第三次作业报告

Fundamental matrix estimation with ground truth matches

算法流程

normalize points

先计算得到所有点坐标 x,y 的均值和标准差，分别使用 `np.mean()`和 `np.std()`方法。用m表示均值，s表示标准差，变换矩阵显然是

$$\begin{pmatrix} \frac{1}{s_x} & 0 & -\frac{m_x}{s_x} \\ 0 & \frac{1}{s_y} & -\frac{m_y}{s_y} \\ 0 & 0 & 1 \end{pmatrix}$$

.

fit_fundamental

首先，输入的坐标都是 2D 坐标而不是齐次坐标。将所有点的坐标进行 `np.hstack((pic, np.ones((pic.shape[0], 1))))` ,添加了一列1作为齐次坐标，方便后续处理。

根据 lecture 7 Epipolar Geometry 46 页公式，构造$A = (x'x, x'y, x', y'x, y'y, y', x, y, 1)$.

建立方程：$AF = 0$,其中$F = \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix}$.

使用 SVD 分解A得到$F = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix}$.这里的$F$应该是一个三阶的矩阵。

对F做二阶限制。将F做 SVD 分解$F = U\Sigma V^T$.将Σ的第三个元素设置为0,再将这三个乘起来，就得到了F的二阶限制。

对于做 normalized 的结果，先对点做 `normalize_points` 操作，用 normalized 的坐标做变换，最后需要将F进行 `np.dot(T2.T, np.dot(F, T1))`的变换。

成果展示

首先展示不使用 `normalize_points` 的结果:

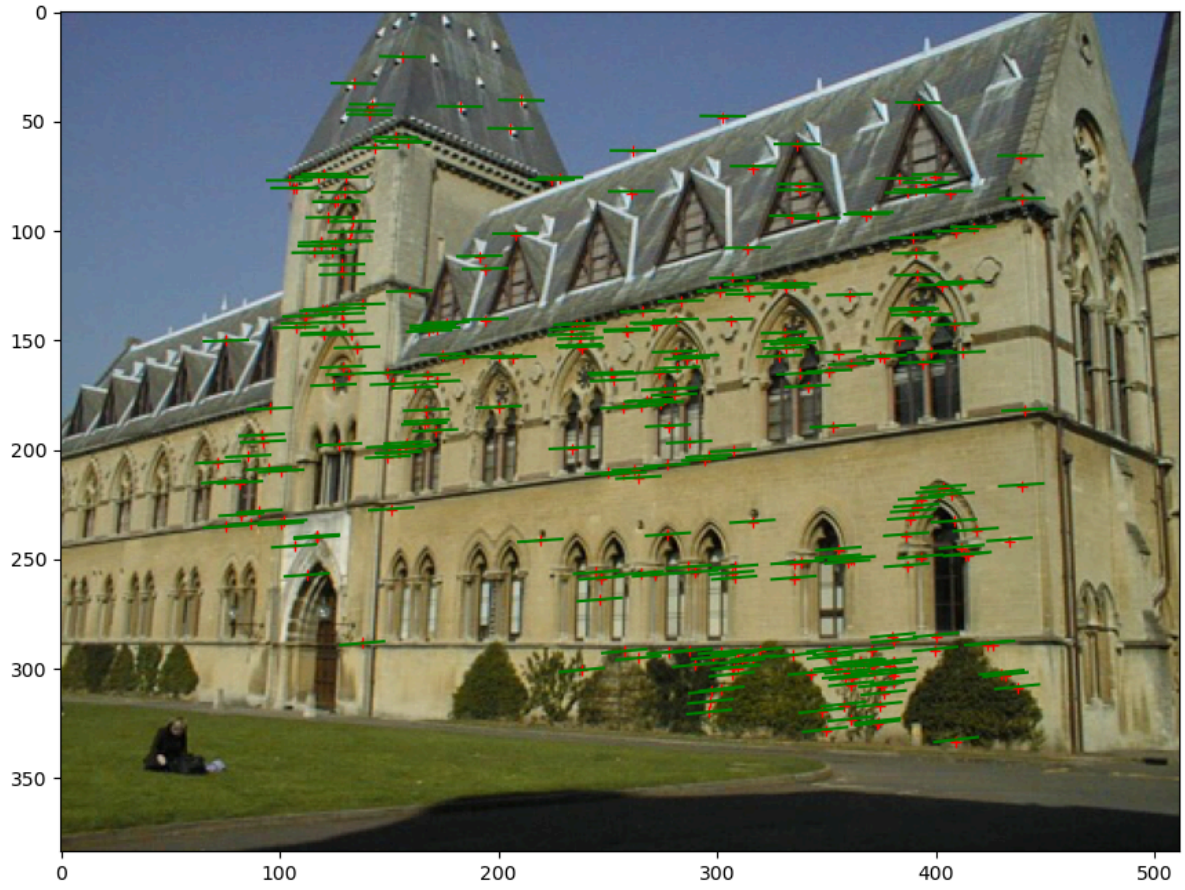

Figure 1: 不使用 normalize_points 的结果

```
library_F = [[-1.32341616e-06  1.36640519e-05 -6.82803870e-04]
 [-2.88178174e-05  2.66440807e-07  4.09069255e-02]
 [ 5.62362952e-03 -3.72771609e-02 -9.98451273e-01]]
error = 0.00016149806537624607
```

