Course5_6_homework_solver_front_end

0. Summary

（1）包含了 course5 和 course6 的 homework；

$$ Course_5_hw_solver

1. BA 求解

❶ 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。

```
                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;
                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(?,?, ?, ?).noalias() += hessian.transpose();
                    H.block(index_j,index_i, dim_j, dim_i).noalias() += hessian.transpose();//!!
                }
            }
            b.segment(index_i, dim_i).noalias() -= JtW * edge.second->Residual();
        }

    }
```

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

myslam › backend › Problem::SolveLinearSystem

```cpp
        // 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 - ? * ?;
        H_pp_schur_ = Hessian_.block( startRow: 0, startCol: 0,reserve_size,reserve_size) - tempH * Hmp;//!!
        b_pp_schur_ = bpp - tempH * bmm;//!!

        // 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_ll = Hmm_inv*(bmm-Hmp*delta_x_pp);//!!

        delta_x_.tail(marg_size) = delta_x_ll;

    }
```

myslam › backend › Problem::SolveLinearSystem

运行结果：

```
(base) root@ep-VirtualBox:/media/sf_vslam_vio/lesson_doc/hw_course5_new/cmake-build-debug# ./app/testMonoBA
cameras.size: 3;  points.size: 20;
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: 78.8305 ms
   makeHessian cost: 68.1854 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 :-0.000477992   0.00115908  0.000366503 || 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
(base) root@ep-VirtualBox:/media/sf_vslam_vio/lesson_doc/hw_course5_new/cmake-build-debug#
```

2.滑动窗口算法

❷ **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。

```
eLists.txt × | problem.cc × | shared_ptr_base.h × | CurveFitting.cpp × | TestMonoBA.cpp × | vertex_pose.h ×
```

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


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

```
myslam › backend › Problem::TestMarginalize
```

```cpp
        // 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 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;

        std::cout << "---------- TEST Marg: after marg------------" << std::endl;
        std::cout << H_prior << std::endl;


    }
```

```
myslam › backend › Problem::TestMarginalize
```

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
(base) root@ep-VirtualBox:/media/sf_vslam_vio/lesson_doc/hw_course5_new/cmake-build-debug#
```

3. 论文总结 H 自由度的处理方法

- **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)**

Paper 中比较了 3 种方法，结论是从计算量和准确性角度，都没有明显的差别；

Paper 种还比较了无约束和有约束时的计算结果的方差，结论是：在“无约束”的计算结果中去掉属于解空间的“假波动量”，剩下的是计算误差的“真波动量”，发现“无约束”和“有约束”的“真波动量”的方差基本上相同，并且有转换公式。

4. Code 中修改约束

- **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)**

原始 code 中没有约束；计算结果有少量漂移；

以下 code 进行约束，fix 第 1、2 帧，结果和 gt 值相同（即 fixed）；

```
    // 所有 Pose
    vector<shared_ptr<VertexPose> > vertexCams_vec;
    for (size_t i = 0; i < cameras.size(); ++i) {
        shared_ptr<VertexPose> vertexCam( p: new VertexPose());
        Eigen::VectorXd pose( x: 7);
        pose << cameras[i].twc, cameras[i].qwc.x(), cameras[i].qwc.y(), cameras[i].qwc.z(), cameras[i].qwc.w();
        vertexCam->SetParameters(pose);

        if(i < 2) //!!
            vertexCam->SetFixed(); //!!

        problem.AddVertex(vertexCam);
        vertexCams_vec.push_back(vertexCam);
    }

    // 所有 Point 及 edge
    std::default_random_engine generator;
```

```
(base) root@ep-VirtualBox:/media/sf_vslam_vio/lesson_doc/hw_course5_new/cmake-build-debug# ./app/testMonoBA
cameras.size: 3;  points.size: 20;
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= 0.0282599 , Lambda= 0.00199132
iter: 2 , chi= 0.000117497 , Lambda= 0.000663774
problem solve cost: 68.1558 ms
   makeHessian cost: 50.0887 ms

