**订阅** IMU、LIDAR 原始数据  IMU 里程计
**发布**点云信息

时间戳对应  找到 IMU 三个角度
对 IMU 角度积分  自己定义队列长度

**激光雷达数据类型定义**
XYZIRT  该公司的点云格式
Veledune 和
ouster

```
/
Veledyne激光雷达的数据类型
+ 这里使用16byts对齐，是因为SIMD(Single Instruction Multiple Data)架构需要使用128bit对齐，跟现代CPU架构和指令集有关
+ 在struct定义union结构体但不赋予union结构体名字，使用了`匿名union`用法，union的成员被认为定义在域中：
https://stackoverflow.com/questions/13624760/how-to-use-c-union-nested-in-struct-with-no-name
+ pcl注册自定义点云类型：https://pcl.readthedocs.io/projects/tutorials/en/latest/adding_custom_ptype.html
*/
struct VelodynePointXYZIRT
{
    PCL_ADD_POINT4D
    PCL_ADD_INTENSITY;
    uint16_t ring;
    float time;
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
} EIGEN_ALIGN16;
POINT_CLOUD_REGISTER_POINT_STRUCT (VelodynePointXYZIRT,
    (float, x, x) (float, y, y) (float, z, z) (float, intensity, intensity)
    (uint16_t, ring, ring) (float, time, time)
)
```

**IGEN_ALIGN16  改成 32 线  ring?**

using PointXYZIRT = VelodynePointXYZIRT;
移植雷神格式

双端队列

rclcpp::Subscription<sensor_msgs::msg::PointCloud2>::SharedPtr subLaserCloud;

Imu 对点云位置初始化
自定义 IMU 数据队列长度

没有使用 IMU 的平移增量去畸变
并行执行

std::bind 返回一个基于 f 的函数对象，其参数被绑定到 args 上。
f 的参数要么被绑定到值，要么被绑定到 placeholders（占位符，如_1, _2, ..., _n）

double callableFunc (double x, double y) {return x/y;}

```
auto NewCallable = std::bind (callableFunc, std::placeholders::_1,2);
std::cout << NewCallable (10) << '\n';
```

- 第一个参数被占位符占用，表示这个参数以调用时传入的参数为准，在这里调用 NewCallable 时，给它传入了 10，其实就想到于调用 callableFunc(10,2);

意义： imuHandler imuOdomHandler cloudHandler?

1. **std::mutex**:

1. std::mutex 是一种独占式互斥量，用于保护共享数据，确保在同一时间只有一个线程可以访问它。
2. 它不支持递归锁定，即同一线程不能多次锁定同一个 std::mutex。
3. 不带超时功能。

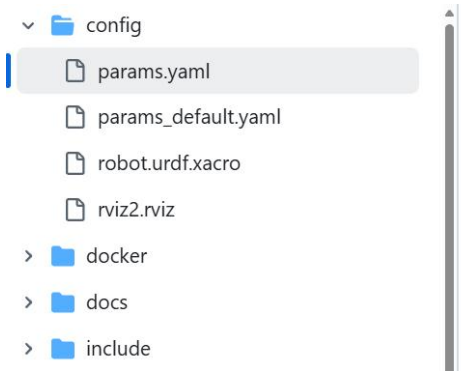
**IMU 和 LIDAR 对齐调试**

```
cachePointCloud
```

点云格式转换

左值、右值引用

- **左值（lvalue）**：表达式结束后依然存在的持久对象。&
- **右值（rvalue）**：表达式结束后就不再存在的临时对象。 &&
- ans=std::move(result)：右值引用，移动构造
- 完美转发函数 **std:forward<T>** 。它可以在模板函数内给另一个函数传递参数时，将参数类型保持原本状态传入（如果形参推导出是右值引用则作为右值传入，如果是左值引用则作为左值传入）




```
21
22      # GPS Settings
23      useImuHeadingInitialization: false          # i
24      useGpsElevation: false                      # i
25      gpsCovThreshold: 2.0                        # m
26      poseCovThreshold: 25.0                      # m
27
28      # Export settings
29      savePCD: false                              # 是
30      savePCDDirectory: "/Downloads/LOAM/"        # i
31
32      # Sensor Settings
33      sensor: velodyne                            # l
```

- 改 sensor

```
/**
 * 2、从IMU数据和IMU里程计数据中提取去畸变信息
 * imu数据：
 *   1) 遍历当前激光帧起止时刻之间的imu数据，初始时刻对应imu的姿态角RPY设为当前帧的初始姿态角
 *   2) 用角速度、时间积分，计算每一时刻相对于初始时刻的旋转量，初始时刻旋转设为0
 * imu里程计数据：
 *   1) 遍历当前激光帧起止时刻之间的imu里程计数据，初始时刻对应imu里程计设为当前帧的初始位姿
 *   2) 用起始、终止时刻对应imu里程计，计算相对位姿变换，保存平移增量
 **/
```

```
// 从IMU队列中计算去畸变信息
imuDeskewInfo();

// 从IMU里程计中计算去畸变信息
odomDeskewInfo();
```

去畸变

```
// 这个while循环可以理解为做IMU数据和点云数据时间戳对齐，不过是一种十分简化的做法。
// 在各个数据源没有做硬件触发对齐的情况下，这不免是一种很好的做法
while (!imuQueue.empty())
{
    if (stamp2Sec(imuQueue.front().header.stamp) < timeScanCur - 0.01)
        imuQueue.pop_front();
    else
        break;
}
if (imuQueue.empty())
    return;
```

**能否成功对齐？**

对不齐的话右边改小

```
// IMU 数据处理成功， 这个标志位标志在图优化中可以使用 IMU 的角度输出作为该帧点云的初始估计位置
        // 注意，只是标志后续节点可以使用该 IMU 初始信息，不一定会被使用
        cloudInfo.imu_available = true;
```

**IMU 数据不一定会用**

```
imuPointerCur
```

imu 帧数

角速度预积分 // 再次强调，对角速度的积分不是简单的角速度乘以间隔时间

// 关于角速度的积分公式可以查阅：https://zhuanlan.zhihu.com/p/591613108

```
void imuOdomHandler(const nav_msgs::msg::Odometry::SharedPtr odometryMsg)
    {
        std::lock_guard<std::mutex> lock2(odoLock);
        imuOdomQueue.push_back(*odometryMsg);
    }
```

**IMU 里程计来源**

```
IMU 里程计话题的回调函数，来自 imuPreintegration 发布的 IMU 里程计
    /// @param odometryMsg
//被 subImuOdom 调用
```

```
// 是否使用IMU里程计数据对点云去畸变的标志位
// 前面找到了点云起始时刻的IMU里程计，只有在同时找到结束时刻的IMU里
// 利用时间插值算出每一个点对应时刻的位姿做去畸变
// 注意：！在官方代码中，最后并没有使用IMU里程计数据做去畸变。所以这
// 下面的这些代码实际上也没有被使用到
// 为什么没有使用IMU里程计做去畸变处理？
// - 原代码中的注释写的是速度较低的情况下不需要做平移去畸变
```

**考虑是否自己添加用里程计数据去平移畸变**

round()四舍五入

```
/**
 * @brief 根据某一个点的时间戳从IMU去畸变信息列表中找到对应的旋转角
 * @param pointTime 点云某一个点的时间戳，秒
 * @param rotXCur 输出的roll角
 * @param rotYCur 输出的pitch角
 * @param rotZCur 输出的yaw角
*/
void findRotation(double pointTime, float *rotXCur, float *rotYCur, float *rot
{
```

**求去畸变旋转角**

线性插值

雷达单点运动矫正

@param point 点云中某个点位置

* @param relTime 该点相对于该帧点云起始时刻的相对时间

Deskew ： 去畸变

**通过 IMU 线性插值形成紧组合**

Horizon_SCAN 设置 ： 角度视野

FLT_MAX:浮点数最大值

Rangemat: 行代表通道数，列代表角度视野

- **Horizon_SCAN**：水平方向的总扫描线数（如 Velodyne HDL-32E 为 1800 线）。
- **ang_res_x**：水平角分辨率（每线对应的角度），例如：
- `Horizon_SCAN = 1800` → `ang_res_x = 0.2°`