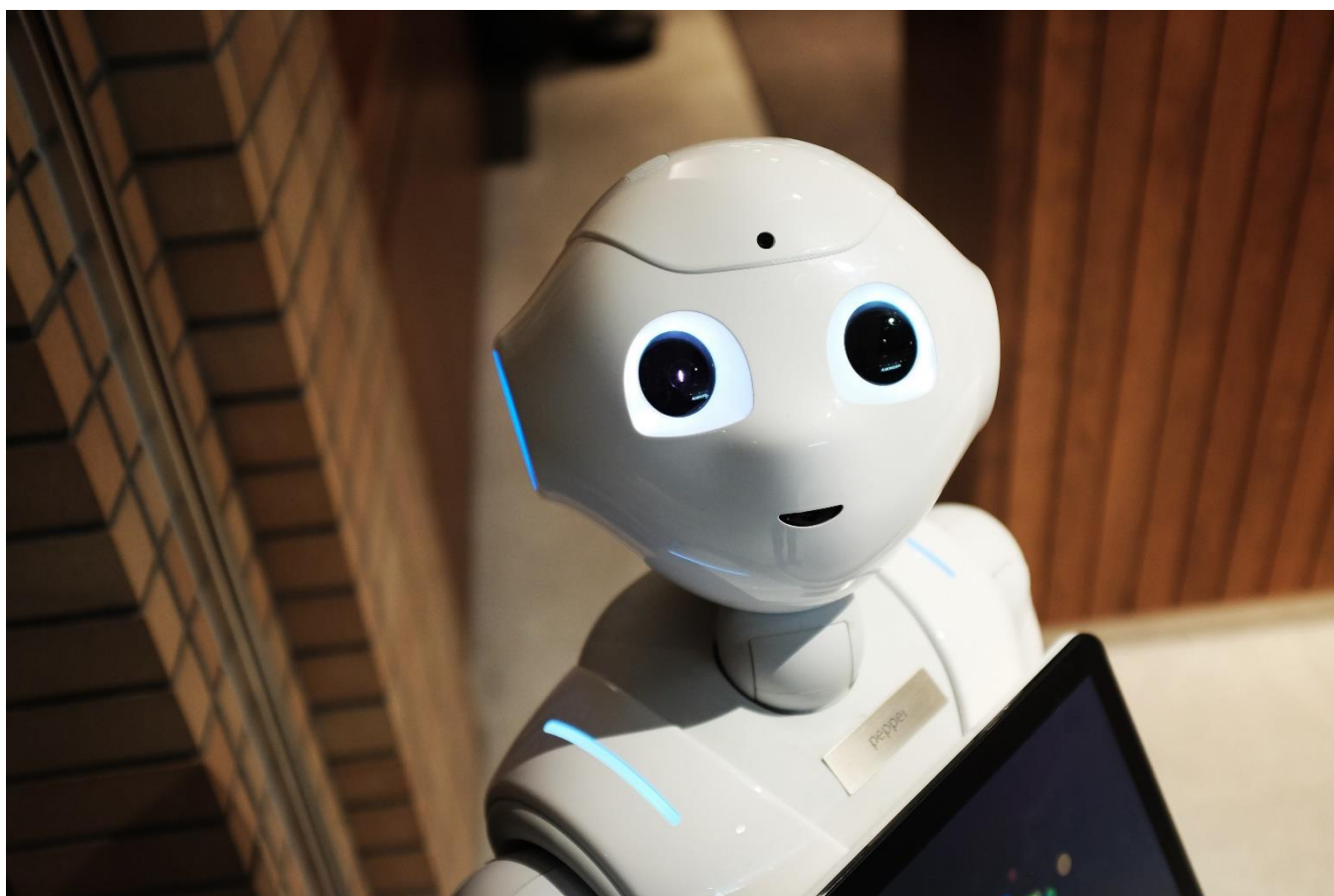




TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP HCM
KHOA CÔNG NGHỆ THÔNG TIN

MÔN THỊ GIÁC MÁY TÍNH

BÁO CÁO BÀI TẬP 3 TRÍCH XUẤT ĐẶC TRƯNG CORNER, BLOB, SIFT CỦA ẢNH



1 Thông tin thành viên.

1.1 Thông tin thành viên.

MSSV	Lớp	Họ và tên	Email
1712822	17TN	Nguyễn Khánh Toàn	ktoan271199@gmail.com

2 Một số lưu ý đối với chương trình.

2.1 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.

2.2 Cách chạy các file mã nguồn.

a. Thuật toán phát hiện góc Harris

```
[ten_chuong_trinh] [duong_dan_anh] harris [tham_so_k] [tham_so_alpha]
```

- Tham số k thường được chọn là $k = 0.05$, $\alpha = 0.01$.
- Ngoài ra chương trình vẫn chạy với tham số mặc định nếu chỉ truyền vào mã lệnh harris, không cần truyền vào các tham số hoặc chỉ truyền một số tham số nhất định.
- Ví dụ về những lệnh hợp lệ:
1712822_Lab03.exe lena.png harris
1712822_Lab03.exe lena.png harris 0.05
1712822_Lab03.exe lena.png harris 0.05 0.01

b. Thuật toán phát hiện điểm Blob – Blob Detector và DoG Detector.

```
[ten_chuong_trinh] [duong_dan_anh] blob [tham_so_threshold_max] [tham_so_sigma]  
[tham_so_k]
```

```
[ten_chuong_trinh] [duong_dan_anh] dog [tham_so_threshold_max] [tham_so_sigma]  
[tham_so_k]
```

- Ở đây:
 - o tham số threshold_max thường được chọn là 0.3, giá trị có thể chọn ra giá trị tốt là [0.1, 0.5](đây giống như bước suppression, có thể tăng threshold_max nếu dư thừa quá nhiều blob).
 - o tham số sigma thường được chọn là 1.0.
 - o tham số k thường được chọn là $\sqrt{2} = 1.41421356$.
- Tuy nhiên có thể chạy chương trình mà không cần truyền vào hết các tham số, có thể không truyền hoặc truyền một vài tham số, khi đó chương trình sẽ chạy với các tham số mặc định nếu không được người dùng truyền vào.
- Ví dụ về những lệnh hợp lệ:
1712822_Lab03.exe sunflower.png blob
1712822_Lab03.exe sunflower.png blob 0.15
1712822_Lab03.exe sunflower.png dog
1712822_Lab03.exe sunflower.png dog 0.15 1.0

c. Thuật toán phát hiện đặc trưng SIFT.

```
[ten_chuong_trinh] [duong_dan_anh] sift [n_octave] [n_scales] [tham_so_sigma]
```

- Ở đây:
 - o n_octave là số octave, thường được chọn là octave = 4.
 - o n_scales là số scales trong mỗi octave, thường được chọn là scales = 5.
 - o tham_so_sigma là giá trị sigma đầu vào của mỗi octave, thường là sigma = 1.6.
- Tuy nhiên có thể chạy chương trình mà không cần truyền vào hết các tham số, có thể không truyền hoặc truyền một vài tham số, khi đó chương trình sẽ chạy với các tham số mặc định nếu không được người dùng truyền vào.
- Thuật toán SIFT còn khá nhiều tham số heuristic, tuy nhiên để mã lệnh đơn giản em chỉ chọn những tham số chính có ảnh hưởng lớn, các tham số còn lại được để mặc định.
- Ví dụ về những lệnh hợp lệ:
1712822_Lab03.exe train.png sift
1712822_Lab03.exe train.png sift 4

1712822_Lab03.exe train.png sift 4 5 1.6

d. Thuật toán matching 2 ảnh.

[ten_chuong_trinh] [duong_dan_anh_train] matching [duong_dan_anh_test] [tham_so_octave]

- Ở đây tham số octave được chọn là 1, các tham số hợp lệ là trong khoảng $[0, 3]$, ngoài ra có thể không truyền tham số octave, chương trình sẽ chạy với tham số octave mặc định, ngoài ra các tham số sử dụng thuật toán SIFT trong việc matching được để mặc định với các tham số chạy tốt trong hầu hết các ảnh thử nghiệm để việc thực thi chương trình đơn giản hơn.

- Ví dụ về những lệnh hợp lệ:

1712822_Lab03.exe train.png matching test.png

1712822_Lab03.exe train.png matching test.png 2

2.3 Kết quả chạy thí nghiệm.

- **Link youtube demo:**

<https://www.youtube.com/watch?v=3XGBbuudbAo>

- **Link drive demo:**

<https://drive.google.com/drive/u/0/folders/1EbqY6cJtCj8akLJ8AOYZWWdZwFfTtVAU>

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

3.1 Tổng quan cách tổ chức các file, class.

- Tổ chức file mã nguồn:
 - o **Harris.h, Harris.cpp**: chứa lớp **HarrisDetector** bao gồm các hàm liên quan đến phát hiện điểm góc dựa trên thuật toán Harris.
 - o **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.

3.2 Thuật toán trích xuất đặc trưng Corner Harris Detector.

a. 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 5x5, với $\sigma = 1.0$).

Bước 3: Thực hiện convolution với bộ lọc Sobel để tính Gradient theo phương x và phương y (G_x, G_y).

Bước 4: Tính các giá trị $G_x^2, G_y^2, G_x G_y$, sau đó lấy các ma trận này thực hiện phép convolution với bộ lọc Gaussian 3x3, với $\sigma = 1.0$.

Bước 5: Tính giá trị $R = \det M - k \cdot (\text{trace} M)^2$, trong đó $M = \begin{bmatrix} G_x^2 & G_x G_y \\ G_x G_y & G_y^2 \end{bmatrix}$, và $k = [0.04, 0.06]$

Bước 6: So sánh R với giá trị ngưỡng, thường là $\alpha R_{\{max\}}$, α thường được chọn là $\alpha = 0.01$. Nếu giá trị R 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 (i, j) ta duyệt lại các điểm góc đã được thêm vào mảng mà có **khoảng cách Manhattan** với điểm $(i, j) < d$, sau đó ta chỉ giữ lại trong mảng điểm có giá trị R lớn nhất trong các điểm này. Ở đây em chọn $d = 10$.



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: k, α, d . Tuy nhiên để cho đơn giản sử dụng, mã lệnh thực thi chương trình chỉ có 2 biến truyền vào là k, α .

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

b. Thực nghiệm và đánh giá kết quả thực nghiệm.

