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)] |

## Kết quả chạy thí nghiệm.

* **Link youtube demo**:

|  |
| --- |
|  |

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

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

* Tổ chức file mã nguồn:
* **Harris**.**h**, **Harris**.**cpp**: chứa lớp HarrisDetector bao gồm các hàm liên quan đến phát hiện điểm góc dựa trên thuật toán Harris.
* **Blob**.**h**, **Blob**.**cpp**: chứa lớp BlobDetector bao gồm các hàm liên quan đến phát hiện điểm Blob dựa trên thuật toán Blob, DoG.
* **Sift**.**h**, **Sift**.**cpp**: chứa lớp SiftDetector bao gồm các hàm liên quan đến phát hiện điểm keypoint (đặc trưng SIFT) và các vector đặc trưng của nó, đồng thời còn có hàm thực hiện việc matching 2 ảnh.
* **Utils**.**h**, **Utils**.**cpp**: chứa các hàm phụ trợ cho chương trình như tạo bộ lọc gaussian, nhân ma trận, chuyển từ ảnh màu sang ảnh xám, …
* **main**.**cpp**: Hàm chương trình chính.

## 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 phép 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: Tuy nhiên để cho đơn giản sử dụng, mã lệnh thực thi chương trình chỉ có 2 biến truyền vào là |

* Cài đặt lớp HarrisDetector, với phương thức public detectHarris có nhiệm vụ thực hiện 7 bước trên và trả về mảng chứa tạo độ điểm góc của ảnh truyền vào.

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 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, sai số trong tính toán; ảnh hưởng của việc Gaussian Blur khá nhiều làm giá trị điểm ảnh tại đó bị nhỏ đi hoặc chọn tham số chưa tối ưu. Ảnh số 1, 3 cũng phát hiện được hầu hết các góc chính của sự vật trong ảnh. Trong 2 tham số truyền vào thì tham số hầu như không thay đổi nhiều với các ảnh đầu vào khác nhau, tham số thay đổi khá nhiều đổi với ảnh đầu vào khác nhau để quyết định xem có cần lấy thêm điểm (giảm hoặc bỏ bớt điểm (tăng

## 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. 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**: 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 giá trị mặc định là 0.3.  **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: |

* Cài đặt lớp BlobDetector, với phương thức public detectBlob có nhiệm vụ thực hiện 4 bước trên và trả về mảng chứa tạo độ điểm blob của ảnh truyền vào.

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

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

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

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

***Nhận xét:*** Nhờ có heuristic ở bước 4, thuật toán hoạt động rất tốt, đã bỏ được phần lớn các vòng tròn bị overlapped và giữ lại các vòng tròn quan trọng, 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, giá trị hầu như không cần thay đổi nhiều so với mọi ảnh truyền vào.

## 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. 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**: 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 làgiá trị mặc định là 0.3.  **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: |

* Cài đặt lớp BlobDetector, với phương thức public detectDOG có nhiệm vụ thực hiện 4 bước trên và trả về mảng chứa tạo độ điểm blob của ảnh truyền vào.

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

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

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

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm Blob thực hiện bởi thuật toán DOG 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, lọc được rất nhiều điểm ảo. Ảnh 1, ảnh 2, ảnh 3 đã phát hiện được phần lớn các điểm Blob của ảnh. Giống như thuật toán Blob, 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, giá trị hầu như không cần thay đổi nhiều so với mọi ảnh truyền vào.

## Thuật toán trích xuất đặc trưng SIFT.

1. **Mã giả thuật toán trích xuất điểm đặc trưng SIFT.**

|  |
| --- |
| **Bước 1**: Đọc ảnh đầu vào và chuyển ảnh đó qua ảnh xám; thực hiện Gaussian Blur ảnh đó để giảm nhiễu với kích thước 5x5, signma = 1.3; sau đó xét nếu như ảnh không quá to (nếu chiều dài và chiều rộng ảnh < 500), thì ảnh sẽ được nhân đôi kích thước.  **Bước 2**: Tạo ra các ma trận Gaussian ở các octave với các signma khác nhau và thực hiện phép convolution với ảnh (mỗi octave kế tiếp sẽ chạy với ảnh của octave trước đó chia đôi kích thước), từ đó lấy hiệu các kết quả convolution sẽ tính được các ma trận DoG ở mỗi octave. Em tổ chức mặc định 4 octave, mỗi octave có 5 kích thước signma khác nhau => có 4 ma trận DoG được sinh ra. Ngoài ra có thể truyền vào tham số  **Octave:** k em chọn mặc định là giá trị khởi đầu của các octave là  **Bước 3:** Tìm điểm Extrema ở mỗi ma trận kết quả trong mảng các ma trận kết quả trả về DoG 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. Ngoài ra, giống như thuật toán Blob và Dog, 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 mặc định em chọn là (vì ra kết quả tốt với đa số trường hợp).  **Bước 4 (Keypoint Localization):** thực hiện như trong slide, trước tiên loại bỏ các điểm ít nhạy cảm với độ contrast (tính giá trị contrast tại mỗi điểm rồi so sánh với ngưỡng thresh\_contrast, ở đây em để mặc định như trong paper là 0.03), tiếp theo là loại bỏ các điểm cạnh loại bỏ các điểm cạnh nếu (ở đây em để mặc định giá trị r như trong paper là r=10).  **Bước 5 (Orientation Assignment):** làm tương tự như trong slide, tại mỗi vị trí của điểm keypoint, duyệt các điểm hàng lân cận với window\_size = 3, em tính giá trị magnitude và giá trị góc theta cho các điểm hàng xóm, sau đó tính giá trị weight của góc theta bằng cách lấy giá trị magnitude \* với giá trị tương ứng tại điểm đó so với Gaussian kích thước 3x3 áp lên window\_size (ở đây ý tưởng là những điểm xa càng xa điểm keypoint keypoint đang xét thì weight càng thấp hơn so với những điểm gần điểm keypoint đang xét, mặt nạ Gaussian dựa theo phân phối chuẩn nên càng về trung tâm giá trị của nó càng lớn). Sau đó ta lần lượt thêm giá trị weight vào các bin ứng với góc tương ứng (ở đây em chọn số bin = 36), tìm bin có giá trị lớn nhất trong 36 bin. Sau đó thêm điểm keypoint này vào mảng các điểm keypoint với giá trị góc lớn nhất ứng với bin có giá trị lớn nhất này (ở đây em xấp xỉ góc lớn nhất bằng cách nội suy đường cong parabol với 3 điểm tương ứng với bin cực đại rồi lấy điểm cực đại của parabol). Lưu ý những bin có giá trị weight > 0.8 \* max\_weight cũng được thêm vào.  **Bước 6 (Keypoint Descriptor):** Làm tương tự như bước orientation assignment chỉ khác window\_size = 16, tính các giá trị weight tương ứng với các bin của histogram (ở đây chỉ có 8 bin) ở mỗi vùng 4x4, cuối cùng ta thu được vector 128 chiều tương ứng với mỗi điểm keypoint.    **Cuối cùng,** ta thu được mảng chứa tọa độ các điểm keypoint và vector đặc trưng 128 chiều ứng với mỗi điểm từ 6 bước trên.  Bài toán trên có các tham số được lựa chọn heuristic như sau: thresh\_edge, thresh\_contrast, windowSize. Tuy nhiên để đơn giản, mã lệnh của chương trình chỉ bao gồm các tham số (vì các tham số còn lại không có sự thay đổi lớn đối với ảnh input đầu vào). |

1. **Kết quả chạy thực nghiệm.**

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm keypoint SIFT với tham số |

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm keypoint SIFT với tham số |

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc ban đầu | Ảnh kết quả chứa điểm keypoint SIFT với tham số |

***Nhận xét:*** Nhìn chung kết quả phát hiện các điểm keypoint của thuật toán SIFT do em cài đặt chạy tốt, phát hiện được gần như hầu hết tất cả những điểm keypoint quan trọng của đối tượng trong hầu hết các ảnh mà em thực nghiệm. Vì tham số khá nhiều và các heuristic được thực hiện nhiều nên việc thử chọn tham số heuristic đối với thuật toán SIFT khá mất thời gian, em đã giảm bớt số lượng tham số có trong mã lệnh chạy chương trình để cho đơn giản, chỉ còn 3 tham số Các giá trị em chạy thực nghiệm cho ra kết quả tốt nhất hiện tại là: các tham số còn lại được truyền sẵn trong mã nguồn chương trình.

## Thuật toán matching hai ảnh.

1. **Mã giả thuật toán matching.**

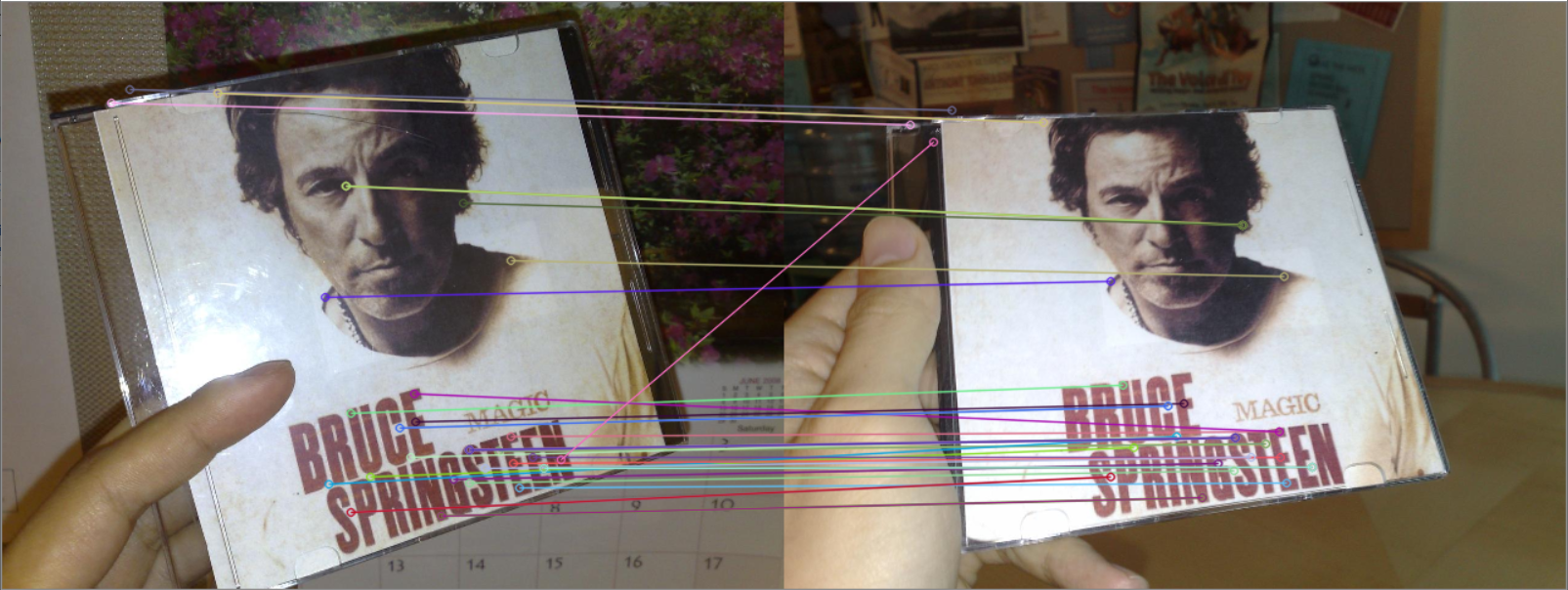
|  |
| --- |
| **Bước 1**: Tìm vector của các 2 ảnh đầu vào cần matching với thuật toán SIFT cài đặt ở trên.  **Bước 2:** Khai báo DescriptorMatcher (với types Matcher là “**BruteForce-L1**”) và sử dụng hàm knnMatch với các vector của ảnh train và ảnh test truyền vào.  **Bước 3:** Thực hiện chọn lại những cặp matches tốt nhất bằng cách sử dụng heuristic: Nếu tỷ lệ khoảng cách giữa hàng xóm gần nhất thứ nhất với hàng xóm gần nhất thứ hai< threshold = 0.8 thì mới chọn cặp matches đó (trong paper của tác giả đề xuất).  **Bước 4:** Sử dụng hàm drawMatches của thư viện OpenCV để vẽ ảnh kết quả matching giữa 2 ảnh đầu vào. |

* Cài đặt lớp SiftDetector, với 2 phương thức public quan trọng matchingTwoImages có nhiệm vụ thực hiện việc matching ảnh train và ảnh test truyền vào, trong hàm matchingTwoImages có gọi hàm siftDetector có nhiệm vụ phát hiện các điểm keypoint của mỗi ảnh và tính các vector đặc trưng ứng với mỗi điểm đặc trưng đó, các hàm chức năng còn lại là phương thức private.

1. **Kết quả chạy thực nghiệm.**

* Em đã thực hiện việc matching chạy bằng thuật toán SIFT tự cài đặt với hầu hết các ảnh trong tập test và train, dưới đây em chỉ liệt kê một số trường hợp chương trình chạy cho ra kết quả tốt và một số trường hợp chạy cho ra kết quả không tốt và nhận xét kết quả. Nhìn chung kết quả matching đối với ảnh train và ảnh test nếu không bị xoay góc quá lớn sẽ matching rất tốt, matching các điểm rất khớp với nhau, tuy nhiên nếu 2 ảnh đầu vào có sự khác biệt quá lớn (góc xoay quá lớn) thì việc detect sẽ bị nhầm lẫn với một vài điểm.

1. **Các trường hợp chạy tốt.**
2. Ảnh train: **06\_3.jpg**; Ảnh test: **03.jpg**



***Nhận xét:*** Nhìn chung kết quả phát hiện các điểm keypoint của thuật toán SIFT do em cài đặt chạy tốt, phát hiện được

1. **Các trường hợp chạy không tốt.**
2. Ảnh

***Nhận xét:*** Nhìn chung kết quả phát hiện các điểm keypoint của thuật toán SIFT do em cài đặt chạy tốt, phát hiện được gần như hầu hết tất cả những điểm keypoint quan trọng của đối tượng trong hầu hết các ảnh mà em thực nghiệm. Vì tham số khá nhiều và các heuristic được thực hiện nhiều nên việc thử chọn tham số heuristic đối với thuật toán SIFT khá mất thời gian, em đã giảm bớt số lượng tham số có trong mã lệnh chạy chương trình để cho đơn giản, chỉ còn 3 tham số Các giá trị em chạy thực nghiệm cho ra kết quả tốt nhất hiện tại là: các tham số còn lại được truyền sẵn trong mã nguồn chương trình.

# 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 | 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.