手写VIO第三章作业讲解

主讲人 海滩游侠

作业总览

深蓝学院 shenlanxueyuan.com

基础题

1. 完成单目 Bundle Adjustment (BA) 求解器 problem.cc 中的部分代码。
   - 完成 Problem::MakeHessian() 中信息矩阵 H 的计算。
   - 完成 Problem::SolveLinearSystem() 中 SLAM 问题的求解。
2. 完成滑动窗口算法测试函数。
   - 完成 Problem::TestMarginalize() 中的代码，并通过测试。

说明：为了便于查找作业位置，代码中留有 TODO:: home work 字样.

提升题

- 请总结论文[a]：优化过程中处理 H 自由度的不同操作方式。内容包括：具体处理方式，实验效果，结论。(加分题，评选良好)
- 在代码中给第一帧和第二帧添加 prior 约束，并比较为 prior 设定不同权重时，BA 求解收敛精度和速度。(加分题，评选优秀)

1.1 MakeHession()填空

深蓝学院 shenlanxueyuan.com

和第四章作业类似，拼接矩阵

$$\mathrm{Hession}(i,j) = J_i^{\,T} w J_j$$

$$\mathrm{Hession}(j,i) = \mathrm{Hession}(i,j)^T$$

```cpp
        MatXX JtW = jacobian_i.transpose() * edge.second->Information();
        for (size_t j = i; j < verticies.size(); ++j) {
            auto v_j = verticies[j];

            if (v_j->IsFixed()) continue;

            auto jacobian_j = jacobians[j];
            ulong index_j = v_j->OrderingId();
            ulong dim_j = v_j->LocalDimension();

            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(?,?, ?, ?).noalias() += hessian;
            if (j != i) {
                // 对称的下三角
        // TODO:: home work. 完成 H index 的填写.
                // H.block(?,?, ?, ?).noalias() += hessian.transpose();
            }
        }
        b.segment(index_i, dim_i).noalias() -= JtW * edge.second->Residual();
```

1.2 SolveLinearSystem() 求解

理论基础：利用矩阵的稀疏性，通过舒尔补求解线性方程

$$\begin{bmatrix} \mathbf{H}_{pp} & \mathbf{H}_{pl} \\ \mathbf{H}_{lp} & \mathbf{H}_{ll} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}_p^* \\ \Delta \mathbf{x}_l^* \end{bmatrix} = \begin{bmatrix} -\mathbf{b}_p \\ -\mathbf{b}_l \end{bmatrix} \tag{4}$$

可以利用舒尔补操作，使上式中信息矩阵变成下三角，从而得到：

$$\left(\mathbf{H}_{pp} - \mathbf{H}_{pl}\mathbf{H}_{ll}^{-1}\mathbf{H}_{pl}^{\top}\right) \Delta \mathbf{x}_p^* = -\mathbf{b}_p + \mathbf{H}_{pl}\mathbf{H}_{ll}^{-1}\mathbf{b}_l \tag{5}$$

求得 $\Delta \mathbf{x}_p^*$ 后，再计算 $\Delta \mathbf{x}_l^*$：

$$\mathbf{H}_{ll}\Delta \mathbf{x}_l^* = -\mathbf{b}_l - \mathbf{H}_{pl}^{\top}\Delta \mathbf{x}_p^* \tag{6}$$

1.2 SolveLinearSystem() 求解

深蓝学院
shenlanxueyuan.com

1、Schur三角化

$$\begin{bmatrix} H_{pp} & H_{pm} \\ H_{mp} & H_{mm} \end{bmatrix} \begin{bmatrix} \Delta x_p \\ \Delta x_m \end{bmatrix} = \begin{bmatrix} b_p \\ b_m \end{bmatrix}$$

