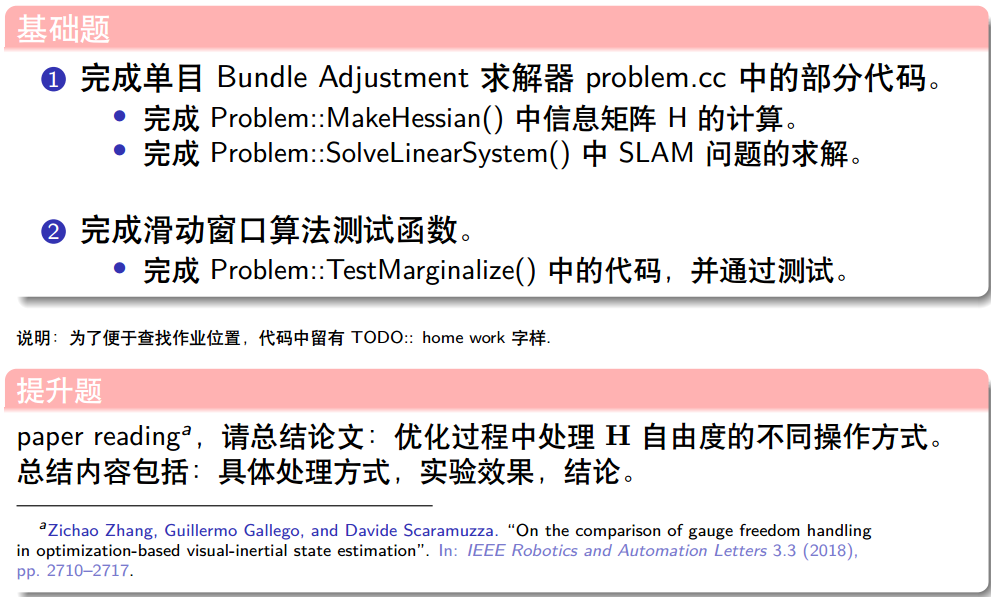
**深蓝学院VIO课程第五课作业**



**1. 完成problem.cc中的部分代码**

参考第三课的作业、第四课与本课的PPT资料和公式，补全problem.cc中TODO部分的代码如下所示，详情可见附件problem.cc文件。程序运行输出如图1所示。

|  |  |
| --- | --- |
| 279  …  314  315  316  317  318  319  320  321  …  348  …  366  367  368  369  370  371  …  381  382  383  384  …  398  399  400  401  402 | void Problem::MakeHessian() {  …  // 所有的信息矩阵叠加起来  // TODO:: home work. 完成 H index 的填写.  H.block(index\_i, index\_j, dim\_i, dim\_j).noalias() += hessian;  if (j != i) {  // 对称的下三角  // TODO:: home work. 完成 H index 的填写.  H.block(index\_j, index\_i, dim\_j, dim\_i).noalias() += hessian.transpose();  }  …  void Problem::SolveLinearSystem() {  …  // TODO:: home work. 完成矩阵块取值，Hmm，Hpm，Hmp，bpp，bmm  MatXX Hmm = Hessian\_.block(reserve\_size, reserve\_size, marg\_size, marg\_size);  MatXX Hpm = Hessian\_.block(0, reserve\_size, reserve\_size, marg\_size);  MatXX Hmp = Hessian\_.block(reserve\_size, 0, marg\_size, reserve\_size);  VecX bpp = b\_.segment(0, reserve\_size);  VecX bmm = b\_.segment(reserve\_size, marg\_size);  …  // TODO:: home work. 完成舒尔补 Hpp, bpp 代码  MatXX tempH = Hpm \* Hmm\_inv;  H\_pp\_schur\_ = Hessian\_.block(0,0,reserve\_size,reserve\_size) - tempH\*Hmp;  b\_pp\_schur\_ = bpp - tempH \* bmm;  …  // TODO:: home work. step3: solve landmark  VecX delta\_x\_ll(marg\_size);  delta\_x\_ll = Hmm\_inv \* (bmm - Hmp \* delta\_x\_pp);  delta\_x\_.tail(marg\_size) = delta\_x\_ll;  … |

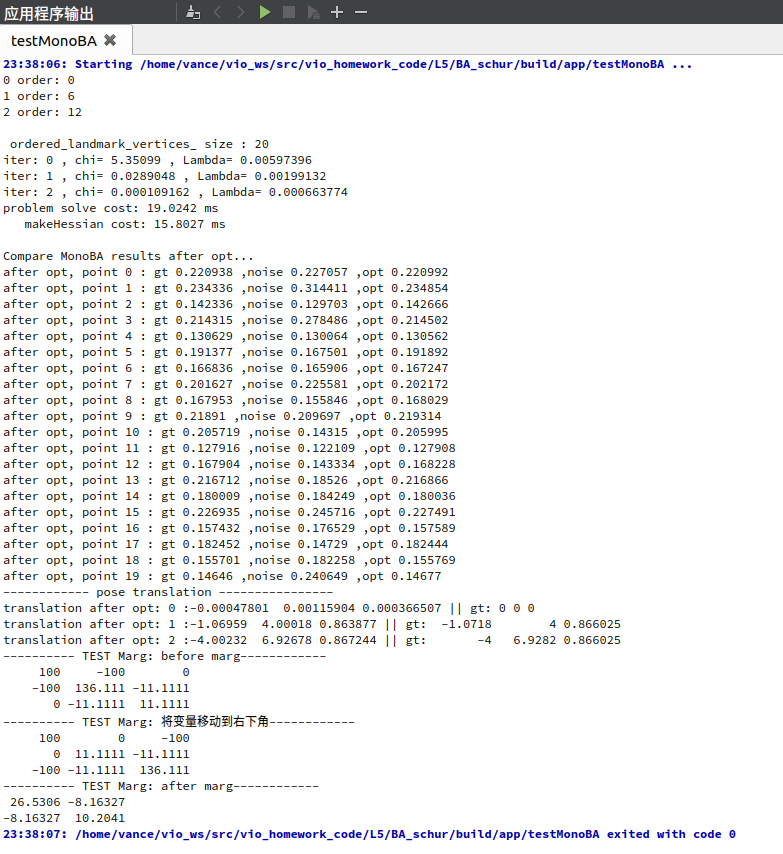


图1 testMonoBA程序运行结果截图

**2. 完成滑动窗口算法测试函数**

补充的代码如下所示，详情可见附件problem.cc文件。程序运行输出同样如上页图1所示。

|  |  |
| --- | --- |
| 554  …  572  573  574  575  576  577  578  600  601  602  603  604  605  606  … | void Problem::TestMarginalize() {  …  // TODO:: home work. 将变量移动到右下角  /// 准备工作： move the marg pose to the Hmm bottown right  // 将 row i 移动矩阵最下面  Eigen::MatrixXd temp\_rows = H\_marg.block(idx, 0, dim, reserve\_size);  Eigen::MatrixXd temp\_botRows = H\_marg.block(idx + dim, 0, reserve\_size - idx - dim, reserve\_size);  H\_marg.block(idx, 0, reserve\_size - idx - dim, reserve\_size) = temp\_botRows;  H\_marg.block(reserve\_size - dim, 0, dim, reserve\_size) = temp\_rows;  …  // TODO:: home work. 完成舒尔补操作  Eigen::MatrixXd Arm = H\_marg.block(0,n2,n2,m2);  Eigen::MatrixXd Amr = H\_marg.block(n2,0,m2,n2);  Eigen::MatrixXd Arr = H\_marg.block(0,0,n2,n2);  Eigen::MatrixXd tempB = Arm \* Amm\_inv;  Eigen::MatrixXd H\_prior = Arr - tempB \* Amr;  … |

**3. 论文总结**

VIO有四个不可观的自由度，优化的时候需要特别处理，这篇论文就这个问题进行了详细讨论。通常有三种方法：①是固定这四个自由度，②是给这四个自由度加先验，③是任意优化这四个自由度再reset。三种方法的对比如表一所示，这里先给出结论：

* 三种方法的精度相当；
* Gauge Prior法需要选择一个合适的先验权重，以避免计算开销上升；
* Gauge Prior法在合适权重的情况下，精度和计算效率可以跟Fixed Gauge法相当；
* Free Gauge法计算速度比另外两种方法略快，因为迭代次数更少；
* Free Gauge法不需要对旋转参数做特殊处理；

表1 VIO协方差矩阵对四个不可观自由度的处理方法对比

综上来看，Free Gauge方法应该是最好的。但是Free Gauge方法不足之处在于他没有一个参考系，信息矩阵的逆得到的协方差没有太多的意义，可以参考引文[14]中的方法解决。将Free Gauge法的协方差通过线性变换成Fixed Gauge的协方差形式，这就解决了这个方法的不足。

|  |  |  |
| --- | --- | --- |
| **方法名称** | **具体处理方式** | **实验效果对比** |
| Fixed Gauge | 在小参数空间中优化，使其状态量均可观，则H可逆。  优化时固定第一个相机的位置和yaw角。 | * 变量维度最少，单次迭代耗时最短 * 精度相当 |
| Gauge Prior | 增加目标函数额外的误差项，满足确定性的约束，使H可逆，目标函数变为：    其中 | * 单次迭代耗时中等 * 精度相当 |
| Free Gauge | 用奇异H的伪逆提供额外约束获得唯一解。  在优化过程中让参数向量自由变化。 | * 单次迭代耗时最长 * 迭代次数最少 * 总耗时最快 * 精度相当 |