Báo cáo project 1

Đề tài: Object detection with SIFT

Thực hiện: Nguyễn Huy Hoàng

MSSV: 20173132

Lớp: KHMT 05-K62

GVHD: Nguyễn Thị Oanh

Đại học Bách Khoa Hà Nội, 9-2020

Nội dung

[1. Giới thiệu đề tài 3](#_Toc50567986)

[2. Phân tích yêu cầu 3](#_Toc50567987)

[3. Giải pháp thực hiện 4](#_Toc50567988)

[3.1. SIFT 4](#_Toc50567989)

[3.2. Features matching 5](#_Toc50567990)

[3.3. Object locating 6](#_Toc50567991)

[4. Triển khai chương trình 7](#_Toc50567992)

[4.1. Import thư viện 7](#_Toc50567993)

[4.2. Đọc file ảnh 7](#_Toc50567994)

[4.3. Tìm các keypoint và descriptor với SIFT 7](#_Toc50567995)

[4.4. Features matching 8](#_Toc50567996)

[4.5. Object locating 9](#_Toc50567997)

[4.5. Detect multiple instances of object 10](#_Toc50567998)

[5. Kết luận 12](#_Toc50567999)

# Giới thiệu đề tài

Nhận diện vật thể (Object Detection) là một vấn đề quan trọng trong Thị giác máy tính (Computer Vision) nói riêng và trong Trí tuệ nhân tạo (Artificial Intelligence) nói chung. Object Detection đề cập đến khả năng của hệ thống máy tính và phần mềm để định vị các đối tượng trong một hình ảnh. Object Detection có nhiều ứng dụng, trong đó phổ biến là phát hiện khuôn mặt, phát hiện xe, đếm số người đi bộ, hệ thống bảo mật và xe không người lái. Vì thế, việc tìm hiểu và nghiên cứu vấn đề này là cần thiết cho sinh viên Khoa học máy tính.

Bài toán Object Detection về cơ bản là việc cần xác định xem một vật thể có nằm trong một bức ảnh hay không. Một chương trình Object Detection đơn giản nhận đầu vào là một bức ảnh chứa vật thể, và một bức ảnh khác chứa hỗn độn các vật thể trong một khung cảnh nào đó. Đầu ra của chương trình là cần xác định vật thể có hay không nằm trong khung cảnh, và nếu có thì cần xác định vị trí của chúng nằm ở đâu.

Với công nghệ hiện nay, có nhiều phương pháp để giải quyết bài toán này. Tuy nhiên đề tài này chỉ đề cập đến phương pháp sử dụng giải thuật SIFT (Scale-Invariant Feature Transform) để phát hiện vật thể. Chương trình Object Detection được triển khai bằng ngôn ngữ Python với thư viện OpenCV. Ngoài ra, chương trình còn sử dụng một số thư viện hỗ trợ khác (numpy, matplotlib, sklearn,...)

# Phân tích yêu cầu

Ta cần tạo ra một chương trình nhận đầu vào là hai bức ảnh, một là bức ảnh chứa vật thể (query image), hai là bức ảnh bối cảnh (train image).



Hình : Query image

Để chương trình hoạt động chính xác, cần crop bức ảnh query image sao cho sát với vật thể, ngoài ra bức hình nên chụp rõ nét, ánh sáng đầy đủ. Đối với mỗi vật thể, chỉ cần một query image.



Hình : Train image

Train image là bức ảnh chứa một hoặc nhiều vật thể trong query image, hoặc cũng có thể không chứa vật thể nào. Yêu cầu đầu ra của chương trình chính là xác định xem có vật thể trong trong train image. Và nếu có, cần xác định rõ vị trí của tất cả các thể hiện của nó.