Compare MonoBA results after opt...
after opt, point 0 : gt 0.220938 ,noise 0.227057 ,opt 0.220909
after opt, point 1 : gt 0.234336 ,noise 0.314411 ,opt 0.234374
after opt, point 2 : gt 0.142336 ,noise 0.129703 ,opt 0.142353
after opt, point 3 : gt 0.214315 ,noise 0.278486 ,opt 0.214501
after opt, point 4 : gt 0.130629 ,noise 0.130064 ,opt 0.130511
after opt, point 5 : gt 0.191377 ,noise 0.167501 ,opt 0.191539
after opt, point 6 : gt 0.166836 ,noise 0.165906 ,opt 0.166965
after opt, point 7 : gt 0.201627 ,noise 0.225581 ,opt 0.201859
after opt, point 8 : gt 0.167953 ,noise 0.155846 ,opt 0.167965
after opt, point 9 : gt 0.21891 ,noise 0.209697 ,opt 0.218834
after opt, point 10 : gt 0.205719 ,noise 0.14315 ,opt 0.205683
after opt, point 11 : gt 0.127916 ,noise 0.122109 ,opt 0.127751
after opt, point 12 : gt 0.167904 ,noise 0.143334 ,opt 0.167924
after opt, point 13 : gt 0.216712 ,noise 0.18526 ,opt 0.216885
after opt, point 14 : gt 0.180009 ,noise 0.184249 ,opt 0.179961
after opt, point 15 : gt 0.226935 ,noise 0.245716 ,opt 0.227114
after opt, point 16 : gt 0.157432 ,noise 0.176529 ,opt 0.157529
after opt, point 17 : gt 0.182452 ,noise 0.14729 ,opt 0.1823
after opt, point 18 : gt 0.155701 ,noise 0.182258 ,opt 0.155627
after opt, point 19 : gt 0.14646 ,noise 0.240649 ,opt 0.146533
------------ 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
```

$$ course_6_hw_front_end

1. 证明

❶ 证明式(15)中，取 $\mathbf{y} = \mathbf{u}_4$ 是该问题的最优解。提示：设 $\mathbf{y}\prime = \mathbf{u}_4 + \mathbf{v}$，其中 $\mathbf{v}$ 正交于 $\mathbf{u}_4$，证明

$$\mathbf{y}\prime^\top \mathbf{D}^\top \mathbf{D} \mathbf{y}\prime \geq \mathbf{y}^\top \mathbf{D}^\top \mathbf{D} \mathbf{y}$$

该方法基于奇异值构造矩阵零空间的理论。

- 由于 $\mathbf{D} \in \mathbb{R}^{2n \times 4}$，在观测次于大于等于两次时，很可能 $\mathbf{D}$ 满秩，无零空间。
- 寻找最小二乘解：

$$\min_{\mathbf{y}} \|\mathbf{D}\mathbf{y}\|_2^2, \quad s.t. \|\mathbf{y}\| = 1 \tag{14}$$

解法：对 $\mathbf{D}^\top \mathbf{D}$ 进行 SVD：

$$\mathbf{D}^\top \mathbf{D} = \sum_{i=1}^{4} \sigma_i^2 \mathbf{u}_i \mathbf{u}_j^\top \tag{15}$$

其中 σ_i 为奇异值，且由大到小排列，$\mathbf{u}_i, \mathbf{u}_j$ 正交。

证明：

SVD分解中 σ_i 沿对角线从大到小排列

$$(\mathbf{u}_4+\mathbf{v})^\top \mathbf{D}^\top \mathbf{D}(\mathbf{u}_4+\mathbf{v})$$

$$=\sum_{i=1}^{4}\sigma_i^2\ (\mathbf{u}_4+\mathbf{v})^\top \mathbf{u}_i\mathbf{u}_i^\top(\mathbf{u}_4+\mathbf{v})$$

$$=\sum_{i=1}^{4}\sigma_i^2\ (\mathbf{u}_4^\top+\mathbf{v}^\top)\ \mathbf{u}_i\mathbf{u}_i^\top(\mathbf{u}_4+\mathbf{v})$$

$$=\sum_{i=1}^{4}\sigma_i^2\ (\mathbf{u}_4^\top\mathbf{u}_i+\mathbf{v}^\top\mathbf{u}_i)\ (\ \mathbf{u}_i^\top\mathbf{u}_4+\mathbf{u}_i^\top\mathbf{v})$$

SVD 分解中,u1~u4 构成完整的空间，且互相正交；

若v=$\mathbf{u}_1$，根据对称矩阵的svd分解的特征向量的正交特性，

可得：

$$\sigma_1^2+\sigma_4^2$$

可知 $\mathbf{y}^\top\mathbf{D}^\top\mathbf{D}\mathbf{y}=\sigma_4^2$

所以 $\sigma_1^2+\sigma_4^2\ \geq\ \sigma_4^2$

其余的v=u2,或u3,等可以类似推理；

$\mathbf{y}'^\top\mathbf{D}^\top\mathbf{D}\mathbf{y}'\geq\mathbf{y}^\top\mathbf{D}^\top\mathbf{D}\mathbf{y}$ 得到证明；

所以y是最优解

2. Code：三角化

Code：

triangulate.cpp

```cpp
    Eigen::Vector3d P_est;           // 结果保存到这个变量
    P_est.setZero();
    /* your code begin */
    int D_rows = 2 * (end_frame_id - start_frame_id);
    Eigen::MatrixXd D = Eigen::MatrixXd::Zero(D_rows, cols: 4);
    for (int i = start_frame_id; i < end_frame_id;++i)
    {

        Eigen::Matrix3d Rcw = camera_pose[i].Rwc.transpose();
        Eigen::Vector3d tcw = -Rcw * camera_pose[i].twc;
        D.block( startRow: 2 * (i - start_frame_id), startCol: 0, blockRows: 1, blockCols: 3) .noalias()
                = camera_pose[i].uv( index: 0) * Rcw.block( startRow: 2, startCol: 0, blockRows: 1, blockCols: 3)
                  - Rcw.block( startRow: 0, startCol: 0, blockRows: 1, blockCols: 3);
        D.block( startRow: 2 * (i - start_frame_id), startCol: 3, blockRows: 1, blockCols: 1).noalias() =
                camera_pose[i].uv[0] * tcw.segment( start: 2, n: 1) - tcw.segment( start: 0, n: 1);
        D.block( startRow: 2 * (i - start_frame_id) + 1, startCol: 0, blockRows: 1, blockCols: 3).noalias()=
                camera_pose[i].uv( index: 1) * Rcw.block( startRow: 2, startCol: 0, blockRows: 1, blockCols: 3) - Rcw.block( startRow: 1, startCol: 0, blockRows: 1, blockCols: 3);
        D.block( startRow: 2 * (i - start_frame_id) + 1, startCol: 3, blockRows: 1, blockCols: 1).noalias()
                = camera_pose[i].uv( index: 1) * tcw.segment( start: 2, n: 1) - tcw.segment( start: 1, n: 1);
```

main

```cpp
    }
    Eigen::JacobiSVD<Eigen::MatrixXd> svd(
            D.transpose() * D, computationOptions: Eigen::ComputeThinU | Eigen::ComputeThinV);
    Eigen::Vector4d lamda = svd.singularValues();
    std::cout << "奇异值 = " << lamda.transpose() << std::endl;
    if(lamda( index: 2)/lamda( index: 3)<1e-3){
        std::cout << "The parallax is not enough! " << std::endl;
        return -1;
    }

    Eigen::Vector4d u4 = svd.matrixU().block( startRow: 0, startCol: 3, blockRows: 4, blockCols: 1);
    if(u4( index: 3)!=0 && u4( index: 2)/u4( index: 3)>0){
        P_est( index: 0) = u4( index: 0) / u4( index: 3);
        P_est( index: 1) = u4( index: 1) / u4( index: 3);
        P_est( index: 2) = u4( index: 2) / u4( index: 3);
    }


    /* your code end */
```

运行结果：

```
(base) ep@ep-VirtualBox:/media/sf__xcloud/notes/vslam_vio/lesson_doc/course6_hw$ ./cmake-build-debug/estimate_depth
奇异值 =     468.406     7.74642     0.723255 5.30104e-16
ground truth:
  -2.9477 -0.330799    8.43792
your result:
  -2.9477 -0.330799    8.43792
```