ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

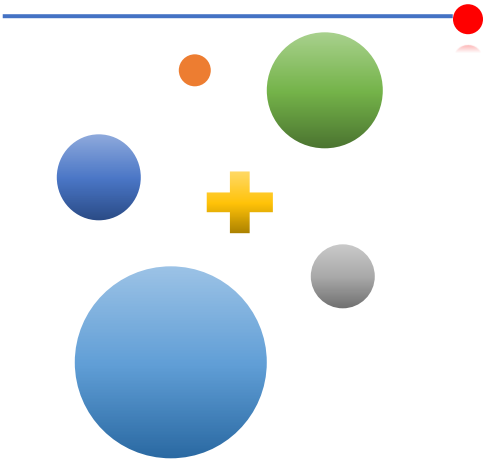
TRƯỜNG ĐẠI HỌC KHOA HOC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

THỊ GIÁC MÁY TÍNH

Báo cáo BTVN-03

Đặc trưng cục bộ



GVHD: Võ Hoài Việt

Contents

A Thành viên nhóm:

STT	MSSV	Họ tên	SĐT	Email
1	1612174	Phùng Tiến Hào	0933642694	tienhaophung@gmail.com
2	1612269	Võ Quốc Huy	01258378481	voquochuy304@gmail.com

B Mức độ hoàn thành:

Mã lệnh	Yêu cầu	Tên hàm đề nghị	Ghi chú	Mức độ hoàn thành (%)
1	Phát hiện điểm đặc trưng sử dụng thuật toán hariss và hiển thị điểm ảnh gốc	Mat detectHarrist(Mat img, …)		100
2	Phát hiện điểm đặc trưng sử dụng thuật toán blob và hiển thị điểm đặc trưng trên ảnh gốc	Mat detectBlob(Mat img, …)		100
3	Phát hiện điểm đặc trưng sử dụng thuật toán DOG và hiển thị điểm đặc trưng trên ảnh gốc	Mat detectDOG(Mat img, …)		100
4	Đối sánh 2 ảnh sử dụng đặc trưng SIFT với thuật toán KNN	Double matchBySIFT(Mat img1, Mat img2, int detector, …)	Sinh viên tìm hiểu thuật toán KNN trong thư mục samples của OpenCV và viết báo cáo	50
5	Thực nghiệm đối sánh các phương pháp trên tập dữ liệu ảnh bìa CD/DVD. Đánh giá kết quả và nêu nhận xét về các thuật toán trên.			100

	Download Dữ liệu: https://drive.google.com/open?id=1EjOew0oXgnz5aeqPpNbIB8IrpUTnPdT1
Tổng cộng	

C Báo cáo:

Một vài lưu ý:

- Ngôn ngữ sử dụng: Python
- Các thư viện sử dụng như: Numpy, openCV, matplotlip và skimage.
- Định nghĩa ký hiệu theo tài liệu nước ngoài:
  - X: Vertical axe
  - Y: Horizontal axe
  - XY: Magnitude của cả 2 hướng trên

I. So sánh kết quả thực hiện với OpenCV:

1. Thuật toán Haris:

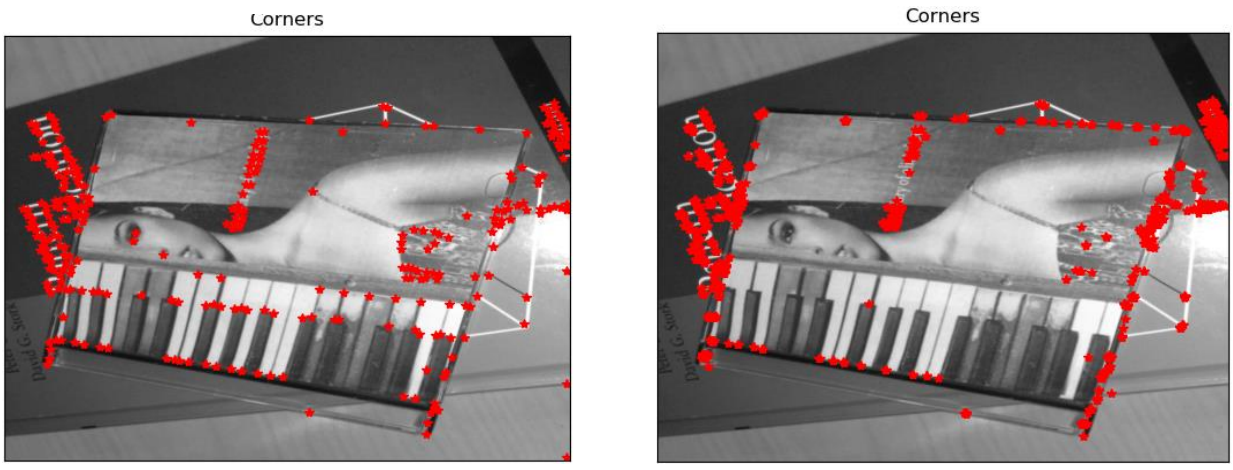


Figure 1 Handcraft (ratio used 0.1) vs OpenCV (ratio used 0.01)

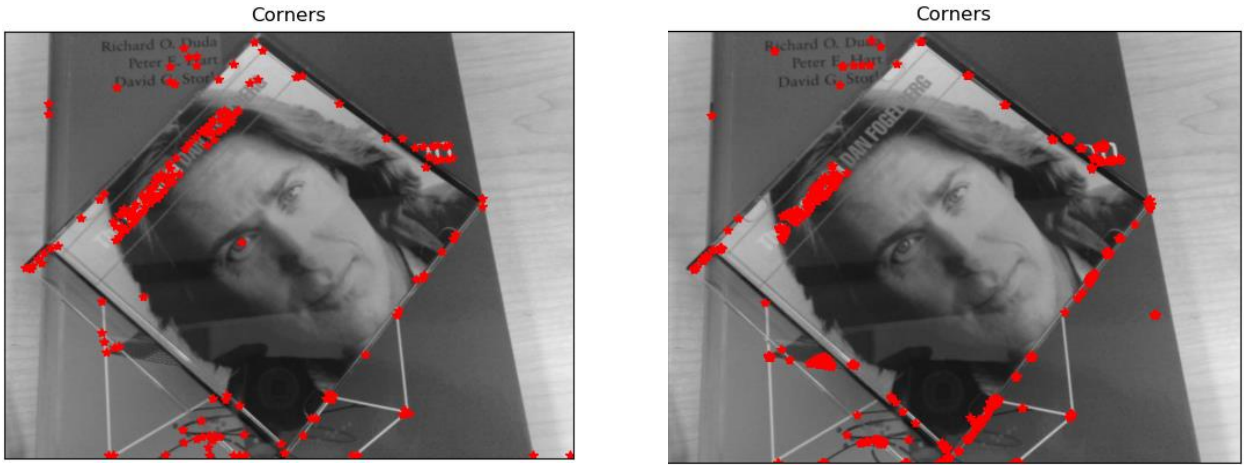


Figure 2 Handcraft (ratio used 0.1) vs OpenCV (ratio used 0.01)

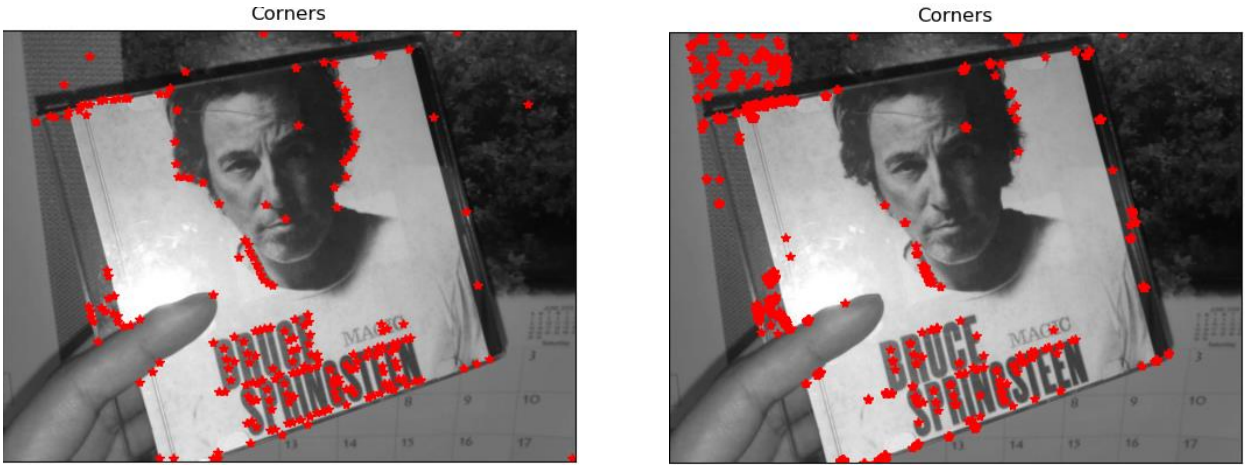


Figure 3 Handcraft (ratio used 0.1) vs OpenCV (ratio used 0.01)

Figure 4 Figure 3 Handcraft (ratio used 0.15) vs OpenCV (ratio used 0.1)

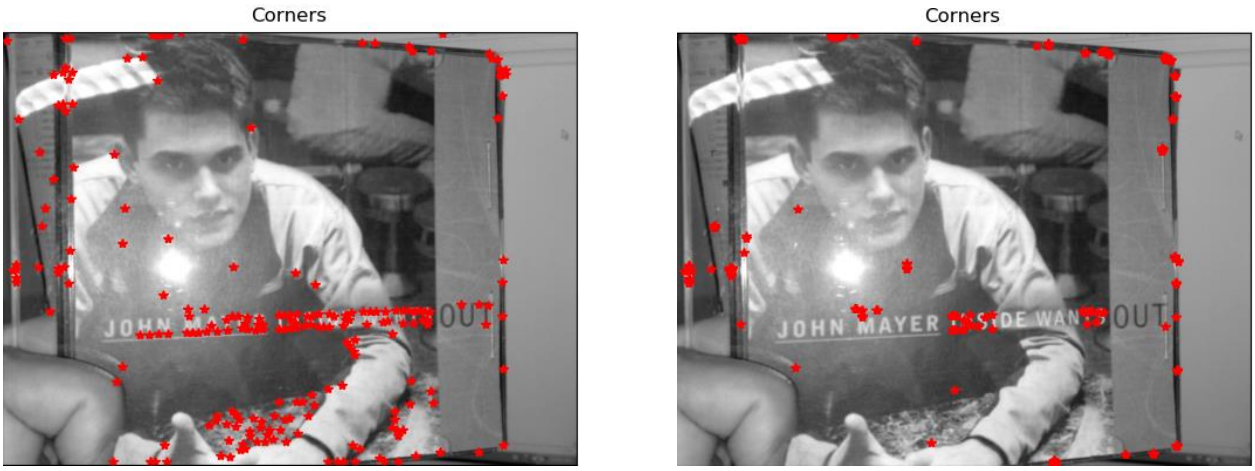


Figure 5 Handcraft (ratio used 0.15) vs OpenCV (ratio used 0.03)

Nhận xét:

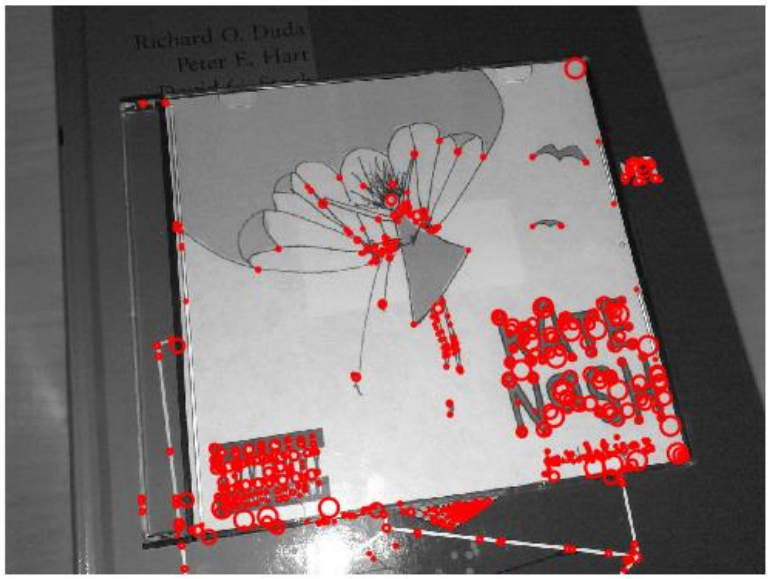
- Thuật toán Harris phát hiện tốt các corner
- Thời gian chạy trung bình của thuật toán chúng tôi qua 5 lần chạy: 73.908s và của OpenCV: 0.818s . Điều này cho thấy, thuật toán của chúng tôi vẫn chưa xử lý nhanh và tối ưu về mặt bộ nhớ như OpenCV. Bù lại, kết quả quả của chúng tôi vẫn cho kết quả tốt và có những ảnh chính xác hơn OpenCV.
- Ở đây, chúng tôi dùng Harmonic mean - Brown, Szeliski, and Winder (2005) để tính response của các điểm vì để khử đi sự phụ thuộc vào alpha của Harris:

$$R = \frac{\det(H)}{trace(H)} \text{ , với } H\text{: } Hessian\ matrix$$

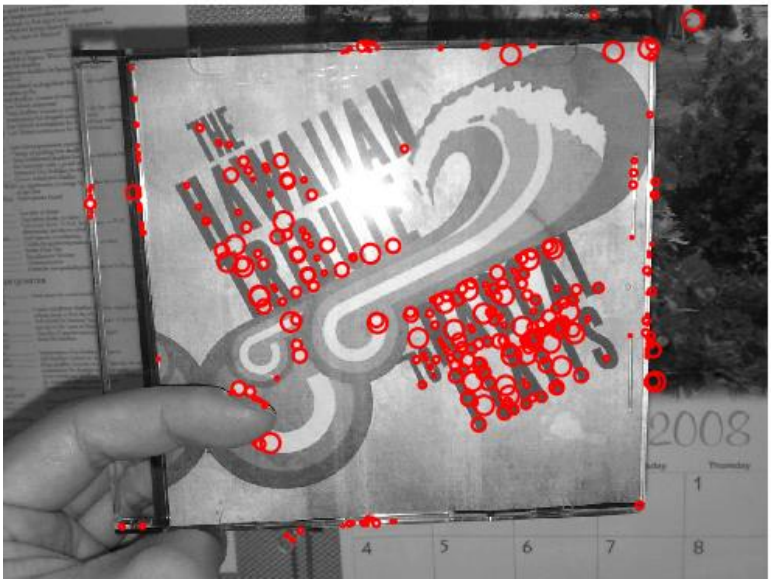
- Theo kết quả chạy, chúng tôi chọn ratio = 0.1 làm tiêu chuẩn để threshold keypoints và cho được kết quả tốt: T = ratio * max(R)
- Chúng tôi lọc ra các keypoint bằng việc lấy ý tưởng của Shi & Tomashi trong việc sắp xếp các corner theo mức độ response giảm dần. Như vậy các corner với độ phản hồi mạnh sẽ được ưu tiên xét và thêm nữa, trong vòng lân cận khoảng cách tối thiểu min_dist = 10 sẽ không có corner khác xuất hiện. Điều này, giúp phát hiện được good keypoints tốt hơn và đặt được kết quả tốt hơn.

2. Thuật toán Blob dùng Lablacian of Gaussian:

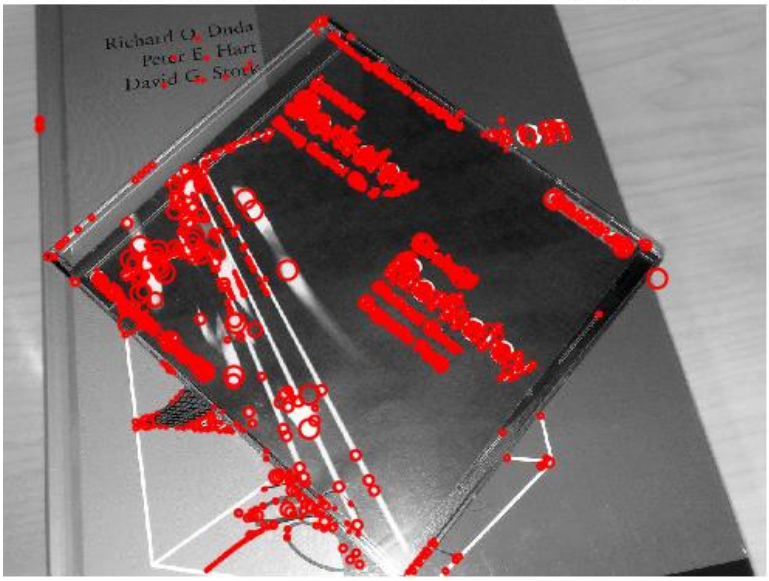
Blob detector used Laplacian of Gaussian



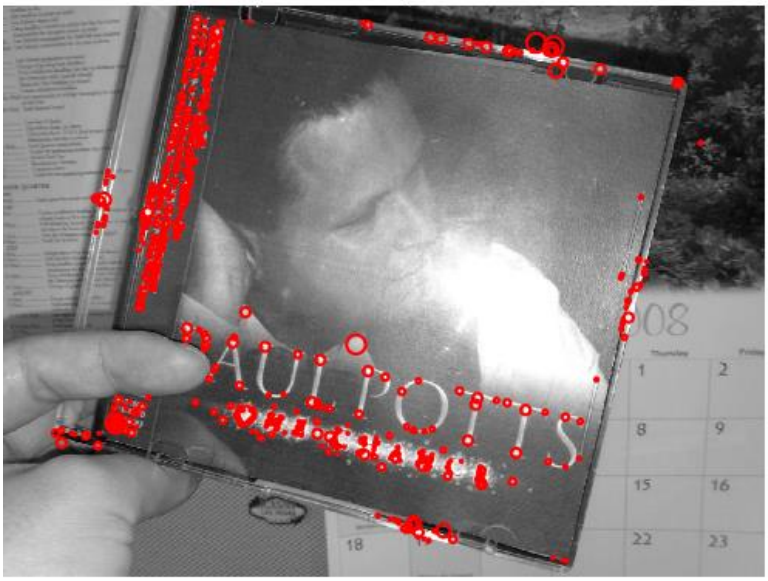
Blob detector used Laplacian of Gaussian

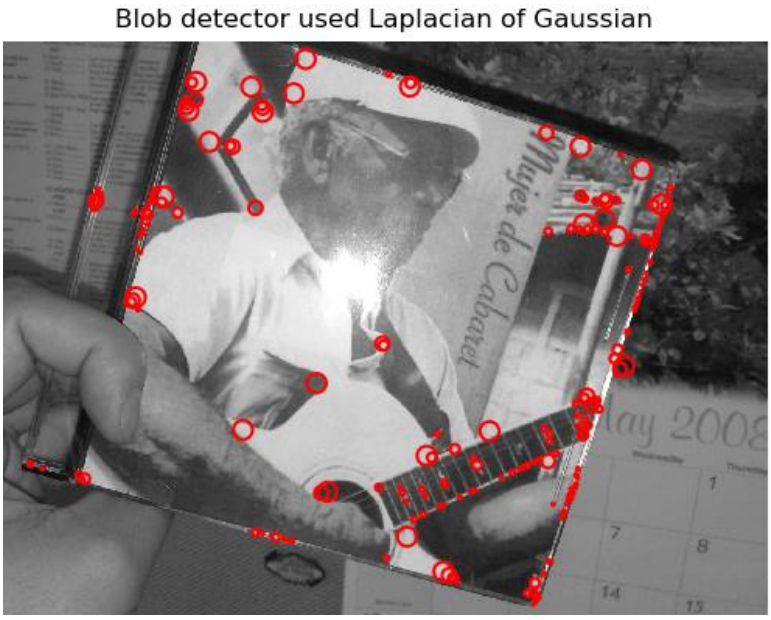


Blob detector used Laplacian of Gaussian



Blob detector used Laplacian of Gaussian





Nhận xét:

- Thuật toán phát hiện blob sử dụng LoG đạt được kết quả tốt, đúng như mong đợi.
- Thời gian chạy trung bình của 5 lần chạy: 228s. Do kích thước của ảnh khá lớn nên việc detect và xử lý phải nhiều hơn dẫn đến thời gian xử lý khá lâu. Tuy rằng chúng tôi đã cố gắng hết sức để cải thiện vấn đề này.
- Bằng kết quả trải nghiệm, chúng tôi chọn số scale của LoG là 9. Các scale được tính bằng $\sqrt{2} * sigma, với$ initial sigma = 1.0 và chúng tôi chọn $kernel\ size = 2 * ceil(3 * sigma) + 1$ điều này cho kết quả rất tốt.
- Bên cạnh đặt ngưỡng CONTRAST_THRESHOLD = 0.03 để loại bỏ các low contrast chúng tôi có mượn ý tưởng của SIFT loại bỏ thêm các edge bằng cách thức tương tự như Harris bằng cách cách sau:

$$\frac{Trace(H)^2}{Det(H)} > \frac{(r+1)^2}{r}, với\ r = 10\ (theo\ bài\ báo)$$

- Điều này giúp giảm thiểu và xác định chính xác được blob với các blob.

3. Thuật toán Blob dùng Differece of Gaussian:

Blob detector used Difference of Gaussian

Blob detector used Difference of Gaussian

Blob detector used Difference of Gaussian

Nhận xét:

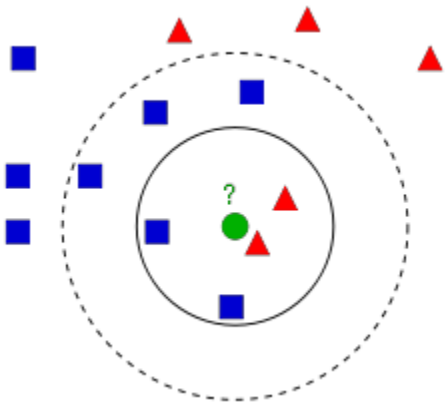
- Kết quả của thuật toán Blob dùng DoG cho kết quả tốt hơn và phát hiện được nhiều blob hơn so với LoG.
- Thời gian chạy trung bình cho 3 lần: 400s. Ta thấy được thời gian chạy khá lâu.
- Việc cài đặt DoG hoàn toàn dựa trên ý tưởng của SIFT bằng việc chọn số octave = 5, số scale = 4.
- Initial sigma = 1.0 cho octave thứ nhất và bằng $next_{sigma} = \sqrt{2} * sigma$ trong cùng 1 octave.
- Các scale của octave kế tiếp sẽ bằng gấp đôi của các scale octave trước.

II. Báo cáo tìm hiểu về kNN (k-nearest neighbors):

kNN là một trong những thuật toán phân lớp của supervised learning.

Ý tưởng: chọn ra điểm phù hợp nhất (closest match) của test data trong không gian.

VD: Ta có không gian 2D gồm các điểm và có gán nhãn lớp (gồm 2 lớp: hình vuông và hình tam giác)



Nói cách khác, trong hình chứa 2 lớp: hình vuông xanh và hình tam giác màu đỏ. Không gian này được gọi là không gian đặc trưng (feature space).

Các điểm trong không gian 2D này được gọi là feature.

Giả sử, chúng ta có 1 điểm mới xuất hiện (hình tròn), việc chúng ta là cần phân loại điểm này vào lớp thích hợp (Vuông hoặc tam giác).

Thì 1 cách thức là kiểm tra láng giềng gần nhất của điểm tròn đó. Nếu nó gần với điểm lớp hình vuông thì nó thuộc về hình vuông và ngược lại.

Trong ví dụ, ta thấy nó gần với điểm thuộc lớp tam giác nhất nên ta nói nó thuộc lớp tam giác.

Đây gọi là phương pháp Nearest neighbor.

Nhưng liệu rằng như vậy có hợp lý vì có thể điểm thuộc lớp tam giác là gần nhất nhưng lại có nhiều điểm thuộc lớp vuông xung quanh hơn.

Cho thấy rằng, lớp hình vuông có phân bố mạnh ở lân cận điểm tròn hơn. Do vậy, nếu chỉ lấy 1 điểm gần nhất để xác định là chưa đủ.

Thay vào đó, chúng ta sẽ chọn ra k nearest points để kiểm tra. Nếu số láng giềng lân cận thuộc về lớp nào nhiều nhất thì điểm tròn sẽ được gán vào lớp đó.

VD: Ta chọn k = 3 thì có được 2 điểm tam giác và 1 điểm hình vuông (Do 2 vuông có cùng khoảng cách nên chỉ lấy 1).

Ta thấy rằng lớp tam giác chiếm đa số nên ta nói điểm đó thuộc về lớp tam giác.

Nhưng nếu ta, chọn k = 7 thì thế nào? Ta có 5 thành viên vuông và 2 thành viên đỏ. Thế nên ta kết luận nó thuộc về lớp vuông.

Như vậy, ta thấy rằng việc điểm mới đó thuộc về lớp nào phụ thuộc vào cách ta chọn k.

Một lưu ý, để tránh tình trằng cả 2 thành viên bằng nhau khi xét thì ta nên chọn k là số lẻ.

Phương pháp này gọi là k-Nearest Neighbour.

Vấn đề nữa lại xảy ra, lỡ như trường hợp k láng giếng có số thành viên lớp này bằng số thành viên lớp kia thì làm sao để xác định.

VD: k = 4 thì ta có 2 vuông và 2 tam giác. Nhưng ta lại thấy rằng, 2 tam giác nó gần với điểm tròn hơn. Do đó, ta có đủ chứng cứ để kết luận điểm đó thuộc về lớp tam giác.

Phương pháp này được gọi là modified kNN.

Ta sẽ đánh trọng số cho các thành viên láng giềng phụ thuộc vào khoảng cách của nó để điểm đang xét. Nếu thành viên nào gần với điểm đáng xét thì đánh trọng số cao hơn và ngược lại.

Lớp nào mà có tổng trọng số cao nhất thì điểm đang xét thuộc về lớp đó.

Một số điểm lưu ý:

- Ta cần phải có thông tin trước về các lớp.
- Phải cần sử dụng rất nhiều bộ nhớ và thời gian để xử lý tính toán cho các dữ liệu huấn luyện.
- Nếu số dữ liệu huấn luyện lớn thì đây là vấn đề.

Các ứng dụng của kNN:

- Face regconition
- Face detection
- Matching features
- Data mining

III. Hướng dẫn sử dụng chương trình:

- Command line:

  python <tenchuongtrinh> -i <duongdandentaptinanh> -c <malenh>

  (-i, -c: Argument Parser của Python)

- Để trợ giúp: gõ “python 1612174_1612269_Lab03.py -h”

```
(opencv-env) E:\K16\Junior\TGMT\OpenCV-Lab02>python 1612174_1612269_Lab03.py -h
usage: 1612174_1612269_Lab03.py [-h] -i INPUT -c CODE

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Path to input image
  -c CODE, --code CODE  Code action
```

- Các mã lệnh:
  - 1: Harris
  - 2: Blob
  - 3: DoG
  - 4: SIFT

- Ví dụ: Dùng Harris để phát hiện biên cạnh cho ảnh empire.jpg “python 1612174_1612269_Lab03.py -i E:\K16\Junior\TGMT\OpenCV---Lab01\Images\lena.png -c 4”

```
(opencv-env) E:\K16\Junior\TGMT\OpenCV-Lab02>python 1612174_1612269_Lab03.py -i empire.jpg -c 1
E:\K16\Junior\TGMT\OpenCV-Lab02\harris.py:140: RuntimeWarning: invalid value encountered in true_divide
  R = np.nan_to_num(Hdet/Htr)
Time: 2.68(s)
```

Lưu ý: Chương trình có trả về thời gian xử lý cho 1 hoạt động.

D Tham khảo: