```
        gtsam::PriorFactor<gtsam::Pose3> priorPose(X(0),
prevPose_, updatedPoseNoise);
        graphFactors.add(priorPose);
        // 把上一次优化出的速度作为重新初始化的priorVel
        gtsam::PriorFactor<gtsam::Vector3> priorVel(V(0),
prevVel_, updatedVelNoise);
        graphFactors.add(priorVel);
        // 把上一次优化出的bias作为重新初始化的priorBias
        gtsam::PriorFactor<gtsam::imuBias::ConstantBias>
priorBias(B(0), prevBias_, updatedBiasNoise);
        graphFactors.add(priorBias);
        // 将prior状态设置成初始估计值
        graphValues.insert(X(0), prevPose_);
        graphValues.insert(V(0), prevVel_);
        graphValues.insert(B(0), prevBias_);
这个graphvalue和graphnode什么区别，
```

下面一句话先给你一个**最核心的理解**：

> **在 GTSAM 里：Factor Graph = Nodes（变量） + Factors（约束）；**
> **Values 是“变量的当前取值（初始解或优化解）”，而不是图本身的一部分。**

因此：

- **graphFactors（或 graph）** —— 存的是 **Factor（约束）**
- **graphNodes（变量）** —— 实际上在 GTSAM 不直接存在一个叫 “graphNodes” 的结构，节点由 **Keys** 决定（例如 X(0), V(0), B(0) 这些 key）。
- **graphValues（Values）** —— 存的是 **变量的值**（初始估计值或者优化后的结果）

🔍 你代码里的三个东西分别是什么？

1. graphFactors（又叫 NonlinearFactorGraph）

它只存“因子”（约束），例如：

cpp 复制代码

```
gtsam::PriorFactor<gtsam::Pose3> priorPose(...)
graphFactors.add(priorPose);
```

因子使用 key（如 X(0), V(0)）来连接图中的变量，但它不管变量值是多少。

🔍 2. graphNodes（概念，不是实际结构）

GTSAM 的图节点其实由 factor 的 key 决定：

例如：

cpp 复制代码

```
priorPose connects to X(0)
priorVel connects to V(0)
priorBias connects to B(0)
```

你一旦给一个 key 添加 factor，它就算是图里的一个节点。

GTSAM 里没有一个独立的 Node 容器。

🔍 3. graphValues（Values）

这是一个 **Key → Value 的字典**，存每个变量的“当前估计”：

cpp 复制代码

```
graphValues.insert(X(0), prevPose_);
graphValues.insert(V(0), prevVel_);
graphValues.insert(B(0), prevBias_);
```

为什么必须提供 Values?

因为优化器需要一个“初始猜测”才能开始求解。

🧠 总结成一句话

名称	在 GTSAM 的作用
FactorGraph / graphFactors	存因子（约束），定义变量之间的关系
Node（隐含概念）	由 key 决定的变量，不需要单独创建
Values / graphValues	变量的具体取值（初值或优化结果）

📌 更形象的理解（非常重要）

把 GTSAM 想成一个“解方程”的框架：

- **Factor = 方程约束**
- **Key = 方程中的未知量名字**
- **Values = 未知量的当前猜测**

优化器在“方程（factor）”和“变量的初猜（values）”之间求出最优解。

如果你需要，我还能帮你画一个 ASCII 图讲清楚 factor、node、values 的结构关系，会更形象。

先验观测约束后，原有node也要有一个初始估计值吗

要的！必须要有！

即使你给某个变量（node）加了先验因子（prior factor），**它仍然需要在 Values 里提供一个初始估计值**，否则优化器无法启动。

🔍 为什么“加了先验因子”还要初始估计？

因为：

- **先验因子（PriorFactor）只是一个约束**（告诉优化器：这个变量应该接近某个值）

- **Values 才是未知量的初始猜测**

GTSAM 是一个非线性优化器，它必须从某个初始值开始迭代。

所以：

> **加因子 != 给初始值**

两者作用完全不同。

✔ 举个例子（你代码中的情况）

你加了先验因子：

cpp 复制代码

```cpp
gtsam::PriorFactor<gtsam::Pose3> priorPose(X(0), prevPose_, noise);
graphFactors.add(priorPose);
```

