大作业提示

—— 预祝各位顺利毕业！

主讲人 会打篮球的猫

Square Root Bundle Adjustment for Large-Scale Reconstruction

Nikolaus Demmel Christiane Sommer Daniel Cremers Vladyslav Usenko
Technical University of Munich

- 读论文（摘要、证明、示意图、公式推导细看，实验部分了解）
- 代码熟悉总体流程、关键步骤（不要仅看TODO部分）
- 多查阅资料

Square Root Bundle Adjustment for Large-Scale Reconstruction

Nikolaus Demmel Christiane Sommer Daniel Cremers Vladyslav Usenko
Technical University of Munich

We propose a new formulation for the bundle adjustment problem which relies on nullspace marginalization of landmark variables by QR decomposition. Our approach, which we call square root bundle adjustment, is algebraically equivalent to the commonly used Schur complement trick, improves the numeric stability of computations, and allows for solving large-scale bundle adjustment problems with single-precision floating-point numbers. We show in real-world experiments with the BAL datasets that even in single precision the proposed solver achieves on average equally accurate solutions compared to Schur complement solvers using double precision. It runs significantly faster, but can require larger amounts of memory on dense problems. The proposed formulation relies on simple linear algebra operations and opens the way for efficient implementations of bundle adjustment on hardware platforms optimized for single-precision linear algebra processing.

- 对路标点进行零空间边缘化（QR分解）
- 代数上等价于舒尔补
- Float达到double的精度

论文所提方法的核心在于：不使用传统的舒尔补的方法，而是直接对 Jl 进行 QR 分解，从而将问题转化为只包含相机位姿的小规模问题，避免了原始高维数 H 矩阵的构建，极大地提升了效率，同时有很好的数值稳定性，即使用 float 型数据依旧能够达到 double 型的精度。

$$
\begin{aligned}
&\| r + \begin{pmatrix} J_p & J_l \end{pmatrix} \begin{pmatrix} \Delta x_p \\ \Delta x_l \end{pmatrix} \|^2 \\
&= \| Q^\top r + \begin{pmatrix} Q^\top J_p & Q^\top J_l \end{pmatrix} \begin{pmatrix} \Delta x_p \\ \Delta x_l \end{pmatrix} \|^2 \qquad (15) \\
&= \| Q_1^\top r + Q_1^\top J_p \Delta x_p + R_1 \Delta x_l \|^2 \\
&\quad + \| Q_2^\top r + Q_2^\top J_p \Delta x_p \|^2 .
\end{aligned}
$$

$$
\Delta x_l^* = -R_1^{-1} (Q_1^\top r + Q_1^\top J_p \Delta x_p^*) . \qquad (16)
$$

So (6) reduces to minimizing the second term in (15):

$$
\min_{\Delta x_p} \| Q_2^\top r + Q_2^\top J_p \Delta x_p \|^2 . \qquad (17)
$$

- Bug——在 kernel_function.hpp 中，核函数一阶导数返回变量名错误

```
/* 计算 Trival 鲁棒核函数的一阶导数 */
Scalar &ComputeOneOrder(const Scalar &x) override {
    this->y_ = Scalar(1);
    return this->y;
}
```

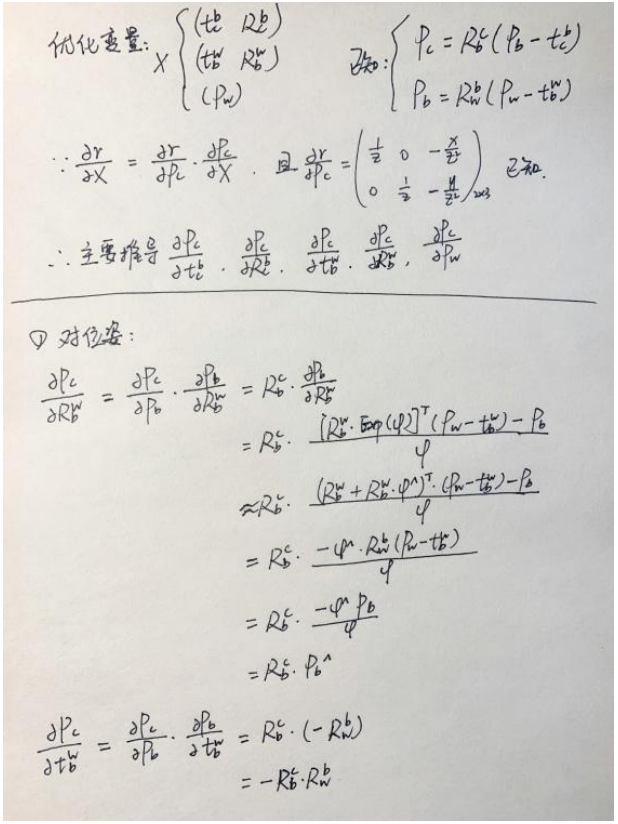
```
/* 计算 Huber 鲁棒核函数的一阶导数 */
Scalar &ComputeOneOrder(const Scalar &x) override {
    this->y_ = Scalar(1);
    if (this->x > this->dsqr) {
        this->y_ = this->delta / std::sqrt(this->x);
    }
    return this->y;
}
```

- 1、雅克比矩阵

首先残差是 3D 到 2D 的投影误差，设优化参数为 X 的话，根据链式法则，有$\frac{\partial r}{\partial X}=\frac{\partial r}{\partial P_c}\cdot\frac{\partial P_c}{\partial X}$，由于$\frac{\partial r}{\partial P_c}$已经给出，因此重点求解后面的$\frac{\partial P_c}{\partial X}$。

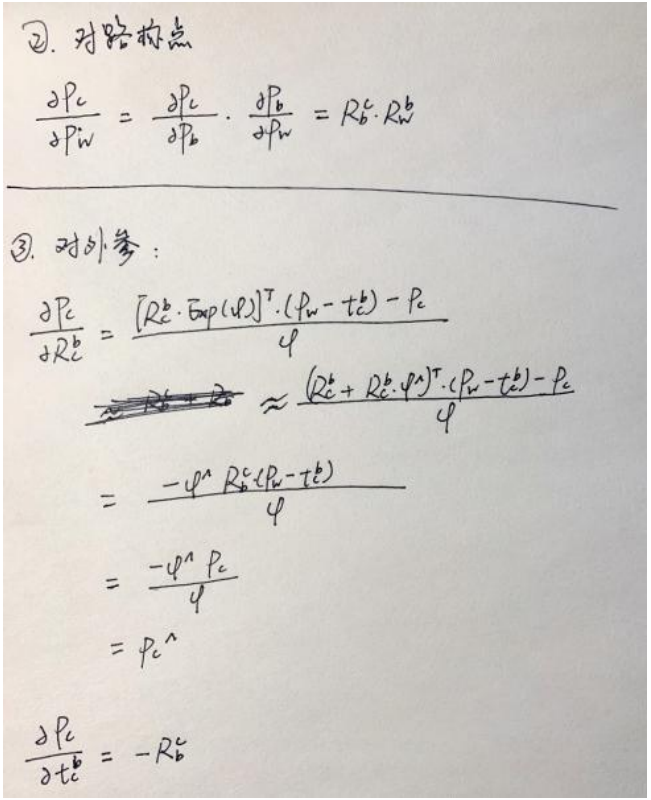
待优化参数包括：外参数（t_c^b，R_c^b）、相机位姿（t_b^W，R_b^W）和路标点在世界系的三维坐标P_W这三部分，也就是代码中对应填写的三部分。

```
// 计算重投影的归一化平面坐标 norm = [ux, uy, 1].T
Vector3<Scalar> p_b = q_wb.inverse() * (p_w - t_wb);
Vector3<Scalar> p_c = q_bc.inverse() * (p_b - t_bc);
Scalar invDep = 1.0 / p_c.z();
Vector2<Scalar> norm = p_c.template head<2>() / p_c.z();

// 计算重投影误差
Vector2<Scalar> residual;
if (std::isnan(invDep) || std::isinf(invDep)) {  // 判断这个点是否正常
    residual.setZero();
} else {
    residual = norm - measure;
}
```



- 1、雅克比矩阵

```
// 计算雅可比矩阵 d(相机坐标位置误差) / d(相机 6 自由度位姿)
//  TODO: task 1 calculate jacobian
Eigen::Matrix<Scalar, 3, 6> tempJp = Eigen::Matrix<Scalar, 3, 6>::Zero();
if (camera->IsFixed() == false) {
    tempJp.template block<3, 3>(0, 0) = -R_cb * R_bw;
    tempJp.template block<3, 3>(0, 3) = R_cb * SkewSymmetricMatrix(p_b);
}

// 计算雅可比矩阵 d(相机坐标位置误差) / d(特征点 3 自由度位置)
Eigen::Matrix<Scalar, 3, 3> tempJl = Eigen::Matrix<Scalar, 3, 3>::Zero();
if (this->landmark->IsFixed() == false) {
    tempJl = R_cb * R_bw;
}

// 计算雅可比矩阵 d(相机坐标位置误差) / d(相机外参 6 自由度位姿)
// Hint: to calculate skew symmetric matrix, you could use function SkewSymmetricMatrix(*)
Eigen::Matrix<Scalar, 3, 6> tempJex = Eigen::Matrix<Scalar, 3, 6>::Zero();
if (this->exPose->IsFixed() == false) {
    tempJex.template block<3, 3>(0, 0) = -R_cb;
    tempJex.template block<3, 3>(0, 3) = SkewSymmetricMatrix(p_c);
}
// Fill in above this line
```



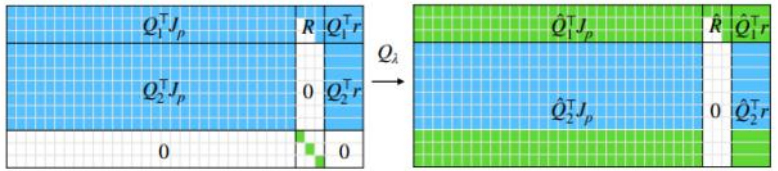
- 2、对Jl进行QR分解，并将Q转置左乘在整个block矩阵上

```
// TODO: task 2 执行QR分解，并更新sorage矩阵

// 对 Jl 进行 QR 分解
//HInt: Eigen::HouseholderQR could be used to perform QR decompose.
Eigen::HouseholderQR<MatrixX<Scalar>> qr;
qr.compute(Jl);
MatrixX<Scalar> Qt = qr.householderQ().transpose();

// // 对 storage 矩阵左乘 Q.T
this->storage.block(0, cols - 4, rows - 3, 3) = Qt * Jl;
this->storage.block(0, 0, rows - 3, cols - 4) = Qt * Jp;
this->storage.block(0, cols - 1, rows - 3, 1) = Qt * r;

// Fill in above this line
```

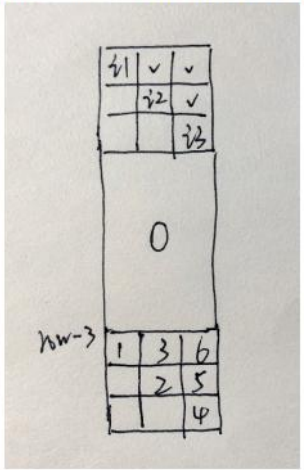
上三角矩阵

- 3、对landmark添加阻尼

对应论文中下图所示的过程，在 block 矩阵下方增加三行，并增加一个 3*3 的对角线阻尼元素，进行 6 次 Givens 旋转即可变成右侧形式。



下图中在草稿本上简单画了一下此处的 6 次 Givens 旋转操作流程。



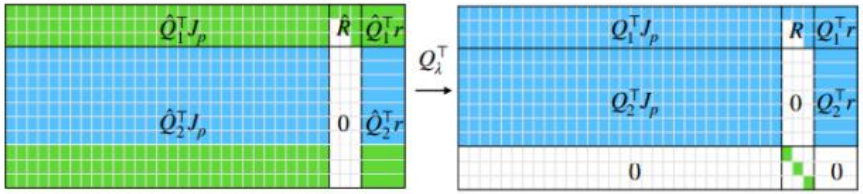
图中123456就是按顺序的6次操作对应的元素。上方i1、i2、i3是每一列对应操作的上方位置元素。

- 3、对landmark添加阻尼

```
// TODO: task 3 进行givens旋转，让storage矩阵指定位置数值为0,并存储中间结果

// Hint: makeGivens()函数可以进行givens旋转，其方法本质上就是传入 x1 和 x2，计算出旋转
//  your code line1: gr.makeGivens(???, ???)
gr.makeGivens(this->storage(i, n), this->storage(m, n));

// 对 this->storage 矩阵进行初等变换，并将连乘结果记录下来
this->storage.applyOnTheLeft(i, m, gr.adjoint());
this->Q_lambda.applyOnTheLeft(i, m, gr.adjoint());

//Fill in above this line
```

