深蓝学院
shenlanxueyuan.com

手写VIO第五章作业分享

主讲人 啦啦啦

题 目

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 $\mathbf{H}$ 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)

[a] Zichao Zhang, Guillermo Gallego, and Davide Scaramuzza. "On the comparison of gauge freedom handling in optimization-based visual-inertial state estimation". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2710–2717.

第一题

$$Hession(i,j) = J_i^T W J_j$$

$$Hession(j,i) = Hession(i,j)^T$$

```
// 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();
}
```

```

        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();
    }

}
Hessian_ = H;
b_ = b;
t_hessian_cost_ += t_h.toc();
```

第一题

深蓝学院 shenlanxueyuan.com

- T1.2：**舒尔补**：利用矩阵的稀疏性，通过舒尔补求解线性方程。

$$\begin{bmatrix} \mathbf{H}_{\mathrm{pp}} & \mathbf{H}_{\mathrm{pl}} \\ \mathbf{H}_{\mathrm{lp}} & \mathbf{H}_{\mathrm{ll}} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}_{\mathrm{p}}^{*} \\ \Delta \mathbf{x}_{\mathrm{l}}^{*} \end{bmatrix} = \begin{bmatrix} -\mathbf{b}_{\mathrm{p}} \\ -\mathbf{b}_{\mathrm{l}} \end{bmatrix} \quad (4)$$

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：

$$\left(\mathbf{H}_{\mathrm{pp}} - \mathbf{H}_{\mathrm{pl}} \mathbf{H}_{\mathrm{ll}}^{-1} \mathbf{H}_{\mathrm{pl}}^{\top}\right) \Delta \mathbf{x}_{\mathrm{p}}^{*} = -\mathbf{b}_{\mathrm{p}} + \mathbf{H}_{\mathrm{pl}} \mathbf{H}_{\mathrm{ll}}^{-1} \mathbf{b}_{\mathrm{l}} \quad (5)$$

求得 $\Delta \mathbf{x}_{\mathrm{p}}^{*}$ 后，再计算 $\Delta \mathbf{x}_{\mathrm{l}}^{*}$：

$$\mathbf{H}_{\mathrm{ll}} \Delta \mathbf{x}_{\mathrm{l}}^{*} = -\mathbf{b}_{\mathrm{l}} - \mathbf{H}_{\mathrm{pl}}^{\top} \Delta \mathbf{x}_{\mathrm{p}}^{*} \quad (6)$$

第一题

T1. 2. 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(高斯消元,消H_{pm})，得：

$$\begin{bmatrix} I & -H_{pm}H_{mm}^{-1} \\ 0 & I \end{bmatrix}$$

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

$$= \begin{bmatrix} b_p - H_{pm}H_{mm}^{-1}b_m \\ b_m \end{bmatrix}$$

```cpp
        // PCG solver
        MatXX H = Hessian_;
        for (ulong i = 0; i < Hessian_.cols(); ++i) {
            H(i, i) += currentLambda_;
        }
//          delta_x_ = PCGSolver(H, b_, H.rows() * 2);
        delta_x_ = Hessian_.inverse() * b_;

    } else {

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

第一题

深蓝学院
shenlanxueyuan.com

对应代码如下：

// TODO:: home work. 完成矩阵块取值，Hmm，Hpm，Hmp，bpp，bmm //因为Hpp被marge掉，所以它不用定义。

```
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： 利用公式求解

```
MatXX tempH = Hpm * Hmm_inv;
H_pp_schur_ = Hessian_.block(0, 0, reserve_size, reserve_size) - tempH * Hmp;
b_pp_schur_ = bpp - tempH * bmm;
```

第一题

T1.2.2.求解方程

消元之后，方程组第一行变成与Δx_m的无关项，先求Δx_p：

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

之后，再求Δx_m：

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

```
        // 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;

    }

}
```

优化结果近似真值，但由于单目SLAM系统7自由度不可观，优化结果会在零空间内漂移，可以通过fix前两帧位姿。代码在TwstMonoBA中已给出。

```
if(i < 2)
    vertexCam->SetFixed();
```

第一题

```cpp
        // 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;

    }

}
```

```cpp
// 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;
```

第一题：代码执行结果

运行结果：

```
0 order: 0
1 order: 6
2 order: 12
```

```
ordered_landmark_vertices_ size : 20
iter: 0 , chi= 5.35099 , Lambda= 0.00597396
iter: 1 , chi= 0.0289048 , Lambda= 0.00199132
iter: 2 , chi= 0.000109162 , Lambda= 0.000663774
problem solve cost: 29.4994 ms
   makeHessian cost: 23.8575 ms
```

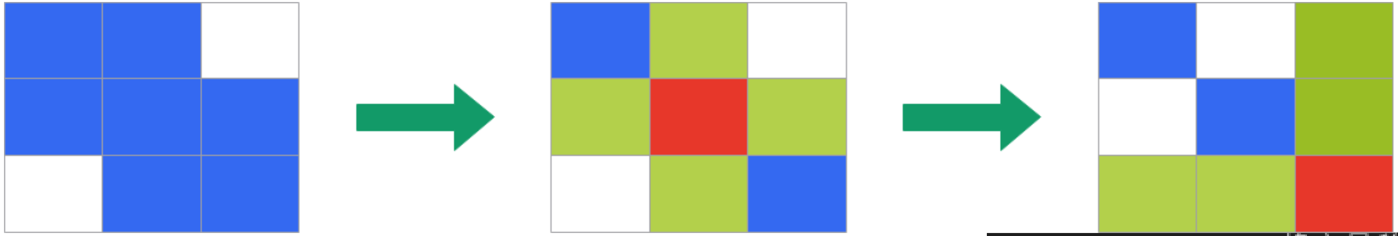
```
Compare MonoBA results after opt...
after opt, point 0 : gt 0.220938 ,noise 0.227057 ,opt 0.220992
after opt, point 1 : gt 0.234336 ,noise 0.314411 ,opt 0.234854
after opt, point 2 : gt 0.142336 ,noise 0.129703 ,opt 0.142666
after opt, point 3 : gt 0.214315 ,noise 0.278486 ,opt 0.214502
after opt, point 4 : gt 0.130629 ,noise 0.130064 ,opt 0.130562
after opt, point 5 : gt 0.191377 ,noise 0.167501 ,opt 0.191892
after opt, point 6 : gt 0.166836 ,noise 0.165906 ,opt 0.167247
after opt, point 7 : gt 0.201627 ,noise 0.225581 ,opt 0.202172
after opt, point 8 : gt 0.167953 ,noise 0.155846 ,opt 0.168029
after opt, point 9 : gt 0.21891 ,noise 0.209697 ,opt 0.219314
after opt, point 10 : gt 0.205719 ,noise 0.14315 ,opt 0.205995
after opt, point 11 : gt 0.127916 ,noise 0.122109 ,opt 0.127908
after opt, point 12 : gt 0.167904 ,noise 0.143334 ,opt 0.168228
after opt, point 13 : gt 0.216712 ,noise 0.18526 ,opt 0.216866
after opt, point 14 : gt 0.180009 ,noise 0.184249 ,opt 0.180036
after opt, point 15 : gt 0.226935 ,noise 0.245716 ,opt 0.227491
after opt, point 16 : gt 0.157432 ,noise 0.176529 ,opt 0.157589
after opt, point 17 : gt 0.182452 ,noise 0.14729 ,opt 0.182444
after opt, point 18 : gt 0.155701 ,noise 0.182258 ,opt 0.155769
after opt, point 19 : gt 0.14646 ,noise 0.240649 ,opt 0.14677
------------ pose translation ----------------
translation after opt: 0 :-0.000477998   0.00115906  0.000366506 || 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
```

第二题

T2. 1. 将被边缘化的变量移到右下角：

```
// 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;

// 将 col i 移动矩阵最右边
```



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

第二题

T2.2. 完成舒尔补操作：

```
// 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
```

第二题

代码以及运行结果如下：

```
// 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 || idx行 与 最后一行互换
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(idx + 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;
```

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

第三题-论文阅读

T3：阅读论文：

Zichao Zhang, Guillermo Gallego, and Davide Scaramuzza. “On the comparison of gauge freedom handling in optimization-based visual-inertial state estimation”. In: IEEE Robotics and Automation Letters 3.3 (2018), pp. 2710–2717

- 目的：对基于优化的视觉惯性状态估计中处理不可观自由度(gauge freedom) 的不同方法进行了比较分析。
- 三种常用方法的实现方式，效果，结论。

第三题-论文阅读

global position and yaw are not observable:

优化目标函数 $$J(\boldsymbol{\theta}) \doteq \underbrace{\|\mathbf{r}^V(\boldsymbol{\theta})\|^2_{\Sigma_V}}_{\text{Visual}} + \underbrace{\|\mathbf{r}^I(\boldsymbol{\theta})\|^2_{\Sigma_I}}_{\text{Inertial}}, \quad (1)$$

优化参数 $$\boldsymbol{\theta} \doteq \{\mathbf{p}_i, \mathrm{R}_i, \mathbf{v}_i, \mathbf{X}_j\},$$