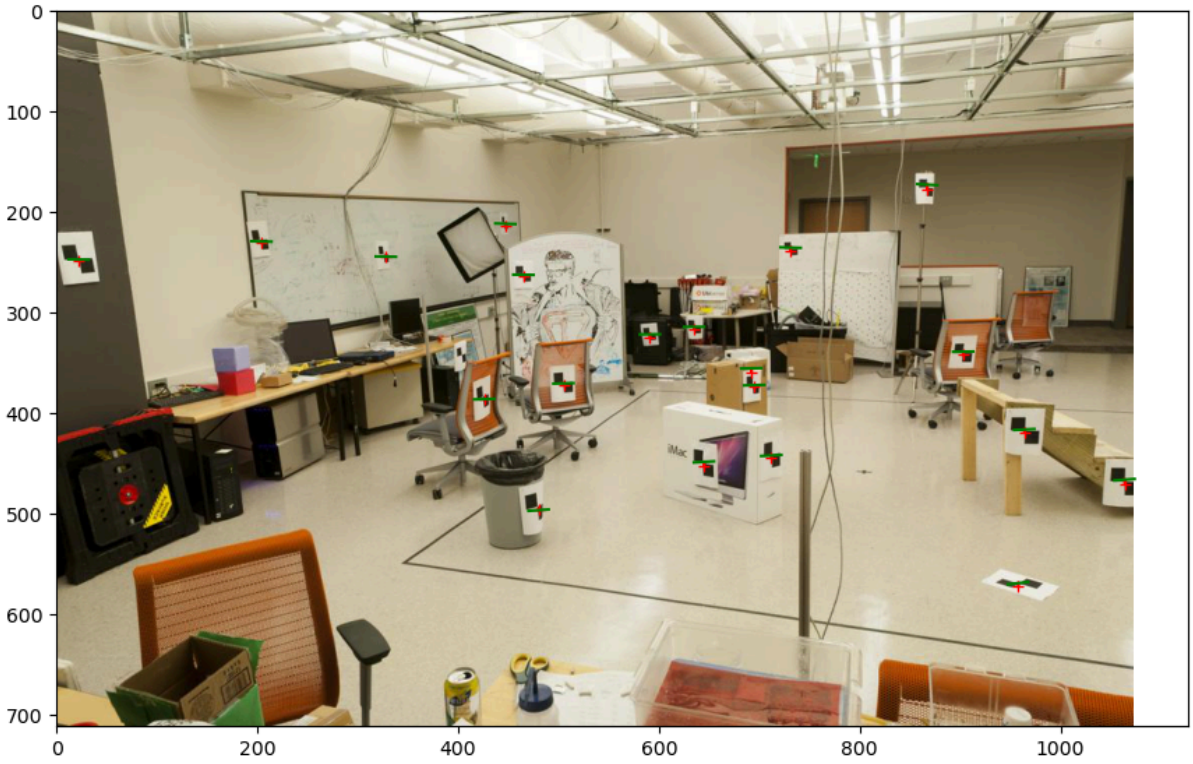

Figure 2: 不使用 normalize_points 的结果

```
lab_F = [[-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]]
error = 0.003763402377057396
```

展示完整结果

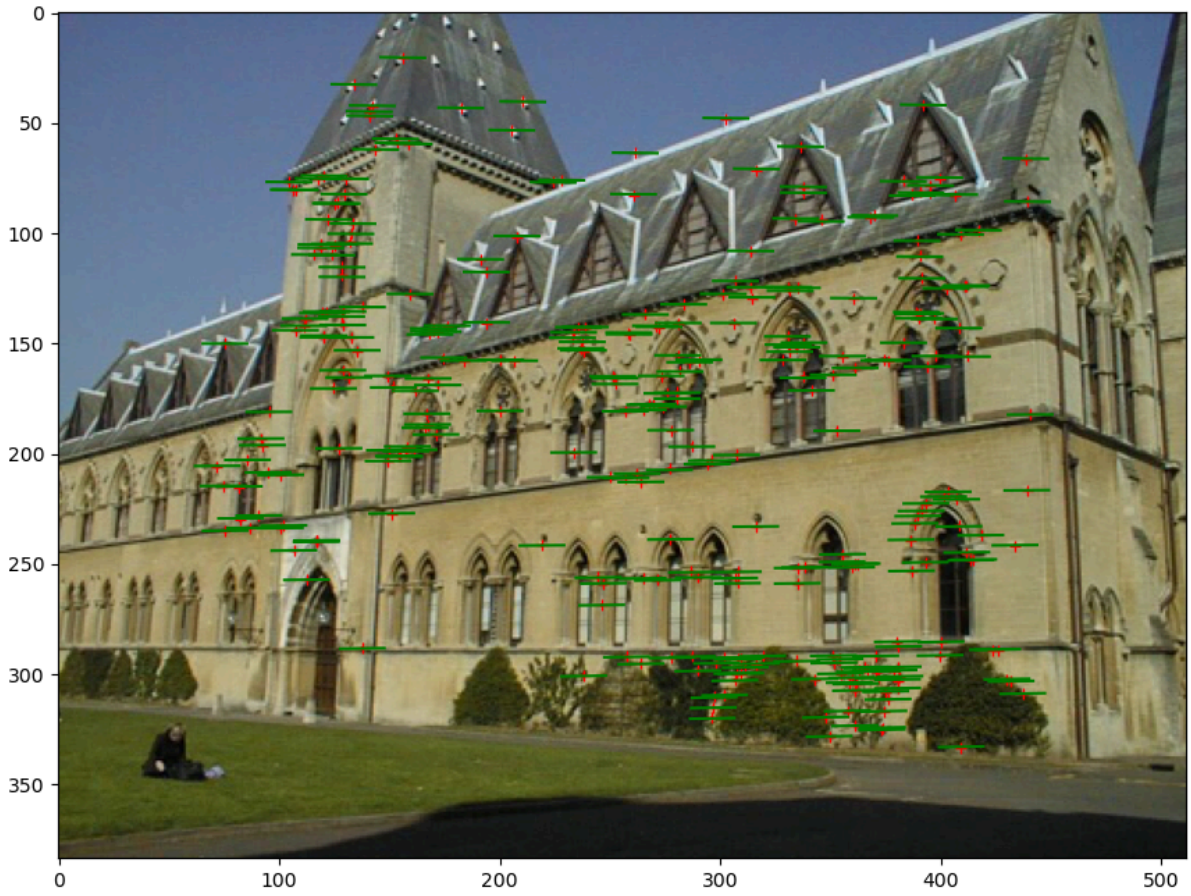

Figure 3: 二阶限制+normalize_points 的结果

```
library_F = [[-4.86210796e-08  9.98237781e-07 -1.47341026e-04]
 [-5.80698246e-06 -5.65060960e-08  1.07254893e-02]
 [ 1.38164930e-03 -9.63586245e-03 -2.60674702e-01]]
error = 4.857831302276383e-06
```

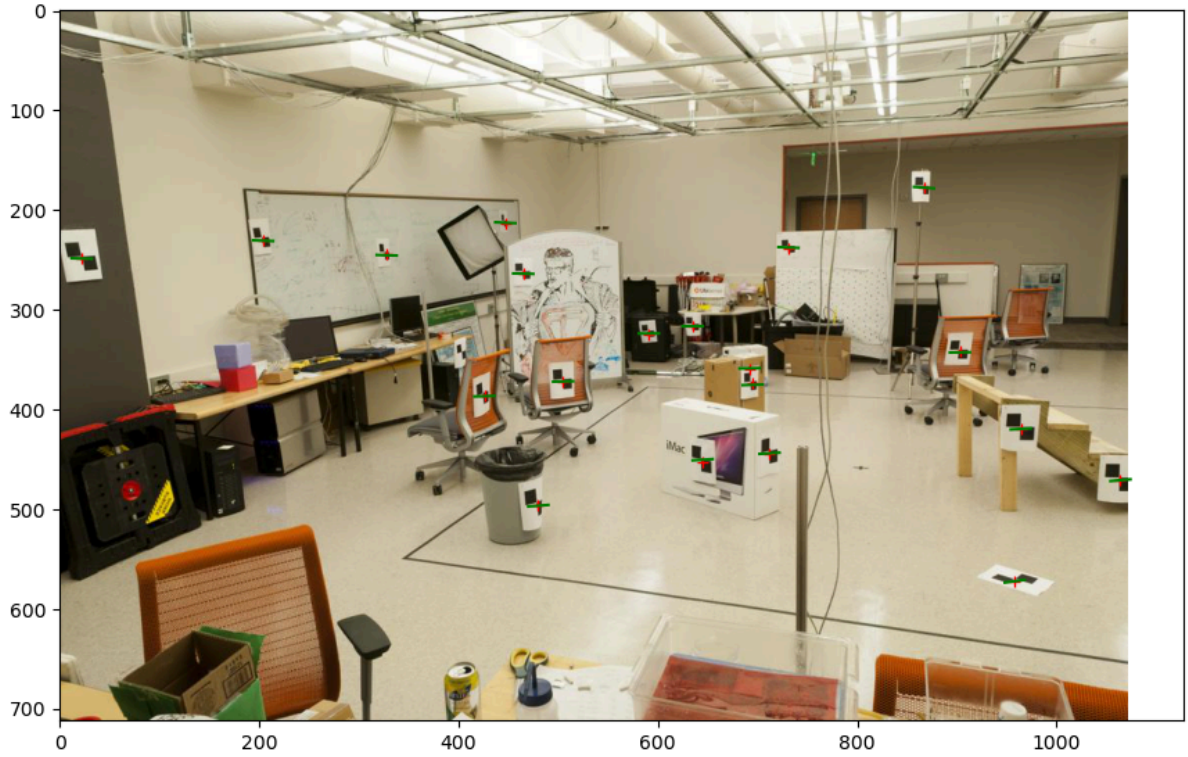

Figure 4: 二阶限制+normalize_points 的结果

```
lab_F = [[-2.02478523e-07  2.78039315e-06 -6.94781145e-04]
 [ 1.92581510e-06 -4.74398800e-07  5.59813726e-03]
 [-4.16157075e-05 -7.69192568e-03  1.78588215e-01]]
error = 2.2277388816998428e-05
```

容易看到，不做 normalized 的结果 evaluate_fundamental 的结果差，这是因为xy的量级在 1e6，而A还有 1，数据差异巨大，导致数值稳定性差。

为了更好展示二阶限制的作用，我修改了 visualize_fundamental 函数，加长了线段长度，期望获得交点信息。由于 library 两相机接近平行，交点很远，这里用 lab 来展示效果。

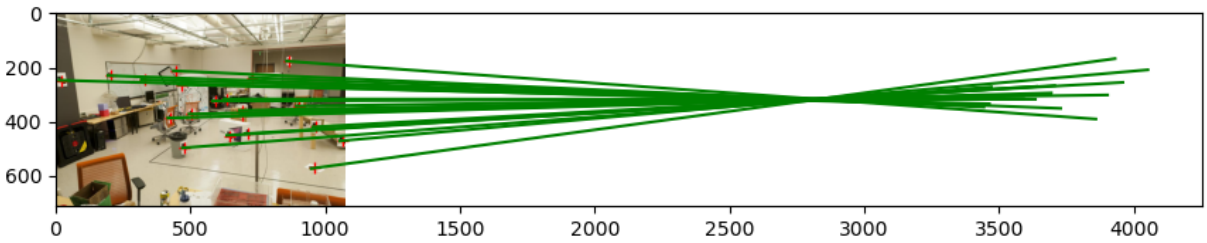

Figure 5: 不使用 rank2 的结果

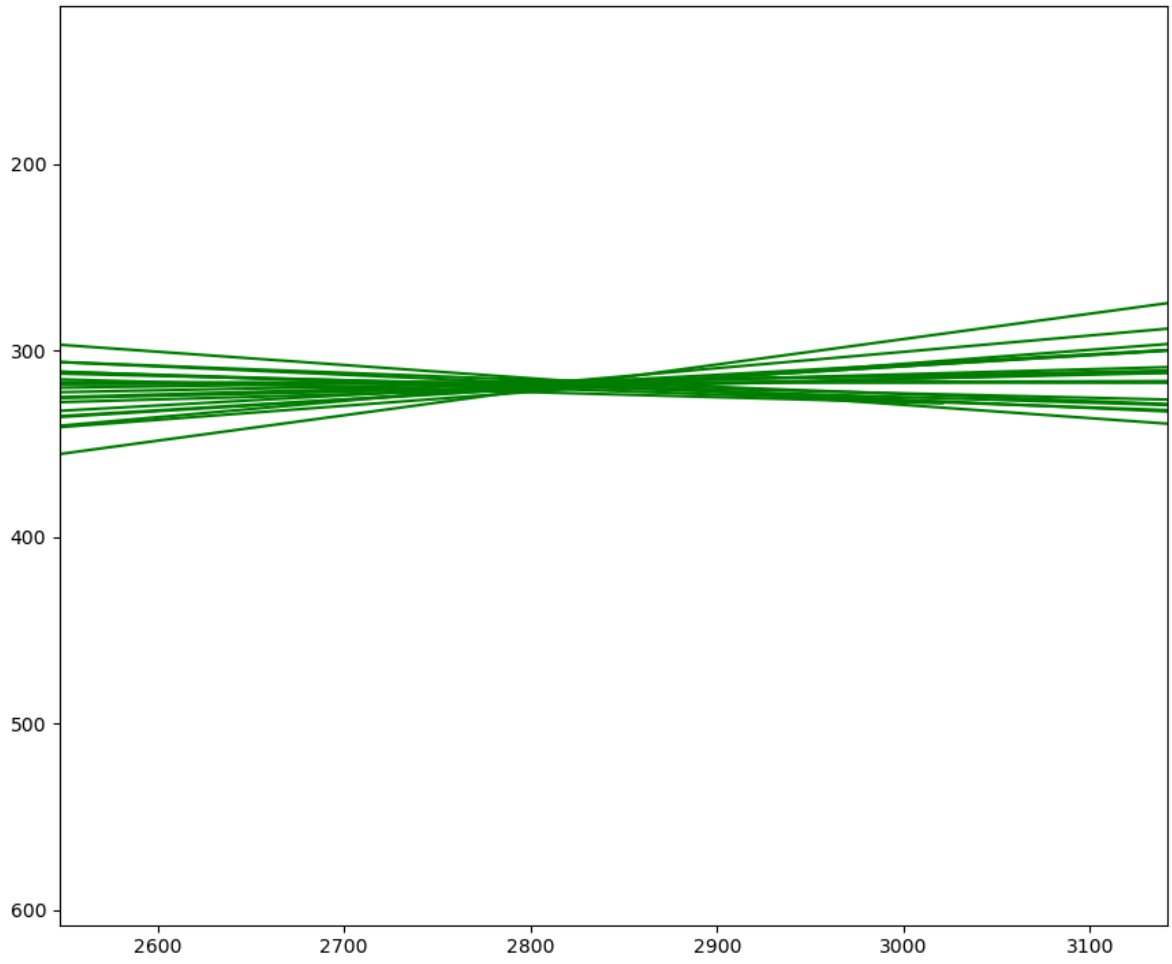

Figure 6: 不使用 rank2 的结果，交点附近

```
lab_F = [[-1.82243665e-07  2.78731312e-06 -7.08672338e-04]
 [ 1.92418726e-06 -4.74955494e-07  5.59925478e-03]
 [-5.29128050e-05 -7.69578909e-03  1.86343651e-01]]
error = 1.8926960469538197e-05
```

