VINS-Mono 理解

Extr15

2017.12.06

1 ProjectionFactor

```
//这里 2，7，7，7，1 分别是残差、(Pi，Qi)，(Pj，Qj)，(tic，qic)的维数， tic 是 imu_t_camera，qic是imu_R_camera
class ProjectionFactor : public ceres::SizedCostFunction<2, 7, 7, 7, 1>
```

把某个 landmark 在 c_i 坐标系下的坐标 $X_l^{c_i}$ 转换成第 j 个相机坐标系 c_j 下的坐标 $X_l^{c_j}$ 的公式为:

$$
\begin{aligned}
X_l^{c_j} &= R_b^c\left[R_w^{b_j}\left[R_{b_i}^w\left[R_c^b X_l^{c_i}+t_c^b\right]+t_{c_i}^w\right]+t_w^{c_j}\right]+t_b^c && (1)\\
&=(R_c^b)^{-1}\left[R_j^{-1}\left[R_i\left[R_c^b X_l^{c_i}+t_c^b\right]+P_i-P_j\right]-t_c^b\right] && (2)\\
&=(R_c^b)^{-1}\left[R_j^{-1}\left[R_i\left[X_l^{b_i}\right]+P_i-P_j\right]-t_c^b\right] && (3)\\
&=(R_c^b)^{-1}\left[R_j^{-1}\left[X_l^w-P_j\right]-t_c^b\right] && (4)\\
&=(R_c^b)^{-1}\left[X_l^{b_j}-t_c^b\right] && (5)\\
& && (6)
\end{aligned}
$$

上式中为了区分，用 b 来表示 imu, R_c^b 就是代码中的 ric, t_c^b 就是 tic，R_i 是 $R_{b_i}^w$ 的简称，是第 i 个时刻 body(imu) 在世界坐标系下位姿的旋转部分，是代码中的 Ri, 同理，R_j 是 $R_{b_j}^w$，是 Rj; P_i 是 $t_{c_i}^w$，是代码中的 Pi; P_j 是 $t_{c_j}^w$，是 Pj;

$X_l^{c_j}$ 是代码中 pts_camera_j ; $X_l^{b_i}$ 是代码中 pts_imu_i 。$X_l^{b_j}$ 是代码中 pts_imu_j 。

而 $X_l^{c_i}=\tilde{p}_i/\lambda_i$，归一化坐标除以逆深度, $\tilde{p}_i$ 是代码中的 pts_i 。

因为投影方程为

$$Z_c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \tag{7}$$

但这里用的不是像素坐标做差求残差，而是用归一化相机坐标系的前两维做差，相当于上式两边都乘以 K^{-1}，残差为

$$\begin{aligned} r &= f(X_l^{c_j}) - \tilde{p_j} && (8) \\ &= f(\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}) - \tilde{p_j} && (9) \\ &= \begin{pmatrix} X_c/Z_c \\ Y_c/Z_c \end{pmatrix} - \tilde{p_j} && (10) \end{aligned}$$

其中 $\tilde{p_j}$ 为检测出的特征的归一化坐标，即代码中的 pts_j
这也对应到 ProjectionFactor::Evaluate 函数中求取 residuals
下面分别对 (Pi, Qi), (Pj, Qj), (tic, qic)，特征的逆深度求导

$$\frac{\partial r}{\partial \delta} = \frac{\partial r}{\partial X_l^{c_j}} \frac{\partial X_l^{c_j}}{\partial \delta} \tag{11}$$

而

$$\frac{\partial r}{\partial X_l^{c_j}} = \frac{\partial f}{\partial X_l^{c_j}} = \begin{pmatrix} 1/Z_c & 0 & -1/Z_c^2 \\ 0 & 1/Z_c & -1/Z_c^2 \end{pmatrix} \tag{12}$$

对应代码

```
double dep_j = pts_camera_j.z();
residual = (pts_camera_j / dep_j).head<2>() - pts_j.head<2>();
...
reduce << 1. / dep_j, 0, -pts_camera_j(0) / (dep_j * dep_j),
0, 1. / dep_j, -pts_camera_j(1) / (dep_j * dep_j);
```

所以代码中在求 jacobian 时，都会在最后左乘一个 reduce 矩阵。

1.1 $X_l^{c_j}$ 对 (P_i, θ_{Qi}) 求导

$X_l^{c_j}$ 对 P_i 的求导很直接:

$$\frac{\partial X_l^{c_j}}{\partial P_i} = (R_c^b)^{-1}(R_j)^{-1} \tag{13}$$

对应代码 jaco_i. leftCols<3>()= ric.transpose() $*$ Rj.transpose();
旋转一个点对 $\delta\theta$ 的求导，先考虑 $R \approx \hat{R}(I + [\delta\theta]_\times)$

$$\frac{\partial R * p}{\partial \delta\theta} = \frac{\partial \hat{R}(I + [\delta\theta]_\times) * p}{\partial \delta\theta} = \frac{\partial \hat{R}[\delta\theta]_\times * p}{\partial \delta\theta} = \frac{\partial(-\hat{R}[p]_\times * \delta\theta)}{\partial \delta\theta} = -\hat{R}[p]_\times \tag{14}$$

所以 $X_l^{c_j}$ 对 R_i 的扰动求导就是

$$\frac{\partial X_l^{c_j}}{\partial \theta_{Ri}} = (R_c^b)^{-1}(R_j)^{-1} * (-R_i * [X_l^{b_i}]_\times) \tag{15}$$

对应代码

```
jaco_i.rightCols<3>() = ric.transpose() * Rj.transpose() * Ri * -Utility::skewSymmetric(pts_imu_i);"
```

注意代码中为了配合 ceres 的 Evaluate， jacobian_pose_i 是一个 2*7 的矩阵，但是因为是直接对 θ 求导，所以实际上是 2*6，相当于把 LocalParameterization 中 ComputeJacobian 的部分放到 Evaluate 中计算了。

同理，下面几个 jacobian 也是一样的。

1.2 $X_l^{c_j}$ 对 (Pj, θ_{Qj}) 求导

$X_l^{c_j}$ 对 P_j 的求导

$$\frac{\partial X_l^{c_j}}{\partial P_i} = (R_c^b)^{-1}(-(R_j)^{-1}) \tag{16}$$

对应

```
jaco_j.leftCols<3>() = ric.transpose() * -Rj.transpose();
```

$X_l^{c_j}$ 对 θ_{Rj} 求导

$$(R_j)^{-1} \approx [\hat{R}_j(1+[\theta]_\times)]^{-1} = (1-[\theta]_\times)\hat{R}_j \tag{17}$$

所以

$$\begin{aligned}\frac{\partial X_l^{c_j}}{\partial \theta_{Rj}} &= \frac{\partial (R_c^b)^{-1}((1-[\theta_{Rj}]_\times)(R_j)^{-1}*(X_l^w-P_j)-t_c^b)}{\partial \theta_{Rj}} \\ &= \frac{\partial (R_c^b)^{-1}(-[\theta_{Rj}]_\times)(R_j)^{-1}*(X_l^w-P_j)}{\partial \theta_{Rj}} \\ &= \frac{\partial (R_c^b)^{-1}[(R_j)^{-1}*(X_l^w-P_j)]_\times \theta_{Rj}}{\partial \theta_{Rj}} \\ &= (R_c^b)^{-1}[X_l^{b_j}]_\times\end{aligned} \tag{18}$$

对应代码 jaco_j.rightCols<3>()= ric.transpose() $*$ Utility ::skewSymmetric(pts_imu_j); 其中 pts_imu_j 就是 $X_l^{b_j}$

1.3 $X_l^{c_j}$ 对 (t_c^b, θ_{ric}) 求导

$X_l^{c_j}$ 对 t_c^b 求导

$$\begin{aligned}\frac{\partial X_l^{c_j}}{\partial t_c^b} &= \frac{\partial (R_c^b)^{-1}[R_j^{-1}[R_i[R_c^b X_l^{c_i}+t_c^b]+P_i-P_j]-t_c^b]}{\partial t_c^b} \\ &= (R_c^b)^{-1}[R_j^{-1}*R_i - I]\end{aligned} \tag{19}$$

对应代码 jaco_ex.leftCols<3>()= ric.transpose() $*$ (Rj.transpose() $*$ Ri $-$ Eigen ::Matrix3d::Identity());

$X_l^{c_j}$ 对 $\theta_{R_c^b}$ 求导，对 $X_l^{c_j}$ 进行一阶近似有

$$\begin{aligned}X_l^{c_j} &= (R_c^b)^{-1}\left[R_j^{-1}\left[R_i\left[R_c^b X_l^{c_i}+t_c^b\right]+P_i-P_j\right]-t_c^b\right] \\ &\approx (I-[\theta]_\times)(\hat{R}_c^b)^{-1}\left[R_j^{-1}\left[R_i\left[\hat{R}_c^b(I+[\theta]_\times)X_l^{c_i}+t_c^b\right]+P_i-P_j\right]-t_c^b\right] \\ &= (I-[\theta]_\times)\left[R_{tmp}(I+[\theta]_\times)X_l^{c_i}+\underbrace{(\hat{R}_c^b)^{-1}\left[R_j^{-1}(R_i t_c^b+P_i-P_j)-t_c^b\right]}_{A}\right] \\ &= (I-[\theta]_\times)\left[R_{tmp}(I+[\theta]_\times)X_l^{c_i}+A\right] \\ &= C-[\theta]_\times R_{tmp}X_l^{c_i}+R_{tmp}[\theta]_\times X_l^{c_i}-[\theta]_\times A \\ &= C+[R_{tmp}X_l^{c_i}]_\times\theta+R_{tmp}(-[X_l^{c_i}]_\times)\theta+[A]_\times\theta\end{aligned} \tag{20}$$

其中 C 表示跟 θ 无关的部分，那么

$$\frac{\partial X_l^{c_j}}{\partial \theta_{ric}} = [R_{tmp} X_l^{c_i}]_\times + R_{tmp}(-[X_l^{c_i}]_\times) + [A]_\times \tag{21}$$

对应代码

```
Eigen::Matrix3d tmp_r = ric.transpose() * Rj.transpose() * Ri * ric;
jaco_ex.rightCols<3>() = -tmp_r * Utility::skewSymmetric(pts_camera_i) + Utility::skewSymmetric(tmp_r * pts_camera_i) +
Utility::skewSymmetric(ric.transpose() * (Rj.transpose() * (Ri * tic + Pi - Pj) - tic));
```

1.4 $X_l^{c_j}$ 对逆深度 λ 求导

易知

$$\frac{\partial X_l^{c_j}}{\partial \lambda} = (R_c^b)^{-1} R_j^{-1} R_i R_c^b p_i * (-\frac{1}{\lambda^2}) \tag{22}$$

对应代码

```
jacobian_feature = reduce * ric.transpose() * Rj.transpose() * Ri * ric * pts_i * -1.0 / (inv_dep_i * inv_dep_i);
```

2 integration_base.h

根据 [1] 公式 5 有

$$\alpha_{b_{k+1}}^{b_k} = \iint_{t\in[k,k+1]} \gamma_{b_t}^{b_k}(\hat{a}_t - b_{a_t})dt^2 \tag{23}$$

$$\beta_{b_{k+1}}^{b_k} = \int_{t\in[k,k+1]} \gamma_{b_t}^{b_k}(\hat{a}_t - b_{a_t})dt \tag{24}$$

$$\gamma_{b_{k+1}}^{b_k} = \int_{t\in[k,k+1]} \gamma_{b_t}^{b_k} \otimes \begin{bmatrix} 0 \\ 1/2(\hat{\omega}_t - b_{\omega_t}) \end{bmatrix} dt \tag{25}$$

2.1 欧拉积分

采用欧拉积分就是

$$\hat{\alpha}_{i+1}^{b_k} = \hat{\alpha}_i^{b_k} + \hat{\beta}_i^{b_k}\delta t + \frac{1}{2}\hat{\gamma}_i^{b_k}(\hat{a}_i - b_{a_i})\delta t^2 \tag{26}$$

$$\hat{\beta}_{i+1}^{b_k} = \hat{\beta}_i^{b_k} + \hat{\gamma}_i^{b_k}(\hat{a}_i - b_{a_i})\delta t \tag{27}$$

$$\hat{\gamma}_{i+1}^{b_k} = \hat{\gamma}_i^{b_k} \otimes \begin{bmatrix} 1 \\ 1/2(\hat{\omega}_i - b_{\omega_i})\delta t \end{bmatrix} \tag{28}$$

$$b_{a_{k+1}} = b_{a_k} \tag{29}$$

$$b_{\omega_{k+1}} = b_{\omega_k} \tag{30}$$

$$\tag{31}$$

对应代码

```
result_delta_p = delta_p + delta_v * _dt + 0.5 * (delta_q * (_acc_1 - linearized_ba)) * _dt * _dt;
result_delta_v = delta_v + delta_q * (_acc_1 - linearized_ba) * _dt;
Vector3d omg = _gyr_1 - linearized_bg;
omg = omg * _dt / 2;
Quaterniond dR(1, omg(0), omg(1), omg(2));
result_delta_q = (delta_q * dR);
result_linearized_ba = linearized_ba;
result_linearized_bg = linearized_bg;
```

以下推导主要参考 [2] 中 ESKF 的真值 = 名义 + 误差。名义会加上 ˆ 符号。以 β 为例，真值

$$\beta_{i+1} = \beta_i + \gamma_i(a_i - b_{a_i} - n_a)\delta t$$

名义

$$\hat{\beta}_{i+1} = \hat{\beta}_i + \hat{\gamma}_i(a_i - \hat{b}_{a_i})\delta t$$

即

$$\dot{\beta} = \gamma_i(a_i - b_{a_i} - n_a)$$

$$\dot{\hat{\beta}} = \hat{\gamma}_i(a_i - \hat{b}_{a_i})$$

而

$$\gamma_i = \hat{\gamma}_i(I + [\delta\theta]_\times)$$

所以

$$\begin{aligned}
\dot{\delta\beta} &= \dot{\beta} - \dot{\hat{\beta}} = \gamma_i(a_i - b_{a_i} - n_a) - \hat{\gamma}_i(a_i - \hat{b}_{a_i}) \\
&= \hat{\gamma}_i(I + [\delta\theta]_\times)(a_i - b_{a_i} - n_a) - \hat{\gamma}_i(a_i - \hat{b}_{a_i}) \\
&\approx \hat{\gamma}_i[\delta\theta]_\times(a_i - b_{a_i}) - \hat{\gamma}_i(b_{a_i} - \hat{b}_{a_i}) - \hat{\gamma}_i n_a \\
&= -\hat{\gamma}_i[a_i - b_{a_i}]_\times\delta\theta - \hat{\gamma}_i\delta b_{a_i} - \hat{\gamma}_i n_a
\end{aligned} \tag{32}$$

其中约等于是忽略了二阶小项。

同理，根据 [2] ESKF 中的推导，有

$$\dot{\delta\theta}_t = -[\hat{\omega}_t - b_{\omega_t}]_\times\delta\theta_t - \delta b_{\omega_t} - n_\omega$$

真值 $\dot{b}_{a_t} = n_{b_a}$，名义 $\dot{\hat{b}}_{a_t} = 0$，所以 $\dot{\delta b}_{a_t} = n_{b_a}$，同理 $\dot{\delta b}_{\omega_t} = n_{b_\omega}$

而对于 α，如果忽略二阶积分项，即 $\hat{\alpha}^{b_k}_{i+1} = \hat{\alpha}^{b_k}_i + \hat{\beta}^{b_k}_i\delta t$，易知 $\dot{\delta\alpha}_t = \delta\beta_t$

这就是 [1] 中的公式 (9)。

而在代码中 α 是不能忽略二阶积分项的，注意到 α 和 β 结构上很类似，只是多乘以了 $1/2 * dt$ 项，所以易知

$$\dot{\delta\alpha}_t = \delta\beta_t - \frac{1}{2}\hat{\gamma}_t[a_t - b_{a_t}]_\times dt\delta\theta - \frac{1}{2}\hat{\gamma}_t dt\delta b_{a_t} - \frac{1}{2}\hat{\gamma}_t dt n_a$$

整理有表 1和表 2(代码中是加上噪声 n_a, n_ω 而不是减去，所以这里和论文中 U 略有不同)：

Table 1: $\dot{\delta X} = A * \delta X + U * n$ 中的 A

A_{15*15}	0, $\delta\alpha$	3, $\delta\theta$	6, $\delta\beta$	9, δb_a	12, δb_ω
$\dot{\delta\alpha} =$	0	$-\frac{1}{2}R(\delta q)[a_m - b_a]_\times dt$	I_3	$-\frac{1}{2}R(\delta q)dt$	0
$\dot{\delta\theta} =$	0	$-[\omega_m - b_\omega]_\times$	0	0	$-I_3$
$\dot{\delta\beta} =$	0	$-R(\delta q)[a_m - b_a]_\times$	0	$-R(\delta q)$	0
$\dot{\delta b}_a =$	0	0	0	0	0
$\dot{\delta b}_\omega =$	0	0	0	0	0

对应代码 eulerIntegration 函数中

```
MatrixXd A = MatrixXd::Zero(15, 15);
// one step euler 0.5
A.block<3, 3>(0, 3) = 0.5 * (-1 * delta_q.toRotationMatrix()) * R_a_x * _dt;
```

Table 2: $\delta\dot{X} = A * \delta X + U * n$ 中的 U

U_{15*12}	$0,\ n_a$	$3,\ n_w$	$6,\ n_{ba}$	$9,\ n_{b\omega}$
idx=0	$\frac{1}{2}R(\delta q)dt$			
idx=3		I_3		
idx=6	$R(\delta q)$			
idx=9			I_3	
idx=12				I_3

```
    A.block<3, 3>(0, 6) = MatrixXd::Identity(3,3);
    A.block<3, 3>(0, 9) = 0.5 * (-1 * delta_q.
        toRotationMatrix()) * _dt;
    A.block<3, 3>(3, 3) = -R_w_x;
    A.block<3, 3>(3, 12) = -1 * MatrixXd::Identity(3,3);
    A.block<3, 3>(6, 3) = (-1 * delta_q.toRotationMatrix
        ()) * R_a_x;
    A.block<3, 3>(6, 9) = (-1 * delta_q.toRotationMatrix
        ());

    MatrixXd U = MatrixXd::Zero(15,12);
    U.block<3, 3>(0, 0) =  0.5 * delta_q.
        toRotationMatrix() * _dt;
    U.block<3, 3>(3, 3) =  MatrixXd::Identity(3,3);
    U.block<3, 3>(6, 0) =  delta_q.toRotationMatrix();
    U.block<3, 3>(9, 6) = MatrixXd::Identity(3,3);
    U.block<3, 3>(12, 9) = MatrixXd::Identity(3,3);
```

因为有 $\delta\dot{X} = A * \delta X + U * n$，其离散形式为

$$
\begin{aligned}
\delta X_{k+1} &= \delta X_k + (A * \delta X_k + U * n)dt \\
&= (I + A * dt)\delta X_k + U * dt * n \\
&= F * \delta X_k + V
\end{aligned}
\tag{33}
$$

对应代码

```
    F = (MatrixXd::Identity(15,15) + _dt * A);
    V = _dt * U;
    step_jacobian = F;
```

```
    step_V = V;
    jacobian = F * jacobian;
    covariance = F * covariance * F.transpose() + V *
        noise * V.transpose();
```

2.2 中值积分

$$\hat{\alpha}^{b_k}_{i+1} = \hat{\alpha}^{b_k}_{i} + \hat{\beta}^{b_k}_{i}\delta t + \frac{1}{2}\left[\frac{1}{2}\left(\hat{\gamma}^{b_k}_{i}(\hat{a}_i - b_{a_i}) + \hat{\gamma}^{b_k}_{i+1}(\hat{a}_{i+1} - b_{a_{i+1}})\right)\right]\delta t^2 \tag{34}$$

$$\hat{\beta}^{b_k}_{i+1} = \hat{\beta}^{b_k}_{i} + \frac{1}{2}\left(\hat{\gamma}^{b_k}_{i}(\hat{a}_i - b_{a_i}) + \hat{\gamma}^{b_k}_{i+1}(\hat{a}_{i+1} - b_{a_{i+1}})\right)\delta t \tag{35}$$

$$\hat{\gamma}^{b_k}_{i+1} = \hat{\gamma}^{b_k}_{i} \otimes \begin{bmatrix} 1 \\ 1/2\left(\frac{1}{2}\left[(\hat{\omega}_i - b_{\omega_i}) + (\hat{\omega}_{i+1} - b_{\omega_{i+1}})\right]\right)\delta t \end{bmatrix} \tag{36}$$

$$b_{a_{k+1}} = b_{a_k} \tag{37}$$

$$b_{\omega_{k+1}} = b_{\omega_k} \tag{38}$$

$$\tag{39}$$

对应代码

```
    Vector3d un_acc_0 = delta_q * (_acc_0 -
        linearized_ba);
    Vector3d un_gyr = 0.5 * (_gyr_0 + _gyr_1) -
        linearized_bg;
    result_delta_q = delta_q * Quaterniond(1, un_gyr(0)
        * _dt / 2, un_gyr(1) * _dt / 2, un_gyr(2) * _dt /
         2); // icra2015shao 公式 5
    Vector3d un_acc_1 = result_delta_q * (_acc_1 -
        linearized_ba);
    Vector3d un_acc = 0.5 * (un_acc_0 + un_acc_1);
    result_delta_p = delta_p + delta_v * _dt + 0.5 *
        un_acc * _dt * _dt;
    result_delta_v = delta_v + un_acc * _dt;
    result_linearized_ba = linearized_ba;
    result_linearized_bg = linearized_bg;
```

和欧拉积分时的 γ 相比较，容易得到

$$
\begin{aligned}
\dot{\delta\theta}_t &= -[\frac{\hat{\omega}_t + \hat{\omega}_{t+1}}{2} - b_\omega]_\times \delta\theta_t - \delta b_{\omega_t} - \frac{n_{\omega_t} + n_{\omega_{t+1}}}{2} && (40)\\
&= -[\bar{\omega} - b_\omega]_\times \delta\theta_t - \delta b_{\omega_t} - \bar{n}_\omega && (41)
\end{aligned}
$$

其中利用了简写：$\bar{\omega} = \frac{\hat{\omega}_t + \hat{\omega}_{t+1}}{2}$; $\bar{n}_\omega = \frac{n_{\omega_t} + n_{\omega_{t+1}}}{2}$
所以

$$
\begin{aligned}
\delta\theta_{k+1} &= \delta\theta_k + dt * [-[\bar{\omega} - b_\omega]_\times \delta\theta_k - \delta b_{\omega_k} - \bar{n}_\omega]\\
&= (I - [\bar{\omega} - b_\omega]_\times)\delta\theta_k - dt * \delta b_{\omega_k} - dt * \bar{n}_\omega\\
&= (I - [\bar{\omega} - b_\omega]_\times)\delta\theta_k - dt * \delta b_{\omega_k} - dt * 1/2 * n_{\omega_k} - dt * 1/2 * n_{\omega_{k+1}}
\end{aligned} \tag{42}
$$

然后推导 β：
真值

$$
\beta_{i+1} = \beta_i + \frac{1}{2}\left[\gamma_i(a_i - b_{a_i} - n_a) + \gamma_{i+1}(a_{i+1} - b_{a_{i+1}} - n_a)\right]\delta t
$$

名义

$$
\hat{\beta}_{i+1} = \hat{\beta}_i + \frac{1}{2}\left[\hat{\gamma}_i(a_i - \hat{b}_{a_i}) + \hat{\gamma}_{i+1}(a_{i+1} - \hat{b}_{a_{i+1}})\right]\delta t
$$

又

$$
\begin{aligned}
\gamma_k &= \hat{\gamma}_k(1 + \delta\theta_k)\\
\gamma_{k+1} &= \hat{\gamma}_{k+1}(1 + \delta\theta_{k+1})
\end{aligned} \tag{43}
$$

为了简写，用下标 1 来表示 $k+1$，用下标 0 来表示 k
对比欧拉积分时的推导，有

$$
\delta\beta_1 = \delta\beta_0 + \frac{1}{2}dt\left[-\hat{\gamma}_0[a_0 - b_a]_\times\delta\theta_0 - \hat{\gamma}_0\delta b_a - \hat{\gamma}_0 n_{a_0} - \hat{\gamma}_1[a_1 - b_a]_\times\delta\theta_1 - \hat{\gamma}_1\delta b_a - \hat{\gamma}_1 n_{a_0}\right] \tag{44}
$$

把公式 42 代入上式并利用简写 $R_0 = \hat{\gamma}_0$; $R_1 = \hat{\gamma}_1$ 有

$$
\begin{aligned}
\delta\beta_1 = \delta\beta_0 \quad & + \frac{1}{2}dt\left[-R_0[a_0 - b_a]_\times\delta\theta_0 - R_0\delta b_a - R_0 n_{a_0}\right. && (45)\\
& \left. -R_1[a_1 - b_a]_\times\left[(I - [\bar{\omega} - b_\omega]_\times)\delta\theta_0 - dt\delta b_\omega - dt n_\omega\right] - R_1\delta b_a - R_1 n_{a_0}\right] && (46)\\
& && (47)
\end{aligned}
$$

$$
\begin{aligned}
\delta\beta_1 = \delta\beta_0 + \frac{1}{2}dt\,[-R_0[a_0 - b_a]_\times\delta\theta_0 - R_0\delta b_a - R_0 n_{a_0}\\
-R_1[a_1 - b_a]_\times\left[(I - [\bar{\omega} - b_\omega]_\times)\,\delta\theta_0 - dt\delta b_\omega - dtn_\omega\right] - R_1\delta b_a - R_1 n_{a_0}]\\
= \delta\beta_0 + \frac{1}{2}dt\,[-R_0[a_0 - b_a]_\times + R_1[a_1 - b_a]_\times(I - [\bar{\omega} - b_\omega]_\times dt)]\,\delta\theta_0\\
+ (-\frac{1}{2}dt)(R_0 + R_1)\delta b_a + \frac{1}{2}dtR_1[a_1 - b_a]_\times dt\delta b_\omega\\
+ (-\frac{1}{2}dt * R_0)n_{a_0} + (-\frac{1}{2}dt * R_1)n_{a_1} + -\frac{1}{2}dtR_1[a_1 - b_a]_\times * dt * \bar{n}_w
\end{aligned} \tag{48}
$$

同样可以对 $\delta\alpha_{k+1}$ 进行类似式 48 的展开，只是在相关的项上都乘以 $\frac{1}{2}dt$
所以 $\delta X_{k+1} = F * \delta X_k + V * n$
有 (噪声项的符号相反，因为代码中是加上噪声)，简写了 $\tilde{\omega} = \bar{\omega} - b_\omega$

Table 3: $\delta X_{k+1} = F * \delta X_k + V * n$ 中的 F

F_{15*15}	0, $\delta\alpha$	3, $\delta\theta$	6, $\delta\beta$	9, δb_a	12, δb_ω
$\delta\alpha_{k+1} =$	I_3	$-\frac{1}{4}R_0[a_0 - b_a]_\times dt^2 + -\frac{1}{4}R_1[a_1 - b_a]_\times(I - [\tilde{\omega}]_\times dt)dt^2$	$I_3 dt$	$-\frac{1}{4}(R_0 + R_1)dt^2$	$-\frac{1}{4}R_1[a_1 - b_a]_\times dt^2(-dt)$
$\delta\theta_{k+1} =$	0	$I_3 - [\tilde{\omega}]_\times dt$	0	0	$-I_3 dt$
$\delta\beta_{k+1} =$	0	$-\frac{1}{2}R_0[a_0 - b_a]_\times dt + -\frac{1}{2}R_1[a_1 - b_a]_\times(I - [\tilde{\omega}]_\times dt)dt$	I_3	$-\frac{1}{2}(R_0 + R_1)dt$	$-\frac{1}{2}R_1[a_1 - b_a]_\times dt(-dt)$
$\delta b_{a_{k+1}} =$	0	0	0	I_3	0
$\delta b_{\omega_{k+1}} =$	0	0	0	0	I_3

Table 4: $\delta X_{k+1} = F * \delta X_k + V * n$ 中的 V

V_{15*18}	0, n_{a_0}	3, n_{ω_0}	6, n_{a_1}	9, n_{ω_1}	12, n_{b_a}	15, n_{b_ω}
$idx = 0$	$\frac{1}{4}R_0 dt^2$	$-\frac{1}{4}R_1[a_1 - b_a]_\times dt^2 * \frac{1}{2}dt$	$\frac{1}{4}R_1 dt^2$	V_{03}	0	0
$idx = 3$	0	$\frac{1}{2}I_3 dt$	0	$\frac{1}{2}I_3 dt$	0	0
$idx = 6$	$\frac{1}{2}R_0 dt$	$-\frac{1}{2}R_1[a_1 - b_a]_\times dt\frac{1}{2}dt$	$\frac{1}{2}R_1 dt$	V_{63}		
$idx = 9$	0	0	0	0	$I_3 dt$	0
$idx = 12$	0	0	0	0	0	$I_3 dt$

对应代码

```
MatrixXd F = MatrixXd::Zero(15, 15);
F.block<3, 3>(0, 0) = Matrix3d::Identity();
F.block<3, 3>(0, 3) = -0.25 * delta_q.toRotationMatrix() * R_a_0_x * _dt * _dt +
-0.25 * result_delta_q.toRotationMatrix() * R_a_1_x * (Matrix3d::Identity() - R_w_x * _dt) * _dt * _dt;
F.block<3, 3>(0, 6) = MatrixXd::Identity(3,3) * _dt;
F.block<3, 3>(0, 9) = -0.25 * (delta_q.toRotationMatrix() + result_delta_q.toRotationMatrix()) * _dt * _dt;
F.block<3, 3>(0, 12) = -0.25 * result_delta_q.toRotationMatrix() * R_a_1_x * _dt * _dt * -_dt;
F.block<3, 3>(3, 3) = Matrix3d::Identity() - R_w_x * _dt;
F.block<3, 3>(3, 12) = -1.0 * MatrixXd::Identity(3,3) * _dt;
F.block<3, 3>(6, 3) = -0.5 * delta_q.toRotationMatrix() * R_a_0_x * _dt +
-0.5 * result_delta_q.toRotationMatrix() * R_a_1_x * (Matrix3d::Identity() - R_w_x * _dt) * _dt;
F.block<3, 3>(6, 6) = Matrix3d::Identity();
F.block<3, 3>(6, 9) = -0.5 * (delta_q.toRotationMatrix() + result_delta_q.toRotationMatrix()) * _dt;
F.block<3, 3>(6, 12) = -0.5 * result_delta_q.toRotationMatrix() * R_a_1_x * _dt * -_dt;
F.block<3, 3>(9, 9) = Matrix3d::Identity();
F.block<3, 3>(12, 12) = Matrix3d::Identity();

MatrixXd V = MatrixXd::Zero(15,18);
V.block<3, 3>(0, 0) =  0.25 * delta_q.toRotationMatrix() * _dt * _dt;
V.block<3, 3>(0, 3) =  0.25 * -result_delta_q.toRotationMatrix() * R_a_1_x  * _dt * _dt * 0.5 * _dt;
V.block<3, 3>(0, 6) =  0.25 * result_delta_q.
```

```
        toRotationMatrix() * _dt * _dt;
    V.block<3, 3>(0, 9) = V.block<3, 3>(0, 3);
    V.block<3, 3>(3, 3) = 0.5 * MatrixXd::Identity(3,3) * _dt;
    V.block<3, 3>(3, 9) = 0.5 * MatrixXd::Identity(3,3) * _dt;
    V.block<3, 3>(6, 0) = 0.5 * delta_q.toRotationMatrix() * _dt;
    V.block<3, 3>(6, 3) = 0.5 * -result_delta_q.toRotationMatrix() * R_a_1_x * _dt * 0.5 * _dt;
    V.block<3, 3>(6, 6) = 0.5 * result_delta_q.toRotationMatrix() * _dt;
    V.block<3, 3>(6, 9) = V.block<3, 3>(6, 3);
    V.block<3, 3>(9, 12) = MatrixXd::Identity(3,3) * _dt;
    V.block<3, 3>(12, 15) = MatrixXd::Identity(3,3) * _dt;
```

如果采用 RK4 积分是怎么样?

3 imu_factor.h

```
// 残差是15维，7是(Pi, Qi), 9是(Vi, Bai, Bgi), 7是(Pj, Qj), 9是(Vj, Baj, Bgj)
class IMUFactor : public ceres::SizedCostFunction<15, 7, 9, 7, 9>
```

残差计算是 [1] 中公式 13

$$\begin{bmatrix} \alpha^{b_k}_{b_{k+1}} \\ \beta^{b_k}_{b_{k+1}} \\ \gamma^{b_k}_{b_{k+1}} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} q^{b_k}_w (p^w_{b_{k+1}} - p^w_{b_k} + \frac{1}{2} g^w \Delta t_k^2 - v^w_{b_k} \Delta t_k) \\ q^{b_k}_w (v^w_{b_{k+1}} + g^w \Delta t_k - v^w_{b_k}) \\ q^{w\ -1}_{b_k} \otimes q^w_{b_{k+1}} \\ b_{a_{b_{k+1}}} - b_{a_{b_k}} \\ b_{\omega_{b_{k+1}}} - b_{\omega_{b_k}} \end{bmatrix} \tag{49}$$

b_k 和 b_{k+1} 对应到窗口中的连续两帧，这两帧之间有多个 imu 的数据，通过预积分可以算出预积分项，并不参与到优化的迭代过程中。

其中式 49 的右边是窗口中视觉帧中的数据计算出来的，也是要优化的参数块，对应到代码 integration_base.h 的 evaluate 中(这个函数被IMUFactor的Evaluate所调用)：

```
residuals.block<3, 1>(O_P, 0) = Qi.inverse() * (0.5 * G * sum_dt * sum_dt + Pj - Pi - Vi * sum_dt) - corrected_delta_p;
residuals.block<3, 1>(O_R, 0) = 2 * (corrected_delta_q.inverse() * (Qi.inverse() * Qj)).vec();
residuals.block<3, 1>(O_V, 0) = Qi.inverse() * (G * sum_dt + Vj - Vi) - corrected_delta_v;
residuals.block<3, 1>(O_BA, 0) = Baj - Bai;
residuals.block<3, 1>(O_BG, 0) = Bgj - Bgi;
```

而式 49 的左边是预积分项，由于在预积分过程中假设了加速度和角速度的 bias 保持不变，所以最后求预积分时要把这部分加回来，即：

$$
\begin{aligned}
\alpha^{b_k}_{b_{k+1}} &= \hat{\alpha}^{b_k}_{b_{k+1}} + J^{\alpha}_{b_a}\delta b_{a_k} + J^{\alpha}_{b_\omega}\delta b_{\omega_k} \\
\beta^{b_k}_{b_{k+1}} &= \hat{\beta}^{b_k}_{b_{k+1}} + J^{\beta}_{b_a}\delta b_{a_k} + J^{\beta}_{b_\omega}\delta b_{\omega_k} \\
\gamma^{b_k}_{b_{k+1}} &= \hat{\gamma}^{b_k}_{b_{k+1}} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}J^{\gamma}_{b_\omega}\delta b_{\omega_k} \end{bmatrix}
\end{aligned}
\tag{50}
$$

其中 $\hat{\alpha}^{b_k}_{b_{k+1}}, \hat{\beta}^{b_k}_{b_{k+1}}, \hat{\gamma}^{b_k}_{b_{k+1}}$ 是不参与迭代优化（也就是只算一次）的预积分项，对应代码中的 IntegrationBase 中的 delta_p, delta_v, delta_q，而 $\alpha^{b_k}_{b_{k+1}}, \beta^{b_k}_{b_{k+1}}, \gamma^{b_k}_{b_{k+1}}$ 则对应 corrected_delta_p, corrected_delta_v, corrected_delta_q 。

所以 IntegrationBase 中的 jacobian 通过 $J_{t+\delta t} = (I + F_t\delta t)J_t, t \in [k, k+1]$ 不断计算，最后也只是为了用到式 50 中出现对 b_a, b_ω 求导的部分。

简写式 49，残差的公式为

$$
\begin{aligned}
\Delta p &= R_i^{-1}(p_j - p_i + \frac{1}{2}g\Delta t^2 - v_i\Delta t) - (\hat{\alpha}^i_j + J^{\alpha}_{b_a}\delta b_{a_i} + J^{\alpha}_{b_\omega}\delta b_{\omega_i}) \\
\Delta\theta &= 2\left[\left(\hat{\gamma}^i_j \otimes \begin{bmatrix} 1 \\ \frac{1}{2}J^{\gamma}_{b_\omega}\delta b_{\omega_i} \end{bmatrix}\right)^{-1}(R_i^{-1}R_j)\right]_{vec} \\
\Delta v &= R_i^{-1}(v_j + g\Delta t - v_i) - (\hat{\beta}^i_j + J^{\beta}_{b_a}\delta b_{a_i} + J^{\beta}_{b_\omega}\delta b_{\omega_i}) \\
\Delta b_a &= b_{a_j} - b_{a_i} \\
\Delta b_\omega &= b_{\omega_j} - b_{\omega_i}
\end{aligned}
\tag{51}
$$

其中需要代入 $\delta b_{a_i}=b_{a_i}-\hat{b}_{a_i}$ 和 $\delta b_{\omega_i}=b_{\omega_i}-\hat{b}_{\omega_i}$
对应代码

```
Eigen::Matrix3d dp_dba = jacobian.block<3, 3>(O_P, O_BA);
Eigen::Matrix3d dp_dbg = jacobian.block<3, 3>(O_P, O_BG);

Eigen::Matrix3d dq_dbg = jacobian.block<3, 3>(O_R, O_BG);

Eigen::Matrix3d dv_dba = jacobian.block<3, 3>(O_V, O_BA);
Eigen::Matrix3d dv_dbg = jacobian.block<3, 3>(O_V, O_BG);

Eigen::Vector3d dba = Bai - linearized_ba;
Eigen::Vector3d dbg = Bgi - linearized_bg;

Eigen::Quaterniond corrected_delta_q = delta_q * Utility::deltaQ(dq_dbg * dbg);
Eigen::Vector3d corrected_delta_v = delta_v + dv_dba * dba + dv_dbg * dbg;
Eigen::Vector3d corrected_delta_p = delta_p + dp_dba * dba + dp_dbg * dbg;
```

注意其中 Eigen::Vector3d dba = Bai − linearized_ba; //求取delta ba 是用一开始的Bai − linearized_bg, linearized_bg在ceres迭代优化过程中不变，而Bai是优化参数，是会变的。线性化的假设 bias 不变,即代码 IntegrationBase的midPointIntegration 中 result_linearized_ba = linearized_ba;,所以 IntegrationBase 的 成员变量linearized_ba ,linearized_bw只在构造函数或者调用repropagate 时传入参数而设置。

那么这个残差，式 49 的右边减去左边，即 $(\Delta p,\Delta\theta,\Delta v,\Delta b_a,\Delta b_\omega)^T$ 分别对 4 个参数块 $(p_i,q_i),(p_j,q_j),(v_i,b_{a_i},b_{\omega_i}),(v_j,b_{a_j},b_{\omega_j})$ 求导 (同上面一样，对四元数 q 实际上是对 θ 求导)。

注意其中 $\hat{\alpha}^i_j,\hat{\beta}^i_j,\hat{\gamma}^i_j$ 是在迭代中都是常数项。

3.1 Δ 对 (p_i,θ_i) 求导

求 $\frac{\partial\Delta\theta}{\partial\theta_i}$, 简写

$$q_c = \gamma^{b_k}_{b_{k+1}} = \hat{\gamma}^{b_k}_{b_{k+1}} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} J^{\gamma}_{b_\omega} \delta b_{\omega_k} \end{bmatrix}$$

$$\begin{aligned}
\Delta\theta &= 2\left[q_c^{-1} \otimes q_i^{-1} \otimes q_j\right]_{vec} \\
&= 2\left[q_j^{-1} \otimes q_i \otimes q_c\right]^{-1}_{vec} \\
&= 2\left[q_j^{-1} \otimes (q_i \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\delta\theta \end{bmatrix}) \otimes q_c\right]^{-1}_{vec} \\
&= 2\left[\left[q_j^{-1} \otimes q_i\right]_L [q_c]_R \begin{bmatrix} 1 \\ \frac{1}{2}\delta\theta \end{bmatrix}\right]^{-1}_{vec} \\
&= [q_\theta]^{-1}_{vec}
\end{aligned} \tag{52}$$

所以

$$\frac{\partial\Delta\theta}{\partial\theta_i} = \frac{\partial\Delta\theta}{\partial q_\theta}\frac{\partial q_\theta}{\partial\theta} = -\left[q_j^{-1} \otimes q_i\right]_L [q_c]_R \tag{53}$$

上面用了四元数的扰动，而在求 $\frac{\partial\Delta p}{\partial\theta_i}$ 和 $\frac{\partial\Delta v}{\partial\theta_i}$ 时，则要用对旋转矩阵的扰动，即 $R = \hat{R}(I + [\theta]_\times)$

这个很好求，比如

$$\begin{aligned}
\Delta v &= R_i^{-1}(v_j + g\Delta t - v_i) - Const \\
&\approx (\hat{R}(I + [\theta]_\times))^{-1}(v_j + g\Delta t - v_i) - Const \\
&\approx (I - [\theta]_\times)\hat{R}^{-1}(v_j + g\Delta t - v_i) - Const \\
&= [\hat{R}^{-1}(v_j + g\Delta t - v_i)]_\times * \theta + Const
\end{aligned} \tag{54}$$

所以

$$\frac{\partial\Delta v}{\partial\theta_i} = [\hat{R}^{-1}(v_j + g\Delta t - v_i)]_\times$$

总结即

$$\begin{pmatrix} \frac{\partial\Delta p}{\partial p_i} & \frac{\partial\Delta p}{\partial\theta_i} \\ \frac{\partial\Delta\theta}{\partial p_i} & \frac{\partial\Delta\theta}{\partial\theta_i} \\ \frac{\partial\Delta v}{\partial p_i} & \frac{\partial\Delta v}{\partial\theta_i} \\ \frac{\partial\Delta b_a}{\partial p_i} & \frac{\partial\Delta b_a}{\partial\theta_i} \\ \frac{\partial\Delta b_w}{\partial p_i} & \frac{\partial\Delta b_w}{\partial\theta_i} \end{pmatrix} = \begin{bmatrix} -R_i^{-1} & [R_i^{-1}(p_j - p_i + \frac{1}{2}g\Delta t^2 - v_i\Delta t)]_\times \\ 0 & -\left[q_j^{-1} \otimes q_i\right]_L [q_c]_R \\ 0 & [R_i^{-1}(v_j + g\Delta t - v_i)]_\times \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{55}$$

对应代码

```
jacobian_pose_i.block<3, 3>(O_P, O_P) = -Qi.inverse
    ().toRotationMatrix();
jacobian_pose_i.block<3, 3>(O_P, O_R) = Utility::
    skewSymmetric(Qi.inverse() * (0.5 * G * sum_dt *
    sum_dt + Pj - Pi - Vi * sum_dt)); // 注意 Qi.
    inverse()是在skewSymmetric 里面

Eigen::Quaterniond corrected_delta_q =
    pre_integration->delta_q * Utility::deltaQ(dq_dbg
     * (Bgi - pre_integration->linearized_bg));
jacobian_pose_i.block<3, 3>(O_R, O_R) = -(Utility::
    Qleft(Qj.inverse() * Qi) * Utility::Qright(
    corrected_delta_q)).bottomRightCorner<3, 3>();

jacobian_pose_i.block<3, 3>(O_V, O_R) = Utility::
    skewSymmetric(Qi.inverse() * (G * sum_dt + Vj -
    Vi));

jacobian_pose_i = sqrt_info * jacobian_pose_i;
```

注意 jacobian_pose_i 是 15*7 的矩阵 Eigen::Map<Eigen::Matrix<double, 15, 7, Eigen::RowMajor>> jacobian_pose_i(jacobians[0]);。

3.2 Δ 对 $(v_i, b_{a_i}, b_{\omega_i})$ 求导

同上，易知

$$\begin{pmatrix} \frac{\partial\Delta p}{\partial v_i} & \frac{\partial\Delta p}{\partial b_{a_i}} & \frac{\partial\Delta p}{\partial b_{\omega_i}} \\ \frac{\partial\Delta \theta}{\partial v_i} & \frac{\partial\Delta \theta}{\partial b_{a_i}} & \frac{\partial\Delta \theta}{\partial b_{\omega_i}} \\ \frac{\partial\Delta v}{\partial v_i} & \frac{\partial\Delta v}{\partial b_{a_i}} & \frac{\partial\Delta v}{\partial b_{\omega_i}} \\ \frac{\partial\Delta b_a}{\partial v_i} & \frac{\partial\Delta b_a}{\partial b_{a_i}} & \frac{\partial\Delta b_a}{\partial b_{\omega_i}} \\ \frac{\partial\Delta b_w}{\partial v_i} & \frac{\partial\Delta b_w}{\partial b_{a_i}} & \frac{\partial\Delta b_w}{\partial b_{\omega_i}} \end{pmatrix} = \begin{bmatrix} -R_i^{-1}\Delta t & -J^{\alpha}_{b_a} & -J^{\alpha}_{b_\omega} \\ 0 & 0 & -\left[q_j^{-1}\otimes q_i \otimes q_c\right]_L J^{\gamma}_{b_\omega} \\ -R_i^{-1} & -J^{\beta}_{b_a} & -J^{\beta}_{b_\omega} \\ 0 & -I_3 & 0 \\ 0 & 0 & -I_3 \end{bmatrix} \tag{56}$$

对应代码

```
jacobian_speedbias_i.block<3, 3>(O_P, O_V - O_V) = -
    Qi.inverse().toRotationMatrix() * sum_dt;
```

```
    jacobian_speedbias_i.block<3, 3>(O_P, O_BA - O_V) = -dp_dba;
    jacobian_speedbias_i.block<3, 3>(O_P, O_BG - O_V) = -dp_dbg;

    Eigen::Quaterniond corrected_delta_q = pre_integration->delta_q * Utility::deltaQ(dq_dbg * (Bgi - pre_integration->linearized_bg));
    jacobian_speedbias_i.block<3, 3>(O_R, O_BG - O_V) = -Utility::Qleft(Qj.inverse() * Qi * corrected_delta_q).bottomRightCorner<3, 3>() * dq_dbg;

    jacobian_speedbias_i.block<3, 3>(O_V, O_V - O_V) = -Qi.inverse().toRotationMatrix();
    jacobian_speedbias_i.block<3, 3>(O_V, O_BA - O_V) = -dv_dba;
    jacobian_speedbias_i.block<3, 3>(O_V, O_BG - O_V) = -dv_dbg;

    jacobian_speedbias_i.block<3, 3>(O_BA, O_BA - O_V) = -Eigen::Matrix3d::Identity();
    jacobian_speedbias_i.block<3, 3>(O_BG, O_BG - O_V) = -Eigen::Matrix3d::Identity();

    jacobian_speedbias_i = sqrt_info * jacobian_speedbias_i;
```

3.3 Δ 对 (p_j, θ_j) 求导

同上，易知

$$\begin{pmatrix} \frac{\partial \Delta p}{\partial p_j} & \frac{\partial \Delta p}{\partial \theta_j} \\ \frac{\partial \Delta \theta}{\partial p_j} & \frac{\partial \Delta \theta}{\partial \theta_j} \\ \frac{\partial \Delta v}{\partial p_j} & \frac{\partial \Delta v}{\partial \theta_j} \\ \frac{\partial \Delta b_a}{\partial p_j} & \frac{\partial \Delta b_a}{\partial \theta_j} \\ \frac{\partial \Delta b_w}{\partial p_j} & \frac{\partial \Delta b_w}{\partial \theta_j} \end{pmatrix} = \begin{bmatrix} R_i^{-1} & 0 \\ 0 & \left[q_c^{-1} \otimes q_i^{-1} q_j\right]_L \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{57}$$

对应代码

```
jacobian_pose_j.block<3, 3>(O_P, O_P) = Qi.inverse()
    .toRotationMatrix();
Eigen::Quaterniond corrected_delta_q =
    pre_integration->delta_q * Utility::deltaQ(dq_dbg
     * (Bgi - pre_integration->linearized_bg));
jacobian_pose_j.block<3, 3>(O_R, O_R) = Utility::
    Qleft(corrected_delta_q.inverse() * Qi.inverse()
    * Qj).bottomRightCorner<3, 3>();
```

3.4 Δ 对 $(v_j, b_{a_j}, b_{\omega_j})$ 求导

同上，易知

$$\begin{pmatrix} \frac{\partial\Delta p}{\partial v_j} & \frac{\partial\Delta p}{\partial b_{a_j}} & \frac{\partial\Delta p}{\partial b_{\omega_j}} \\ \frac{\partial\Delta\theta}{\partial v_j} & \frac{\partial\Delta\theta}{\partial b_{a_j}} & \frac{\partial\Delta\theta}{\partial b_{\omega_j}} \\ \frac{\partial\Delta v}{\partial v_j} & \frac{\partial\Delta v}{\partial b_{a_j}} & \frac{\partial\Delta v}{\partial b_{\omega_j}} \\ \frac{\partial\Delta b_a}{\partial v_j} & \frac{\partial\Delta b_a}{\partial b_{a_j}} & \frac{\partial\Delta b_a}{\partial b_{\omega_j}} \\ \frac{\partial\Delta b_w}{\partial v_j} & \frac{\partial\Delta b_w}{\partial b_{a_j}} & \frac{\partial\Delta b_w}{\partial b_{\omega_j}} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ R_i^{-1} & 0 & 0 \\ 0 & I_3 & 0 \\ 0 & 0 & I_3 \end{bmatrix} \tag{58}$$

对应代码

```
jacobian_speedbias_j.block<3, 3>(O_V, O_V - O_V) =
    Qi.inverse().toRotationMatrix();
jacobian_speedbias_j.block<3, 3>(O_BA, O_BA - O_V) =
     Eigen::Matrix3d::Identity();
jacobian_speedbias_j.block<3, 3>(O_BG, O_BG - O_V) =
     Eigen::Matrix3d::Identity();
jacobian_speedbias_j = sqrt_info *
    jacobian_speedbias_j;
```

4 initial_aligment.cpp

4.1 求陀螺仪的零偏，solveGyroscopeBias

根据 [1] 公式 15，有

$$\gamma_j^i = q_i^{-1} \otimes q_j \approx= \hat{\gamma}_j^i \otimes \begin{bmatrix} 1 \\ \frac{1}{2} J_{b_\omega}^{\gamma} \delta b_{\omega_k} \end{bmatrix} \tag{59}$$

其中 q_i, q_j 为视觉 sfm 获取的 pose，而 $\hat{\gamma}_j^i, J_{b_\omega}^{\gamma}$ 为 i 和 j 帧之间的 imu 预积分所得。

所以

$$J_{b_\omega}^{\gamma} \delta b_{\omega_k} = \left[(\hat{\gamma}_j^i)^{-1} \otimes q_i^{-1} \otimes q_j \right]_{vec} \tag{60}$$

对应代码

```
for (frame_i = all_image_frame.begin(); next(frame_i) != all_image_frame.end(); frame_i++)
{
    frame_j = next(frame_i);
    ...
    Eigen::Quaterniond q_ij(frame_i->second.R.transpose() * frame_j->second.R); // i_R_j
    tmp_A = frame_j->second.pre_integration->jacobian.template block<3, 3>(O_R, O_BG); // O_R = 3; O_BG=12;
    tmp_b = 2 * (frame_j->second.pre_integration->delta_q.inverse() * q_ij).vec();
    A += tmp_A.transpose() * tmp_A;
    b += tmp_A.transpose() * tmp_b;
}
delta_bg = A.ldlt().solve(b);
```

4.2 LinearAlignment

根据 [1] 公式 17 有

$$\begin{aligned} \hat{\alpha}_{b_{k+1}}^{b_k} &= q_{c_0}^{b_k} (s(\bar{p}_{b_{k+1}}^{c_0} - \bar{p}_{b_k}^{c_0}) + \frac{1}{2} g^{c_0} \Delta t_k^2 - v_{b_k}^{c_0} \Delta t_k) \\ \hat{\beta}_{b_{k+1}}^{b_k} &= q_{c_0}^{b_k} (v_{b_{k+1}}^{c_0} + g^{c_0} \Delta t_k - v_{b_k}^{c_0}) \end{aligned} \tag{61}$$

代入

$$\begin{aligned} s\bar{p}_{b_k}^{c_0} &= s\bar{p}_{c_k}^{c_0} - R_{b_k}^{c_0} p_c^b \\ s\bar{p}_{b_{k+1}}^{c_0} &= s\bar{p}_{c_{k+1}}^{c_0} - R_{b_{k+1}}^{c_0} p_c^b \end{aligned} \tag{62}$$

$$
\begin{aligned}
q_{c_0}^{b_k} s(\bar{p}_{b_{k+1}}^{c_0} - \bar{p}_{b_k}^{c_0}) &= q_{c_0}^{b_k}\left[s(\bar{p}_{c_{k+1}}^{c_0} - \bar{p}_{c_k}^{c_0}) - (R_{b_{k+1}}^{c_0} p_c^b - R_{b_k}^{c_0} p_c^b) \right] \\
&= q_{c_0}^{b_k}(\bar{p}_{c_{k+1}}^{c_0} - \bar{p}_{c_k}^{c_0}) s - (R_{b_{k+1}}^{b_k} p_c^b - p_c^b)
\end{aligned} \tag{63}
$$

再整理有

$$
\begin{bmatrix} -\Delta t & 0 & \frac{1}{2} q_{c_0}^{b_k} \Delta t_k^2 & q_{c_0}^{b_k}(\bar{p}_{c_{k+1}}^{c_0} - \bar{p}_{c_k}^{c_0}) \\ -I_3 & q_{c_0}^{b_k} q_{b_{k+1}}^{c_0} & q_{c_0}^{b_k} \Delta t_k & 0 \end{bmatrix} \begin{bmatrix} q_{c_0}^{b_k} v_{b_k}^{c_0} \\ q_{c_0}^{b_{k+1}} v_{b_{k+1}}^{c_0} \\ g^{c_0} \\ s \end{bmatrix} = \begin{bmatrix} \hat{\alpha}_{b_{k+1}}^{b_k} + R_{b_{k+1}}^{b_k} p_c^b - p_c^b \\ \hat{\beta}_{b_{k+1}}^{b_k} \end{bmatrix} \tag{64}
$$

其中 $q_{b_k}^{c_0}$ 对应代码 frame_i−>second.R，$q_{b_{k+1}}^{c_0}$ 对应代码 frame_j−>second.R，而且 $R_{b_{k+1}}^{b_k} = (R_{b_k}^{c_0})^{-1} R_{b_{k+1}}^{c_0}$，$\bar{p}_{c_{k+1}}^{c_0}$ 对应 frame_j−>second.T，$\bar{p}_{c_k}^{c_0}$ 对应 frame_i−>second.T (R 是相机在 c_0 坐标系下，T 是 body 也就是 imu 在 c_0 坐标系下)。

对应代码

```
double dt = frame_j->second.pre_integration->sum_dt;

tmp_A.block<3, 3>(0, 0) = -dt * Matrix3d::Identity();
tmp_A.block<3, 3>(0, 6) = frame_i->second.R.transpose() * dt * dt / 2 * Matrix3d::Identity();
tmp_A.block<3, 1>(0, 9) = frame_i->second.R.transpose() * (frame_j->second.T - frame_i->second.T) / 100.0; // 这里除了100，所以下面求s时也要除以100
tmp_b.block<3, 1>(0, 0) = frame_j->second.pre_integration->delta_p + frame_i->second.R.transpose() * frame_j->second.R * TIC[0] - TIC[0];
tmp_A.block<3, 3>(3, 0) = -Matrix3d::Identity();
tmp_A.block<3, 3>(3, 3) = frame_i->second.R.transpose() * frame_j->second.R;
tmp_A.block<3, 3>(3, 6) = frame_i->second.R.transpose() * dt * Matrix3d::Identity();
tmp_b.block<3, 1>(3, 0) = frame_j->second.pre_integration->delta_v;
```

5 Marginalization

marg 就是把过去的信息转换为下一次迭代优化过程中的先验，上一次的迭代过程中已经有了 $H_0\delta x_0 = J_0^T J_0 \delta x_0 = J_0^T b_0$，如果 sliding window 的方法不使用之前的信息，那么就是残差计算中不包括上一次迭代计算完后的残差，也就是这一次的先验，那么 J_0, b_0 不需要保留，因为残差的计算中只有 imu 和 visual 的残差。如果要保留之前的残差，由于新的变量 x_1 和 x_0 已经在维数上有不同 (加了一些新变量，滑动窗口去掉了一些变量)，因此也就用到 marg。即去掉 H_0, b_0 中被 marg 的部分，只保留剩下的部分 (记为 J_{r0}, b_{r0})，使用 schur 补可以做到这一点。那么在这次迭代过程中，这部分的残差为 $b_1 = b_{r0} + J_{r0}(x_1 - x_{r0})$，即 marginalization_factor .cpp 中的 Eigen::Map<Eigen::VectorXd>(residuals, n)= marginalization_info−> linearized_residuals + marginalization_info−>linearized_jacobians ∗ dx; 残差的 jacobian 也就是上一次 marg 后的 J_r0 部分

References

[1] Technical Report VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator

[2] [kinematics] Quaternion kinematics for the error-state KF.pdf