	
<p>Ảnh gốc ban đầu</p>	<p>Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Harris với tham số $k = 0.05, \alpha = 0.01$.</p>
	
<p>Ảnh gốc ban đầu</p>	<p>Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Harris với tham số $k = 0.05, \alpha = 0.00005$.</p>

	
<p>Ảnh gốc ban đầu</p>	<p>Ảnh kết quả chứa điểm góc thực hiện bởi thuật toán Harris với tham số $k = 0.05, \alpha = 0.01$.</p>

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 $k = 0.05$ hầu như không thay đổi nhiều với các ảnh đầu vào khác nhau, tham số α thay đổi khá nhiều đối với ảnh đầu vào khác nhau để quyết định xem có cần lấy thêm điểm (giảm α) hoặc bỏ bớt điểm (tăng α).

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

a. 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 10 scales khác nhau, scale đầu tiên có giá trị $\text{signma} = \sigma$, mỗi scale kế tiếp sẽ có giá trị signma bằng giá trị signma của scale trước đó nhân với k , trả về mảng các ma trận kết quả với kích

thước mảng bằng số scales (kích thước = 10), kết quả trả về được nhân với σ^2 ứng với sigma 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 $\alpha \cdot \text{Max}(\text{giá trị bình phương của ma trận kết quả ứng với điểm Extrema hiện tại})$, α thường được chọn là $[0.1, 0.6]$, 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: σ, k, α .

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

b. Thực nghiệm và đánh giá kết quả thực nghiệm.



Ảnh gốc ban đầu




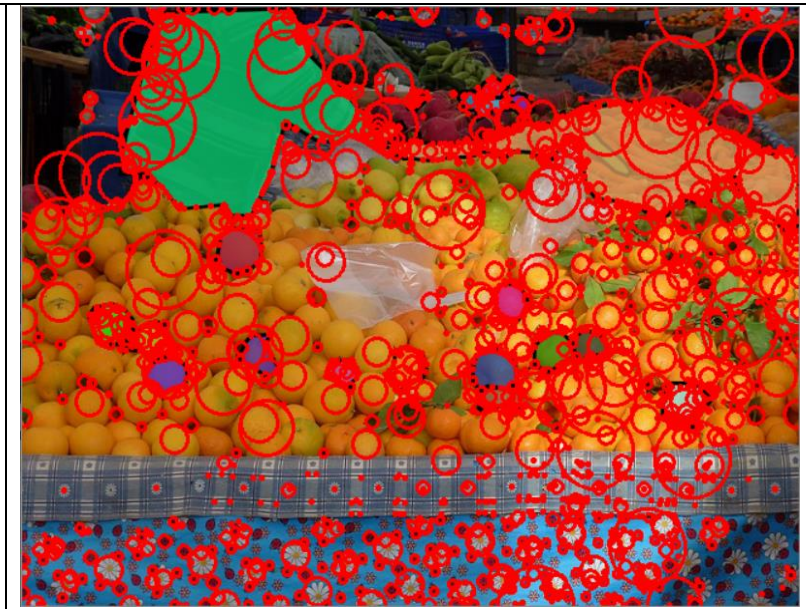
Ảnh kết quả chứa điểm Blob thực hiện bởi thuật toán Blob với tham số $\sigma = 1.0, k = \sqrt{2}$, $\alpha = 0.4$.



Ảnh gốc ban đầu



Ảnh kết quả chứa điểm Blob thực hiện bởi thuật toán Blob với tham số $\sigma = 1.0, k = \sqrt{2}, \alpha = 0.15$.

	
<p>Ảnh gốc ban đầu</p>	<p>Ảnh kết quả chứa điểm Blob thực hiện bởi thuật toán Blob với tham số $\sigma = 1.0, k = \sqrt{2}, \alpha = 0.2$.</p>

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 α khá nhiều, giá trị này dao động khác biệt rất lớn đối với ảnh đầu vào khác nhau, giá trị σ, k hầu như không cần thay đổi nhiều so với mọi ảnh truyền vào.

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

a. 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 sigma khác nhau (lưu ý rằng kích thước mỗi mặt nạ càng tăng dần với giá trị sigma 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 $\alpha \cdot \text{Max}(\text{giá trị bình phương của ma trận kết quả ứng với điểm Extrema hiện tại})$, α thường được chọn là $[0.1, 0.6]$, 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: σ, k, α .

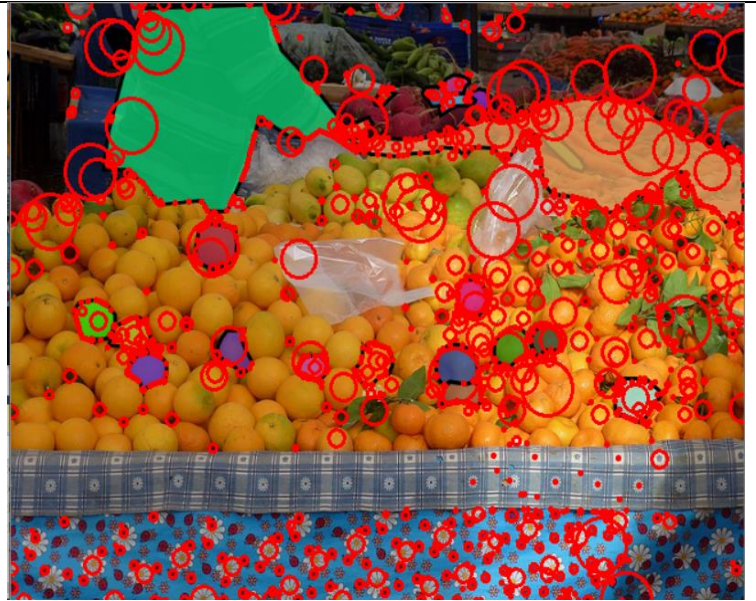
- Cài đặt lớp **BlobDetector**, với phương thức public `detectDOG` có nhiệm vụ thực hiện 4 bước trên và trả về mảng chứa tọa độ điểm blob của ảnh truyền vào.

b. 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 $\sigma = 1.0, k = \sqrt{2}, \alpha = 0.3$.



