手写VIO第五章作业分享

主讲人 徐炜波

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 $\mathbf{H}$ 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)

1.1 MakeHession()填空

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

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

// 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();
```

1.2 SolveLinearSystem()求解

深蓝学院 shenlanxueyuan.com

利用舒尔补加速 SLAM 问题的求解

直接求解 $\Delta\mathbf{x} = -\mathbf{H}^{-1}\mathbf{b}$，计算量大。解决办法：舒尔补，利用 SLAM 问题的稀疏性求解。

比如，某单目 BA 问题，其信息矩阵如有图所示，可以将其分为：

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

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：

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

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

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

1.2 SolveLinearSystem()求解

深蓝学院 shenlanxueyuan.com

```cpp
// 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(?,?);

```

```cpp
// 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);
```

1.2 SolveLinearSystem()求解

```
// 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 - ? * ?;
```

```
// 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;
```

1.2 SolveLinearSystem()求解

深蓝学院
shenlanxueyuan.com

```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;
```

1.2 SolveLinearSystem()求解

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
```

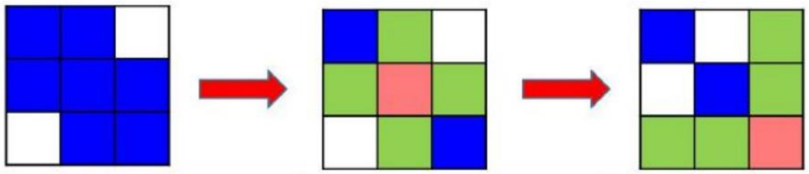
```
Compare MonoBA results after opt...
after opt, point 0 : gt 0.220938 ,noise 0.227057 ,opt 0.220992
after opt, point 1 : gt 0.234336 ,noise 0.314411 ,opt 0.234854
after opt, point 2 : gt 0.142336 ,noise 0.129703 ,opt 0.142666
after opt, point 3 : gt 0.214315 ,noise 0.278486 ,opt 0.214502
after opt, 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
```

2 TestMarginalize()求解

```
// 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, dim, reserve_size) = temp_botRows;
H_marg.block(idx + dim, 0, reserve_size - idx - 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;
```



运行结果:

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

3 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.

3 提升题论文阅读

处理H自由度的方法：

gauge fixation 方法主要是在优化过程中固定第一个相机的位置和偏航角。参数设定：

$$p_0 = p_0^0, \qquad \Delta\phi_{0z} \doteq e_z^T \Delta\phi_0 = 0 \tag{1}$$

p_0^0 是第一个相机的初始化位置。

gauge prior 方法是在目标函数中增加一个惩罚。

$$\left\| r_0^p \right\|_{\Sigma_0^p}^2, \text{其中} r_0^p(\theta) \doteq \left(p_0 - p_0^0, \Delta\phi_{0z} \right) \tag{2}$$

通常设定 $\Sigma_0^p = \sigma_0^2 I$，则 $\left\| r_0^p \right\|_{\Sigma_0^p}^2 = \frac{1}{\sigma_0^2} \left\| r_0^p \right\|^2$

free gauge 方法就是在优化过程中让参数自由的演化，处理奇异的 H 矩阵时，使用伪逆矩阵或者像 LM 算法中加入阻尼求解更新量。

3 提升题论文阅读

实验流程：

首先确定先验权重，通过比较设定不同先验权重下，gauge prior 方法估计的精度和效率。最终选择了 $w_p\left(\text{即}1/\sigma_0^2\right)=10^5$ 作为 gauge prior 方法的先验权重。

而后对轨迹和路标点添加干扰，分别对精度和计算效率进行仿真，结论如下：

1、 三种方法的精度几乎相同。
2、 在 gauge prior 方法中需要选择合适的先验权重以避免计算时间增加。
3、 当 gauge prior 方法中的先验权重选择合适时，该方法和 gauge fixation 方法几乎有同样的精度和计算效率。
4、 free gauge 方法比其他两种方法计算效率更高，因为其收敛所需的迭代次数更少。

最后对三种方法的协方差矩阵进行了比较。由于 gauge prior 方法和 gauge fixation 方法相似，就直接忽略了。仿真结果发现，free gauge 方法的协方差矩阵和其他两种方法不同，但它们可以通过线性矩阵进行转换。

4 提升题添加Prior约束

代码修改：

（1）添加头文件#include "backend/edge_prior.h"

（2）在main函数中添加先验的边：

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

4 提升题添加Prior约束

表 1 不同先验权重的迭代次数和运行时间

w_p	0	7	100	500	5×10^3	5×10^4	5×10^5	5×10^6
迭代次数	2	2	3	4	2	2	2	2
计算时间（ms）	74.7	57.3	60.9	75.0	45.1	45.9	44.2	43.6

在线问答

Q&A

感谢各位聆听！

Thanks for Listening