

Bài tập lớn 3: Camera Calibration and Fundamental Matrix Estimation with RANSAC

Giới thiệu

Nhận dạng và so trùng đặc trưng giữa hai ảnh được thực hiện bằng các xác định các điểm góc (corner points), và khớp các đặc trưng xung quanh các điểm góc đó. Các giải thuật như SIFT (thực hiện trong bài tập lớn 2) phụ thuộc vào gradient của pixel xung quanh điểm góc, nên rất nhạy cảm với các hiệu chỉnh thông số bên ngoài như camera calibration. Vì vậy, nếu camera được xoay để chụp một vật thể ở một góc khác, SIFT sẽ cho kết quả rất kém.

Vì vậy, trong bài tập lớn này sẽ yêu cầu tính toán các thông số bên ngoài như vị trí máy ảnh, tỷ lệ khung hình, độ lệch giữa các trục dựa vào các mối tương quan hình học giữa các hình ảnh được chụp từ nhiều góc chụp khác nhau.

Bài tập lớn được chia làm 3 phần:

- Phần 1: Tính toán ma trận Camera Projection Matrix (M) dựa vào các cặp điểm tương ứng trong không gian ảnh 2D và không gian thế giới thực 3D, sau đó tính toán camera's center dựa vào ma trận M.
- Phần 2: Tính toán ma trận Fundamental Matrix dựa vào các cặp điểm tương ứng trên 2 ảnh.
- Phần 3: Hiện thực giải thuật RANSAC (Random sample consensus) để tính toán ma trận Fundamental Matrix

Camera Projection Matrix

Chi tiết hiện thực

Projection matrix được sử dụng để chuyển đổi các điểm từ không gian thế giới thực 3D về không gian ảnh 2D.

$$x = K[R \ t]X$$

trong đó:

- x : tọa độ trong không gian ảnh ($u, v, 1$)
- K : Intrinsic Matrix (3x3)
- R : Rotation (3x3)

- t : Translation (3x1)
- X : toạ độ trong không gian thế giới thực ($X, Y, Z, 1$)

$$\Leftrightarrow w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} wu \\ wv \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} wu \\ wv \\ w \\ 1 \end{bmatrix}$$

$$u = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$\rightarrow (m_{31}X + m_{32}Y + m_{33}Z + m_{34})u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$\rightarrow m_{11}X + m_{12}Y + m_{13}Z + m_{14} - m_{31}uX - m_{32}uY + m_{33}uZ - m_{34}u = 0$$

$$v = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$\rightarrow (m_{31}X + m_{32}Y + m_{33}Z + m_{34})v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$\rightarrow m_{21}X + m_{22}Y + m_{23}Z + m_{24} - m_{31}vX - m_{32}vY + m_{33}vZ - m_{34}v = 0$$

với n cặp điểm ta sẽ có $2n$ phương trình tương ứng. Để giải hệ phương trình tìm m ta có hai cách:

Cách 1: Đặt $m_{34} = 1$. Hệ phương trình sẽ tương đương với:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1 X_1 & -u_1 Y_1 & -u_1 Z_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1 X_1 & -v_1 Y_1 & -v_1 Z_1 \\ & & & & & & \vdots & & & & \\ X_n & Y_n & Z_n & n & 0 & 0 & 0 & 0 & -u_n X_n & -u_n Y_n & -u_n Z_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & n & -v_n X_n & -v_n Y_n & -v_n Z_n \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{bmatrix}$$

Chi tiết hiện thực trong python/numpy:

```
arr = np.column_stack((Points_3D, [1]*Points_3D.shape[0]))

A1 = np.concatenate((arr, np.zeros_like(arr)),
                     axis=1).reshape((-1, 4))
A2 = np.concatenate((np.zeros_like(arr), arr),
                     axis=1).reshape((-1, 4))
A3 = -np.multiply(Points_2D.reshape((-1, 1)).repeat(3, axis=1),
                  Points_3D.repeat(2, axis=0))

A = np.concatenate((A1, A2, A3), axis=1)
b = Points_2D.reshape((-1, 1))

M = np.append(np.linalg.lstsq(A, b)[0], [1]).reshape((3, 4))
```

Cách 2: Sử dụng SVD để giải phương trình $Ax = 0$:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1 X_1 & -u_1 Y_1 & -u_1 Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1 X_1 & -v_1 Y_1 & -v_1 Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & n & 0 & 0 & 0 & 0 & -u_n X_n & -u_n Y_n & -u_n Z_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & n & -v_n X_n & -v_n Y_n & -v_n Z_n & -v_n \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Chi tiết hiện thực trong python/numpy:

```

arr = np.column_stack((Points_3D, [1]*Points_3D.shape[0]))

A1 = np.concatenate((arr, np.zeros_like(arr)),
                     axis=1).reshape((-1, 4))
A2 = np.concatenate((np.zeros_like(arr), arr),
                     axis=1).reshape((-1, 4))
A3 = -np.multiply(np.tile(Points_2D.reshape((-1, 1)), 4),
                   arr.repeat(2, axis=0))

A = np.concatenate((A1, A2, A3), axis=1)
U, s, V = np.linalg.svd(A)
M = V[-1]
M = M.reshape((3, 4))

```

Tính camera's center từ ma trận Projection Matrix bằng công thức sau:

$$M = (Q|m_4) \Rightarrow C = -Q^{-1}m_4$$

Chi tiết hiện thực trong python/numpy:

```

M_ = np.split(M, [3], axis=1)
C = np.squeeze(-np.matmul(np.linalg.inv(M_[0]), M_[1]))

```

Kết quả

Normalized image coordinates (hard_points=False)

The projection matrix is:

$$\begin{bmatrix} 0.76785834 & -0.49384797 & -0.02339781 & 0.00674445 \\ -0.0852134 & -0.09146818 & -0.90652332 & -0.08775678 \\ 0.18265016 & 0.29882917 & -0.07419242 & 1. \end{bmatrix}$$

The total residual is:

0.044534993949316135

The estimated location of the camera is:

$$[-1.51263977 \quad -2.35165965 \quad 0.28266502]$$

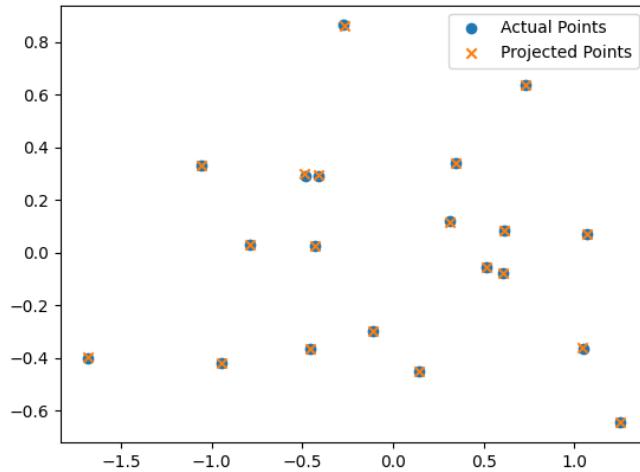


Figure 1: Actual vs projected points

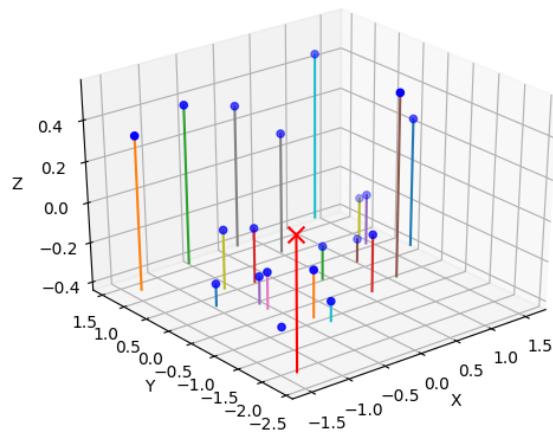


Figure 2: Mô hình về camera's center và các điểm quan trọng trong thế giới thực '+' shows the camera centre, 'o' are the interest points

Non-Normalized image coordinates (hard_points=True)

The projection matrix is:

$$\begin{bmatrix} -2.04662532e+00 & 1.18743052e+00 & 3.88938200e-01 & 2.43732985e+02 \\ -4.56886722e-01 & -3.02017128e-01 & 2.14721848e+00 & 1.65932475e+02 \\ -2.24678720e-03 & -1.09380146e-03 & 5.58547111e-04 & 1.00000000e+00 \end{bmatrix}$$

The total residual is:

15.621732302525796

The estimated location of the camera is:

$$[303.09666406 \quad 307.18423708 \quad 30.4222733]$$

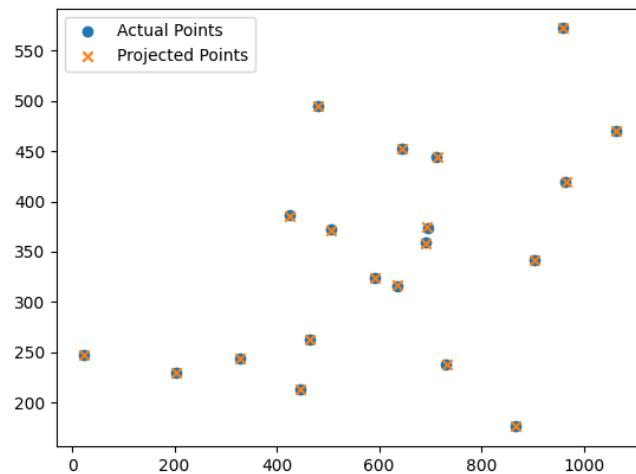


Figure 3: Actual vs projected points

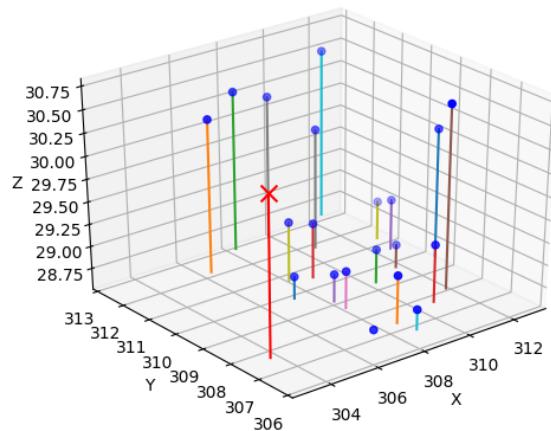


Figure 4: Mô hình về camera's center và các điểm quan trọng trong thế giới thực '+' shows the camera centre, 'o' are the interest points

Estimation of Fundamental Matrix

Chi tiết hiện thực

Giải thuật 8-point được sử dụng để tính toán ma trận fundamental matrix (F) khi có các cặp điểm tương ứng với 2 ảnh. Với mỗi cặp điểm $x = (u, v)$ trong ảnh bên trái và $x' = (u', v')$ trong ảnh bên phải, ta có:

$$x'^T F x = 0$$

với n cặp điểm ta có:

$$\begin{bmatrix} u'_1 u_1 & u'_1 v_1 & u'_1 & v'_1 u_1 & v'_1 v_1 & v'_1 & u_1 & v_1 & 1 \\ \vdots & \vdots \\ u'_n u_n & u'_n v_n & u'_n & v'_n u_n & v'_n v_n & v'_n & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

giải ma trận F bằng cách sử dụng SVD:

Chi tiết hiện thực trong python/numpy:

```
arr_a = np.column_stack((Points_a, [1]*Points_a.shape[0]))
arr_b = np.column_stack((Points_b, [1]*Points_b.shape[0]))

arr_a = np.tile(arr_a, 3)
arr_b = arr_b.repeat(3, axis=1)
A = np.multiply(arr_a, arr_b)

'''Solve f from Af=0'''
U, s, V = np.linalg.svd(A)
F_matrix = V[-1]
F_matrix = np.reshape(F_matrix, (3, 3))

'''Resolve det(F) = 0 constraint using SVD'''
U, S, Vh = np.linalg.svd(F_matrix)
S[-1] = 0
F_matrix = U @ np.diagflat(S) @ Vh
```

Normalized image coordinates

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\bar{u} \\ 0 & 1 & -\bar{v} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

trong đó:

- $(u', v', 1)$ là toạ độ đã được chuẩn hoá.
- $(u, v, 1)$ là toạ độ chưa được chuẩn hoá.
- s là hệ số sacle.
- \bar{u}, \bar{v} là giá trị trung bình trên tất cả các điểm.

Các bước thực hiện:

- Tính toán trọng tâm của tất cả các điểm tương ứng với hai ảnh.

$$\bar{u} = \frac{1}{n} \sum_{i=1}^n u_i$$

$$\bar{v} = \frac{1}{n} \sum_{i=1}^n v_i$$

- Trừ toạ độ các điểm với mean.

$$\tilde{u} = u - \bar{u}$$

$$\tilde{v} = v - \bar{v}$$

- Tính toán hệ số sacle s dựa theo công thức:

$$s = \frac{\sqrt{2}}{\sqrt{\left(\frac{1}{n} \sum_{i=1}^n (\tilde{u}_i^2 + \tilde{v}_i^2)\right)}}$$

$$s' = \frac{\sqrt{2}}{\sqrt{\left(\frac{1}{n} \sum_{i=1}^n (\tilde{u}'_i^2 + \tilde{v}'_i^2)\right)}}$$

- Xây dựng ma trận T_a, T_b :

$$T_a = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\bar{u} \\ 0 & 1 & -\bar{v} \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_b = \begin{bmatrix} s' & 0 & 0 \\ 0 & s' & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\bar{u}' \\ 0 & 1 & -\bar{v}' \\ 0 & 0 & 1 \end{bmatrix}$$

- Tính toạ độ sau khi chuẩn hoá:

$$\hat{x}_i = T_a x$$

$$\hat{x}'_i = T_b x'$$

- Tính ma trận Fundamental Matrix dựa vào toạ độ sau khi chuẩn hoá tương tự như giải thuật 8-point ở trên.
- Sau khi tính được ma trận F_{norm} , ma trận F_{orig} được tính dựa theo công thức sau:

$$F_{orig} = T_b^T * F_{norm} * T_a$$

Chi tiết hiện thực trong python/numpy:

```

mean_a = Points_a.mean(axis=0)
mean_b = Points_b.mean(axis=0)
std_a = np.sqrt(np.mean(np.sum((Points_a-mean_a)**2, axis=1), axis=0))
std_b = np.sqrt(np.mean(np.sum((Points_b-mean_b)**2, axis=1), axis=0))

Ta1 = np.diagflat(np.array([np.sqrt(2)/std_a, np.sqrt(2)/std_a, 1]))
Ta2 = np.column_stack((np.row_stack((np.eye(2), [[0, 0]])), [-mean_a[0]
                                                               , -mean_a[1], 1]))

Tb1 = np.diagflat(np.array([np.sqrt(2)/std_b, np.sqrt(2)/std_b, 1]))
Tb2 = np.column_stack((np.row_stack((np.eye(2), [[0, 0]])), [-mean_b[0]
                                                               , -mean_b[1], 1]))

Ta = np.matmul(Ta1, Ta2)
Tb = np.matmul(Tb1, Tb2)

arr_a = np.column_stack((Points_a, [1]*Points_a.shape[0]))
arr_b = np.column_stack((Points_b, [1]*Points_b.shape[0]))

arr_a = np.matmul(Ta, arr_a.T)
arr_b = np.matmul(Tb, arr_b.T)

arr_a = arr_a.T
arr_b = arr_b.T

arr_a = np.tile(arr_a, 3)
arr_b = arr_b.repeat(3, axis=1)
A = np.multiply(arr_a, arr_b)

'''Solve f from Af=0'''
U, s, V = np.linalg.svd(A)
F_matrix = V[-1]
F_matrix = np.reshape(F_matrix, (3, 3))

'''Resolve det(F) = 0 constraint using SVD'''
U, S, Vh = np.linalg.svd(F_matrix)
S[-1] = 0
F_matrix = U @ np.diagflat(S) @ Vh

F_matrix = Tb.T @ F_matrix @ Ta

```

Kết quả

Normalized image coordinates

Fundamental Matrix:

$$\begin{bmatrix} -1.17248591e - 07 & 1.60824663e - 06 & -4.01980786e - 04 \\ 1.11212887e - 06 & -2.73443755e - 07 & 3.23319884e - 03 \\ -2.36400817e - 05 & -4.44404958e - 03 & 1.03455561e - 01 \end{bmatrix}$$



Figure 5: Left image, Normalized image coordinates



Figure 6: Right image, Normalized image coordinates

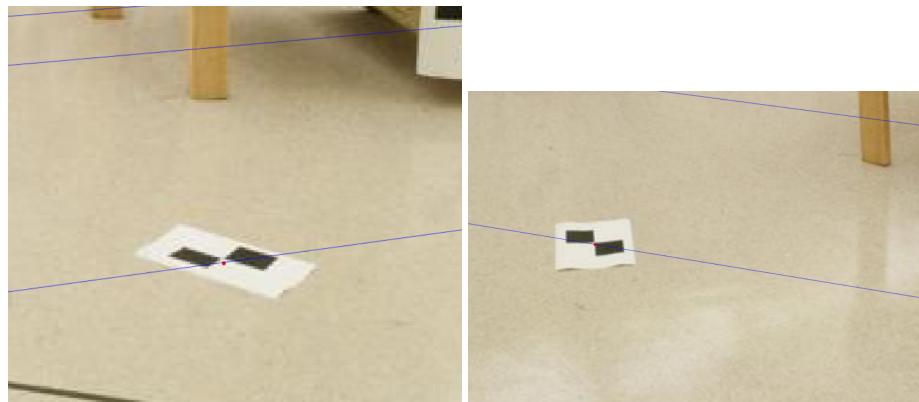


Figure 7: *Left:* Left zoom, *Right:* Right zoom

Non-Normalized image coordinates

Fundamental Matrix:

$$\begin{bmatrix} -5.36264198e-07 & 7.90364771e-06 & -1.88600204e-03 \\ 8.83539184e-06 & 1.21321685e-06 & 1.72332901e-02 \\ -9.07382264e-04 & -2.64234650e-02 & 9.99500092e-01 \end{bmatrix}$$



Figure 8: Left image, Non-normalized image coordinates

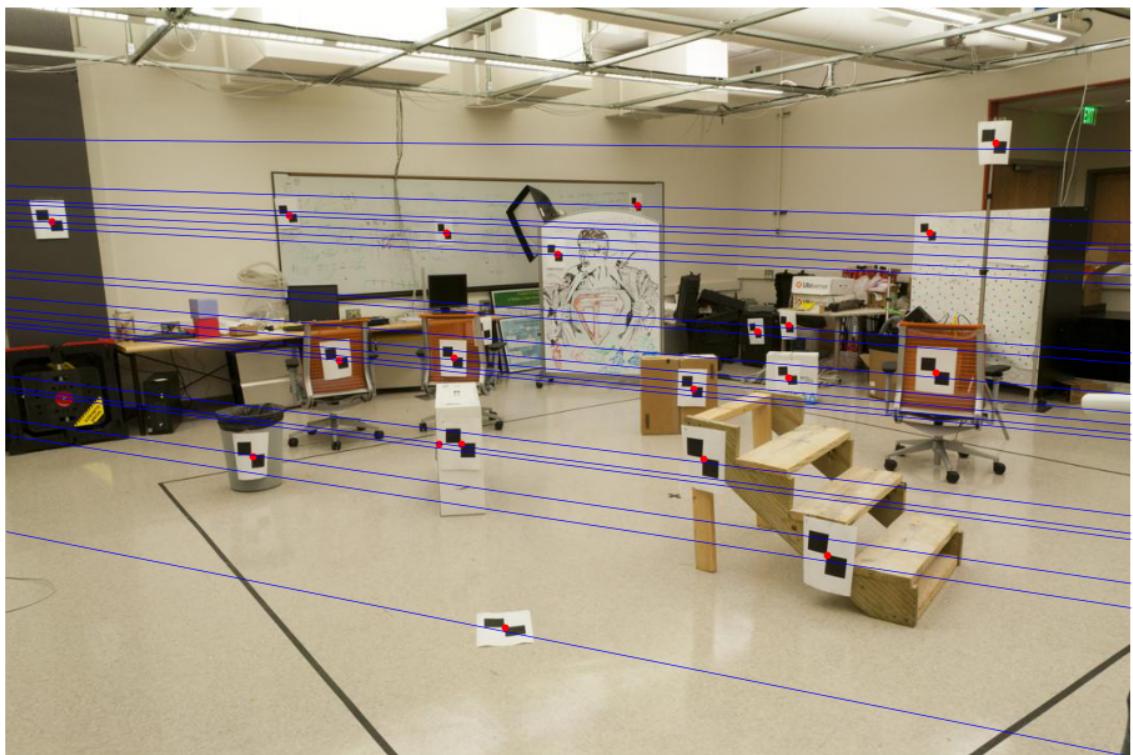


Figure 9: Right image, Non-normalized image coordinates

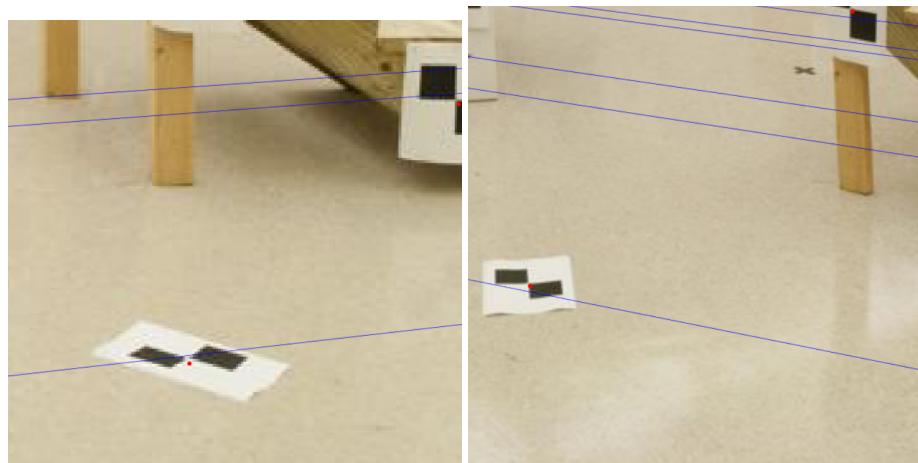


Figure 10: *Left:* Left zoom, *Right:* Right zoom

Fundamental Matrix with RANSAC

Chi tiết hiện thực

Các bước thực hiện giải thuật RANSAC:

- Chọn ngẫu nhiên 8 cặp điểm giữa 2 ảnh.
- Tính toán ma trận Fundamental dựa trên các cặp điểm đã chọn.
- Từ ma trận F đã tính được, tính tổng số inlier dựa theo công thức:

$$\text{if } (x'^T F x \leq \text{threshold}) \Rightarrow \text{inlier}$$

- Lặp lại giải thuật một số lần lặp nhất định.
- Chọn ma trận Fundamental cho số inlier lớn nhất.

Chi tiết hiện thực trong python/numpy:

```

num_iterator = 1500
threshold = 0.005
best_F_matrix = np.zeros((3, 3))
max_inlier = 0
num_sample_rand = 8

xa = np.column_stack((matches_a, [1]*matches_a.shape[0]))
xb = np.column_stack((matches_b, [1]*matches_b.shape[0]))
xa = np.tile(xa, 3)
xb = xb.repeat(3, axis=1)
A = np.multiply(xa, xb)

for i in range(num_iterator):
    index_rand = np.random.randint(matches_a.shape[0], size=
                                    num_sample_rand)
    F_matrix = estimate_fundamental_matrix_with_normalize(matches_a[
        index_rand, :],
        matches_b[index_rand, :])
    err = np.abs(np.matmul(A, F_matrix.reshape((-1))))
    current_inlier = np.sum(err <= threshold)
    if current_inlier > max_inlier:
        best_F_matrix = F_matrix
        max_inlier = current_inlier

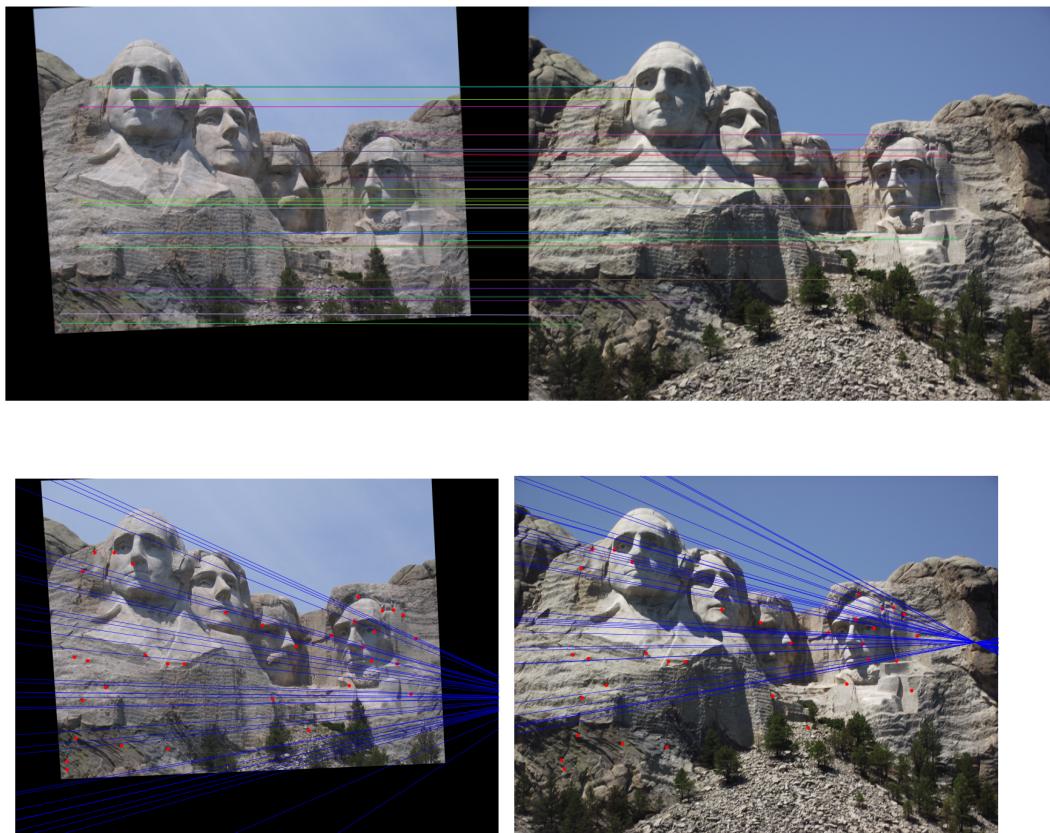
err = np.abs(np.matmul(A, best_F_matrix.reshape((-1))))
index = np.argsort(err)
# print(best_F_matrix)
# print(np.sum(err <= threshold), "/", err.shape[0])
return best_F_matrix, matches_a[index[:34]], matches_b[index[:34]]

```

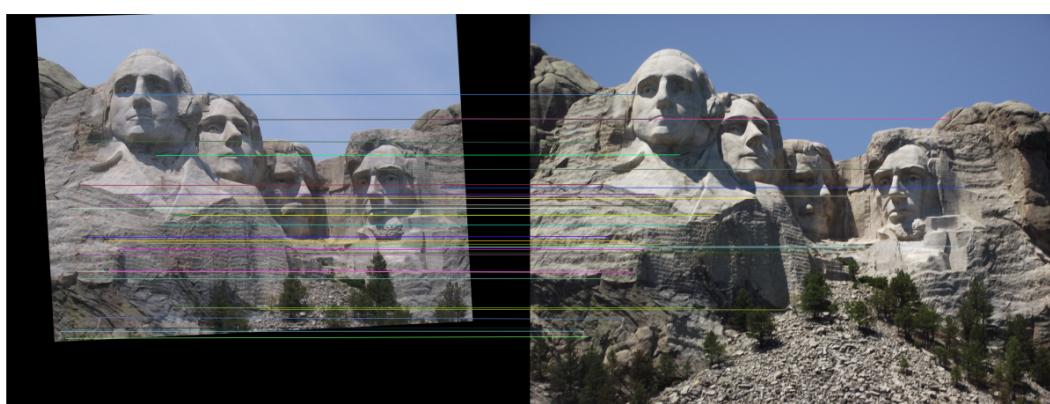
Kết quả

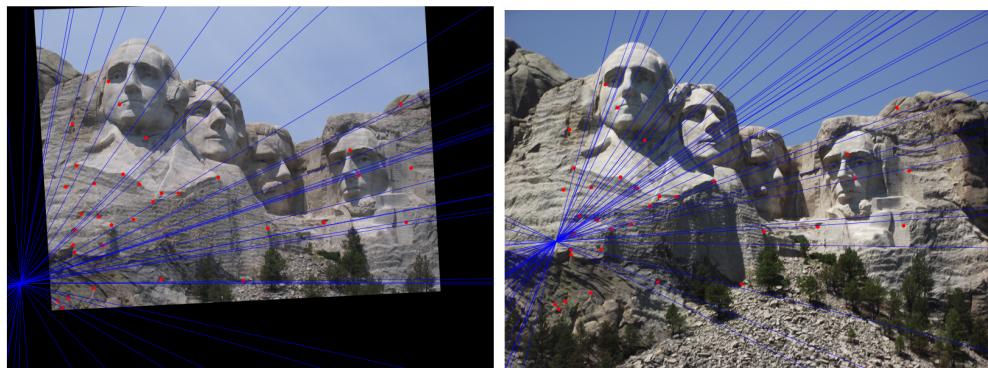
mt_rushmore

Noise ratio = 0.0

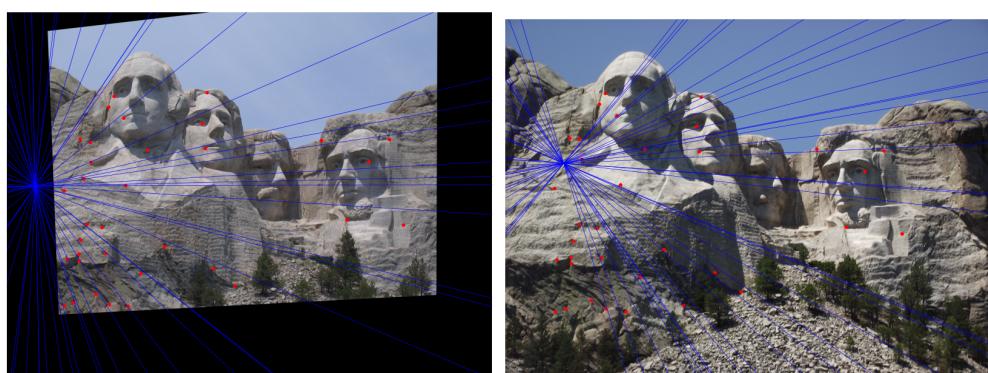
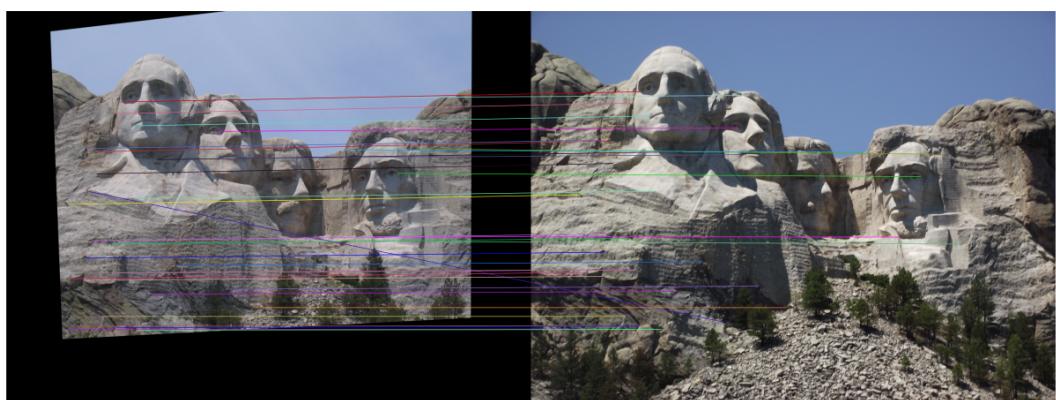


Noise ratio = 0.1

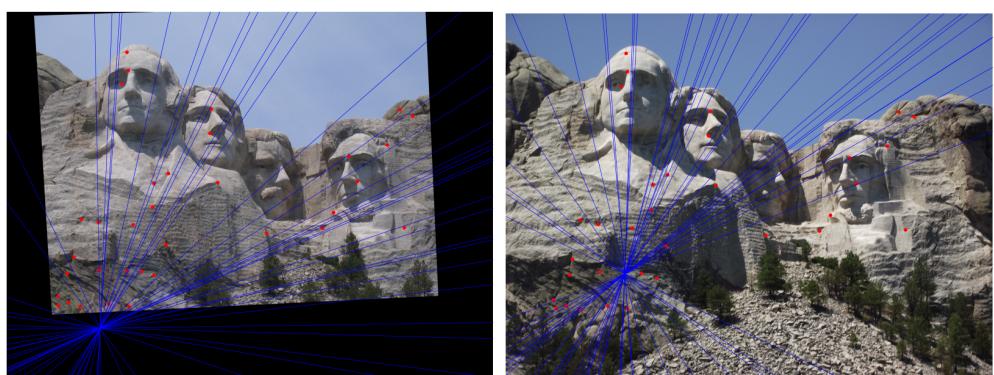
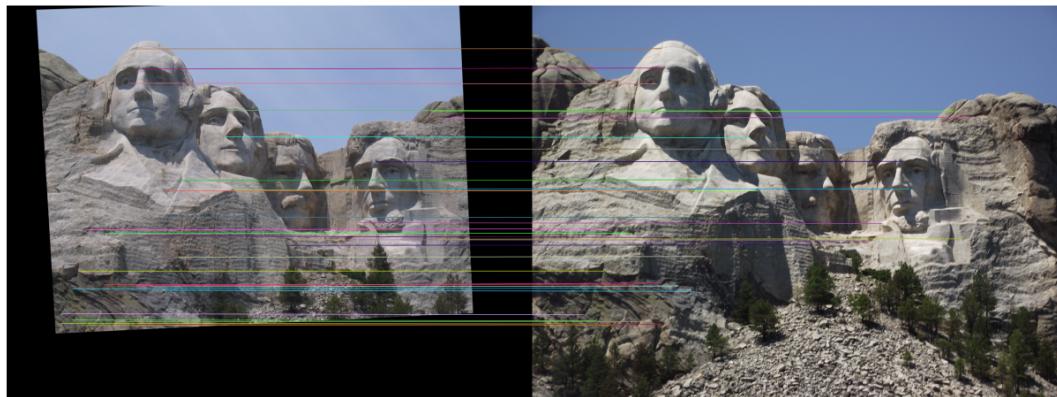




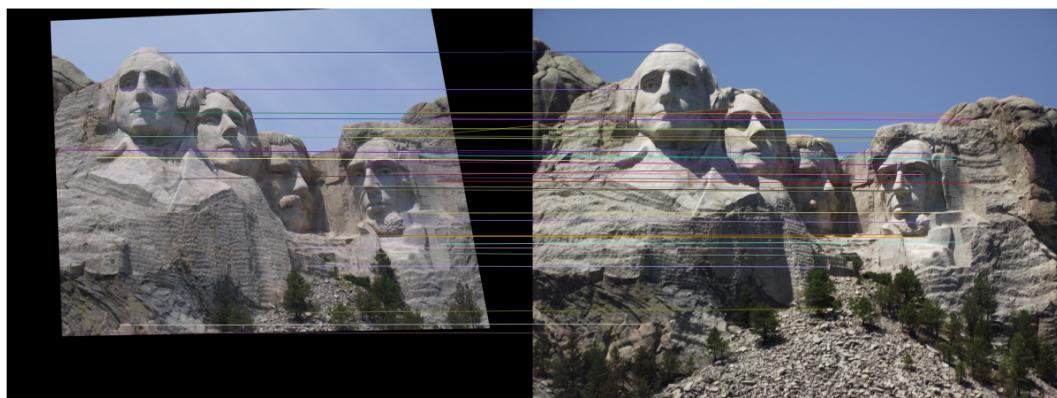
Noise ratio = 0.2

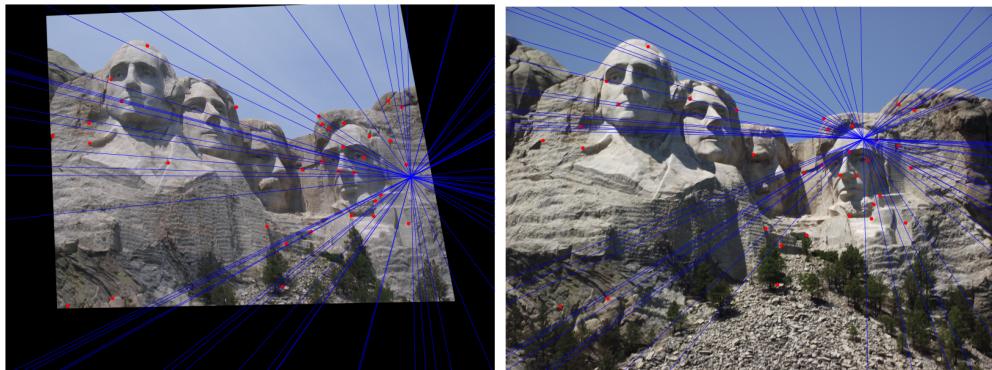


Noise ratio = 0.3



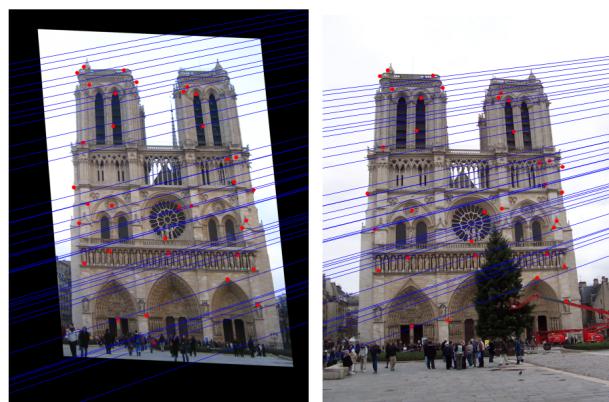
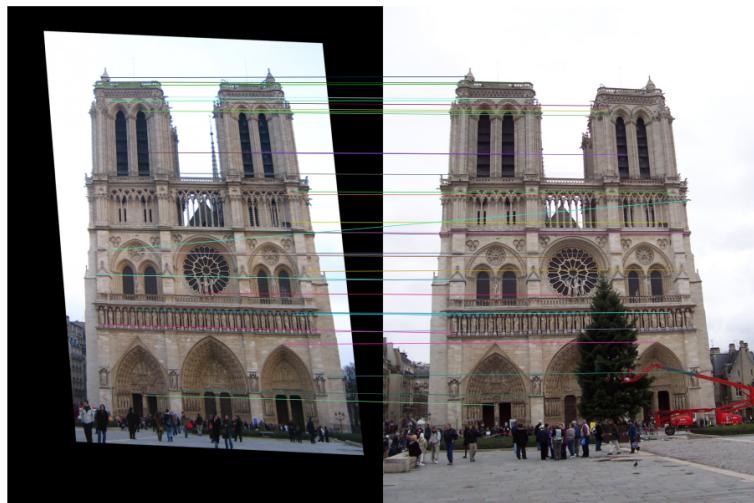
Noise ratio = 0.4





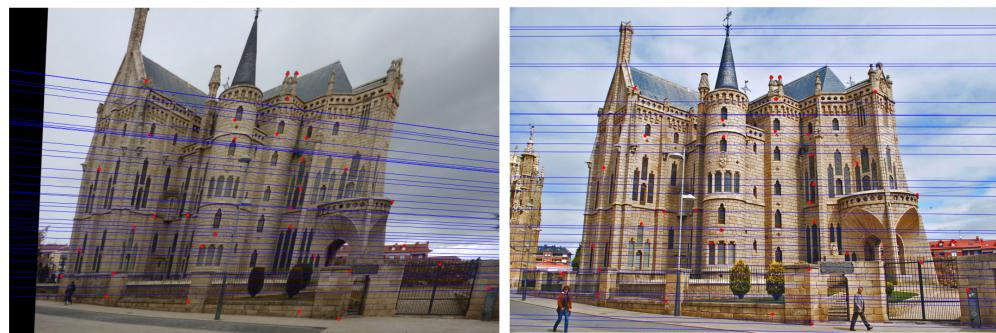
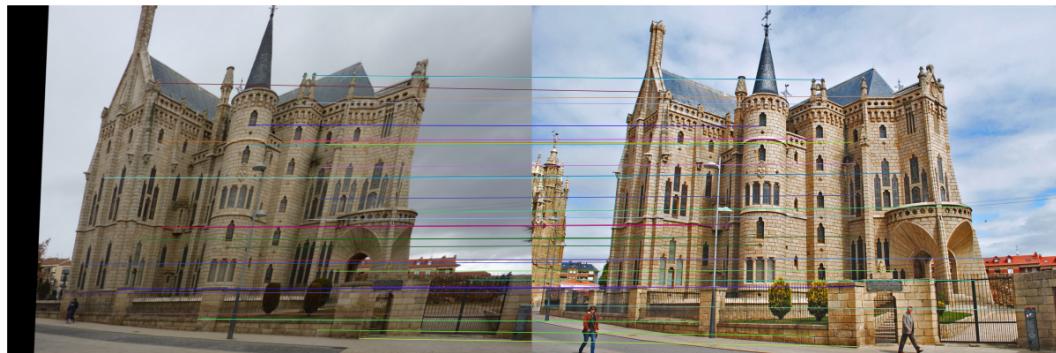
notre_dame

Noise ratio = 0.2



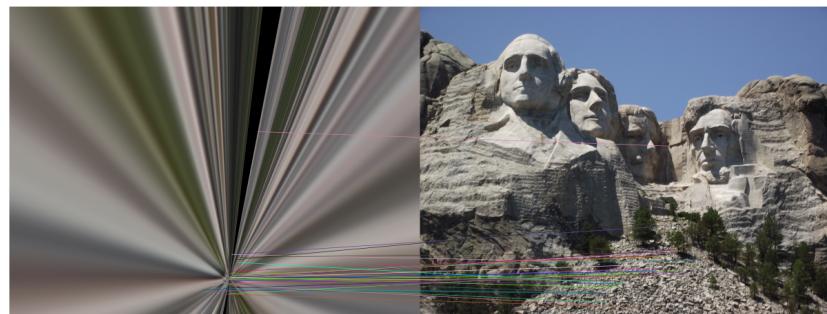
gaudi

Noise ratio = 0.2

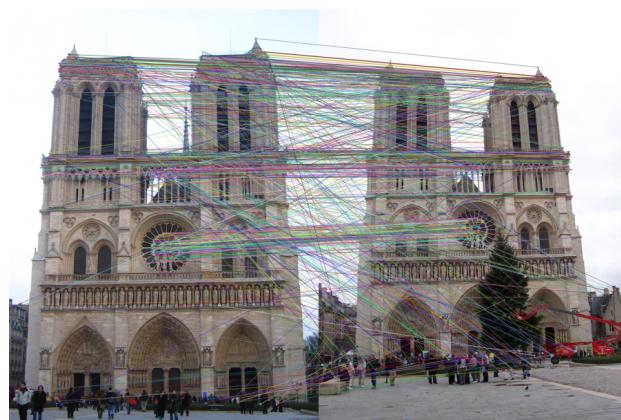


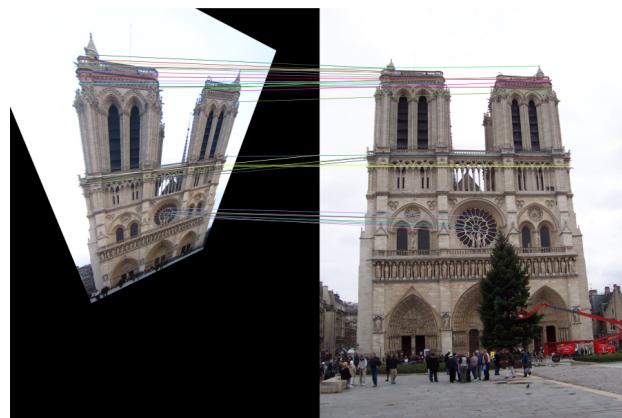
Sử dụng ORB features mt_rushmore:





notre_dame:





gaudi:

