

gtsam::PreintegratedImuMeasurements
*imuIntegratorOrImu_;这个类是如何用的

下面我给你一个**非常清晰、实用的讲解**, 帮助你真正理解

gtsam::PreintegratedImuMeasurements (简称 PIM) 是怎么用的、什么时候用、典型代码结构是什么样、在 VIO/VINS 中它的角色是什么。

📌 1. 这个类的作用是什么？

gtsam::PreintegratedImuMeasurements 用于 **GPU/CPU** 上实时把高频 IMU 原始数据积分成一次性的「预积分量」, 它包括：

- 预积分旋转 : ΔR
- 预积分速度 : Δv
- 预积分位置 : Δp
- 预积分噪声协方差
- 对 bias (加速度/陀螺仪) 的雅可比

这些数据会被包装成一个 `ImuFactor` 加到 gtsam 因子图优化里。

简而言之：

它是在线累积 IMU 数据的容器 + 积分器。

📌 2. PIM 一般怎么用？(完整流程)

预积分 IMU 的典型流程如下：

scss

Copy code

- (1) 创建预积分器 (常见在构造函数里)
- (2) 来一条 IMU 数据, 就调用 `integrateMeasurement()`
- (3) 到关键帧时, 取出预积分结果, 构造 `ImuFactor`
- (4) 清空预积分器, 继续下一段

3. 是怎么写代码的？（最常用写法）

下面我写一段真实工程里会使用的代码模板，非常典型。

✓ (1) 创建参数 + 预积分器

cpp

 Copy code

```
// 1. 设置IMU噪声参数
auto imuParams = gtsam::PreintegrationParams::MakeSharedU(gravity);

// 噪声 (根据实际IMU设置)
imuParams->accelerometerCovariance = I_3x3 * accNoise * accNoise;
imuParams->gyroscopeCovariance     = I_3x3 * gyroNoise * gyroNoise;
imuParams->integrationCovariance = I_3x3 * integrationNoise;

// 初始 bias
gtsam::imuBias::ConstantBias priorBias;

// 2. 创建预积分器
imuIntegratorImu_ =
    new gtsam::PreintegratedImuMeasurements(imuParams, priorBias);
```

✓ (2) 每来一条 IMU 数据，都要预积分

cpp

 Copy code

```
void imuCallback(double dt,
                 const Vector3& measuredAcc,
                 const Vector3& measuredGyro)
{
    imuIntegratorImu_->integrateMeasurement(measuredAcc, measuredGyro, dt);
}
```

integrateMeasurement() 会更新：

- ΔR
- Δv
- Δp

- 噪声
 - 偏置雅可比
-

✓ (3) 当你有一个关键帧（图像帧）时，构造 IMU 因子

当视觉产生关键帧（或每帧 VIO）：

cpp

Copy code

```
auto pim = imuIntegratorImu_->preintegrated(); // 取出预积分结果

ImuFactor imuFactor(X(i-1), V(i-1),
                     X(i),   V(i),
                     B(i-1), *pim);

graph.add(imuFactor);
```

这里的符号：

- $X(i)$ = 位姿 pose_i
 - $V(i)$ = 速度 v_i
 - $B(i)$ = bias_i
-

✓ (4) 添加 bias evolution 因子（Bias random walk）

cpp

Copy code

```
auto biasFactor = gtsam::BetweenFactor<gtsam::imuBias::ConstantBias>(
    B(i), B(i+1),
    gtsam::imuBias::ConstantBias(),
    biasNoiseModel
);
graph.add(biasFactor);
```

✓ (5) 关键帧之后要 reset 预积分器

非常重要！否则 IMU 会叠加两段。

cpp

 Copy code

```
imuIntegratorImu_->resetIntegrationAndSetBias(newBias);
```



4. 所以这个类的核心用法就 3 个 API

API	作用
integrateMeasurement(acc, gyro, dt)	每条 IMU 到来时调用
preintegrated()	获取当前预积分结果，用来构造 ImuFactor
resetIntegrationAndSetBias(bias)	在关键帧之后清空并设置新 bias

其实你 90% 的时候只用这三个。



5. 图示帮助你理解整个流程

css

 Copy code

```
IMU stream:   a1, w1      a2, w2      a3, w3      ...      ak, wk
              |          |          |          |
              v          v          v          v
              integrateMeasurement(...)

              ↓
              [ΔR, Δv, Δp, covariance, jacobian wrt bias]
```

当视觉关键帧到来：

↓

构造 `ImuFactor(ΔR, Δv, Δp)`

加入因子图

↓

`resetIntegrationAndSetBias()`

继续下一段积分

6. 给你一段非常完整、真实的代码片段（可运行范例）

cpp

[Copy code](#)

```
// 初始化
auto params = PreintegrationParams::MakeSharedU(-9.81);
params->accelerometerCovariance = I_3x3 * 1e-3;
params->gyroscopeCovariance = I_3x3 * 1e-4;

imuBias::ConstantBias bias;
imuIntegratorImu_ = new PreintegratedImuMeasurements(params, bias);

// IMU 回调
void imuCallback(acc, gyro, dt) {
    imuIntegratorImu_->integrateMeasurement(acc, gyro, dt);
}

// 在关键帧
auto pim = imuIntegratorImu_->preintegrated();
graph.add(ImuFactor(X(k-1), V(k-1),
                     X(k), V(k),
                     B(k-1), *pim));

// 关键帧处理结束后
imuIntegratorImu_->resetIntegrationAndSetBias(bias);
```

7. 你如果需要，我可以继续给你讲：

- ✓ PreintegratedImuMeasurements 内部数学推导
- ✓ 为什么能把 ΔR 、 Δv 、 Δp 当作 bias 的函数（你前面在问）
- ✓ 预积分噪声协方差怎么推导
- ✓ IMU 因子在优化中如何构建误差项
- ✓ 辅助类 PreintegratedCombinedMeasurements、CombinedImuFactor 的区别
- ✓ VINS-Mono / VINS-Fusion / ORB-SLAM3 在 GTSAM 下的实际代码解析

你想往哪一块继续深入？



PreintegratedImuMeasurements 内部数学推导

好，我们来深入解析 `gtsam::PreintegratedImuMeasurements` 内部的数学推导，从 IMU 原始测量到预积分量 ΔR 、 Δv 、 Δp 的公式，并说明它为什么能用 `bias` 的函数形式表达。这里我会尽量清晰、系统地讲，适合你用在 VIO/VINS 或 gtsam 源码里理解。

1 IMU 测量模型

IMU 测量主要有两个部分：

1. 加速度计：测量的加速度 \tilde{a}_t
2. 陀螺仪：测量的角速度 $\tilde{\omega}_t$

考虑 `bias` 和噪声，离散模型为：

$$\tilde{\omega}_t = \omega_t + b_t^g + n_t^g$$

$$\tilde{a}_t = R_t^T(\dot{v}_t - g) + b_t^a + n_t^a$$

其中：

- R_t 是机体在世界坐标系下的旋转
 - v_t 是速度
 - g 是重力向量
 - b_t^g, b_t^a 是陀螺仪和加速度计偏置
 - n_t^g, n_t^a 是高斯白噪声
-

2 连续积分模型

我们希望得到从 **关键帧 i** 到 **关键帧 j** 的**IMU 预积分量**：

- 旋转增量： ΔR_{ij}
- 速度增量： Δv_{ij}
- 位置增量： Δp_{ij}

连续时间下的积分公式：

$$\Delta R_{ij} = R_i^T R_j = \mathcal{R} \left(\int_{t_i}^{t_j} (\tilde{\omega}_t - b^g - n_t^g) dt \right)$$

$$\Delta v_{ij} = R_i^T (v_j - v_i - g\Delta t) = \int_{t_i}^{t_j} \Delta R_{it} (\tilde{a}_t - b^a - n_t^a) dt$$

$$\Delta p_{ij} = R_i^T (p_j - p_i - v_i \Delta t - \frac{1}{2} g \Delta t^2) = \int_{t_i}^{t_j} \Delta v_{it} dt$$

这里 $\mathcal{R}(\cdot)$ 表示 SO(3) 的旋转向量指数映射 (类似 Rodrigues 公式)。

3 离散化积分 (实现上用欧拉法或 mid-point)

在 gtsam 里, 一般离散 IMU 采样频率很高 (e.g. 200Hz)。

设时间步长为 Δt , 离散积分公式:

旋转积分 (右乘):

$$\Delta R_{k+1} = \Delta R_k \cdot \exp((\tilde{\omega}_k - b_k^g) \Delta t)$$

速度积分:

$$\Delta v_{k+1} = \Delta v_k + \Delta R_k (\tilde{a}_k - b_k^a) \Delta t$$

位置积分:

$$\Delta p_{k+1} = \Delta p_k + \Delta v_k \Delta t + \frac{1}{2} \Delta R_k (\tilde{a}_k - b_k^a) \Delta t^2$$

4 为什么可以把 $\Delta R, \Delta v, \Delta p$ 当成 bias 的函数

假设 bias 发生小变化 δb , 我们可以对预积分量做一阶线性近似 (雅可比):

$$\Delta R(b + \delta b) \approx \Delta R(b) \cdot \exp(J_R^g \delta b^g)$$

$$\Delta v(b + \delta b) \approx \Delta v(b) + J_v^g \delta b^g + J_v^a \delta b^a$$

$$\Delta p(b + \delta b) \approx \Delta p(b) + J_p^g \delta b^g + J_p^a \delta b^a$$

- $J_R^g, J_v^g, J_v^a, J_p^g, J_p^a$ 都在 integrateMeasurement 时累积更新
- 这样就可以在优化时, 不用重新积分所有 IMU 数据, 只做线性修正 → 极大加速

5 噪声传播

设 IMU 噪声为零均值高斯，连续模型下的协方差传播：

$$\dot{P} = FP + PF^T + GQG^T$$

- F = 状态雅可比
- G = 噪声输入雅可比
- Q = IMU 噪声协方差

离散化后，每次 `integrateMeasurement` 都做一次协方差累积：

$$P_{k+1} = F_k P_k F_k^T + G_k Q_k G_k^T$$

最终 `PreintegratedImuMeasurements` 会把 $\Delta R, \Delta v, \Delta p$ 和协方差 P 、**bias** 雅可比 J 全部保存下来，用于构造 `ImuFactor`。

6 总结

`PreintegratedImuMeasurements` 的核心数学逻辑：

1. IMU 测量去 **bias** → 离散积分 $\Delta R, \Delta v, \Delta p$
2. 一阶偏导累积 → 线性化 **bias** 对 Δ 的影响
3. 噪声协方差传播 → IMU 因子权重
4. 关键帧到来 → 构造因子图 → `reset` 继续下一段

它实际上就是一个连续时间 IMU 的「快速线性化积分器」，核心优势：

- 避免每次优化都重新积分全部高频 IMU
- 可直接作为 gtsam 因子图优化输入
- 支持 **bias** 更新和噪声传播

如果你需要，我可以画一个完整的 $\Delta R, \Delta v, \Delta p$ 以及偏置雅可比累积图，可视化每个 IMU 样本是怎么影响预积分量和协方差的，非常直观。

你希望我画这个图吗？

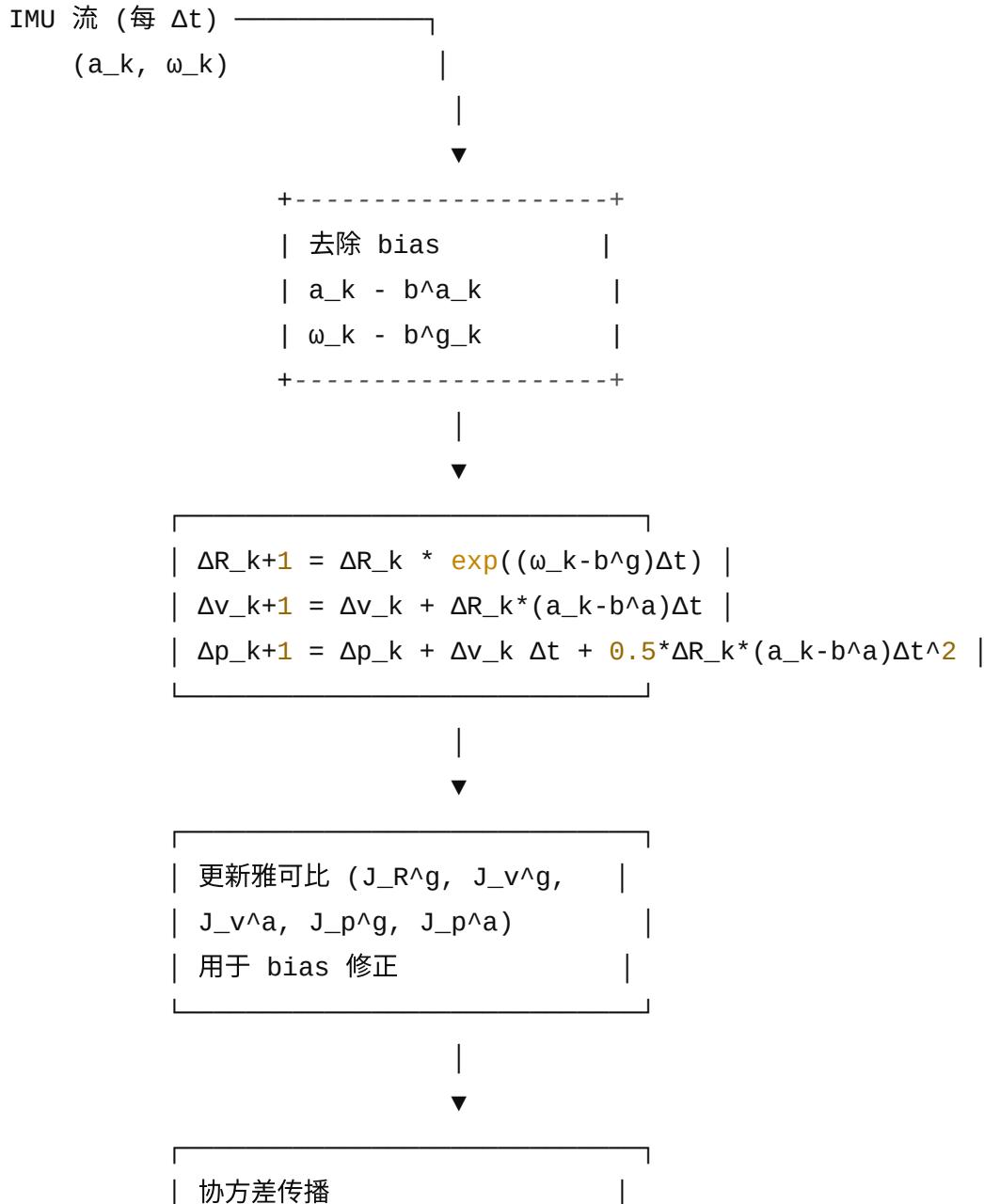


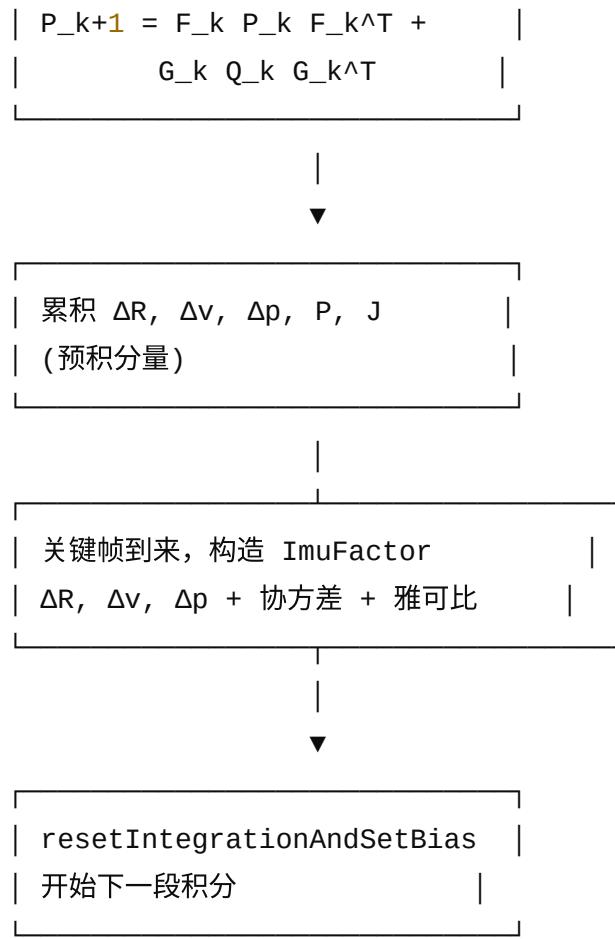
可以的

好，我给你画一个完整的预积分流程图，把 ΔR 、 Δv 、 Δp 、偏置雅可比和协方差传播直观展示出来，让你能像看流水线一样理解 PreintegratedImuMeasurements 内部工作原理。

预积分可视化流程图

lua

 Copy code



◆ 流程说明：

1. 每条 IMU 测量 → 去除当前 bias
2. 离散积分 → 更新 $\Delta R, \Delta v, \Delta p$
3. 偏置雅可比累积 → 支持优化时线性修正
4. 协方差传播 → 用于构造因子图的权重
5. 关键帧到来 → 构造 `ImuFactor`
6. 重置积分器 → 准备下一段积分

如果你愿意，我可以画一个带向量箭头和旋转矩阵的几何示意图，更直观展示 ΔR 如何作用在 $\Delta v, \Delta p$ 上，跟 IMU 世界/机体坐标系对应关系一目了然。

你希望我画这个几何图吗？



F_k 是如何推导的

好的，我们来仔细推导 PreintegratedImuMeasurements 中的 状态转移矩阵 F_k (雅可比)。这是预积分里非常核心的内容，因为它用来传播 协方差 P 和 bias 对预积分量的影响。我会一步步讲清楚，公式和逻辑都完整展示。

1 状态定义

在 IMU 预积分里，我们定义离散时间的状态为：

$$x_k = \begin{bmatrix} \delta\theta_k \\ \delta v_k \\ \delta p_k \\ \delta b_k^g \\ \delta b_k^a \end{bmatrix}$$

- $\delta\theta_k \in \mathbb{R}^3$: 旋转小误差 ($so(3)$)
- $\delta v_k \in \mathbb{R}^3$: 速度误差
- $\delta p_k \in \mathbb{R}^3$: 位置误差
- $\delta b_k^g, \delta b_k^a \in \mathbb{R}^3$: IMU 偏置误差

2 离散误差动力学

假设 IMU 预积分公式：

$$\Delta R_{k+1} = \Delta R_k \exp((\tilde{\omega}_k - b_k^g)\Delta t)$$

$$\Delta v_{k+1} = \Delta v_k + \Delta R_k(\tilde{a}_k - b_k^a)\Delta t$$

$$\Delta p_{k+1} = \Delta p_k + \Delta v_k \Delta t + \frac{1}{2} \Delta R_k(\tilde{a}_k - b_k^a)\Delta t^2$$

考虑小误差，定义：

$$\Delta R_k = \bar{\Delta R}_k \exp(\delta\theta_k)$$

$$\Delta v_k = \bar{\Delta v}_k + \delta v_k$$

$$\Delta p_k = \bar{\Delta p}_k + \delta p_k$$

线性化后可得到误差动力学（连续形式）：

$$\begin{aligned}\delta\theta_{k+1} &= \delta\theta_k - J_r^{-1} \delta b_k^g \Delta t - n_k^g \Delta t \\ \delta v_{k+1} &= \delta v_k - \Delta R_k \delta b_k^a \Delta t + [\Delta R_k (\tilde{a}_k - b_k^a)]_\times \delta\theta_k \Delta t - \Delta R_k n_k^a \Delta t \\ \delta p_{k+1} &= \delta p_k + \delta v_k \Delta t\end{aligned}$$

其中 $[v]_\times$ 表示向量的反对称矩阵，产生旋转作用。

3 状态转移矩阵 F_k 定义

离散误差动力学可写成矩阵形式：

$$x_{k+1} = F_k x_k + G_k n_k$$

其中 $x_k = [\delta\theta_k, \delta v_k, \delta p_k, \delta b_k^g, \delta b_k^a]^T$, $n_k = [n_k^g, n_k^a]^T$ 。

根据前面的线性化公式， F_k 的结构：

$$F_k = \begin{bmatrix} I_3 & 0 & 0 & -I_3 \Delta t & 0 \\ [\Delta R_k (\tilde{a}_k - b_k^a)]_\times \Delta t & I_3 & 0 & 0 & -\Delta R_k \Delta t \\ 0 & I_3 \Delta t & I_3 & 0 & 0 \\ 0 & 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & 0 & I_3 \end{bmatrix}$$

逐块解释：

块	作用
I_3	旋转/速度/位置自身误差传递
$-I_3 \Delta t$	陀螺仪偏置对旋转误差影响
$[\Delta R_k (a - b^a)]_\times \Delta t$	旋转误差对速度误差的影响
$-\Delta R_k \Delta t$	加速度偏置对速度误差的影响
$I_3 \Delta t$	速度误差对位置误差的影响

注意：位置误差仅受速度误差线性影响；旋转误差影响速度，速度误差影响位置，偏置影响旋转和速度。

4 GTSAM 的实现

在 GTSAM 中：

cpp

Copy code

```
Matrix Fk = Matrix::Identity(15, 15);
Fk.block<3, 3>(0, 3) = Zeros; // 旋转对速度
Fk.block<3, 3>(0, 6) = Zeros; // 旋转对位置
Fk.block<3, 3>(0, 9) = -I3 * dt; // 旋转对陀螺仪偏置
Fk.block<3, 3>(1, 0) = -DeltaR * skew(acc)* dt; // 速度对旋转误差
Fk.block<3, 3>(1, 12) = -DeltaR * dt; // 速度对加速度偏置
Fk.block<3, 3>(2, 3) = I3 * dt; // 位置对速度误差
```

然后协方差传播：

$$P_{k+1} = F_k P_k F_k^T + G_k Q_k G_k^T$$

✓ 5 总结

1. F_k 是从 IMU 离散积分公式线性化得到的雅可比矩阵
2. 核心思想：**旋转误差影响速度误差 → 速度误差影响位置误差**
3. 偏置影响：陀螺仪 bias 影响 ΔR ，进而影响 Δv ；加速度 bias 直接影响 Δv
4. 协方差传播公式直接使用 F_k 和 G_k

如果你愿意，我可以画一个带箭头的 $\Delta R \rightarrow \Delta v \rightarrow \Delta p$ 误差传播示意图，把 F_k 的每个块对应的作用直观展示出来，非常适合理解预积分协方差是如何累积的。

你希望我画这个吗？



