ChatGPT

Tạo ảnh panorama từ 3 ảnh sử dụng SIFT

Giới thiệu

Trong bài toán này, chúng ta sẽ ghép ba ảnh riêng lẻ thành một ảnh panorama bằng kỹ thuật **SIFT** (Scale-Invariant Feature Transform). SIFT cho phép phát hiện các điểm đặc trưng nổi bật trên ảnh và mô tả chúng sao cho có thể so sánh giữa các ảnh. Bằng cách tìm các cặp điểm tương ứng giữa các ảnh chồng lấn, chúng ta có thể tính toán phép biến đổi phối cảnh (**homography**) để căn chỉnh các ảnh với nhau.

Các bước thực hiện:

1. **Đọc ảnh đầu vào:** Đọc ba ảnh `image_left_side.jpg`, `image_middle.jpg`, `image_right_side.jpg` từ thư mục chỉ định.
2. **Phát hiện đặc trưng SIFT:** Với mỗi ảnh, dùng SIFT để phát hiện các keypoint (điểm đặc trưng) và mô tả chúng.
3. **So khớp đặc trưng giữa các ảnh chồng lấn:**
4. Ghép ảnh trái với ảnh giữa: Tìm các cặp điểm tương ứng giữa ảnh trái và ảnh giữa bằng mô tả SIFT, sử dụng BFMatcher và áp dụng tiêu chí Lowe's ratio test để lọc match tốt. Từ các cặp điểm, tính toán homography (phép biến đổi phối cảnh) sử dụng RANSAC để đảm bảo độ chính xác.
5. Ghép ảnh phải với ảnh giữa: Tương tự, tìm các điểm tương ứng giữa ảnh phải và ảnh giữa, rồi tính homography.
6. **Warp và ghép ảnh:**
7. Sử dụng homography ảnh trái→ảnh giữa để **warp** (chiếu phối cảnh) ảnh trái sang hệ tọa độ của ảnh giữa, sau đó kết hợp ảnh trái đã warp với ảnh giữa trên cùng một canvas. Kết quả thu được là panorama tạm thời của ảnh trái và ảnh giữa.
8. Sử dụng homography ảnh phải→ảnh giữa để warp ảnh phải sang hệ tọa độ ảnh giữa, rồi ghép tiếp vào panorama tạm thời.
9. **Kết quả:** Ảnh panorama hoàn chỉnh bao gồm cả ba ảnh. Ảnh này sẽ được hiển thị và lưu ra file kết quả.

Các bước trung gian (keypoints, match, warp) sẽ được hiển thị để người đọc theo dõi quá trình ghép ảnh. Bây giờ chúng ta sẽ đi vào từng bước chi tiết.

Đọc ảnh đầu vào

Trước hết, chúng ta đọc ba file ảnh đầu vào từ thư mục đã cho. Sử dụng OpenCV (`cv2.imread`) để đọc ảnh màu. Đồng thời, hiển thị kích thước mỗi ảnh để xác nhận đã đọc thành công.

```
import cv2

# Đường dẫn thư mục chứa ảnh panorama
image_dir = r"C:\Users\Public\panorama_sift_three_pictures"

# Đọc ảnh màu từ các file
left_img   = cv2.imread(f"{image_dir}\image_left_side.jpg")
```

```
middle_img = cv2.imread(f"{image_dir}\image_middle.jpg")
right_img  = cv2.imread(f"{image_dir}\image_right_side.jpg")

# Kiểm tra kích thước ảnh đã đọc
print("Kích thước ảnh trái:", left_img.shape)
print("Kích thước ảnh giữa:", middle_img.shape)
print("Kích thước ảnh phải:", right_img.shape)
```

Giải thích: Đoạn code trên dùng `cv2.imread` để đọc ảnh từ đường dẫn. Biến `left_img`, `middle_img`, `right_img` là các numpy array 3 chiều (height, width, BGR channels). Việc in ra `shape` của mỗi ảnh giúp xác nhận ảnh được đọc đúng (ví dụ như cả ba ảnh có cùng độ phân giải). Tiếp theo, chúng ta sẽ tiến hành phát hiện các điểm đặc trưng SIFT trên từng ảnh.

Phát hiện và hiển thị keypoints SIFT trên từng ảnh

Chúng ta sử dụng thuật toán SIFT để tìm các keypoint (đặc trưng) trên mỗi ảnh. OpenCV cung cấp lớp `cv2.SIFT_create()` để tạo đối tượng SIFT. Sau đó, dùng phương thức `detectAndCompute` để tìm keypoints và mô tả (descriptor) cho mỗi ảnh.

Để trực quan, ta sẽ vẽ các keypoint này lên ảnh (dùng `cv2.drawKeypoints`) và hiển thị bằng Matplotlib. Mỗi keypoint được đánh dấu bằng vòng tròn nhỏ trên ảnh tại vị trí nó được phát hiện. Điều này cho thấy các điểm SIFT phát hiện được thường tập trung ở các vùng có đặc trưng như góc cạnh, họa tiết nổi bật trong ảnh.

Ảnh bên trái (image_left_side.jpg) - Keypoints SIFT

```
import matplotlib.pyplot as plt

# Tạo đối tượng SIFT
sift = cv2.SIFT_create()

# Ảnh bên trái: chuyển sang grayscale và phát hiện keypoint
gray_left = cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY)
kp_left, des_left = sift.detectAndCompute(gray_left, None)
print(f"Số lượng keypoint SIFT trên ảnh trái: {len(kp_left)}")

# Vẽ các keypoint SIFT lên ảnh màu
img_left_keypoints = cv2.drawKeypoints(left_img, kp_left, None,
flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)

# Hiển thị ảnh với keypoints
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(img_left_keypoints, cv2.COLOR_BGR2RGB))
plt.title("Keypoints SIFT trên ảnh trái")
plt.axis('off')
plt.show()
```

Giải thích: Trên đây, ảnh trái được chuyển sang mức xám (`cv2.cvtColor`) trước khi detect SIFT vì SIFT hoạt động trên ảnh xám. Hàm `detectAndCompute` trả về danh sách `kp_left` chứa các keypoint và

ma trận `des_left` chứa descriptor tương ứng (vector 128 chiều mô tả vùng lân cận mỗi keypoint). Kết quả in ra cho thấy số lượng keypoint SIFT phát hiện được trên ảnh trái. Sau đó, `cv2.drawKeypoints` dùng để vẽ các vị trí keypoint lên ảnh (mặc định với chấm tròn nhỏ màu). Hình vẽ cho thấy các điểm SIFT tập trung ở các vùng chi tiết như góc tòa nhà, mái nhà, v.v.

Ảnh ở giữa (image_middle.jpg) - Keypoints SIFT

```
# Ảnh ở giữa: phát hiện keypoint SIFT
gray_middle = cv2.cvtColor(middle_img, cv2.COLOR_BGR2GRAY)
kp_middle, des_middle = sift.detectAndCompute(gray_middle, None)
print(f"Số lượng keypoint SIFT trên ảnh giữa: {len(kp_middle)}")

# Vẽ keypoint lên ảnh giữa
img_mid_keypoints = cv2.drawKeypoints(middle_img, kp_middle, None,
flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)

# Hiển thị ảnh giữa với các keypoint SIFT
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(img_mid_keypoints, cv2.COLOR_BGR2RGB))
plt.title("Keypoints SIFT trên ảnh giữa")
plt.axis('off')
plt.show()
```

Giải thích: Tương tự, chúng ta phát hiện được rất nhiều keypoint trên ảnh giữa (in ra số lượng). Ảnh giữa có chung vùng cảnh với ảnh trái và ảnh phải, do đó các keypoint ở rìa trái ảnh giữa có thể trùng khớp với keypoint ở rìa phải ảnh trái, và keypoint ở rìa phải ảnh giữa có thể trùng với ảnh phải. Việc vẽ lên ảnh cho thấy các điểm đặc trưng phân bố trên toàn bộ bức ảnh, tập trung ở những chỗ giàu chi tiết (như cửa sổ, họa tiết trên tòa nhà, etc.).