- 4、回滚

若当前次迭代无效，则需要回滚操作，更新阻尼因子等，因此利用上一步保存的Q_λ矩阵，将它的转置左乘在 block 矩阵上即可回到之前状态，对应论文图如下：

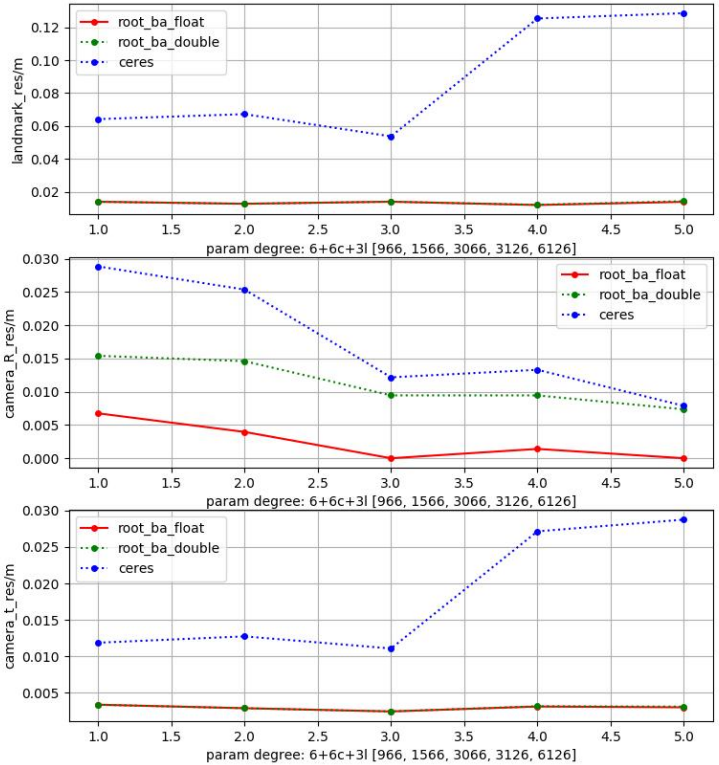



```
//  TODO: task 3 对storage矩阵实现状态回退

//  your code line1:  this->storage = ????
this->storage.noalias() = this->Q_lambda.transpose() * this->storage;

// Fill in above this line
```

- 5、测试 —— 精度分析（参数维度）

在固定观测噪声为 1/1000 的情况下，修改了参数维度，即问题规模，相机数量和路标点数量分别取（10，300）、（10，500）、（10，1000）、（20，1000）、（20，2000），进行了 5 组测试。

对比了 3 种方法——rootBA_float、rootBA_double、ceres（默认 double），结果如下图所示。三张图纵轴代表结果的误差大小，从上到下三个子图分别展示了路标点、相机姿态、相机位置的误差。其中红色为 rootBA_float、绿色为 rootBA_double、蓝色为 ceres。（这图其实不用连线，只放点就可以==）



① rootBA 的误差明显更小，在精度的表现上优于 ceres，不论是 double 还是 float 型都表现出了更高的精度；

② rootBA 在使用单精度浮点数（float）时，相较于 double，并没有损失精度，甚至可能有更优的表现，这充分体现了 rootBA 的具有很好的数值稳定性。

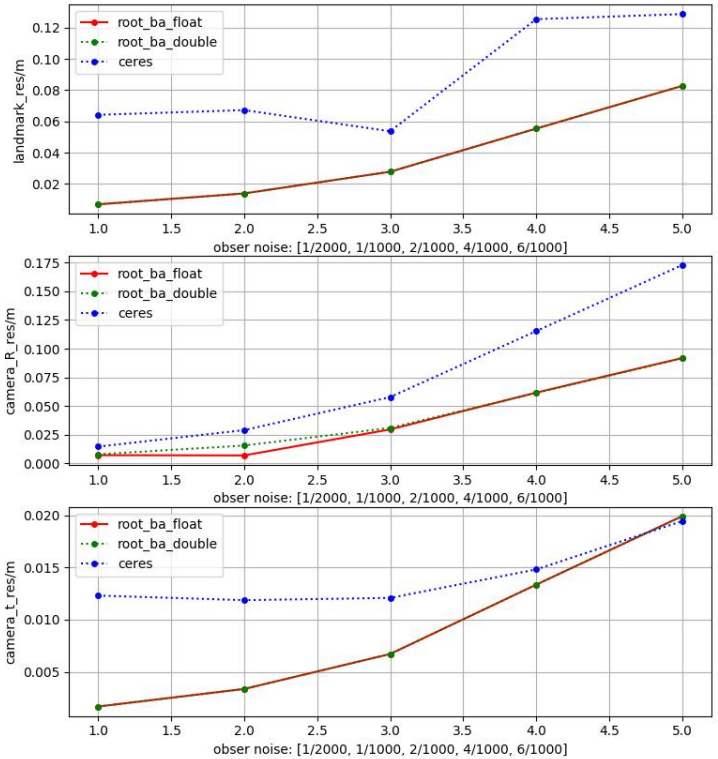
- 5、测试——精度分析（噪声大小）

在固定参数维数为（相机 10、路标点 300）的情况下，修改了观测噪声大小，观测噪声分别取 1/2000、1/1000、2/1000、4/1000、6/1000，同样进行了 5 组测试。测试结果如下图所示：

① 随着观测噪声的增大，所有方法的优化结果误差也在逐渐增大；

② 在不同观测噪声下，rootBA 的误差相较于 ceres 总体上依旧更小，具有更高的精度；

③ rootBA 在使用 float 和 double 时，在不同噪声下依旧有几乎相同的精度表现，这再次验证了 rootBA 良好的数值稳定性。



- 5、测试 —— 效率分析

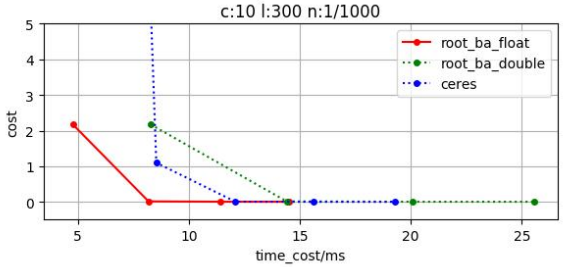
与上述精度测试类似的思路，在三种情况下进行了测试：

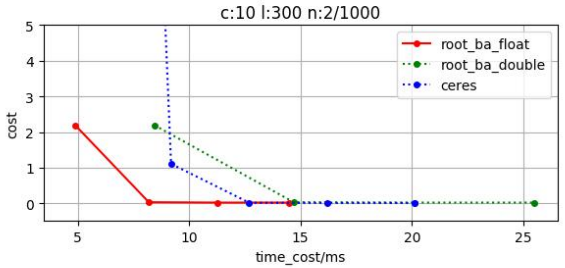
相机 10，路标点 300，观测噪声 1/1000；相机 10，路标点 300，观测噪声 2/1000；相机 10，路标点 500，观测噪声 1/1000。

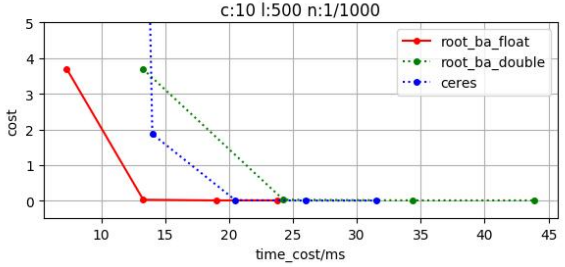
绘制了三种情况下，rootBA_float、rootBA_double、ceres 三种方法残差随时间的变化的折线图，如下图所示：

① rootBA 在使用 float 型时，相较于其他方法，收敛速度最快；

② rootBA 的 float 型相较于 double 型，速度提升了约一倍，这与论文中结论相同。







- 5、测试 —— 总结

对 rootBA 的精度和效率进行了一系列测试，除了对比使用单双精度数据的结果，还与 ceres 的 DENSE_SHUR 进行了比较，也是再次印证了论文中所提到的一些结论：

① rootBA 是一种不使用传统舒尔补方法，且不需要直接构建 H 矩阵同样可以有效且快速求解大规模 BA 问题的新方法；

② rootBA 具有良好的数值稳定性，即使在使用 float 型数据时，依旧能够达到 double 型的精度表现（不论是 rootBA 还是舒尔补）；

③ rootBA 在使用 float 型数据时，（得益于 float）效率能够大幅度提升。

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening