选题 4:手写 Bundle Adjustment

1、BAL 数据集介绍

16 22106 83718

0 0   -3.859900e+02 3.871200e+02

1 0   -3.844000e+01 4.921200e+02

16 为相机个数， 22106 为路标个数， 83718 为观测数据个数

第一行：0 为第 0 个相机， 0 为第 0 个路标， 后面 2 个为观测数据，像素坐标，83718 后面是相关参数，前面是相机参数有 9 维：-R(罗德里格斯向量 3 维)，t(3 维),f(相机焦距),k1(畸变参数),k2(畸变参数)。依次对应相机 0 – num_cameras，再后面是路标点的空间 3D 参数。

2、相机模型

P = R * X + t   (conversion from world to camera coordinates)//把世界坐标转换为相机坐标

p = -P / P.z   (perspective division)//相机坐标归一化处理

p' = f * r(p) * p   (conversion to pixel coordinates)//转换得到像素坐标

r(p) = 1.0 + k1 * ||p||^2 + k2 * ||p||^4.

3、残差和雅克比

3.1 残差

```
const Eigen::Vector3d ptc = camera_->getPose() * map_point_->getPosition();
Eigen::Vector2d p = -Eigen::Vector2d(ptc[0]/ptc[2],ptc[1]/ptc[2]);
double r_p = 1.0 + camera_->getK1() * p.squaredNorm() + camera_->getK2() * p.squaredNorm() * p.squaredNorm();
Eigen::Vector2d u(camera_->getFocus()*r_p*p[0], camera_->getFocus()*r_p*p[1]);
e = ob_z_ - u;
```

3.2 位姿的雅克比

```
const Eigen::Vector3d ptc = camera_->getPose() * map_point_->getPosition();
Eigen::Vector2d p = -Eigen::Vector2d(ptc[0]/ptc[2],ptc[1]/ptc[2]);
double f = camera_->getFocus();
double r_p = 1.0 + camera_->getK1() * p.squaredNorm() + camera_->getK2() * p.squaredNorm() * p.squaredNorm();
const double& x = ptc(0);
const double& y = ptc(1);
const double& z = ptc(2);

JT.setZero();
const double z2 = z*z;

JT(0,0) = f*r_p/z;
JT(0,1) = 0;
JT(0,2) = -f*r_p*x/z2;
JT(0,3) = -f*r_p*x*y/z2;
JT(0,4) = f*r_p+f*r_p*x*x/z2;
JT(0,5) = -f*r_p*y/z;

JT(1,0) = 0;
JT(1,1) = f*r_p/z;
JT(1,2) = -f*r_p*y/z2;
JT(1,3) = -f*r_p-f*r_p*y*y/z2;
JT(1,4) = f*r_p*x*y/z2;
JT(1,5) = f*r_p*x/z;
```

