C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TPHCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**MÔN THỊ GIÁC MÁY TÍNH**

**BÁO CÁO BÀI TẬP 3**

**TRÍCH XUẤT ĐẶC TRƯNG CORNER, BLOB, SIFT CỦA ẢNH**

A picture containing indoor, sitting, computer, laptop

Description automatically generated

# Thông tin thành viên.

## Thông tin thành viên.

|  |  |  |  |
| --- | --- | --- | --- |
| **MSSV** | **Lớp** | **Họ và tên** | **Email** |
| *1712822* | *17TN* | Nguyễn Khánh Toàn | [ktoan271199@gmail.com](mailto:ktoan271199@gmail.com) |

# Một số lưu ý đối với chương trình.

## Cấu trúc thư mục nộp bài.

* **Source**: chứa toàn bộ mã nguồn của bài tập.
* **Release**: chứa file thực thi mã nguồn (.exe) và các file .dll, .lib đi kèm.
* **Docs**: chứa báo cáo bài tập.
* **Demo**: chứa file text chứa link youtube demo.

## Cách chạy các file mã nguồn.

* Chương trình được chạy với lệnh sau:

|  |
| --- |
| [ten\_chuong\_trinh] [duong\_dan\_anh] [ma\_lenh] [gaussian\_size] [gaussian\_signma] [tham\_so\_nguong(neu co)] |

* Trong đó:

+) ma\_lenh có thể là {**detect\_sobel, detect\_prewitt, detect\_laplacian, detect\_canny**}.

+) gaussian\_size ở đây là kích thước của mặt nạ Gaussian (thường là 3 hoặc 5).

+) gaussian\_sigma ở đây là giá trị signma của hàm Gaussian (thường là 1).

(mặt nạ Gaussian được dùng để giảm nhiễu của ảnh trước khi chạy các thuật toán phát hiện cạnh nên giá trị signma và kích thước ma trận khác nhau cũng ảnh hưởng đến kết quả cạnh khác nhau).

+) Các tham số ngưỡng (tham\_so\_nguong) nếu có sẽ là các giá trị ngưỡng (threshold), ví dụ Sobel, Prewitt, Laplace sẽ có 1 tham số ngưỡng đầu vào; trong khi Canny sẽ có 2 tham số ngưỡng đầu vào (tham số đầu tiên là ngưỡng nhỏ, tham số thứ hai là ngưỡng lớn). Mỗi loại tham số có một đặc điểm riêng biệt, cụ thể:

* **Sobel, Prewitt:** giá trị ngưỡng là giá trị Gradient để điểm nào có độ lớn Gradient vượt quá ngưỡng thì sẽ là cạnh.
* **Laplacian:** giá trị tham số của Laplacian sẽ là tham số khi đó ngưỡng sẽ là (ở đây là giá trị pixels lớn nhất trong ma trận chứa đạo hàm bậc 2 của ảnh gốc), chi tiết về tác dụng của ngưỡng được nói ở mục 3.

( tuy nhiên thường được chọn là **0.05, 0.1, 0.2, tốt nhất là chọn trong khoảng [0.05-0.2]**

* **Canny:** giá trị tham số thứ nhất là ngưỡng nhỏ, tham số thứ nhất là ngưỡng lớn (giá trị Gradient trong khoảng [0, 255]).
* Ví dụ những lệnh **hợp** **lệ**:

+ 1712122\_1712822\_BT02.exe lena.png detect\_sobel 3 1.0 100.

+ 1712122\_1712822\_BT02.exe lena.png detect\_prewitt 3 1.0 100.

+ 1712122\_1712822\_BT02.exe lena.png detect\_laplacian 3 1.0 0.2.

+ 1712122\_1712822\_BT02.exe lena.png detect\_canny 3 1.0 10 50

(tham số 0.2 ở thuật toán Laplacian có nghĩa là 0.2 \* giá trị max của ma trận ảnh gradient bậc 2, được giải thích ở mục 3).

## Kết quả chạy thí nghiệm.

* **Link youtube demo**:

|  |
| --- |
|  |

* Video trên minh họa đầy đủ cách sử dụng các tính năng của chương trình : mở và hiển thị ảnh, thực hiện phát cạnh của ảnh với các thuật toán Sobel, Prewitt, Laplace, Canny với ảnh đầu vào và hiển thị kết quả, ngoài ra có hiển thị ảnh gradient theo 2 hướng x và y đối với thuật toán Sobel, Prewitt.

# Mã giả các thuật toán và thực nghiệm, đánh giá kết quả.

## Tổng quan cách tổ chức các file, class.

## Thuật toán trích xuất đặc trưng Corner Harris Detector.

1. **Mã giả.**

|  |
| --- |
| **Bước 1**: Đọc ảnh đầu vào và chuyển ảnh đó qua ảnh xám.  **Bước 2**: Thực hiện Gaussian Blur, làm mờ ảnh để giảm thiểu nhiễu ảnh xám đó (sử dụng ơheps convolution với bộ lọc Gaussian với ).  **Bước 3**: Thực hiện convolution với bộ lọc Sobel để tính Gradient theo phương và phương (.  **Bước 4**: Tính các giá trị sau đó lấy các ma trận này thực hiện phép convolution với bộ lọc Gaussian với .  **Bước 5**: Tính giá trị , trong đó và  **Bước 6**: So sánh với giá trị ngưỡng, thường là thường được chọn là Nếu giá trị tại pixel đó lớn hơn giá trị ngưỡng thì đây được xem là góc.  **Bước 7**: Duyệt tất cả các điểm góc đã chọn ở bước trên, với mỗi điểm ta duyệt lại các điểm góc đã được thêm vào mảng mà có **khoảng** **cách** **Manhattan** với điểm sau đó ta chỉ giữ lại trong mảng điểm có giá trị lớn nhất trong các điểm này. Ở đây em chọn  **Cuối cùng,** ta thu được mảng chứa tọa độ các điểm là góc từ 7 bước trên.  Bài toán trên có các tham số được lựa chọn heuristic như sau: |

1. **Sơ lược về cách cài đặt.**

* Cài đặt lớp HarrisDetector, với thuộc tính public detectHarris có nhiệm vụ thực hiện 7 bước trên và trả về mảng chứa tạo độ điểm góc của ảnh truyền vào.
* **Bước** **1** thực hiện đọc ảnh bằng imread, và chuyển ảnh qua ảnh xám bằng hàm cvtColor.
* **Bước** **2** thực hiện bằng hàm filter2D của thư viện OpenCV, với bộ lọc Gaussian tự phát sinh từ công thức bằng hàm createGaussianKernel trong file mã nguồn Utils.cpp.
* **Bước** **3** thực hiện bằng hàm filter2D của thư viện OpenCV, với bộ lọc Sobel có sẵn.
* **Bước** **4** duyệt toàn bộ các ma trận và thực hiện các **phép nhân element-wise**, rồi thực hiện filter2D với các bộ lọc Gaussian.
* **Bước** **5** thực hiện tính toán như mã giả với giá trị truyền vào, kết quả trả về là ma trận chứa giá trị tại mỗi pixel (
* **Bước** **6** duyệt toàn bộ ma trận thu được và so sánh với ngưỡng, rồi thêm các điểm góc vào một **vector<CornerPoint>,** CornerPoint là lớp tự thiết kế với các thuộc tính là tọa độ tương ứng với điểm pixel và giá trị tương ứng tại vị trí đó.
* **Bước** **7**: Thực hiện sắp xếp (sorting) lại **vector<CornerPoint>** thu được ở bước 6 theo giá trị của mỗi phần tử giảm dần. Khai báo một **vector<CornerPoint> output** mới chứa các **CornerPoint** cuối cùng, thực hiện duyệt lần lượt **vector<CornerPoint>** đã sắp xếp trước đó, nếu tại mỗi vị trí không tìm thấy lân cận của nó (khoảng cách **Manhattan** ) trong **vector<CornerPoint> output,** thì điểm này được thêm mới vào **vector<CornerPoint> output,** nếu có tìm thấy thì điểm này sẽ được bỏ qua, không được thêm vào kết quả.

1. **Thực nghiệm và đánh giá kết quả thực nghiệm.**

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Harris với tham số |

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Harris với tham số |

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Harris với tham số |

***Nhận******xét****:* Dựa vào kết quả thực nghiệm, có thể thấy thuật toán được em cài đặt chạy khá tốt, ảnh số 2 đã phát hiện được hầu hết các điểm góc của ảnh, tuy nhiên vẫn còn một số điểm góc chưa phát hiện được có thể do ảnh hưởng của nhiễu hoặc ảnh hưởng của việc Gaussian Blur khá nhiều là giá trị điểm ảnh tại đó bị nhỏ đi. Ảnh số 1, 3 cũng phát hiện được hầu hết các góc chính của sự vật trong ảnh.

## Thuật toán trích xuất đặc trưng Blob bằng Blob Detector.

1. **Mã giả.**

|  |
| --- |
| **Bước 1**: Đọc ảnh đầu vào và chuyển ảnh đó qua ảnh xám.  **Bước 2**: Thực hiện phép toán convolution ảnh xám với bộ lọc LoG (Laplacian of Gaussian) với scales khác nhau, scale đầu tiên có giá trị signma = , mỗi scale kế tiếp sẽ có giá trị signma bằng giá trị signma của scale trước đó nhân với , trả về mảng các ma trận kết quả với kích thước mảng bằng số scales (kích thước = ), kết quả trả về được nhân với ứng với signma tương ứng tại ma trận kết quả đó.  **Bước 3**: Tìm điểm Extrema ở mỗi ma trận kết quả trong mảng các ma trận kết quả trả về ở bước trên, điểm này phải có giá trị lớn hơn hoặc nhỏ hơn tất cả 26 điểm (nếu điểm đó ở biên thì chỉ cần lớn hơn hoặc nhỏ hơn tất cả 17 điểm) lân cận với nó ở scale này và các điểm với lân cận với vị trí tương ứng của nó nhưng ở các scale lân cận.  **Bước 4**: Ngoài ra, cần có một heuristic nữa để loại bỏ bớt các điểm Blob dư thừa, các điểm Extrema tìm được ở bước trên phải có giá trị **bình** **phương** lớn hơn ngưỡng thường được chọn là.  **Cuối cùng,** ta thu được mảng chứa tọa độ các điểm là điểm blob từ 4 bước trên.  Bài toán trên có các tham số được lựa chọn heuristic như sau: |

1. **Sơ lược về cách cài đặt.**

* Cài đặt lớp BlobDetector, với thuộc tính public detectBlob có nhiệm vụ thực hiện 4 bước trên và trả về mảng chứa tạo độ điểm blob của ảnh truyền vào.
* **Bước** **1:** thực hiện đọc ảnh bằng imread, và chuyển ảnh qua ảnh xám bằng hàm cvtColor.
* **Bước 2**: tạo các bộ lộc LoG (Laplacian of Gaussian) bằng hàm createLoG\_Kernel được code từ công thức của hàm Laplacian of Gaussian, đặc biệt hàm này có khác công thức gốc là có nhân thêm , nhằm hạn chế việc các giá trị kết quả sau phép convolution quá nhỏ và gần nhau và bị vanishing về 0, từ đó đảm bảo việc detect được các điểm Blob. Các bộ lọc với các giá trị signma tăng theo cấp số nhân này tạo thành mảng các bộ lọc. Sau đó thực hiện phép convolution ảnh gốc với các bộ lọc ta thu được mảng kết quả.
* **Bước 3**: Tìm các điểm Extrema trong mảng kết quả, lưu ý thay vì tìm cực đại và cực tiểu của mảng kết quả, có thể chỉ tìm cực đại của ma trận mới chứa bình phương giá trị của mảng kết quả.
* **Bước 4**: So sánh giá trị **bình** **phương** của các điểm Extrema tìm được với ngưỡng sau quá trình thử chọn em thấy Heuristic này cho kết quả rất tốt với .

1. **Thực nghiệm và đánh giá kết quả thực nghiệm.**

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Blob với tham số |

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Blob với tham số |

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Harris với tham số |

***Nhận xét:*** Nhờ có heuristic ở bước 4, thuật toán hoạt động khá tốt, đã phá thiện được phần lớn các điểm blob chính trong các ảnh thực nghiệm, tuy nhiên mỗi lần chạy cần phải lựa chọn khá nhiều, giá trị này dao động khác biệt rất lớn đối với ảnh đầu vào khác nhau.

## Thuật toán trích xuất đặc trưng Blob bằng DOG Detector.

1. **Mã giả.**

|  |
| --- |
| **Bước 1**: Đọc ảnh đầu vào và chuyển ảnh đó qua ảnh xám.  **Bước 2**: Đây là bước duy nhất khác của thuật toán DOG so với thuật toán Blob sử dụng LOG. Tạo mảng các mặt nạ Gaussian với giá trị signma khác nhau (lưu ý rằng kích thước mỗi mặt nạ càng tăng dần với giá trị signma tăng dần), sau đó lần lượt lấy ảnh gốc thực hiện phép convolution với các mặt nạ Gaussian này rồi lấy hiệu của mỗi 2 kết quả convolution liên tiếp, lần lượt thêm các hiệu này vào ma trận kết quả.  **Bước 3**: Tìm điểm Extrema ở mỗi ma trận kết quả trong mảng các ma trận kết quả trả về ở bước trên, điểm này phải có giá trị lớn hơn hoặc nhỏ hơn tất cả 26 điểm (nếu điểm đó ở biên thì chỉ cần lớn hơn hoặc nhỏ hơn tất cả 17 điểm) lân cận với nó ở scale này và các điểm với lân cận với vị trí tương ứng của nó nhưng ở các scale lân cận.  **Bước 4**: Ngoài ra, cần có một heuristic nữa để loại bỏ bớt các điểm Blob dư thừa, các điểm Extrema tìm được ở bước trên phải có giá trị **bình** **phương** lớn hơn ngưỡng thường được chọn là.  **Cuối cùng,** ta thu được mảng chứa tọa độ các điểm là điểm blob từ 4 bước trên.  Bài toán trên có các tham số được lựa chọn heuristic như sau: |

1. **Sơ lược về cách cài đặt.**

* Cài đặt lớp BlobDetector, với thuộc tính public detectDOG có nhiệm vụ thực hiện 4 bước trên và trả về mảng chứa tạo độ điểm blob của ảnh truyền vào.
* **Bước** **1:** thực hiện đọc ảnh bằng imread, và chuyển ảnh qua ảnh xám bằng hàm cvtColor.
* **Bước 2**: sinh các bộ lọc Gaussian bằng lệnh createGaussianKernel được cài đặt trong file Utils.cpp có tác dụng sinh ra các bộ lọc Gaussian ứng với signma truyền vào và kích thước truyền vào. Sau đó thực hiện
* **Bước 3**: Tìm các điểm Extrema trong mảng kết quả, lưu ý thay vì tìm cực đại và cực tiểu của mảng kết quả, có thể chỉ tìm cực đại của ma trận mới chứa bình phương giá trị của mảng kết quả.
* **Bước 4**: So sánh giá trị **bình** **phương** của các điểm Extrema tìm được với ngưỡng sau quá trình thử chọn em thấy Heuristic này cho kết quả rất tốt với .

1. **Thực nghiệm và đánh giá kết quả thực nghiệm.**

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Harris với tham số |

***Nhận xét:*** Nhìn chung kết quả phát hiện các điểm Blob của thuật toán DOG gần giống như thuật toán Blob sử dụng LOG, đúng như bản chất của lý thuyết hai thuật toán này khác nhau chủ yếu ở bước DOG là xấp xỉ của LOG. Kết quả các điểm Blob phát hiện được cũng khá tốt, việc lọc nhiễu cho ra kết quả thành công đáng kể. Ảnh 1, ảnh 2, ảnh 3.

## Thuật toán trích xuất đặc trưng SIFT.

# Đánh giá kết quả hoàn thành.

|  |  |  |
| --- | --- | --- |
| **Yêu cầu** | **Mức độ hoàn thành** | **Ghi chú** |
|  |  |  |
|  |  |  |
| Chọn 5 bức ảnh để chạy và so sánh, báo cáo kết quả | 100% | Đã thử nghiệm, báo cáo so sánh và nhận xét, so sánh các kết quả chạy được với các thuật toán, đồng thời có thực hiện quay lại clip demo. |
| Cấu trúc chương trình rõ ràng, mã nguồn chú thích đầy đủ, tên biến hàm đúng quy định, minh họa chương trình bằng clip demo. | 100% |  |

# Tài liệu tham khảo

5.1. Slide bài giảng môn Thị Giác Máy Tính, Đại Học Khoa Học Tự Nhiên TPHCM.

5.2. Bài giảng UCF Computer Vision Video Lectures 2014.

5.3. OpenCV Documentation.