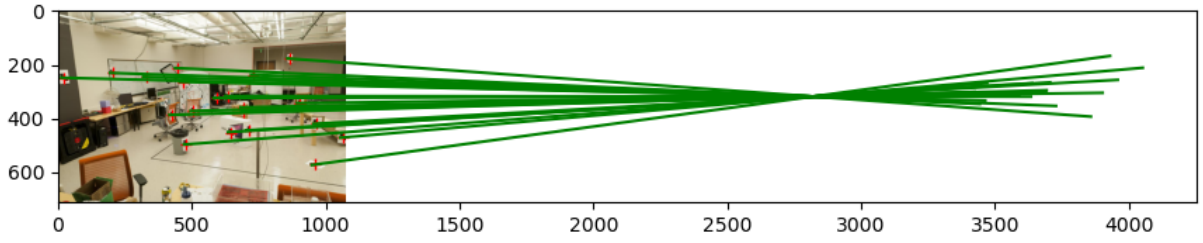

Figure 7: 使用 rank2 的结果

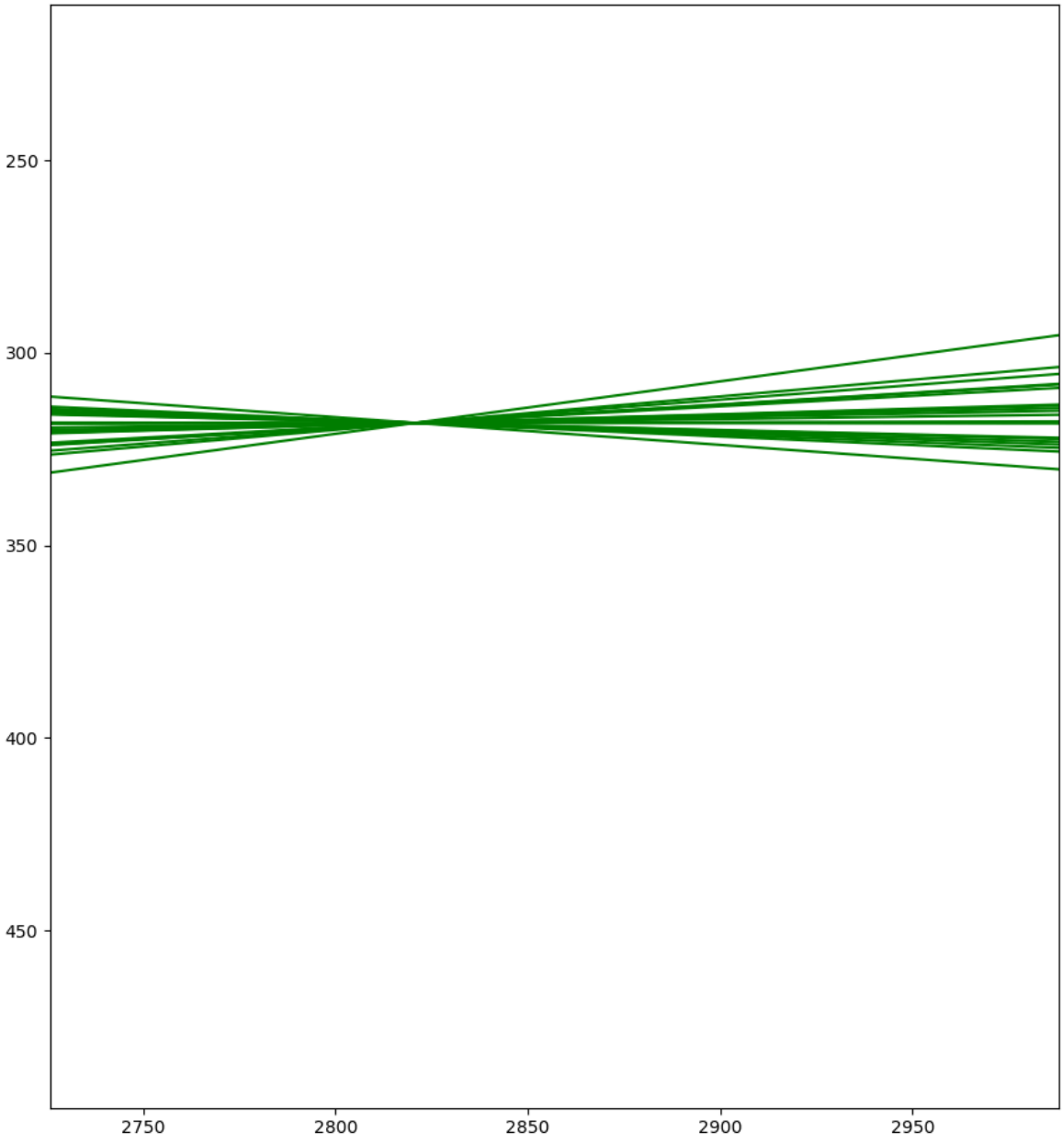

Figure 8: 使用 rank2 的结果，交点附近

```
lab_F = [[-2.02478523e-07  2.78039315e-06 -6.94781145e-04]
 [ 1.92581510e-06 -4.74398800e-07  5.59813726e-03]
 [-4.16157075e-05 -7.69192568e-03  1.78588215e-01]]
error = 2.2277388816998428e-05
```

可以观察到，使用二阶限制后，各极线可以保证相交到同一个交点，虽然 error 有轻微上升，但保证了极点只有一个，为后续操作提供了便利。

Camera calibration

根据 ppt 给的公式，建立方程 $Ap = 0$,其中

$$p = \begin{pmatrix} p_{11} \\ p_{12} \\ \vdots \\ p_{34} \end{pmatrix}$$

$$A = \begin{pmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -x_iX_i & -x_iY_i & -x_iZ_i & x_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -y_iX_i & -y_iY_i & -y_iZ_i & y_i \end{pmatrix}$$

这个公式是由$x_i = \frac{p_{11}X_i+p_{12}Y_i+p_{13}Z_i+p_{14}}{p_{31}X_i+p_{32}Y_i+p_{33}Z_i+p_{34}}$，$y_i = \frac{p_{21}X_i+p_{22}Y_i+p_{23}Z_i+p_{24}}{p_{31}X_i+p_{32}Y_i+p_{33}Z_i+p_{34}}$推导出来的。对$A$做 SVD 分解，取其特征向量，就是待求的$p$.

按照要求，计算得到如下数据（输出格式有一定调整）。

```
lab_a residual: 13.545832902721846, distance: 3.96522959141256
lab_b residual: 15.544953451380152, distance: 3.861780696389741
{
'lab_a': array(
[[ 3.09963996e-03,1.46204548e-04,-4.48497465e-04,-9.78930678e-01],
[3.07018252e-04,6.37193664e-04,-2.77356178e-03,-2.04144405e-01],
[1.67933533e-06,2.74767684e-06,-6.83964827e-07,-1.32882928e-03]]
),
'lab_b': array(
[[6.93154686e-03,-4.01684470e-03,-1.32602928e-03,-8.26700554e-01],
[1.54768732e-03,1.02452760e-03,-7.27440714e-03,-5.62523256e-01],
[7.60946050e-06,3.70953989e-06,-1.90203244e-06,-3.38807712e-03]]
)}
```

效果不好，residual 和 distance 都在要求边缘。首先考虑到可能是数据跨度过大，导致出现了数值不稳定的情况。使用标准化。

```
lab_a residual: 13.5381466167412, distance: 3.973751890320967
lab_b residual: 16.759821084663233, distance: 4.156735113400656
{
'lab_a': array(
[[-8.99278804e+01, -4.22448464e+00, 1.29243940e+01,2.83982903e+04],
[-8.89912312e+00, -1.84861272e+01, 8.04469706e+01,5.92064335e+03],
[-4.86945589e-02, -7.97162209e-02,1.97019314e-02,3.85481065e+01]]
),
'lab_b': array(
[[ 7.61166580e+01,-4.45744104e+01,-1.35108635e+01,-8.96480443e+03],
[ 1.69477440e+01, 1.11143148e+01, -7.95831840e+01,-6.12882425e+03],
[ 8.34676190e-02,4.02760463e-02,-1.93990512e-02,-3.70772887e+01]]
)}
```

结果证明是负优化。其原因应该为标准化过程中出现了更多的误差累计，使得效果更差。

Calculate the camera matrices

直接对矩阵做 RQ 分解即可。R做标准化作为K，将标准化系数乘到Q上，取Q的前三列作为R, 最后一列作为T即可。

```
lab_a
P =
[[ 3.09963996e-03  1.46204548e-04 -4.48497465e-04,-9.78930678e-01]
 [ 3.07018252e-04  6.37193664e-04 -2.77356178e-03 -2.04144405e-01]
 [ 1.67933533e-06  2.74767684e-06 -6.83964827e-07 -1.32882928e-03]]
k= [[ 1.98664468e+00 -1.29380880e-01  7.36685395e+02]
 [ 0.00000000e+00  2.01499133e+00  1.53628678e+02]
```

```
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
r= [[ 9.39843569e-04  9.34829160e-04  9.26128967e-05]
 [ 9.39093923e-04 -9.38345590e-04 -5.83756780e-05]
 [ 2.43297244e-05  1.06735796e-04 -1.32431595e-03]
 [ 1.67933533e-06  2.74767684e-06 -6.83964827e-07]]
t= [ 3.07306367e-06 -7.23411157e-07  9.33091101e-07 -1.32882928e-03]
lab_b
P =
[[ 6.93154686e-03 -4.01684470e-03 -1.32602928e-03 -8.26700554e-01]
 [ 1.54768732e-03  1.02452760e-03 -7.27440714e-03 -5.62523256e-01]
 [ 7.60946050e-06  3.70953989e-06 -1.90203244e-06 -3.38807712e-03]]
k= [[2.08947648e+00 2.29694883e-01 2.44004790e+02]
 [0.00000000e+00 2.05909423e+00 1.66031713e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
r= [[ 2.37375649e-03  2.40574261e-03  2.38250162e-04]
 [ 2.41356663e-03 -2.37742473e-03 -4.10062051e-05]
 [ 1.38058547e-04  1.98449554e-04 -3.37945167e-03]
 [ 7.60946050e-06  3.70953989e-06 -1.90203244e-06]]
t= [ 7.83159413e-06  2.84078045e-06  2.42454172e-06 -3.38807712e-03]
library_a
P =
[[-4.5250208e+01  4.8215478e+02  4.0948922e+02  3.4440464e+03]
 [ 4.8858466e+02  2.7346374e+02 -1.3977268e+02  4.8030231e+03]
 [-1.9787463e-01  8.8042214e-01 -4.3093212e-01  2.8032556e+01]]
k= [[ 21.21286351  -0.29129139 123.02928104]
 [ -0.         -19.27661824 171.37923857]
 [ -0.          -0.           1.        ]]
r= [[ -7.08877068  20.93105381 -17.24872755]
 [ -1.35772999  17.5358122   21.85007281]
 [-27.10518292  -6.35887802   3.41967977]
 [ -0.19787463   0.88042214  -0.43093212]]
t= [-0.97257923 -0.22444187  0.0609548  28.032556  ]
library_b
P =
[[-5.9593834e+01  5.5643970e+02  2.3093716e+02  3.5683545e+03]
 [ 4.6419679e+02  2.2628430e+02 -1.9605278e+02  4.8734171e+03]
 [-1.9116708e-01  7.2057697e-01 -6.6650130e-01  2.8015392e+01]]
k= [[ 20.07841117  -0.28195256 127.53798518]
 [ -0.         -18.33074841 173.99447118]
 [ -0.          -0.           1.        ]]
r= [[ -6.69328871  14.94407738 -22.73310409]
 [ -2.13484931  23.05892914  15.7967288 ]
 [-27.13794297  -5.50484295   4.36890175]
 [ -0.19116708   0.72057697  -0.6665013 ]]
t= [-0.97087836 -0.23184776  0.06034756 28.015392  ]
```

Triangulation

依题意容易构造

```
A[0] = [P1[0, 0] - point1[0] * P1[2,0],P1[0, 1] - point1[0] * P1[2,1],P1[0, 2] -
point1[0] * P1[2,2],P1[0, 3] - point1[0] * P1[2,3]]
A[1] = [P1[1, 0] - point1[1] * P1[2,0],P1[1, 1] - point1[1] * P1[2,1],P1[1, 2] -
point1[1] * P1[2,2],P1[1, 3] - point1[1] * P1[2,3]]
A[2] = [P2[0, 0] - point2[0] * P2[2,0],P2[0, 1] - point2[0] * P2[2,1],P2[0, 2] -
point2[0] * P2[2,2],P2[0, 3] - point2[0] * P2[2,3]]
A[3] = [P2[1, 0] - point2[1] * P2[2,0],P2[1, 1] - point2[1] * P2[2,1],P2[1, 2] -
point2[1] * P2[2,2],P2[1, 3] - point2[1] * P2[2,3]]
```

使得$AX = 0$,X为三维坐标。

做 SVD 分解，取其特征向量，将三维齐次坐标归一化，就得到了待求的三维坐标。

residual 和坐标信息如下

```
Lab Triangulation:
residual_3d:0.021
residual_3d:0.003
residual_3d:0.011
residual_3d:0.004
residual_3d:0.025
residual_3d:0.009
residual_3d:0.003
residual_3d:0.019
residual_3d:0.006
residual_3d:0.006
residual_3d:0.009
residual_3d:0.007
residual_3d:0.028
residual_3d:0.007
residual_3d:0.011
residual_3d:0.028
residual_3d:0.010
residual_3d:0.013
residual_3d:0.023
residual_3d:0.022
312.766,309.148,30.090
305.794,311.651,30.357
307.699,312.368,30.416
310.147,307.182,29.299
311.953,310.124,29.219
311.195,307.570,30.677
307.107,306.879,28.659
309.309,312.472,30.231
307.437,310.145,29.316
308.249,306.296,28.884
306.644,309.296,28.908
308.071,306.837,29.191
309.650,308.817,29.035
308.258,309.961,29.264
307.557,308.614,28.964
311.014,309.189,28.908
307.522,308.184,29.066
309.959,311.271,29.991
312.177,310.787,29.075
311.971,312.695,30.515
residual_a: 10.899
residual_b: 1.549
Library Triangulation:
312.766,309.148,30.090
305.794,311.651,30.357
307.699,312.368,30.416
310.147,307.182,29.299
311.953,310.124,29.219
311.195,307.570,30.677
307.107,306.879,28.659
```

```
309.309,312.472,30.231
307.437,310.145,29.316
308.249,306.296,28.884
306.644,309.296,28.908
308.071,306.837,29.191
309.650,308.817,29.035
308.258,309.961,29.264
307.557,308.614,28.964
311.014,309.189,28.908
307.522,308.184,29.066
309.959,311.271,29.991
312.177,310.787,29.075
311.971,312.695,30.515
residual_a: 24.662
residual_b: 28.650
```

Fundamental matrix estimation without ground-truth matches

首先通过 SIFT 提取特征点和对应的 SIFT feature,使用 `cv2.BFMatcher()`进行匹配，得到匹配结果 `matches`。

对匹配结果使用 RANSAC 方法，随机抽 8 个点对求 fundamental 矩阵。使用求得的 fundamental 矩阵计算极线，通过极线到匹配点之间的距离计算 inlier 数量。选取抽样中 inlier 数量最多的点对作为最终 inlier 点集，使用该点集计算 fundamental 矩阵，得到最终的 fundamental 矩阵。可视化结果如下。

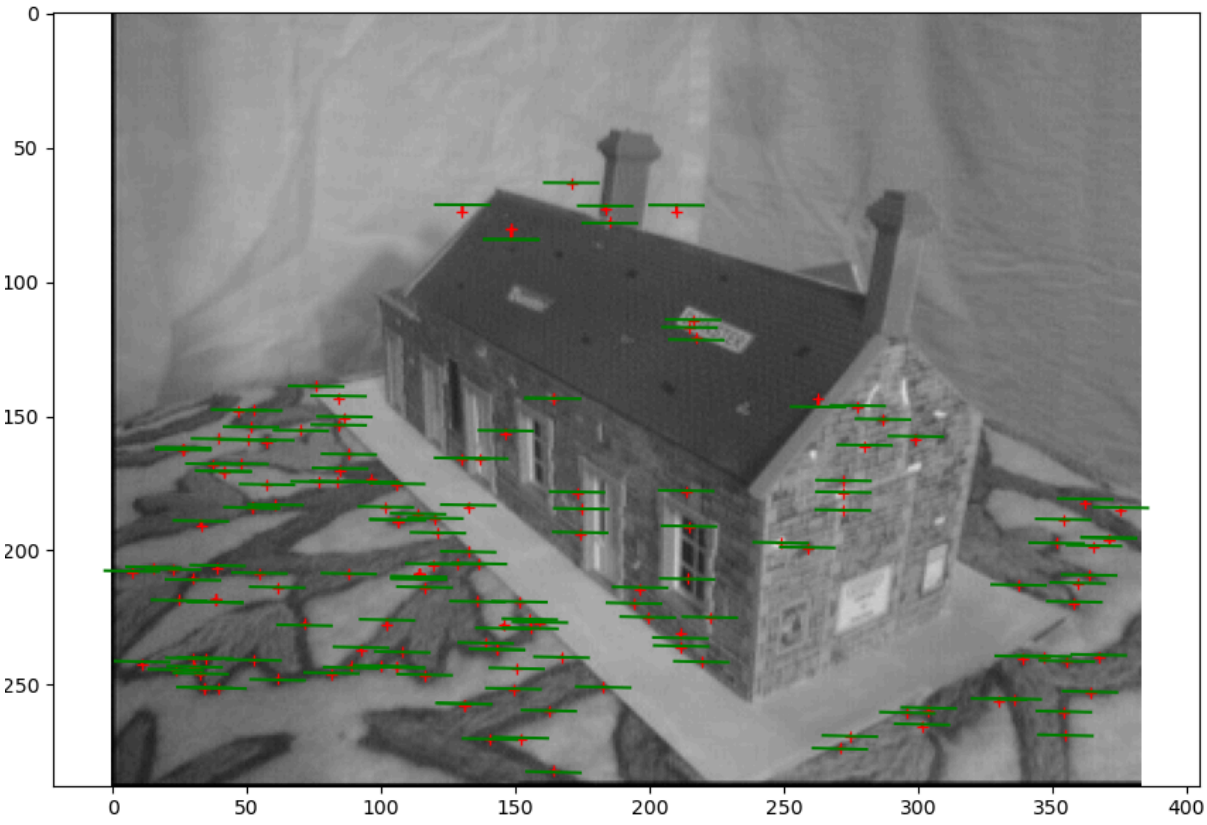

Figure 9: house 结果

```
Error =  0.06431870403436402
inlier = 160
house_F = [[ 7.84355918e-07  7.09044305e-06 -1.68567138e-03]
```

```
 [ 2.71888729e-06  1.26061615e-06 -1.63713602e-02]
 [ 9.74132904e-04  1.38309445e-02  1.75291457e-01]]
```

由于 gaudi 中特征点过多，我选择进行更严格的筛选。对响应值进行排序，选择响应值最大的前 500 个点对，计算 fundamental 矩阵，可视化结果如下。

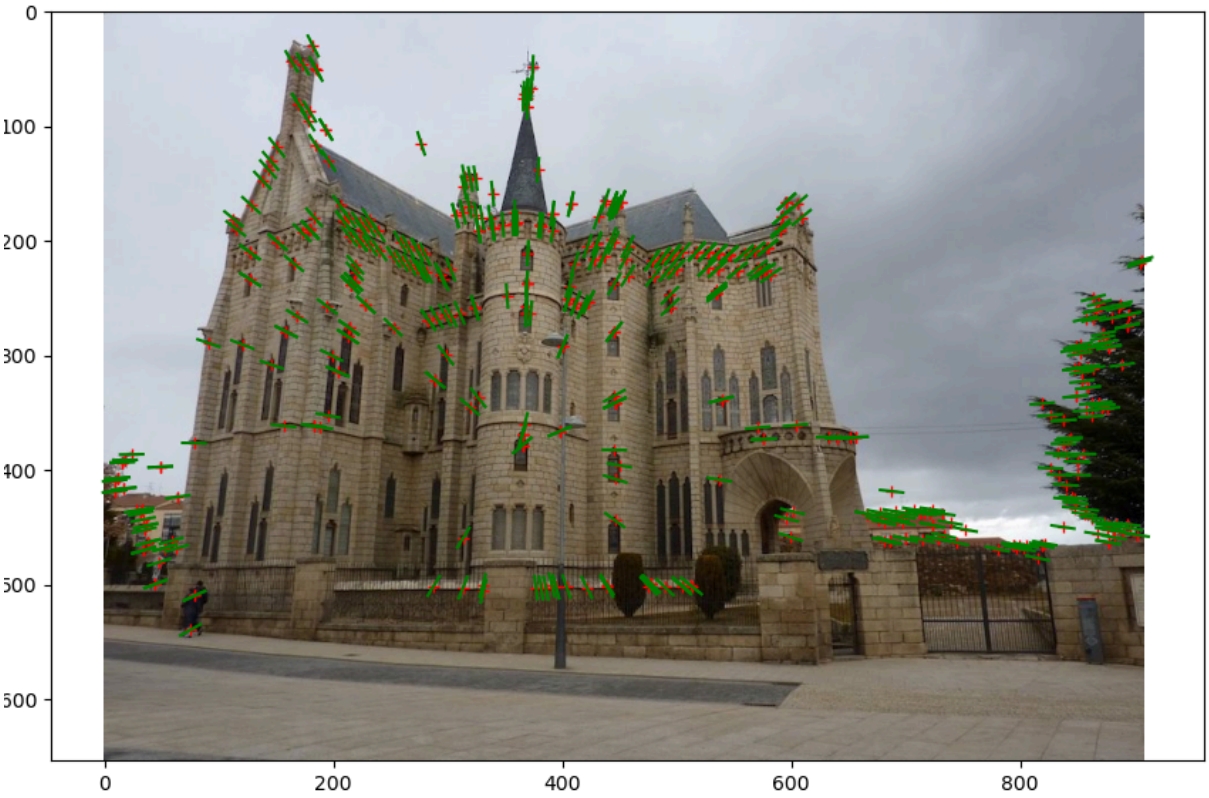

Figure 10: gaudi 结果

```
error =  1.1421137607235971e-30
inlier = 500
gaudi_F = [[ 8.07641557e-22  1.78720913e-05 -6.81182853e-03]
 [-1.78720913e-05  1.77433039e-20  6.46038511e-03]
 [ 6.81182853e-03 -6.46038511e-03 -3.10862447e-15]]
```