

## 1. 推导雅克比矩阵

对号是已推的，剩下的很简单 只有一个  $p$  对  $\theta$  的求导，参考  $q$  对  $\theta$  的也很简单。

$$\begin{aligned}
 r_p &= q_{wbi}^* (p_{wbj} - p_{wbi} - v_i^w \cdot \Delta t + \frac{1}{2} q_{wbi}^w \Delta t^2) - \Delta b_{bi} \\
 r_q &= \frac{1}{2} [q_{wbi}^* \otimes (q_{wbi}^* \otimes q_{wbi}^*)] \Delta t^2 \\
 r_v &= q_{wbi}^* (v_i^w - v_i^w + q_{wbi}^w \cdot \Delta t) - p_{bi} - b_j \\
 r_{ba} &= b_i^a - b_j^a \\
 r_{bg} &= b_i^g - b_j^g \\
 x &= [s p_{wbi}, s \theta_{wbi}, s v_i^w, s b_i^a, s b_i^g, s p_{wbi}, s \theta_{wbi}, s v_i^w, s b_j^a, s b_j^g] \\
 \begin{aligned}
 s r_q & \quad 0 \quad \checkmark \quad 0 \quad 0 \quad \checkmark \quad 0 \quad \checkmark \quad 0 \quad 0 \quad 0 \\
 s r_{pv} & \quad 0 \quad \checkmark \quad \checkmark \quad \checkmark \quad \checkmark \quad 0 \quad 0 \quad q_{wbi}^* \quad 0 \quad 0 \\
 s r_p & \quad -q_{wbi}^* \quad \Delta t \quad -q_{wbi}^* \cdot \Delta t \quad -J_{b_i^a}^B \quad -J_{b_i^g}^B \quad q_{wbi}^* \quad 0 \quad 0 \quad 0 \quad 0 \\
 s b_a & \quad 0 \quad 0 \quad 0 \quad -I \quad 0 \quad 0 \quad 0 \quad 0 \quad I \quad 0 \\
 s b_g & \quad 0 \quad 0 \quad 0 \quad 0 \quad -I \quad 0 \quad 0 \quad 0 \quad 0 \quad I
 \end{aligned} \\
 \frac{s r_v}{s b_i^g} &= \frac{-\partial p_{bi}}{\partial b_i^g} = -J_{b_i^g}^B \quad \frac{s r_v}{s v_i^w} = q_{wbi}^* \\
 \frac{s r_p}{\partial \theta_{wbi}} &= \frac{\partial (q_{wbi}^* \otimes [\frac{1}{2} s \theta_{wbi}])}{\partial \theta_{wbi}} (p_{wbj} - p_{wbi} - v_i^w \cdot \Delta t + \frac{1}{2} q_{wbi}^w \Delta t^2) \\
 &= [R_{biw} (p_{wbj} - p_{wbi} - v_i^w \cdot \Delta t + \frac{1}{2} q_{wbi}^w \Delta t^2)]_x \\
 &\text{此处与 } \frac{\partial r_v}{\partial \theta_{wbi}} \text{ 类似}
 \end{aligned}$$

