深蓝学院
shenlanxueyuan.com

第五章：后端优化实践 逐行手写求解器作业分享

主讲人 梁章科

纲要

深蓝学院
shenlanxueyuan.com

- 第一部分：完成单目BA求解器代码
- 第二部分：完成滑动窗口算法测试函数
- 第二部分：论文总结
- 第三部分：添加prior约束

完成单目BA求解器代码

深蓝学院 shenlanxueyuan.com

- Problem::MakeHessian() 中信息矩阵 H 的计算 参考代码

```
        assert(v_j->OrderingId() != -1);
        MatXX hessian = JtW * jacobian_j;
        // 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();
        }
    }
    b.segment(index_i, dim_i).noalias() -= JtW * edge.second->Residual();
}
```

完成单目BA求解器代码

深蓝学院 shenlanxueyuan.com

- Problem::SolveLinearSystem()参考代码

```
// 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( startRow: 0,reserve_size, reserve_size, marg_size);
 MatXX Hmp = Hessian_.block(reserve_size, startCol: 0, marg_size, reserve_size);
 VecX bpp = b_.segment( start: 0,reserve_size);
 VecX bmm = b_.segment(reserve_size,marg_size);

// Hmm 是对角线矩阵，它的求逆可以直接为对角线块分别求逆，如果是逆深度，对角线块为1维的，则直接为对角线的倒数，这里可以加速
MatXX Hmm_inv(MatXX::Zero(marg_size, marg_size));
for (auto landmarkVertex : idx_landmark_vertices_) {
    int idx = landmarkVertex.second->OrderingId() - reserve_size;
    int size = landmarkVertex.second->LocalDimension();
    Hmm_inv.block(idx, idx, size, size) = Hmm.block(idx, idx, size, size).inverse();
}

// TODO:: home work. 完成舒尔补 Hpp, bpp 代码
MatXX tempH = Hpm * Hmm_inv;
 H_pp_schur_ = Hessian_.block( startRow: 0, startCol: 0,reserve_size,reserve_size) - tempH * Hmp;
 b_pp_schur_ = bpp - tempH * bmm;
```

完成单目BA求解器代码

深蓝学院 shenlanxueyuan.com

- Problem::SolveLinearSystem()参考代码

```
// step2: solve Hpp * delta_x = bpp
VecX delta_x_pp(VecX::Zero(reserve_size));
// PCG Solver
for (ulong i = 0; i < ordering_poses_; ++i) {
    H_pp_schur_(i, i) += currentLambda_;
}

int n = H_pp_schur_.rows() * 2;                        // 迭代次数
delta_x_pp = PCGSolver(H_pp_schur_, b_pp_schur_, n);  // 哈哈，小规模问题，搞 pcg 花里胡哨
delta_x_.head(reserve_size) = delta_x_pp;
//        std::cout << delta_x_pp.transpose() << std::endl;

// 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;
```

完成单目BA求解器代码

深蓝学院 shenlanxueyuan.com

- 结果分析

第一帧的位姿有漂移，导致后面两帧的位姿也随着漂移

```
------------ pose translation ----------------
translation after opt: 0 :-0.00047801  0.00115904 0.000366507 || gt: 0 0 0
translation after opt: 1 :-1.06959  4.00018 0.863877 || gt:  -1.0718        4 0.866025
translation after opt: 2 :-4.00232  6.92678 0.867244 || gt:       -4   6.9282 0.866025
```

当把第一帧位姿固定为（0,0,0）时，后面的两帧位姿不再发生漂移，且与真实值接近

```
------------ pose translation ----------------
translation after opt: 0 :0 0 0 || gt: 0 0 0
translation after opt: 1 : -1.0718        4 0.866025 || gt:  -1.0718        4 0.866025
translation after opt: 2 :-3.99917  6.92852 0.859878 || gt:       -4   6.9282 0.866025
```

纲要

- 第一部分：完成单目BA求解器代码
- **第二部分：完成滑动窗口算法测试函数**
- 第二部分：论文总结
- 第三部分：添加prior约束

滑动窗口算法测试函数

深蓝学院 shenlanxueyuan.com

- Problem::TestMarginalize()参考代码

```
// 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, startCol: 0, dim, reserve_size);
Eigen::MatrixXd temp_botRows = H_marg.block( startRow: idx + dim, startCol: 0, blockRows: reserve_size - idx - dim, reserve_size);
H_marg.block(idx, startCol: 0, blockRows: reserve_size - idx - dim, reserve_size) = temp_botRows;
H_marg.block( startRow: idx + dim, startCol: 0,dim, reserve_size) = temp_rows;

// 将 col i 移动矩阵最右边
Eigen::MatrixXd temp_cols = H_marg.block( startRow: 0, idx, reserve_size, dim);
Eigen::MatrixXd temp_rightCols = H_marg.block( startRow: 0, startCol: idx + dim, reserve_size, blockCols: reserve_size - idx - dim);
H_marg.block( startRow: 0, idx, reserve_size, blockCols: reserve_size - idx - dim) = temp_rightCols;
H_marg.block( startRow: 0, startCol: reserve_size - dim, reserve_size, dim) = temp_cols;

std::cout << "---------- TEST Marg: 将变量移动到右下角------------"<< std::endl;
std::cout<< H_marg <<std::endl;

/// 开始 marg ： schur
double eps = 1e-8;
int m2 = dim;
int n2 = reserve_size - dim;   // 剩余变量的维度
Eigen::MatrixXd Amm = 0.5 * (H_marg.block(n2, n2, m2, m2) + H_marg.block(n2, n2, m2, m2).transpose());

Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> saes(Amm);
Eigen::MatrixXd Amm_inv = saes.eigenvectors() * Eigen::VectorXd(
        x: (saes.eigenvalues().array() > eps).select(saes.eigenvalues().array().inverse(), elseScalar: 0)).asDiagonal() *
                          saes.eigenvectors().transpose();

// TODO:: home work. 完成舒尔补操作
Eigen::MatrixXd Arm = H_marg.block( startRow: 0,n2,n2,m2);
Eigen::MatrixXd Amr = H_marg.block(n2, startCol: 0,m2,n2);
Eigen::MatrixXd Arr = H_marg.block( startRow: 0, startCol: 0,n2,n2);

Eigen::MatrixXd tempB = Arm * Amm_inv;
Eigen::MatrixXd H_prior = Arr - tempB * Amr;
```

滑动窗口算法测试函数

深蓝学院 shenlanxueyuan.com

- 结果分析

```
---------- TEST Marg: before marg------------
     100      -100         0
    -100   136.111 -11.1111
       0 -11.1111  11.1111
---------- TEST Marg: 将变量移动到右下角------------
     100         0      -100
       0  11.1111 -11.1111
    -100 -11.1111   136.111
---------- TEST Marg: after marg------------
 26.5306 -8.16327
-8.16327  10.2041
```

○ 无关变量

▭ Marg后无关变量变成有关变量

纲要

- 第一部分：完成单目BA求解器代码
- 第二部分：完成滑动窗口算法测试函数
- **第二部分：论文总结**
- 第三部分：添加prior约束

论文总结

深蓝学院 shenlanxueyuan.com

- 处理H自由度的三种方式

1. Free gauge approach
2. Gauge prior approach
3. Gauge fixation approach

其余内容请参考助教的作业讲解PPT，这里不再赘述

solver 求解中的小疑问

1. 上节课说到信息矩阵 $\mathbf{H}$ 不满秩，那求解的时候如何操作呢？
   - 使用 LM 算法，加阻尼因子使得系统满秩，可求解，但是求得的结果可能会往零空间变化。
   - 添加先验约束，增加系统的可观性。比如 g2o tutorial 中对第一个 pose 的信息矩阵加上单位阵 $\mathbf{H}_{[11]} += \mathbf{I}$.
2. orbslam，svo 等等求 mono BA 问题时，fix 一个相机 pose 和一个特征点，或者 fix 两个相机 pose，也是为了限定优化值不乱飘。那代码如何实现 fix 呢？
   - 添加超强先验，使得对应的信息矩阵巨大（如，10^{15}），就能使得 $\Delta x = 0$;
   - 设定对应雅克比矩阵为 0，意味着残差等于 0. 求解方程为 $(\mathbf{0} + \lambda \mathbf{I})\, \Delta \mathbf{x} = \mathbf{0}$，只能 $\Delta x = 0$。

纲要

- 第一部分：完成单目BA求解器代码
- 第二部分：完成滑动窗口算法测试函数
- 第二部分：论文总结
- **第三部分：添加prior约束**

添加prior约束

- 第一帧和第二帧添加prior 约束 参考代码

```
// 添加先验prior顶点
double weight = 0;      //先验值
int prior_number = 2;  //前两帧pose添加先验约束
for (int n = 0; n < prior_number; ++n){
    shared_ptr<EdgeSE3Prior> edge_prior( p: new EdgeSE3Prior(cameras[n].twc, cameras[n].qwc));
    std::vector<std::shared_ptr<Vertex> > edge_prior_vertex;
    edge_prior_vertex.push_back(vertexCams_vec[n]);
    edge_prior->SetVertex(edge_prior_vertex);
    edge_prior->SetInformation( information: edge_prior->Information()*weight);
    problem.AddEdge( edge: edge_prior);
}
```

添加prior约束

深蓝学院 shenlanxueyuan.com

- BA求解收敛精度和速度，参考论文第四页IV. 部分对收敛精度和计算成本的处理方式

对齐矩阵： $T_{align} = T_{gt_0}^T \cdot T_{opt_0}$

对齐后的优化位姿： $T_{opt_i} = T_{align} \cdot T_{opt_i}$ ， $i = 1, 2, \ldots$

平移量误差采用欧拉距离公式： $transdiff_i = (t_{gt_i} - t_{opt_i})^2$

优化旋转矩阵与真值旋转矩阵的相对旋转矩阵： $R_{relative} = R_{gt_i}^T \cdot R_{opt_i}$

旋转量误差即是相对旋转矩阵 $R_{relative}$ 对应的旋转向量欧拉距离 $rotdiff_i = V_i^2$

采用 $RMSE$ （均方根误差）计算收敛精度： $trans - RMSE = \sqrt{\frac{\sum_1^n transdiff_i}{n}}$

$$rot - RMSE = \sqrt{\frac{\sum_1^n rotdiff_i}{n}}$$

添加prior约束

深蓝学院
shenlanxueyuan.com

- 参考代码

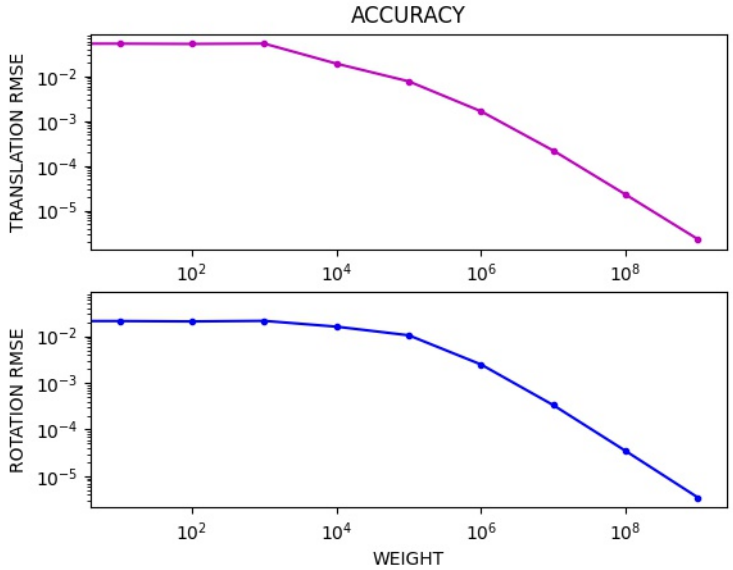
```
double translation_rmse; //平移的RMSE
double rotation_rmse;    //旋转矩阵的RMSE
//优化后得到的第一帧位姿
double qx = vertexCams_vec[0]->Parameters()( index: 3);
double qy = vertexCams_vec[0]->Parameters()( index: 4);
double qz = vertexCams_vec[0]->Parameters()( index: 5);
double qw = vertexCams_vec[0]->Parameters()( index: 6);