Ảnh gốc ban đầu

Ảnh kết quả chứa điểm Blob thực hiện bởi thuật toán DOG với tham số $\sigma = 1.0, k = \sqrt{2}, \alpha = 0.25$.



Ảnh gốc ban đầu

Ảnh kết quả chứa điểm Blob thực hiện bởi thuật toán DOG với tham số $\sigma = 1.0, k = \sqrt{2}, \alpha = 0.15$.

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 α 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 α khá nhiều, giá trị này dao động khác biệt rất lớn đối với ảnh đầu vào khác nhau, giá trị σ, k hầu như không cần thay đổi nhiều so với mọi ảnh truyền vào.

3.5 Thuật toán trích xuất đặc trưng SIFT.

a. 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 5×5 , $\sigma = 1.3$; 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 sigma 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 sigma khác nhau \Rightarrow có 4 ma trận DoG được sinh ra. Ngoài ra có thể truyền vào tham số $n_{octave}, n_{scales}, \sigma$.

Octave: $\sigma, k \times \sigma, k^2 \times \sigma, k^3 \times \sigma, k^4 \times \sigma$ (k em chọn mặc định là $\sqrt{2}$, giá trị σ khởi đầu của các octave là 1.6).

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

α . Max(giá trị **bình phương** của ma trận kết quả ứng với điểm Extrema hiện tại), α mặc định em chọn là 0.3. (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 $\frac{Trace(H)}{Det(H)} > \frac{(r+1)^2}{r}$ (ở đây em để mặc định giá trị r như trong paper là r=10).

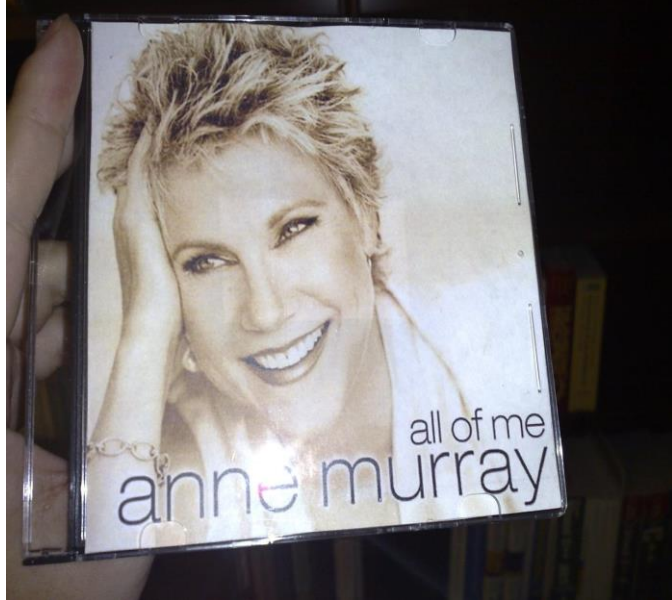
Bước 5 (Orientation Assignment): làm tương tự như trong slide, tại mỗi vị trí của điểm keypoint, duyệt các điểm hàng lân cận với window_size = 3, em tính giá trị magnitude và giá trị góc theta cho các điểm hàng xóm, sau đó tính giá trị weight của góc theta bằng cách lấy giá trị magnitude * với giá trị 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.

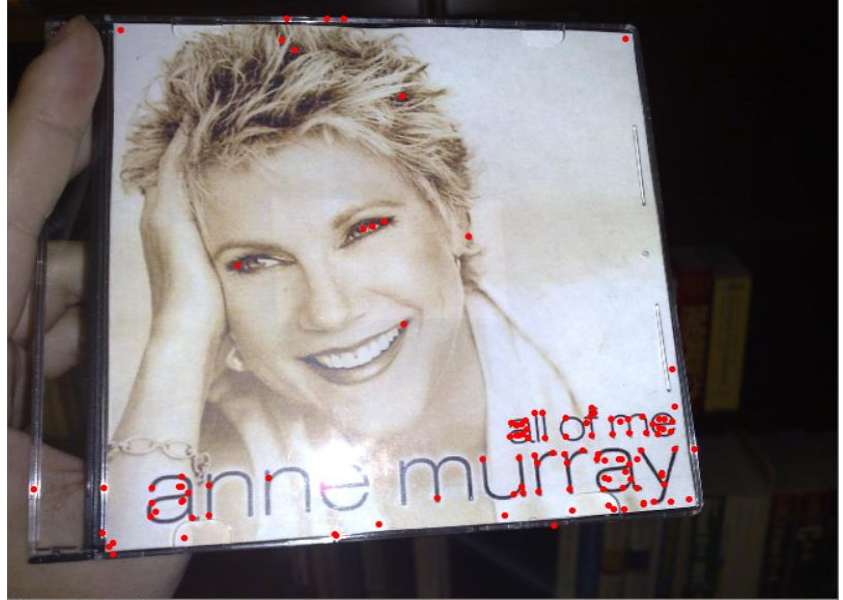
Cuối cùng, ta thu được mảng chứa tọa độ các điểm keypoint và vector đặc trưng 128 chiều ứng với mỗi điểm từ 6 bước trên.

Bài toán trên có các tham số được lựa chọn heuristic như sau: n_{octive} , n_{scales} , σ , k , thresh_edge, thresh_contrast, windowSize. Tuy nhiên để đơn giản, mã lệnh của chương trình chỉ bao gồm các tham số n_{octive} , n_{scales} , σ . (vì các tham số còn lại không có sự thay đổi lớn đối với ảnh input đầu vào).

b. Kết quả chạy thực nghiệm.



Ảnh gốc ban đầu



Ảnh kết quả chứa điểm keypoint SIFT với tham

số $n_{octive} = 4, n_{scales} = 5, \sigma = 1.6$.

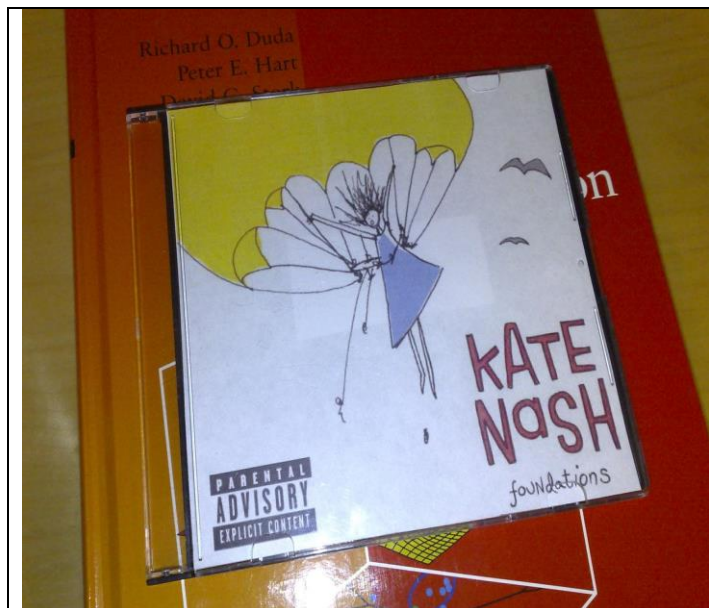


Ảnh gốc ban đầu



Ảnh kết quả chứa điểm keypoint SIFT với tham

số $n_{octive} = 4, n_{scales} = 5, \sigma = 1.6$.



Ảnh gốc ban đầu



Ảnh kết quả chứa điểm keypoint SIFT với tham

$$\text{số } n_{\text{octive}} = 4, n_{\text{scales}} = 5, \sigma = 1.6.$$

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 $n_{\text{octive}}, n_{\text{scales}}, \sigma$. Các giá trị em chạy thực nghiệm cho ra kết quả tốt nhất hiện tại là: $n_{\text{octive}} = 4, n_{\text{scales}} = 5, \sigma = 1.6$, các tham số còn lại được truyền sẵn trong mã nguồn chương trình.

3.6 Thuật toán matching hai ảnh.

a. 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 $< \text{threshold} = 0.8$ 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.

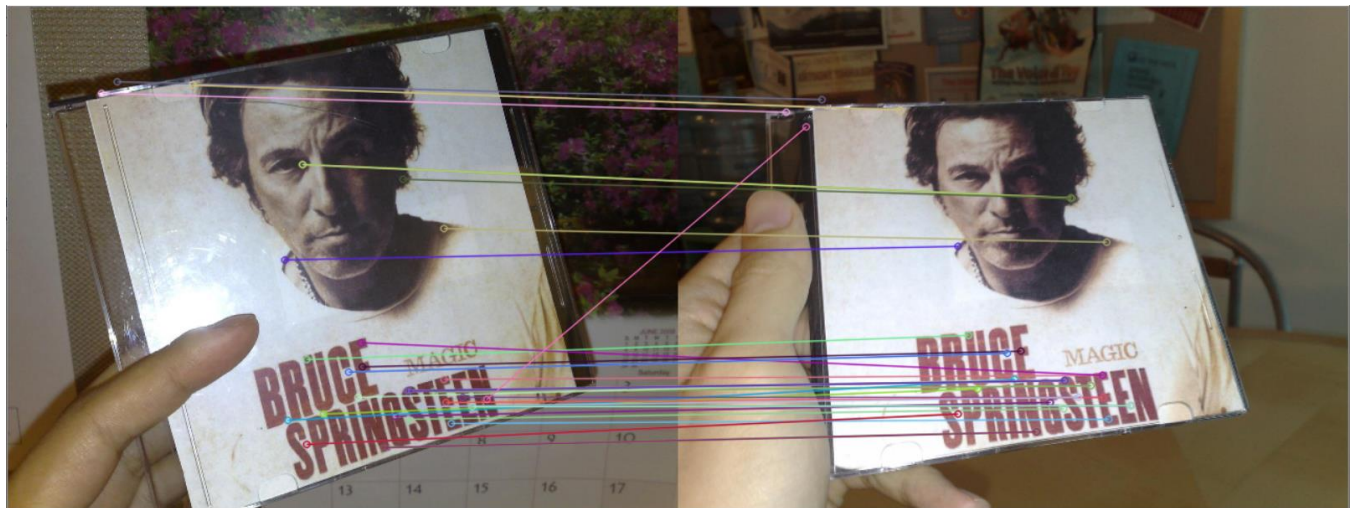
- Cài đặt lớp `SiftDetector`, với phương thức public `matchingTwoImages` có nhiệm vụ thực hiện việc matching ảnh train và ảnh test truyền vào, trong hàm `matchingTwoImages` có gọi hàm `siftDetector` có nhiệm vụ phát hiện các điểm keypoint của mỗi ảnh và tính các vector đặc trưng ứng với mỗi điểm đặc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

b. 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 tốt, matching các điểm khá khớp với nhau, tuy nhiên nếu 2 ảnh đầu vào có sự khác biệt quá lớn (góc xoay quá lớn) hoặc các điểm ảnh có màu và vùng xung quanh có màu gần tương tự nhau thì việc detect sẽ bị nhầm lẫn với một vài điểm.

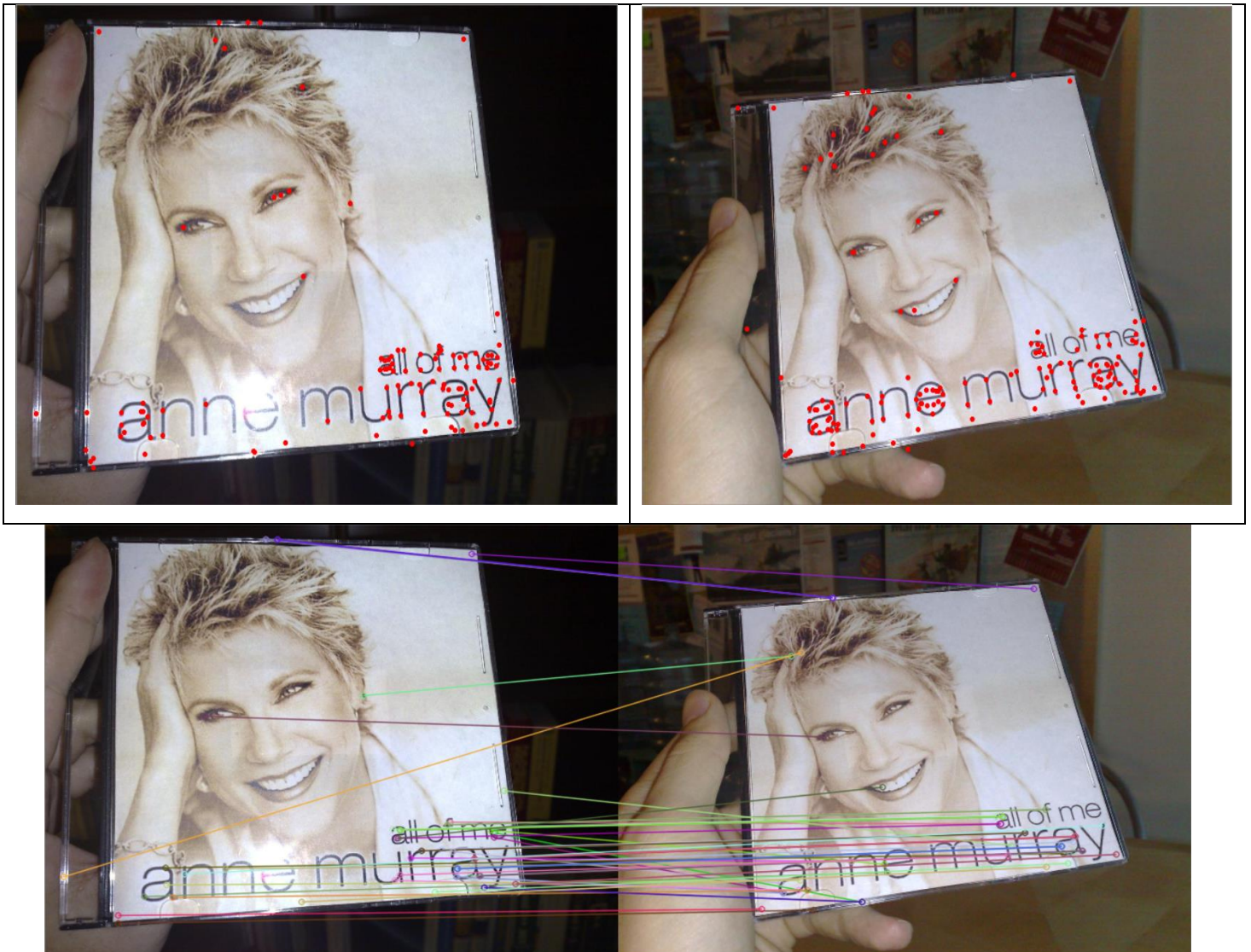
i. Các trường hợp chạy tốt.

1. Ảnh train: **06_3.jpg**; Ảnh test: **03.jpg**



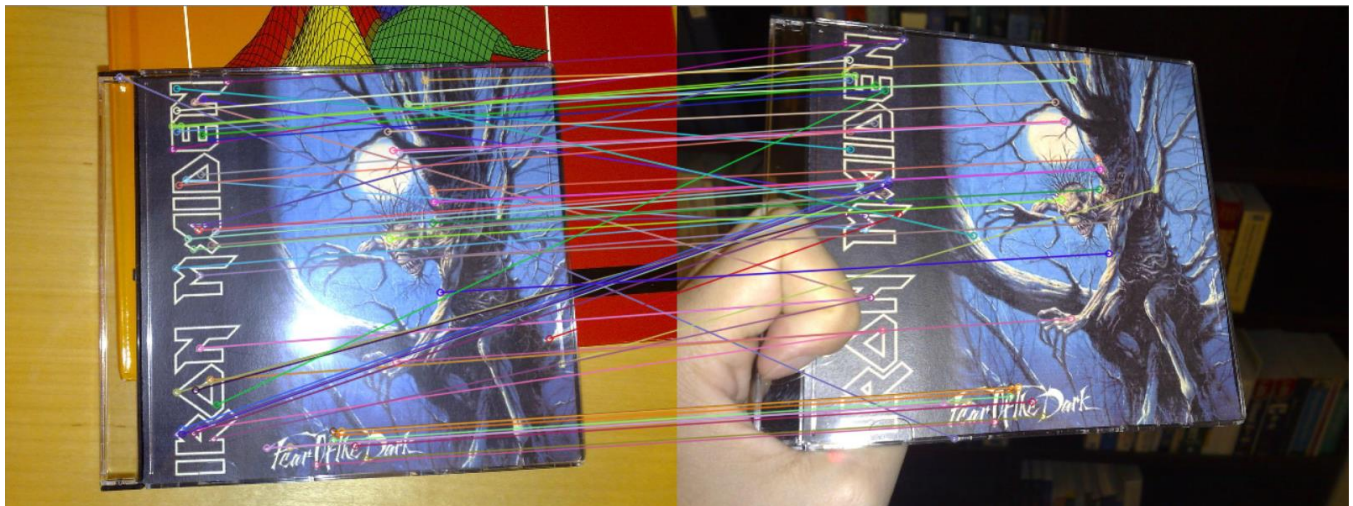
Nhận xét: Hai ảnh trên được matching khá khớp với nhau (chữ và đặc điểm trên khuôn mặt matching khớp dựa theo những đặc trưng SIFT đã phát hiện được), tuy nhiên ảnh hưởng bởi background và các điểm ảnh có màu và vùng xung quanh có màu tương tự nhau nên có thể bị matching nhầm ở số điểm (ví dụ điểm màu vàng ở mắt ảnh bên trái).

2. Ảnh train: 02_2.jpg; Ảnh test: 02_3.jpg



Nhận xét: Hai ảnh trên được matching khá tốt (chữ và đặc điểm trên khuôn mặt matching khớp dựa theo những đặc trưng SIFT đã phát hiện được, mặc dù các điểm trên khuôn mặt còn hơi ít). Màu sắc của tóc, background và khuôn mặt khác nhau nên những điểm trong những vùng này còn nhầm lẫn matching một số ít. Vùng chữ có màu sắc riêng được matching rất khớp với nhau.

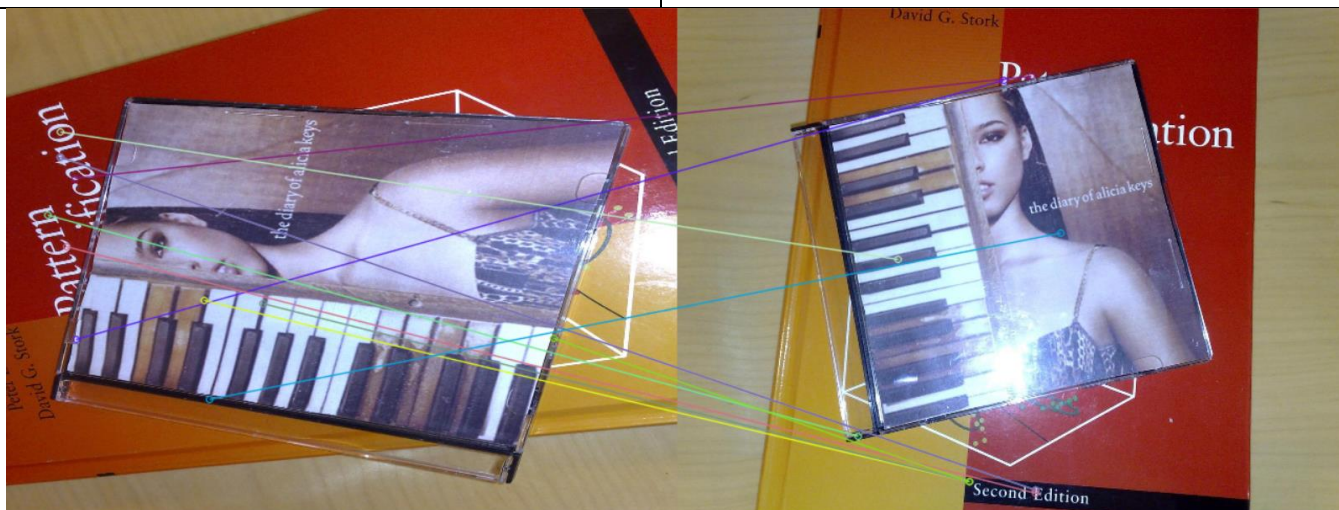
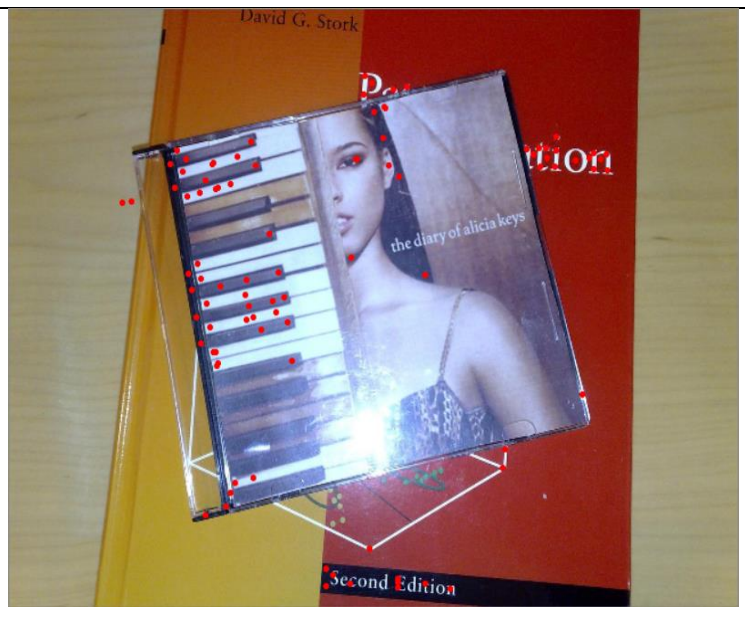
3. Ảnh train: 13_2.jpg; Ảnh test: 13_1.jpg