Hình : Ví dụ về output của chương trình

# Giải pháp thực hiện

## 3.1. SIFT

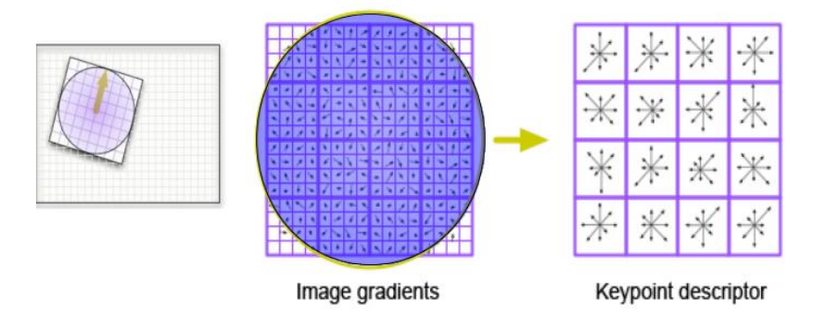
Mấu chốt của chương trình Object Detection này nằm ở phép biến đổi đặc trưng bất biến theo tỉ lệ (SIFT). Vậy SIFT là gì? SIFT là một thuật toán phát hiện đặc trưng (feature detection) trong thị giác máy tính dùng để mô tả và phát hiện các feature cục bộ trong hình ảnh. Nó được công bố bởi David Lowe vào năm 1999. Các ứng dụng bao gồm nhận dạng đối tượng, lập bản đồ và điều hướng bằng robot, ghép hình ảnh, mô hình 3D, nhận dạng cử chỉ, theo dõi video, nhận dạng cá thể động vật hoang dã và khớp chuyển động.



Hình : Ví dụ về feature detection – nguồn: wikipedia

Với đầu vào là một hình ảnh, chạy qua thuật toán SIFT, ta thu được các keypoint. Mỗi đối tượng trong hình ảnh sẽ cho ra rất nhiều các keypoint khác nhau, để ta phân biệt được các keypoint này với nhau sẽ thông qua một vector 128 chiều hay còn gọi là descriptor. Các descriptor này sẽ được dùng để nhận dạng đối tượng trong ảnh, hay dùng cho các bài toán classication.

Hình ảnh sau khi áp dụng biến đổi SIFT, ứng với mỗi keypoint ta sẽ thu được: toạ độ keypoint, scale và orientation của keypoint, descriptor. Các mũi tên trong hình dưới vẽ nhờ vào scale và orientation.



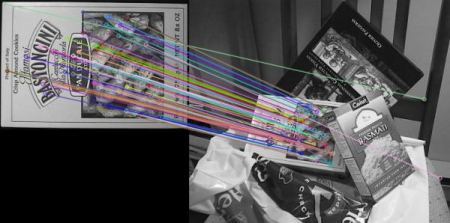
Hình : Minh hoạ về keypoint và descriptor

Đặc điểm :

* Các keypoint sẽ ít bị phụ thuộc bởi cường độ sáng, nhiễu, góc xoay của ảnh do các descriptor được tạo ra từ gradients do đó nó đã bất biến với các thay đổi về độ sáng.
* Nhanh và hiệu quả, tốc độ xử lý gần như với thời gian thực (realtime)
* Có thể xử lý khi xoay ảnh

## 3.2. Features matching

Sau khi thu được các feature (chính là keypoint và descriptor của chúng) trong query image cũng như train image, việc tiếp theo cần làm đó là đối sánh các feature (features matching) giữa query image và train image.



Hình : Features matching

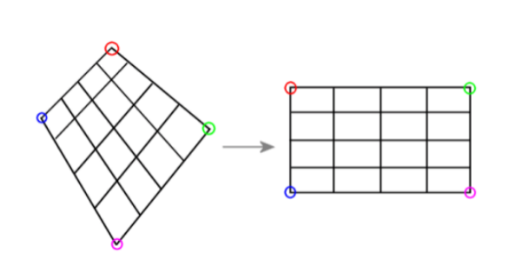
Về cơ bản, features matching chính là việc tìm ra các cặp keypoint giống nhau (hoặc gần giống) trong query image và train image. Từ đó có thể xác định vật thể nằm ở đâu trong hình.

Có hai phương pháp chính để thực hiện công việc này: Brute-Force matching hoặc FLANN(Fast Library for Approximate Nearest Neighbors) matching. Ở đây ta sử dụng phương pháp thứ nhất, Brute-Force: Duyệt toàn bộ tất cả các keypoint trong query image, đối với mỗi keypoint, tìm ra hai keypoint (thuộc train image) có khoảng cách gần nó nhất. Khoảng cách ở đây chính là khoảng cách euclid của 2 vector descriptor 128 chiều. Sau đó tính tỉ lệ của khoảng cách gần nhất và khoảng cách gần thứ hai, nếu như nó nhỏ hơn một ngưỡng cho trước (0.8 chẳng hạn) thì cặp so khớp này được gọi là good match, ta lưu lại nó. Ngược lại, cặp so khớp này được gọi là bad match, ta loại bỏ nó.

## 3.3. Object locating

Công việc cuối cùng là cần xác định xem vật thể có nằm ở trong hình hay không, và nếu có thì nằm ở đâu. Mỗi vật thể thường có một số lượng keypoint nhất định. Nếu số lượng cặp match nhỏ hơn quá nhiều, thì coi như không đủ tin cậy để tìm ra vật thể, ta kết luận là không tìm thấy vật thể. Ngược lại, ta sẽ xác định vị trí của vật thể. Để làm được việc này, ta cần sử dụng đến khái niệm Homography và thuật toán RANSAC.

Trong toán học, Homography là sự dịch chuyển sử dụng phép chiếu hình học, hay nói cách khác nó là sự kết hợp của cặp điểm trong phép chiếu phối cảnh. Ảnh thực trong không gian ba chiều có thể biến đổi về không gian ảnh bằng phép chiếu thông qua ma trận biến đổi Homography hay còn gọi là ma trận H. Các phép chiếu biến đổi thông qua ma trận Homography không đảm bảo về kích thước và góc của vật được chiếu, nhưng lại đảm bảo về tỉ lệ.



Hình : Homography

Thuật toán RANSAC (Random sample consensus) là một phương pháp lặp để tìm ra ma trận homography tối ưu nhất cho tập các cặp so khớp (match) một cách hiệu quả.

Sau khi tìm ra ma trận homography, ta sử dụng nó để tìm ra 4 cặp điểm tương ứng với 4 góc của query image, nối chúng lại để đánh dấu vị trí của vật thể.

Vấn đề được đặt ra là nếu như có nhiều hơn một thể hiện của vật thể trong train image, thì ta sẽ giải quyết như thế nào? Lúc này ta sẽ phải dùng thêm một phương pháp nữa, gọi là phân cụm Mean Shift. Dựa vào đặc điểm của các keypoint thường nằm co cụm lại trên vật thể, nên khi ta phân cụm các keypoint trong bức ảnh train image, ta sẽ thu được các vùng khác nhau chứa riêng rẽ các vật thể. Sau đó ta mới thực hiện matching tìm kiếm vật thể query image trong từng cụm.

# Triển khai chương trình

## 4.1. Import thư viện

**import** cv2 **as** cv  
**from** sklearn.cluster **import** MeanShift, estimate\_bandwidth  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**import** math  
**import** random

## 4.2. Đọc file ảnh

img1 = cv.imread(**'img/object1.jpg'**, cv.IMREAD\_GRAYSCALE) *# queryImage*img2 = cv.imread(**'img/scene1.jpg'**, cv.IMREAD\_GRAYSCALE) *# trainImage*

Ta đọc vào 2 file ảnh object và scene dưới chế độ đen trắng (GRAY) vì thuật toán SIFT chỉ làm việc với ảnh đen trắng. Theo như các nghiên cứu, màu sắc của ảnh không giúp ích nhiều cho việc xác định các đặc trưng, mà là độ sáng của ảnh.

Để chương trình không chạy quá lâu, ta cần lưu ý về kích thước của ảnh không nên quá lớn. Kích thước ảnh cũng không nên quá nhỏ, vì như thế sẽ mất chi tiết của bức ảnh, và việc tính toán sẽ trở nên kém hiệu quả.

## 4.3. Tìm các keypoint và descriptor với SIFT

sift = cv.SIFT\_create()  
kp1, des1 = sift.detectAndCompute(img1, **None**)  
kp2, des2 = sift.detectAndCompute(img2, **None**)

Nếu ta sử dụng hàm cv.drawKeypoint() để vẽ ra các keypoint, thì ta sẽ thu được kết quả tương tự như hình dưới đây:



Hình : Các keypoint cùng với scale và orientation của chúng

## 4.4. Features matching

OpenCV cung cấp sẵn cho ta phương thức để hỗ trợ việc matching như sau:

matches = []

bf = cv.BFMatcher()  
matches = bf.knnMatch(des1,des2,k=2)

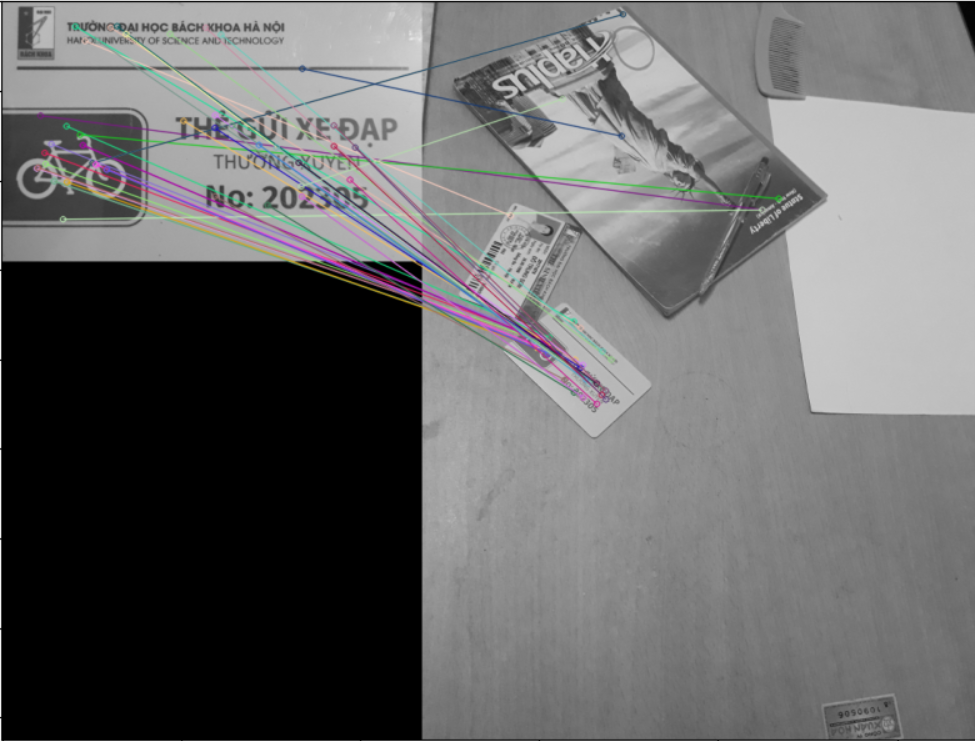
Tuy nhiên nếu ta tự triển khai thì sẽ như sau:

matches = []  
  
**for** i **in** range(0, len(des1)):  
 minDistSq1 = minDistSq2 = 99999999  
 minIdx1 = minIdx2 = -1  
 **for** j **in** range(0, len(des2)):  
 diffVec = des2[j] - des1[i]  
 distSq = np.sum(np.square(diffVec))  
 **if** distSq < minDistSq1:  
 minDistSq2 = minDistSq1  
 minDistSq1 = distSq  
 minIdx2 = minIdx1  
 minIdx1 = j  
 **elif** distSq < minDistSq2:  
 minDistSq2 = distSq  
 minIdx2 = j  
  
 matches.append([  
 cv.DMatch(i, minIdx1, 0, math.sqrt(minDistSq1)),  
 cv.DMatch(i, minIdx2, 0, math.sqrt(minDistSq2))  
 ])

Kết thúc quá trình, ta thu được mảng matches trong đó mỗi phần tử là một list gồm 2 đối tượng DMatch. Số lượng phần tử trong matches bằng với số keypoint trong train image. Mỗi phần tử là thông tin về keypoint gần nhất và keypoint gần nhì cùng với khoảng cách giữa chúng. Ta sẽ sử dụng hai khoảng cách này để tính toán tỉ lệ và lọc ra các good matches:

good = []  
**for** m,n **in** matches:  
 **if** m.distance < 0.75\*n.distance:  
 good.append([m])

Nếu ta sử dụng hàm cv.drawMatches() cho mảng good thì sẽ thu được kết quả có dạng như sau:



Hình : Kết quả matching

## 4.5. Object locating

Việc tiếp theo cần làm đó là xác định vị trí của vật thể trong train image. Như đã phân tích ở trên, ta sẽ phải tìm ma trận homography. OpenCV hỗ trợ phương thức cv.findHomography() để tìm ma trận homography. Để tìm biến đổi phối cảnh của một tập các điểm, ta sử dụng cv.perspectiveTransform().

MIN\_MATCH\_COUNT = 10  
  
src\_pts = np.float32([kp1[m.queryIdx].pt **for** m **in** good])  
dst\_pts = np.float32([kp2[m.trainIdx].pt **for** m **in** good])  
  
**if** len(good) > MIN\_MATCH\_COUNT:  
 M, mask = cv.findHomography(src\_pts, dst\_pts, cv.RANSAC,5.0)  
 h, w = img1.shape  
 pts = np.float32([[0,0], [0,h-1], [w-1,h-1], [w-1, 0]]).reshape(-1,1,2)  
 dst = cv.perspectiveTransform(pts, M)  
 img2 = cv.polylines(img2, [np.int32(dst)], **True**, 255, 3, cv.LINE\_AA)  
  
**else**:  
 print(**"Not enough matches are found - {}/{}"**.format(len(good), MIN\_MATCH\_COUNT))

MIN\_MATCH\_COUNT là hằng số chỉ số lượng match tối thiểu để xác định vật thể, nếu số lượng good matches nhỏ hơn MIN\_MATCH\_COUNT, ta in ra thông báo tương ứng với việc không tìm thấy vật thể.

src\_pts và dst\_pts theo thứ tự là tập các toạ độ của các keypoint thuộc các good matches ở query image và train image.

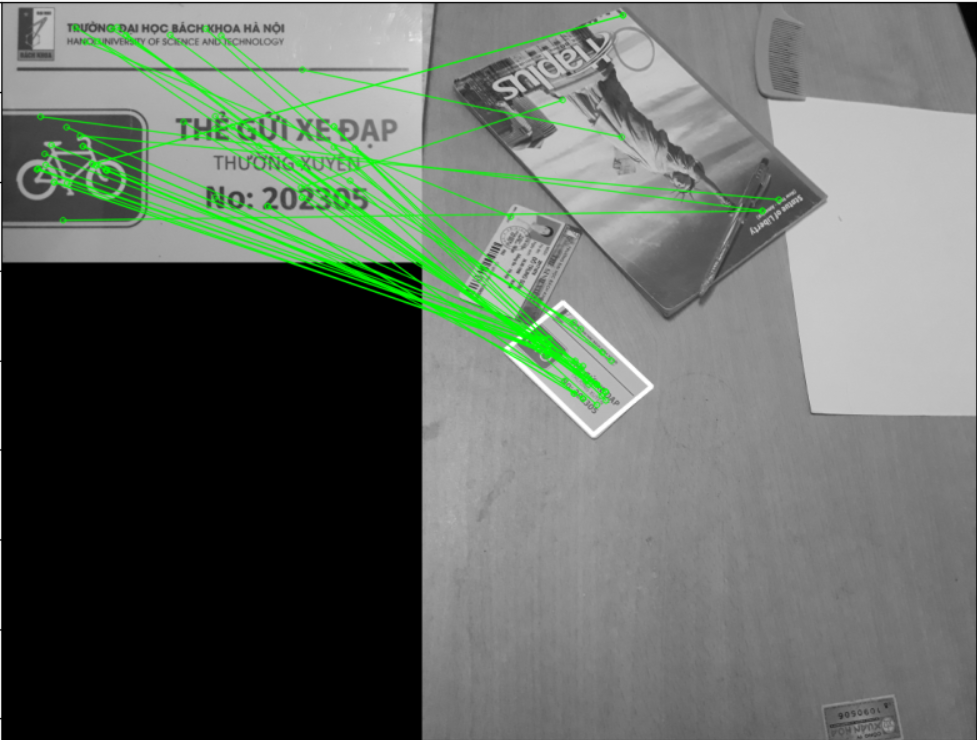
M là ma trận homography tối ưu tìm được. mask là mảng đánh dấu các phần tử inliner và outliner, hay nói cách khác là đánh dấu các keypoint nằm trong vật thể và nằm ngoài vật thể.

Sau khi tìm được M, ta sử dụng nó để ánh xạ 4 điểm tương ứng với 4 góc của query image sang 4 điểm tương ứng với 4 góc của vật thể trên train image. Sau đó vẽ ra tứ giác đánh dấu vật thể trên hình.

Cuối cùng, vẽ ra màn hình kết quả:

draw\_params = dict(matchColor = (0,255,0),singlePointColor = **None**,  
 matchesMask = **None**, flags = 2)  
img3 = cv.drawMatches(img1,kp1,img2,kp2,good,**None**,\*\*draw\_params)

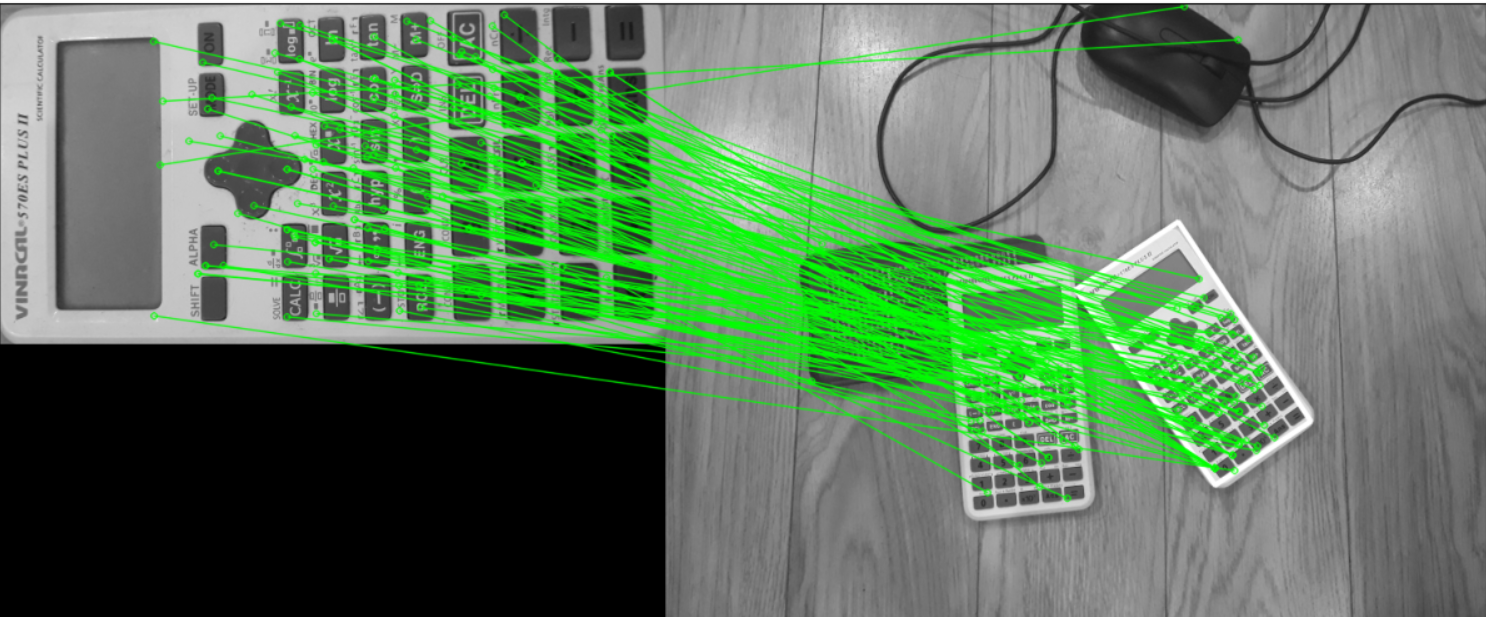
plt.imshow(img3, **'gray'**),plt.show()



Hình : Kết quả

## 4.5. Detect multiple instances of object

Chương trình đến đây đã nhận diện tốt được một vật thể, tuy nhiên nếu train image có nhiều hơn 1 thể hiện của vật thể, thì chương trình mới chỉ phát hiện được một vật thể giống nhất. Ví dụ như sau:



Hình : Ví dụ

Giải pháp là ta sẽ phân cụm các keypoint trong train image, sau đó thực hiện matching và locating tương tự như trước đối vưới từng cụm.

scene\_points = np.float32([kp.pt **for** kp **in** kp2])  
bandwidth = estimate\_bandwidth(scene\_points, quantile=0.2)  
ms = MeanShift(bandwidth=bandwidth, bin\_seeding=**True**)  
ms.fit(scene\_points)

Kết quả có dạng như sau:



Hình : Minh hoạ các cụm keypoint, mỗi cụm tương ứng với một màu

Sau đó thực hiện features matching và locating object đối với từng cụm, được kết quả như sau:



Hình : Kết quả



Hình : Kết quả

# Kết luận

Chương trình về cơ bản đã hoạt động tương đối tốt, nhận diện và khoanh vùng chính xác các vật thể. Tuy nhiên đôi lúc vẫn còn sai. Để chương trình hoạt động tốt, cần điều chỉnh các tham số đầu vào như bandwidth cho hàm Mean Shift và MIN\_MATCH\_COUNT cần tương ứng với vật thể (ví dụ như trường hợp nhận diện chiếc máy tính, MIN\_MATCH\_COUNT = 100, còn trường hợp nhận diện thẻ gửi xe, MIN\_MATCH\_COUNT = 50).

Chiến lược được sử dụng trong chương trình vẫn còn hạn chế. Ví dụ như bề mặt của vật thể nhẵn mịn, ít keypoint tập trung tại vật thể, khiến cho việc phân cụm bị sai, từ đó dẫn tới kết quả sai. Hoặc ví dụ như một phần của vật thể bị che khuất, dẫn tới các keypoint của vật thể bị chia thành nhiều cụm, thì khi nhận diện ta thu được nhiều vật thể nhưng thực tế chỉ là một. Trong trường hợp này, cần phải hậu xử lí kết quả để thu được đầu ra chính xác.