3.3 3D 点的雅克比

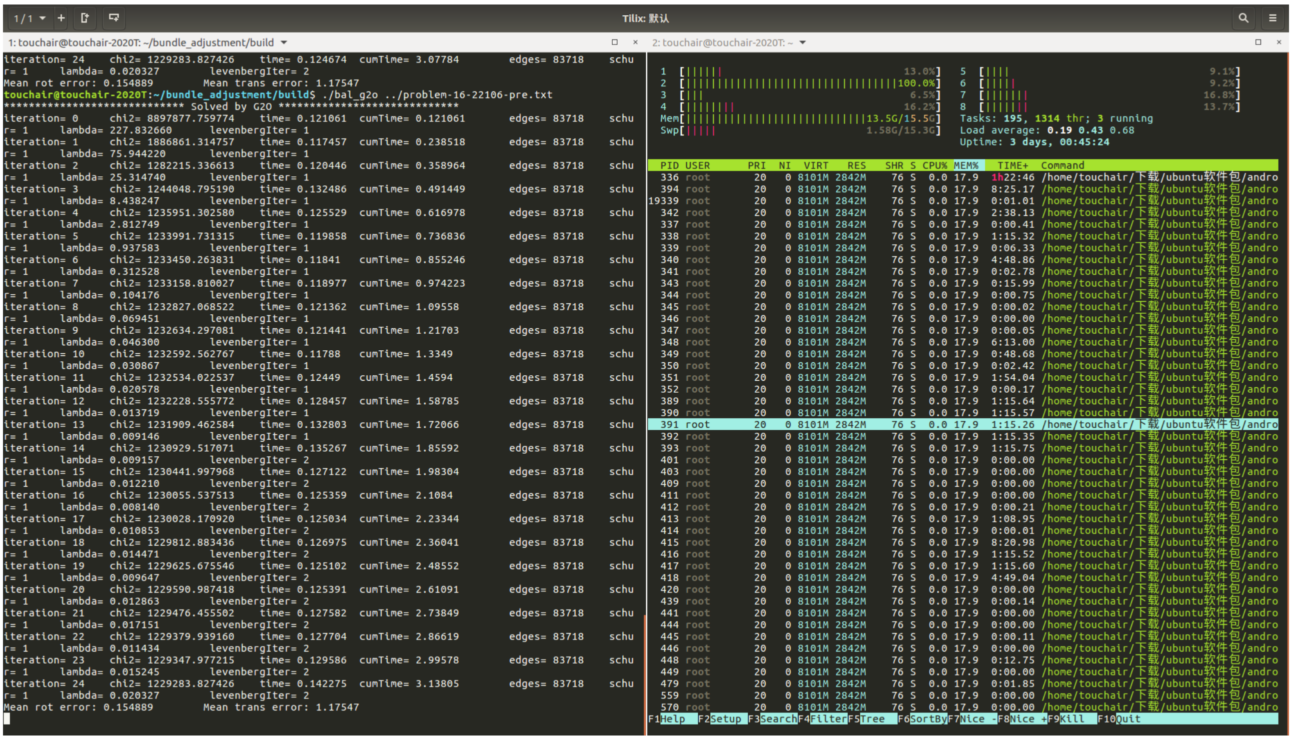
```
const Eigen::Vector3d ptc = camera_->getPose() * map_point_->getPosition();
Eigen::Vector2d p = -Eigen::Vector2d(ptc[0]/ptc[2],ptc[1]/ptc[2]);
double f = camera_->getFocus();
double r_p = 1.0 + camera_->getK1() * p.squaredNorm() + camera_->getK2() * p.squaredNorm() * p.squaredNorm();
const double& x = ptc(0);
const double& y = ptc(1);
const double& z = ptc(2);

JX.setZero();

Eigen::Matrix<double, 2, 3> Jtmp;
Jtmp.setZero();

Jtmp(0,0) = f*r_p / z;
Jtmp(0,1) = 0;
Jtmp(0,2) = -f*r_p*x/(z*z);
Jtmp(1,0) = 0;
Jtmp(1,1) = f*r_p/z;
Jtmp(1,2) = -f*r_p*y/(z*z);

JX = Jtmp * camera_->getPose().rotationMatrix();
```

4、与 g2o 对比

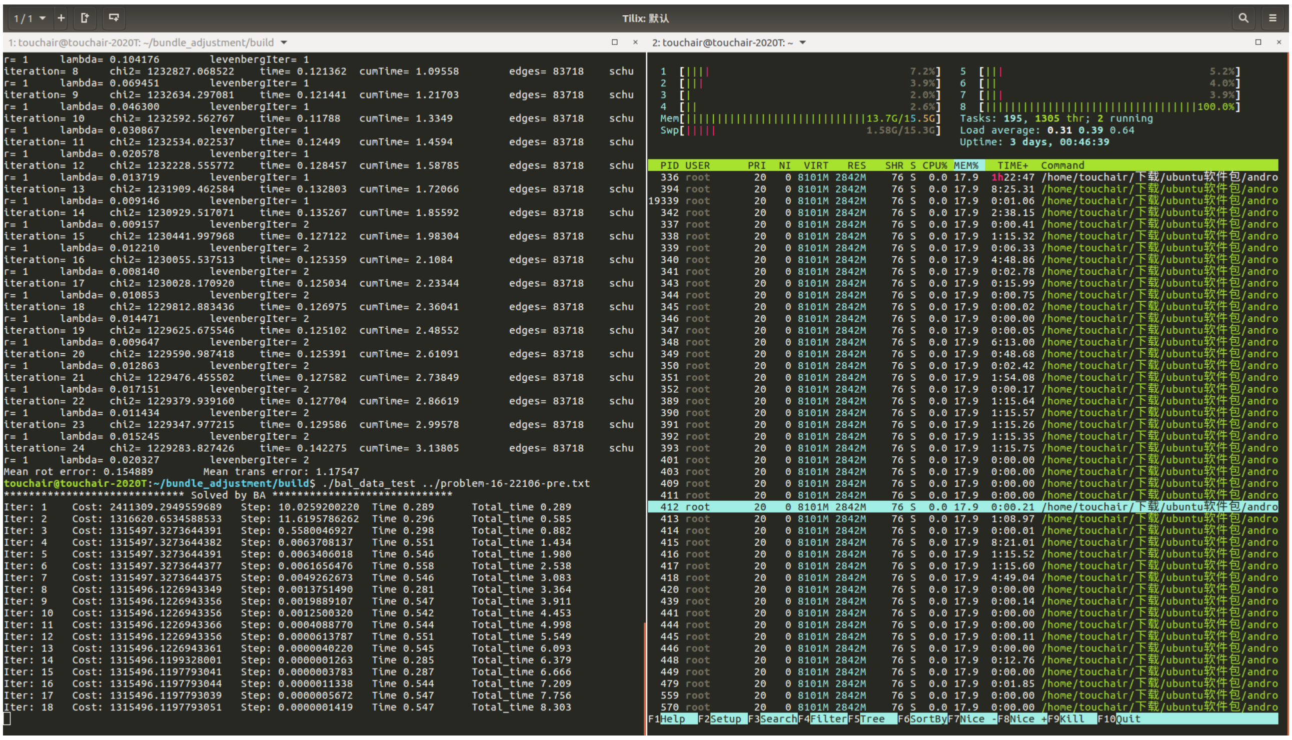
只对比了优化相机坐标，固定 3D 点，如果同时优化 3D 点，内存消耗过大，程序崩溃

```
touchair@touchair-2020T:~/bundle_adjustment/build$ ./bal_g2o ../problem-16-22106-pre.txt
****************************** Solved by G2O ******************************
iteration= 0     chi2= 8897877.759774    time= 0.128303  cumTime= 0.128303       edges= 83718    schur= 1        lambda= 227.832660      levenbergIter= 1
iteration= 1     chi2= 1886861.314757    time= 0.119045  cumTime= 0.247348       edges= 83718    schur= 1        lambda= 75.944220       levenbergIter= 1
iteration= 2     chi2= 1282215.336613    time= 0.121327  cumTime= 0.368675       edges= 83718    schur= 1        lambda= 25.314740       levenbergIter= 1
iteration= 3     chi2= 1244048.795190    time= 0.118969  cumTime= 0.487644       edges= 83718    schur= 1        lambda= 8.438247        levenbergIter= 1
iteration= 4     chi2= 1235951.302580    time= 0.118371  cumTime= 0.606015       edges= 83718    schur= 1        lambda= 2.812749        levenbergIter= 1
iteration= 5     chi2= 1233991.731315    time= 0.122274  cumTime= 0.728289       edges= 83718    schur= 1        lambda= 0.937583        levenbergIter= 1
iteration= 6     chi2= 1233450.263831    time= 0.126224  cumTime= 0.854513       edges= 83718    schur= 1        lambda= 0.312528        levenbergIter= 1
iteration= 7     chi2= 1233158.810027    time= 0.121271  cumTime= 0.975784       edges= 83718    schur= 1        lambda= 0.104176        levenbergIter= 1
iteration= 8     chi2= 1232827.068522    time= 0.121051  cumTime= 1.09684        edges= 83718    schur= 1        lambda= 0.069451        levenbergIter= 1
iteration= 9     chi2= 1232634.297081    time= 0.116734  cumTime= 1.21357        edges= 83718    schur= 1        lambda= 0.046300        levenbergIter= 1
iteration= 10    chi2= 1232592.562767    time= 0.115511  cumTime= 1.32908        edges= 83718    schur= 1        lambda= 0.030867        levenbergIter= 1
iteration= 11    chi2= 1232534.022537    time= 0.116121  cumTime= 1.4452         edges= 83718    schur= 1        lambda= 0.020578        levenbergIter= 1
iteration= 12    chi2= 1232228.555772    time= 0.117457  cumTime= 1.56266        edges= 83718    schur= 1        lambda= 0.013719        levenbergIter= 1
iteration= 13    chi2= 1231909.462584    time= 0.118091  cumTime= 1.68075        edges= 83718    schur= 1        lambda= 0.009146        levenbergIter= 1
iteration= 14    chi2= 1230929.517071    time= 0.124128  cumTime= 1.80488        edges= 83718    schur= 1        lambda= 0.009157        levenbergIter= 2
iteration= 15    chi2= 1230441.997968    time= 0.126282  cumTime= 1.93116        edges= 83718    schur= 1        lambda= 0.012210        levenbergIter= 2
iteration= 16    chi2= 1230055.537513    time= 0.123687  cumTime= 2.05485        edges= 83718    schur= 1        lambda= 0.008140        levenbergIter= 2
iteration= 17    chi2= 1230028.170920    time= 0.125612  cumTime= 2.18046        edges= 83718    schur= 1        lambda= 0.010853        levenbergIter= 2
iteration= 18    chi2= 1229812.883436    time= 0.12563   cumTime= 2.30609        edges= 83718    schur= 1        lambda= 0.014471        levenbergIter= 2
iteration= 19    chi2= 1229625.675546    time= 0.123964  cumTime= 2.43005        edges= 83718    schur= 1        lambda= 0.009647        levenbergIter= 2