Ảnh bên phải (image_right_side.jpg) - Keypoints SIFT

```
# Ảnh bên phải: phát hiện keypoint SIFT
gray_right = cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY)
kp_right, des_right = sift.detectAndCompute(gray_right, None)
print(f"Số lượng keypoint SIFT trên ảnh phải: {len(kp_right)}")

# Vẽ keypoint lên ảnh phải
img_right_keypoints = cv2.drawKeypoints(right_img, kp_right, None,
flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)

# Hiển thị ảnh phải với các keypoint SIFT
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(img_right_keypoints, cv2.COLOR_BGR2RGB))
plt.title("Keypoints SIFT trên ảnh phải")
plt.axis('off')
plt.show()
```

Giải thích: Ảnh bên phải cũng được xử lý tương tự, cho thấy số lượng keypoint SIFT tìm được. Nhìn vào hình vẽ, ta thấy các keypoint tập trung ở mép bên trái của ảnh phải (vùng tòa nhà, con đường) – đây là

phần có thể trùng với mép bên phải của ảnh giữa. Sau khi đã có các đặc trưng SIFT cho cả ba ảnh, ta sẽ tiến hành ghép ảnh. Tiếp theo, chúng ta ghép ảnh trái và ảnh giữa trước, sau đó ghép thêm ảnh phải.

Ghép ảnh trái với ảnh giữa

Tìm kiếm match giữa ảnh trái và ảnh giữa

Để ghép ảnh, trước tiên cần tìm các điểm tương ứng giữa ảnh trái và ảnh giữa. Chúng ta sử dụng **BFMatcher** (Brute-Force Matcher) của OpenCV để so sánh các descriptor SIFT giữa hai ảnh. Với mỗi keypoint ở ảnh trái, BFMatcher sẽ tìm 2 keypoint gần nhất (dựa trên khoảng cách Euclidean giữa các vector descriptor) trong ảnh giữa. Sau đó áp dụng tiêu chí Lowe's ratio test: chỉ chấp nhận những match mà khoảng cách của match tốt nhất nhỏ hơn 75% khoảng cách của match tốt thứ hai. Điều này giúp loại bỏ các match không đáng tin cậy.

Sau khi lọc, chúng ta sử dụng `cv2.findHomography` để tính homography (phép biến đổi phối cảnh) giữa ảnh trái và ảnh giữa. Homography được tính từ các cặp điểm tương ứng (tọa độ trong ảnh trái và tọa độ trong ảnh giữa) và dùng RANSAC để loại bỏ các outlier (những match sai lệch). Kết quả homography là một ma trận 3x3 cho phép biến đổi điểm từ ảnh trái sang ảnh giữa.

Chúng ta cũng sẽ trực quan hóa một số match tốt giữa hai ảnh bằng cách vẽ các đường nối giữa cặp điểm tương ứng (dùng `cv2.drawMatches`). Điều này giúp kiểm chứng rằng homography được tính dựa trên các vùng chồng lấn hợp lý (ví dụ: các góc tòa nhà, cửa sổ trùng nhau giữa hai ảnh).

```
# Tìm match giữa ảnh trái và ảnh giữa
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)

# Tìm 2 match gần nhất cho mỗi descriptor của ảnh trái
matches_left_mid = bf.knnMatch(des_left, des_middle, k=2)

# Áp dụng Lowe's ratio test để lọc match tốt
good_matches_lm = []
ratio_thresh = 0.75
for m, n in matches_left_mid:
    if m.distance < ratio_thresh * n.distance:
        good_matches_lm.append(m)
print(f"Số match tốt giữa ảnh trái và ảnh giữa: {len(good_matches_lm)}")

# Tính homography sử dụng các match tốt
if len(good_matches_lm) >= 4:
    # Lấy tọa độ các điểm match
    pts_left  = [kp_left[m.queryIdx].pt for m in good_matches_lm]
    pts_mid   = [kp_middle[m.trainIdx].pt for m in good_matches_lm]
    pts_left  = np.array(pts_left, dtype=np.float32)
    pts_mid   = np.array(pts_mid, dtype=np.float32)
    # Tính H (homography) sao cho H * (point_left) = point_middle
    H_left_to_mid, mask = cv2.findHomography(pts_left, pts_mid, cv2.RANSAC)
    print("Homography (ảnh trái -> ảnh giữa):")
    print(H_left_to_mid)
else:
    H_left_to_mid = None
```

```
    print("Không đủ match để tính homography!")

# Vẽ 50 match tốt đầu tiên để minh họa
matches_draw = good_matches_lm[:50]  # lấy 50 match đầu (nếu ít hơn 50 thì
lấy tất cả)
match_img = cv2.drawMatches(left_img, kp_left, middle_img, kp_middle,
matches_draw, None,

flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
# Chuyển ảnh match BGR->RGB và hiển thị
plt.figure(figsize=(8,6))
plt.imshow(cv2.cvtColor(match_img, cv2.COLOR_BGR2RGB))
plt.title("Các match SIFT giữa ảnh trái và ảnh giữa")
plt.axis('off')
plt.show()
```

Giải thích: Kết quả in ra cho biết số lượng match tốt (sau khi áp dụng ratio test) giữa ảnh trái và ảnh giữa. Nếu có đủ 4 cặp điểm trở lên, `cv2.findHomography` sẽ tính được homography `H_left_to_mid`. Ma trận homography in ra ở trên (nếu có) cho ta ý tưởng về phép biến đổi: các phần tử bao gồm phép xoay, tỷ lệ và tịnh tiến cần thiết để chồng ảnh trái lên ảnh giữa.

Hình ảnh trực quan bên trên nối các cặp keypoint tương ứng giữa hai ảnh. Ta có thể thấy các đường nối tập trung ở khu vực chung giữa hai ảnh (ví dụ: phần tòa nhà và cảnh quan chồng lấn). Điều này xác nhận các match tìm được là hợp lý. Những match này sẽ được dùng để căn chỉnh ảnh trái với ảnh giữa.

Warp ảnh trái và kết hợp với ảnh giữa

Với homography `H_left_to_mid` vừa tính, chúng ta tiến hành **warp** ảnh trái sang hệ tọa độ của ảnh giữa. Nói cách khác, chúng ta biến đổi phối cảnh ảnh trái sao cho nó thẳng hàng với ảnh giữa (các điểm tương ứng trùng khớp vị trí nhau).

Cụ thể, dùng `cv2.warpPerspective` với ma trận homography để biến đổi ảnh trái. Kết quả thu được là ảnh trái đã được biến đổi, đặt trong không gian tọa độ của ảnh giữa. Sau đó, chúng ta cần tạo một **canvas** đủ lớn để chứa cả ảnh giữa gốc và ảnh trái đã warp. Do ảnh trái có thể mở rộng ra ngoài ranh giới ảnh giữa (ví dụ phần bên trái ảnh), ta sẽ tính toán kích thước canvas bằng cách xét tọa độ các góc ảnh sau khi biến đổi.

Các bước thực hiện warp và ghép:

1. Sử dụng `cv2.perspectiveTransform` để biến đổi 4 góc của mỗi ảnh (trái, giữa) bằng homography, từ đó xác định phạm vi (min/max x, y) của ảnh sau khi warp trong hệ tọa độ chung.
2. Xác định kích thước của canvas dựa trên min/max toạ độ này. Đồng thời tính toán độ dời (offset) cần thiết nếu có toạ độ âm (tức là ảnh trái mở rộng về phía bên trái hoặc phía trên gốc toạ độ của ảnh giữa).
3. Khởi tạo canvas (ảnh nền) màu đen với kích thước tính được, sau đó vẽ ảnh giữa vào đúng vị trí (ảnh giữa đóng vai trò hệ tọa độ gốc, nhưng nếu có offset ta cũng cần dịch ảnh giữa).
4. Vẽ ảnh trái đã warp vào canvas tại vị trí tương ứng (áp dụng offset nếu có). Trong vùng chồng lấn giữa hai ảnh, có thể xảy ra việc cả hai ảnh đều có nội dung. Ở đây để đơn giản, ta sẽ ưu tiên giữ lại nội dung của ảnh giữa (tránh ghi đè ảnh trái lên vùng đã có ảnh giữa).

Sau bước này, chúng ta sẽ thu được một ảnh panorama tạm thời kết hợp ảnh trái và ảnh giữa.

```
import numpy as np

# Nếu tính được homography H_left_to_mid, tiến hành warp
if H_left_to_mid is not None:
    # Kích thước ảnh
    h_left, w_left = left_img.shape[:2]
    h_mid,  w_mid  = middle_img.shape[:2]

    # Tọa độ 4 góc của ảnh trái và ảnh giữa trong hệ tọa độ gốc của chúng
    corners_left = np.float32([[0,0], [w_left,0], [0,h_left], [w_left,
h_left]]).reshape(-1,1,2)
    corners_mid  = np.float32([[0,0], [w_mid,0], [0,h_mid], [w_mid,
h_mid]]).reshape(-1,1,2)

    # Chuyển đổi tọa độ góc ảnh trái sang hệ ảnh giữa bằng homography
    corners_left_warp = cv2.perspectiveTransform(corners_left, H_left_to_mid)
    # Góc ảnh giữa trong hệ tọa độ ảnh giữa (identity transform)
    corners_mid_warp  = corners_mid

    # Gộp tất cả các tọa độ góc để tính khung bao lớn nhất
    all_corners = np.concatenate((corners_left_warp, corners_mid_warp),
axis=0)
    [x_min, y_min] = all_corners.min(axis=0).ravel()
    [x_max, y_max] = all_corners.max(axis=0).ravel()

    # Tính kích thước canvas (chú ý làm tròn lên và trừ đi giá trị âm nếu có)
    x_min, y_min = int(np.floor(x_min)), int(np.floor(y_min))
    x_max, y_max = int(np.ceil(x_max)),  int(np.ceil(y_max))
    canvas_width  = x_max - x_min
    canvas_height = y_max - y_min

    # Tạo ma trận dịch chuyển (offset) để đảm bảo tọa độ không âm
    # (dịch chuyển ảnh về phải x=-x_min, về dưới y=-y_min)
    offset_matrix = np.array([[1, 0, -x_min],
                              [0, 1, -y_min],
                              [0, 0, 1]], dtype=float)

    # Warp ảnh trái sang hệ tọa độ ảnh giữa + áp dụng offset
    left_warp = cv2.warpPerspective(left_img,
offset_matrix.dot(H_left_to_mid), (canvas_width, canvas_height))
    # Warp ảnh giữa (thực ra chỉ áp dụng offset vì ảnh giữa là gốc)
    middle_warp = cv2.warpPerspective(middle_img, offset_matrix,
(canvas_width, canvas_height))

    # Kết hợp ảnh trái đã warp và ảnh giữa trên canvas
    panorama_left_mid = middle_warp.copy()
    # Với mỗi pixel, nếu pixel đó trống (đen) trong ảnh giữa thì lấy từ ảnh
trái
```

```
    mask_overlap = (panorama_left_mid == 0) & (left_warp != 0)
    panorama_left_mid[mask_overlap] = left_warp[mask_overlap]

    # Hiển thị kết quả panorama tạm thời (trái + giữa)
    plt.figure(figsize=(8,5))
    plt.imshow(cv2.cvtColor(panorama_left_mid, cv2.COLOR_BGR2RGB))
    plt.title("Panorama tạm thời (ảnh trái + ảnh giữa)")
    plt.axis('off')
    plt.show()
```

Giải thích: Trước tiên, ta tính `corners_left_warp` – tọa độ các góc ảnh trái sau khi biến đổi sang hệ ảnh giữa. Dựa trên toạ độ min/max của góc ảnh trái **và** góc ảnh giữa, ta xác định được kích thước vùng không gian cần thiết để chứa cả hai ảnh sau khi ghép. Nếu `x_min` hoặc `y_min` âm, điều đó có nghĩa ảnh trái vươn ra bên trái hoặc bên trên so với ảnh giữa, vì vậy ta cần dịch toàn bộ phối cảnh sang phải/xuống dưới bằng cách nhân với `offset_matrix`.

Sau khi warp: - `left_warp` là ảnh trái đã được biến đổi phối cảnh và đặt trên một nền đen kích thước canvas. - `middle_warp` là ảnh giữa (chưa biến đổi, chỉ áp dụng phép dịch) trên cùng canvas.

Tiếp đó, ta kết hợp hai ảnh: bắt đầu từ `panorama_left_mid` là ảnh giữa, rồi chồng ảnh trái đã warp lên. Vùng nào ảnh giữa trống (màu đen) thì lấy nội dung từ ảnh trái. Làm như vậy đảm bảo những chỗ chung giữa hai ảnh sẽ giữ lại ảnh giữa (tránh đè bằng ảnh trái, phòng trường hợp có sai khác nhỏ). Kết quả là `panorama_left_mid` – ảnh panorama ghép từ ảnh trái và ảnh giữa. Hình ảnh hiển thị cho thấy hai ảnh đã trùng khớp ở phần chồng lấn (ta có thể kiểm chứng bằng cách so sánh tòa nhà, hàng cây ở vùng nối tiếp giữa hai ảnh).

Ghép thêm ảnh phải vào panorama

Sau khi có panorama tạm (trái + giữa), chúng ta ghép tiếp ảnh phải vào. Thay vì so khớp ảnh phải với panorama tạm thời (vì panorama không có thông tin keypoint dễ sử dụng trực tiếp), ta sẽ tiếp cận bằng cách so khớp ảnh phải với ảnh giữa (vốn đã có đặc trưng SIFT). Ảnh giữa đóng vai trò "cầu nối" giữa trái và phải. Do ảnh giữa và ảnh phải có vùng chồng lấn, ta sẽ tìm match SIFT giữa ảnh phải và ảnh giữa, tương tự như bước làm với ảnh trái.

Tìm kiếm match giữa ảnh phải và ảnh giữa

```
# Tìm match giữa ảnh phải và ảnh giữa
matches_right_mid = bf.knnMatch(des_right, des_middle, k=2)

# Lọc match bằng Lowe's ratio test
good_matches_rm = []
for m, n in matches_right_mid:
    if m.distance < ratio_thresh * n.distance:
        good_matches_rm.append(m)
print(f"Số match tốt giữa ảnh phải và ảnh giữa: {len(good_matches_rm)}")

# Tính homography (ảnh phải -> ảnh giữa)
if len(good_matches_rm) >= 4:
    pts_right = np.float32([kp_right[m.queryIdx].pt for m in
```

```
good_matches_rm])
    pts_mid2  = np.float32([kp_middle[m.trainIdx].pt for m in
good_matches_rm])
    H_right_to_mid, mask2 = cv2.findHomography(pts_right, pts_mid2,
cv2.RANSAC)
    print("Homography (ảnh phải -> ảnh giữa):")
    print(H_right_to_mid)
else:
    H_right_to_mid = None
    print("Không đủ match để tính homography cho ảnh phải!")

# Vẽ 50 match tốt giữa ảnh phải và ảnh giữa
matches_draw2 = good_matches_rm[:50]
match_img2 = cv2.drawMatches(right_img, kp_right, middle_img, kp_middle,
matches_draw2, None,

flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.figure(figsize=(8,6))
plt.imshow(cv2.cvtColor(match_img2, cv2.COLOR_BGR2RGB))
plt.title("Các match SIFT giữa ảnh phải và ảnh giữa")
plt.axis('off')
plt.show()
```

Giải thích: Tương tự như trước, chúng ta dùng BFMatcher để tìm các match giữa descriptor của ảnh phải và ảnh giữa. Áp dụng ratio test thu được `good_matches_rm` – danh sách các match tốt. Dựa trên đó, tính homography `H_right_to_mid` (ma trận biến đổi ảnh phải sang ảnh giữa). Số lượng match tốt cũng như ma trận homography được in ra để tham khảo.

Hình ảnh trực quan cho thấy các match kết nối ảnh phải và ảnh giữa, chủ yếu nằm ở vùng chồng lấn (ví dụ: phần bên phải tòa nhà và con đường trong ảnh giữa trùng với phần bên trái của ảnh phải). Những match này xác nhận rằng homography sẽ căn chỉnh đúng vùng chung đó.

Warp ảnh phải vào panorama và hoàn thiện ảnh ghép

Bây giờ, sử dụng homography `H_right_to_mid`, chúng ta warp ảnh phải sang hệ tọa độ ảnh giữa (cũng là hệ của panorama tạm thời hiện tại). Sau đó, ghép ảnh phải này vào **panorama_left_mid**.

Quy trình tương tự như khi ghép ảnh trái: tính toán kích thước canvas mới nếu cần để chứa thêm ảnh phải, áp dụng offset, và chèn ảnh phải vào. Tuy nhiên, ta có thể tái sử dụng canvas đã có từ bước trước (panorama tạm) vì nó đã đủ lớn để chứa ảnh trái và giữa; ta chỉ cần mở rộng nếu ảnh phải vươn ra ngoài. Do ở bước tính trước, ta đã gộp cả góc ảnh phải khi tính `all_corners` (bao gồm corners_right_warp), canvas của panorama tạm *đã* có kích thước đủ bao trùm cả ảnh phải. Vì vậy, ở đây ta sẽ sử dụng lại canvas đó.

Cuối cùng, ta kết hợp ảnh phải vào panorama: tại những vùng trống (đen) của panorama hiện tại, chèn pixel từ ảnh phải đã warp. Sau thao tác này, ta thu được ảnh panorama hoàn chỉnh ghép cả ba ảnh. Ta sẽ hiển thị ảnh panorama kết quả và lưu nó ra file.

```
if H_right_to_mid is not None:
    # Warp ảnh phải với homography và offset_matrix đã tính trước (canvas
chung)
    right_warp = cv2.warpPerspective(right_img,
offset_matrix.dot(H_right_to_mid), (canvas_width, canvas_height))

    # Ghép ảnh phải vào panorama tạm thời (trái+giữa)
    panorama_full = panorama_left_mid.copy()
    mask_overlap2 = (panorama_full == 0) & (right_warp != 0)
    panorama_full[mask_overlap2] = right_warp[mask_overlap2]

    # Hiển thị ảnh panorama hoàn chỉnh
    plt.figure(figsize=(10,6))
    plt.imshow(cv2.cvtColor(panorama_full, cv2.COLOR_BGR2RGB))
    plt.title("Ảnh panorama ghép hoàn chỉnh (trái + giữa + phải)")
    plt.axis('off')
    plt.show()

    # Lưu ảnh panorama ra file
    save_path = rf"{image_dir}\panorama_result.jpg"
    cv2.imwrite(save_path, panorama_full)
    print("Đã lưu ảnh panorama kết quả tại:", save_path)
```

Giải thích: Ảnh phải được biến đổi phối cảnh bằng `cv2.warpPerspective` tương tự như ảnh trái, sử dụng cùng `offset_matrix` (đã tính ở bước trước) để đảm bảo căn chỉnh đúng trên canvas chung. Kết quả `right_warp` là ảnh phải trên nền đen, thẳng hàng với ảnh giữa.

Chúng ta tạo `panorama_full` từ panorama tạm (trái+giữa) và thêm ảnh phải: duyệt qua mỗi pixel, nếu pixel đó đang trống (đen) trong panorama tạm thời và ảnh phải warp có nội dung tại đó thì chèn pixel từ ảnh phải. Điều này đảm bảo giữ nguyên nội dung ảnh trái/giữa ở vùng đã có, chỉ bổ sung những phần mới từ ảnh phải.

Kết quả thu được là ảnh panorama cuối cùng kết hợp cả ba ảnh. Hình hiển thị cho thấy toàn bộ cảnh rộng được tái tạo: phần bên trái từ ảnh trái, phần giữa từ ảnh giữa, và phần bên phải từ ảnh phải ghép lại liền mạch. Cuối cùng, ảnh panorama được lưu vào file `panorama_result.jpg` trong thư mục ban đầu. Bạn có thể mở file này để kiểm tra kết quả ghép ảnh panorama sử dụng SIFT.