Nhận xét: Các điểm đặc trưng SIFT được phát hiện bởi 2 ảnh là khá nhiều, matching được rất nhiều điểm nên hình ảnh matching khá rối, tuy nhiên nếu để ý kỹ thì việc matching rất khớp, các vùng chữ khớp với nhau, hình mặt trăng và quỷ ở hai ảnh cũng khớp với nhau, số điểm matching đúng nhiều hơn số điểm matching không đúng.

ii. Các trường hợp chạy không tốt.

1. Ảnh train: 01_1.jpg; Ảnh test: 01_2.jpg



Nhận xét: Mặc dù hai ảnh trên mỗi ảnh đều phát hiện các điểm keypoint theo đặc trưng SIFT rất tốt nhưng khi matching lại với nhau thì kết quả không tốt lắm. Các vùng chữ màu trắng ảnh bên trái bị matching nhầm với vùng piano màu trắng ảnh bên phải, màu đen phím piano bên trái matching nhầm với vùng tóc của người bên phải. Có thể thấy ảnh hưởng của nhiễu (các điểm ảnh có màu gần giống nhau và vùng xung quanh nó cũng có màu gần giống nhau) là khá nghiêm trọng với việc matching, ngoài ra việc ảnh bị xoay khá lớn cũng khiến các vector đặc trưng ứng với mỗi điểm ảnh bị thay đổi sai số khá nhiều.

2. Ảnh train: 06_3.jpg; Ảnh test: 03.jpg



Nhận xét: Lại một lần nữa, việc matching lại không tốt trong khi việc phát hiện các đặc trưng SIFT cho kết quả khá tốt. Nguyên nhân thất bại ở 2 ảnh này có thể do chữ giống nhau quá nhiều hay mặt khác các điểm ảnh có màu gần giống nhau và vùng xung quanh cũng có màu gần giống nhau, việc matching các chữ với nhau không ăn khớp (chẳng hạn điểm keypoint ở chữ R bên ảnh trái matching với điểm keypoint ở chữ S ảnh bên phải, v.v.).

4 Đánh giá kết quả hoàn thành.

Yêu cầu	Mức độ hoàn thành	Ghi chú
Cài đặt thuật toán phát hiện điểm góc HarrisDetector.	100%	Kết quả phát hiện tốt, có hiển thị demo phát hiện đặc trưng trên bản gốc.
Cài đặt thuật toán phát hiện điểm Blob bằng BlobDetector	100%	Kết quả phát hiện tốt, có hiển thị demo phát hiện đặc trưng trên bản gốc.
Cài đặt thuật toán phát hiện điểm Blob bằng DogDetector.	100%	Kết quả phát hiện tốt, có hiển thị demo phát hiện đặc trưng trên bản gốc.
Cài đặt thuật toán phát hiện đặc trưng SIFT của ảnh.	100%	Kết quả phát hiện tốt, có hiển thị demo phát hiện đặc trưng trên bản gốc.
Cài đặt thuật toán matching 2 ảnh sử dụng thuật toán KNN.	100%	Sử dụng hàm knnMatch của thư viện OpenCV, việc matching tùy vào ảnh đầu vào và phải điều chỉnh khá nhiều tham số đối với mỗi ảnh để cho ra kết quả tốt.
Chạy thử nghiệm đối sánh các phương pháp trên bộ ảnh train và ảnh test	100%	Đã hoàn thành và trích vào báo cáo những trường hợp chạy tốt, những trường hợp không chạy tốt, đồng thời nhận xét về các kết quả đạt được.

5 Tài liệu tham khảo

- 5.1. Slide bài giảng môn Thị Giác Máy Tính, Đại Học Khoa Học Tự Nhiên TP HCM.
- 5.2. Bài giảng lecture 06 – SIFT – UCF Computer Vision Lectures 2014.
- 5.3. [Distinctive Image Features from Scale-Invariant Keypoints - David G. Lowe, 2004.](#)
- 5.4. [Blog medium: Sift Implementation in Python.](#)
- 5.5. [Blog VietCS bài viết về Harris Detector Corner Python.](#)
- 5.6. [Blog ProjectsFlix bài viết về Laplacian Blob Detector Using Python.](#)