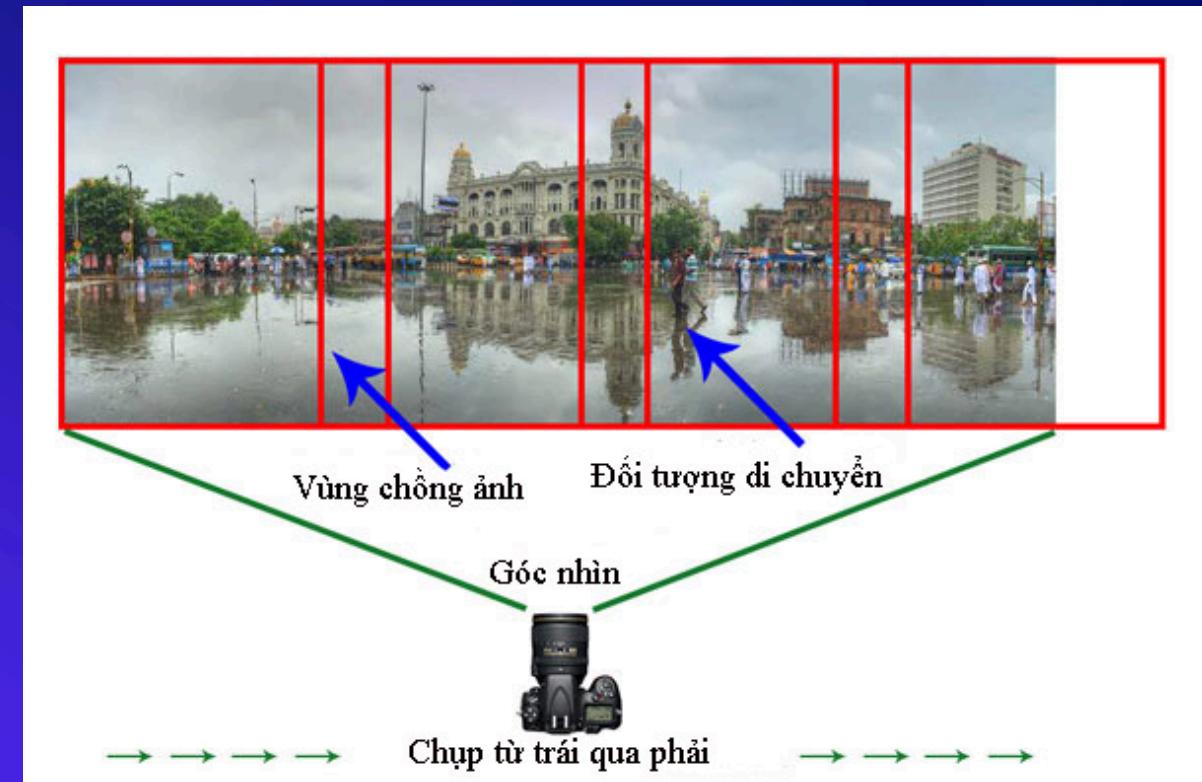


# GHÉP ẢNH PANORAMA SỬ DỤNG SIFT

THÀNH VIÊN:  
PHẠM HẢI DƯƠNG - B22DCCN169  
DƯƠNG VĂN THUẬN - B22DCCN841

Nhóm: 09

Bộ môn: Xử lý ảnh





1. GIỚI THIỆU ĐỀ TÀI

2. CƠ SỞ LÝ THUYẾT(SIFT, PANORAMA, HOMOGRAPHY)

3. THIẾT KẾ VÀ CÀI ĐẶT

4. ĐÁNH GIÁ



# LÝ DO CHỌN ĐỀ TÀI & MỤC TIÊU

## LÝ DO CHỌN ĐỀ TÀI:

- Nhu cầu ảnh góc rộng trong: du lịch, street view, bản đồ, camera...
- Ghép ảnh thủ công mất công → cần thuật toán tự động.

## MỤC TIÊU:

- Xây dựng chương trình ghép panorama từ 2 ảnh, 3 ảnh.
- Sử dụng SIFT + homography + warpPerspective.
- Đánh giá kết quả và rút ra ưu/nhược điểm.

# KHÁI NIỆM ẢNH PANORAMA & QUY TRÌNH CHUNG

## Ảnh panorama là gì?

Ảnh panorama là loại ảnh có góc nhìn rất rộng (có thể lên đến  $180^\circ$  hoặc  $360^\circ$ ), thu được bằng cách:

- Sử dụng ống kính góc rộng chuyên dụng.
- **Ghép (stitch) nhiều ảnh thường lại với nhau.**



# QUY TRÌNH GHÉP ẢNH

1. Tiền xử lý ảnh (nếu cần): chuyển sang ảnh xám, cân bằng sáng, lọc nhiễu...
2. Phát hiện và mô tả điểm đặc trưng trên từng ảnh: dùng SIFT
3. Ghép cặp đặc trưng giữa các ảnh: sử dụng BFMatcher kèm các tiêu chí lọc (ratio test).
4. Ước lượng phép biến đổi hình học giữa các ảnh: thường dùng homography  $3 \times 3$  cho trường hợp camera quay quanh trục và cảnh gần phẳng.
5. Warp (biến đổi) một ảnh sang hệ tọa độ của ảnh còn lại bằng phép biến đổi đã ước lượng.
6. Ghép và blending vùng chồng lấp để giảm đường nối (seam) lộ rõ.

# TIỀN XỬ LÝ ẢNH

– Mục đích: Tăng chất lượng đặc trưng.

## 1. Giảm nhiễu và tăng độ tương phản

- Nhiễu ảnh làm xuất hiện các điểm giả, gây sai lệch khi tìm keypoints.
- Lọc nhiễu (Gaussian Blur) giúp làm mượt ảnh, giữ lại chi tiết quan trọng.
- Cân bằng sáng (Histogram Equalization) làm nổi bật các vùng chi tiết, giúp đặc trưng dễ nhận diện.

## 4. Giảm lỗi ghép và tăng độ chính xác

- Đặc trưng tốt → matching chính xác hơn → homography tính đúng → ảnh ghép mượt.

## 2. Chuẩn hóa dữ liệu đầu vào

- Chuyển sang grayscale để giảm tính toán (thuật toán đặc trưng thường chỉ cần cường độ sáng).
- Đảm bảo ảnh có cùng định dạng và kích thước hợp lý → tránh sai lệch khi ghép.

## 3. Tăng độ ổn định khi phát hiện đặc trưng

- Nếu ảnh quá tối hoặc quá sáng, keypoints sẽ bị thiếu hoặc sai.
- Sau tiền xử lý, các điểm đặc trưng sẽ rõ ràng hơn, giúp thuật toán mô tả (descriptor) chính xác.

# PHÁT HIỆN & MÔ TẢ ĐẶC TRƯNG

## Phát hiện đặc trưng (Feature Detection)

1. Mục tiêu: Tìm các điểm nổi bật trong ảnh (keypoints) mà dễ nhận diện và phân biệt giữa các ảnh.

### 2. Đặc trưng tốt cần có:

- Ổn định với thay đổi: Góc nhìn, tỷ lệ, xoay, ánh sáng.
- Phân biệt cao: Không bị nhầm lẫn với vùng khác.

3. Ví dụ: Góc cạnh, điểm giao nhau, họa tiết nổi bật.

## Mô tả đặc trưng (Feature Description)

1. Mục tiêu: Biểu diễn mỗi keypoint bằng một vector đặc trưng (descriptor) để so sánh giữa các ảnh.

2. Đặc điểm: Descriptor phải bền vững trước biến đổi (xoay, scale).

3. Vai trò: Giúp ghép cặp đặc trưng giữa ảnh này và ảnh kia.

# GIỚI THIỆU SIFT (SCALE-INVARIANT FEATURE TRANSFORM)

- SIFT làm gì?
  - Phép biến đổi đặc trưng bất biến tỷ lệ (Scale-Invariant Feature Transform - SIFT) là một thuật toán được phát triển để trích xuất các đặc điểm bất biến từ một hình ảnh.
  - Điểm đặc trưng (keypoint) là những vị trí trong ảnh có tính chất nổi bật và ổn định, dễ nhận dạng và phân biệt so với vùng xung quanh
- Tính chất:
  - Bất biến tỉ lệ, quay, tương đối bền với thay đổi sáng + nhiễu.
- Đầu vào là ảnh xám.
- Đầu ra là tập các keypoint (tọa độ, tỉ lệ, hướng) và descriptor (vector 128 chiều mô tả vùng lân cận keypoint).
  - Descriptor không phải là ảnh, mà là một chuỗi số mô tả đặc trưng cục bộ quanh keypoint.
  - Nhờ descriptor, ta có thể so khớp keypoint giữa hai ảnh bằng cách so sánh độ giống nhau của hai vector

# CÁC BƯỚC CHÍNH CỦA SIFT

01

1. Xây dựng không gian tỷ lệ & Difference of Gaussian (DoG)

- Tạo nhiều phiên bản làm mờ của ảnh bằng bộ lọc Gaussian với các giá trị  $\sigma$  khác nhau.
- Tổ chức thành các octave (mỗi octave gồm nhiều mức scale).
- Tính DoG bằng cách lấy hiệu giữa hai ảnh Gaussian liên tiếp.
- Mục đích: tìm điểm đặc trưng bất biến theo tỉ lệ.

02

2. Tìm điểm cực trị cục bộ trong không gian DoG

- Mỗi điểm ảnh được so sánh với 26 điểm lân cận (8 cùng mức, 9 ở mức trước, 9 ở mức sau).
- Nếu là lớn nhất hoặc nhỏ nhất  $\rightarrow$  ứng viên keypoint.

03

3. Lọc và tinh chỉnh keypoint

- Nội suy vị trí để xác định tọa độ chính xác hơn.
- Loại bỏ:
  - Điểm có độ tương phản thấp (ít thông tin).
  - Điểm nằm trên cạnh mạnh nhưng không phải góc (dựa trên ma trận Hessian).

04

4. Gán hướng & tạo descriptor

- Với mỗi keypoint:
  - Tính gradient magnitude và orientation trong vùng lân cận.
  - Xây dựng histogram hướng (36 bin cho hướng chính, 8 bin cho descriptor).
  - Chọn hướng mạnh nhất làm hướng chính  $\rightarrow$  đảm bảo bất biến quay.
- Chia vùng  $16 \times 16$  thành  $4 \times 4$  ô, mỗi ô tạo histogram 8 hướng  $\rightarrow$  descriptor 128 chiều.
- Chuẩn hóa descriptor để giảm ảnh hưởng ánh sáng.

# KHÔNG GIAN TỶ LỆ

- Mục tiêu: Tìm đặc trưng ổn định trước thay đổi tỷ lệ.
- Ý tưởng chính:
  - Vật thể xuất hiện khác nhau ở các tỷ lệ → cần xử lý đa tỷ lệ.
  - Không gian tỷ lệ = tập ảnh được làm mịn với tham số tỷ lệ.
- Vai trò: Mô phỏng sự mất chi tiết khi ảnh thu nhỏ.
- Tham số tỷ lệ: Kiểm soát mức độ làm mịn.
- Kết quả: Đặc trưng được phát hiện ở nhiều thang đo → bất biến với scale.

# KHÔNG GIAN TỶ LỆ

- Trong SIFT, nhân Gaussian được sử dụng để thực hiện làm mịn, do đó tham số tỷ lệ là độ lệch chuẩn.
- Lý do sử dụng nhân Gaussian là bởi nhiều nghiên cứu đã chỉ ra rằng nhân thông thấp Gaussian là nhân làm mịn duy nhất đáp ứng được một tập hợp các ràng buộc quan trọng, chẳng hạn như tính tuyến tính và tính bất biến dịch chuyển.
- Dựa trên điều này, không gian tỷ lệ  $L(x, y, \sigma)$  của ảnh thang độ xám,  $f(x, y)$ , được tạo ra bằng cách kết hợp  $f$  với bộ lọc Gaussian  $G(x, y, \sigma)$  có tỷ lệ thay đổi:

$$L(x, y, \sigma) = G(x, y, \sigma) \star f(x, y)$$

- Thang đo được điều khiển bởi tham số  $\sigma$  và  $G$  có dạng:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

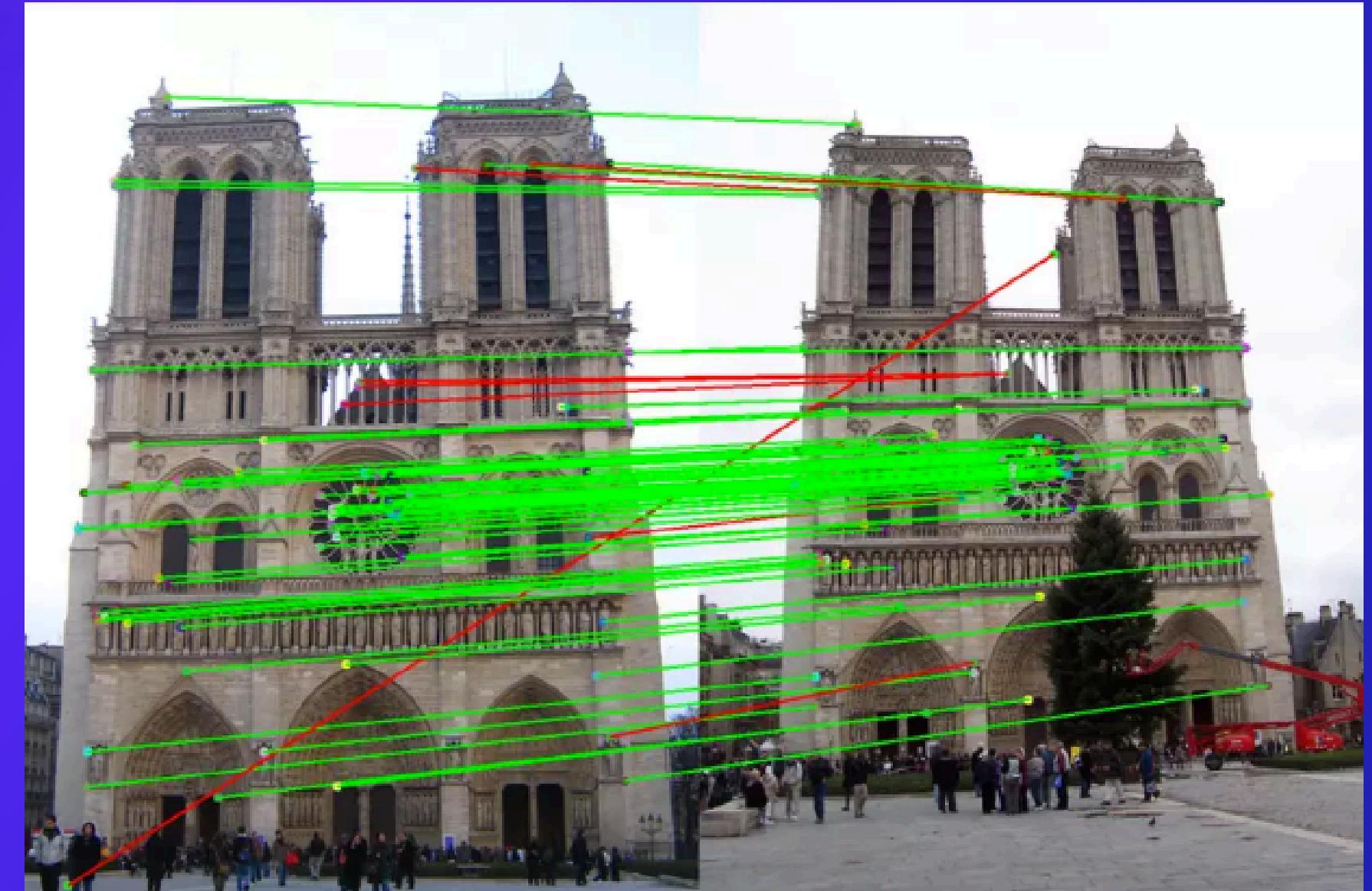
# GHÉP CẶP ĐẶC TRƯNG

Mục tiêu

- Tìm các cặp điểm đặc trưng tương ứng giữa hai ảnh dựa trên descriptor đã tính ở bước trước.

Phương pháp ghép cặp

- BFMatcher (Brute Force Matcher)



# ƯỚC LƯỢNG PHÉP BIẾN ĐỔI HÌNH HỌC

## 1. Phép biến đổi:

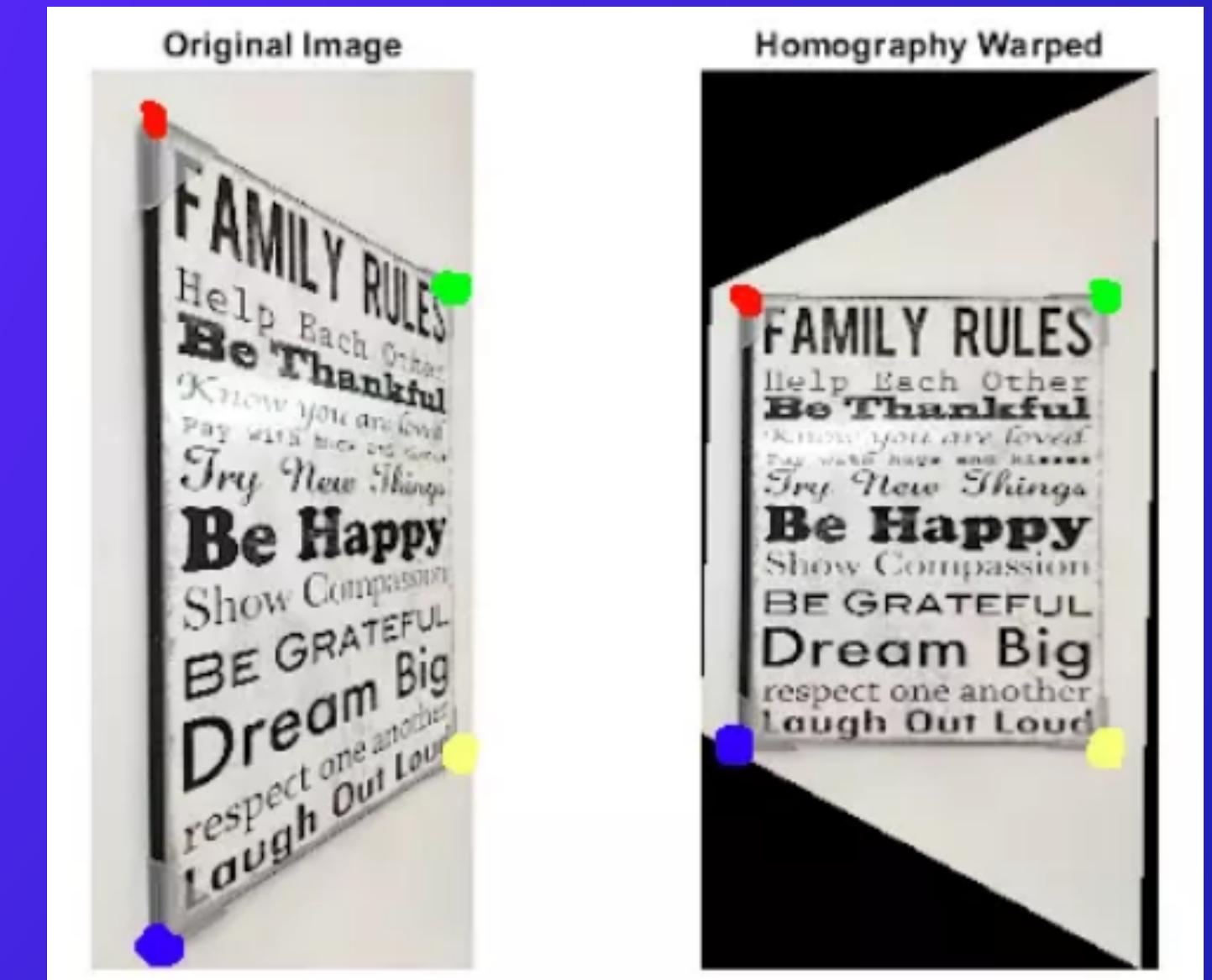
- Khi ghép ảnh panorama, các ảnh được chụp từ cùng một vị trí nhưng khác góc nhìn.
- Để ghép chính xác, ta phải biến đổi một ảnh sang hệ tọa độ của ảnh tham chiếu.
- Phép biến đổi này được biểu diễn bằng ma trận Homography  $3 \times 3$ .

## 2. Công cụ:

- `cv2.findHomography()` + RANSAC để loại bỏ outliers.

# WARP ẢNH

- Trong các phép biến đổi ảnh, Perspective transform là 1 phép biến đổi “vi diệu”, nó không bảo toàn góc, độ dài, tính song song... mà chỉ bảo toàn đường thẳng
- Tấm ảnh bên trái là ảnh chụp 1 tấm bảng được chụp lệch trái, ảnh bên phải là ảnh biến đổi với perspective transform bằng 1 ma trận H. Ngược lại, nếu ta có sẵn ảnh gốc và ảnh sau biến đổi, ta có thể tính lại được ma trận H.



# HOMOGRAPHY

- Khái niệm: Homography là phép biến đổi hình học ánh xạ một mặt phẳng này sang mặt phẳng khác.
- Vai trò: Dùng để căn chỉnh ảnh phải về cùng hệ tọa độ với ảnh trái.
- Công thức:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# GHÉP & BLENDING

## Mục tiêu

- Sau khi warp ảnh về cùng hệ tọa độ, ta cần ghép các ảnh lại thành một ảnh duy nhất.
- Xử lý vùng chồng lấn để tránh đường nối bị lộ và chênh lệch ánh sáng.

## Vấn đề thường gặp

- Đường nối rõ rệt do:
  - Khác biệt độ sáng, màu sắc.
  - Sai lệch nhỏ trong căn chỉnh.
- Vùng chồng lấn bị bóng mờ nếu có chuyển động trong cảnh.

## Giải pháp Blending

- Feathering: Làm mờ vùng chồng lấn bằng cách dùng trọng số giảm dần.



# THIẾT KẾ & CÀI ĐẶT



```
# Cell 1 - Import & hàm hiển thị
import cv2 # dùng OpenCV cho xử lý ảnh (đọc ảnh, SIFT, warp, v.v.)
import numpy as np # xử lý ma trận, tính toán số
import matplotlib.pyplot as plt # dùng để hiển thị ảnh trong Jupyter
|
# Đặt kích thước mặc định cho figure của matplotlib
plt.rcParams['figure.figsize'] = (8, 5)

def show_img(img_bgr, title=""):
    """
    Hiển thị ảnh BGR bằng matplotlib.
    - OpenCV đọc ảnh theo thứ tự kênh BGR
    - matplotlib hiển thị theo thứ tự kênh RGB
    => Cần đổi BGR -> RGB trước khi vẽ
    """

    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB) # Đổi không gian màu BGR -> RGB
    plt.figure()                                     # Hiển thị ảnh màu
    plt.imshow(img_rgb)                             # Tiêu đề
    plt.title(title)                                # Tắt trục toạ độ cho đẹp
    plt.axis("off")
```

# CELL 1 – NHẬP THƯ VIỆN VÀ ĐỊNH NGHĨA HÀM HIỂN THỊ ẢNH

```
# Cell 2 - Đường dẫn & đọc 3 ảnh

# Đường dẫn thư mục chứa 3 ảnh (bộ ảnh thí nghiệm)
image_dir = r"C:\Users\Public\panorama_sift_three_pictures"

# Đọc 3 ảnh: trái, giữa, phải (định dạng BGR)
left_img    = cv2.imread(fr"{image_dir}\img_left.jpg")
middle_img  = cv2.imread(fr"{image_dir}\img_middle.jpg")
right_img   = cv2.imread(fr"{image_dir}\img_right.jpg")

# In kích thước (h, w, c) của từng ảnh để kiểm tra
print("Kích thước ảnh trái:", left_img.shape)
print("Kích thước ảnh giữa:", middle_img.shape)
print("Kích thước ảnh phải:", right_img.shape)

# Hiển thị 3 ảnh gốc
show_img(left_img, "Ảnh trái")
show_img(middle_img, "Ảnh giữa")
show_img(right_img, "Ảnh phải")
```

Kích thước ảnh trái : (1360, 2048, 3)

Kích thước ảnh giữa: (1360, 2048, 3)

Kích thước ảnh phải: (1360, 2048, 3)

## CELL 2 – ĐỌC VÀ HIỂN THỊ 3 ẢNH THÀNH PHẦN

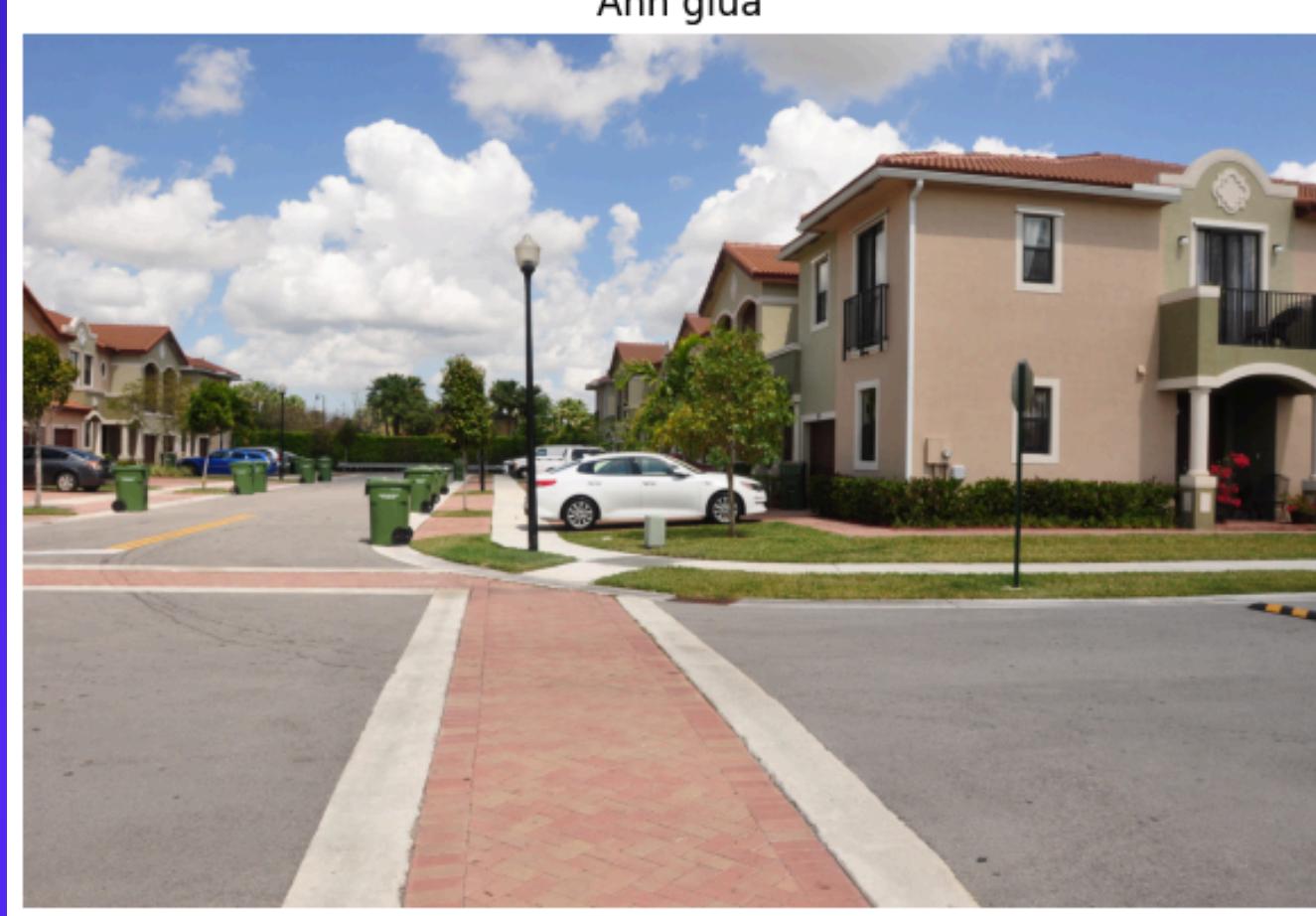
Ảnh trái



Ảnh phải



Ảnh giữa



# CELL 3 – HÀM TÍNH SIFT VÀ KHỚP (MATCHING) GIỮA 2 ẢNH

```
# Cell 3 - SIFT + matching giữa 2 ảnh

def sift_feature(image_bgr):
    """
    Tính keypoint & descriptor SIFT cho 1 ảnh BGR.
    - Bước 1: chuyển sang ảnh xám (SIFT làm việc trên grayscale)
    - Bước 2: tạo đối tượng SIFT
    - Bước 3: detect (tìm keypoint) và compute (tính descriptor)
    """
    gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY) # BGR -> Gray
    sift = cv2.SIFT_create() # Khởi tạo SIFT
    kp, des = sift.detectAndCompute(gray, None) # kp: List keypoints, des: ma trận descriptor
    # Keypoints Là các điểm đặc trưng như góc và đỉnh
    # descriptors Là vectơ mô tả cục bộ quanh mỗi keypoint
    return kp, des

#Hàm match sift giữa 2 ảnh
def match_sift(kp1, des1, kp2, des2,
               ratio_thresh=0.75,
               debug_title=None,
               img1=None, img2=None):
    """
    KNN matching + Lowe ratio test.
    Trả về:
    - good_matches: danh sách match tốt (lọc theo tỉ lệ khoảng cách)
    - H: homography (ảnh 2 -> ảnh 1) nếu tính được

    kp1, des1: keypoint & descriptor của ảnh 1 (reference)
    kp2, des2: keypoint & descriptor của ảnh 2 (ảnh cần warp)
    """
    # Brute Force Matcher dùng khoảng cách L2 - Kcách Euclid (chuẩn cho SIFT), không bật crosscheck
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)

    # KNN matching: với mỗi descriptor của ảnh 1 -> Lấy 2 match gần nhất ở ảnh 2
    knn_matches = bf.knnMatch(des1, des2, k=2)

    good = []
    for m, n in knn_matches:
        # Áp dụng Lowe's ratio test:
        # Nếu khoảng cách m nhỏ hơn ratio_thresh * n -> m Là match đáng tin
        if m.distance < ratio_thresh * n.distance:
            good.append(m)

    print(f"[{debug_title}] Số match tốt: {len(good)}")
```

```

H = None
"""

Homography là "cầu nối toán học" dùng các điểm tương ứng đó để căn chỉnh 2 (hoặc 3) ảnh lên cùng một hệ tọa độ → mới ghép được panorama.
Hãy tưởng tượng em chụp cùng một mặt phẳng (bức tường, mặt đường, mặt phẳng cảnh xa...) từ hai góc khác nhau.
Ảnh nhìn khác nhau (bị nghiêng, xa-gần, méo phôi cảnh), nhưng thật ra các điểm trên mặt phẳng đó vẫn có một phép biến
đổi hình học "chuẩn" ảnh xạ từ ảnh này sang ảnh kia.

Phép biến đổi đó gọi là homography (phép biến đổi phối cảnh mặt phẳng-mặt phẳng).
"""

# Để tính homography cần ít nhất 4 cặp điểm
if len(good) >= 4:
    # Lấy tọa độ điểm ở ảnh 1 (reference)
    pts1 = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
    # Lấy tọa độ điểm tương ứng ở ảnh 2 (moving)
    pts2 = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)

    # Tính homography: pts2 -> pts1, dùng RANSAC để loại bỏ outlier
    H, mask = cv2.findHomography(pts2, pts1, cv2.RANSAC)
    print(f"[{debug_title}] Homography:\n", H)
else:
    print(f"[{debug_title}] Không đủ match để tính homography!")

# Hàm vẽ các match tốt
if debug_title is not None and img1 is not None and img2 is not None and len(good) > 0:
    draw_n = min(100, len(good))  # Giới hạn số match vẽ (tránh quá nhiều)
    matches_draw = good[:draw_n]
    match_img = cv2.drawMatches( # ghép 2 ảnh cạnh nhau, nối cặp điểm match bằng đường thẳng
        img1, kp1,  # Ảnh 1 + keypoints
        img2, kp2,  # Ảnh 2 + keypoints
        matches_draw,
        None,
        flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
    )
    show_img(match_img, f"Match SIFT - {debug_title} (top {draw_n})")

return good, H

# Mục đích Cell 3: chuẩn bị hàm phát hiện đặc trưng và so khớp giữa hai ảnh, đồng thời ước lượng homography.

```

# CELL 3 – HÀM ƯỚC LƯỢNG HOMOGRAPHY VÀ VẼ MATCH

# CELL 4 – TÍNH SIFT CHO 3 ẢNH VÀ TÌM HOMOGRAPHY

```
# Cell 4 - Tính SIFT cho 3 ảnh & homography

# Tính SIFT cho từng ảnh (trái, giữa, phải)
kp_left, des_left = sift_feature(left_img)
kp_mid, des_mid = sift_feature(middle_img)
kp_right, des_right = sift_feature(right_img)

# In ra số Lượng keypoint của từng ảnh để xem ảnh nào “nhiều đặc trưng” hơn.
print("SIFT keypoints:")
print(" Trái : ", len(kp_left))
print(" Giữa: ", len(kp_mid))
print(" Phải:", len(kp_right))

# Match: TRÁI <-> GIỮA
# - Ảnh 1: middle_img (reference)
# - Ảnh 2: Left_img (moving)
# => Tính homography: left -> middle (H_left_to_mid)
good_lm, H_left_to_mid = match_sift(
    kp_mid, des_mid,           # kp1, des1 (ảnh giữa)
    kp_left, des_left,         # kp2, des2 (ảnh trái)
    ratio_thresh=0.75,
    debug_title="LEFT <-> MIDDLE",
    img1=middle_img, img2=left_img
)

# Match: PHẢI <-> GIỮA
# - Ảnh 1: middle_img (reference)
# - Ảnh 2: right_img (moving)
# => Tính homography: right -> middle (H_right_to_mid)
good_rm, H_right_to_mid = match_sift(
    kp_mid, des_mid,
    kp_right, des_right,
    ratio_thresh=0.75,
    debug_title="RIGHT <-> MIDDLE",
    img1=middle_img, img2=right_img
)

# => Mục đích Cell 4: tính đặc trưng cho 3 ảnh và tìm 2 homography:
```

**SIFT keypoints:**  
Trái : 7722  
Giữa: 7188  
Phải: 6970



Ảnh trái có 7722 điểm đặc trưng

Ảnh GIỮA có 7188 điểm đặc trưng

Ảnh PHẢI có 6970 điểm đặc trưng

```
[LEFT <-> MIDDLE] Số match tốt: 3098
```

```
[LEFT <-> MIDDLE] Homography:
```

```
[[ 1.22889589e+00 -1.70045223e-03 -4.47891693e+02]
 [ 6.99157979e-02 1.12559985e+00 -7.83179049e+01]
 [ 1.13870897e-04 -5.78437748e-06 1.00000000e+00]]
```

Match SIFT - LEFT <-> MIDDLE (top 1000)



```
[RIGHT <-> MIDDLE] Số match tốt: 2831
```

```
[RIGHT <-> MIDDLE] Homography:
```

```
[[ 8.09960038e-01 3.16010194e-03 3.71060946e+02]
 [-5.93898167e-02 9.18919060e-01 4.89257310e+01]
 [-9.28165290e-05 2.07406902e-07 1.00000000e+00]]
```

Match SIFT - RIGHT <-> MIDDLE (top 1000)



```
# Cell 5 - Hàm tính canvas chung + feather blending + auto crop
```

☰ ⌂ ⌃ ⌄ ⌅ ⌆

```
def compute_global_canvas(left_img, middle_img, right_img,
                           H_left_to_mid, H_right_to_mid):
```

"""

Tính kích thước canvas chung & ma trận offset\_matrix cho cả 3 ảnh.

Ý tưởng:

- Lấy 4 góc mỗi ảnh  $(0,0)$ ,  $(w,0)$ ,  $(0,h)$ ,  $(w,h)$
- Warp góc ảnh trái & phải sang hệ tọa độ ảnh giữa (dùng homography)
- Gộp tất cả điểm  $\rightarrow$  tìm min/max x,y  $\rightarrow$  ra khung bao ngoài cùng (bounding box)
- Từ đó suy ra chiều rộng/chiều cao canvas và ma trận dịch (offset)

"""

```
hL, wL = left_img.shape[:2]
hM, wM = middle_img.shape[:2]
hR, wR = right_img.shape[:2]
```

# Lấy 4 góc của 3 ảnh (trong hệ tọa độ riêng của từng ảnh)

```
corners_left = np.float32([[0,0], [wL,0], [0,hL], [wL,hL]]).reshape(-1,1,2)
corners_mid = np.float32([[0,0], [wM,0], [0,hM], [wM,hM]]).reshape(-1,1,2)
corners_right = np.float32([[0,0], [wR,0], [0,hR], [wR,hR]]).reshape(-1,1,2)
```

# Lấy 4 góc của mỗi ảnh trong hệ tọa độ riêng  $(0,0)$ ,  $(w,0)$ ,  $(0,h)$ ,  $(w,h)$

# Warp góc ảnh trái & phải sang hệ tọa độ ảnh giữa

```
corners_left_warp = cv2.perspectiveTransform(corners_left, H_left_to_mid) # vị trí bốn góc ảnh trái sau khi đưa về hệ tọa độ ảnh giữa.
corners_right_warp = cv2.perspectiveTransform(corners_right, H_right_to_mid) # tương tự cho ảnh phải.
corners_mid_warp = corners_mid # Ảnh giữa là gốc nên không cần warp
```

## CELL 5 – HÀM TÍNH CANVAS CHUNG

```
# vị trí bốn góc ảnh trái sau khi đưa về hệ tọa độ ảnh giữa
corners_left_warp = cv2.perspectiveTransform(corners_left, H_left_to_mid)
# vị trí bốn góc ảnh phải sau khi đưa về hệ tọa độ ảnh giữa
corners_right_warp = cv2.perspectiveTransform(corners_right, H_right_to_mid)
corners_mid_warp = corners_mid # Ảnh giữa Là gốc nên không cần warp

# Gộp toàn bộ góc lại để tính bounding box - khung của ảnh panorama
all_corners = np.concatenate(
    (corners_left_warp, corners_mid_warp, corners_right_warp),
    axis=0
)
# Tìm min/max theo trục x,y
[x_min, y_min] = all_corners.min(axis=0).ravel()
[x_max, y_max] = all_corners.max(axis=0).ravel()

# Làm tròn xuống & Làm tròn lên để an toàn
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))

# Kích thước canvas (chiều rộng, chiều cao)
canvas_w = x_max - x_min
canvas_h = y_max - y_min

# Ma trận offset:
# Dịch toàn bộ điểm sao cho (x_min, y_min) -> (0,0)
offset_matrix = np.array([
    [1, 0, -x_min], # Dịch theo X
    [0, 1, -y_min], # Dịch theo Y
    [0, 0, 1]
], dtype=float)

print("Canvas size:", canvas_w, "x", canvas_h)
return canvas_w, canvas_h, offset_matrix
```

```
def feather_blend_two(imgA, imgB):
    """
        Feather blending 2 ảnh cùng kích thước.
        - Pixel = 0 (đen cả 3 kênh) được coi là nền (no data).
        - Vùng chồng lấp (overlap) sẽ được blend theo trọng số tuyến tính theo toạ độ x.

    Trả về: ảnh đã blend.
    """
    assert imgA.shape == imgB.shape
    h, w = imgA.shape[:2]

    # mask: pixel nào có dữ liệu (ít nhất 1 kênh != 0)
    maskA = np.any(imgA != 0, axis=2)
    maskB = np.any(imgB != 0, axis=2)

    onlyA = maskA & ~maskB # chỉ A
    onlyB = maskB & ~maskA # chỉ B
    overlap = maskA & maskB # chồng Lấp

    result = np.zeros_like(imgA)

    # Vùng chỉ có A hoặc chỉ có B -> copy trực tiếp
    result[onlyA] = imgA[onlyA]
    result[onlyB] = imgB[onlyB]
```

## CELL 5 – HÀM GHÉP ẢNH

```

if np.any(overlap):
    # X: ma trận toạ độ cột (0..w-1) Lắp theo hàng (h)
    X = np.tile(np.arange(w), (h, 1))

    # x0: cột nhỏ nhất trong vùng overlap; x1: cột lớn nhất
    x0 = X[overlap].min()
    x1 = X[overlap].max()
    if x1 == x0:
        # Trường hợp overlap rất nhỏ -> dùng trọng số 0.5 đều cho 2 ảnh
        wA = np.full((h, w), 0.5, dtype=np.float32)
    else:
        # Trọng số A giảm dần từ trái sang phải trong vùng chồng Lắp
        wA = (x1 - X).astype(np.float32) / (x1 - x0)
        wB = 1.0 - wA # Trọng số B bù lại

    # Chỉ áp dụng trọng số cho vùng overlap
    for c in range(3): # 3 kênh màu BGR
        chanA = imgA[:, :, c].astype(np.float32)
        chanB = imgB[:, :, c].astype(np.float32)
        blend = chanA * wA + chanB * wB      # Blend theo trọng số
        tmp = result[:, :, c].astype(np.float32)
        tmp[overlap] = blend[overlap]          # Gán vào vùng overlap
    result[:, :, c] = np.clip(tmp, 0, 255).astype(np.uint8)

return result # => Kết quả: vùng giao giữa 2 ảnh được "nhòa" dần, tránh đường nối gắt

```

```
def auto_crop_non_black(pano_bgr):
    """
        Tự động crop bỏ viền đen quanh ảnh panorama.
        Ý tưởng:
        - Đổi sang gray
        - Pixel > 0 coi là có dữ liệu
        - Tìm contour ngoài cùng (lớn nhất)
        - Cắt ảnh theo bounding box của contour đó
    """

    gray = cv2.cvtColor(pano_bgr, cv2.COLOR_BGR2GRAY)
    # Pixel > 0 -> 255 (trắng); pixel = 0 -> 0 (đen)
    _, thresh = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)

    # Tìm các contour ngoài (các vùng có dữ liệu)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
                                    cv2.CHAIN_APPROX_SIMPLE)

    if not contours:
        # Không tìm thấy contour -> trả ảnh gốc
        return pano_bgr

    # Chọn contour có diện tích Lớn nhất (vùng ảnh chính)
    largest = max(contours, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(largest)
    cropped = pano_bgr[y:y+h, x:x+w]
    return cropped
```

## CELL 5 – HÀM CẮT VIỀN ĐEN

```
# Cell 6 - Warp 3 ảnh vào canvas chung
# Nếu không tính được homography cho 1 trong 2 cặp -> dừng
if H_left_to_mid is None or H_right_to_mid is None:
    raise RuntimeError("Không tính được homography cho một trong hai cặp ảnh!")

# Tính kích thước canvas chung và ma trận offset (dịch)
canvas_w, canvas_h, offset_matrix = compute_global_canvas(
    left_img, middle_img, right_img,
    H_left_to_mid, H_right_to_mid
)

# Warp từng ảnh sang canvas chung.
# Lưu ý: transform tổng = offset_matrix * H (nhân ma trận 3x3)
left_warp = cv2.warpPerspective(
    left_img, offset_matrix.dot(H_left_to_mid),
    (canvas_w, canvas_h)
)
mid_warp = cv2.warpPerspective(
    middle_img, offset_matrix,
    (canvas_w, canvas_h)
)
right_warp = cv2.warpPerspective(
    right_img, offset_matrix.dot(H_right_to_mid),
    (canvas_w, canvas_h)
)

# Xem thử 3 ảnh sau khi warp lên canvas
show_img(left_warp, "Left warp")
show_img(mid_warp, "Middle warp")
show_img(right_warp, "Right warp")
```

Canvas size: 1282 x 603

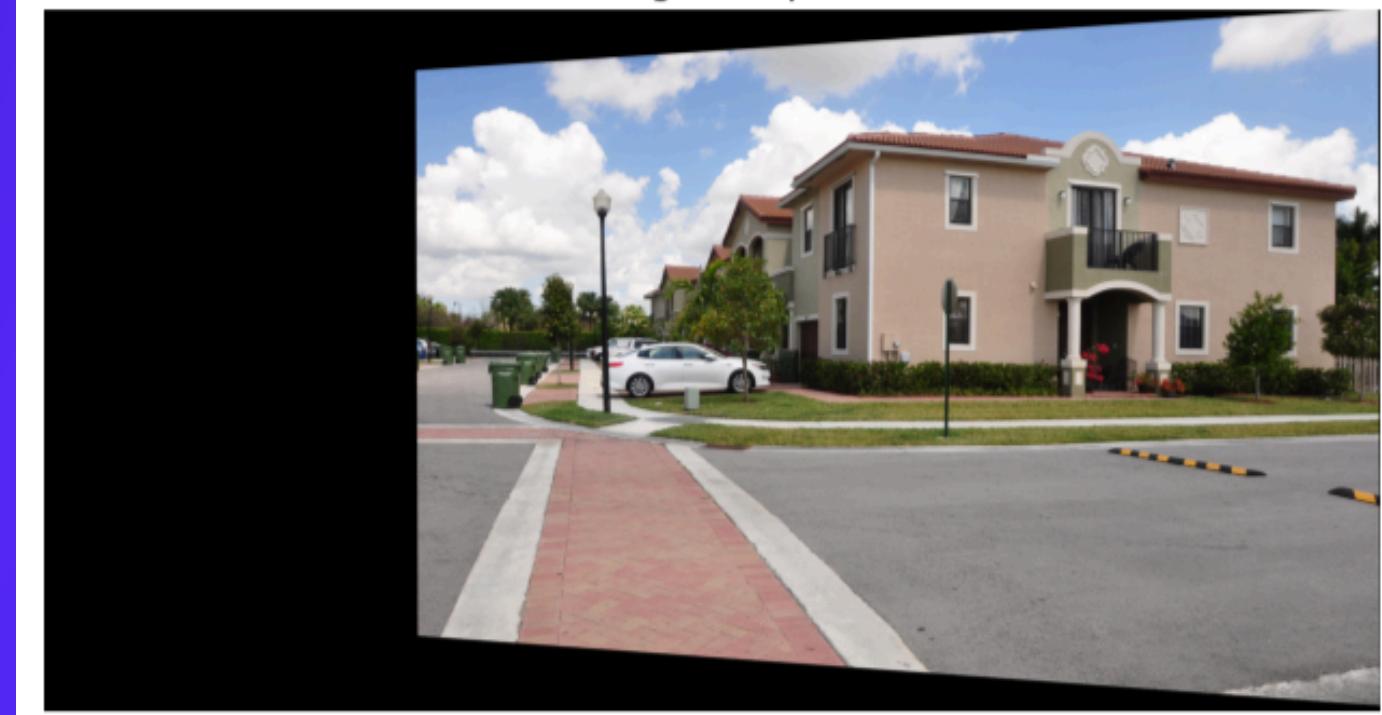
## CELL 6 - WARP CÁC ẢNH VÀO CANVAS CHUNG

Canvas size: 2965 x 1555

Left warp



Right warp



Middle warp



```
# Cell 7 – Panorama tạm thời TRÁI + GIỮA  
# Blend ảnh giữa (mid_warp) với ảnh trái (left_warp)  
pano_left_mid = feather_blend_two(mid_warp, left_warp)  
show_img(pano_left_mid, "Panorama tạm thời (trái + giữa)")
```

Panorama tạm thời (trái + giữa)



## CELL 7 – PHA TRỘN TẠM THỜI ẢNH TRÁI + GIỮA

```
# Cell 8 - Panorama tạm thời GIỮA + PHẢI  
# Blend ảnh giữa (mid_warp) với ảnh phải (right_warp)  
pano_mid_right = feather_blend_two(mid_warp, right_warp)  
show_img(pano_mid_right, "Panorama tạm thời (giữa + phải)")
```

Panorama tạm thời (giữa + phải)



## CELL 8 – PHA TRỘN TẠM THỜI ẢNH GIỮA + PHẢI

```
# Cell 9 - Panorama 3 ảnh

# Ghép 3 ảnh: (trái + giữa) rồi ghép thêm phải
pano_full = feather_blend_two(pano_left_mid, right_warp)

# Auto crop bỏ viền đen
pano_cropped = auto_crop_non_black(pano_full)
show_img(pano_cropped, "Panorama 3 ảnh")

# Lưu kết quả
save_crop = fr"{image_dir}\panorama_result_cropped_city.jpg"

cv2.imwrite(save_crop, pano_cropped)

print("Đã lưu panorama (crop):", save_crop)
```

## CELL 9 – GHÉP TOÀN BỘ 3 ẢNH VÀ LƯU KẾT QUẢ

## Panorama 3 ảnh



Đã lưu panorama (crop): C:\Users\Public\panorama\_sift\_three\_pictures\panorama\_result\_cropped\_city.jpg

# ĐÁNH GIÁ, HẠN CHẾ & HƯỚNG PHÁT TRIỂN

(01)

- **Ưu điểm:**

- SIFT ổn định, có tính bất biến cao với thay đổi về tỉ lệ và xoay, mô tả đặc trưng chi tiết giúp khớp ảnh chính xác dưới những điều kiện biến đổi hình học.
- SIFT thường tìm được nhiều match đúng đắn, phù hợp với nhiều ứng dụng ghép ảnh phức tạp.

(02)

- **Hạn chế:**

- SIFT có độ phức tạp tính toán cao, mô tả dài, dẫn đến tốc độ xử lý chậm và tốn bộ nhớ.
- Do đó SIFT không thích hợp cho các ứng dụng yêu cầu xử lý thời gian thực

(03)

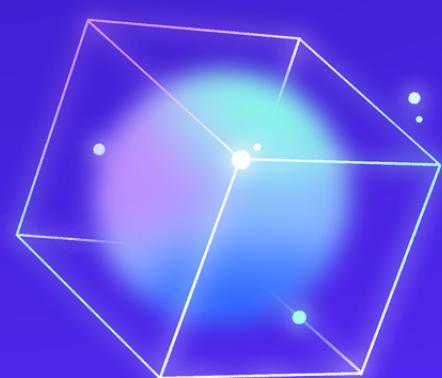
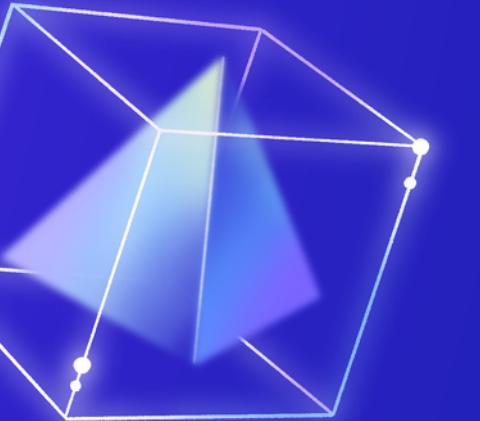
- **Hướng phát triển:**

- Áp dụng các kỹ thuật warp hình cầu/cylinder để ghép các cảnh góc rộng hơn
- Hỗ trợ ghép nhiều ảnh hơn hoặc panorama 360°.
- Thủ kết hợp học sâu để làm thêm việc tái tạo ảnh, nhận dạng cảnh.

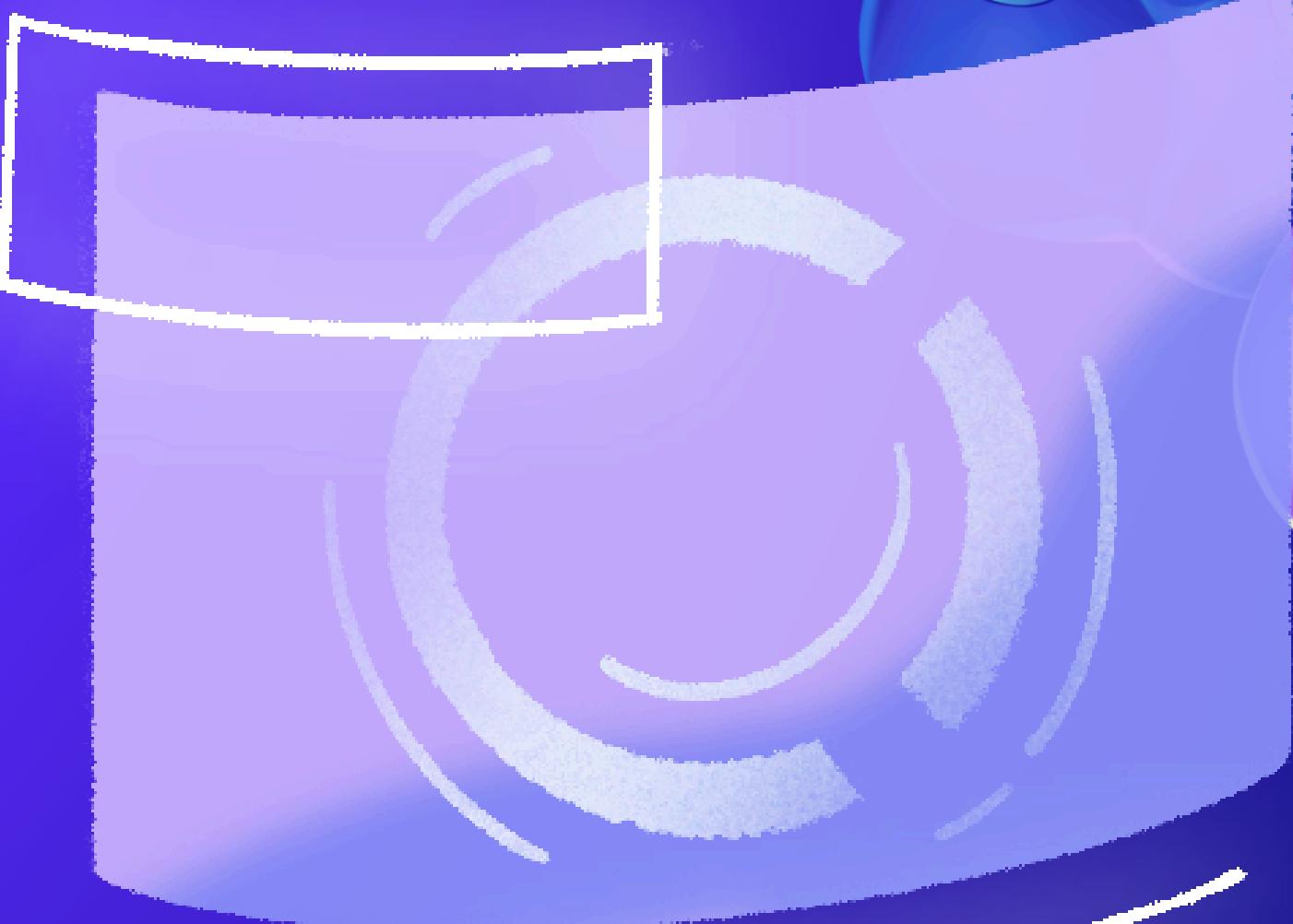


# TÀI LIỆU THAM KHẢO

1. TÀI LIỆU VÀ SLIDE MÔN XỬ LÝ ẢNH – CHƯƠNG 6: PHÉP BIẾN ĐỔI ĐẶC TRƯNG BẤT BIẾN TỶ LỆ (SIFT) VÀ ỨNG DỤNG
2. [HTTPS://PYIMAGESearch.COM/2018/12/17/IMAGE-STITCHING-WITH-OPENCV-AND-PYTHON/](https://pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/)
3. [https://docs.opencv.org/4.x/da/df5/tutorial\\_py\\_sift\\_intro.html](https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html)
4. [https://trungthanhnguyen0502.github.io/image\\_processing/2020/05/01/Image-processing-Image-stiching-panorama/](https://trungthanhnguyen0502.github.io/image_processing/2020/05/01/Image-processing-Image-stiching-panorama/)



THANK YOU!



# RESOURCE PAGE

