \
-DOPENCV_EXTRA_MODULES_PATH=../opencv_contrib-4.9.0/modules/xfeatures2d \
-DBUILD_TESTS=OFF ../opencv-4.9.0
$ ninja
$ sudo ninja install
```

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

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

- Sau đó thực hiện build source code bằng:

```
$ cmake ..
$ make
```

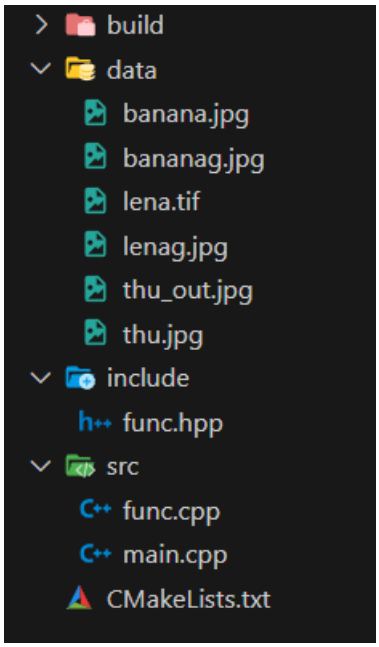
- Khi đã build source code thành công, thì có thể chạy bằng command sau:

```
$ ./21120511 -sift <TemplateInput> <SceneInput> <Output>
```

Lưu ý: đường dẫn tới ảnh (hoặc bất cứ file nào trong thư mục source) được hiểu là sẽ xuất phát từ file executable (nằm trong build) đi ra ngoài. Giả sử muốn dùng ảnh thu.jpg trong thư mục data làm ảnh template, ảnh banana.jpg làm ảnh scene và ảnh thu_out.jpg làm ảnh output, ta có thể dùng command như sau:

```
$ ./21120511 -sift ../data/thu.jpg
                   ../data/banana.jpg
                   ../data/thu_out.jpg

// 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

1.2 Chạy bằng file executable

Trong folder nộp bài sẽ gồm 4 folder nhỏ:

- **doc**: chứa báo cáo.
- **source**: 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 source).
- **exe**: bao gồm file executable có tên là 21120511.

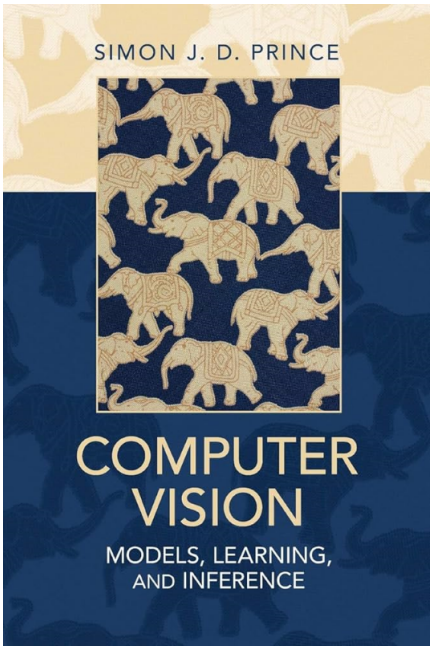
Để chạy bằng file executable, đầu tiên cho phép quyền chạy file bằng:

```
$ cd exe
$ chmod +x 21120511
```