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

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

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

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

优化目标函数中没有一个唯一的最小值，因为有无穷多个重构可以达到相同的最小误差。VI估计问题有一些不确定性或不可观测状态:没有足够的方程来完全指定唯一的解

第三题–论文阅读

深蓝学院 shenlanxueyuan.com

三种处理gauge freedom的方法

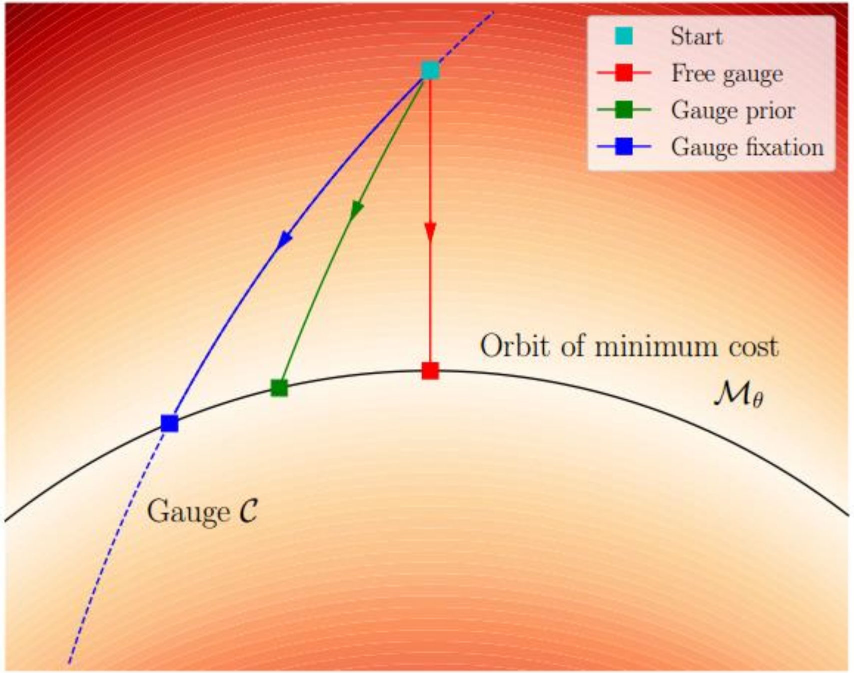
	Size of parameter vec.	Hessian (Normal eqs)
Fixed gauge	$n-4$	inverse, $(n-4)\times(n-4)$
Gauge prior	n	inverse, $n\times n$
Free gauge	n	pseudoinverse, $n\times n$

- **gauge fixation :** 固定第一帧的**position and yaw**。是在一个较小的参数空间中进行优化，在这个空间中不存在不可观的状态，因此Hessian是可逆的。
- **gauge prior**：使用额外的信息(产生可逆的**Hessian**)来扩大目标函数，以处理不可观状态。改变残差权重，Hessian满秩，求逆得到唯一解。
- **free gauge**：不处理不可观状态，得到的解会在零空间漂移。Hessian矩阵不满秩，可用Moore-Penrose广义逆求解得到范数最小的最小二乘解。

第三题-论文阅读

三种方法图形化对比：

- **gauge fixation :** 始终在gauges C上移动
- **gauge prior**：在两者之间的路径上。
- **free gauge**：求得范数最小的最小二乘解，因此垂直。



第三题-论文阅读

深蓝学院
shenlanxueyuan.com

具体处理方法：

- **gauge fixation :**固定第一相机姿态的 position and yaw，等价于将对应雅克比设为零；将状态量不可观的变量去除；将gauge prior的先验设为无穷大。
- **gauge prior：** 使用额外的惩罚项，即添加先验边。
- **free gauge：** 用Moore-Penrose广义逆求解得到范数最小的最小二乘解；给Hessian添加一些阻尼(Levenberg-Marquardt算法)；将gauge prior的先验设为无穷小。

这是什么意思？

第三题-论文阅读

实验结论：

- **先验权重**：不同的先验权重对求解的精度没有明显影响，但需要正确选择先验权重，以保证较小的计算成本。
- **三种方式的对比**：这三种方法的准确性几乎相同。在gauge prior方法中，需要选择适当的先验权重以避免增加计算量；free gauge迭代次数更少，计算速度更快，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。

第三题-添加prior约束

由于代码是单目SLAM(7自由度)，所以需要给第一帧和第二帧添加先验约束。

代码中以及提供了 EdgeSE3Prior 先验边，所以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

problem.AddEdge(...)

第三题-添加prior约束

先验边的残差和雅克比的推导：

$$r^p = \begin{bmatrix} r^p_R \\ r^p_p \end{bmatrix} = \begin{bmatrix} \Delta\phi \\ p - p^0 \end{bmatrix} = \begin{bmatrix} ln((R^0)^{-1}R)^\vee \\ p - p^0 \end{bmatrix} \tag{12}$$

$$J^p_r = \frac{\partial r^p}{\partial \begin{bmatrix} R \\ p \end{bmatrix}} = \begin{bmatrix} \frac{\partial r^p_R}{\partial R} & \frac{\partial r^p_R}{\partial p} \\ \frac{\partial r^p_p}{\partial R} & \frac{\partial r^p_p}{\partial p} \end{bmatrix} = \begin{bmatrix} \frac{\partial r^p_R}{\partial R} & 0 \\ 0 & \frac{\partial r^p_p}{\partial p} \end{bmatrix} \tag{13}$$

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

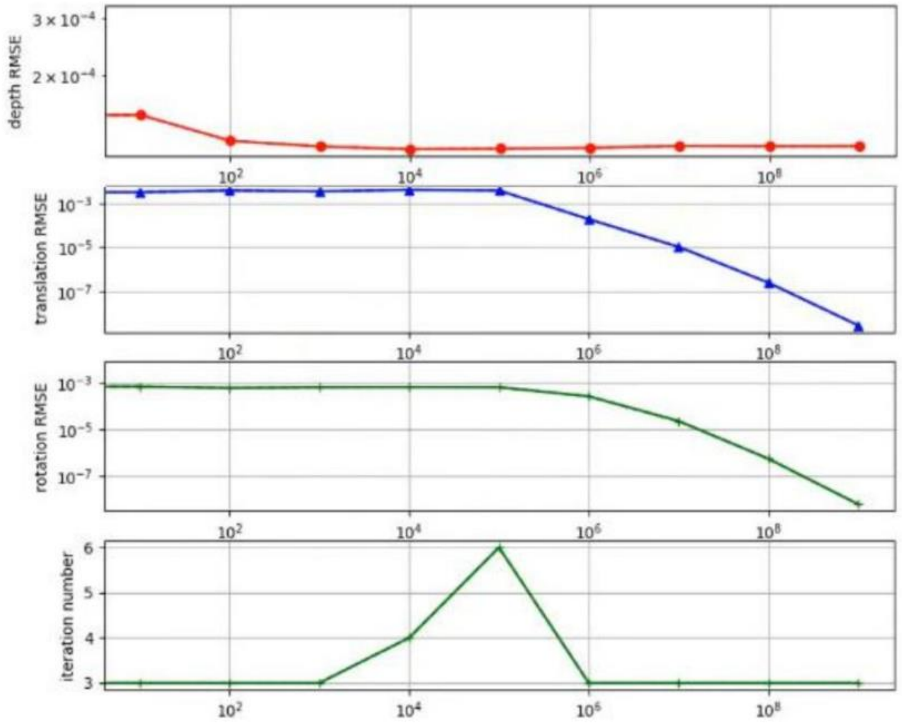
第三题-添加prior约束

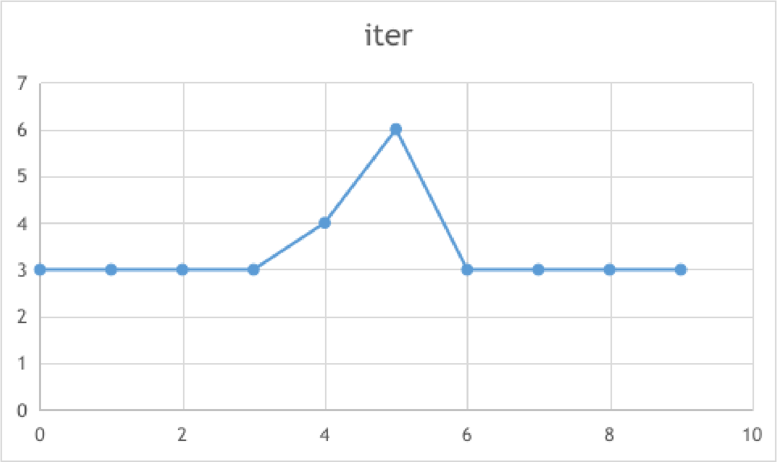
代码修改：

- 添加头文件 #include ″backend/edge_prior.h″
- 在main函数中添加先验边：

```
double Wp = Weight[i];
for (size_t i = 0; i < 2; ++i) {
    shared_ptr<EdgeSE3Prior> edge_prior(new EdgeSE3Prior(cameras[i].twc,
      cameras[i].qwc));
    std::vectorstd::shared_ptr<Vertex> edge_prior_vertex;
    edge_prior_vertex.push_back(vertexCams_vec[i]);
    edge_prior->SetVertex(edge_prior_vertex);
    edge_prior->SetInformation(edge_prior->Information() * Wp);
    problem.AddEdge(edge_prior);
}
```

第三题-添加prior约束





随着权重的增加，迭代次数变换如图所示。

在线问答

Q&A

感谢各位聆听！

Thanks for Listening