两边同时左乘 $\begin{bmatrix} I & -H_{pm}H_{pp}^{-1} \\ 0 & I \end{bmatrix}$

得到

$$\begin{bmatrix} H_{pp} - H_{pm}H_{pp}^{-1}H_{mp} & 0 \\ H_{mp} & H_{mm} \end{bmatrix} \begin{bmatrix} \Delta x_p \\ \Delta x_m \end{bmatrix} = \begin{bmatrix} b_p - H_{pm}H_{pp}^{-1}b_m \\ b_m \end{bmatrix}$$

```
// SLAM 问题采用舒尔补的计算方式
// step1: schur marginalization --> Hpp, bpp
int reserve_size = ordering_poses_;
int marg_size = ordering_landmarks_;

// TODO:: home work. 完成矩阵块取值，Hmm, Hpm, Hmp, bpp, bmm
// MatXX Hmm = Hessian_.block(?,?, ?, ?);
// MatXX Hpm = Hessian_.block(?,?, ?, ?);
// MatXX Hmp = Hessian_.block(?,?, ?, ?);
// VecX bpp = b_.segment(?,?);
// VecX bmm = b_.segment(?,?);

// 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
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(?,?,?,?) - tempH * Hmp;
// b_pp_schur_ = bpp - ? * ?;
```

1.2 SolveLinearSystem() 求解

深蓝学院
shenlanxueyuan.com

2、求解方程

先解

$$(H_{pp} - H_{pm} H_{pp}^{-1} H_{mp}) \Delta x_p = b_p - H_{pm} H_{pp}^{-1} b_m$$

再解

$$H_{mm} \Delta x_m = b_m - H_{mp} \Delta x_p$$

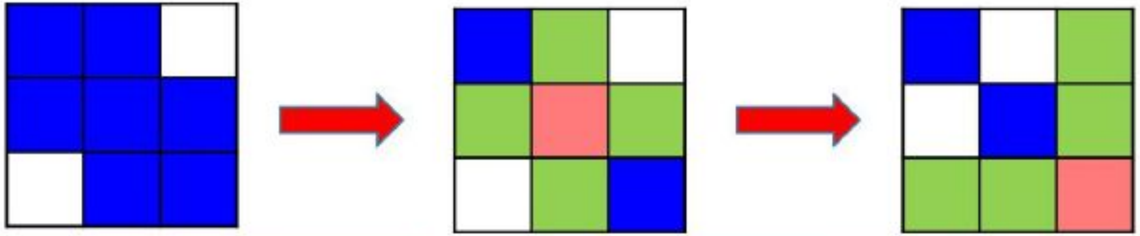
```
// step2: solve Hpp * delta_x = bpp
VecX delta_x_pp(VecX::Zero(reserve_size));
// PCG Solver
for (ulong i = 0; i < ordering_poses_; ++i) {
    H_pp_schur_(i, i) += currentLambda_;
}

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 = ???;
delta_x_.tail(marg_size) = delta_x_ll;
```

优化结果接近真值，但是由于单目系统具有7个状态不可观，优化结果会在零空间内漂移，可以fix前两帧位姿，代码在TestMonoBA中已经写好。

2 TestMarginalize() 求解

第一步，将被边缘化的变量移到右下角：

```cpp
// 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(?,?,?,?) = temp_botRows;
// H_marg.block(?,?,?,?) = temp_rows;
```



```
---------- 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
```

2 TestMarginalize() 求解

深蓝学院
shenlanxueyuan.com

第二步，舒尔补：

```
    // TODO:: home work. 完成舒尔补操作
    //Eigen::MatrixXd Arm = H_marg.block(?,?,?,?);
    //Eigen::MatrixXd Amr = H_marg.block(?,?,?,?);
    //Eigen::MatrixXd Arr = H_marg.block(?,?,?,?);

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

```
---------- TEST Marg: after marg------------
 26.5306 -8.16327
-8.16327  10.2041
```

提升题 论文阅读

深蓝学院
shenlanxueyuan.com

Zhang Z, Gallego G, Scaramuzza D. On the comparison of gauge freedom handling in optimization-based visual-inertial state estimation[J]. IEEE Robotics and Automation Letters, 2018, 3(3): 2710-2717.

- 目的：讨论使用最小二乘法解决VIO问题时处理不可观自由度(Gauge Freedom)的三种处理方法和性能对比。
- 关注：三种处理方法的具体处理方式、性能对比。

提升题 论文阅读

深蓝学院
shenlanxueyuan.com

Zhang Z, Gallego G, Scaramuzza D. On the comparison of gauge freedom handling in optimization-based visual-inertial state estimation[J]. IEEE Robotics and Automation Letters, 2018, 3(3): 2710-2717.

- 目的：讨论使用最小二乘法解决VIO问题时处理不可观自由度(Gauge Freedom)的三种处理方法和性能对比。
- 关注：三种处理方法的具体处理方式、性能对比。

提升题 论文阅读

Global position和Global yaw不可观的后果：

对于代价函数： $J(\boldsymbol{\theta}) \doteq \underbrace{\|\mathbf{r}^V(\boldsymbol{\theta})\|^2_{\Sigma_V}}_{\text{Visual}} + \underbrace{\|\mathbf{r}^I(\boldsymbol{\theta})\|^2_{\Sigma_I}}_{\text{Inertial}},$

$$J(\boldsymbol{\theta}) = J(g(\boldsymbol{\theta})). \quad \text{where} \quad g \doteq \begin{pmatrix} \mathrm{R}_z & \mathbf{t} \\ 0 & 1 \end{pmatrix}$$

$$g(\boldsymbol{\theta}) = \boldsymbol{\theta}' \equiv \{\mathbf{p}'_i, \mathrm{R}'_i, \mathbf{v}'_i, \mathbf{X}'_j\}$$

$$\mathbf{p}'_i = \mathrm{R}_z \mathbf{p}_i + \mathbf{t} \qquad \mathrm{R}'_i = \mathrm{R}_z \mathrm{R}_i$$
$$\mathbf{v}'_i = \mathrm{R}_z \mathbf{v}_i \qquad \mathbf{X}'_j = \mathrm{R}_z \mathbf{X}_j + \mathbf{t}$$

任意对全局z轴的旋转变换和全局平移都不会改变代价函数的大小，即存在多个最优解。

提升题 论文阅读

深蓝学院 shenlanxueyuan.com

处理gauge freedom的三种方法：

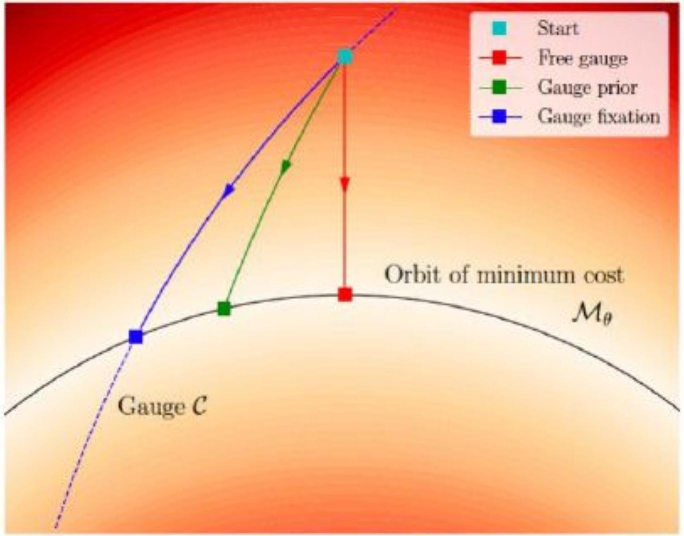
	Size of parameter vec.	Hessian (Normal eqs)
Fixed gauge	$n-4$	inverse, $(n-4)\times(n-4)$
Gauge prior	n	inverse, $n\times n$
Free gauge	n	pseudoinverse, $n\times n$

- Fixed gauge：固定第一帧的position和yaw。相当于引入了新的测量信息，使原本不可观的状态变得可观。信息矩阵H满秩，可求逆得到唯一最优解；
- Free gauge：不处理不可观的状态，得到的解会在零空间漂移。信息矩阵H不满秩，可用Moore-Penrose广义逆求解得到范数最小的最小二乘解；
- Gauge prior：介于上面两个方法之间，改变残差权重，添加额外的信息以处理不可观状态。信息矩阵满秩，可通过求逆得到唯一最优解。

提升题 论文阅读

三种方法的图形化对比：

- Fixed gauge： 将解限定在某一个流形上，最优解 $\theta_C = \mathcal{C} \cap \mathcal{M}_{\theta}$
- Free gauge： 获得最小范数的最小二乘解，“垂线段”最短
- Prior gauge： 介于二者之间



提升题 论文阅读

深蓝学院 shenlanxueyuan.com

三种方式的具体处理方法：

- Gauge prior：添加先验边（额外的惩罚项）。
- Gauge fixation：对应雅克比置零；将状态量不可观的变量直接剔除（在一个较小的参数空间进行优化）；将Gauge prior的先验设为无穷大。
- Free gauge：使用Moore-Penrose广义逆求解得到范数最小的最小二乘解；给H矩阵添加阻尼，即LM算法；将Gauge prior的先验设为无穷小。

如果使用Free gauge的方式，需要在每次优化过后将第一帧的position和yaw重新拉回到起始状态，整个窗口内的优化变量都会因此更新。

提升题 论文阅读

深蓝学院
shenlanxueyuan.com

Gauge fixation和Gauge prior的处理方法：

使用LM算法得到增量后，第Q次迭代后的旋转量为：

$$R^Q = \prod_{q=0}^{Q-1} \mathrm{Exp}(\delta\phi^q) R^0$$

即使我们限制z轴分量为0，每一次迭代都会在上一次的基础上改变roll和pitch，使得这一帧的z轴和最初的z轴不重合，总的来说还是改变了yaw。

所以我们对第一帧的位姿更新都是对0时刻的更新（相当于只更新一次）：

$$R_0 = \mathrm{Exp}(\Delta\phi_0) R_0^0$$

0时刻

初始帧

提升题 论文阅读

实验效果与结论：

- **先验权重。**不同的先验权值对求解的精度没有明显的影响， 但需要正确选择先验权值， 以保持较小的计算成本。
- **三种操作方式对比。**三种操作方式精度几乎相同； gauge prior 需要正确选择先验权值， 以保持较小的计算成本； free gauge 迭代次数更少， 计算更快一些， 并且还具有“通用” 的优势（直接广义逆或LM求解）；
- **协方差对比。**对于 gauge fixation， 第一帧的不确定度为零， 之后帧的不确定度增加； 对于 free gauge， 不确定度平摊到了每一帧，但可以使用 Covariance Transformation 将协方差转换成有意义的形式。

提升题 添加Prior约束

代码还是仿真的单目系统，所以需要同时给第一帧和第二帧添加先验约束。

代码中已经提供EdgeSE3Prior先验边，只需仿照其他残差在前两帧添加先验边即可。

```cpp
void EdgeSE3Prior::ComputeJacobians() {

    VecX param_i = verticies_[0]->Parameters();
    Qd Qi(param_i[6], param_i[3], param_i[4], param_i[5]);

    // w.r.t. pose i
    Eigen::Matrix<double, 6, 6> jacobian_pose_i = Eigen::Matrix<double, 6, 6>::Zero();

#ifdef USE_SO3_JACOBIAN
    Sophus::SO3d ri(Qi);
    Sophus::SO3d rp(Qp_);
    Sophus::SO3d res_r = rp.inverse() * ri;
    // http://rpg.ifi.uzh.ch/docs/RSS15_Forster.pdf  公式A.32
    jacobian_pose_i.block<3,3>(0,3) = Sophus::SO3d::JacobianRInv(res_r.log());
#else
    jacobian_pose_i.block<3,3>(0,3) = Qleft(Qp_.inverse() * Qi).bottomRightCorner<3, 3>();
#endif
    jacobian_pose_i.block<3,3>(3,0) = Mat33::Identity();

    jacobians_[0] = jacobian_pose_i;
//    std::cout << jacobian_pose_i << std::endl;
}
```

使用智能指针初始化先验边
Type edge_prior(...)
对边设置顶点
Something->SetVertex(...)
对边设置信息矩阵
Something->SetInformation(...)
对问题添加边
Something.AddEdge(..)

提升题 添加Prior约束

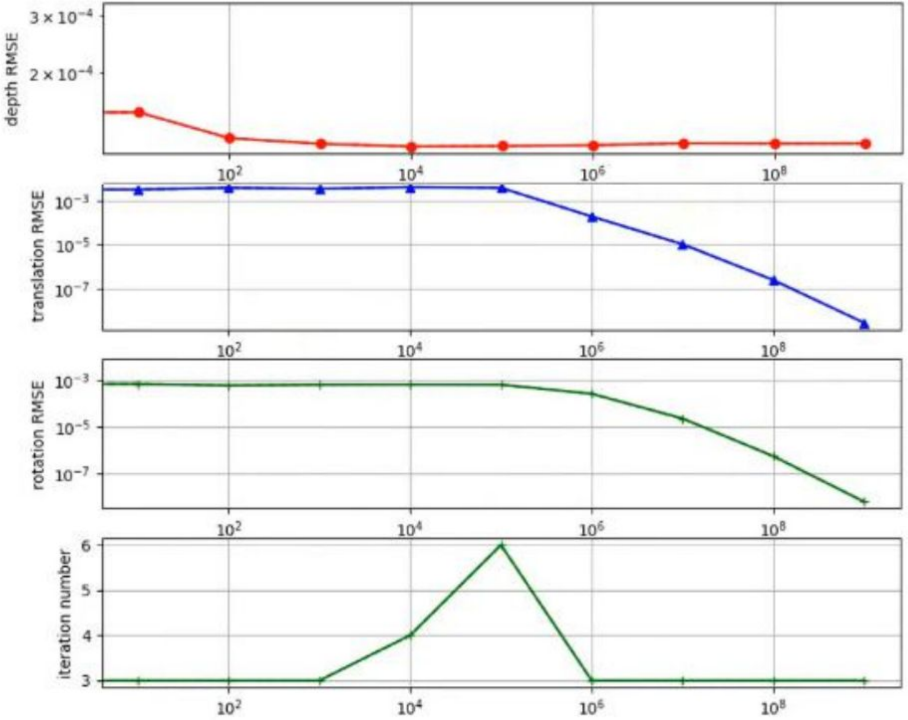
深蓝学院 shenlanxueyuan.com

先验边的残差和雅克比推导：

$$r^P = \begin{bmatrix} r_R^P \\ r_p^P \end{bmatrix} = \begin{bmatrix} \Delta\phi \\ p - p^0 \end{bmatrix} = \begin{bmatrix} \ln((R^0)^{-1}R)^\vee \\ p - p^0 \end{bmatrix}$$

$$J_{r^P} = \frac{\partial r^P}{\partial \begin{bmatrix} R \\ p \end{bmatrix}} = \begin{bmatrix} \frac{\partial r_R^P}{\partial R} & \frac{\partial r_R^P}{\partial p} \\ \frac{\partial r_p^P}{\partial R} & \frac{\partial r_p^P}{\partial p} \end{bmatrix} = \begin{bmatrix} \frac{\partial r_R^P}{\partial R} & 0 \\ 0 & \frac{\partial r_p^P}{\partial p} \end{bmatrix} = \begin{bmatrix} \frac{\partial \ln((R^0)^{-1}R)^\vee}{\partial R} & 0 \\ 0 & \frac{\partial (p - p^0)}{\partial p} \end{bmatrix} = \begin{bmatrix} J_r^{-1}(\ln((R^0)^{-1}R)^\vee) & 0 \\ 0 & I \end{bmatrix}$$

提升题 添加Prior约束



随着权重增加，有一个计算峰值

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

感谢各位聆听！

Thanks for Listening