iteration= 20    chi2= 1229590.987418    time= 0.123795  cumTime= 2.55385        edges= 83718    schur= 1        lambda= 0.012863        levenbergIter= 2
iteration= 21    chi2= 1229476.455502    time= 0.134155  cumTime= 2.688  edges= 83718    schur= 1        lambda= 0.017151        levenbergIter= 2
iteration= 22    chi2= 1229379.939160    time= 0.127164  cumTime= 2.81517        edges= 83718    schur= 1        lambda= 0.011434        levenbergIter= 2
iteration= 23    chi2= 1229347.977215    time= 0.125134  cumTime= 2.9403         edges= 83718    schur= 1        lambda= 0.015245        levenbergIter= 2
iteration= 24    chi2= 1229283.827426    time= 0.123595  cumTime= 3.06389        edges= 83718    schur= 1        lambda= 0.020327        levenbergIter= 2
Mean rot error: 0.154889        Mean trans error: 1.17547
touchair@touchair-2020T:~/bundle_adjustment/build$ ./bal_data_test ../problem-16-22106-pre.txt
****************************** Solved by BA ******************************
Iter: 1    Cost: 2411309.2949559689   Step: 10.0259200220  Time 0.263      Total_time 0.263
Iter: 2    Cost: 1316620.6534588533   Step: 11.6195786262  Time 0.270      Total_time 0.533
Iter: 3    Cost: 1315497.3273644391   Step: 0.5580046927   Time 0.273      Total_time 0.806
Iter: 4    Cost: 1315497.3273644382   Step: 0.0063708137   Time 0.536      Total_time 1.342
Iter: 5    Cost: 1315497.3273644391   Step: 0.0063406018   Time 0.530      Total_time 1.872
Iter: 6    Cost: 1315497.3273644377   Step: 0.0061656476   Time 0.536      Total_time 2.407
Iter: 7    Cost: 1315497.3273644375   Step: 0.0049262673   Time 0.534      Total_time 2.941
Iter: 8    Cost: 1315496.1226943349   Step: 0.0013751490   Time 0.273      Total_time 3.214
Iter: 9    Cost: 1315496.1226943356   Step: 0.0019889107   Time 0.529      Total_time 3.743
Iter: 10   Cost: 1315496.1226943356   Step: 0.0012500320   Time 0.535      Total_time 4.278
Iter: 11   Cost: 1315496.1226943366   Step: 0.0004088770   Time 0.534      Total_time 4.812
Iter: 12   Cost: 1315496.1226943356   Step: 0.0000613787   Time 0.534      Total_time 5.346
Iter: 13   Cost: 1315496.1226943361   Step: 0.0000040220   Time 0.533      Total_time 5.879
Iter: 14   Cost: 1315496.1199328001   Step: 0.0000001263   Time 0.270      Total_time 6.149
Iter: 15   Cost: 1315496.1197793041   Step: 0.0000003783   Time 0.270      Total_time 6.419
Iter: 16   Cost: 1315496.1197793044   Step: 0.0000011338   Time 0.533      Total_time 6.952
Iter: 17   Cost: 1315496.1197793039   Step: 0.0000005672   Time 0.532      Total_time 7.484
Iter: 18   Cost: 1315496.1197793051   Step: 0.0000001419   Time 0.535      Total_time 8.019
Iter: 19   Cost: 1315496.1197790757   Step: 0.0000000177   Time 0.271      Total_time 8.290
Iter: 20   Cost: 1315496.1197790757   Step: 0.0000000128   Time 0.541      Total_time 8.832
Iter: 21   Cost: 1315496.1197790750   Step: 0.0000000064   Time 0.534      Total_time 9.366
Iter: 22   Cost: 1315496.1197790760   Step: 0.0000000016   Time 0.536      Total_time 9.902
Iter: 23   Cost: 1315496.1197790417   Step: 0.0000000002   Time 0.270      Total_time 10.172
Iter: 24   Cost: 1315496.1197790422   Step: 0.0000000006   Time 0.532      Total_time 10.704
Iter: 25   Cost: 1315496.1197790422   Step: 0.0000000003   Time 0.533      Total_time 11.237
Mean rot error: 0.155   Mean trans error: 2.041
```

G2O



手写 BA



上面是 G2O 的测试结果，下面是手写 BA 的测试结果，可以看出精度方面，手写 BA 和 G2O 计算的相机位姿旋转误差基本一致，平移误差手写 BA 略高于 G2O，单次迭代时间和总迭代时间手写 BA 均高于 G2O，CPU 占用百分比手写 BA 略低于 G2O