深蓝学院
shenlanxueyuan.com

多传感器融合讲评

主讲人 雍川

补全代码

第十章需要补全的代码主要有两大类。第一类是实现预积分、地图匹配、边缘化、帧间匹配四种优化因子。对应文件如下：

```
factor_prvag_imu_pre_integration.hpp sensor-fusion-for-localization-and-mapping\workspace\assi
    // TODO: get square root of information matrix:
    // TODO: compute residual:
    // TODO: compute jacobians:
    // TODO: correct residual by square root of information matrix:
factor_prvag_map_matching_pose.hpp sensor-fusion-for-localization-and-mapping\workspace\ass
    // TODO: get square root of information matrix:
    // TODO: compute residual:
    // TODO: compute jacobians:
    // TODO: correct residual by square root of information matrix:
factor_prvag_marginalization.hpp sensor-fusion-for-localization-and-mapping\workspace\assignme
    // TODO: Update H:
    // TODO: Update b:
    // TODO: Update H:
    // TODO: Update b:
    // TODO: Update H:
    // TODO: implement marginalization logic
    // TODO: compute residual:
    // TODO: compute jacobian:
factor_prvag_relative_pose.hpp sensor-fusion-for-localization-and-mapping\workspace\assignment
    // TODO: get square root of information matrix:
    // TODO: compute residual:
    // TODO: compute jacobians:
    // TODO: correct residual by square root of information matrix:
```

补全代码

深蓝学院
shenlanxueyuan.com

第十章需要补全的代码主要有两大类。第二类是将上述四种约束因子，加入滑窗，进行优化。对应文件如下：

```
sliding_window.cpp  sensor-fusion-for-localization-and-mapping\workspace\assignments\10-sliding-window\src\lidar_localization\src\matching...
    // TODO: add init key frame
    // TODO: add current key frame
    // TODO: add constraint, GNSS position:
    // TODO: add constraint, lidar frontend / loop closure detection:
    // TODO: add constraint, IMU pre-integraion:
ceres_sliding_window.cpp  sensor-fusion-for-localization-and-mapping\workspace\assignments\10-sliding-window\src\lidar_localization\src\m...
    // TODO: create new sliding window optimization problem:
    // TODO: a. add parameter blocks:
    // TODO: add parameter block:
    // TODO: add residual blocks:
    // TODO: b.2. map matching pose constraint:
    // TODO: add map matching factor into sliding window
    // TODO: b.3. relative pose constraint:
    // TODO: add relative pose factor into sliding window
    // TODO: b.4. IMU pre-integration constraint
    // TODO: add IMU factor into sliding window
```

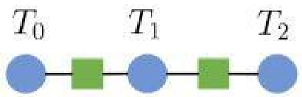
里程计约束关系

（1）里程计示意图

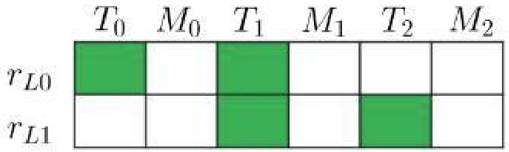
3）激光里程计相对位姿和优化变量的残差

该残差对应的因子为激光里程计因子。

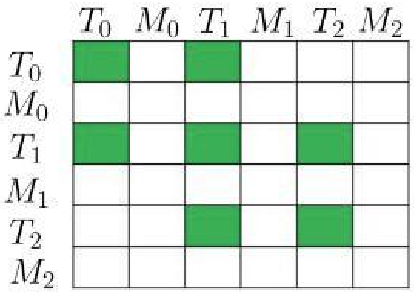
一个因子约束两个位姿，其模型如下：



残差关于优化变量的雅可比，可视化如下：



因此，对应的Hessian矩阵可视化为：



里程计约束关系

（2）里程计残差函数和雅可比矩阵

3.b. relative pose from lidar frontend.

Residual:

$$\because T_i^{-1} T_j = \begin{bmatrix} R_i^T & -R_i^T t_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_j, & t_j \\ 0, & 1 \end{bmatrix}$$

$$= \begin{bmatrix} R_i^T R_j, & R_i^T (t_j - t_i) \\ 0, & 1 \end{bmatrix}$$

$$\therefore r_p = R_i^T (t_j - t_i) - t_{obs}$$

里程计约束关系

深蓝学院 shenlanxueyuan.com

（2）里程计残差函数和雅可比矩阵

$$r_q = \ln(R_i^T R_j R_{obs}^T)^\vee$$

Jacobian, pos:

$$\frac{\partial r_p}{\partial t_i} = -R_i^T$$

$$\frac{\partial r_p}{\partial t_j} = R_i^T$$

Jacobian, ori:

$$\frac{\partial r_q}{\partial R_i} = \lim_{\phi \to 0} \frac{\ln[\exp(-\phi^\wedge) R_i^T R_j R_{obs}^T]^\vee - \ln[R_i^T R_j R_{obs}^T]^\vee}{\phi}$$

$$= \lim_{\phi \to 0} \frac{\ln[R_i^T R_j R_{obs}^T \cdot \exp(-R_{obs} R_j^T R_i \phi)^\wedge]^\vee - \ln(R_i^T R_j R_{obs}^T)^\vee}{\phi}$$

$$= -J_r^{-1}(r_q)\,\exp(r_q^\wedge).\mathrm{inverse}()$$

里程计约束关系

（2）里程计残差函数和雅可比矩阵

$$\frac{\partial r_q}{\partial R_j} = \lim_{\phi \to 0} \frac{\ln(R_i^T R_j \exp\phi^{\wedge} R_{obs}^T)^{\vee} - \ln(R_i^T R_j R_{obs}^T)^{\vee}}{\phi}$$

$$= \lim_{\phi \to 0} \frac{\ln[R_i^T R_j R_{obs}^T \cdot \exp(R_{obs}\phi)^{\wedge}]^{\vee} - \ln(R_i^T R_j R_{obs}^T)^{\vee}}{\phi}$$

$$= J_r^{-1}(r_q) R_{obs}$$

J_r should be implemented.

里程计约束关系

```
//
// TODO: get square root of information matrix:
//
Eigen::Matrix<double, 6, 6> sqrt_info = Eigen::LLT<Eigen::Matrix<double, 6, 6>>(
    I_
).matrixL().transpose();

//
// TODO: compute residual:
//
Eigen::Map<Eigen::Matrix<double, 6, 1>> residual(residuals);

residual.block(INDEX_P, startCol: 0, blockRows: 3, blockCols: 1) = ori_i.inverse() * (pos_j - pos_i) - pos_ij;
residual.block(INDEX_R, startCol: 0, blockRows: 3, blockCols: 1) = (ori_i.inverse()*ori_j*ori_ij.inverse()).log();

//
```

里程计约束关系


```cpp
// TODO: compute jacobians:
//
if ( jacobians ) {
  // compute shared intermediate results:
  const Eigen::Matrix3d R_i_inv = ori_i.inverse().matrix();
  const Eigen::Matrix3d J_r_inv = JacobianRInv( w: residual.block(INDEX_R,  startCol: 0,  blockRows: 3,  blockCols: 1));

  if ( jacobians[0] ) {
    Eigen::Map<Eigen::Matrix<double, 6, 15, Eigen::RowMajor>> jacobian_i( dataPtr: jacobians[0] );
    jacobian_i.setZero();

    jacobian_i.block<3, 3>(INDEX_P, INDEX_P) = -R_i_inv;
    jacobian_i.block<3, 3>(INDEX_R, INDEX_R) = -J_r_inv*(ori_ij*ori_j.inverse()*ori_i).matrix();

    jacobian_i = sqrt_info * jacobian_i;
  }

  if ( jacobians[1] ) {
    Eigen::Map<Eigen::Matrix<double, 6, 15, Eigen::RowMajor>> jacobian_j( dataPtr: jacobians[1]);
    jacobian_j.setZero();

    jacobian_j.block<3, 3>(INDEX_P, INDEX_P) = R_i_inv;
    jacobian_j.block<3, 3>(INDEX_R, INDEX_R) = J_r_inv*ori_ij.matrix();

    jacobian_j = sqrt_info * jacobian_j;
  }
```

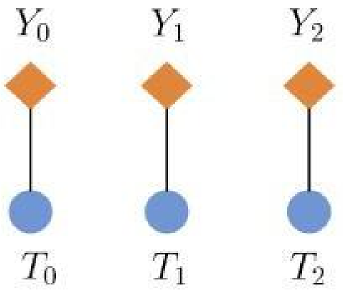
补全代码

（3）地图匹配示意图

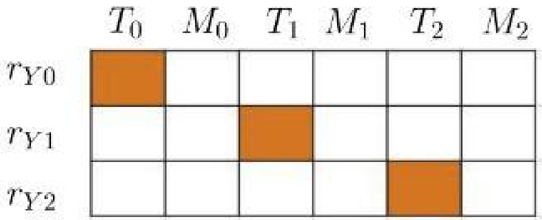
2）地图匹配位姿和优化变量的残差

该残差对应的因子为地图先验因子。

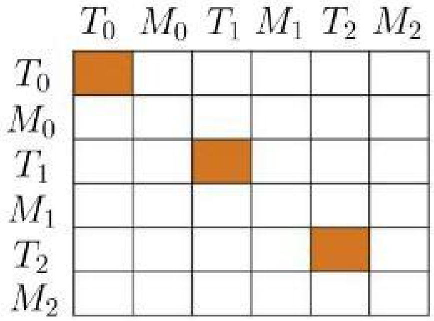
一个因子仅约束一个位姿，其模型如下：



残差关于优化变量的雅可比，可视化如下：



因此，对应的Hessian矩阵的可视化为：



地图匹配约束关系

深蓝学院
shenlanxueyuan.com

3. a. pose from map matching / GNSS position

Residual:

$$r_p = t - t_{obs}$$

$$r_q = \ln(R \cdot R_{obs}^T)^\vee$$

Jacobian:

$$\frac{\partial r_p}{\partial t} = I_3$$

地图匹配约束关系

深蓝学院
shenlanxueyuan.com

$$\frac{\partial r_q}{\partial R} = \lim_{\phi \to 0} \frac{\ln(R \exp \phi^{\wedge} R_{obs}^T)^{\vee} - \ln(R R_{obs}^T)^{\vee}}{\phi}$$

$$= \lim_{\phi \to 0} \frac{\ln(R R_{obs}^T R_{obs} \exp \phi^{\wedge} R_{obs}^T)^{\vee} - \ln(R R_{obs}^T)^{\vee}}{\phi}$$

$$= \lim_{\phi \to 0} \frac{\ln[R R_{obs}^T \cdot \exp(R_{obs}\phi)^{\wedge}]^{\vee} - \ln(R R_{obs}^T)^{\vee}}{\phi}$$

$$= \lim_{\phi \to 0} \frac{\ln(R R_{obs}^T)^{\vee} + J_r^{-1} R_{obs} \phi - \ln(R R_{obs}^T)^{\vee}}{\phi}$$

$$= J_r^{-1}(r_q) R_{obs}$$

J_r should be implemented.

地图匹配约束关系

```
//
// TODO: compute residual:
//
Eigen::Map<Eigen::Matrix<double, 6, 1>> residual(residuals);

    residual.block(INDEX_P, startCol: 0, blockRows: 3, blockCols: 1) = pos - pos_prior;
    residual.block(INDEX_R, startCol: 0, blockRows: 3, blockCols: 1) = (ori*ori_prior.inverse()).log();

//
// TODO: compute jacobians:
//
if ( jacobians ) {
  if ( jacobians[0] ) {
    Eigen::Map<Eigen::Matrix<double, 6, 15, Eigen::RowMajor> > jacobian_prior( dataPtr: jacobians[0] );
    jacobian_prior.setZero();

    jacobian_prior.block<3, 3>(INDEX_P, INDEX_P) = Eigen::Matrix3d::Identity();
    jacobian_prior.block<3, 3>(INDEX_R, INDEX_R) = JacobianRInv(
            w: residual.block(INDEX_R, startCol: 0, blockRows: 3, blockCols: 1)) * ori_prior.matrix();

    jacobian_prior = sqrt_info * jacobian_prior;
  }
}

//
```

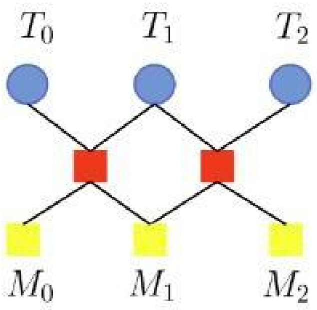
补全代码

(3). 预积分示意图

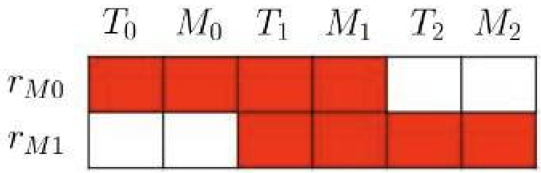
4）IMU预积分和优化变量的残差

该残差对应的因子为IMU因子。

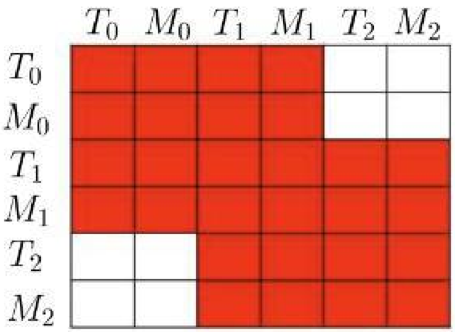
一个因子约束两个位姿，并约束两个时刻 IMU 的速度与 bias。



残差关于优化变量的雅可比，可视化如下：



因此，对应的Hessian矩阵可视化为：



预积分约束

3.c IMU pre-integration:

The residual, when parameterized using so3, is as follows:

①

$$r_p = R_i^T\left(p_j - p_i - v_i T + \tfrac{1}{2} g T^2\right) - \alpha_{ij}$$

$$\propto \; p_i, r_i, v_i, b_{ai}, b_{gi} \; / \; p_j, b_{aj}, b_{gj}$$

∴ i: $\dfrac{\partial r_p}{\partial p_i} = -R_i^T$ ∴ j: $\dfrac{\partial r_p}{\partial p_j} = R_i^T$

$$\frac{\partial r_p}{\partial r_i} = \left[R_i^T\left(p_j - p_i - v_i T + \tfrac{1}{2} g T^2\right)\right]^\wedge \qquad \frac{\partial r_p}{\partial r_j} = 0$$

预积分约束

$$\frac{\partial r_p}{\partial v_i} = -T \cdot R_i^T \qquad \frac{\partial r_p}{\partial v_j} = 0$$

$$\frac{\partial r_p}{\partial b_{ai}} = -J.\text{block}<3,3>(P,A) \qquad \frac{\partial r_p}{\partial b_{aj}} = 0$$

$$\frac{\partial r_p}{\partial b_{gi}} = -J.\text{block}<3,3>(P,G) \qquad \frac{\partial r_p}{\partial b_{gj}} = 0$$

预积分约束

②

$$r_r = \ln(R_{ij}^T R_i^T R_j)^\vee$$

$$\propto \quad r_i,\ b_{gi} \qquad r_j,\ b_{gj}$$

$$\therefore \underline{\underline{i}} \quad \frac{\partial r_r}{\partial r_i} = \lim_{\varphi \to 0} \frac{\ln[R_{ij}^T R_i^T R_j \exp(-R_j^T R_i \varphi)^\wedge]^\vee - r_r}{\varphi} = -J_r^{-1}(r_r) R_j^T R_i$$

$$\frac{\partial r_r}{\partial b_{gi}} = \lim_{\delta b_{gi} \to 0} \frac{\ln[\exp(-J_{b_{gi}}^{q} \delta b_{gi})^\wedge R_{ij}^T R_i^T R_j)]^\vee - r_r}{\delta b_{gi}}$$

$$= -J_r^{-1}(r_r)\ \exp(r_r^\wedge).\text{inverse}()\ J.\text{block}<3,3>(R,G)$$

$$\therefore \underline{\underline{j}} \quad \frac{\partial r_r}{\partial r_j} = \lim_{\varphi \to 0} \frac{\ln[R_{ij}^T R_i^T R_j \exp\varphi^\wedge]^\vee - r_r}{\varphi} = J_r^{-1}(r_r)$$

$$\frac{\partial r_r}{\partial b_{gj}} = 0$$

预积分约束

③.

$$r_v = R_i^T (v_j - v_i + gT) - \beta_{ij}$$

$$\propto r_i, v_i, b_{ai}, b_{gi} \qquad v_j, b_{aj}, b_{gj}$$

$\therefore$ i:

$$\frac{\partial r_v}{\partial r_i} = \left[R_i^T (v_j - v_i + gT)\right]^{\wedge}$$

$$\frac{\partial r_v}{\partial v_i} = -R_i^T \qquad \frac{\partial r_v}{\partial v_j} = R_i^T$$

$$\frac{\partial r_v}{\partial b_{ai}} = -J.block<3,3>(V,A) \qquad \frac{\partial r_v}{\partial b_{aj}} = 0$$

$$\frac{\partial r_v}{\partial b_{gi}} = -J.block<3,3>(V,G) \qquad \frac{\partial r_v}{\partial b_{gj}} = 0$$

预积分约束

④: $r_a = b_{a_j} - b_{a_i}$ $r_g = b_{g_j} - b_{g_i}$

$$\frac{\partial r_a}{\partial b_{a_i}} = -I \quad \frac{\partial r_a}{\partial b_{a_j}} = I \qquad \frac{\partial r_g}{\partial b_{g_i}} = -I \quad \frac{\partial r_g}{\partial b_{g_j}} = I$$

预积分约束


```
//
// TODO: compute residual:
//
Eigen::Map<Eigen::Matrix<double, 15, 1>> residual(residuals);

residual.block<3, 1>(INDEX_P, startCol: 0) = ori_i.inverse().matrix() * (pos_j - pos_i - (vel_i - 0.50 * g_ * T_) * T_) - alpha_ij;
residual.block<3, 1>(INDEX_R, startCol: 0) = (Sophus::SO3d::exp(theta_ij).inverse()*ori_i.inverse()*ori_j).log();
residual.block<3, 1>(INDEX_V, startCol: 0) = ori_i.inverse().matrix() * (vel_j - vel_i + g_ * T_) - beta_ij;
residual.block<3, 1>(INDEX_A, startCol: 0) = b_a_j - b_a_i;
residual.block<3, 1>(INDEX_G, startCol: 0) = b_g_j - b_g_i;

//
// TODO: compute jacobians:
//
if ( jacobians ) {
  // compute shared intermediate results:
  const Eigen::Matrix3d R_i_inv = ori_i.inverse().matrix();
  const Eigen::Matrix3d J_r_inv = JacobianRInv( w: residual.block(INDEX_R, startCol: 0, blockRows: 3, blockCols: 1));

  if ( jacobians[0] ) {
```

预积分约束


```
// a. residual, position:
jacobian_i.block<3, 3>(INDEX_P, INDEX_P) = -R_i_inv;
jacobian_i.block<3, 3>(INDEX_P, INDEX_R) = Sophus::SO3d::hat(
    omega: ori_i.inverse() * (pos_j - pos_i - (vel_i - 0.50 * g_ * T_) * T_)
);
jacobian_i.block<3, 3>(INDEX_P, INDEX_V) = -T_ * R_i_inv;
jacobian_i.block<3, 3>(INDEX_P, INDEX_A) = -J_.block<3,3>(INDEX_P, INDEX_A);
jacobian_i.block<3, 3>(INDEX_P, INDEX_G) = -J_.block<3,3>(INDEX_P, INDEX_G);

// b. residual, orientation:
jacobian_i.block<3, 3>(INDEX_R, INDEX_R) = -J_r_inv * (ori_j.inverse() * ori_i).matrix();
jacobian_i.block<3, 3>(INDEX_R, INDEX_G) = -J_r_inv*(
  Sophus::SO3d::exp( omega: residual.block<3, 1>(INDEX_R,  startCol: 0))
).matrix().inverse()*J_.block<3,3>(INDEX_R, INDEX_G);

// c. residual, velocity:
jacobian_i.block<3, 3>(INDEX_V, INDEX_R) = Sophus::SO3d::hat(
    omega: ori_i.inverse() * (vel_j - vel_i + g_ * T_)
);
jacobian_i.block<3, 3>(INDEX_V, INDEX_V) = -R_i_inv;
jacobian_i.block<3, 3>(INDEX_V, INDEX_A) = -J_.block<3,3>(INDEX_V, INDEX_A);
jacobian_i.block<3, 3>(INDEX_V, INDEX_G) = -J_.block<3,3>(INDEX_V, INDEX_G);

// d. residual, bias accel:
jacobian_i.block<3, 3>(INDEX_A, INDEX_A) = -Eigen::Matrix3d::Identity();

// d. residual, bias accel:
jacobian_i.block<3, 3>(INDEX_G, INDEX_G) = -Eigen::Matrix3d::Identity();

jacobian_i = sqrt_info * jacobian_i;
```

预积分约束

深蓝学院 shenlanxueyuan.com

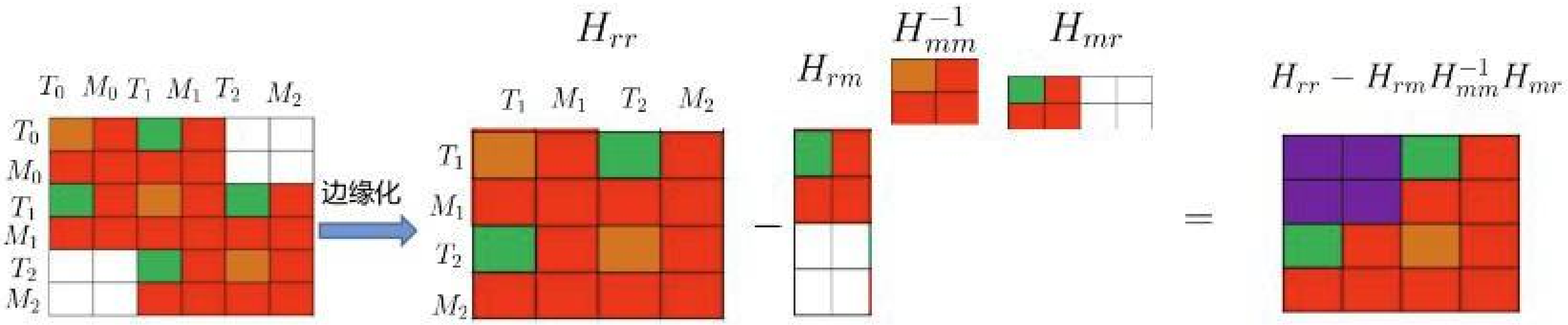
```
    if ( jacobians[1] ) {
      Eigen::Map<Eigen::Matrix<double, 15, 15, Eigen::RowMajor>> jacobian_j( dataPtr: jacobians[1]);
      jacobian_j.setZero();

      // a. residual, position:
      jacobian_j.block<3, 3>(INDEX_P, INDEX_P) = R_i_inv;

      // b. residual, orientation:
      jacobian_j.block<3, 3>(INDEX_R, INDEX_R) = J_r_inv;

      // c. residual, velocity:
      jacobian_j.block<3, 3>(INDEX_V, INDEX_V) = R_i_inv;

      // d. residual, bias accel:
      jacobian_j.block<3, 3>(INDEX_A, INDEX_A) = Eigen::Matrix3d::Identity();

      // d. residual, bias accel:
      jacobian_j.block<3, 3>(INDEX_G, INDEX_G) = Eigen::Matrix3d::Identity();

      jacobian_j = sqrt_info * jacobian_j;
    }
  }
```

补全代码

边缘化因子

1）移除老的帧

上述过程，通过可视化可以表示为



边缘化因子

Marginalization的实现理念参考关键计算步骤的推导

4. In order to adapt Marginalization to Ceres solver

a. When the sliding window has been filled, start to create Marginalization factor for next optimization

边缘化因子

深蓝学院 shenlanxueyuan.com

b. In order to fit into Ceres solver marginalization has to be implemented as follows:

$$\because \begin{cases} H_{rr} - H_{rm} H_{mm}^{-1} H_{mr} \doteq H = J^T J \\ b_r - H_{rm} H_{mm}^{-1} b_m \doteq b = -J^T r \end{cases}$$

$\because H = V \Lambda V^T$

$\therefore J = \sqrt{\Lambda} V^T$ → Marg. Res. Block. Jacobians

$\therefore r = J^{-T} b = \sqrt{\Lambda}^{-1} V^T b$ → Marg. Res. Block. Residuals.

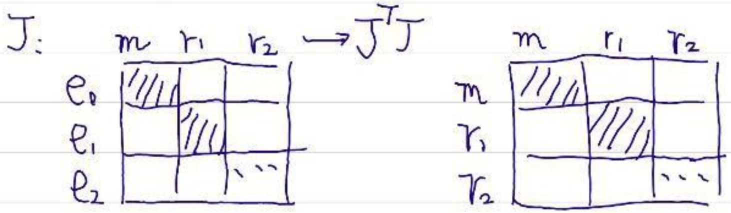
边缘化因子

c. Add the Marg. Factor directly to next Ceres problem.

Marginalization Residual Block Building, PRVAG in SO3

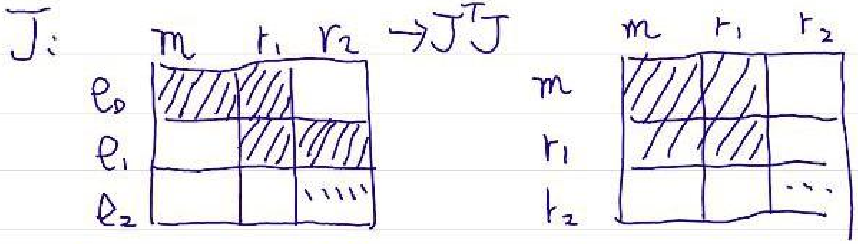
for the to-be-marginalized param block m
and its next param block r_1

a. Map Matching:



边缘化因子

b. Relative Pose & IMU Pre Integration



∴ The marginalization res. block is only relevant to param block $\underline{\underline{r_1}}$

and H_{rr}, H_{rm}, H_{mm}, H_{mr}, b_r, b_m

should be computed using res. block $\left\{ \begin{array}{l} \text{MapMatching } (m) \\ \text{RelativePose } (m, r_1) \\ \text{IMU } (m, r_1) \end{array} \right.$

边缘化因子

$$\frac{1}{2}(r+J\Delta x)^T(r+J\Delta x)=\frac{1}{2}\Delta x^T J^T J\Delta x+r^T J\Delta x+r^T r$$

$$J^T J\Delta x+J^T r=0$$

$$J^T J\Delta x=-J^T r$$

Sliding Window Marginalization

```
//
// TODO: Update H:
//
// a. H_mm:
H_.block<15, 15>(INDEX_M, INDEX_M) += J_m.transpose() * J_m;

//
// TODO: Update b:
//
// a. b_m:
b_.block<15,  1>(INDEX_M,      startCol: 0) += J_m.transpose() * residuals;
```

```
//
// TODO: Update H:
//
// a. H_mm:
H_.block<15, 15>(INDEX_M, INDEX_M) += J_m.transpose() * J_m;
// b. H_mr:
H_.block<15, 15>(INDEX_M, INDEX_R) += J_m.transpose() * J_r;
// c. H_rm:
H_.block<15, 15>(INDEX_R, INDEX_M) += J_r.transpose() * J_m;
// d. H_rr:
H_.block<15, 15>(INDEX_R, INDEX_R) += J_r.transpose() * J_r;


//
// TODO: Update b:
//
// a. b_m:
b_.block<15,  1>(INDEX_M,      startCol: 0) += J_m.transpose() * residuals;
// a. b_r:
b_.block<15,  1>(INDEX_R,      startCol: 0) += J_r.transpose() * residuals;
```

Sliding Window Marginalization

```
//
// TODO: Update H:
//
// a. H_mm:
H_.block<15, 15>(INDEX_M, INDEX_M) += J_m.transpose() * J_m;
// b. H_mr:
H_.block<15, 15>(INDEX_M, INDEX_R) += J_m.transpose() * J_r;
// c. H_rm:
H_.block<15, 15>(INDEX_R, INDEX_M) += J_r.transpose() * J_m;
// d. H_rr:
H_.block<15, 15>(INDEX_R, INDEX_R) += J_r.transpose() * J_r;


//
// Update b:
//
// a. b_m:
b_.block<15,  1>(INDEX_M,       startCol: 0) += J_m.transpose() * residuals;
// a. b_r:
b_.block<15,  1>(INDEX_R,       startCol: 0) += J_r.transpose() * residuals;
```

Sliding Window Marginalization


```cpp
  // TODO: implement marginalization logic
// save x_m_0:
Eigen::Map<const Eigen::Matrix<double, 15, 1>> x_0(raw_param_r_0);
x_0_ = x_0;
// marginalize:
const Eigen::MatrixXd &H_mm = H_.block<15, 15>(INDEX_M, INDEX_M);
const Eigen::MatrixXd &H_mr = H_.block<15, 15>(INDEX_M, INDEX_R);
const Eigen::MatrixXd &H_rm = H_.block<15, 15>(INDEX_R, INDEX_M);
const Eigen::MatrixXd &H_rr = H_.block<15, 15>(INDEX_R, INDEX_R);

const Eigen::VectorXd &b_m = b_.block<15, 1>(INDEX_M,  startCol: 0);
const Eigen::VectorXd &b_r = b_.block<15, 1>(INDEX_R,  startCol: 0);

Eigen::MatrixXd H_mm_inv = H_mm.inverse();
Eigen::MatrixXd H_marginalized = H_rr - H_rm * H_mm_inv * H_mr;
Eigen::MatrixXd b_marginalized = b_r - H_rm * H_mm_inv * b_m;
// solve linearized residual & Jacobian:
Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> saes(H_marginalized);
Eigen::VectorXd S = Eigen::VectorXd(
    x: (saes.eigenvalues().array() > 1.0e-5).select(saes.eigenvalues().array(),  elseScalar: 0)
);
Eigen::VectorXd S_inv = Eigen::VectorXd(
    x: (saes.eigenvalues().array() > 1.0e-5).select(saes.eigenvalues().array().inverse(),  elseScalar: 0)
);

Eigen::VectorXd S_sqrt = S.cwiseSqrt();
Eigen::VectorXd S_inv_sqrt = S_inv.cwiseSqrt();

J_ = S_sqrt.asDiagonal() * saes.eigenvectors().transpose();
e_ = S_inv_sqrt.asDiagonal() * saes.eigenvectors().transpose() * b_marginalized;
```

Sliding Window Marginalization


```
//
// parse parameters:
//
Eigen::Map<const Eigen::Matrix<double, 15, 1>> x( dataPtr: parameters[0]);
Eigen::VectorXd dx = x - x_0_;

//
// TODO: compute residual:
//
Eigen::Map<Eigen::Matrix<double, 15, 1>> residual(residuals);
residual = e_ + J_ * dx;

//
// TODO: compute jacobian:
//
if ( jacobians ) {
  if ( jacobians[0] ) {
    Eigen::Map<Eigen::Matrix<double, 15, 15, Eigen::RowMajor> > jacobian_marginalization( dataPtr: jacobians[0] );
    jacobian_marginalization.setZero();

    jacobian_marginalization = J_;
  }
}
```

sliding window optimization problem


```
// TODO: create new sliding window optimization problem:
ceres::Problem problem;

// TODO: a. add parameter blocks:
for ( int i = 1; i <= kWindowSize + 1; ++i) {
    auto &target_key_frame = optimized_key_frames_.at( n: N - i);

    ceres::LocalParameterization *local_parameterization = new sliding_window::ParamPRVAG();
    // TODO: add parameter block:
    problem.AddParameterBlock(target_key_frame.prvag,  size: 15, local_parameterization);

    if ( target_key_frame.fixed ) {
        problem.SetParameterBlockConstant(target_key_frame.prvag);
    }
}
```

sliding window optimization problem


```
// TODO: add residual blocks:
// b.1. marginalization constraint:
if (
    !residual_blocks_.map_matching_pose.empty() &&
    !residual_blocks_.relative_pose.empty() &&
    !residual_blocks_.imu_pre_integration.empty()
) {
    auto &key_frame_m = optimized_key_frames_.at( n: N - kWindowSize - 1);
    auto &key_frame_r = optimized_key_frames_.at( n: N - kWindowSize - 0);

    const ceres::CostFunction *factor_map_matching_pose = GetResMapMatchingPose(
        res_map_matching_pose: residual_blocks_.map_matching_pose.front()
    );
    const ceres::CostFunction *factor_relative_pose = GetResRelativePose(
        res_relative_pose: residual_blocks_.relative_pose.front()
    );
    const ceres::CostFunction *factor_imu_pre_integration = GetResIMUPreIntegration(
        res_imu_pre_integration: residual_blocks_.imu_pre_integration.front()
    );

    sliding_window::FactorPRVAGMarginalization *factor_marginalization = new sliding_window::FactorPRVAGMarginalization();

    factor_marginalization->SetResMapMatchingPose(
        factor_map_matching_pose,
        parameter_blocks: std::vector<double *>{key_frame_m.prvag}
```

sliding window optimization problem

```
factor_marginalization->SetResRelativePose(
    factor_relative_pose,
    parameter_blocks: std::vector<double *>{key_frame_m.prvag, key_frame_r.prvag}
);
factor_marginalization->SetResIMUPreIntegration(
    factor_imu_pre_integration,
    parameter_blocks: std::vector<double *>{key_frame_m.prvag, key_frame_r.prvag}
);
factor_marginalization->Marginalize(key_frame_r.prvag);

// add marginalization factor into sliding window
problem.AddResidualBlock(
    factor_marginalization,
    loss_function: NULL,
    key_frame_r.prvag
);

residual_blocks_.map_matching_pose.pop_front();
residual_blocks_.relative_pose.pop_front();
residual_blocks_.imu_pre_integration.pop_front();
}
```

sliding window optimization problem

```
// TODO: b.2. map matching pose constraint:
if ( !residual_blocks_.map_matching_pose.empty() ) {
    for ( const auto &residual_map_matching_pose: residual_blocks_.map_matching_pose ) {
        auto &key_frame = optimized_key_frames_.at(residual_map_matching_pose.param_index);

        sliding_window::FactorPRVAGMapMatchingPose *factor_map_matching_pose = GetResMapMatchingPose(
            residual_map_matching_pose
        );

        // TODO: add map matching factor into sliding window
        problem.AddResidualBlock(
            factor_map_matching_pose,
            loss_function: NULL,
            key_frame.prvag
        );
    }
}

// TODO: b.3. relative pose constraint:
```

sliding window optimization problem

```
// TODO: b.3. relative pose constraint:
if ( !residual_blocks_.relative_pose.empty() ) {
    for ( const auto &residual_relative_pose: residual_blocks_.relative_pose ) {
        auto &key_frame_i = optimized_key_frames_.at(residual_relative_pose.param_index_i);
        auto &key_frame_j = optimized_key_frames_.at(residual_relative_pose.param_index_j);

        sliding_window::FactorPRVAGRelativePose *factor_relative_pose = GetResRelativePose(
            residual_relative_pose
        );

        // TODO: add relative pose factor into sliding window
        problem.AddResidualBlock(
            factor_relative_pose,
            loss_function: NULL,
            key_frame_i.prvag, key_frame_j.prvag
        );
    }
}
```

sliding window optimization problem

深蓝学院
shenlanxueyuan.com

```
// TODO: b.4. IMU pre-integration constraint
if ( !residual_blocks_.imu_pre_integration.empty() ) {
    for ( const auto &residual_imu_pre_integration: residual_blocks_.imu_pre_integration ) {
        auto &key_frame_i = optimized_key_frames_.at(residual_imu_pre_integration.param_index_i);
        auto &key_frame_j = optimized_key_frames_.at(residual_imu_pre_integration.param_index_j);

        sliding_window::FactorPRVAGIMUPreIntegration *factor_imu_pre_integration = GetResIMUPreIntegration(
            residual_imu_pre_integration
        );

        // TODO: add IMU factor into sliding window
        problem.AddResidualBlock(
            factor_imu_pre_integration,
            loss_function: NULL,
            key_frame_i.prvag, key_frame_j.prvag
        );
    }
}

// solve:
```

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening