

mapOptimization.cpp

File

liosam中地图全局优化，这部分我有些不理解，比如回环优化结束后，自身坐标一定会发生变化，这个变化我没在代码中找到呢，就是map-》odom坐标系的变化

下面我用 **LIO-SAM** 的真实代码逻辑 来帮你把 “回环之后自身坐标一定改变，但代码中为什么没看到 `map → odom` 的变化” 这个核心问题解释清楚。

🌟核心答案（一句话）

LIO-SAM 中回环优化改变的是 `map → odom` 的 TF，而不是 `odom → base_link`。

`map → odom` 的变化不是显式写在 `mapOptimization.cpp` 中，而是由 `TransformFusion` 节点根据优化后的 `poseGraph`（`map` 下的位姿）自动计算并发布。

🧠为什么你在 `mapOptimization.cpp` 找不到 `map → odom` 更新？

原因：

`mapOptimization.cpp` 只做两件事：

1. 建立因子图（IMU + 激光 + 回环因子）
2. 进行优化并更新关键帧 pose（都在 `map` 坐标系）

👉 它从来不负责广播 TF，也不会改 `odom` 坐标系。

🌍LIO-SAM 的三个关键坐标系

arduino

📄 Copy code

```
map ----- 由因子图优化（回环会使其 jump）
|
| map→odom 由 TransformFusion 维持，使 odom 平滑
|
```

```
odom ----- IMU 预积分预测（无回环跳变）
|
| odom→base_link 由 IMU 里程计给出
|
base_link
```

map → odom 是在哪里更新的？

它在 **TransformFusion.cpp** 内动态计算，而不是在 mapOptimization 中硬编码更新。

路径：

```
bash
```

 Copy code

```
LIO-SAM/src/mapOptmization/transformFusion.cpp
```

关键代码：

publishTF() 内部逻辑：

```
cpp
```

 Copy code

```
tf::Transform tF;
tF.setOrigin(tf::Vector3(transformAftMapped[3], transformAftMapped[4], tra
tF.setRotation(tf::Quaternion(...));
tfBroadcaster.sendTransform(tf::StampedTransform(tF, time, "map", "odom"));
```

这里的 transformAftMapped 就是 因子图优化后的位姿（map 下）。

map → odom 是如何产生的？

TransformFusion 做了一件非常重要的事：

将 map 下的优化位姿 - odom 下的IMU里程计 = map → odom 的 TF

即：

```
ini
```

 Copy code

$$T_{\text{map_odom}} = T_{\text{map_base}} * (T_{\text{odom_base}})^{-1}$$

这样可以保证：

- map 跳变（回环导致）✓
- odom 依旧连续、平滑，不会跳变 ✓
- base_link 在 odom 下是连续的 ✓

关键理解：回环优化后坐标系“跳”的不是 odom，而是 map → odom

举例：

- 优化后位姿突然从 (10,10) 跳到 (7,8)
- odom 下的 IMU 位姿没变（因为 odom 不做闭环）
- 因此 map → odom 必须 变化，用来吸收这部分跳变

所以你看到的现象是：

关键帧在 map 坐标系中跳变

但是

 odom → base_link 不跳

因为

 map → odom 被实时修改以吸收优化偏差

✓ 为什么你在代码中找不到“map → odom 被修改”的位置？

因为：

1. 它不是显式修改某个变量，而是每次 TF 发布时动态计算。
2. 它不在 mapOptimization.cpp，而是在 TransformFusion.cpp 中。

mapOptimization 只负责更新全局位姿（map），真正处理 TF 的是 TransformFusion。

总结（非常重要）

LIO-SAM 中：

- 回环优化确实让 map 下的位姿跳变 ✓
- 但 odom 下的位姿不跳变 ✓
- 所以 map → odom TF 会实时变化 ✓
- 这一过程发生在 TransformFusion.cpp ✓
- mapOptimization.cpp 本身不会发布或修改 map → odom ✓

如果你需要的话，我可以进一步：

- ✓ 画出完整 TF 流程图（非常清晰）
- ✓ 给你标注 mapOptimization 和 TransformFusion 的核心段代码
- ✓ 分析 LIO-SAM 为什么必须设计 map → odom 来保持 odom 平滑

你想继续深入哪一部分？



这种回环优化后的pose,会影响雷达里程计吗，由于imu订阅了雷达里程计，这种跳变会有影响吗

结论先说：

🔥 回环优化后的 pose（map 下跳变）

不会直接影响雷达里程计（激光里程计），
不会影响 IMU 订阅的雷达里程计，
不会让里程计跳变。

LIO-SAM 的设计就是为了 让回环“只影响 map，不影响 odom”。

下面我给你拆开讲清楚。

✓ 1. 回环导致跳变的是：关键帧在 map 坐标系下的 pose

在 LIO-SAM 中：

- **map** 下的关键帧坐标 → 会因为回环优化发生突变 (jump)
- **odom** 下的当前帧位姿 (激光里程计) → 不会突变

因为激光里程计是 **实时估计**，它工作在 **odom** 坐标系下，不受 map 优化影响。

✓ 2. 为什么激光里程计不会被回环影响？

因为 lio_sam 的架构是：

go

Copy code

```
IMU / 激光前端  —更新—>  odom→base_link (平滑连续, real-time)
                                     |
                                     | 基于这个进行特征提取/匹配
                                     |
因子图优化  —只更新—>  map 下的关键帧位姿 (非实时, 会跳变)
```

也就是说：

激光里程计只依赖于 IMU 预积分和激光帧间配准，不依赖 map 坐标系。

map 优化只用于：

- 全局一致性
- 回环校正后的轨迹、地图

它跟实时前端里程计无关。

✓ 3. IMU 为什么不会被回环影响？

你说的“IMU 订阅了雷达里程计”，指的是：

IMU 预积分因子使用 **雷达里程计的姿态**作为优化约束 (**GTSAM**)。

但是非常关键：

IMU 不会直接使用 map 下跳变的位姿做积分工作。

IMU 前端做的是：

根据上一帧 odom 下的预测 (T_{odom}) 进行预积分

而 map 下的位姿跳变后, 对 IMU 里程计有什么影响?

答案是:

🔥 完全没有影响, 因为优化后的 pose 没有写回到 IMU 前端中。

IMU 用的状态是:

nginx

 Copy code

```
transformCur      (实时)
transformBefMapped (未优化前)
transformAftMapped (优化后, 只用于 TF)
```

其中影响 TF 和全局位置的是 transformAftMapped, 但是不会写入 IMU 前端的 odom 滤波器中。

✅ 4. 回环跳变如何避免影响前端?

靠的是 LIO-SAM 的坐标体系划分:

🌐 map (全局坐标)

- 会跳变 (闭环优化)
- 后端因子图使用

🚗 odom (前端坐标)

- 连续平滑 (用于里程计)
- IMU、激光前端使用

🤖 base_link

- 机器人身体本体坐标系

然后靠一个动态 TF:

arduino

 Copy code

map → odom

用来吸收全部的优化跳变。

因此 odom 和 base_link 的关系不变，前端不会跳变。

🔥 5. 所以最终事实是：

部分	回环后跳变？	是否影响前端？
map 下关键帧 pose	✓ 会跳变	✗ 不影响
map → odom TF	✓ 会变化	✗ 不影响
odom → base_link（前端里程计）	✗ 不跳变	✓ 前端核心
激光里程计	✗ 不跳变	✓ 实时连续
IMU 预积分	✗ 不受跳变影响	✓ 正常运行

🧠 用一句话总结

回环优化是后端过程，只影响 map 下的关键帧和 map → odom TF，而前端的 IMU+激光里程计完全不受影响，所以不会跳变。

如果你愿意，我可以进一步给你画：

- 完整 TF 流程图（map/odom/base_link）
- 激光前端与后端优化的数据流图
- 回环发生前后 pose、TF 怎样变化（示意图）

你需要这部分吗？

📄 🍷 🗨️ ⬆️ ↺️ ...

我还有一个疑问，map坐标系的更新频率和odom是一样的吗

结论先说：

 **map 坐标系的“更新频率”远低于 odom，不是一样的。**

更准确的说法是：

- **odom 是实时更新（几十 Hz）**
- **map 只在关键帧插入、回环检测、后端优化完成时才更新（不定时、低频）**

所以两者的更新机制和时间特性完全不同。

下面我详细拆开解释。

1. 什么叫“odom 高频更新”？

在 LIO-SAM 中：

odom 是由 IMU + 当前激光配准（scan2scan 或 scan2map）实时输出的

更新频率通常：

- **10Hz-20Hz 的激光帧频率**
- **外加 IMU 预积分（200Hz-500Hz）**

因此 **odom → base_link** 是高频、连续、实时的。

2. 那 map 的更新频率是什么？

map 坐标系中的关键帧 pose 更新来源于：

1. 关键帧插入（通常每隔 1m 或一定旋转角）
2. 回环检测（不一定发生）
3. 因子图优化（运行周期通常为几秒~几十秒）
4. 优化后重新计算关键帧 pose（较慢）

所以 **map 坐标系不实时更新**，特点是：

- 不以定时器驱动
- 不跟激光帧同步
- 不跟 IMU 同步
- 不连续、不平滑

它是 **按事件更新 (Event-Driven)**。

map 什么时候变化？

只有两种情况：

情况 1：插入关键帧（局部优化）

map 下的轨迹轻微更新（很慢）

情况 2：回环因子加入 + 后端全局优化

map pose 会发生“跳变”（更慢）

这就是你看到的优化“跳动”原因。

其更新间隔通常为：0.5 ~ 5 秒 而非 10Hz。

3. map 和 odom 是如何对齐的？

靠的是这条 TF：

```
arduino
```

 Copy code

```
map → odom
```

它由 TransformFusion 以 **实时频率** 发布，但注意：

TF 的发布频率高（比如 50Hz）

但 map 的实际位姿变化低频甚至不变。

TF 发布的内容来源于：

ini

 Copy code

```
T_map_odom = T_map_robot / T_odom_robot
```

其中：

- T_odom_robot 高频连续
- T_map_robot 低频、不定时更新

因此：

TF 的频率高，但 TF 中的 map 部分大部分时间是恒定的。

总结（最关键）

map 更新 = 低频（事件驱动，按关键帧/回环/优化触发）

odom 更新 = 高频（10Hz~20Hz 激光 + IMU 200Hz）

二者更新完全不同步、不同频率

由 map → odom TF 来桥接它们，使前端保持 smooth，后端保持 global consistency

如果你愿意，我还可以给你：

- ◆ 一张完整的 TF 更新示意图（map、odom、base_link）
- ◆ map/odom 联动的数学公式
- ◆ 回环前后具体示例数值变化（装逼但能看懂）

你需要哪一个？

     ...

那既然map更新频率低，可是robot实时在移动

你的疑惑非常关键，也是很多人第一次理解 LIO-SAM 时最难跨过去的点：

? map 更新频率低，但机器人在实时移动，怎么不出问题？

答案是：

****🔥 机器人实时移动用的是 odom，不用 map。**

map 是后端的世界基准，odom 是前端实时里程计两个体系分工明确。 **

你可以把它们想象成：

- **odom = 实时的车轮速度表**（连续但会漂移）
- **map = 后端的导航地图**（不连续，但全局正确）
- **map → odom TF = 把两者对齐的桥梁**

下面我把整个逻辑讲到你彻底理解。

🚗 1. 机器人实时移动，用的是 odom 坐标系（高频）

机器人实时在动：

- IMU 预积分：200Hz–500Hz
- 激光里程计（scan-to-map）：10–20Hz

这些实时信息全部在 **odom** 坐标系下进行。

所以机器人的“实时运动轨迹”，实际上是：

```
nginx
```

📄 Copy code

```
odom → base_link
```

它完全与 map 无关。

★无论 map 更新频率多低，都不影响 odom 实时跟踪。

🌐 2. map 更新频率低，因为 map 只用于“全局一致性”

map 坐标系下的位姿来自：

- 关键帧插入
- 回环因子加入
- 因子图优化

这些都不是实时的，因此 map 的更新频率低。

但是你要注意：

map 下的坐标不是实时运动坐标，而是“优化后的历史关键帧坐标”。

机器人实时行动时根本不会用 map 的最新坐标来驱动运动。



3. map 与 odom 的连接靠 TF：map → odom

机器人实时位置发布的是：

```
nginx
```

[Copy code](#)

```
odom → base_link
```

但导航/可视化等组件想看 global pose，需要 map→base_link：

```
arduino
```

[Copy code](#)

```
map → odom → base_link
```

其中 map → odom 是一个不断修正的变换（TransformFusion计算）。

它做的事情是：

**用 map 的低频全局优化结果

去校正 odom 的高频漂移**

并且：

- 不会让 odom 跳变
- 不会让 base_link 跳变
- 只会改变 map → odom 的 TF 来吸收跳变

4. 动态例子（最关键）

假设机器人正在实时移动：

SCSS

 Copy code

odom 下：

base_link 连续从 $(0, 0) \rightarrow (1, 0) \rightarrow (2, 0) \rightarrow (3, 0) \dots$

某一时刻回环发生：

- 后端 fusing 后发现：
map 坐标下机器人“真实”轨迹应该偏左一点

于是 map 中关键帧位置会突然跳变，比如：

SCSS

 Copy code

$(3, 0) \rightarrow (2.7, 0.1)$

但是你猜 odom 会跳变吗？

不会！

那怎么对齐的？

TransformFusion 做了一件事：

SCSS

 Copy code

map→odom 从 $(0, 0)$ 改为 $(-0.3, -0.1)$

于是：

arduino

 Copy code

$\text{map} = \text{odom} + \text{map} \rightarrow \text{odom}$

仍然保持全局一致性。

而 odom 本身毫无变化，继续平滑运行。



5. 所以结构其实是这样的：

实时运动轨迹 (odom)

- 高频
- 平滑
- 有累积漂移
- 不受 map 更新影响

全局地图轨迹 (map)

- 低频
- 不平滑 (回环会跳)
- 全局一致性好
- 不参与实时运动

map → odom

- 高频发布 (比如 50Hz)
- 动态补偿 map 的跳变
- 保证 odom 和 map 对齐



用一句话总结 (最关键)

****机器人实时运动用的是 odom (前端高频)**

map 只是历史关键帧的全局校正 (后端低频)

二者通过 map → odom TF 实时结合**

因此：

- 机器人可以实时移动
- map 可以低频更新
- odom 不会被优化打断
- 前端不会因回环跳变

整个系统非常稳定。

如果你愿意，我可以给你画一张非常清晰的图，展示：

- odom 曲线（实时但漂移）
- map 曲线（不实时但全局正确）
- map \rightarrow odom 如何吸收跳变

要不要我画一张？

     ...

那在rviz中看到的全局坐标实际上是map+odom吗，

结论：

****🔥 是的！**

你在 Rviz 中看到的机器人的全局位置，就是：

map \rightarrow odom \rightarrow base_link

这三个 TF 相乘得到的最终姿态。 **

换句话说：

Rviz 中的机器人坐标 **不是 odom**，

也 **不是 map**，

而是：

****map 坐标系下的 base_link（全局位姿）**

= map \rightarrow odom \times odom \rightarrow base_link **

 **为什么 Rviz 要用 map 作为根坐标系？**

因为：

- map 是**全局一致**的坐标
- odom 是**连续但漂移**的坐标
- base_link 是机器人本体坐标

Rviz 会自动从 TF 树中找到 root（通常是 map），然后计算：

ini

 Copy code

$$T_map_base = T_map_odom \cdot T_odom_base$$

这样显示出来的就是机器人全局位置。

✖ 举个具体例子

假设：

在 odom 系下：

```
rust
```

[Copy code](#)

```
odom->base_link = (10, 0)
```

而回环后系统发现真正全局位置应该稍微左一点，于是调整：

```
arduino
```

[Copy code](#)

```
map->odom = (-2, 1)
```

那么 Rviz 中呈现的 base_link 就是：

```
arduino
```

[Copy code](#)

```
map->base_link = (-2, 1) × (10, 0) = (8, 1)
```

🔥 核心点（非常重要）

即使 **map 跳变**（回环优化），

只会引起：

```
arduino
```

[Copy code](#)

```
map->odom 的 TF 跳变
```

但：

- **odom → base_link 不跳变**（前端平滑）
- **base_link 在 Rviz 中也不会跳来跳去**

因为 $\text{map} \rightarrow \text{odom}$ 会吸收所有跳变。

简单一句话总结

Rviz 显示的机器人位置是 $\text{map} + \text{odom}$ (TF 链的结果)。

严格等式：

 Copy code

机器人全局位姿 = $T_{\text{map_odom}} \times T_{\text{odom_base_link}}$

如果你需要：

- 我可以给你画 LIO-SAM 的完整 TF 树
- 给你画一张 Rviz 坐标查看的流程图
- 也能跑一个例子对比 $\text{map}/\text{odom}/\text{base_link}$ 三者的数值关系

你想要哪一种？

     ...

odom和baselink关系是怎样的

下面我把 **odom** 和 **base_link** 的关系 在 **LIO-SAM** 中的**真实含义** 讲清楚。

很多人第一次接触 LIO-SAM、VINS 或 SLAM 时，最容易误解的就是这个 TF。

结论先说：

****odom \rightarrow base_link 是前端里程计 (IMU + 激光) 实时估计的机器人姿态。**

它是连续、平滑、高频、不做回环、不全局一致。 **

你可以把它看成 **机器人的实时里程计轨迹**。

1. base_link 是什么？

base_link 是机器人机体坐标系（body frame），例如：

- 机器人中心位置
- 激光雷达安装点
- IMU 安装点

通常它是所有传感器外参的基准。

2. odom 是什么？

odom 是“局部、连续、不跳变”的坐标系。

特点：

- **0 点通常是程序启动时候的位置**
- 连续、不跳变（No discontinuity）
- 随着时间会漂移（no loop closure）
- 前端实时估计的结果（IMU+激光）

odom 的目标是：

****提供一个稳定、连续的短期参考坐标系**

让机器人状态估计不会被回环干扰。 **

3. odom → base_link 如何生成？

在 LIO-SAM 中，它来自三部分：

1. IMU 预积分预测（高频 200–500 Hz）
2. 激光帧间配准（10–20 Hz）
3. 融合校正（局部优化）

对应代码在：

```
imuPreintegration.cpp  
imageProjection.cpp  
mapOptmization.cpp (部分)
```

前端的流程：

```
diff
```

[Copy code](#)

```
上一时刻的 odom pose  
+ IMU 预积分增量  
= 预测位姿 (odom)
```

然后再通过激光匹配修正

最终发布：

```
nginx
```

[Copy code](#)

```
odom → base_link
```

这条 TF 是 实时运动轨迹。

★最重要的一点：

****odom → base_link 是实时、高频、连续的，**

不会因为回环优化而跳变。**

🧠 4. 那 odom 和 map 的关系呢？

map 是全局坐标系（回环时会跳变）

odom 是局部连续坐标系（不跳变）

二者通过：

```
arduino
```

[Copy code](#)

map \rightarrow odom

动态对齐。

所以：

base_link 的全局真实位置 =

arduino

 Copy code

map \rightarrow odom \times odom \rightarrow base_link

用图理解：

arduino

 Copy code

map (全局、不连续)

↑ map \rightarrow odom (吸收回环跳变)

odom (局部、连续、高频)

↑ odom \rightarrow base_link (实时前端状态)

base_link (机器人本体)

5. 举个具体例子

假设机器人正在走直线：

odom \rightarrow base_link：

ini

 Copy code

t = 1s: (1, 0, 0)

t = 2s: (2, 0, 0)

t = 3s: (3, 0, 0)

回环触发后后端发现轨迹偏右一点，map 会跳变：

map 下的关键帧从：

SCSS

 Copy code $(3, 0) \rightarrow (2.7, 0.1)$

但 odom 完全不变：

SCSS

 Copy code

依然是 $(3, 0)$

于是 $\text{map} \rightarrow \text{odom}$ 被调整为：

java

 Copy code
$$(2.7, 0.1) \times \text{inv}(3, 0)$$
$$= (-0.3, 0.1)$$

最终 Rviz 中看到：

ini

 Copy code
$$\text{base_link_global} = \text{map} \rightarrow \text{odom} \times \text{odom} \rightarrow \text{base_link}$$

不会跳变。

最终总结（最重要的三句话）

- ✓ $\text{odom} \rightarrow \text{base_link}$ = 前端 IMU + 激光里程计（实时输出）
- ✓ 不会受回环影响（连续、稳定、局部坐标系）
- ✓ $\text{map} \rightarrow \text{odom}$ 动态调整来吸收回环跳变（后端全局优化）

如果你愿意，我可以继续给你讲：

- odom 的数学模型（IMU 预积分如何提供预测）
- $\text{odom} \rightarrow \text{base_link}$ 与激光配准的融合方式
- odom 和 base_link 的 TF 如何从 GTSAM 状态中取出

你想继续深入哪一部分？



map->baselink

你想问的 **map** → **base_link** 是机器人在 RViz 中看到的最终全局姿态。

下面我把它讲到清晰透彻，让你知道：

- **map** → **base_link** 是怎么来的
- 它由哪些 TF 相乘得到
- **map** 和 **odom** 如何协作
- 回环发生时 **map** → **base_link** 如何变化

结论先说：

map → **base_link** = **map** → **odom** × **odom** → **base_link**

这是 RViz 看到的全局机器人姿态。

1. **odom** → **base_link**（前端实时位姿）

来自前端（IMU + 激光），特点：

- 高频（50–500Hz）
- 连续不跳变
- 实时跟踪机器人运动
- 有漂移，但不会中断

也就是说：

odom → **base_link** 表示机器人当前时刻的实时位置。

2. **map**（后端全局坐标）

map 坐标系来自后端因子图（GTSAM）：

- 关键帧插入 → 局部优化

- 回环发现 → 添加约束
- 回环优化完成 → map 帧跳变（可能突变）
- 更新到 transformAftMapped

map 是全局一致坐标，但：

- 更新频率低
- 不连续，会因为回环跳变

3. map → odom（桥梁）

TransformFusion 节点根据：

- 后端优化后的 map 下位姿（transformAftMapped）
- 前端 odom 下 IMU/激光里程计（transformBefMapped）

计算出 map → odom：

```
ini
```

 Copy code

```
T_map_odom = T_map_base_after_opt × inv(T_odom_base_real_time)
```

并以 **高频（30–50Hz）** 发布 TF。

map → odom 用来：

- 吸收回环导致的跳变
- 对齐前端 (odom) 与后端 (map)

4. map → base_link 是这样得到的

最终在 RViz 中：

```
arduino
```

 Copy code

```
map → base_link = map → odom × odom → base_link
```

带入数值举例：

假设：

arduino

 Copy code

```
map→odom = (-0.3, 0.1)
odom→base_link = (3.0, 0)
```

那么：

arduino

 Copy code

```
map→base_link = (-0.3, 0.1) × (3.0, 0)
                = (2.7, 0.1)
```

这就是后端认为更准确的机器人全局位置。

5. 回环发生时，map → base_link 如何变化？

关键点：odom → base_link 不变（保证实时性）

回环优化后：

- map 下的关键帧位置跳变
- 为了保持前端不跳，系统调整 map → odom
- 因此 map → base_link 会产生一个平滑修正（不是瞬间跳）

RViz 中看到的：

- 机器人不会闪跳（因为 map → odom 修正了跳变）
- 轨迹变得更正确（map → base_link 更新了）

最终总结（最重要的）

1. 真实实时位置：odom → base_link

（IMU+雷达前端，连续、高频、不跳变）

2. 全局校正位置：map → base_link

（map 后端优化 + map → odom 修正）

3. Rviz 显示：map → base_link = map → odom × odom → base_link

4. 回环优化只影响 $\text{map} \rightarrow \text{odom}$ ，不会影响 $\text{odom} \rightarrow \text{base_link}$ （前端平滑）

如果你愿意，我还可以继续讲：

- $\text{map} \rightarrow \text{odom}$ 的计算公式推导