感知与人机交互课程实验报告

71121117 马骁宇

实验一：三维空间刚体运动

实验内容:

2 三维空间刚体运动：实验作业

参考《视觉SLAM十四讲》3.6.1 P62

- 已知：两个坐标系A和B
  - A和B之间的关系：坐标系A以Z轴为旋转轴，旋转45°后与坐标系B重合
- 已知：一向量在坐标系A中的坐标为$(1,0,0)^T$
- 问题1：请求出可描述两坐标系之间变换关系的旋转矩阵
- 问题2：请利用旋转矩阵，确定该向量在坐标系B中的坐标
- 问题3：请利用旋转矩阵，确定两坐标系之间的欧拉角
- 问题4：若两坐标系之间除了旋转关系外，还存在平移量 $(1,3,4)^T$，请求出可描述两坐标系之间变换关系的变换矩阵，并确定该向量在坐标系B中的坐标

74

实验结果:

Q1. $[e_1,e_2,e_3]x_A = [e_1',e_2',e_3']x_B$

$$\Rightarrow R_{BA} = \begin{bmatrix} \cos45°, & -\sin45°, & 0 \\ \sin45°, & \cos45°, & 0 \\ 0, & 0, & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2}, & -\frac{\sqrt{2}}{2}, & 0 \\ \frac{\sqrt{2}}{2}, & \frac{\sqrt{2}}{2}, & 0 \\ 0, & 0, & 1 \end{bmatrix}$$

Q2. $[r]_B = R_{BA}^T [r]_A$

$$= \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{bmatrix}$$

Q3. $\varphi = 45°,\ \theta = 0°,\ \psi = 0°$.

Q4. $[r]_A = R_{BA}[r]_B + \vec{t_p}$ $\quad \vec{t_p} = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$

$[r]_A' = \begin{bmatrix} R_{BA} & \vec{t_p} \\ 0 & 1 \end{bmatrix} [r]_B'$；即：$[r]_B' = T_{BA}^{-1}[r]_A'$

$$\because R_{BA}^T = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \therefore R_{BA}^{-1}\vec{t_p} = \begin{bmatrix} 2\sqrt{2} \\ \sqrt{2} \\ 4 \end{bmatrix}$$

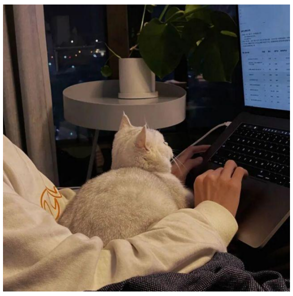
$\therefore [r]_B = [-\frac{3}{2}\sqrt{2}, -\frac{3}{2}\sqrt{2}, -4]^T$

实验二：图像去畸变

实验内容：

实验结果：

1. 原图+畸变后+恢复后

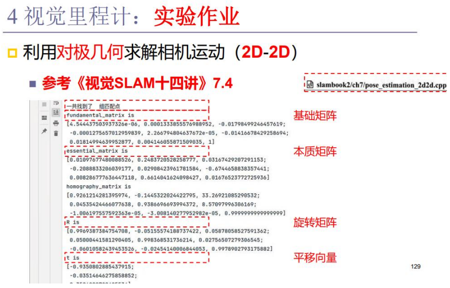


2. 总结

对原图进行畸变变换与畸变矫正，给定的畸变参数是一组互为相反数的参数，虽然理论上可以恢复，但由于精确度等问题可以发现在一些边缘上出现了模糊和扭曲

实验三：求解相机运动

实验内容：



实验结果：

一、使用对极几何方法可以实现 2d-2d 的相机信息求解

```
-- Max dist : 95.000000
-- Min dist : 4.000000
一共找到了79组匹配点
fundamental_matrix is
[4.844484382466111e-06, 0.0001222601840188731, -0.01786737827487386;
 -0.0001174326832719333, 2.122888800459598e-05, -0.01775877156212593;
 0.01799658210895528, 0.008143605989020664, 1]
essential_matrix is
[-0.02036185505234771, -0.4007110038118444, -0.033240742498241;
 0.3939270778216368, -0.03506401846698084, 0.5857110303721015;
 -0.006788487241438231, -0.5815434272915687, -0.01438258684486259]
homography_matrix is
[0.9497129583105288, -0.143556453147626, 31.20121878625771;
 0.04154536627445031, 0.9715568969832015, 5.306887618807696;
 -2.818136769787960e-05, 4.353702039810921e-05, 1]
R is
[0.9985961798781877, -0.05169917220143662, 0.01152671359827862;
 0.05139607508976053, 0.9983603445075083, 0.02520051547522452;
 -0.01281065954813537, -0.02457271064688494, 0.9996159607036126]
t is
[-0.8220841067933339;
```

二、使用 ICP-SVD 方法可以实现 3d-3d 的相机信息求解

```
2_depth.png
-- Max dist : 95.000000
-- Min dist : 4.000000
一共找到了79组匹配点
3d-3d pairs: 74
W=  298.51 -14.1815  41.0456
-44.8208  107.825 -164.404
 78.1978 -163.954  271.439
U=  0.474143  -0.880373 -0.0114952
 -0.460275  -0.258979   0.849163
  0.750556   0.397334   0.528006
V=  0.535211  -0.844064 -0.0332488
 -0.434767  -0.309001    0.84587
  0.724242   0.438263   0.532352
ICP via SVD results:
R = [0.9972395976914051, 0.05617039049497492, -0.04855998381307917;
 -0.05598344580804108, 0.9984181433274508, 0.005202390798842771;
 0.04877538920134405, -0.002469474885032463, 0.9988067195919588]
t = [0.7086246277241866;
 -0.2775515782948794;
 -0.1559573762377191]
R_inv = [0.9972395976914051, -0.05598344580804108, 0.04877538920134405;
 0.05617039049497492, 0.9984181433274508, -0.002469474885032463;
 -0.04855998381307917, 0.005202390798842771, 0.9988067195919588]
t_inv = [-0.714599950683482;
 0.2369236766013987;
 0.1916260075851264]
calling bundle adjustment
iteration= 0     chi2= 452884.696837     time= 3.94e-05  cumTime= 3.94e-05       edges= 74       schur
= 0
iteration= 1     chi2= 452762.638918     time= 1.1416e-05        cumTime= 5.0816e-05     edges= 74
schur= 0
iteration= 2     chi2= 452762.618632     time= 9.824e-06         cumTime= 6.064e-05      edges= 74
schur= 0
iteration= 3     chi2= 452762.618615     time= 9.382e-06         cumTime= 7.0022e-05     edges= 74
schur= 0
iteration= 4     chi2= 452762.618615     time= 2.3322e-05        cumTime= 9.3344e-05     edges= 74
```

实验四：非线性优化

实验内容：

6 非线性优化：实验作业

- 考虑一条满足以下方程的曲线

$$y = \exp(ax^2 + bx + c) + w$$

  - 曲线的参数：a, b, c
  - 问题：通过构造关于x, y的观测数据，求出曲线的参数
    - 工具：高斯牛顿法、Ceres优化库、g2o优化库
    - 比较：优化时间、参数估计精度等
    - 画出曲线的拟合结果

slambook2/ch6/gaussNewton.cpp
slambook/ch6/ceresCurveFitting.cpp
slambook/ch6/g2oCurveFitting.cpp

参考《视觉SLAM十四讲》6.3.1-6.3.3

39

实验结果

1、优化代码运行结果

(1) 准确值（r）与初始化（e）：

```
// minimize function: y - exp(a*x^2 + b*x + c)
double a_r = 1.0, b_r = -2.0, c_r = 2.0;
double a_e = -1.0, b_e = 3.0, c_e = 5.0;
```

(2) 牛顿高斯法运行结果

```
total cost: 3.81112e+07, estimated params: a=-0.954443, b=2.92491, c=4.03376
total cost: 5.12042e+06, estimated params: a=-0.833882, b=2.72595, c=3.12354
total cost: 679607, estimated params: a=-0.52937, b=2.22146, c=2.35351
total cost: 87414.9, estimated params: a=0.146165, b=1.08819, c=1.88663
total cost: 10450.1, estimated params: a=1.18318, b=-0.744447, c=1.85178
total cost: 1105.09, estimated params: a=1.69434, b=-2.04685, c=1.97409
total cost: 159.619, estimated params: a=1.38724, b=-2.12988, c=1.98561
total cost: 102.969, estimated params: a=1.2494, b=-2.06647, c=1.98124
total cost: 101.997, estimated params: a=1.24461, b=-2.06372, c=1.981
total cost: 101.996, estimated params: a=1.24461, b=-2.06371, c=1.98099
total cost: 101.996, estimated params: a=1.24461, b=-2.06371, c=1.98099
total cost: 101.996, estimated params: a=1.24461, b=-2.06371, c=1.98099
total cost: 101.996, estimated params: a=1.24461, b=-2.06371, c=1.98099
cost: 101.996 >= last cost: 101.996, break.
solve time cost = 0.000143419 seconds.
estimated a=1.24461, b=-2.06371, c=1.98099
```

(3) 使用 ceres 库运行结果

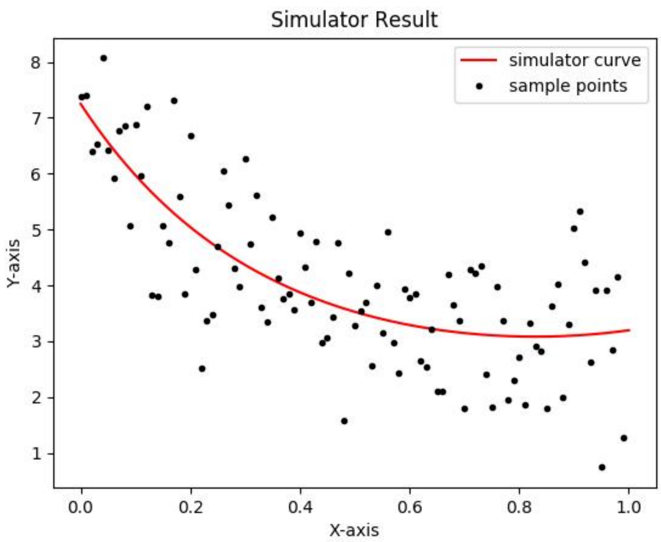
```
0.99 1.28179
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter  iter_time  total_time
   0  1.905562e+07    0.00e+00    3.83e+07   0.00e+00   0.00e+00  1.00e+04        0    2.48e-05    5.36e-05
   1  2.560703e+06    1.65e+07    5.19e+06   0.00e+00   8.66e-01  1.64e+04        1    3.60e-05    1.15e-04
   2  3.399038e+05    2.22e+06    7.05e+05   9.39e-01   8.67e-01  2.72e+04        1    1.91e-05    1.51e-04
   3  4.372178e+04    2.96e+05    9.64e+04   9.67e-01   8.72e-01  4.61e+04        1    1.79e-05    1.83e-04
   4  5.226705e+03    3.85e+04    1.33e+04   1.39e+00   8.82e-01  8.31e+04        1    1.60e-05    2.06e-04
   5  5.527616e+02    4.67e+03    1.83e+03   2.10e+00   9.04e-01  1.76e+05        1    1.79e-05    2.39e-04
   6  7.981350e+01    4.73e+02    2.39e+02   1.41e+00   9.43e-01  5.27e+05        1    2.79e-05    2.81e-04
   7  5.148373e+01    2.83e+01    2.00e+01   3.17e-01   9.83e-01  1.58e+06        1    1.81e-05    3.14e-04
   8  5.099831e+01    4.85e-01    4.52e-01   1.51e-01   9.98e-01  4.74e+06        1    1.69e-05    3.35e-04
   9  5.099798e+01    3.28e-04    4.51e-04   5.51e-03   9.99e-01  1.42e+07        1    1.60e-05    3.57e-04
solve time cost = 0.000389865 seconds.
Ceres Solver Report: Iterations: 10, Initial cost: 1.905562e+07, Final cost: 5.099798e+01, Termination: CONVERGENCE
estimated a,b,c = 1.24461 -2.06372 1.981
```

(4) 使用 g2o 库运行结果

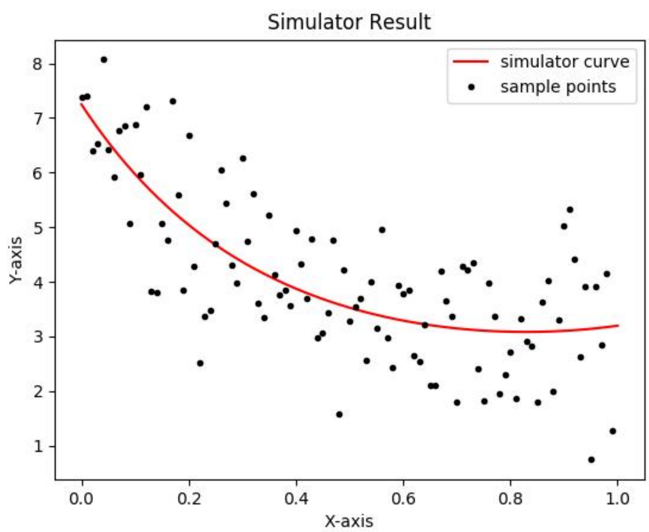
```
start optimization
iteration= 0     chi2= 1071.076974     time= 4.6282e-05     cumTime= 4.6282e-05      edges= 100     schur= 0     lambda= 699.050775     levenbergIter= 7
iteration= 1     chi2= 728.002830      time= 2.3235e-05     cumTime= 6.9517e-05      edges= 100     schur= 0     lambda= 233.016925     levenbergIter= 1
iteration= 2     chi2= 320.154258      time= 2.2813e-05     cumTime= 9.233e-05       edges= 100     schur= 0     lambda= 77.672308      levenbergIter= 1
iteration= 3     chi2= 154.890153      time= 2.2751e-05     cumTime= 0.000115081     edges= 100     schur= 0     lambda= 25.890769      levenbergIter= 1
iteration= 4     chi2= 122.661855      time= 2.2902e-05     cumTime= 0.000137983     edges= 100     schur= 0     lambda= 8.630256       levenbergIter= 1
iteration= 5     chi2= 110.228591      time= 2.2945e-05     cumTime= 0.000160928     edges= 100     schur= 0     lambda= 2.876752       levenbergIter= 1
iteration= 6     chi2= 103.101938      time= 2.2749e-05     cumTime= 0.000183677     edges= 100     schur= 0     lambda= 0.958917       levenbergIter= 1
iteration= 7     chi2= 102.023922      time= 2.2715e-05     cumTime= 0.000206392     edges= 100     schur= 0     lambda= 0.319639       levenbergIter= 1
iteration= 8     chi2= 101.996073      time= 2.2814e-05     cumTime= 0.000229206     edges= 100     schur= 0     lambda= 0.106546       levenbergIter= 1
iteration= 9     chi2= 101.995960      time= 2.3082e-05     cumTime= 0.000252288     edges= 100     schur= 0     lambda= 0.071031       levenbergIter= 1
iteration= 10    chi2= 101.995960      time= 2.2612e-05     cumTime= 0.0002749       edges= 100     schur= 0     lambda= 0.047354       levenbergIter= 1
iteration= 11    chi2= 101.995960      time= 2.2544e-05     cumTime= 0.000297444     edges= 100     schur= 0     lambda= 0.031569       levenbergIter= 1
iteration= 12    chi2= 101.995960      time= 3.2875e-05     cumTime= 0.000330319     edges= 100     schur= 0     lambda= 0.021046       levenbergIter= 1
iteration= 13    chi2= 101.995960      time= 2.9034e-05     cumTime= 0.000359353     edges= 100     schur= 0     lambda= 0.897971       levenbergIter= 4
iteration= 14    chi2= 101.995960      time= 2.6939e-05     cumTime= 0.000386292     edges= 100     schur= 0     lambda= 4.789179       levenbergIter= 3
iteration= 15    chi2= 101.995960      time= 2.2605e-05     cumTime= 0.000408897     edges= 100     schur= 0     lambda= 9.578358       levenbergIter= 1
solve time cost = 0.000643395 seconds.
estimated model:  1.24461 -2.06371  1.98099
```

2、绘制拟合曲线图像

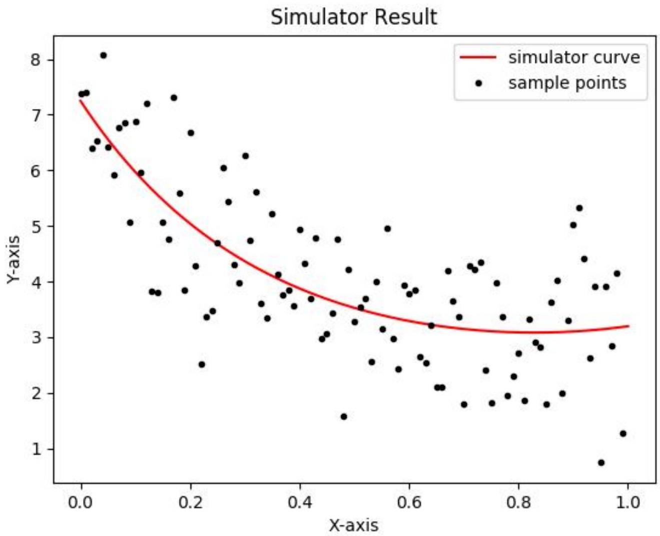
(1) 牛顿高斯法运行结果



(2) 使用 ceres 库运行结果



(3) **使用 g2o 库运行结果**



3、结果分析

从当前任务来看，三种方法拟合精度差异不大，但合理推测在拟合要求更高情况下使用库函数应该具有更高的通用性，高斯牛顿法本身也可能陷入局部最优解。从时间来看高斯牛顿法所需时间小于库函数的两种方法，核心在于通过损失可以提前退出优化过程。

总结来说使用库大多为了更为广泛的适用而进行了必要的冗余运算，可能使得运算速度较慢，同时基于图优化的 g2o 速度也明显慢于 ceres。