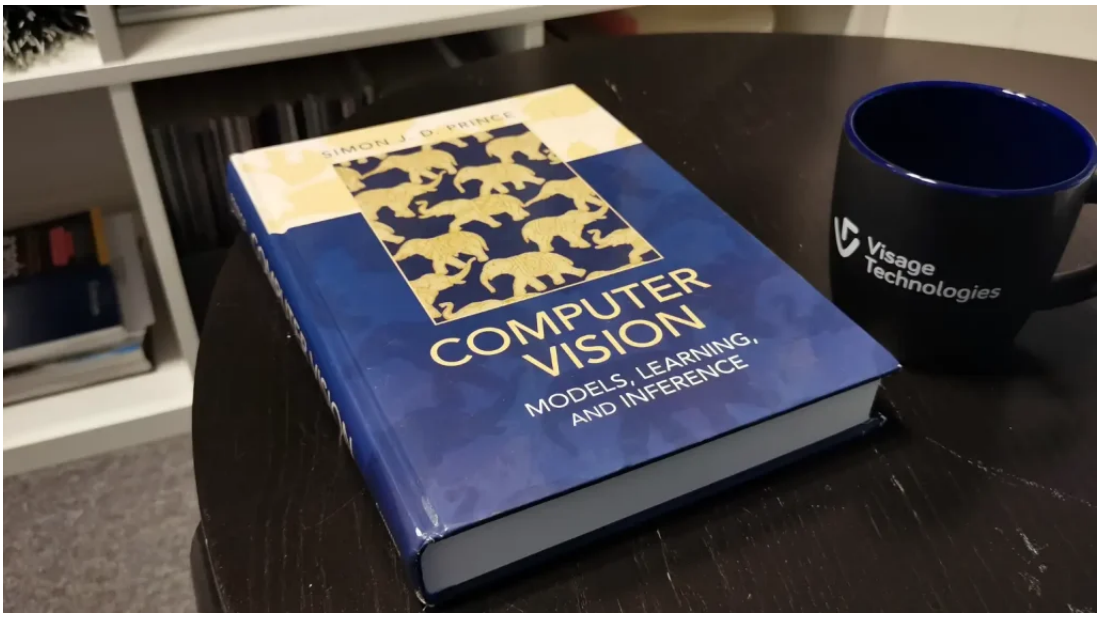
Sau đó có thể thực hiện chạy file bằng command như phần trên. Giả sử muốn input gồm ảnh template có tên là template.jpg, ảnh scene có tên là scene.jpg đều nằm trong folder **data** và trích xuất đặc trưng SIFT, xong lưu vào ảnh out.jpg nằm trong folder hiện tại thì ta nhập như sau:

```
$ ./21120511 -sift ../data/template.jpg ../data/scene.jpg out.jpg
```

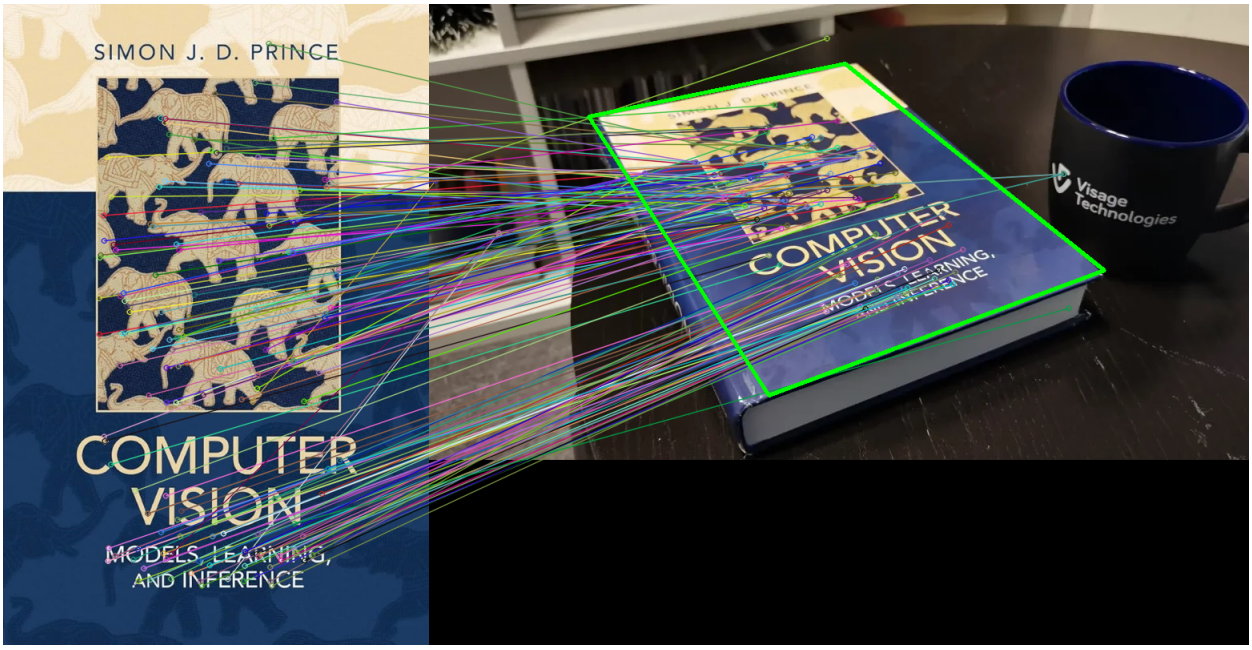
2 Thực nghiệm



Hình 2: Ảnh được dùng làm template



Hình 3: Ảnh được dùng làm scene



Hình 4: Ảnh kết quả sau khi dùng SIFT feature cho image matching

3 Chi tiết chương trình

```
12 class Image
11 {
10 public:
 9     Image();
 8     Image(const cv::Mat& img);
 7
 6     static Image ReadImg(std::string imgPath);
 5     void Write(std::string imgPath);
 4     void Show(std::string windowName);
 3
 2     Image ConvertGrayScale();
 1     Image AdjustBrightness(int value);
28     Image AdjustContrast(float value);
 1     Image AverageFilter(int k);
 2     Image MedianFilter(int k);
 3     Image GaussianFilter(int k);
 4     Image SobelDetect();
 5     Image LaplaceDetect();
 6     Image HarrisCornerDetect(double k);
 7
 8 #ifdef HAVE_OPENCV_XFEATURES2D
 9     static Image MatchingImgSIFT(const Image& object, const Image& scene);
10
11     // Matching descriptor vectors with a FLANN based matcher
12     static std::vector<cv::DMatch> Matching(const cv::Mat& descObj, const cv::Mat& descScene, float threshold);
13
14     // Draw line between matching point (homography)
15     static void DrawHomography(cv::Mat& imgMatches, const std::vector<cv::Point2f>& objPoint, const std::vector<cv
   ::Point2f>& scenePoint, const cv::Mat& object);
16 #endif
17
18 private:
NORMAL  func.hpp                                   utf-8  cpp  28%  28:38
```

Hình 5: Danh sách các hàm

Các hàm ở Lab 3 nằm trong phần `#ifdef HAVE_OPENCV_XFEATURES2D` cho đến `#endif`, các hàm còn lại ở Lab 2 và Lab 1 nhưng em sử dụng lại để thuận tiện hơn.

- `Matching`: tìm các điểm giống nhau (match) trong hai ảnh `object` và `scene`. Đây chính là tìm các điểm tròn mà ta thấy trong hình 4.
- `DrawHomography`: dựa vào các điểm match giữa hai ảnh (`imgMatches`) và các local point của ảnh (`objPoint` và `scenePoint`) để tiến hành nối hai ảnh `scene` và `object` lại với nhau bằng cách đường thẳng như ta thấy ở hình 4.
- `MatchingImgSIFT`: Sử dụng 2 ảnh và trả về ảnh mới là ảnh nối lại các matching point giữa 2 ảnh input. Cách hoạt động như sau:
  1. Đầu tiên tìm các key points của ảnh scene và template, đồng thời tính luôn descriptor của cả 2 (sử dụng đặc trưng SIFT).
  2. Tiếp theo tìm các điểm giống nhau tốt nhất bằng hàm `Matching`, sau khi đã tìm xong các điểm, ta tiến hành vẽ các điểm đó và thu được `imgMatches`.
  3. Sau khi đã có `imgMatches`, ta tiến hành vẽ Homography giữa hai ảnh scene và template bằng cách gọi hàm `DrawHomography`. Cuối cùng thu được ảnh kết quả (giống hình 4).