但你还必须：

cpp 复制代码

```cpp
graphValues.insert(X(0), prevPose_);
```

否则 optimizer 会报以下错误：

pgsql 复制代码

```pgsql
Values has no initial estimate for key X(0)
```

🔍 那如果我 *不给 prior factor*，只给初始值行不行？

可以，但这时变量只是 **有一个起点，没有被约束**。

优化器会根据其他因子（例如 IMU、GPS、Lidar）来推它。

🔍 那如果我 *给了 prior factor*，但不给初始值会怎样?

会报错。

因为 prior factor 只定义“约束”，不定义“初始估计”。

优化器需要在迭代的起点就知道：“变量现在的值是多少，用来开始求解”。

🧠 更形象地理解

可以把 factor graph 看成下面这样：

- **Factor**：方程
- **Values**：方程里未知数的初始值
- **Optimizer**：拿着初值，不断更新变量

如果你只有方程没有初值，优化器无从下手。

✔ 结论（最重要一句）

> **任何出现在因子图中的变量（node）都必须在 Values 里提供初始估计，无论它有没有 prior factor。**

prior 只是一个约束，不是初始估计。

如果你愿意，我可以再画一个更直观的图，明确对比：

- factor
- node(key)
- values

  之间的关系。

// 使用IMU预积分的结果构建IMU因子，并加入因子图中

```
const gtsam::PreintegratedImuMeasurements&
preint_imu = dynamic_cast<const
gtsam::PreintegratedImuMeasurements&>
(*imuIntegratorOpt_);
gtsam::ImuFactor imu_factor(X(key - 1), V(key - 1),
X(key), V(key), B(key - 1), preint_imu);
graphFactors.add(imu_factor);
// 添加IMU的bias的BetweenFactor
// 这里的deltaTij()获取的是IMU预积分器从上一次积分
输入到当前输入的时间
// noiseModelBetweenBias是我们提前标定的IMU的偏
差的噪声
// 两者的乘积就等于这两个时刻之间IMU偏差的漂移

graphFactors.add(gtsam::BetweenFactor<gtsam::imuBias
::ConstantBias>(B(key - 1), B(key),
gtsam::imuBias::ConstantBias(),

gtsam::noiseModel::Diagonal::Sigmas(sqrt(imuIntegrator
Opt_->deltaTij()) * noiseModelBetweenBias)));

// 将雷达的位姿平移到IMU坐标系（只做平移）
gtsam::Pose3 curPose =
lidarPose.compose(lidar2Imu);
// 添加Pose因子。这里的degenerate调整噪声的大
小。correctionNoise2是大噪声
gtsam::PriorFactor<gtsam::Pose3>
pose_factor(X(key), curPose, degenerate ?
correctionNoise2 : correctionNoise);
graphFactors.add(pose_factor);
// imuIntegratorOpt_->predict输入之前时刻的状态和
偏差，预测当前时刻的状态
gtsam::NavState propState_ = imuIntegratorOpt_-
>predict(prevState_, prevBias_);
// 将IMU预积分的结果作为当前时刻因子图变量的初值
graphValues.insert(X(key), propState_.pose());
graphValues.insert(V(key), propState_.v());
graphValues.insert(B(key), prevBias_);
```

这里的imufactor和betweentfactor是有什么区别

下面给你一个**最清晰、最实战角度**的解释：
在 VIO/LIO-SAM 这种系统里，**ImuFactor 和 BetweenFactor（bias 部分）是配套使用的**，分别约束不同的东西。

🧩 一句话总结两者的区别

因子类型	约束对象	用途
ImuFactor	{X(k-1), V(k-1), B(k-1), X(k), V(k)}（姿态、速度、Bias）	**约束运动学**：用 IMU 预积分结果约束前后两个时刻的 Pose 和 Velocity
BetweenFactor（Bias）	{B(k-1), B(k)}	**约束 IMU 偏置随时间的漂移模型（Brownian motion）**

🧠 为什么 IMU 需要两个因子?

🔹 1. ImuFactor 约束“运动学”

ImuFactor 使用预积分量：

- ΔR（旋转变化）
- Δv（速度变化）
- Δp（位置变化）

这些量都依赖 bias：

$$f(X_{k-1}, V_{k-1}, B_{k-1}) \rightarrow (X_k, V_k)$$

它在 factor graph 中提供的是：

> **运动方程约束（动态模型）**

例如：

- 车辆前一帧和后一帧应该符合 IMU 的惯性运动学
- IMU 积分告诉你 X(k) 大概应该是从 X(k-1) 跨过去的多少

但是注意：

ImuFactor 不能强制 B(k) 如何演化

它只是使用 B(k-1)，但不输出 B(k)。

因此要补一个 bias 的约束因子。

🔹 2. BetweenFactor（Bias）约束偏置随时间的漂移

IMU bias（gyroscope bias / accelerometer bias）在物理中通常满足 **随机游走模型（Random Walk）**：

$$B(k) = B(k-1) + \eta$$

η 是高斯噪声，方差与时间有关：

$$\sigma^2 = \delta t \cdot \sigma^2_{\text{bias\_random\_walk}}$$

你的代码等效于：

cpp 复制代码

```
BetweenFactor(B(k-1), B(k), 0, sqrt(dt) * biasRandomWalkNoise)
```

意思是：

> **Bias(k) 应该与 Bias(k-1) 接近，但可以有随机游走噪声**

它只约束 bias，不涉及 pose 或 velocity。

🎯 为什么不能只用 ImuFactor?

因为 ImuFactor 只使用 B(k-1) 来修正预积分，并不提供约束告诉图：

- B(k) 应该是什么
- B(k) 和 B(k-1) 的关系

如果没有 BetweenFactor，则 bias 会漂到无穷大，导致 pose 和 velocity 都无法收敛。

🎯 为什么不能只用 BetweenFactor?

BetweenFactor 只是说：

B(k) 和 B(k-1) 要接近

但不告诉 pose / velocity 该怎么从 IMU 得到。

运动学还是要靠 ImuFactor。

🧩 两者合在一起才能构成完整的 IMU 模型

1. **ImuFactor**：
   - 提供 “IMU 驱动的运动学”
   - 关联 pose、velocity、bias
2. **BetweenFactor（Bias）**：
   - 提供“IMU bias 的随机游走先验”
   - 只关联 bias 变量

这两个因子合在一起使得：

- pose 受到 IMU 积分的约束
- velocity 受到 IMU 积分的约束
- bias 受到随机游走约束
- 整个状态（X, V, B）可以稳定优化

🧩 更直观的 “状态图” 示意（非常形象）

lua 复制代码

```
X(k-1) --\
          \                        /-- X(k)
V(k-1) ---- ImuFactor ---------- /
          /                    /
B(k-1) --/                   /
                            /
B(k-1) ---- BetweenFactor --- B(k)
```

🔚 最简结论（非常重要）

- **ImuFactor = 动力学约束（pose、速度、bias）**
- **BetweenFactor = bias 随时间的漂移模型**
- 两者缺一不可，共同组成完整 IMU 模型

如果你想，我可以继续讲：

- 为什么 preintegration 需要 Bias
- BetweenFactor 的 sigma 如何选
- 为什么 bias 的 Between 噪声要乘 √dt
- LIO-SAM IMU 初始化方式

你想继续深入哪部分？

gtsam::NavState propState_ = imuIntegratorOpt_->predict(prevState_, prevBias_);解读下

下面是对这句代码 **最完整、最直观、结合 IMU 预积分理论的深度解读**，适合你这样已经在读 LIO-SAM/IMU 因子代码的人。

🧩 代码本身

cpp 复制代码

```
gtsam::NavState propState_ = imuIntegratorOpt_->predict(prevState_, prevBi
```

意思是：

> **用 IMU 预积分器（包含 ΔR、Δv、Δp）去预测当前时刻的导航状态（姿态 + 速度 + 位置）**

这是 LIO-SAM 里用于 **给优化器提供变量初始值** 的关键步骤。

🧠 1. `prevState_` 是什么?

它是上一帧的导航状态，包含：

- 姿态 R(k-1)
- 位置 p(k-1)
- 速度 v(k-1)

即 NavState：

cpp 复制代码

```cpp
class NavState {
  Rot3 R;
  Point3 p;
  Vector3 v;
}
```

🧠 2. `prevBias_` 是什么?

IMU bias，包括：

- 加速度计偏置 ba(k-1)
- 陀螺仪偏置 bg(k-1)

类型：

cpp 复制代码

```cpp
gtsam::imuBias::ConstantBias
```

🧠 3. 预积分器 `imuIntegratorOpt_` 里存了什么?

它存的是从上一帧到当前帧的 **IMU measurements 的预积分量**：

- ΔR（预积分旋转）
- Δv（预积分速度变化）
- Δp（预积分位置变化）

- 预积分的协方差矩阵
- 使用的时间 Δt

其数学模型：

$$\Delta R = \int_{t_{k-1}}^{t_k} (\omega - b_g) dt$$

$$\Delta v = \int R(\tau)(a - b_a) d\tau$$

$$\Delta p = \iint R(\tau)(a - b_a) d\tau^2$$

这些都已经被预积过，所以预测时无需再重积分原始 IMU 数据。

🧠 4. `predict(prevState_, prevBias_)` 数学意义

根据 IMU 预积分模型：

$$R_k = R_{k-1} \cdot \Delta R$$

$$v_k = v_{k-1} + g\Delta t + R_{k-1}\Delta v$$

$$p_k = p_{k-1} + v_{k-1}\Delta t + \frac{1}{2} g \Delta t^2 + R_{k-1}\Delta p$$

predict() 做的就是这些推算。

它只使用：

- **previous NavState**
- **previous Bias**
- **预积分 ΔR, Δv, Δp**

🧩 5. 这里的预测结果用于什么？

非常关键：

👉 用于提供当前优化变量的初始估计（initial guess）

你后面这么写：

cpp 复制代码

```cpp
graphValues.insert(X(key), propState_.pose());
graphValues.insert(V(key), propState_.v());
graphValues.insert(B(key), prevBias_);
```

也就是说：

- Pose(key) ← 用 predict 的结果
- Vel(key) ← 用 predict 的结果
- Bias(key) ← 用上一帧 bias（因为预积分没有输出 bias）

LIO-SAM 认为 IMU bias 在两帧之间保持不变（由 BetweenFactor 约束其随机游走）。

🧩 6. 为什么需要 predict()?

因为：

- IMU factor 是非线性约束，优化需要 good initial guess
- 直接让 X(k) 初始值 = X(k-1) 会导致优化难收敛
- 用 predict() 得到的 X(k) 初值非常接近真实轨迹

也就是说：

> **predict() 是对真实运动轨迹最准确的一阶预测，用于帮助优化器快速收敛**

🧠 7. 预测与优化之间的关系

预测 ≠ 最终状态

预测是初值

最终由 Factor Graph 结合：

- IMU factor
- Bias between factor
- Lidar pose factor
- Prior factor

共同优化出来。

🧩 最终总结（最重要的）

`predict(prevState_, prevBias_)` 的作用：

1. **用 IMU 预积分结果把上一帧状态外推到当前帧**
2. **推算当前 pose / velocity 的初始估计值**
3. **为 GTSAM 优化提供 good initial guess（非常关键）**
4. **不负责 bias 的预测（bias 交给 BetweenFactor）**

这句话你要记住：

> **IMU predict() 给优化器提供初值，IMU Factor 负责误差约束，两者配合才能得到正确的优化结果。**

如果你愿意，我还能给你：

- predict 在 gtsam 源码中的关键实现
- 手推预测公式
- 与 LIO-SAM 的整体流程图融合分析