## 2. 实现代码：

主要是预积分的过程和优化 error 的设计

```

Eigen::Matrix3d dp_dba = J_.block<3, 3>(INDEX_P, INDEX_A);
Eigen::Matrix3d dp_dbg = J_.block<3, 3>(INDEX_P, INDEX_G);

Eigen::Matrix3d dq_dbg = J_.block<3, 3>(INDEX_R, INDEX_G);

Eigen::Matrix3d dv_dba = J_.block<3, 3>(INDEX_V, INDEX_A);
Eigen::Matrix3d dv_dbg = J_.block<3, 3>(INDEX_V, INDEX_G);

//
// TODO: update pre-integration measurement caused by bias change:
if(v0->isUpdated())
{
    Eigen::Vector3d ba,bg;
    v0->getDeltaBiases(ba,bg);
    updateMeasurement(ba,bg);
}
//
// J_.resize(15,30);
// J_.setZero();
// J_.block<3, 3>(INDEX_P, INDEX_P) = -ori_i.inverse().matrix();
// J_.block<3, 3>(INDEX_P, INDEX_R) = Sophus::S03d::hat(ori_i.inverse().matrix()*(pos_j-pos_i-vel_i*T+0.5*g*T*T));
// J_.block<3, 3>(INDEX_P, INDEX_V) = -ori_i.inverse().matrix()*T_;
// J_.block<3, 3>(INDEX_P, INDEX_A) = -dp_dba;
// J_.block<3, 3>(INDEX_P, INDEX_G) = -dp_dbg;
// J_.block<3, 3>(INDEX_P, INDEX_P+15) = -ori_j.inverse().matrix();

//
// TODO: compute error:
//
_error.block<3, 1>(INDEX_P, 0) = Eigen::Vector3d::Zero();
_error.block<3, 1>(INDEX_R, 0) = Eigen::Vector3d::Zero();
_error.block<3, 1>(INDEX_V, 0) = Eigen::Vector3d::Zero();
_error.block<3, 1>(INDEX_A, 0) = Eigen::Vector3d::Zero();
_error.block<3, 1>(INDEX_G, 0) = Eigen::Vector3d::Zero();

_error.block<3, 1>(INDEX_P, 0) = ori_i.inverse().matrix()*(pos_j-pos_i-vel_i*T+0.5*g*T*T)-measurement.block<3, 1>(INDEX_P, 0);
_error.block<3, 1>(INDEX_R, 0) = 2*Eigen::Quaterniond(Sophus::S03d::exp(measurement.block<3, 1>(INDEX_R, 0)).inverse().matrix()*ori_i.inverse().matrix()*ori_j.inverse().matrix());
_error.block<3, 1>(INDEX_V, 0) = ori_i.inverse()*(vel_j-vel_i+g*T)-measurement.block<3, 1>(INDEX_V, 0);
_error.block<3, 1>(INDEX_A, 0) = b_a_j-b_a_i;
_error.block<3, 1>(INDEX_G, 0) = b_g_j-b_g_i;

```

## 顶点的更新

```

virtual void opusImpl(const double *update) override {
    //
    // TODO: do update
    //
    estimate.pos+=Eigen::Vector3d(update[PRVAG::INDEX_POS+0],update[PRVAG::INDEX_POS+1],update[PRVAG::INDEX_POS+2]);
    estimate.ori= estimate.ori*Sophus::S03d::exp(Eigen::Vector3d(update[PRVAG::INDEX_ORI+0],update[PRVAG::INDEX_ORI+1],update[PRVAG::INDEX_ORI+2]));
    estimate.vel+=Eigen::Vector3d(update[PRVAG::INDEX_VEL+0],update[PRVAG::INDEX_VEL+1],update[PRVAG::INDEX_VEL+2]);
    estimate.b_a+=Eigen::Vector3d(update[PRVAG::INDEX_B_A+0],update[PRVAG::INDEX_B_A+1],update[PRVAG::INDEX_B_A+2]);
    estimate.b_g+=Eigen::Vector3d(update[PRVAG::INDEX_B_G+0],update[PRVAG::INDEX_B_G+1],update[PRVAG::INDEX_B_G+2]);
    updateDeltaBiases(Eigen::Vector3d(update[PRVAG::INDEX_B_A+0],update[PRVAG::INDEX_B_A+1],update[PRVAG::INDEX_B_A+2]),
        Eigen::Vector3d(update[PRVAG::INDEX_B_G+0],update[PRVAG::INDEX_B_G+1],update[PRVAG::INDEX_B_G+2]));
}

```

```

void IMUPreIntegrator::UpdateState(void) {
    static double T = 0.0;

    static Eigen::Vector3d w_mid = Eigen::Vector3d::Zero();
    static Eigen::Vector3d a_mid = Eigen::Vector3d::Zero();

    static Sophus::S03d prev_theta_ij = Sophus::S03d();
    static Sophus::S03d curr_theta_ij = Sophus::S03d();
    static Sophus::S03d d_theta_ij = Sophus::S03d();

    static Eigen::Matrix3d dR_inv = Eigen::Matrix3d::Zero();
    static Eigen::Matrix3d prev_R = Eigen::Matrix3d::Zero();
    static Eigen::Matrix3d curr_R = Eigen::Matrix3d::Zero();
    static Eigen::Matrix3d prev_R_a_hat = Eigen::Matrix3d::Zero();
    static Eigen::Matrix3d curr_R_a_hat = Eigen::Matrix3d::Zero();

    //
    // parse measurements:
    //
    // get measurement handlers:
    const IMUData &prev_imu_data = imu_data_buff_.at(0);
    const IMUData &curr_imu_data = imu_data_buff_.at(1);

    // get time delta:
    T = curr_imu_data.time - prev_imu_data.time;
}

```

```

// get measurements:
const Eigen::Vector3d prev_w(
    prev_imu_data.angular_velocity.x - state.b_g_i_.x(),
    prev_imu_data.angular_velocity.y - state.b_g_i_.y(),
    prev_imu_data.angular_velocity.z - state.b_g_i_.z()
);
const Eigen::Vector3d curr_w(
    curr_imu_data.angular_velocity.x - state.b_g_i_.x(),
    curr_imu_data.angular_velocity.y - state.b_g_i_.y(),
    curr_imu_data.angular_velocity.z - state.b_g_i_.z()
);

const Eigen::Vector3d prev_a(
    prev_imu_data.linear_acceleration.x - state.b_a_i_.x(),
    prev_imu_data.linear_acceleration.y - state.b_a_i_.y(),
    prev_imu_data.linear_acceleration.z - state.b_a_i_.z()
);
const Eigen::Vector3d curr_a(
    curr_imu_data.linear_acceleration.x - state.b_a_i_.x(),
    curr_imu_data.linear_acceleration.y - state.b_a_i_.y(),
    curr_imu_data.linear_acceleration.z - state.b_a_i_.z()
);

// 1. get w_mid:
w_mid = 0.5 * ( prev_w + curr_w );
// 2. update relative orientation, so3:
prev_theta_ij = state.theta_ij_;
d_theta_ij = Sophus::S03d::exp(w_mid * T);
state.theta_ij_ = state.theta_ij_ * d_theta_ij;
curr_theta_ij = state.theta_ij_;
// 3. get a_mid:
a_mid = 0.5 * ( prev_theta_ij*prev_a + prev_theta_ij*curr_a );
// 4. update relative translation:
state.alpha_ij_ += state.beta_ij_ * T + 0.5*a_mid*T*T;
// 5. update relative velocity:
state.beta_ij_ += a_mid*T;

//
// TODO0: b. update covariance:
//
// 1. intermediate results:
dR_inv = d_theta_ij.inverse().matrix();
prev_R = prev_theta_ij.matrix();
curr_R = curr_theta_ij.matrix();
prev_R_a_hat = prev_R * Sophus::S03d::hat(prev_a);
curr_R_a_hat = curr_R * Sophus::S03d::hat(curr_a);

//
// TODO0: 2. set up F:
//
// F12 & F32:
F = MatrixF::Identity();
F.block<3,3>(INDEX_ALPHA, INDEX_THETA) += -0.25*T*T*(prev_R_a_hat + curr_R_a_hat*(Eigen::Matrix3d::Identity() - Sophus::S03d::hat(w_mid)*T));
F.block<3,3>(INDEX_BETA, INDEX_THETA) += -0.5*T*(prev_R_a_hat + curr_R_a_hat*(Eigen::Matrix3d::Identity() - Sophus::S03d::hat(w_mid)*T));
// F14 & F34:
F.block<3,3>(INDEX_ALPHA, INDEX_BETA) += Eigen::Matrix3d::Identity()*T;
F.block<3,3>(INDEX_ALPHA, INDEX_B_A) += -0.25*T*T*(prev_R + curr_R);
// F15 & F35:
F.block<3,3>(INDEX_ALPHA, INDEX_B_G) += 0.25*T*T*T*curr_R*Sophus::S03d::hat(curr_a);
F.block<3,3>(INDEX_THETA, INDEX_B_G) += -Eigen::Matrix3d::Identity()*T;
F.block<3,3>(INDEX_BETA, INDEX_B_A) += -0.5*T*(prev_R + curr_R);
F.block<3,3>(INDEX_BETA, INDEX_B_G) += 0.5*T*T*curr_R*Sophus::S03d::hat(curr_a);

// F22:
F.block<3,3>(INDEX_THETA, INDEX_THETA) += Eigen::Matrix3d::Identity() - Sophus::S03d::hat(w_mid)*T;

```



```

B_ = MatrixB::Zero();
B_.block<3,3>(INDEX_ALPHA, INDEX_M_ACC_PREV) = 0.25 * prev_R * T;
B_.block<3,3>(INDEX_ALPHA, INDEX_M_GYR_PREV) = -0.125 * curr_R * T * T * Sophus::S03d::hat(curr_a);
B_.block<3,3>(INDEX_ALPHA, INDEX_M_ACC_CURR) = 0.25 * curr_R * T;
B_.block<3,3>(INDEX_ALPHA, INDEX_M_GYR_CURR) = -0.125 * curr_R * T * T * Sophus::S03d::hat(curr_a);

B_.block<3,3>(INDEX_THETA, INDEX_M_GYR_PREV) = 0.5 * Eigen::Matrix3d::Identity() * T;
B_.block<3,3>(INDEX_THETA, INDEX_M_GYR_CURR) = 0.5 * Eigen::Matrix3d::Identity() * T;

B_.block<3,3>(INDEX_BETA, INDEX_M_ACC_PREV) = 0.5 * prev_R * T;
B_.block<3,3>(INDEX_BETA, INDEX_M_GYR_PREV) = -0.25 * curr_R * T * T * Sophus::S03d::hat(curr_a);
B_.block<3,3>(INDEX_BETA, INDEX_M_ACC_CURR) = 0.5 * curr_R * T;
B_.block<3,3>(INDEX_BETA, INDEX_M_GYR_CURR) = -0.25 * curr_R * T * T * Sophus::S03d::hat(curr_a);

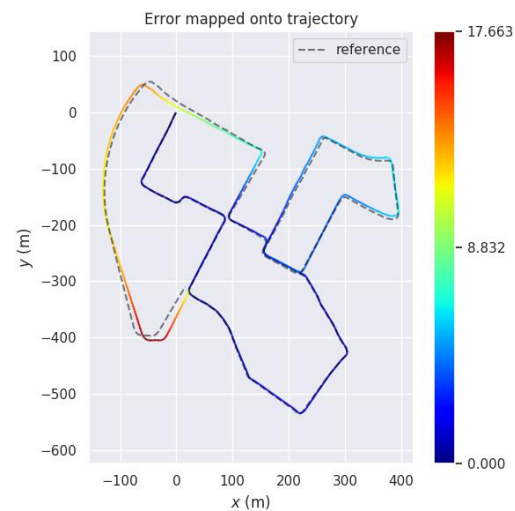
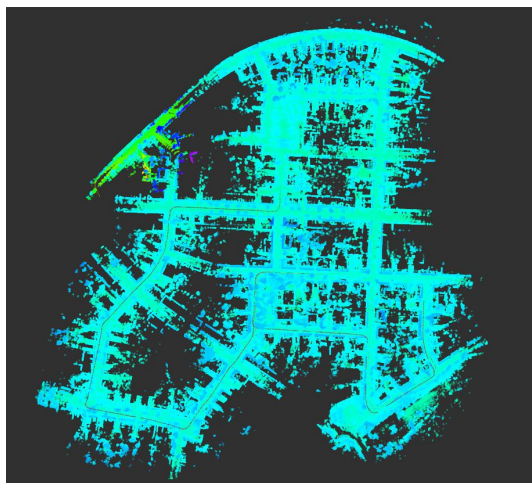
B_.block<3,3>(INDEX_B_A, INDEX_R_ACC_PREV) = Eigen::Matrix3d::Identity() * T;
B_.block<3,3>(INDEX_B_G, INDEX_R_GYR_PREV) = Eigen::Matrix3d::Identity() * T;

// G11 & G31:
// G12 & G32:
// G13 & G33:
// G14 & G34:

// TODO0: 4. update P_:
P = F_ * P_ * F_.transpose() + B_ * Q_ * B_.transpose();
// ;
// TODO0: 5. update Jacobian:
//
J_ = F_ * J_;

```

Gnss、loop、imu\_pre、odom\_pre 的开关在 config 中



有 imu 积分时

```

max 17.577519
mean 4.669541
median 3.168825
min 0.000001
rmse 6.336322
sse 59259.885910
std 4.283032

```

无 imu 积分时

```

max 17.663317

```

```
mean 5.082181
median 3.510857
min 0.000001
rmse 6.896013
sse 73234.693898
std 4.661162
```

有 imu 还是 rmse 稍小点，看了 error 随时间的变化，没发现加 imu 应该有的路段会平滑的现象。之后加了码盘再试试看

3、推导编码器有了 imu 的经验，也会很简单，跟加速度相关项去掉差不多就一致了

然而 \$S\$ 矩阵: 线性: 
$$\begin{bmatrix} r_r \\ r_\theta \\ r_{bg} \end{bmatrix} = \begin{bmatrix} q_{rbs1} (P_{rbs1} - P_{rbs1}) - d_{rbs1} \\ 2 \times (q_{rbs1} \otimes q_{rbs1} + q_{rbs1} \otimes q_{rbs1}) + 4s \\ \frac{1}{2} - \frac{1}{2} \end{bmatrix}$$

若 \$b\$ 为 \$S\$ 矩阵:

$$2. \quad \delta \theta_k^g = -[w_k - b_{rbs1}] \times \delta \theta_k^g + \pi w_k - \delta w_k$$

$$X_{k+1} = F_k \cdot X_k + G_k \cdot \frac{N_k}{P_k}$$

$$\begin{matrix} \delta \theta_{k+1} & \delta x_k & \pi w_k \\ \delta \theta_k & \delta \theta_k & \pi w_{k+1} \\ \delta b_{k+1} & \delta b_k & \pi b_k^g \end{matrix}$$

$$1. \quad \delta \alpha_{k+1} = \tilde{\alpha}_{k+1} - \alpha_{k+1} \\ = \alpha_{k+1} + \delta \alpha_k - \alpha_{k+1}$$

$$\alpha_{k+1} + \delta \alpha_{k+1} = \alpha_{k+1} + \delta \alpha_k + R_{k+1} [I + \delta \theta_k] (\phi_k + \pi \phi_k)$$

$$\alpha_{k+1} + R_{k+1} \cdot \phi_k + \delta \alpha_{k+1} = \dots$$

$$\Rightarrow \alpha_{k+1} = \alpha_{k+1} - R_{k+1} [\phi_k] \times \delta \theta_k + R_{k+1} \pi \phi_k$$

$$3. \quad \delta b_k^g = \pi g$$

$$4. \quad \delta \theta_k = - \left[ \frac{w_k + w_{k+1}}{2} - b_k^g \right] \times \delta \theta_k + \pi w_k - \delta b_k^g$$

$$\Rightarrow \delta \theta_{k+1} = [I - [\tilde{w}_k] \delta t] \delta \theta_k + \frac{1}{2} \delta t \pi w_k + \frac{1}{2} \delta t \pi w_{k+1} - \delta t \cdot \delta b_k^g$$

$$5. \quad \delta \alpha_{k+1} = \delta \alpha_k - R_k [\phi_k] \times \delta \theta_k + R_k \pi \phi_k$$

$$6. \quad \delta b_{k+1}^g = \delta b_k^g + \pi b_k^g \delta t$$

$$F = \begin{bmatrix} I & -R_k [\phi_k] \times & 0 \\ 0 & I - [\tilde{w}_k] \delta t & -I \cdot \delta t \\ 0 & 0 & I \end{bmatrix} \quad G_k = \begin{bmatrix} R_k & 0 & 0 & 0 \\ 0 & \frac{1}{2} I \delta t & \frac{1}{2} I \delta t & 0 \\ 0 & 0 & 0 & I \delta t \end{bmatrix}$$

$$\text{例} \quad \begin{bmatrix} r_0 \\ r_q \\ r_{bg} \end{bmatrix} = \begin{bmatrix} q_{wb}^* (P_{wb} - P_{wb}) - \alpha b_i b_i \\ \frac{1}{2} (q_{wb}^* \otimes q_{wb}^* \otimes q_{wb}^*) \cdot y_2 \\ b_i - y_i \end{bmatrix}$$

$$x = [SP_{wb}, SQ_{wb}, SB_i^s, SP_{wb}, SQ_{wb}, SB_i^q]$$

$$r_q: -P_{bw}, RF, \frac{1}{2} P_{bw} \quad 0 \quad 0$$

$$r_g: 0 \quad RF, RF \quad 0 \quad RF \quad 0$$

$$r_{bg}: 0 \quad 0 \quad -I \quad 0 \quad 0 \quad I$$

推导基本与 LRU 预积分类似:

$$\frac{\partial r}{\partial S_{bi}^s} = \frac{\partial (P_{wb} \otimes [\frac{1}{2} S_{bi}^s])^T (P_{wb} - P_{wb})}{\partial S_{bi}^s}$$

$$= \frac{\partial (P_{wb} \cdot \frac{\partial (S_{bi}^s)}{\partial S_{bi}^s})^T (P_{wb} - P_{wb})}{\partial S_{bi}^s}$$

$$= \frac{\partial (I - [S_{bi}^s] \otimes P_{bw})^T (P_{wb} - P_{wb})}{\partial S_{bi}^s} (A \cdot B = B \cdot A)$$

$$= (P_{bw} - P_{wb} - P_{wb})^T x$$

$$\frac{\partial r}{\partial S_{bi}^s} = -2[0, I] \begin{bmatrix} q_{wb}^* \otimes q_{wb} \\ q_{wb}^* \otimes q_{wb} \end{bmatrix} L \begin{bmatrix} q_{bi}^* \\ \frac{1}{2} I \end{bmatrix} \text{ 与 } S_{bi}^s \text{ 相关} - 2x$$

$$\frac{\partial r}{\partial S_{bi}^q} = -2[0, I] \begin{bmatrix} q_{wb}^* \otimes q_{wb} \otimes q_{wb}^* \\ q_{wb}^* \otimes q_{wb} \end{bmatrix} L \begin{bmatrix} 0 \\ \frac{1}{2} I \end{bmatrix} \text{ 与 } S_{bi}^q \text{ 相关} - 2x$$

$$\frac{\partial r}{\partial S_{bi}^q} = -2[0, I] \begin{bmatrix} q_{wb}^* \otimes q_{wb} \otimes q_{bi}^* \\ q_{wb}^* \otimes q_{wb} \end{bmatrix} L \begin{bmatrix} 0 \\ \frac{1}{2} I \end{bmatrix} - 2x$$