ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO KẾT QUẢ
BÀI TẬP THỰC HÀNH SỐ 5

Họ và tên: Nguyễn Văn Diễn

MSV: 22027541

Câu 1:

Thêm các thư viện cần thiết:

```
import cv2
import numpy as np
```

Đọc ảnh đầu vào:

```
img = cv2.imread("../images/img6.1.jpg")
```

- Dùng hàm `cv2.imread()` để đọc ảnh đầu vào và gán vào biến `img` dưới dạng mảng NumPy.

Hình 1. Ảnh đầu vào (Câu 1)

Chuyển từ ảnh màu sang ảnh xám:

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

- Vì thuật toán Harris hoạt động dựa trên sự thay đổi cường độ sáng của điểm ảnh, nên yêu câu ảnh đầu vào phải là ảnh xám (grayscale).
- Hàm `cv2.cvtColor()` được dùng để chuyển đổi không gian màu của ảnh từ BGR (mặc định của OpenCV) sang ảnh xám theo công thức:

$$gray = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Chuyển kiểu dữ liệu của ảnh:

```
gray = np.float32(gray)
```

- Vì hàm `cv2.cornerHarris()` yêu cầu ảnh đầu vào phải có kiểu dữ liệu là `float32` nên dùng hàm `np.float32()` để chuyển từ kiểu `uint8` (kiểu dữ liệu mặc định của ảnh khi được đọc bằng hàm `cv2.imread()`) sang kiểu `float32`.

Áp dụng thuật toán Harris:

```
dest = cv2.cornerHarris(gray, 2, 3, 0.04)
```

- Thuật toán Harris xác định một góc bằng cách xét một cửa sổ nhỏ gồm các điểm ảnh. Nếu cửa sổ này được dịch chuyển theo bất kỳ hướng nào, cường độ sáng bên trong cửa sổ thay đổi lớn, thì điểm trung tâm của cửa sổ đó được coi là một góc.
- Các tham số của hàm `cv2.cornerHarris()`:
  - `gray`: Ảnh xám kiểu `float32`.
  - `blockSize=2`: Kích thước của cửa sổ được xét cho mỗi điểm ảnh.
  - `ksize=3`: Kích thước của bộ lọc Sobel để tính đạo hàm theo hướng x và y.
  - `k=0.04`: Là một tham số tự do trong phương trình tính điểm Harris, thường có giá trị trong khoảng [0.04, 0.06]:

$$R = \det(M) - k \times (\text{trace}(M))^2$$

    - Tham số k kiểm soát độ nhạy của hàm. Giá trị k càng lớn (càng gần giá trị 0.06) thì càng tập trung vào góc sắc và loại bỏ các góc mềm mại hơn. Giá trị k càng nhỏ (gần 0.04) sẽ tăng số lượng điểm góc được phát hiện.
    - Kernel Sobel có kích thước `ksize` × `ksize` được sử dụng để tính toán đạo hàm theo hướng x (I_x) và y (I_y) của cường độ sáng ảnh. I_x và I_y là đầu vào để xây dựng ma trận cấu trúc M.
    - Ma trận cấu trúc M, là ma trận có kích thước `blockSize` × `blockSize`, xác định quy mô của góc mà thuật toán đang cố gắng tìm kiếm. `blockSize` nhỏ nhạy với góc nhỏ/chi tiết, `blockSize` lớn nhạy với góc lớn hơn.
    - Điểm đáp ứng R cho biết loại cấu trúc hình học tại một điểm ảnh:

- $|R|$ nhỏ ($R \approx 0$): Vùng phẳng, không có sự thay đổi cường độ sáng đáng kể theo bất kỳ hướng nào.
- $R < 0$: Vùng cạnh, có sự thay đổi cường độ sáng lớn theo một hướng.
- R lớn và dương: Vùng góc, có sự thay đổi cường độ sáng lớn theo nhiều hướng.

- Kết quả `dest` trả về một ma trận có cùng kích thước với ảnh đầu vào, trong đó mỗi điểm ảnh có một giá trị. Điểm ảnh nào có giá trị càng cao thì khả năng nó là một góc càng lớn.

Tạo ngưỡng để xác định góc:

```
img[dest > 0.01 * dest.max()] = [0, 0, 255]
```

- Để quyết định điểm nào là góc, ta cần một ngưỡng bằng 1% giá trị lớn nhất trong ma trận kết quả dest. Bất kỳ điểm nào có giá trị lớn hơn ngưỡng sẽ được coi là góc.
- `dest > 0.01 * dest.max()` tạo một mảng kiểu boolean (True/False) để thay đổi màu của tất cả các điểm ảnh tương ứng với vị trí True thành màu đỏ.

Chuyển đổi ảnh từ BGR sang RGB để hiển thị:

```
result = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

Kết quả:

Ảnh gốc

Ảnh kết quả

- Thuật toán đã được áp dụng vào ảnh toà nhà có nhiều chi tiết kiến trúc.
- Kết quả là một ảnh mới, trong đó các điểm được xác định là góc đã được đánh dấu bằng các điểm màu đỏ.
- Thuật toán hoạt động hiệu quả đối với các góc có cấu trúc rõ ràng và độ tương phản cao:
    - Góc ban công: Hầu hết tất các các góc nhọn, vuông của các khối ban công đều được xác định chính xác. Đây là những điểm có sự thay đổi cường độ sáng rõ rệt theo các hướng.
    - Góc cửa sổ: Phần lớn các góc của cửa sổ cũng được phát hiện chính xác.
- Thuật toán phát hiện nhầm một số điểm không phải là góc:
    - Có nhiều chấm đỏ xuất hiện dày đặc trên bề mặt lan can của ban công có đục lỗ, mỗi lỗ này đều có cạnh sáng/tối giống hệt một góc đối với thuật toán.
    - Một vài điểm trên các bề mặt phẳng do các chi tiết nhỏ trên bề mặt.
- Một số góc thật đã bị bỏ qua, chủ yếu ở các khu vực có độ tương phản thấp:
    - Các góc nằm hoàn toàn trong vùng bóng râm.
    - Các góc cửa sổ do độ tương phản không rõ ràng.

Câu 2:

a. Phát hiện và mô tả đặc trưng bằng SIFT:

Thêm thư viện OpenCV:

```
import cv2
```

Đọc ảnh đầu vào:

```
image = cv2.imread('../images/img6.2.jpg')
```

- Dùng hàm `cv2.imread()` để đọc ảnh đầu vào và gán vào biến `img` dưới dạng mảng NumPy.

Hình 2. Ảnh đầu vào (Câu 2a)

Chuyển từ ảnh màu sang ảnh xám:

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

- Thuật toán SIFT hoạt động dựa trên gradent, là sự thay đổi đột ngột về cường độ sáng (từ sáng sang tối hoặc ngược lại), nên cần chuyển ảnh sang ảnh xám.
- Hàm `cv2.cvtColor()` được dùng để chuyển đổi không gian màu của ảnh từ BGR (mặc định của OpenCV) sang ảnh xám theo công thức:

$$gray = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Khởi tạo đối tượng SIFT:

```
sift = cv2.SIFT_create()
```

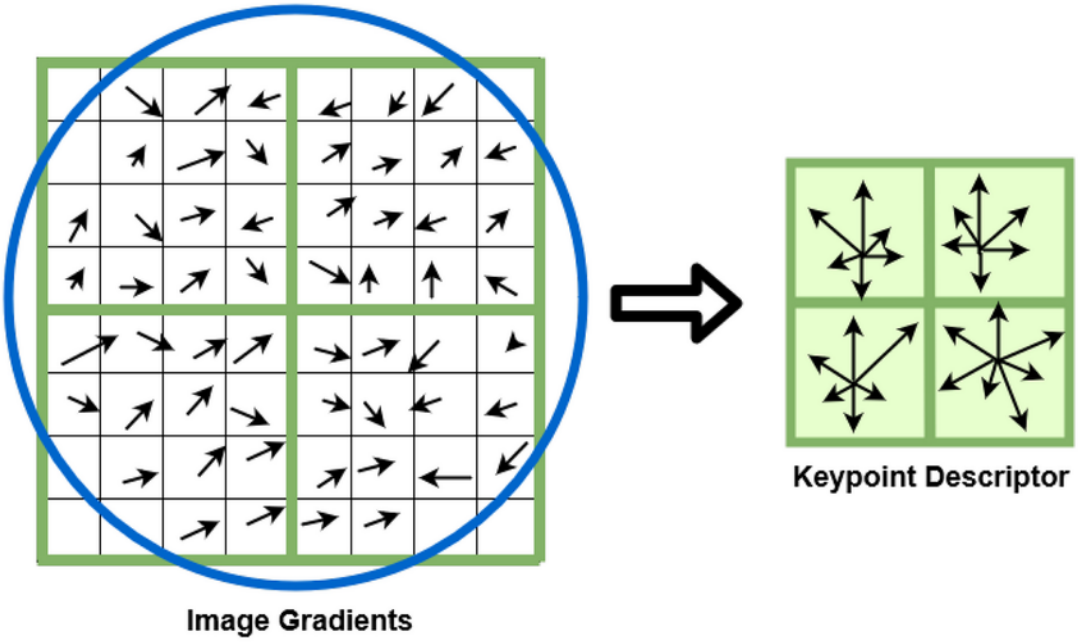
- Tạo đối tượng `sift` với các tham số mặc định để thực hiện thuật toán SIFT.

Phát hiện các điểm đặc trưng và tính toán vectơ mô tả:

```
keypoints, descriptors = sift.detectAndCompute(gray, None)
```

- Phát hiện các điểm đặc trưng:
    - Bước 1: Dò tìm cực trị trong không gian tỷ lệ: Mục tiêu của bước này là tìm ra các điểm ứng viên có tính bất biến với tỷ lệ.
        - Thuật toán xây dựng một không gian tỷ lệ bằng cách liên tục làm mờ ảnh gốc bằng bộ lọc Gauss với các độ lệch chuẩn σ tăng dần, sau đó giảm kích thước ảnh để tạo ra các octaves.
        - Lấy hiệu giữa hai ảnh liền kề trong không gian tỷ lệ để tạo ra các ảnh DoG (Difference of Gaussians).
        - Một điểm ảnh trong ảnh DoG được so sánh với 26 lân cận của nó (8 lân cận trong cùng một ảnh, 9 lân cận ở ảnh tỷ lệ trên và 9 lân cận ở ảnh tỷ lệ dưới). Nếu nó là cực đại hoặc cực tiểu so với tất cả 26 lân cận này, nó được chọn là một điểm có khả năng.
        - Kết quả: Xác định được vị trí `pt` và tỷ lệ `size` của điểm đó.
    - Bước 2: Định vị và lọc các điểm có khả năng:
        - Thuật toán sử dụng một mô hình Taylor bậc hai để định vị chính xác hơn vị trí của điểm ứng cử viên.
        - Các điểm có giá trị cường độ (độ tương phản) thấp sẽ bị loại bỏ, vì chúng nhạy cảm với nhiễu.
        - Các điểm nằm trên cạnh cũng bị loại bỏ. Một điểm đặc trưng tốt là một góc có sự thay đổi gradient rõ ràng theo nhiều hướng. Các điểm trên cạnh chỉ có gradient mạnh theo một hướng và không ổn định.
        - Kết quả: Đảm bảo danh sách `keypoints` chỉ chứa các đặc trưng mạnh và ổn định.
    - Bước 3: Mục tiêu của bước này là gán một hoặc nhiều hướng cho mỗi điểm đặc trưng, làm cơ sở cho tính bất biến với phép xoay:

        - Với mỗi điểm đặc trưng, một vùng lân cận được xem xét (kích thước vùng lân cận phụ thuộc vào kích thước của điểm đặc trưng).
        - Hướng và độ lớn gradient được tính toán cho tất cả các điểm ảnh trong vùng này.
        - Một biểu đồ hướng 36-bin (mỗi bin 10 độ) được xây dựng. Mỗi điểm ảnh đóng góp vào biểu đồ dựa trên độ lớn gradient và được làm trọng số bởi một cửa sổ Gauss.
        - Đỉnh cao nhất trong biểu đồ được chọn làm hướng chính. Bất kỳ đỉnh nào khác cao hơn 80% đỉnh chính cũng được xem xét để tạo ra một keypoint mới (cùng vị trí, cùng tỷ lệ, nhưng khác hướng).
        - Kết quả: Bước này tính toán và gán thuộc tính `angle` (góc) cho mỗi đối tượng `cv2.KeyPoint`.
    - Sau 3 bước, biến `keypoints` là một list các đối tượng `cv2.KeyPoint` được tạo ra.
- Tính toán vectơ mô tả:
    - Bước 4: Tạo vectơ mô tả:
        - Một vùng lân cận 16×16 điểm ảnh được lấy xung quanh điểm đặc trưng. Vùng này được xoay về hướng chính (dựa theo `angle` ở bước 3) để đảm bảo tính bất biến với phép xoay.
        - Vùng 16×16 này được chia thành một lưới 4×4, tạo ra 16 ô, mỗi ô có kích thước 4×4 điểm ảnh.
        - Trong mỗi ô 4×4, một biểu đồ 8 hướng gradient (mỗi bin 45 độ) được tính toán.
        - Các giá trị từ 16 biểu đồ (mỗi biểu đồ 8 giá trị) được nối lại thành một vectơ duy nhất.
        - Kết quả: 16 ô × 8 hướng/ô = 128 chiều. Vectơ 128 chiều này chính là vectơ mô tả của điểm đặc trưng. Vectơ này được chuẩn hoá về vectơ đơn vị để giảm ảnh hưởng của sự thay đổi ánh sáng.
    - Biến `descriptors` là một mảng NumPy có kích thước $(N, 128)$, với N là tổng số các điểm đặc trưng.



Hình 3. Biểu diễn vectơ mô tả

- Hàm `sift.detectAndCompute()` đóng góp toàn bộ 4 bước của thuật toán SIFT:
    - Tham số `gray`: Là ảnh xám đầu vào.
    - Tham số mặt nạ `None`: Chỉ thị cho thuật toán thực hiện trên toàn bộ ảnh.
    - Kết quả:
        - `keypoints`: Là đầu ra của các bước 1, 2, và 3, chứa một danh sách các đối tượng với thông tin `pt`, `size` và `angle`.
        - `descriptions`: Là đầu ra của bước 4, chứa mảng $(N, 128)$ các vectơ đặc trưng, được tính toán dựa trên danh sách `keypoints`.

Vẽ các điểm đặc trưng lên ảnh:

```
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

- Hàm `cv2.drawKeypoints` giúp vẽ các điểm đặc trưng lên ảnh.
- Dùng cờ `cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS` để vẽ vòng tròn tại các điểm đặc trưng với bán kính của vòng tròn thể hiện `size` (tỷ lệ) và đoạn thẳng từ tâm thể hiện `angle` (hướng).

In ra tổng số các điểm đặc trưng và kích thước của ma trận descriptors:

```
print(f"Tổng số điểm đặc trưng SIFT tìm được: {len(keypoints)}")
print(f"Kích thước của ma trận descriptors SIFT: {descriptors.shape}")
```

Kết quả:

```
Tổng số điểm đặc trưng SIFT tìm được: 19424
Kích thước của ma trận descriptors SIFT: (19424, 128)
```

Hình 4. Ảnh đầu ra (Câu 2a)

Nhận xét, phân tích:

- Gồm hai thành phần chính:
    - Các điểm đặc trưng được biểu thị bằng các chấm nhỏ, dày đặc, nhiều màu sắc.
    - Thông tin thuộc tính các điểm đặc trưng được biểu thị bằng các vòng tròn có đường kẻ bán kính, thể hiện các thuộc tính của một số điểm đặc trưng được chọn lọc.
- Các vùng có mật độ điểm đặc trưng cao:
    - Tán cây: Đây là khu vực có mật độ điểm đặc trưng cao nhất do cấu trúc phức tạp của lá cây tạo ra vô số điểm góc và các vùng có gradient thay đổi đột ngột ở nhiều hướng và nhiều tỷ lệ khác nhau.
    - Mặt đất và đường mòn: Lớp sỏi đá trên đường mòn và vùng cỏ xung quanh tạo thành một bề mặt có kết cấu phức tạp. Vì SIFT nhạy với các vùng có kết cấu cao, do đó nó phát hiện được một số lượng lớn các điểm đặc trưng tại đây.
- Các vùng có mật độ điểm đặc trưng thấp:
    - Bầu trời: Bầu trời là một khu vực lớn có màu sắc đồng nhất (hoặc dải màu chuyển tiếp mượt mà). Do thiếu sự thay đổi đột ngột về gradient, SIFT không tìm thấy bất kỳ đặc trưng ổn định nào ở đây. Điều này là hoàn toàn chính xác và mong đợi, vì các vùng đồng nhất không chứa thông tin đặc trưng.
    - Thân cây: Thân cây có mật độ điểm đặc trưng thấp hơn đáng kể so với tán lá, nhưng vẫn nhiều hơn so với bầu trời, do nó vẫn có các chi tiết về kết cấu vỏ cây.
- Các vòng tròn lớn hơn được vẽ trên ảnh (ví dụ: trên tán cây, trên đường mòn, trên con chim) biểu diễn các thuộc tính cốt lõi của SIFT cho từng điểm đặc trưng cụ thể, thường được kích hoạt bằng cờ `cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS`:
    - Tâm vòng tròn: Là vị trí (x, y) của điểm đặc trưng được phát hiện.
    - Bán kính vòng tròn (Scale - Tỷ lệ): Đây là thuộc tính quan trọng nhất của SIFT. Bán kính của vòng tròn tỷ lệ thuận với scale (độ lớn, hay mức độ "zoom") mà tại đó điểm đặc trưng này được phát hiện là ổn định nhất.
        - Các vòng tròn có bán kính nhỏ (ví dụ: trên các con chim) cho thấy đặc trưng đó là một chi tiết nhỏ, được phát hiện ở lớp ảnh có độ phân giải cao trong kim tự tháp Gaussian (Gaussian Pyramid).

        - Các vòng tròn có bán kính lớn (ví dụ: một số vòng tròn trong tán cây) cho thấy đặc trưng đó ổn định trên một vùng lân cận rộng hơn, được phát hiện ở các lớp ảnh mờ hơn (scale cao hơn) trong kim tự tháp.
        - Khả năng phát hiện các đặc trưng ở nhiều scale khác nhau chính là yếu tố "Scale-Invariant" (Bất biến về Tỷ lệ) của thuật toán.
    - Đường kẻ (Orientation - Hướng): Đường thẳng kẻ từ tâm ra mép vòng tròn biểu thị hướng chính của điểm đặc trưng. Hướng này được tính toán dựa trên hướng gradient của các điểm ảnh trong vùng lân cận điểm đặc trưng.
        - Việc gán một hướng cho mỗi điểm đặc trưng cho phép thuật toán đạt được tính "Rotation-Invariant" (Bất biến về Phép xoay). Khi so khớp, mô tả SIFT có thể được xoay tương ứng với hướng này để bù trừ cho sự xoay của đối tượng trong ảnh.
- Hình ảnh đầu ra xác nhận rằng thuật toán SIFT đã hoạt động chính xác:
    - Tính chọn lọc: Thuật toán đã phát hiện thành công các điểm đặc trưng tại các khu vực có kết cấu phức tạp và nhiều thông tin (lá cây, sỏi đá, con chim) và bỏ qua các vùng đồng nhất (bầu trời).
    - Tính bất biến: Việc trực quan hóa các vòng tròn với bán kính và hướng khác nhau cho thấy rõ khả năng phát hiện đặc trưng ở nhiều tỷ lệ và gán hướng cho chúng.

b. Phát hiện và mô tả đặc trưng bằng ORB:

- ORB là một sự kết hợp hiệu quả:
    - FAST (Features from Accelerated Segment Test) để tìm các điểm đặc trưng (nhanh).
    - BRIEF (Binary Robust Independent Elementary Features) để tạo các vectơ mô tả (nhanh và nhẹ).

Thêm các thư viện cần thiết:

```
import cv2
import numpy as np
```

Đọc ảnh đầu vào (tương tự câu a):

```
image = cv2.imread('../images/img6.2.jpg')
```

- Dùng hàm `cv2.imread()` để đọc ảnh đầu vào và gán vào biến `img` dưới dạng mảng NumPy.

Hình 5. Ảnh đầu vào (Câu 2b)

Chuyển từ ảnh màu sang ảnh xám:

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

- Chuyển về ảnh xám làm giảm bài toán xuống còn 1 kênh duy nhất, giúp tính toán nhanh hơn.
- Các đặc trưng quan trọng nhất của một vật thể (góc, cạnh, kết cấu) thường được thể hiện rõ nhất qua sự thay đổi độ sáng, không phải màu sắc.
- Màu sắc của một vật thể thay đổi rất nhiều tùy thuộc vào điều kiện ánh sáng. Cường độ sáng ổn định hơn, giúp thuật toán nhận diện vật thể trong các điều kiện ánh sáng khác nhau.

- Hàm `cv2.cvtColor()` được dùng để chuyển đổi không gian màu của ảnh từ BGR (mặc định của OpenCV) sang ảnh xám theo công thức:

$$gray = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Khởi tạo đối tượng ORB:

```
orb = cv2.ORB_create(nfeatures=1000)
```

- Tham số `nfeatures=1000` yêu cầu ORB tìm tối đa 1000 điểm đặc trưng tốt nhất.
- SIFT sẽ tìm tất cả các điểm nó cho là đủ tiêu chuẩn, có thể là vài nghìn điểm, còn ORB cho phép kiểm soát số lượng.

Phát hiện các điểm đặc trưng và tính toán vectơ mô tả:

```
keypoints, descriptors = orb.detectAndCompute(gray_image, None)
```

- Phát hiện các điểm đặc trưng:
    - Mục tiêu: Tìm các góc nhanh chóng.
    - Thực hiện:
        - Thuật toán quét qua từng pixel p trong ảnh.
        - Nó nhìn vào một vòng tròn 16 điểm ảnh xung quanh p.
        - Nó kiểm tra xem có 9 điểm ảnh liên tiếp trên vòng tròn đó tất cả đều sáng hơn p (cộng thêm một ngưỡng t), hoặc tất cả đều tối hơn p (trừ đi một ngưỡng t) hay không.
        - Nếu có, p được đánh dấu là một điểm đặc trưng tiềm năng.
    - Ưu điểm: Nhanh vì nó chỉ là các phép so sánh `if...then` đơn giản.
    - Đạt được tính bất biến tỷ lệ:
        - Thuật toán tạo ra nhiều phiên bản của ảnh với các kích thước nhỏ dần.
        - Chạy thuật toán FAST trên từng lớp ảnh vừa tạo.

        - Điều này đảm bảo nó tìm thấy cả các góc "lớn" (ở các lớp ảnh gốc) và các góc "nhỏ" (ở các lớp ảnh thu nhỏ).
        - Kết quả: Biến `keypoints` bây giờ sẽ chứa các điểm đặc trưng ở nhiều kích thước khác nhau.
    - Lọc các điểm đặc trưng (Giới hạn `nfeatures`):
        - ORB sử dụng một phép đo chất lượng góc gọi là Harris Corner Score để chấm điểm cho tất cả các điểm đặc trưng tìm được.
        - Nó sắp xếp các điểm đặc trưng theo điểm số và chỉ giữ lại 1000 (hoặc `nfeatures`) điểm đặc trưng tốt nhất.
    - Gán hướng: FAST không cung cấp thông tin về hướng, làm cho nó không có khả năng chống xoay. ORB không dùng biểu đồ gradient. Thay vào đó, nó dùng một phương pháp nhanh hơn gọi là Intensity Centroid (Trọng tâm Cường độ).
        - Với mỗi điểm đặc trưng, nó xem xét một vùng ảnh xung quanh.
        - Nó tính toán "trọng tâm" của vùng này, nhưng các điểm ảnh sáng hơn sẽ "nặng" hơn (có trọng số cao hơn).
        - Vectơ đi từ tâm điểm đặc trưng đến "trọng tâm cường độ" này được coi là hướng chính của điểm đặc trưng.
        - Kết quả: Mỗi điểm đặc trưng bây giờ có một thuộc tính `angle` (góc).
- Tính toán vectơ mô tả:
    - Tạo vectơ mô tả bằng BRIEF (Binary Robust Independent Elementary Features):
        - Mục tiêu: Tạo một thành phần mô tả cho mỗi điểm đặc trưng.
        - Đặc điểm: BRIEF tạo ra một vectơ mô tả nhị phân, chỉ chứa 0 và 1.
        - Thực hiện:
            - Với mỗi điểm đặc trưng, nó lấy một vùng ảnh xung quanh.
            - Nó chọn ra 256 cặp điểm ảnh (A, B) ngẫu nhiên bên trong vùng đó theo một mẫu cố định.
            - Với mỗi cặp, nó thực hiện một phép so sánh cường độ sáng: Nếu `intensity(A) > intensity(B)` thì là 1, còn lại là 0.
            - Sau 256 phép so sánh, nó thu được một chuỗi 256 bit.

        - Kết quả: Chuỗi 256 bit này chính là thành phần mô tả. Để lưu trữ hiệu quả, nó được nén thành 32 bytes ($256 \text{ bits}/8 \text{ bits/ byte} = 32 \text{ bytes}$). Vì vậy `descriptors.shape` là $(N, 32)$.
    - Đạt được tính bất biến xoay:
        - Vấn đề: Thuật toán BRIEF gốc rất tệ khi bị xoay. Nếu ảnh xoay đi, các cặp (A, B) sẽ chỉ vào các điểm ảnh hoàn toàn khác, tạo ra một thành phần mô tả hoàn toàn khác.
        - Giải pháp:
            - ORB lấy thông tin `angle` đã được tính ở trên (bằng Intensity Centroid).
            - Thay vì dùng mẫu 256 cặp điểm ảnh cố định, nó xoay toàn bộ mẫu đó đi một góc `angle` sao cho mẫu này thẳng hàng với hướng của điểm đặc trưng.
            - Bây giờ, nó mới thực hiện 256 phép so sánh A > B trên mẫu đã xoay này.
        - Kết quả: Cho dù vật thể bị xoay, góc `angle` sẽ thay đổi, mẫu sẽ xoay theo, và 256 phép so sánh sẽ luôn cho ra cùng một chuỗi bit. Thành phần mô tả trở nên bất biến với phép xoay.
    - Kết quả: Biến vectơ mô tả `descriptors` được tạo ra, là một mảng NumPy có kích thước $(N, 32)$ kiểu `unit8`, trong đó N là số điểm đặc trưng.
- Tham số mặt nạ `None` để tìm kiếm điểm đặc trưng trên toàn bộ bức ảnh.

Vẽ các điểm đặc trưng lên ảnh:

```
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

- Hàm `cv2.drawKeypoints` giúp vẽ các điểm đặc trưng lên ảnh.
- Dùng cờ `cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS` để vẽ vòng tròn tại các điểm đặc trưng với bán kính của vòng tròn thể hiện `size` (tỷ lệ) và đoạn thẳng từ tâm thể hiện `angle` (hướng).

In ra tổng số các điểm đặc trưng và kích thước của ma trận `descriptors`:

```
print(f"Tổng số điểm đặc trưng SIFT tìm được: {len(keypoints)}")
print(f"Kích thước của ma trận descriptors SIFT: {descriptors.shape}")
```

```
Tổng số điểm đặc trưng ORB tìm được: 1000
Kích thước của ma trận descriptors ORB: (1000, 32)
```

Hình 6. Ảnh đầu ra (Câu 2b)

Nhận xét, phân tích:

- Vùng có mật độ điểm đặc trưng cao:
    - Tán cây: Tương tự như SIFT, tán cây là nơi tập trung nhiều điểm đặc trưng nhất. Cấu trúc phức tạp của lá và cành cây tạo ra vô số điểm góc với độ tương phản cao, đây là đối tượng lý tưởng cho thuật toán FAST.

    - Đàn chim: Mỗi con chim, với hình dạng rõ rệt tương phản trên nền trời, được xác định là một tập hợp các điểm góc. ORB phát hiện chúng một cách hiệu quả.
- Vùng có mật độ điểm đặc trưng thấp:
    - Đường mòn và Đồng cỏ: So với kết quả của SIFT, thuật toán ORB dường như phát hiện ít đặc trưng hơn đáng kể ở các vùng có kết cấu nhưng ít góc cạnh rõ rệt như đường mòn và vùng cỏ. FAST (với ngưỡng mặc định) tập trung chủ yếu vào các điểm "góc" mạnh, trong khi SIFT (dựa trên Difference of Gaussians) nhạy hơn với các "blob" (đốm sáng/tối) và các vùng kết cấu mịn.
    - Thân cây: Chỉ một vài điểm đặc trưng được phát hiện dọc theo mép (cạnh) của thân cây, nơi có sự thay đổi đột ngột về cường độ so với nền.
- Vùng trống (Không có điểm đặc trưng):
    - Bầu trời: Giống như SIFT, ORB hoàn toàn chính xác khi bỏ qua vùng bầu trời đồng nhất vì không có bất kỳ điểm góc hoặc cạnh nào để phát hiện.
- Hình ảnh đầu ra cho thấy ORB đã thực hiện đúng chức năng của mình: phát hiện nhanh các điểm đặc trưng dựa trên góc cạnh. Mật độ điểm đặc trưng tập trung dày đặc ở các vùng có cấu trúc phức tạp (tán cây) và các đối tượng có cạnh rõ ràng (đàn chim), đồng thời bỏ qua các vùng đồng nhất.

Câu 3:

a. Hai ảnh của cùng một cảnh hoặc đối tượng nhưng được chụp từ hai góc độ khác nhau:

Hình 7. Ảnh đầu vào 1 (Câu 3)

Hình 8. Ảnh đầu vào 2 (Câu 3)

Thêm thư viện OpenCV:

```
import cv2
```

Đọc hai ảnh đầu vào:

```
img1 = cv2.imread('../images/img6.3.jpg')
img2 = cv2.imread('../images/img6.4.jpg')
```

- Dùng hàm `cv2.imread()` để đọc ảnh đầu vào và gán vào các biến `img1` và `img2` dưới dạng mảng NumPy.

Chuyển từ ảnh màu sang ảnh xám:

```
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
```

- Chuyển về ảnh xám làm giảm bài toán xuống còn 1 kênh duy nhất, giúp tính toán nhanh hơn.
- Các đặc trưng quan trọng nhất của một vật thể (góc, cạnh, kết cấu) thường được thể hiện rõ nhất qua sự thay đổi độ sáng, không phải màu sắc.
- Màu sắc của một vật thể thay đổi rất nhiều tùy thuộc vào điều kiện ánh sáng. Cường độ sáng ổn định hơn, giúp thuật toán nhận diện vật thể trong các điều kiện ánh sáng khác nhau.
- Hàm `cv2.cvtColor()` được dùng để chuyển đổi không gian màu của ảnh từ BGR (mặc định của OpenCV) sang ảnh xám theo công thức:

$$gray = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

b. Sử dụng ORB để tìm các điểm đặc trưng và vectơ mô tả cho cả hai ảnh:

Khởi tạo đối tượng ORB:

```
orb = cv2.ORB_create(nfeatures=1000)
```

- Tham số `nfeatures=1000` yêu cầu ORB tìm tối đa 1000 điểm đặc trưng tốt nhất.
- SIFT sẽ tìm tất cả các điểm nó cho là đủ tiêu chuẩn, có thể là vài nghìn điểm, còn ORB cho phép kiểm soát số lượng.

Phát hiện các điểm đặc trưng và tính toán vectơ mô tả:

```
kp1, des1 = orb.detectAndCompute(gray1, None)
kp2, des2 = orb.detectAndCompute(gray2, None)
```

- Phát hiện các điểm đặc trưng:
  - Mục tiêu: Tìm các góc nhanh chóng.
  - Thực hiện:
    - Thuật toán quét qua từng pixel p trong ảnh.
    - Nó nhìn vào một vòng tròn 16 điểm ảnh xung quanh p.
    - Nó kiểm tra xem có 9 điểm ảnh liên tiếp trên vòng tròn đó tất cả đều sáng hơn p (cộng thêm một ngưỡng t), hoặc tất cả đều tối hơn p (trừ đi một ngưỡng t) hay không.
    - Nếu có, p được đánh dấu là một điểm đặc trưng tiềm năng.
  - Ưu điểm: Nhanh vì nó chỉ là các phép so sánh `if...then` đơn giản.
  - Đạt được tính bất biến tỷ lệ:
    - Thuật toán tạo ra nhiều phiên bản của ảnh với các kích thước nhỏ dần.
    - Chạy thuật toán FAST trên từng lớp ảnh vừa tạo.
    - Điều này đảm bảo nó tìm thấy cả các góc "lớn" (ở các lớp ảnh gốc) và các góc "nhỏ" (ở các lớp ảnh thu nhỏ).
    - Kết quả: Biến `keypoints` bây giờ sẽ chứa các điểm đặc trưng ở nhiều kích thước khác nhau.
  - Lọc các điểm đặc trưng (Giới hạn `nfeatures`):
    - ORB sử dụng một phép đo chất lượng góc gọi là Harris Corner Score để chấm điểm cho tất cả các điểm đặc trưng tìm được.

        - Nó sắp xếp các điểm đặc trưng theo điểm số và chỉ giữ lại 1000 (hoặc `nfeatures`) điểm đặc trưng tốt nhất.
    - Gán hướng: FAST không cung cấp thông tin về hướng, làm cho nó không có khả năng chống xoay. ORB không dùng biểu đồ gradient. Thay vào đó, nó dùng một phương pháp nhanh hơn gọi là Intensity Centroid (Trọng tâm Cường độ).
        - Với mỗi điểm đặc trưng, nó xem xét một vùng ảnh xung quanh.
        - Nó tính toán "trọng tâm" của vùng này, nhưng các điểm ảnh sáng hơn sẽ "nặng" hơn (có trọng số cao hơn).
        - Vectơ đi từ tâm điểm đặc trưng đến "trọng tâm cường độ" này được coi là hướng chính của điểm đặc trưng.
        - Kết quả: Mỗi điểm đặc trưng bây giờ có một thuộc tính `angle` (góc).
- Tính toán vectơ mô tả:
    - Tạo vectơ mô tả bằng BRIEF (Binary Robust Independent Elementary Features):
        - Mục tiêu: Tạo một thành phần mô tả cho mỗi điểm đặc trưng.
        - Đặc điểm: BRIEF tạo ra một vectơ mô tả nhị phân, chỉ chứa 0 và 1.
        - Thực hiện:
            - Với mỗi điểm đặc trưng, nó lấy một vùng ảnh xung quanh.
            - Nó chọn ra 256 cặp điểm ảnh (A, B) ngẫu nhiên bên trong vùng đó theo một mẫu cố định.
            - Với mỗi cặp, nó thực hiện một phép so sánh cường độ sáng: Nếu `intensity(A) > intensity(B)` thì là 1, còn lại là 0.
            - Sau 256 phép so sánh, nó thu được một chuỗi 256 bit.
        - Kết quả: Chuỗi 256 bit này chính là thành phần mô tả. Để lưu trữ hiệu quả, nó được nén thành 32 bytes ($256 \text{ bits} / 8 \text{ bits/byte} = 32 \text{ bytes}$). Vì vậy `descriptors.shape` là $(N, 32)$.
    - Đạt được tính bất biến xoay:
        - Vấn đề: Thuật toán BRIEF gốc rất tệ khi bị xoay. Nếu ảnh xoay đi, các cặp (A, B) sẽ chỉ vào các điểm ảnh hoàn toàn khác, tạo ra một thành phần mô tả hoàn toàn khác.
        - Giải pháp:

- ORB lấy thông tin `angle` đã được tính ở trên (bằng Intensity Centroid).
- Thay vì dùng mẫu 256 cặp điểm ảnh cố định, nó xoay toàn bộ mẫu đó đi một góc `angle` sao cho mẫu này thẳng hàng với hướng của điểm đặc trưng.
- Bây giờ, nó mới thực hiện 256 phép so sánh A > B trên mẫu đã xoay này.

  - Kết quả: Cho dù vật thể bị xoay, góc `angle` sẽ thay đổi, mẫu sẽ xoay theo, và 256 phép so sánh sẽ luôn cho ra cùng một chuỗi bit. Thành phần mô tả trở nên bất biến với phép xoay.
- Kết quả: Biến vectơ mô tả `descriptors` được tạo ra, là một mảng NumPy có kích thước $(N, 32)$ kiểu `unit8`, trong đó N là số điểm đặc trưng.

- Tham số mặt nạ `None` để tìm kiếm điểm đặc trưng trên toàn bộ bức ảnh.

c. Dùng BFMatcher (Brute-Force Matcher) để tìm các cặp đặc trưng có khả năng tương ứng nhất giữa hai ảnh:

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
matches = bf.knnMatch(des1, des2, k=2)
```

- Đây là phương pháp đối sánh đơn giản và trực quan nhất:
  - Nó lấy một thành phần mô tả từ ảnh 1.
  - Nó so sánh thành phần mô tả đó với tất cả các thành phần mô tả trong ảnh 2.
  - Nó tính toán khoảng cách (mức độ khác biệt) cho mỗi phép so sánh.
  - Nó lặp lại điều này cho tất cả thành phần mô tả trong ảnh 1.
- Thực thi:
  - `bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)`:
    - `cv2.NORM_HAMMING`: Đây là tham số quan trọng nhất. Vì ORB tạo ra vectơ mô tả nhị phân (là chuỗi bit), chúng ta không thể dùng khoảng cách Euclid. Thay vào đó, chúng ta dùng khoảng cách Hamming. Khoảng cách Hamming giữa hai chuỗi bit là số lượng bit khác nhau giữa chúng. Nó cực kỳ nhanh để tính toán.
    - `crossCheck=False`: Chúng ta đặt là False vì chúng ta sẽ tự thực hiện một phương pháp lọc tinh vi hơn (Lowe's Ratio Test). (Nếu đặt True, nó sẽ tự động chỉ giữ lại các cặp (A, B) nếu A là đối sánh tốt nhất của B và B là đối sánh tốt nhất của A).

- `matches = bf.knnMatch(des1, des2, k=2)`:
  - Chúng ta sử dụng `knnMatch` (k-nearest neighbors match) thay vì `match`.
  - `k=2` yêu cầu `cv2.BFMatcher`, với mỗi thành phần mô tả trong `des1`, tìm ra 2 thành phần mô tả gần nhất (tốt nhất) trong `des2`.
  - Kết quả `matches` là một danh sách các cặp. Mỗi cặp chứa 2 đối tượng `DMatch`: `m` (đối sánh tốt nhất, khoảng cách nhỏ nhất) và `n` (đối sánh tốt thứ hai, khoảng cách nhỏ nhì).
  - Chúng ta cần 2 đối sánh này để thực hiện Ratio Test.

d. Áp dụng Ratio Test của Lowe để lọc và giữ lại những cặp đối sánh tốt nhất (good matches):

- Đối sánh Brute-Force sẽ luôn tìm ra một đối sánh "tốt nhất" cho mọi điểm đặc trưng, ngay cả khi điểm đặc trưng đó không thực sự tồn tại trong ảnh kia (ví dụ: một điểm đặc trưng trên nền trời ở ảnh 1 không có ở ảnh 2). Điều này tạo ra rất nhiều đối sánh sai.
- Một "đối sánh tốt" phải là một đối sánh duy nhất và rõ ràng. Nếu đối sánh tốt nhất (m) chỉ hơi tốt hơn đối sánh tốt thứ hai (n), thì khả năng cao đây là một đối sánh không rõ ràng và có thể là sai. Ngược lại, nếu đối sánh tốt nhất (m) tốt hơn đáng kể (khoảng cách nhỏ hơn nhiều) so với đối sánh tốt thứ hai (n), thì ta có thể tin tưởng rằng đây là một đối sánh chính xác.

Tạo danh sách các đối sánh tốt:

```
good_matches = []
```

Đặt ngưỡng tỷ lệ:

```
ratio_thresh = 0.75
```

Lọc kết quả:

```
for m, n in matches:
    if m.distance < ratio_thresh * n.distance:
        good_matches.append(m)
```

- Chúng ta lặp qua danh sách `matches` đó. Trong mỗi vòng lặp:
    - `m`: Là đối sánh tốt nhất. Nó có khoảng cách `m.distance` nhỏ nhất.
    - `n`: Là đối sánh tốt thứ hai. Nó có khoảng cách `n.distance` nhỏ thứ hai.
- Chỉ chấp nhận đối sánh `m` nếu khoảng cách (độ sai khác) của nó nhỏ hơn 75% khoảng cách của đối sánh tốt thứ hai (`n`):
    - Trường hợp 1 (Đối sánh tốt): `m` tốt hơn `n` rất nhiều. Đây là một đặc trưng rõ ràng, đáng tin cậy. Chúng ta giữ nó lại.
    - Trường hợp 2 (Đối sánh xấu/Không rõ ràng): `m` chỉ tốt hơn `n` một chút xíu. Rất có thể đặc trưng này không độc nhất và `m` có thể là một đối sánh sai. Chúng ta loại bỏ nó.
- Nếu đối sánh `m` vượt qua được bài kiểm tra tỷ lệ, chúng ta mới coi nó là một đối sánh tốt và thêm nó vào danh sách `good_matches` cuối cùng.
- Chúng ta chỉ thêm `m` (đối sánh tốt nhất), còn `n` (đối sánh tốt thứ hai) sẽ bị loại bỏ hoàn toàn.

e. Hiển thị kết quả bằng cách vẽ các đường nối giữa các cặp đặc trưng đã được đối sánh tốt trên hai ảnh:

```
img_matches = cv2.drawMatches(img1, kp1, img2, kp2, good_matches,
None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
```

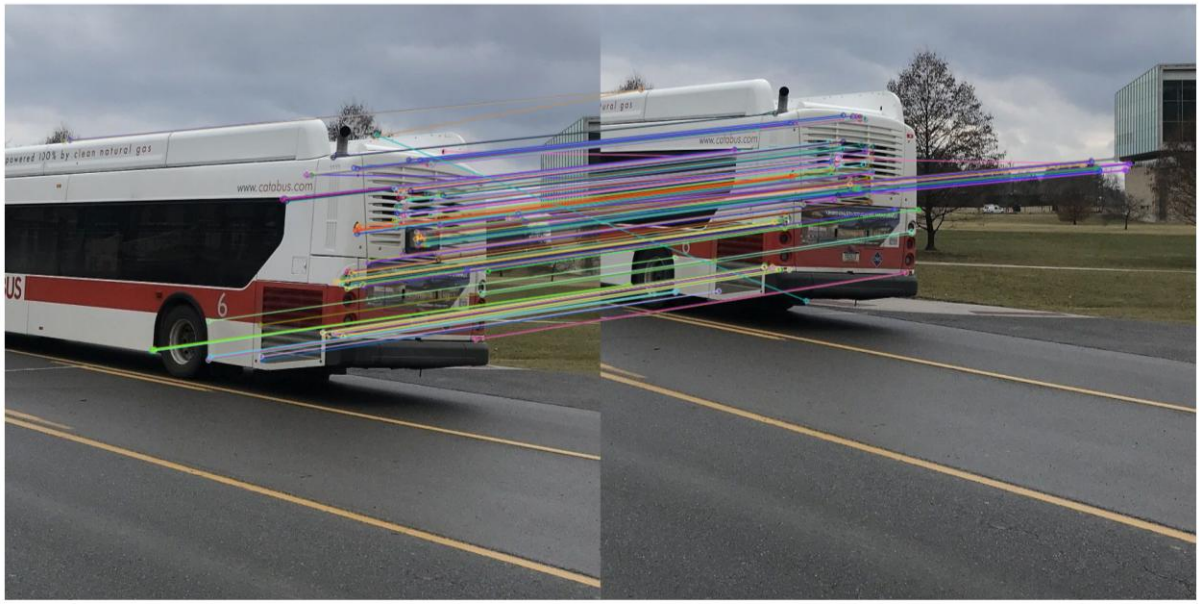
- Sử dụng hàm cv2.drawMatches() để vẽ kết quả.
- Các tham số của hàm:
    - `img1, kp1`: Ảnh 1 và các điểm đặc trưng của nó.
    - `img2, kp2`: Ảnh 2 và các điểm đặc trưng của nó.
    - `good_matches`: Danh sách các đối sánh tốt đã lọc ở phần d.
    - `None`: Một mặt nạ (mask) đầu ra, do không dùng nên để là None.
    - `flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS`: Cờ này yêu cầu hàm không vẽ các điểm đặc trưng không có đối sánh. Điều này làm cho hình ảnh kết quả sạch sẽ và dễ đọc hơn, chỉ tập trung vào các đường nối thành công.
- Ảnh đầu ra sẽ hiển thị hai ảnh đặt cạnh nhau. Mỗi đường kẻ màu nối một điểm đặc trưng ở ảnh 1 với một điểm đặc trưng ở ảnh 2.

Chuyển đổi màu cho ảnh kết quả:

```
img_matches_rgb = cv2.cvtColor(img_matches, cv2.COLOR_BGR2RGB)
```

- Vì OpenCV tải ảnh dưới dạng BGR nên cần chuyển đổi từ BGR sang RGB để màu sắc hiển thị chính xác.

Kết quả:



Nhận xét, phân tích:

- Mật độ đối sánh tốt cao: Một số lượng lớn các cặp đặc trưng đã được đối sánh thành công, thể hiện qua mật độ dày đặc của các đường nối giữa hai ảnh.
- Độ chính xác vị trí: Hầu hết các đường nối đều liên kết chính xác các điểm tương đồng trên đối tượng (xe buýt) ở cả hai ảnh.
- Hai bức ảnh được chụp từ hai góc nhìn và có sự thay đổi nhẹ về tỷ lệ khi chiếc xe buýt di chuyển. Các đường đối sánh đã tuân thủ chính xác sự thay đổi về phối cảnh này. Thay vì song song tuyệt đối, các đường nối có xu hướng hội tụ, phản ánh đúng sự thay đổi 3D của đối tượng.
- Lowe's Ratio Test với `ratio_thresh = 0.75` đã hoạt động cực kỳ hiệu quả trong việc lọc bỏ các cặp đối sánh không rõ ràng hoặc sai lệch, chỉ giữ lại những đối sánh có độ tin cậy cao nhất.

- Các điểm đặc trưng (điểm đầu cuối của các đường nối) được phát hiện bởi thuật toán ORB tập trung một cách hợp lý vào các khu vực có nhiều thông tin và độ tương phản cao.
- Các vùng có bề mặt đồng nhất, ít chi tiết (như phần thân xe màu trắng trơn hoặc mặt đường màu xám) có rất ít điểm đặc trưng được phát hiện.
- Kết quả đối sánh này rất tốt. Sự kết hợp của bộ phát hiện ORB và bộ lọc Lowe's Ratio Test đã chứng minh khả năng xử lý mạnh mẽ trước các thay đổi về góc nhìn và tỷ lệ.

Câu 4:

a. So sánh kết quả giữa bộ phát hiện góc Harris (Câu 1) và các bộ phát hiện điểm đặc trưng SIFT/ORB (Câu 2):

Kết quả thực nghiệm cho thấy sự khác biệt rõ rệt giữa bộ phát hiện góc Harris và các bộ phát hiện đặc trưng như SIFT/ORB, bắt nguồn từ bản chất và mục tiêu của thuật toán:

- Về bản chất:
    - Bộ phát hiện góc Harris: Là một bộ phát hiện góc, hoạt động dựa trên ma trận tự tương quan để tìm các điểm có sự thay đổi lớn về cường độ theo cả hai hướng trực giao. Kết quả đầu ra của Harris chỉ là tọa độ của các góc được phát hiện.
    - SIFT/ORB: Là các bộ phát hiện và mô tả đặc trưng. Chúng không chỉ xác định các điểm đặc trưng ổn định mà còn tính toán một vector mô tả cho vùng lân cận của điểm đó. Bộ mô tả này có tính bất biến với các phép biến đổi như tỷ lệ, xoay (SIFT, ORB) và thay đổi ánh sáng (SIFT).
- Về số lượng và vị trí:
    - Bộ phát hiện góc Harris: Như kết quả ở **Câu 1**, Harris có xu hướng phát hiện một số lượng rất lớn các điểm, tập trung chủ yếu tại các góc cạnh hình học rõ ràng (ví dụ: các góc ban công, giao điểm của các cạnh tòa nhà). Số lượng này rất nhạy cảm với tham số ngưỡng được cài đặt.
    - SIFT/ORB: Các thuật toán này tìm kiếm các điểm không chỉ là góc mà còn có tính ổn định cao qua các phép biến đổi (ví dụ: các "blob" hoặc các vùng có texture độc đáo). Do đó, số lượng đặc trưng SIFT/ORB phát hiện được thường ít hơn so với tổng số góc mà Harris

tìm thấy, nhưng các điểm này mang nhiều thông tin hơn (nhờ có bộ mô tả) và có thể không hoàn toàn trùng khớp với vị trí góc hình học của Harris.

b. So sánh giữa SIFT và ORB:

Việc áp dụng hai thuật toán SIFT và ORB trên cùng một tập dữ liệu ảnh cho thấy sự đánh đổi giữa số lượng đặc trưng và hiệu suất tính toán:

- Về số lượng đặc trưng:
    - SIFT hầu như luôn tạo ra số lượng đặc trưng nhiều hơn đáng kể so với ORB. Nguyên nhân là do SIFT thực hiện một cuộc tìm kiếm toàn diện trên không gian tỷ lệ bằng cách sử dụng hàm DoG (Difference of Gaussians), cho phép nó phát hiện các đặc trưng ở nhiều kích thước khác nhau.
    - ORB sử dụng bộ phát hiện FAST, vốn được tối ưu cho tốc độ, nên có thể bỏ sót một số đặc trưng mà SIFT phát hiện được.
- Thuật toán ORB chạy nhanh hơn so với SIFT.

c. Vai trò của việc lọc các cặp đối sánh bằng Ratio Test ở Câu 3 và kết quả nếu bỏ qua bước lọc này:

- Vai trò: Loại bỏ các cặp đối sánh không rõ ràng và các cặp đối sánh sai.
- Kết quả nếu bỏ qua bước lọc:
    - Tổng số lượng cặp đối sánh sẽ tăng lên đáng kể, nhưng phần lớn các cặp tăng thêm này là các điểm ngoại lai hoặc các đối sánh sai.
    - Sự gia tăng đột biến của các cặp sai này (nhiễu) sẽ gây ảnh hưởng nghiêm trọng đến các bước xử lý hậu kỳ.