Sophus::SE3 opt_frame1( quaternion: Qd(qw, qx, qy, qz),  translation: vertexCams_vec[0]->Parameters().head( n: 3));
//真值的第一帧位姿
Sophus::SE3 gt_frame1(cameras[0].qwc, cameras[0].twc);
//把优化后第一帧的位姿与真值第一帧位姿对齐，后续所有的优化后的位姿都做对应的变换，以便计算RMSE
Sophus::SE3 opt_to_gt(gt_frame1.inverse()*opt_frame1);
for (int i=1; i<vertexCams_vec.size(); ++i)
{
    qx = vertexCams_vec[i]->Parameters()( index: 3);
    qy = vertexCams_vec[i]->Parameters()( index: 4);
    qz = vertexCams_vec[i]->Parameters()( index: 5);
    qw = vertexCams_vec[i]->Parameters()( index: 6);
    Sophus::SE3 frame( quaternion: Qd(qw, qx, qy, qz),  translation: vertexCams_vec[i]->Parameters().head( n: 3));
    frame = opt_to_gt * frame;
    //采用欧拉距离公式计算平移量的误差
    double translation_diff = (frame.translation() - cameras[i].twc).norm();
    translation_rmse += translation_diff;
    //先求优化后的旋转矩阵与真值旋转矩阵的相对旋转矩阵，相对旋转矩阵的对应的旋转向量就是旋转矩阵的误差
    double rotation_diff = Sophus::SO3::log(Sophus::SO3(cameras[i].Rwc.inverse() * frame.rotationMatrix())).norm();
    rotation_rmse += rotation_diff;
}
//计算平移量与旋转量的RMSE
translation_rmse = std::sqrt( x: translation_rmse / (vertexCams_vec.size() - 1));
rotation_rmse = std::sqrt( x: rotation_rmse / (vertexCams_vec.size() -1));
std::cout<<"translation RMSE: "<<translation_rmse<<std::endl;
std::cout<<"rotation RMSE: "<<rotation_rmse<<std::endl;
```

添加prior约束

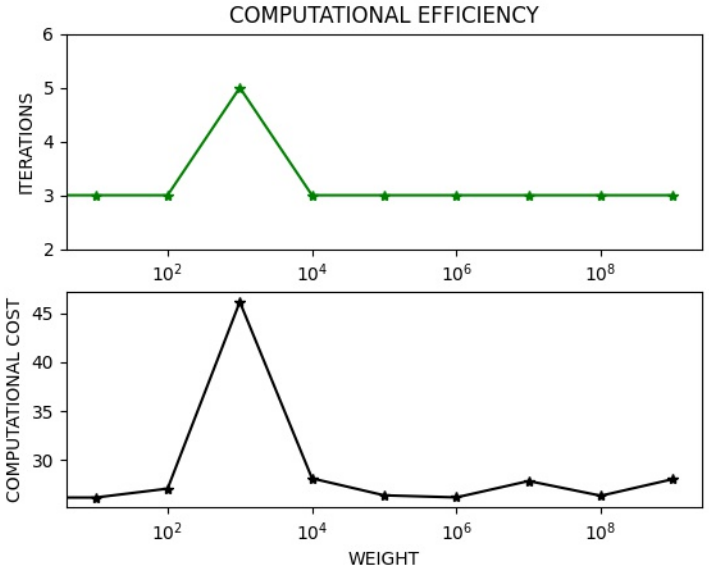
深蓝学院 shenlanxueyuan.com

- 结果分析

随着先验权重的增大，旋转量与平移量的均方根误差是变小的



计算时间在权重在10的3次方高点，其余基本相差无几。



在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening