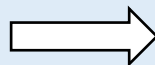


# 三维激光SLAM原理及开源方案对比



浙江省北大信息技术高等研究院  
智能计算中心  
机器人导航算法工程师：赵锴  
zkyy828@163.com

阿木实验室微信公众号后台回复“SLAM”获取直播讲义

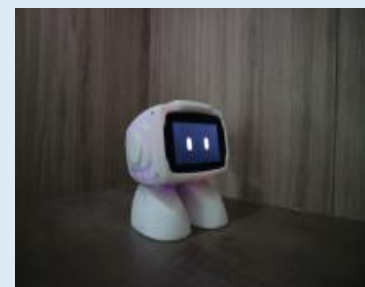


# 介绍



实验室公众号

**情感智能机器人实验室 (LAIR)** 主任为北京大学信息科学技术学院王韬老师。  
北京大学校本部 (北京) 实验室 (**PKU-LAIR**):  
注重机器人对人类情感识别、注意力分析等学术研究。  
北京大学信息技术高等研究院 (杭州) 实验室 (**AIIT-LAIR**):  
注重科技研发及智能机器人设计与制造。



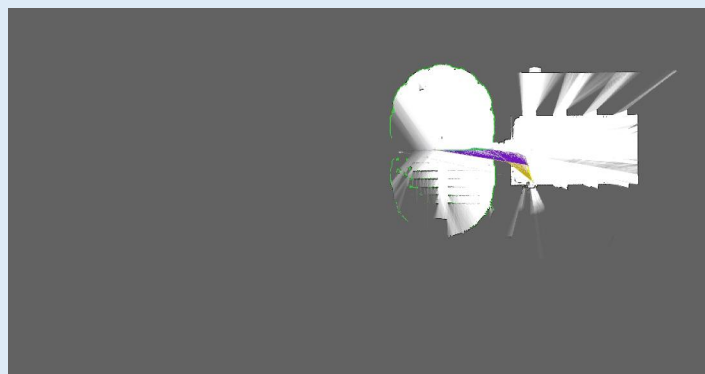
- 赵锴，现就职于北京大学信息技术高等研究院-情感智能机器人实验室 (杭州, AIIT-LAIR)
- 担任**机器人导航工程师**，方向为激光SLAM与深度强化学习。
- 联系邮箱: zkyy828@163.com

# 介绍

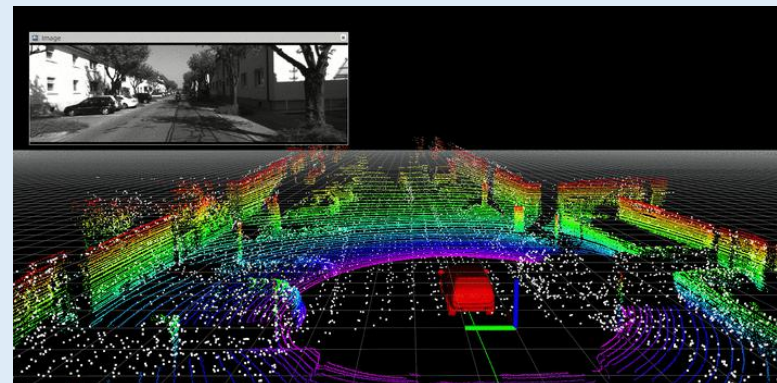
- 2D VS 3D



单线激光雷达，  
可以看到空间的一个截面。  
构建二维栅格地图，适用于室内机器人



多线激光雷达，  
可以看到空间的多个截面。  
构建三维点云地图，适用于室外机器人

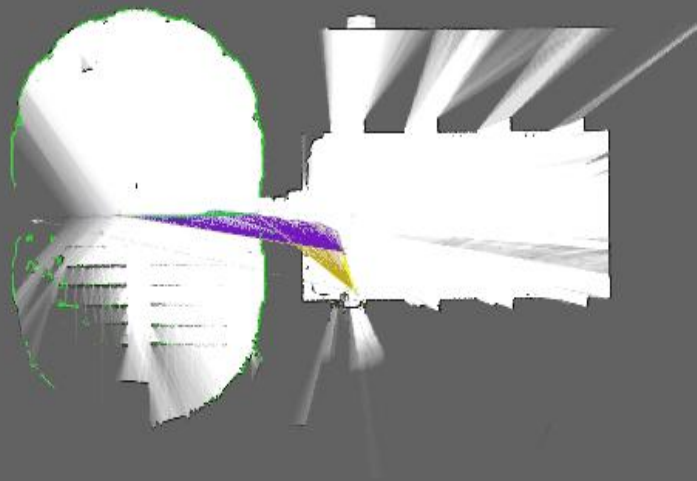


# 目录

- 从2D到3D——Cartographer (ICRA2016)
- LOAM (RSS 2014) ——扩展: A-LOAM, F-LOAM
- Lego-LOAM (IROS 2018) ——扩展: SC-Lego-Loam
- LIO-SAM (IROS 2020)
- LVI-SAM (ICRA 2021)
- 固态激光雷达livox-LOAM (IROS 2019)
- LIO-Mapping (ICRA 2019)
- LINS (ICRA 2020)
- Fast-LIO (RA-L 2021)



# 从2D到3D——Cartographer



本节目录：

介绍

2D-3D建图比较

开源代码特点

地图设计

匹配方法

初始位姿

第一阶段解算

第二阶段解算

后端

如何检测回环

检测回环之后应该怎么做

# 从2D到3D——Cartographer

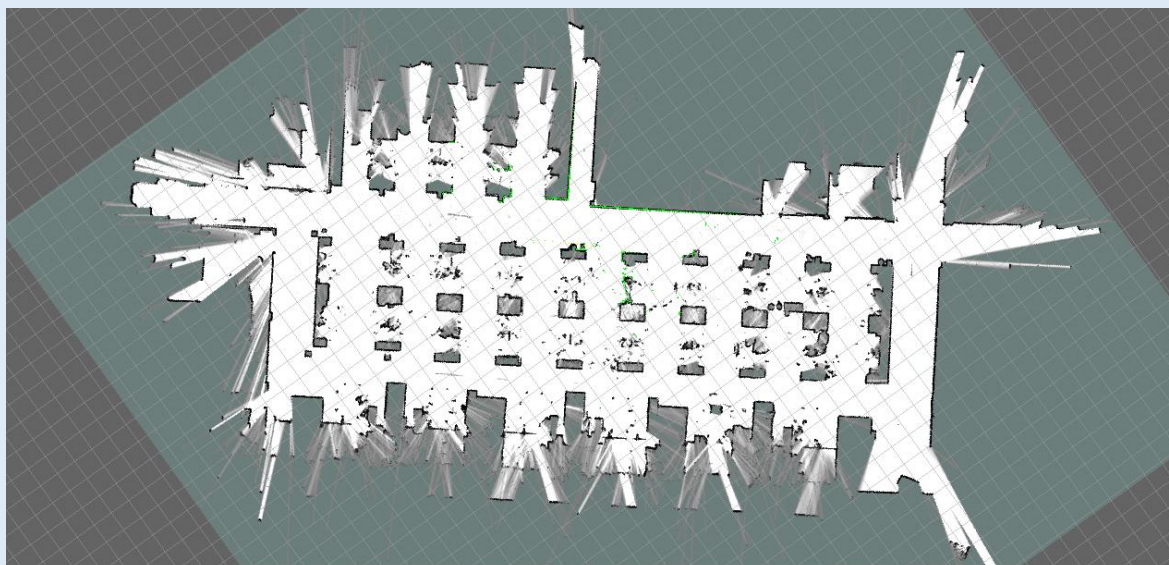
- Cartographer是由谷歌于2016年开源的一个支持ROS的室内SLAM库，并在截至目前为止，仍然处于不断的更新维护之中。
- 特点：**代码极为工程**，多态、继承、层层封装的十分完善。提供了方便的接口，便于接入IMU、（单/多线）雷达、里程计、甚至为二维码辅助等视觉识别方式也预留了接口（Landmark）。
- 根据其代码的结构设计来看，显然是由一个大型的研发团队集中研发，而非少数科研人员的demo型代码。
- “明显不是搞科研的玩儿法，就是奔着产品去的。算法需要的计算资源少，而且因为依赖很少，因此**几乎可以直接应用在一个产品级的嵌入式系统上**。以前学术界出来的开源2D/3D SLAM算法不少，但能**几乎直接拿来就用在产品上的，恕我孤陋寡闻还真想不出来**。因此，我认为进入相关领域SLAM算法的门槛被显著降低了。” ——梅卡曼德机器人创始人 邵天兰



# 从2D到3D——Cartographer

## 2D-3D建图比较

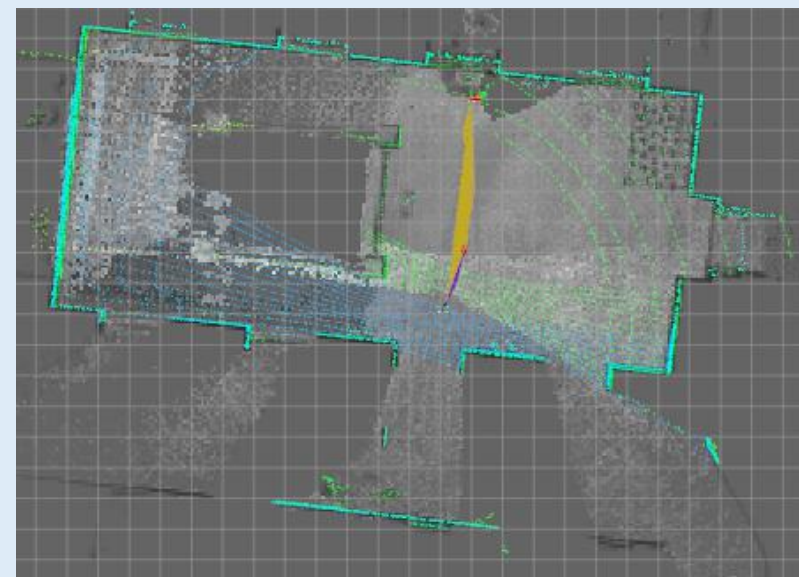
- Cartographer支持2D和3D激光雷达的输入，**实现机器人定位，并构建栅格地图。**



2D-SLAM：基于2D栅格地图，可以直接用于导航。

使用方法：

1. 直接使用Ros的Move\_base等方式。
2. 导航代码中订阅/map消息，（data部分为一维数组，根据height和width可恢复地图图像。-1为未知，0~100依次增加表示被占据概率增大）



3D-SLAM：基于hybridGrid，译为混合概率地图，可以理解为3D栅格地图。

**明确：RViz 仅显示3D混合概率网络的2D 投影(以灰度形式)**

该地图难以直接使用。

# 从2D到3D——Cartographer

- Cartographer的理论部分，被发表在《Real-Time Loop Closure in 2D LIDAR SLAM》中 (ICRA 2016)，其中仅包含2D部分。**3D部分目前未见任何发表。**
- 其主要用到的库：
  - 1.线性代数库Eigen3 2.用来读取配置文件中的参数库Lua, 3.谷歌开源的非线性最小二乘优化库Ceres, 4.Google开源的很流行的跨平台通信库Protobuf
- 分为Cartographer和Cartographer\_ros, 后者为前者基于ros的接口，换言之，**Ros在其中的作用仅为面向用户，算法本身不以ros传输数据**，而是以Protobuf进行各个进程之间的数据编码，再进行通信。



# 从2D到3D——Cartographer

- Cartographer的地图（map）以子地图（submap）的形式组成。
- 分为前端和后端。
  - 前端：根据帧间匹配算法（scan-match），实时根据激光(scan)来推测累积的scan相对于submap的位姿。
  - 后端：检测回环（发现在已到达的位置附近），修正各个submap之间的位姿。
- 根据代码可以判断，2D和3D基于的是同一套思路，但是在实现上有一定区别。
- 接下来结合2D和3D部分，**对比介绍**实现定位和建图的方法。

# 从2D到3D——Cartographer

- 在介绍定位和建图思路之前，先介绍一下地图的更新方式。
- 概率栅格地图：

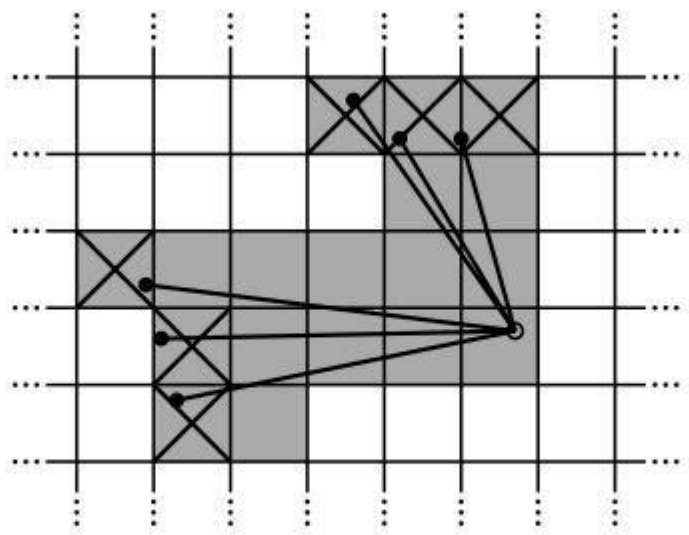


Fig. 2. A scan and pixels associated with *hits* (shaded and crossed out) and *misses* (shaded only).

以方格代表地图块，内部存储数据用来表示被占据的概率。

发出一束激光，打到一个障碍物点，被打到的称为hit点，中间连线上的空区域，称为miss点。

2d和3d都是存储的这样的地图。3d相当于是3维的栅格地图。

宏观上：多次观测到，提升其概率。

**问题是，如何用公式表达这个“多次观测”来实现“概率提升”？**

# 从2D到3D——Cartographer

地图设计

$$\text{odds}(p) = \frac{p}{1-p}, \quad (2)$$

$$M_{\text{new}}(x) = \text{clamp}(\text{odds}^{-1}(\text{odds}(M_{\text{old}}(x)) \cdot \text{odds}(p_{\text{hit}}))) \quad (3)$$

- $p$ 表示占据概率，当 $p=0.5$ 时，概率比值 $\text{odds}=1$ ，表示占据和空闲各占一半。 $\text{odds}^{-1}$ 表示函数逆运算。
- $p_{\text{hit}}=0.55$ 代表该位置被激光打到一次的概率，第一次观测会被直接赋值。
- $M(x)$ 表示地图中 $x$ 位置处的概率值。

举例：

- 初始时刻，栅格未知状态，激光第一次打到了位置 $x$ 处， $M(x)$ 概率更新为0.55。
- 随后，激光第二次重复打到了同一个位置：

$\text{odds}(p_{\text{hit}})=0.55/(1-0.55)=1.22$ ,  $\text{odds}(M_{\text{old}}(x))=\text{odds}(0.55)=1.22$

$\text{odds}(p_{\text{hit}})$ 和 $\text{odds}(M_{\text{old}}(x))$ 相乘为1.484，再求函数逆运算，恢复出更新后的 $M_{\text{new}}(x)=0.597$ 。

实际代码中，采用了多种工程技巧加速运算。包括：映射到整数范围，预计算，查表等方法，此处不深入展开了。

# 从2D到3D——Cartographer

- 介绍完地图的形式，回到2D和3D的前端：
- Cartographer采用的是**scan-map**的匹配方法。
- scan-scan：这个意味着利用两帧激光数据（每帧激光束的数目相同），计算二者之间的变换。典型方法：ICP。
- scan-map：利用一帧激光数据和地图数据，找到激光数据在地图中的位置。
- map-map：利用一个子地图数据，在一个更大的地图中找到它合适的位置。

# 从2D到3D——Cartographer

- Cartographer在scan-map的匹配上，采用分阶段的方法，组合各种方式，计算scan在map中的位姿变换。
- 不管是2D还是3D，首先要有一个初始的位姿，在此基础上进行优化：
  - 有IMU，则采纳其角速度积分作为初始姿态。**不信任IMU任何加速度信息。**
  - 有里程计，则采纳里程计的线速度积分作为初始平移。
  - 二者都没有，根据之前的运动做一个匀速的假设。
- 可以看出，**cartographer的多传感器融合是一个松耦合**，主要依赖激光来定位。IMU和里程计数据并没有被构建到真正优化的目标函数中。



# 从2D到3D——Cartographer

## 第一阶段解算

- 在得到了初始位姿以后，初始位姿要经过**第一阶段解算**：CSM（Correlation Scan Match 相关扫描匹配）——构建似然场
- 即对原先的地图map进行一个高斯模糊，让它膨胀一些，然后把激光scan在一个**搜索窗口**内**暴力匹配**，计算得分。

**两个问题**：1.得分怎么算？ 2.地图不是无限大的吗，你怎么保证就在搜索窗口里就能找到位姿呢？

- 1.如果scan的点落在障碍物模糊区域内，落的越多，得分越高。
- 2.**因为有初始位姿**。误差肯定在一个范围内而不会马上发散到很远，**所以可以在一个位姿的窗口内，对位姿进行暴力匹配搜索**。（初始位姿估计中，里程计数据不会突然激增；imu的加速度信息会漂移，但是算法对于imu加速度数据选择直接丢弃不看；而根据之前**位姿匀速假设**也不会飘走）

# 从2D到3D——Cartographer

- 什么是位姿？ 位置+姿态。
- 对于2Dslam而言，有三个变量， $x$ ， $y$ ， $yaw$ 角。
- 对于3Dslam而言，有 $x$ ， $y$ ， $z$ ， $roll$ ， $pitch$ ， $yaw$ 六个变量。
- 2dslam中，采用三层循环，（最外层为 $\theta$ ，减小 $\sin$ 和 $\cos$ 的频繁计算），对 $x$ ， $y$ ， $\theta$ 在给定大小的搜索窗口内进行**穷举**，计算最高得分的 $x$ ， $y$ ， $\theta$ 作为一阶段解算的输出位姿。
- 3dslam中，采用六层循环，对 $x$ ， $y$ ， $z$ ， $roll$ ， $pitch$ ， $yaw$ 六个变量在搜索窗口内穷举，计算得分最高的作为一阶段解算输出位姿。
- 很显然，3d-slam的这种方式对于计算资源依赖较大，复杂度达到 $O(n^6)$ 级别。因此3d-slam的CSM方法，作为一个配置选项，默认是不开启的。当然如果用户机器比较牛逼，也可以选择开启。

# 从2D到3D——Cartographer

- 我们可以看出，第一阶段CSM解算中，**位姿在其中是一个离散的变量**，通过暴力枚举获得输出结果；
- 但是暴力枚举也是存在分辨率的，例如：如果角度步长设为1度，但如果刚好真正的角度是5.5度，那么CSM只能搜索到5或6度，而无法进一步细化，逐步累积将会造成误差。
- 因此，引入**第二阶段位姿解算：非线性优化**

$$E(T) = \arg \min_T \sum [1 - M(S_i(T))]^2$$

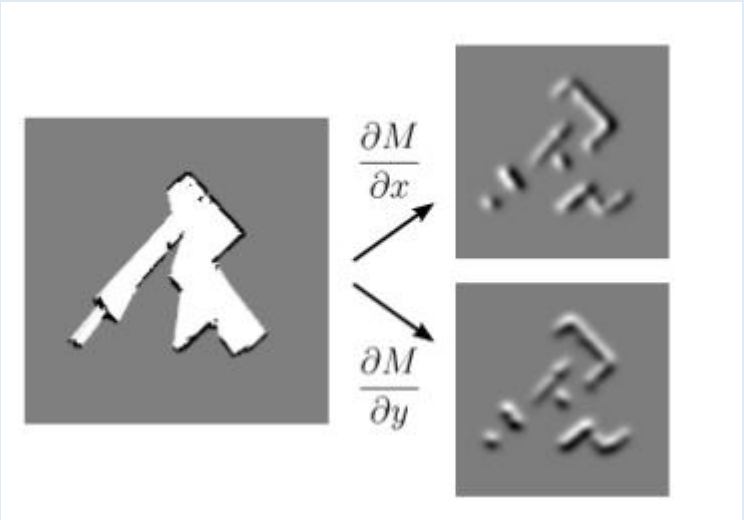
$S_i(T)$ 表示把激光数据 $S$ 用位姿 $T$ 进行转换

$M(x)$ 表示得到坐标 $x$ 的地图占用概率

思路：s代表了激光击中障碍物，将激光点在机器人坐标系下的位置，经过T转换到世界坐标系下以后，应该尽可能的落在已有地图的障碍物上

## 从2D到3D——Cartographer

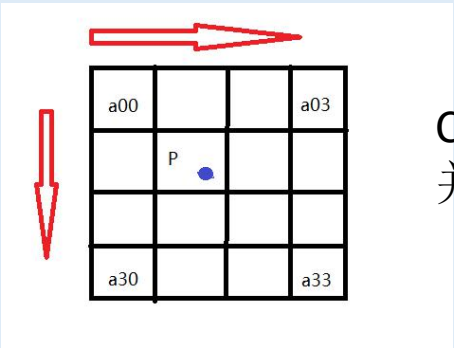
- 第二阶段的位姿求解，显然**位姿在其中是一个连续的变量**，通过梯度下降的方法求解目标函数。
- 地图是离散的，因此需要对地图进行**插值处理**，使地图也变成一个可以求导的连续变量，这样才能优化前述目标函数。



线性插值：已知数据  $(x_0, y_0)$  与  $(x_1, y_1)$ ，要计算  $[x_0, x_1]$  区间内某一位置  $x$  在直线上的  $y$  值；

双线性插值本质上就是在两个方向上做线性插值。

双三次插值：更加复杂的插值方式，它能创造出比双线性插值更平滑的图像边缘。使用最近16个点插值。



Cartographer用的是这一种并且采用Ceres自带的双三插值器

# 从2D到3D——Cartographer

- 根据代码判断2D和3D对于此部分内容基本相同。
- 2D：三个误差项：位姿转换误差+ 旋转惩罚+平移惩罚，后二者限制了旋转和平移的修改不能距离初始位姿太大。
- 3D：四个误差项：低分辨率位姿转换误差+ 高分辨率位姿转换误差+ 旋转惩罚+平移惩罚。低分辨率位姿转换误差权重低于高分辨率。
- 旋转和平移的权重也可以在配置文件中调参。



# 从2D到3D——Cartographer

后端

- Cartographer在后端主要寻找回环，并根据建立的约束对所有的sumap进行统一优化。
- **回环检测目的是：检测当前位置是否曾经来过**，即采用当前scan在历史中搜索，确认是否匹配。
- 为什么要有回环检测：1. 已有地图时位姿初始化，不知道当前帧初始位姿，也就不清楚在地图中哪个位置，无法做定位。 2.有累积误差，仅靠前端递推，不进行修正的话，地图很容易变形。
- 两个问题：**1.如何检测回环。2.检测回环后该怎么做。**


# 从2D到3D——Cartographer

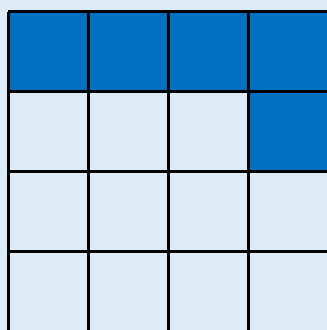
如何检测回环

- **检测回环和前端的思路也比较相似，先通过穷举暴力匹配，再通过优化精细修正。**
- 但是，前端的暴力穷举，是在有个初始位姿的基础上在一个小窗口内穷举。
- 后端重定位，**没有初始位姿了**，暴力匹配的范围变成了整个地图。
- 必须采用算法加速处理。
- 多分辨率地图+分枝定界操作。

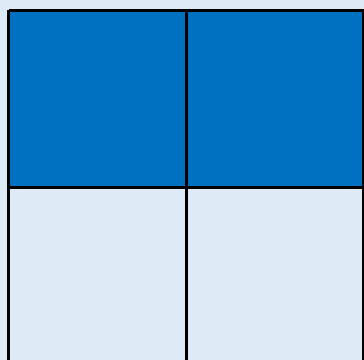
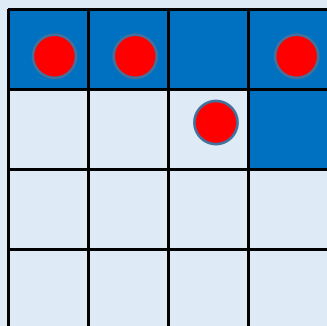
# 从2D到3D——Cartographer

## 如何检测回环

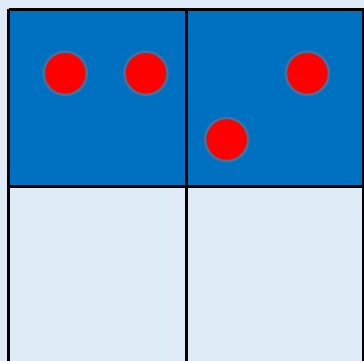
- 假设有一帧激光： 蓝色代表障碍物



高分辨率地图



低分辨率地图



在高分辨率的地图上，四个点命中3个；  
在低分辨率的地图上，四个点全部命中。

激光在低分辨率的地图上匹配情况：  
代表得分的上界（再往精细展开，匹配得分只能更低，不能更高）

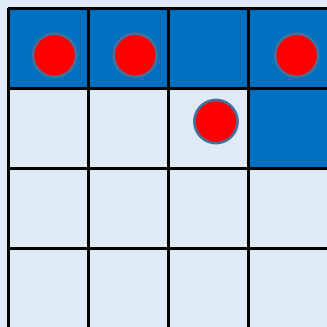
在高分辨率的地图上匹配情况：  
代表得分的下界（再往粗略缩放，匹配得分只能更高，不会更低）

# 从2D到3D——Cartographer

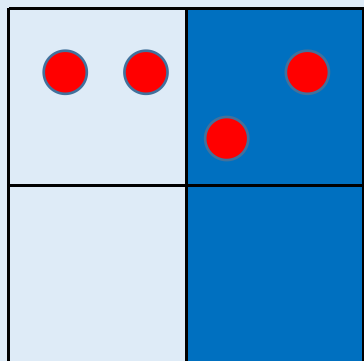
## 如何检测回环

分支定界:

1. 先把整个地图中的一个区域展开到底（最高分辨率），得到一个匹配分数（得分下界）；
2. 然后把其他区域不展开，算匹配分数。（得分上界）
3. 如果低分辨率区域的得分上界，还没有已展开到底的高分辨率区域的下界高，这个低分辨率区域就不再展开了，统统丢掉不要。



子区域A



子区域B

左图四个命中3个，得分75；

右图四个命中2个，得分50；

那么激光打在子区域A的可能性就要大于B，因此B就无需继续展开成更精细的地图了。

# 从2D到3D——Cartographer

如何检测回环

- 2D-slam的思路比较简单

- 前端：小范围内穷举+非线性优化方法修正位姿。
- 回环检测：大范围内穷举（利用分支定界加速） +非线性优化修正位姿。

- 3D-slam有所不同。

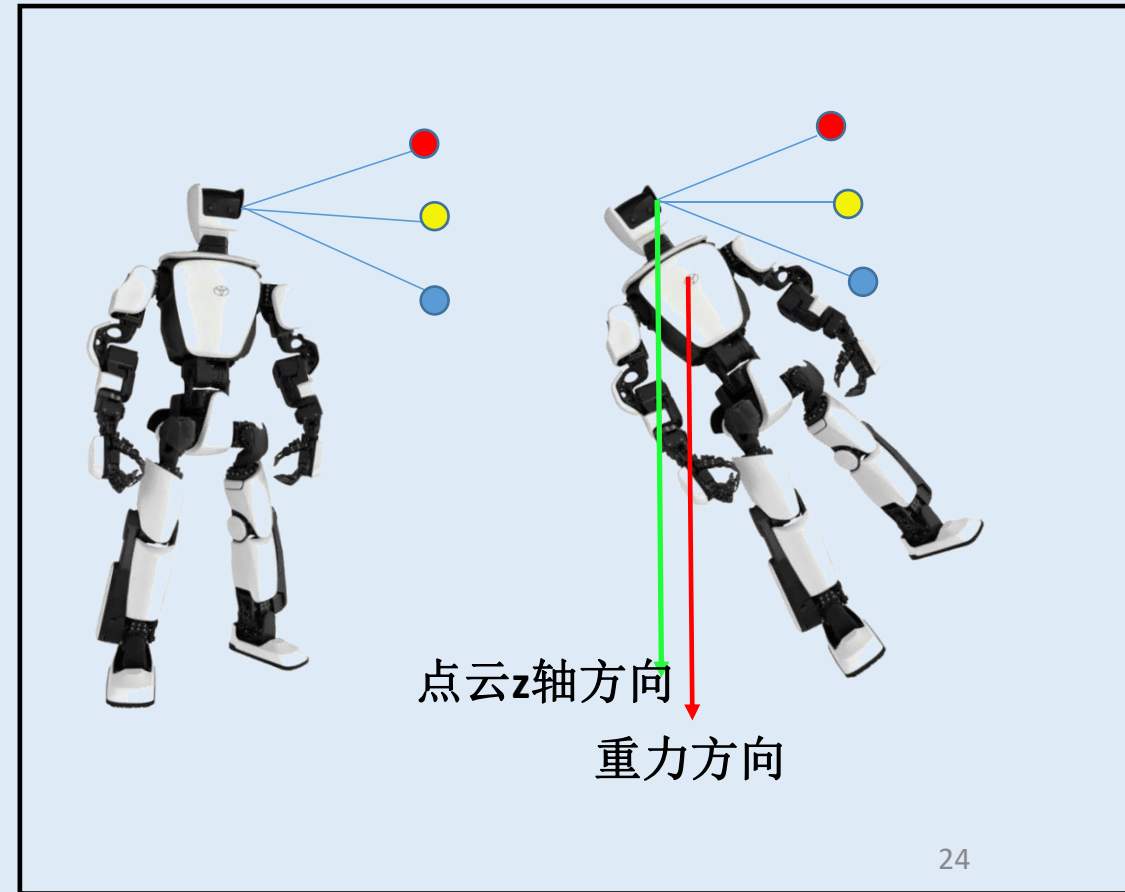
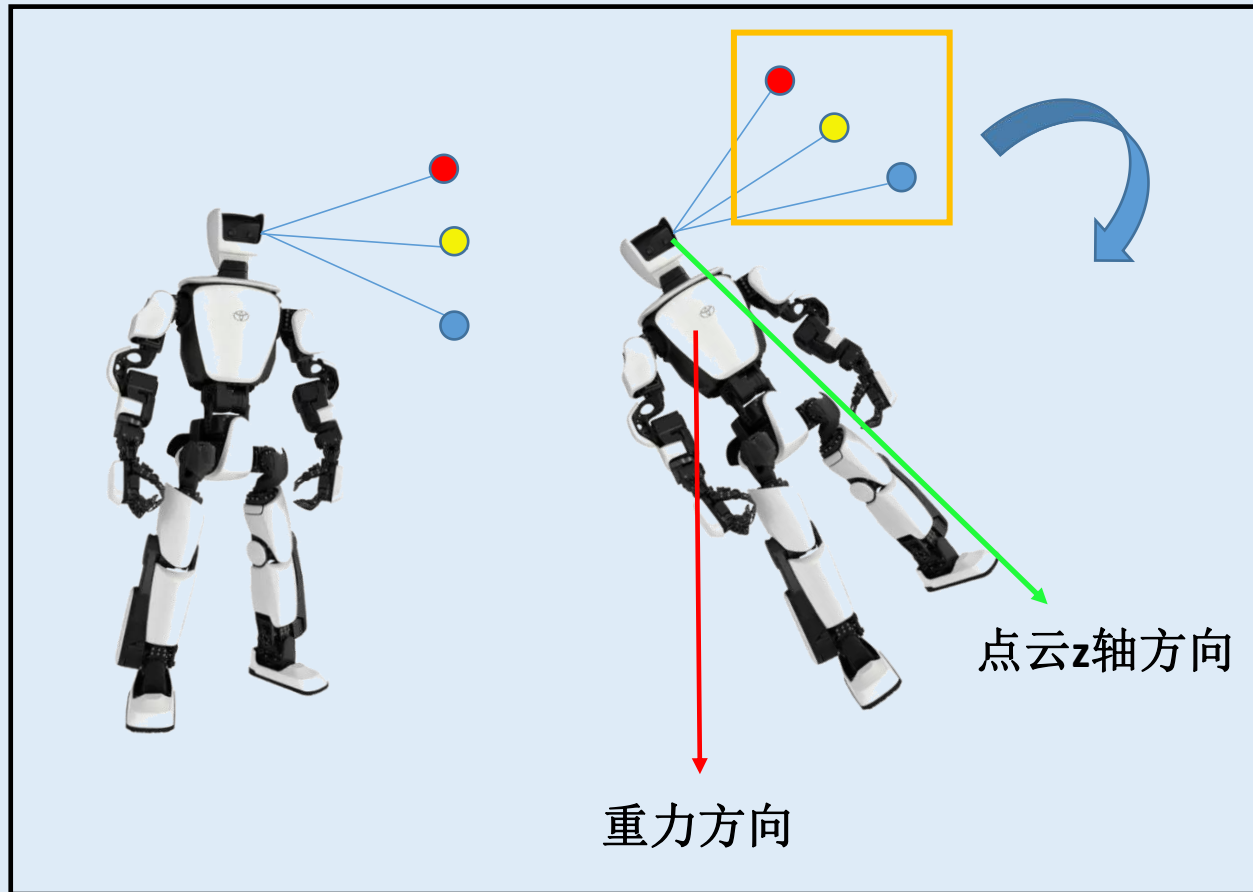
- 前端的暴力匹配方法，是直接6层循环暴力枚举的，因此配置文件中默认不开启，而是在初始通过IMU、里程计等预测位姿基础上，直接非线性优化修正位姿。
- 回环检测：大范围内6个循环穷举+分支定界？小范围都嫌慢，大范围更别提。

直接对位姿非线性优化？ 1.没有初值，会算到猴年马月。 2.会落入局部最优值。



# 从2D到3D——Cartographer

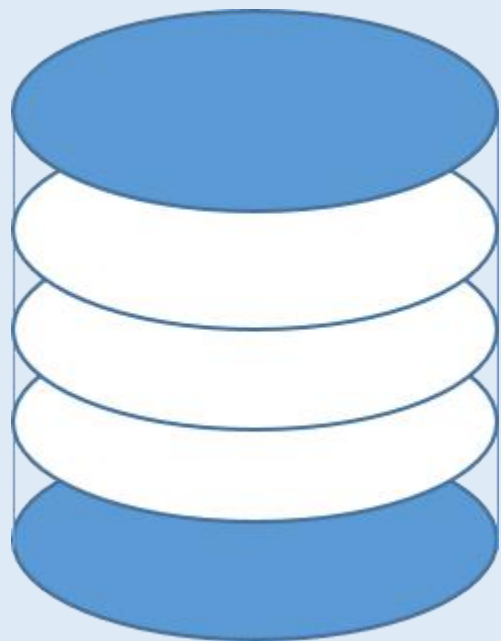
- 首先，Cartographer3D用IMU确定重力方向，**让要匹配的点云的Z轴方向和重力方向对齐**。(六维搜索就变成了四维：少了俯仰角roll，横滚角pitch)



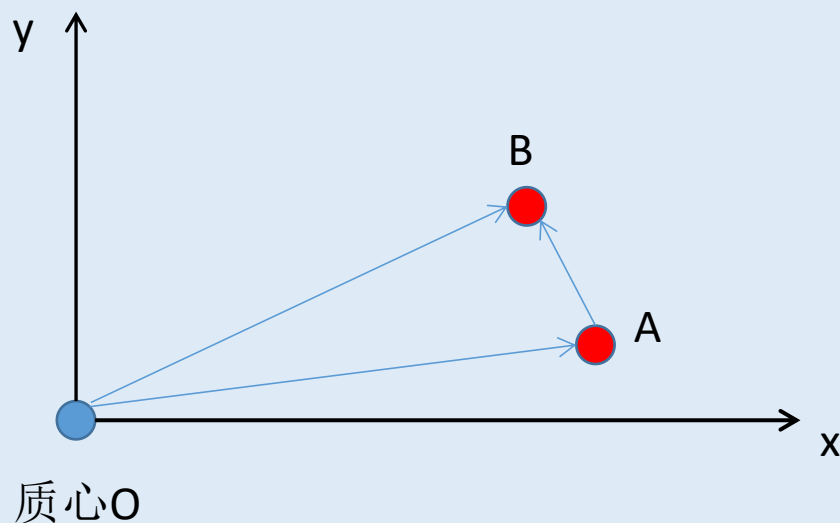
# 从2D到3D——Cartographer

如何检测回环

不再投影点云，而是**投影特征**。  
特征如何获取？



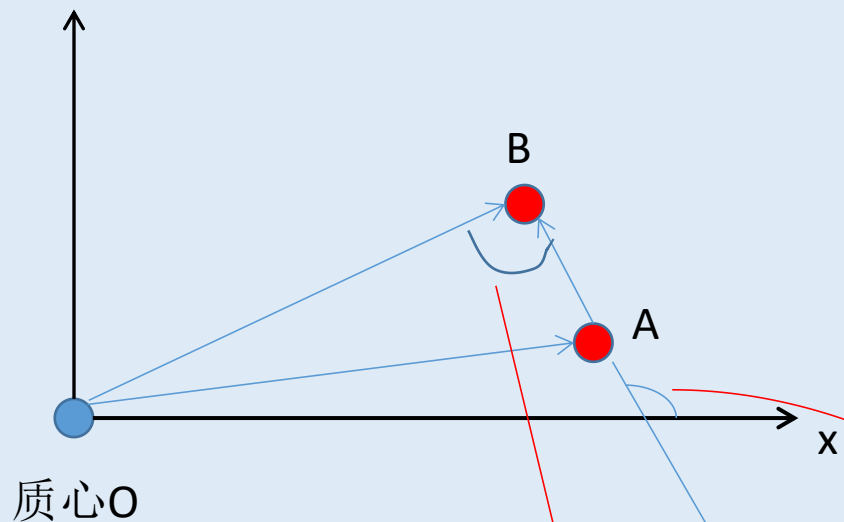
1. 首先，在z轴方向对点云切成n个片；
2. 对每个切片中的点，求解质心；
3. 计算每个点，与质心连线，和x轴所成的角度，并依据角度排序。



OA距离X轴夹角最小，从A点开始，B是第二个，以此类推

# 从2D到3D——Cartographer

如何检测回环



之后：

1. A点作为参考点：计算AB和x轴的夹角，映射到直方图。  
相当于：把点云分类，分成size个类，  
分类标准为**当前点B与参考点A**连线AB和x轴的夹角。

2.再计算一下质心到AB的夹角OBA，作为直方图中这个点对应的权重。

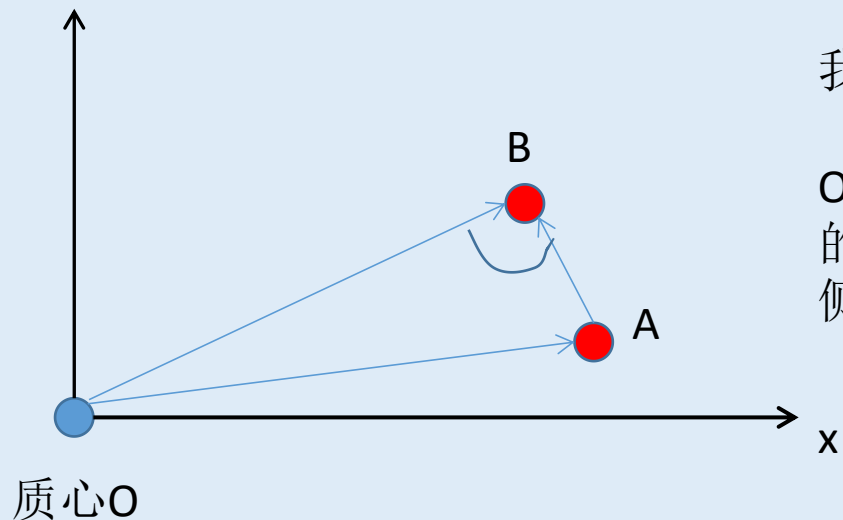
注意，这两个角不是同一个角

B点被映射到某一个分类中，分类依据为AB和X的夹角；（AB代表“参考点与当前点”，当参考点B距离A很远时，参考点则会被重新选择）

B在此分类中的权重，由角OBA的大小决定（变换至-90~90度，主要是想看垂直程度）。

# 从2D到3D——Cartographer

## 如何检测回环



我们可以知道，一圈360度激光，求质心实际就是激光发射器的位置，

OB和AB的夹角越垂直，证明一束激光OB，更接近垂直的打到了AB构成的障碍物平面上，反射强度更高，更可信。越不垂直，说明激光OB以侧面的形式打到平面AB上，探测误差可能更大。

回到原先问题上：

刚刚准备在yaw，x，y，z四个维度对当前点云在地图中暴力搜索一个位置初值；

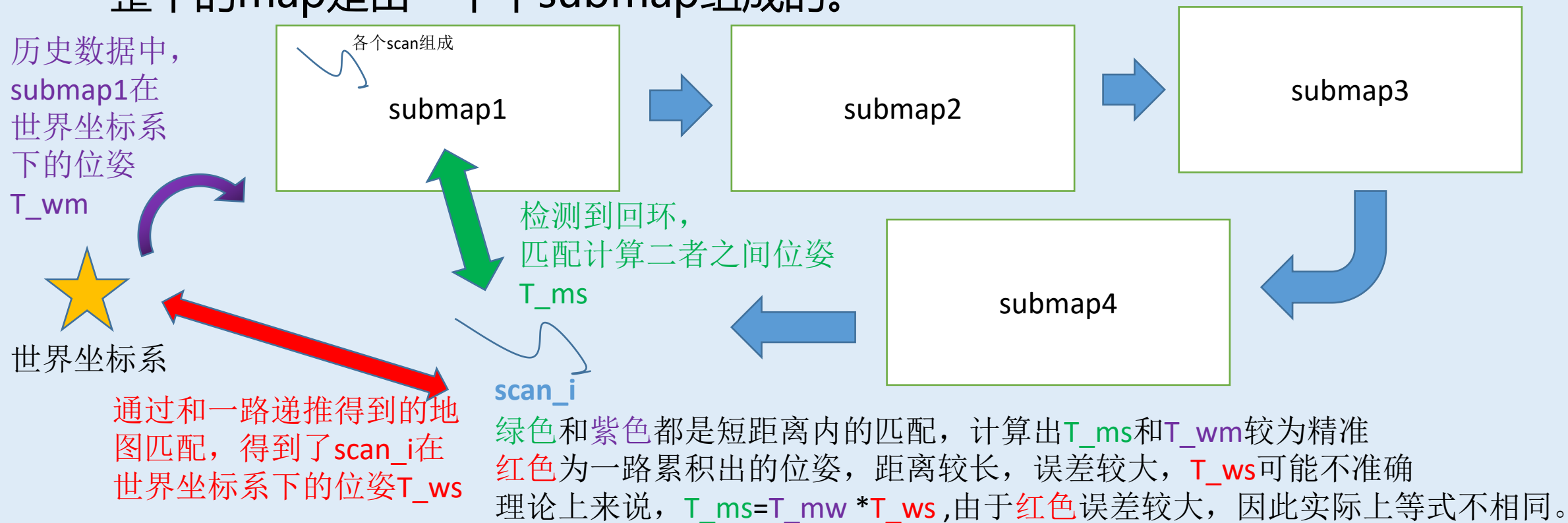
提取直方图的目的，相当于是对当前点云进行一个降维，提取特征。

根据直方图中提取的特征（根据切片上每个点与参考点的直线AB与x轴的夹角分成n个类，类的值是OBA的大小），和历史数据进行匹配，**筛选掉一批不够阈值的yaw角。**

# 从2D到3D——Cartographer

检测到回环后该怎么做：  
优化问题

- 回顾：在Cartographer前端中，不断维护的是scan和submap之间的位姿。整个的map是由一个个submap组成的。



因此后端的本质，就是根据准确的  $T_{ms}$  和  $T_{wm}$  对不准确的  $T_{ws}$  进行修正。



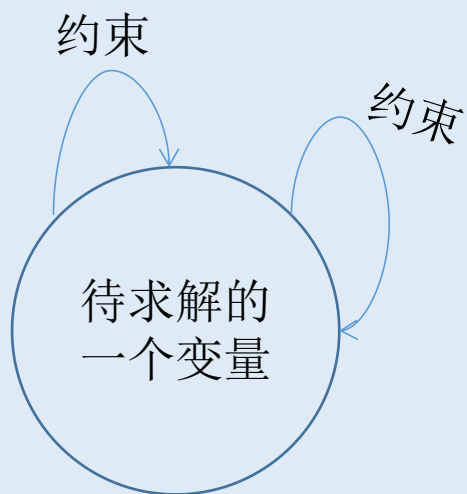
# 从2D到3D——Cartographer

检测回环后该怎么做：  
图优化

## • 图优化

而 $T_{ws}$ 是一路递推下来的，牵一发而动全身，整条轨迹都会被修正。怎么修正？

1. 各个submap的位姿，就构成了一个个圆形的变量
2. 前述 $T_{ms} = T_{mw} * T_{ws}$  构成其中一个约束。

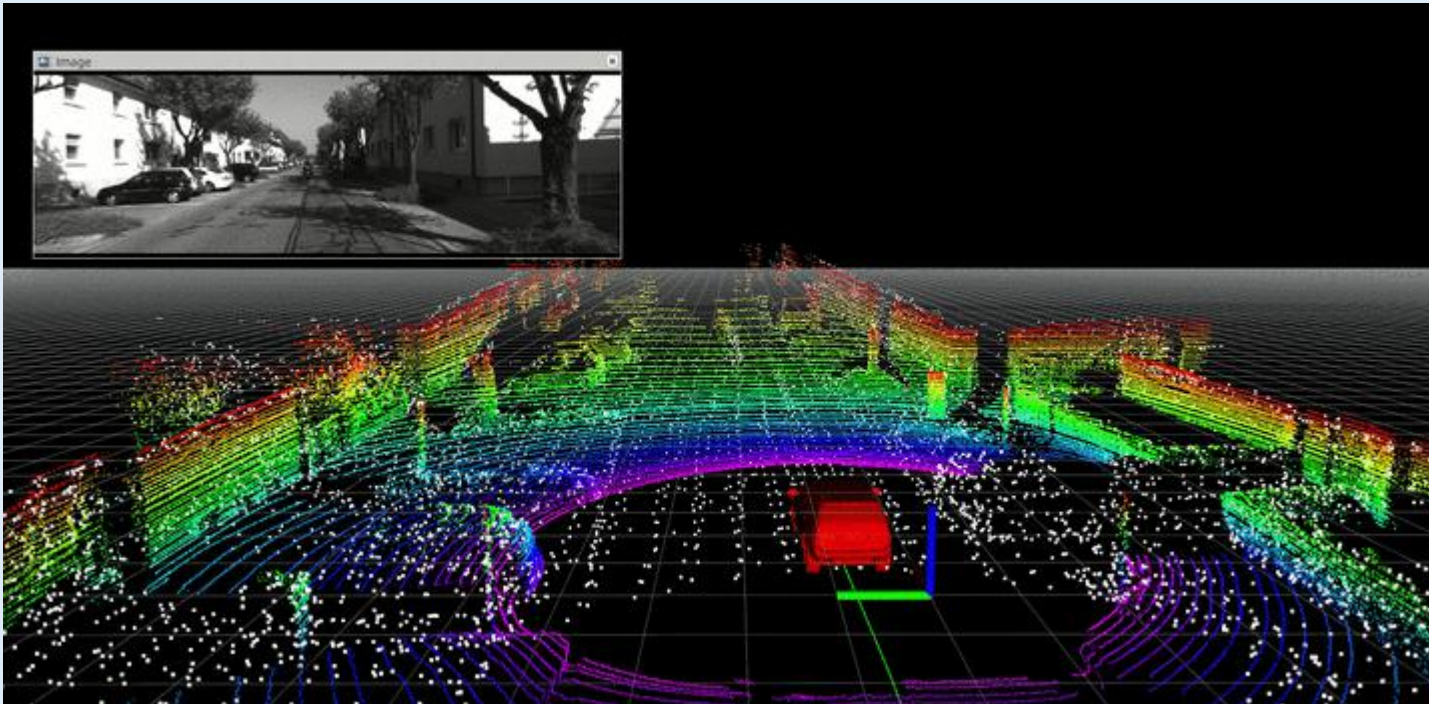


图优化就是调整变量中的值，让它能够尽可能的满足约束。

本质上还是一个求解非线性最小二乘的问题。  
Cartographer使用Ceres库来求解。

参与计算的只有各个位姿数据及其对应的约束，因此2D和3Dslam在求解上区别仅在于：待求解变量维度不同（即位姿 $T$ 的表示不同：2dslam仅有 $x$ ， $y$ 和 $yaw$ ）

# LOAM: Lidar Odometry and Mapping in Real-time



本节目录:

介绍

什么, 是Kitti数据集?

框架

特征点

运动估计

地图构建和全局位姿

优缺点

# LOAM

## 介绍

- LOAM为清华自动化本科毕业的Zhang Ji博士在CMU读博期间，于2014年在RSS期刊发表的关于三维激光传感器的SLAM算法。
- 它和Cartographer完全不是同一个思路。
  - ✓ Cartographer的3D部分，更像是2D的扩展：即用2D的思路去做3D的事情。而LOAM则主要解决3D问题，其核心思路难以解决2D问题。
  - ✓ 从代码风格来看，我认为它属于——学院派。根据其开源代码来看，和Cartographer令人望而生畏的代码量完全不是一个层次。当然这也解释了，为什么基于LOAM的衍生算法众多，但基于Cartographer的衍生算法却鲜有所闻。
- 那么，代码少，时间老，就证明它不行吗？作为一个7年前的算法，至今在自动驾驶数据集KITTI上的Odometry模块的激光SLAM排行榜上，仍然霸占榜一。

# LOAM

## 什么，是KITTI数据集

- 2011年，Andreas Geiger（KIT，德国卡尔斯鲁厄理工学院）、Philip Lenz（KIT）、Raquel Urtasun（TTIC，美国丰田工业大学芝加哥分校）三位年轻人发现，阻碍视觉感知系统在自动驾驶领域应用的主要原因之一，是缺乏合适的benchmark。而现有的数据集无论是在数据量，还是采集环境上都与实际需求相差甚远。于是他们利用自己的自动驾驶平台，建立起庞大的基于真实场景下的数据集，以此推动计算机视觉和机器人算法在自动驾驶领域的发展。这便是KITTI数据集的诞生背景。
- KITTI数据采集平台：1个惯性导航系统，1个64线3D激光雷达，2个灰度摄像机，2个彩色摄像机，以及4个光学镜头。





# LOAM

## 什么，是KITTI数据集

- KITTI数据集主要有以下Benchmark: road（用于道路分割）， semantics（用于语义分割）， object（2D、3D和鸟瞰三种视角，用于目标检测）， depth（用于视觉深度评估）， stereo（用于双目立体视觉和三维重建）， flow（用于光流预测）， tracking（用于目标跟踪）， odometry（里程计）
- 里程计数据集地址: [http://www.cvlibs.net/datasets/kitti/eval\\_odometry.php](http://www.cvlibs.net/datasets/kitti/eval_odometry.php)
- 数据集中包含了22个立体视觉序列以无损png的形式保存，另外还提供了velodyne64线激光雷达数据。评估标准: 平移误差（百分比）， 旋转度数（度每米）。

前三名:

1. SOFT2: 萨格勒布大学电气工程与计算学院采用的方法，未发表论文，使用的是双目视觉;

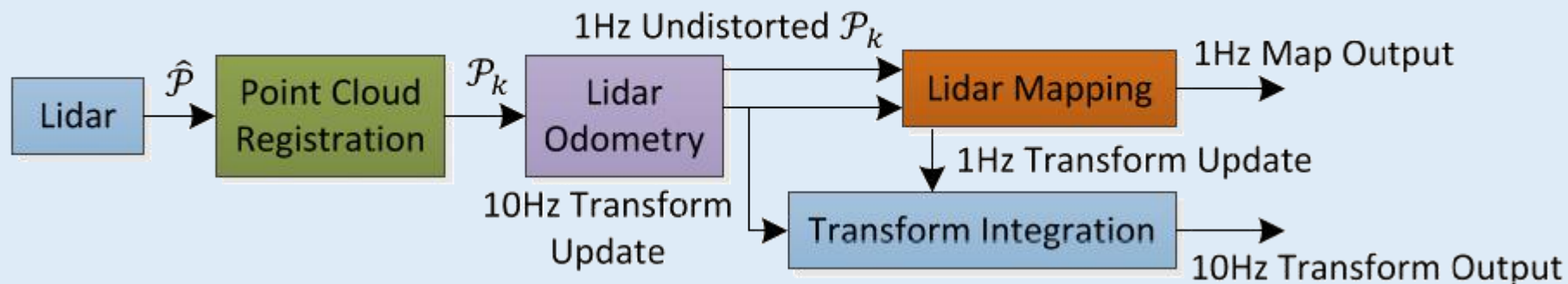
2.V-LOAM: 基于LOAM的改进版本，同时使用了激光-视觉数据进行融合，也是LOAM作者Zhang Ji于ICRA2015所发，但代码未开源。

3.LOAM: 本节所述方法。

	Method	Setting	Code	Translation	Rotation
1	<a href="#">SOFT2</a>			0.53 %	0.0009 [deg/m]
2	<a href="#">V-LOAM</a>			0.54 %	0.0013 [deg/m]
J. Zhang and S. Singh: <a href="#">Visual-lidar Odometry and Mapping: Low drift, Robust, and Fast</a> , IEEE International C					
3	<a href="#">LOAM</a>			0.55 %	0.0013 [deg/m]
J. Zhang and S. Singh: <a href="#">LOAM: Lidar Odometry and Mapping in Real-time</a> , Robotics: Science and Systems Co					
4	<a href="#">TVL-SLAM+</a>			0.56 %	0.0015 [deg/m]
5	<a href="#">CT-ICP</a>			0.59 %	0.0014 [deg/m]
6	<a href="#">zPICP</a>			0.62 %	0.0016 [deg/m]
7	<a href="#">HELO</a>			0.62 %	0.0018 [deg/m]
8	<a href="#">HMLO-whu</a>			0.63 %	0.0014 [deg/m]
9	<a href="#">HMLO</a>			0.64 %	0.0015 [deg/m]
10	<a href="#">FMLO</a>			0.64 %	0.0013 [deg/m]
11	<a href="#">SOFT-SLAM</a>			0.65 %	0.0014 [deg/m]
I. Cvišić, J. Česić, I. Marković and I. Petrović: <a href="#">SOFT-SLAM: Computationally Efficient Stereo Visual SLAM for Au</a>					

# LOAM

框架



**Point Cloud Registration:** 点云不是同一时刻获取的，每一个帧点云，其中的每一个点，都是不同时刻获取的，因此把它进行运动补偿：获取每个点的时间戳，位姿插值，把点云先投影到同一时刻；**提取特征点。**

**Lidar Odometry:** 估计两帧点云之间的位姿变换，获得**两个时刻之间的相对位姿**，频率较高 10Hz

**Lidar Mapping:** 建图模块，把连续10帧的点云数据和整个地图匹配，获得**世界坐标系下的位姿**，频率较低 1Hz。

**Transform Intergration:** 实时利用世界坐标系下的位姿和两个时刻之间的相对位姿，更新各个时刻世界坐标系下的位姿

# LOAM

## 特征点

- 回顾：Cartographer使用栅格地图，地图中存储着占据概率，通过把点云投影到栅格地图，计算匹配得分，找到最合适的投影，作为位姿变换。
- 但是，LOAM使用的是点云地图，那么点云投影后，进行匹配的就还是点云地图。



# LOAM

- LOAM作者决定对点云提取特征，然后根据稀疏的特征来计算位姿变换。
- 作者决定提取两种特征点：**平面点**和**边缘点**。
- 如何区分这两种点？
- 作者引入了一种计算方法——曲率。
- 计算曲率听起来是一个很麻烦的事情，在高等数学中，一条曲线的曲率以如下公式进行计算：

$$K = \frac{|y''|}{(1 + y'^2)^{\frac{3}{2}}}$$

# LOAM

- 但事实上作者并不是这样算的。他直接利用激光的每条扫描线中，**一个点前后各五个点，计算平均值到该点的距离。**

$$c = \frac{1}{|S| \cdot \| \mathbf{X}_{(k,i)}^L \|} \sum_{j \in S, j \neq i} \| \mathbf{X}_{(k,i)}^L - \mathbf{X}_{(k,j)}^L \| \quad (1)$$



假如有11个点均匀分布在同一条直线上，红点附近的点求均值，就是它自身，到它的距离就是0  
曲线越弯曲，这个值越大。

**平面点：**在三维空间中处于平滑平面上的点，其和周围点的大小差距不大，**曲率较低**，平滑度较低。

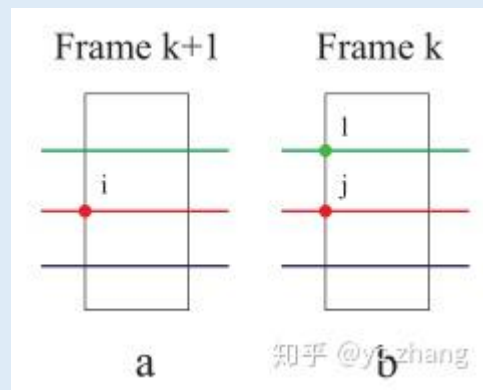
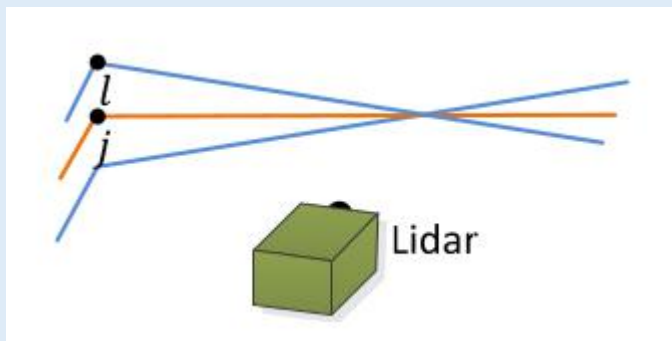
**边缘点：**在三维空间中处于尖锐边缘上的点，其和周围点的大小差距较大，**曲率较高**，平滑度较高。

作者对整个扫描进行分段，分四段，每段各取2个边缘点和4个平面点。

# LOAM

在确定了边缘点和平面点后，两帧点云将根据这两种特征点进行运动估计。

边缘点匹配方法：



红、绿、蓝代表多线激光雷达的扫描线；  
在第k+1帧中，边缘点i处在红色的扫描线上；

在第k帧中，红线上的边缘点j和i更近，在相邻绿线上再找到一个最近的边缘点l，那么l和j构成一个边缘。

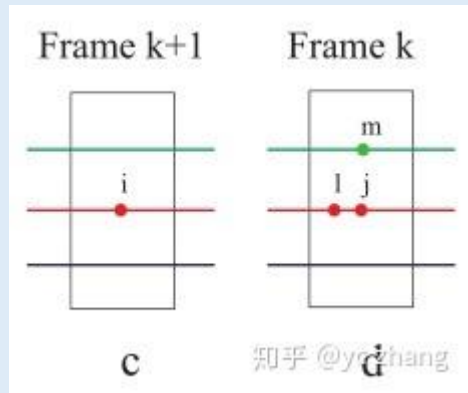
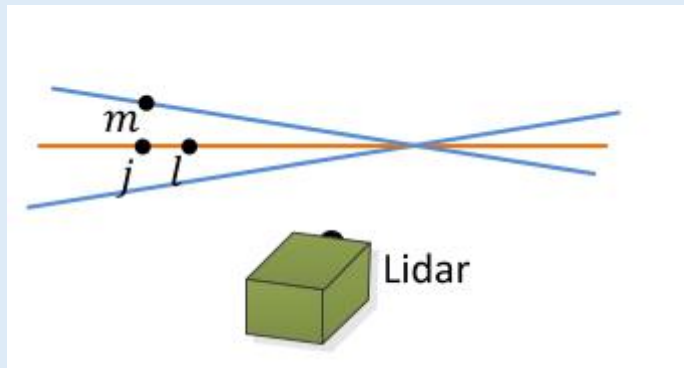
因此，对边缘特征点来说，优化的目标就是，**i到直线l的距离最近。**

$$d = \frac{|(X_2 - X_1) \times (X_1 - X_0)|}{|X_2 - X_1|}$$

向量叉乘的模长，代表平行四边形的面积；  
除以底，代表平行四边形的高；  
也就是点到直线的距离。

## LOAM

平面点匹配方法:



$$d = \frac{|((X_2 - X_1) \times (X_3 - X_1)) \cdot (X_0 - X_1)|}{|(X_2 - X_1) \times (X_3 - X_1)|}$$

分子：三维物体的体积；

分母：地面构成的平行四边形的面积；

高=体积/面积

也就是点到平面的距离。

红、绿、蓝代表多线激光雷达的扫描线；  
在第k+1帧中，平面点i处在红色的扫描线上；

在第k帧中，红线上的平面点j和i更近，**在同线上**再找到一个附近的平面点l，**在相邻绿线上**再找到一个最近的平面点m，那么lmj构成一个平面。

因此，对平面特征点来说，优化的目标就是，i到平面mlj的距离最近。

# LOAM

- 作者论文中使用的求解方法是**列文伯格-马尔夸特算法 (LM)**，又叫阻尼牛顿法，可以避免求解中遇到的非奇异和病态问题。
- 有趣细节：作者发表时的源代码，和目前流传的代码并不相同（源代码可能由于后续维护和其他商业原因下架）。其声称使用了LM算法，但实际上在代码中被人发现用的是比较简单的高斯牛顿法（也许重要内容没有开源，也许是这样论文看起来更复杂），使用的是OpenCV自带的solve函数迭代求解增量，轮子是自己造的。
- 目前流传的代码为**A-LOAM**，代码简介，易于学习。主要是后人对其工程代码进行了改进，使用了和Cartographer一样的Ceres库，直接用求解器自动求导，省去了对雅可比矩阵的手工推导，因此代码简洁了不少。
- **F-LOAM**：和A-LOAM类似，但是使用的是**解析式求导**，速度更快。

# LOAM

- 到目前为止，根据特征点，计算了**相邻两个时刻点云之间的位姿变换**。然而，**我们希望得到的是世界坐标系下的位姿**。
- 是否可以直接从第1帧开始，逐步递推，得到每个时刻点云之间的位姿变换呢？
- 毫无疑问，这样会把误差越累积越大，就好比只用轮式里程计递推定位一样。
- LOAM虽然没有回环检测（不能发现自己到了相同的位置），但这并不代表它仅靠前端递推。
- LOAM在建图部分，采用map-map的匹配方法，用连续10帧的激光点云数据，和10立方米之内的地图做一个匹配。**也就是说，在第25帧的位姿，不是从0帧-1帧，1帧-2帧.....一直递推到第25帧的。而是10~20帧之间构建的点云地图和全局地图匹配得到的第20帧位姿开始递推，从20~21，21~22.....直到第25帧。**

# LOAM

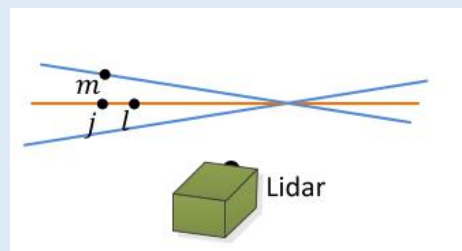
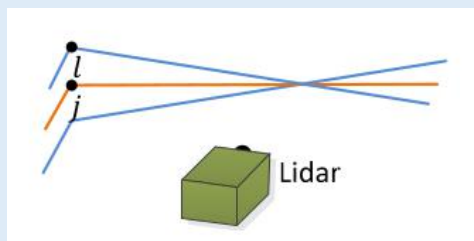
## 地图构建和全局位姿

- 建图部分采用map-map的匹配，仍然是基于边缘点和平面点，投影点云，计算点到直线和点到平面的距离。
- 里程计的匹配，是用两帧点云数据；
- 建图的匹配，是用10帧点云数据，和10立方米范围内的整个地图匹配。
- 特征点增加了10倍！

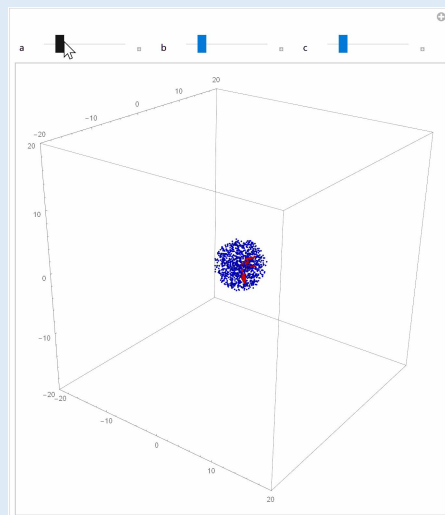


## LOAM

- 前端寻找边缘线和平面，使用的是最近邻的方法。



- 后端则不同，使用的是特征点周围的点云簇，通过主成分分析（求解协方差矩阵的特征值和特征向量）确定边缘线和平面。



红色箭头代表特征值方向。  
特征值两长一短，则是平面，短的那个作为平面的法向量。  
特征值两短一长，则是边缘，长的那个作为边缘的方向向量。

三个向量相交于几何中心，由此确定边缘线和平面。  
后端其他内容与前端相同。

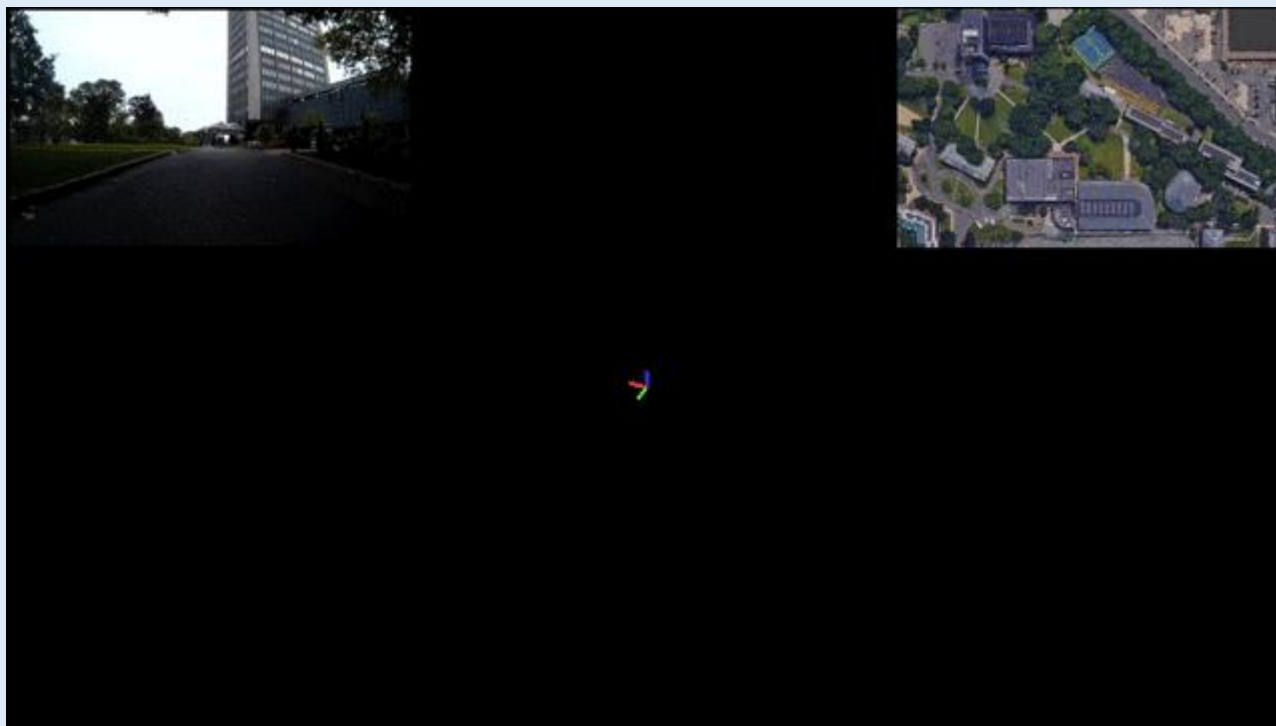
# LOAM

## 优缺点

- LOAM的优点是：1.提出了新颖的特征提取方式。2.根据时间戳，对旋转的雷达采集时间不一致进行运动补偿。3.融合了 scan to scan（里程计部分） map to map（建图部分）的思想。
- LOAM的缺点也很明显：1.没有回环检测。（发表时间较早，比谷歌Cartographer要早两年） 2.不能处理大规模的旋转变换。

- 作为一个14年提出的算法，为什么能一直占据Kitti的榜首呢？之后的算法就都不如它吗？
- 我的看法：
- 首先，kitti中odometry指的是里程计，里程计不等同于定位，里程计的评价指标为每100m误差多少米——LOAM在0.55%左右。而有回环检测的算法，则发现回环时会修正全局轨迹，取得更高的结果。而Kitti中并非所有序列都有回环。
- 其次，Kitti数据集中累积的公里数大概为39km左右，由汽车采集，也没有大范围的旋转运动。

# LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain .



本节目录:

介绍

框架

LOAM存在的问题

分割

特征点提取

运动估计

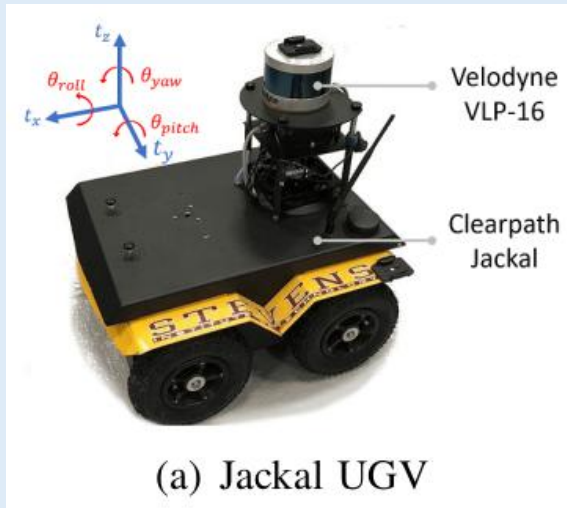
地图构建和全局位姿

回环检测

不足之处

# Lego-LOAM

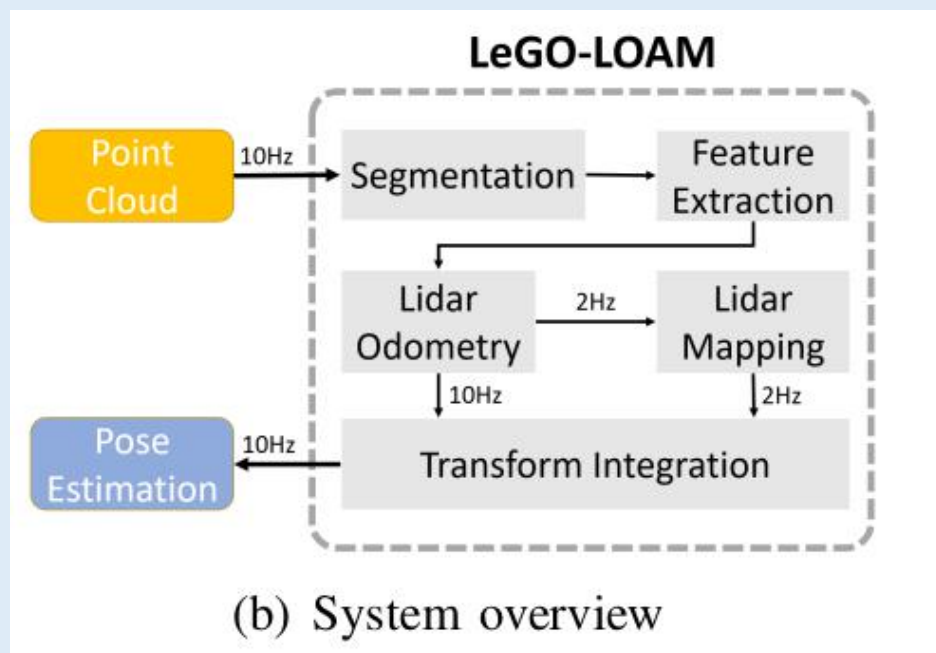
- Lego-LOAM是TiXiao Shan发表在IROS2018的文章，文章叫：可变地形下的轻量级和地面优化的雷达里程计与建图。
- 其是以LOAM为框架衍生出的新算法，主要在于两点提升：轻量级 和 地面优化。



使用的是加拿大clearpath公司的 Jackal小车（这一款价格比较昂贵）  
装载VLP-16线激光雷达

# Lego-LOAM

## 框架



Segmentation: 对点云进行分割, **分割为地面和非地面区域**;

Feature Extraction: 基于分割后的点, 和LOAM类似的算法提取出边缘点和平面点;

Lidar Odometry: 基于提取的特征点, **scan-to-scan推算两帧激光之间的相对位姿变换 (使用两次LM优化)**, 频率较高 (10Hz) ;

Lidar Mapping: scan-to-map, 构建全局地图, 获得世界坐标系下的位姿, 频率较低 (2Hz) ;

Transform Intergration: 与LOAM相同, 实时利用世界坐标系下的位姿和两个时刻之间的相对位姿, 更新各个时刻世界坐标系下的位姿

# Lego-LOAM

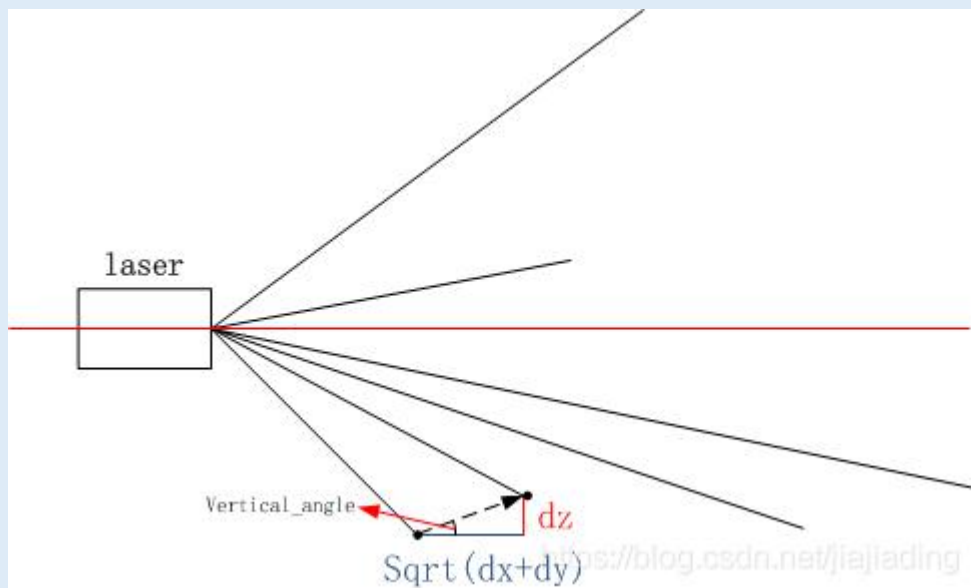
## LOAM存在的问题

- 没有回环检测
- 计算时间复杂度较高，基于三维空间中的位姿进行优化；
- 户外可能受到各种噪声影响，例如树上摇晃的树叶，地上的杂草。而这些点未必会重复出现在前后两帧激光中。而错误的特征点将会影响位姿精度。
- LOAM需要提取平面点和边缘点，由于车体上下颠簸，竖直维度提取的平面点很容易造成误差。
- 因此Lego-LOAM的contribution：
  - 1.着重于解决一些非城市化道路或非平整道路上LOAM存在的问题。
  - 2.轻量化，改进算法，使其在TX2上也可以实时运行。

# Lego-LOAM

## 分割

- 分割部分首先把点云投影到一张距离图像中：
- 以16线激光雷达为例，其竖直方向为16根，因此竖直方向的分辨率为16；其水平方向角度分辨率为0.2度，因此水平方向分辨率为 $360/0.2=1800$ ，即构建一张 $1800*16$ 的距离图像，其中每个像素的值为对应点云到传感器的欧几里得距离。
- 区分地面点和非地面点：



要求激光雷达安装平行于地面。

为什么要提取地面点？

**就算车体颠簸，路面基本在相邻帧之间变化是不大的。**

先利用提取的地面平面点，调整z, roll, pitch;

再利用地面上竖直的边缘点，调整x, y, yaw;

不仅更准确了，还实际缩小了优化的规模。

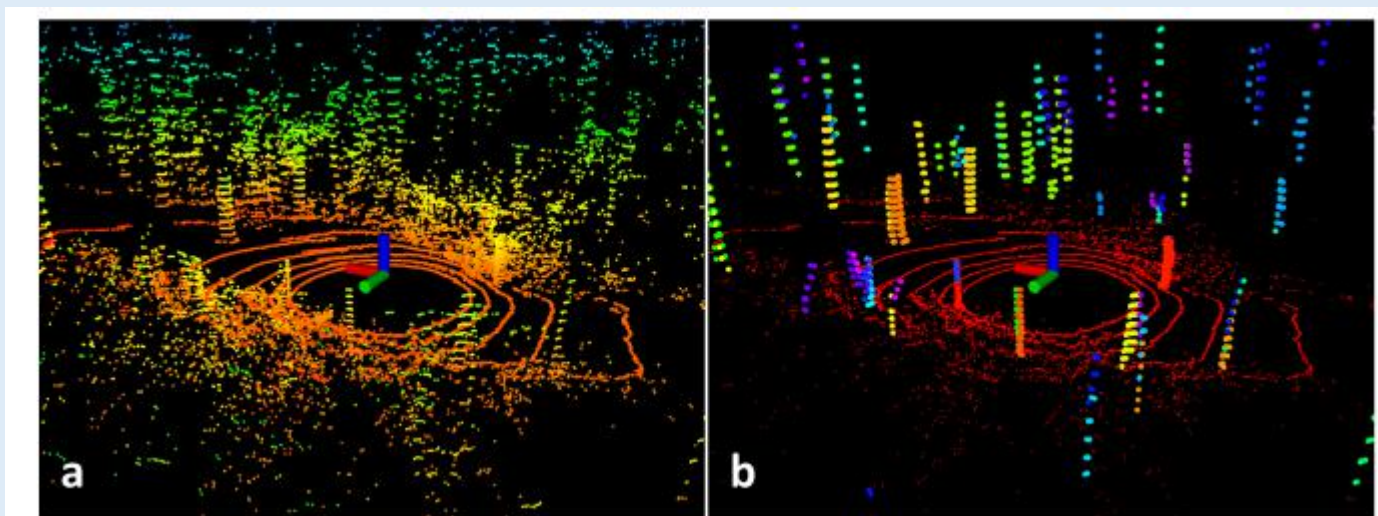
把一个 $O(n^6)$ 的问题变成了两个 $O(n^3)$ 的问题。



# Lego-LOAM

## 特征点提取

- 在提取特征点之前，对点云进行一个聚类操作：低于30个数据点的类别就当成噪声处理，这样保存下来的点就是一些相对比较静态的物体了。



右图为聚类处理后的点云，红色部分被分割为地面点。

聚类和分割的内容应该不是其首先提出，细节可以参考：

I. Bogoslavskyi and C. Stachniss, “**Efficient online segmentation for sparse 3d laser scans,**” PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science, vol. 85, no. 1, pp. 41–52, 2017.

# Lego-LOAM

对前述形成的 $16 \times 1800$ 的距离图像，水平分成若干个子图像（sub-image），

对**每一个子图像**都进行特征提取操作：

依旧是和LOAM中相同的边缘点和平面点评判标准，根据曲率 $c$ ，区分边缘点和平面点。

之后，要划分两个**大集合**：

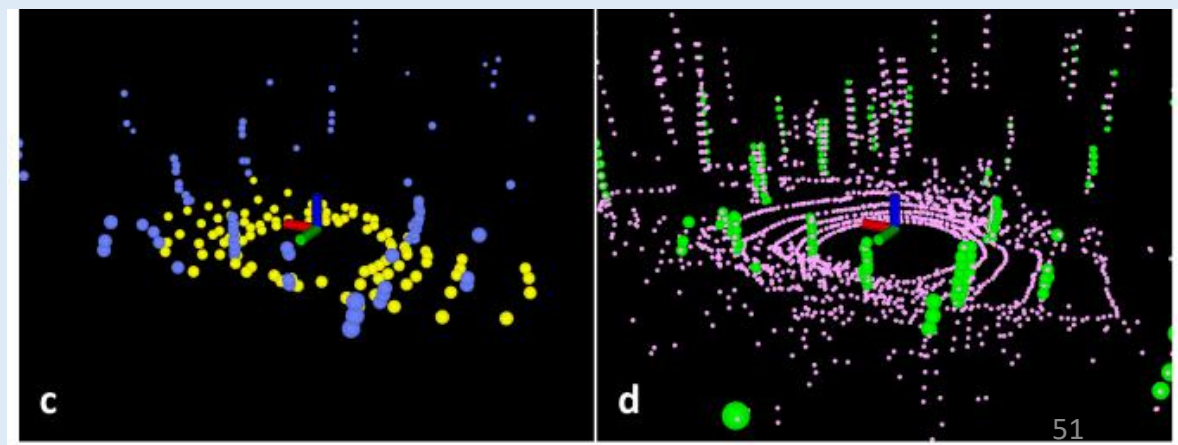
每一行中，选取有最大 $c$ 值的 $N\_Fme$ （40个）**边缘点**，要求**不得属于地面点**，组成集合 $F\_me$ ；

每一行中，选取有最小 $c$ 值的 $N\_Fmp$ （80个）**平面点**，组成集合 $F\_mp$ ；

之后，再从 $F\_me$ 和 $F\_mp$ 中，划分两个小集合：

$n\_fe$ （2个）边缘点；

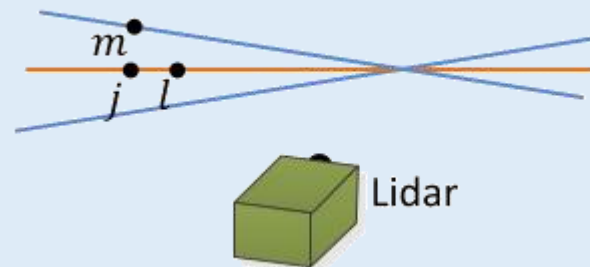
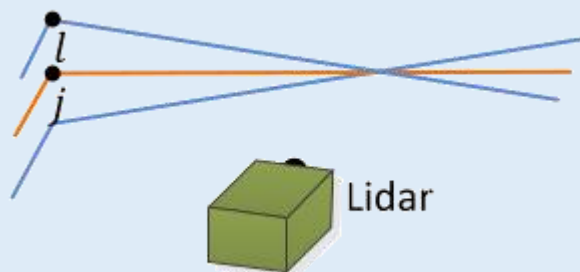
$n\_fp$ （4个）平面点，**要求是地面点**。



# Lego-LOAM

运动估计

- 雷达里程计部分，要根据两帧点云，确定相对的位姿变换：
- 从 $t$ 时刻的**小集合**中，选取边缘点，和 $t-1$ 时刻的**大集合**中的边缘点，构建点到线的关系，构建方法和LOAM一致。
- 从 $t$ 时刻的**小集合**中，选取平面点，和 $t-1$ 时刻的**大集合**中的地面点，构建点到面的关系，构建方法和LOAM一致。



- 构建了目标函数以后，仍然采用LOAM中的列文伯格-马尔夸特优化算法，但是分了两阶段，第一阶段计算竖直维度的 $z$ ,  $roll$ ,  $pitch$ 。**(根据地面的平面点来优化)**
- 第二阶段计算水平维度的 $x$ ,  $y$ ,  $yaw$ ，根据边缘点来优化。
- 两次LM优化得到相同的精度，实验计算时间减少35%。
- 代码中其实也不是LM算法，和LOAM类似，自己构建H矩阵并通过`opencv.solve()`函数求解。

# Lego-LOAM

- 与LOAM一致，Lego-LOAM同样也是采用了一快一慢两个部分，快的Lidar Odometry 10Hz来计算相对变换，慢的Lidar Mapping 2Hz来计算世界坐标系下的位姿。相当于10Hz在2Hz的基础上递推。
- **Lego-LOAM不再存储所有的传感器点云数据，而是只存储特征点集（前述的大集合）。**
- LOAM采取的是map-to-map的优化，**最近的十帧点云**与10立方米内的全局地图进行匹配，主成分分析，构建点到线、点到面的误差；
- Lego-LOAM采取的是scan-to-map的优化，scan为**当前帧点云中的特征点集**；map有两种取法：第一种，和LOAM中一致，10立方米；第二种，选一组时间上相近的**特征点云**，构建图优化问题。当前时刻的特征点云作为观测数据，当前位姿作为优化节点。
- 使用的优化库：不再是谷歌公司的Ceres，而是因子图优化库gtstam。

# Lego-LOAM

- 回顾：Cartographer中的回环检测：激光点云投影到栅格地图，**暴力搜索全局地图**，分支定界加速，确定初值以后用非线性优化精修；
- Lego-LOAM中加入了回环检测，是单独用一个进程运行的。
- 流程：
- 第一步，寻找历史位姿：1.**与当前位姿的距离上最近**。2.**与当前位姿间隔时间较长**。（这样才是回环）
- 第二步，把历史位姿作为候选，用ICP算法迭代，修正位姿。

# Lego-LOAM

我们可以看出：

- 1.相比Cartographer, **Lego-LOAM的回环检测就是一个简单的回环检测，其默认偏移较小。如果偏移较大，就不能修正了。**
- 2.不具备重定位功能。因为这种算法的前提是需要知道自己的大致位置，和历史中附近的位置进行匹配，修正位姿。

引入后续：SC-Lego-LOAM: <https://github.com/irapkaist/SC-LeGO-LOAM>

大致思路：从3D点云上空进行俯视，利用俯视图进行一个特征匹配。**从而获取范围更为广阔的回环检测与重定位。**

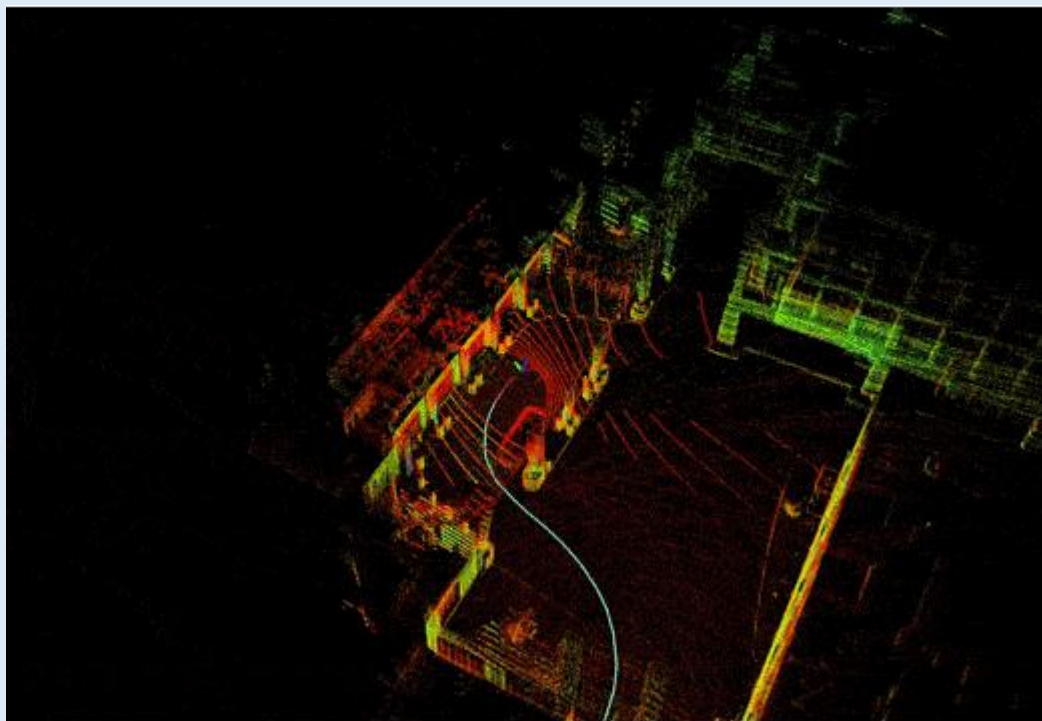
# Lego-LOAM

不足之处

- Lego-LOAM有一个显著的缺陷——依赖地面。如果用无人机，那么就难以确定地面了。
- 当然论文作者提到，对于无人机，则不提取地面点，直接就像LOAM中那样正常提取边缘点和平面点。但是我认为这样算法的核心优势就丢掉了。



# LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping



本节目录:

- 介绍
- 框架
- 预积分因子
- 雷达里程计因子
- GPS因子
- 回环检测因子
- 总结

# LIO-SAM

- LIO-SAM是TixiaoShan在2020年IROS发表的Lego-LOAM续作。
- 实际上也是Lego-LOAM的扩展版本，添加了IMU预积分因子和GPS因子。



(a) Handheld device



(b) Clearpath Jackal



(c) Duffy 21

使用了三种设备进行实验

# LIO-SAM

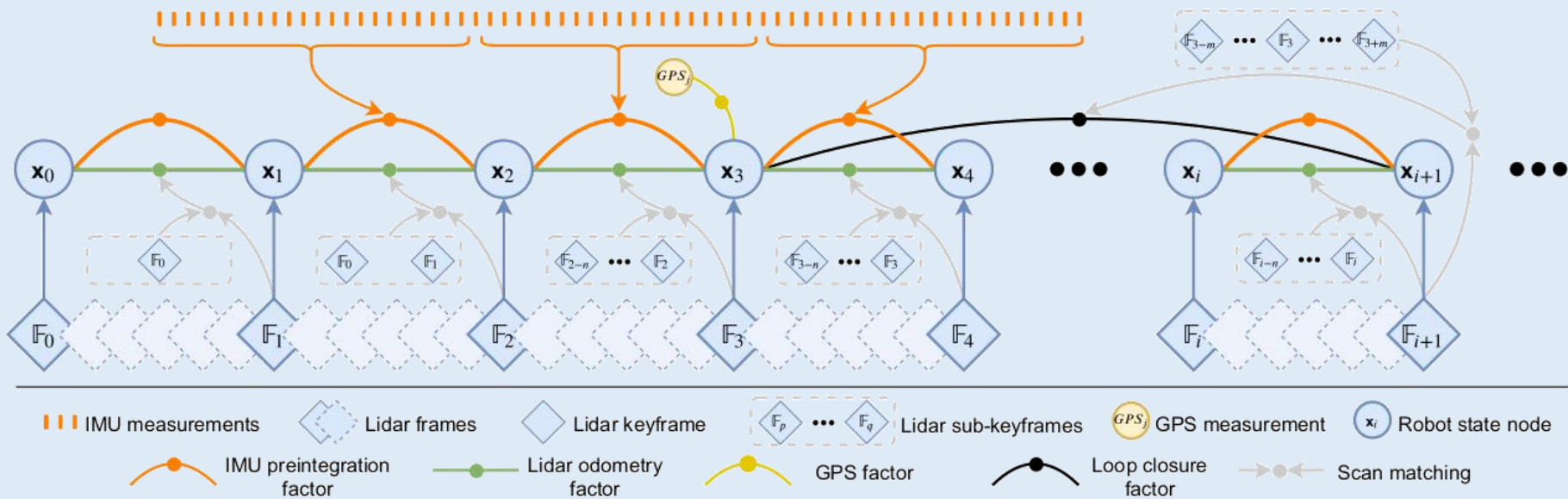


Fig. 1: The system structure of LIO-SAM. The system receives input from a 3D lidar, an IMU and optionally a GPS. Four types of factors are introduced to construct the factor graph.: (a) IMU preintegration factor, (b) lidar odometry factor, (c) GPS factor, and (d) loop closure factor. The generation of these factors is discussed in Section III

使用**关键帧Keyframe**，使用关键帧进行匹配，丢掉了关键帧之间的帧。（阈值设置为1m和10度）  
四种因子：

橙色：IMU预积分因子；

绿色：激光“关键帧”和“之前的N个关键帧构成的体素地图”进行匹配

黄色：GPS因子：当估计位姿的方差大于GPS位置方差时加入

黑色：回环检测因子，由关键帧和候选关键帧相邻的 $2m+1$ 个关键帧帧图匹配得到

# LIO-SAM

## 预积分因子

$$\hat{\omega}_t = \omega_t + \mathbf{b}_t^\omega + \mathbf{n}_t^\omega \quad (2)$$

$$\hat{\mathbf{a}}_t = \mathbf{R}_t^{\text{BW}}(\mathbf{a}_t - \mathbf{g}) + \mathbf{b}_t^{\mathbf{a}} + \mathbf{n}_t^{\mathbf{a}}, \quad (3)$$

$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \mathbf{g}\Delta t + \mathbf{R}_t(\hat{\mathbf{a}}_t - \mathbf{b}_t^{\mathbf{a}} - \mathbf{n}_t^{\mathbf{a}})\Delta t \quad (4)$$

$$\mathbf{p}_{t+\Delta t} = \mathbf{p}_t + \mathbf{v}_t\Delta t + \frac{1}{2}\mathbf{g}\Delta t^2 + \frac{1}{2}\mathbf{R}_t(\hat{\mathbf{a}}_t - \mathbf{b}_t^{\mathbf{a}} - \mathbf{n}_t^{\mathbf{a}})\Delta t^2 \quad (5)$$

$$\mathbf{R}_{t+\Delta t} = \mathbf{R}_t \exp((\hat{\omega}_t - \mathbf{b}_t^\omega - \mathbf{n}_t^\omega)\Delta t), \quad (6)$$

$$\Delta \mathbf{v}_{ij} = \mathbf{R}_i^\top (\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}\Delta t_{ij}) \quad (7)$$

$$\Delta \mathbf{p}_{ij} = \mathbf{R}_i^\top (\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i\Delta t_{ij} - \frac{1}{2}\mathbf{g}\Delta t_{ij}^2) \quad (8)$$

$$\Delta \mathbf{R}_{ij} = \mathbf{R}_i^\top \mathbf{R}_j. \quad (9)$$

- (2)~(3) :IMU可以测量自身的加速度和角速度，但一般认为其测量数据包括了两大噪声：随机游走噪声  $\mathbf{b}$  + 高斯白噪声  $\mathbf{n}$

- $\mathbf{B}$ 表示body坐标系，即imu坐标系； $\mathbf{W}$ 表示世界坐标系， $\mathbf{R}^{\text{BW}}$ 表示了从世界坐标系转换至IMU坐标系；

- (4)~(6) 分别表示  $t + \Delta t$  时间内，IMU的速度、位移和旋转；**注意，并非直接用原始数据积分，而是减去了随机游走 $\mathbf{b}$ 和高斯白噪声 $\mathbf{n}$ 。** bias未知，该变量也参与优化；

- (7)~(8)为*i*到*j*时刻的预积分，即把它**作为两时刻位姿的一个约束**，参与整体的图优化过程。

# LIO-SAM

## 雷达里程计因子

- 雷达里程计部分主要得到两帧之间的位姿变换;
- 方法基本与LOAM或Lego-LOAM的一致, 区别: 仅用**关键帧**和之前 $n+1$ 个**关键帧**中的特征集合构成地图, 进行匹配, 构建点到线、点到面的约束; (原先是使用帧到帧的匹配)
- 关于这点, 该方法没有使用Lego-LOAM的提取地面特征的方式, 因为在实验部分, 采用手持建图、轮船水面建图, 可能不适合Lego-LOAM该方式。



# LIO-SAM

GPS因子

- 收到一个GPS测量，即将其转换到笛卡尔坐标系下，构建一个新的约束；
- 时间戳不同步问题：插值解决；
- 当估计的位置的协方差大于GPS位置协方差时，才插入一个约束因子。

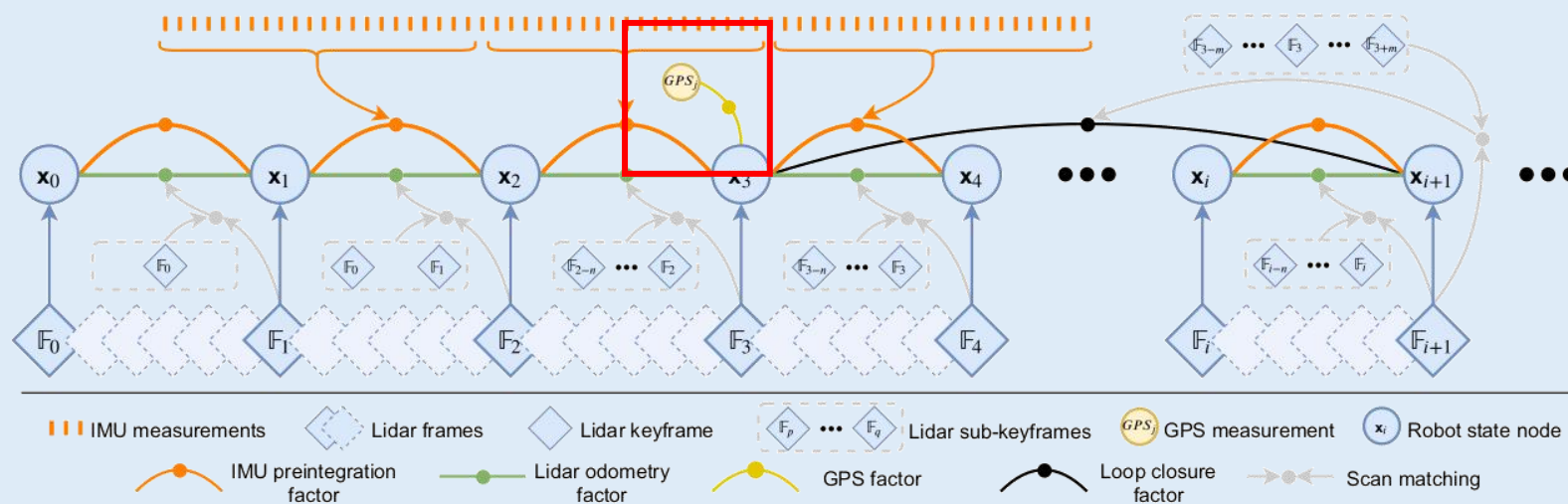


Fig. 1: The system structure of LIO-SAM. The system receives input from a 3D lidar, an IMU and optionally a GPS. Four types of factors are introduced to construct the factor graph.: (a) IMU preintegration factor, (b) lidar odometry factor, (c) GPS factor, and (d) loop closure factor. The generation of these factors is discussed in Section III

# LIO-SAM

## 回环检测因子

- 该方法使用的回环检测方法，应该和Lego-LOAM中的一致；
- 搜索当前位置15m内的最近历史位置，使用该历史位置的前后分别12个关键帧的特征，和当前匹配，构建约束。

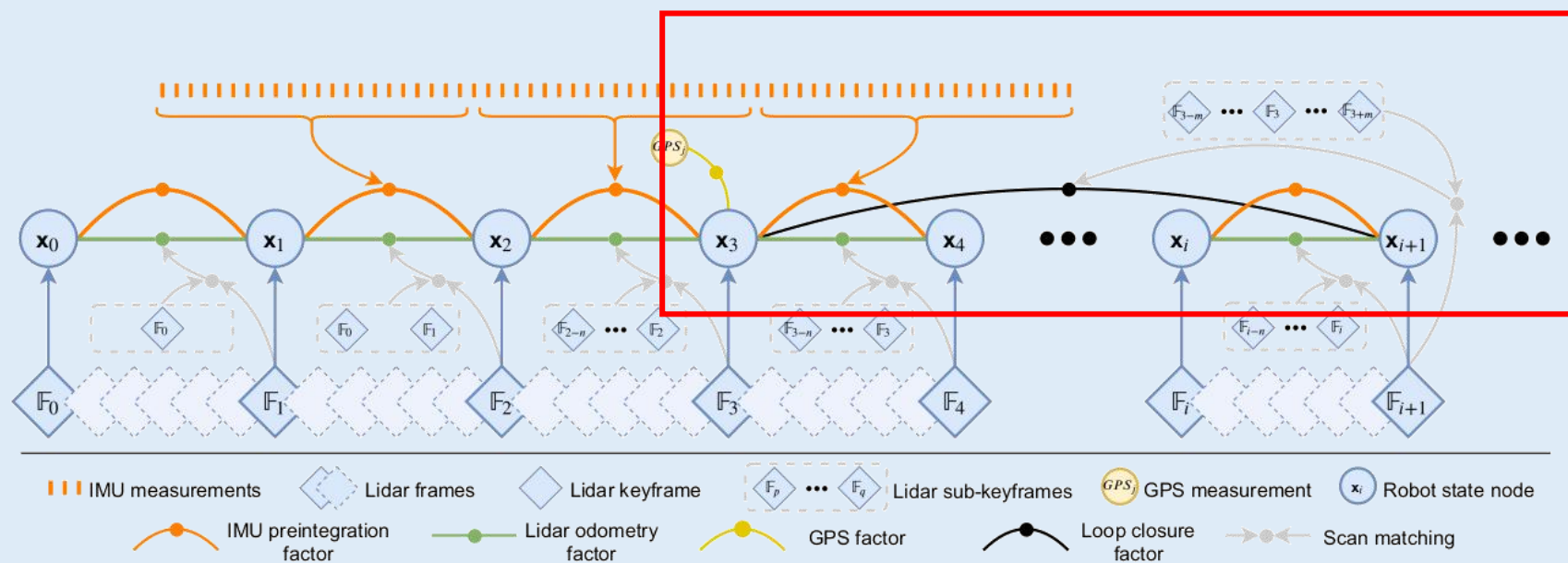


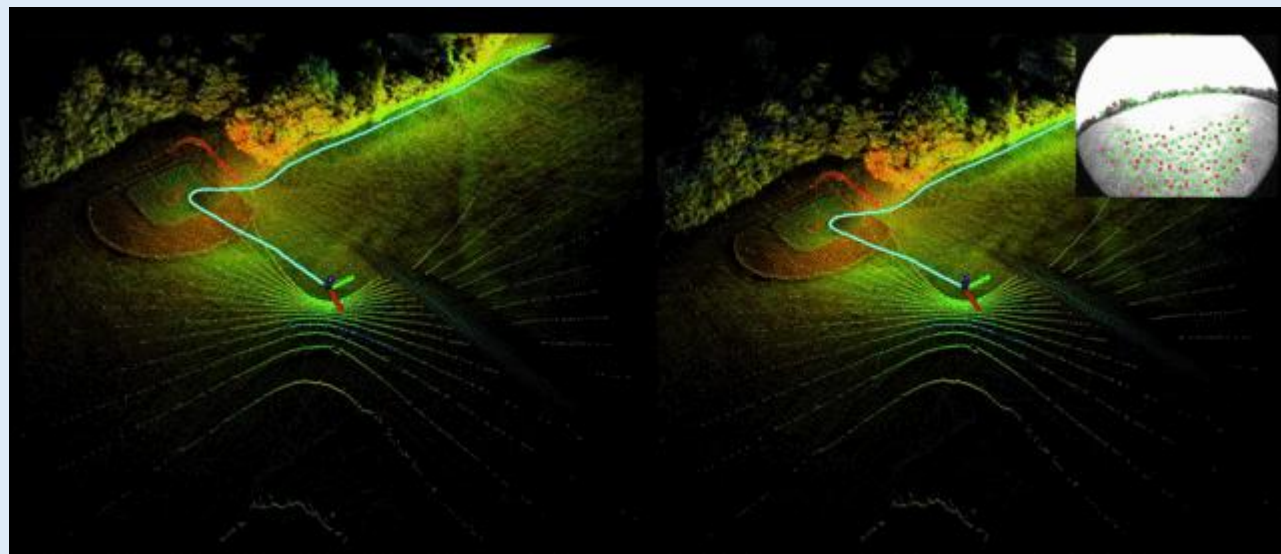
Fig. 1: The system structure of LIO-SAM. The system receives input from a 3D lidar, an IMU and optionally a GPS. Four types of factors are introduced to construct the factor graph.: (a) IMU preintegration factor, (b) lidar odometry factor, (c) GPS factor, and (d) loop closure factor. The generation of these factors is discussed in Section III

# LIO-SAM

- 该方法使用多传感器融合的方法，利用因子图优化，计算位姿；
- 可以把节点（位姿）理解成一个待求解的变量；**各种传感器数据构建的约束当成一个方程组；通过不断加入各种因子，相当于给方程组中加入更多的方程**，联合来求解最小二乘问题。
- 可以看出，回环检测部分基本和Lego-LOAM的特点一致，即不具备重定位能力（该方法前提是需要知道自己的大致位置，和历史中附近的位置进行匹配）
- 代码详细解析：<https://blog.csdn.net/zkk9527/article/details/117957067>



# LVI-SAM: Tightly-coupled Lidar-Visual-Inertial Odometry via Smoothing and Mapping



本节目录:

- 介绍
- 已有方法的不足
- 框架概述
- 视觉-惯性框架
- 雷达-惯性框架
- 回环检测

# LVI-SAM

## 介绍

- LVI-SAM为Lego-LOAM和LIO-SAM作者Tixiao Shan的最新工作，发表在ICRA 2021上。
- 提出了一个基于图优化的多传感器融合框架，具有多个子系统：**视觉惯性子系统**（VIS）和**雷达惯性子系统**（LIS）；单目+雷达+imu融合
- 鲁棒性：任一子系统失效，不会导致整个系统挂掉。

# LVI-SAM

- 基于激光的方法，在结构比较简单的环境里，容易失效——因为特征只有一个简单的“距离”信息；
- 基于视觉的方法，容易受到光照改变、快速运动导致的图像模糊等问题；所以，一般会加入IMU传感器——但IMU具有bias，其估计的并不是很准。

- 视觉-惯性里程计的著名工作——Vins-mono（香港科技大学沈劭劼团队于2018年发表在IEEE Transactions on Robotics上的工作，可以利用单目的视觉和IMU实现融合）可以达到非常好的效果，但是**其需要一个初始化过程**。
- 简单的说：**单目相机无法观测实际尺度，因此就无法和具有实际尺度的IMU数据进行融合**，必须经过一个初始化过程让二者联系在一起。**Vins-mono在初始静止或匀速运动时，IMU没有加速度，初始化会失败**，进而导致后续其他问题——这也是单目视觉惯性里程计的通病。
- 引申：Vins-mono的初始化过程：
- 给一个滑窗，选择参考帧，计算参考帧位姿；
- SFM方式估计路标点深度，求解出窗口内的所有位姿；
- 对齐相机和IMU的轨迹来估计外参，用旋转来优化陀螺仪bias，用平移来优化重力方向，速度，尺度；
- .....（非线性系统需要基于初值来不断优化）

# LVI-SAM

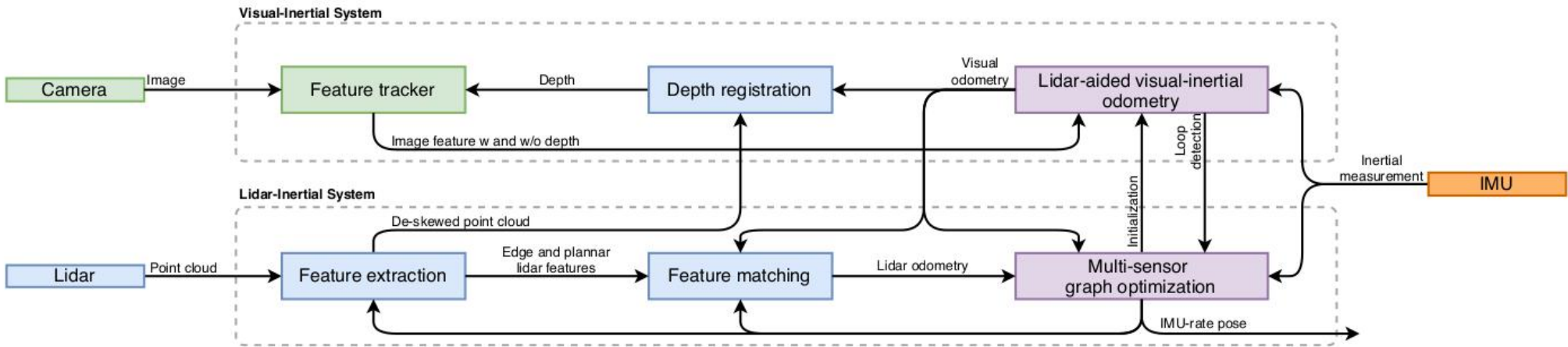


Fig. 1: The system structure of LVI-SAM. The system, which receives input from a 3D lidar, a camera and an IMU, can be divided into two sub-systems: a visual-inertial system (VIS) and a lidar-inertial system (LIS). The VIS and LIS can function independently while using information from each other to increase system accuracy and robustness. The system outputs pose estimates at the IMU rate.

LVI-SAM使用了两个子系统，视觉惯性系统（VIS）和雷达惯性系统（LIS）：

- 1.视觉惯性系统用雷达惯性系统来初始化；——用LIS计算的位姿，作为VIS的初始优化值；
- 2.VIS中需要根据**图像计算特征的深度**，可以利用**雷达的测量数据**来辅助优化；
3. LIS计算两个点云位姿变换的优化初值，同样也可以用VIS的视觉估计来计算；
4. 回环检测可以用**视觉信息确定初值**，再用雷达数据优化。

## LVI-SAM

例如:

LVI-SAM中视觉初始化  
入口部分:

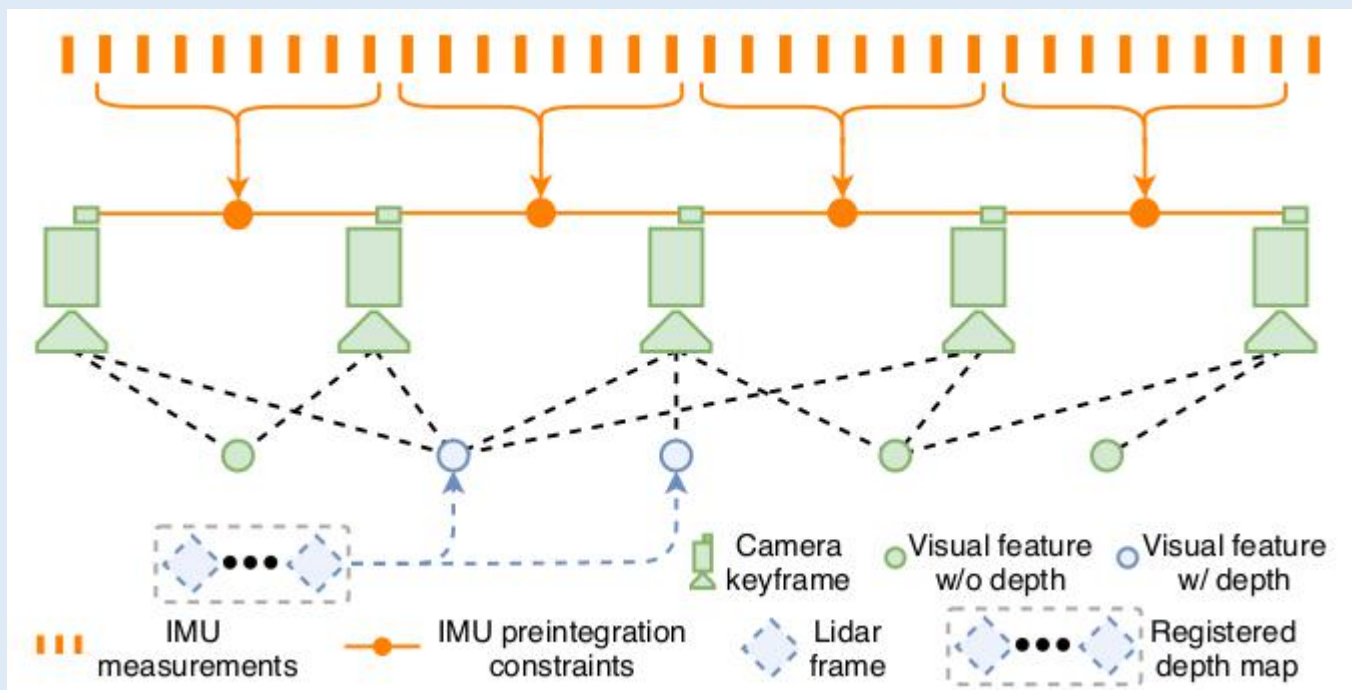
```
// Get initialization info from lidar odometry  
vector<float> initialization_info;  
m_odom.lock();  
initialization_info = odomRegister->getOdometry(odomQueue, img_msg->  
header.stamp.toSec() + estimator.td);  
m_odom.unlock();  
estimator.processImage(image, initialization_info, img_msg->header);
```

vins-mono视觉初始化  
入口部分:

```
estimator.processImage(image, img_msg->header);
```

# LVI-SAM

## 视觉-惯性框架



视觉-惯性框架（VIS）基本上采用Vins-mono的思路，但是区别有三：

- 1.初始化位姿和IMU的初始bias直接由雷达-惯性框架来估计（前述 vins-mono缺陷）
- 2.特征像素的深度，从雷达数据中获取；（vins-mono是通过三角测量计算，不够准确）
- 3.检测失效：当图像的特征点数目太少——无法确定匹配关系；

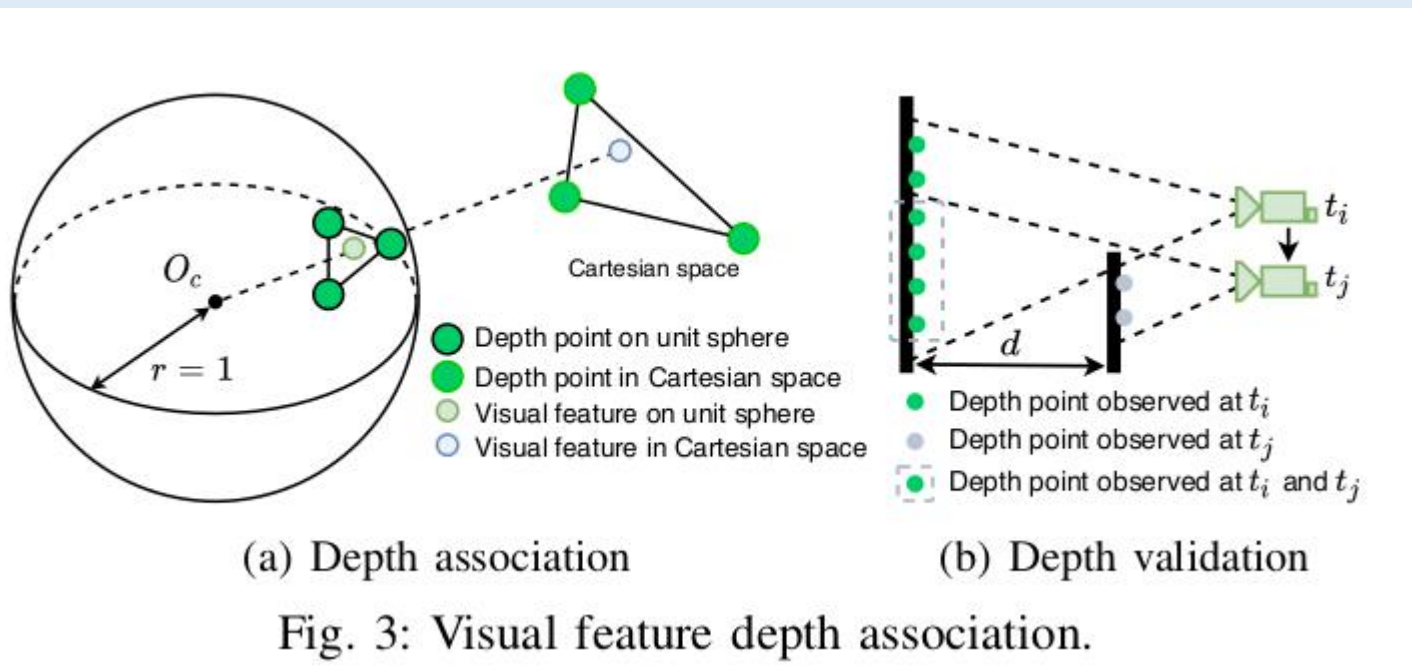
当vins-mono失效时——发现估计出的IMU偏置bias很大；

一旦检测到失效，立即用雷达-惯性框架的结果重新对视觉-惯性框架进行初始化。



# LVI-SAM

- 特征像素的深度，如何从雷达中获取？

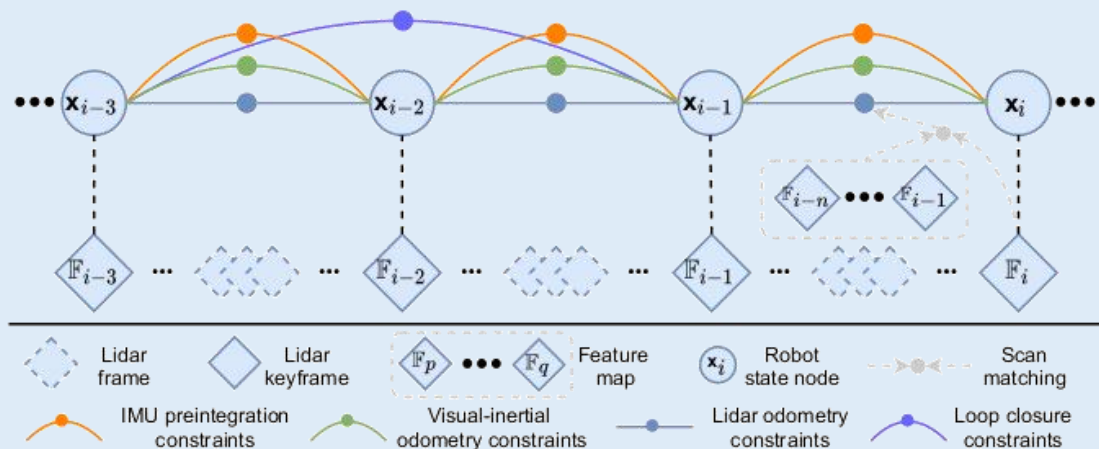


- 把视觉特征点和雷达点云，投影到归一化相机球体坐标系下，找到像素点最近的三个雷达点云，从而确立关联深度。
- 周围的三个点如果自身差距太大（超过2m），则丢弃不要。

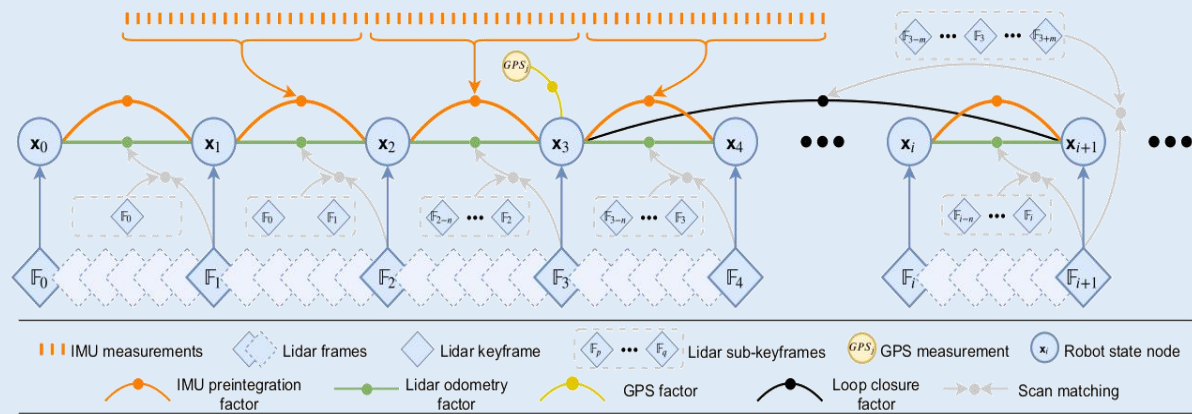


# LVI-SAM

## 雷达-惯性框架



LIS

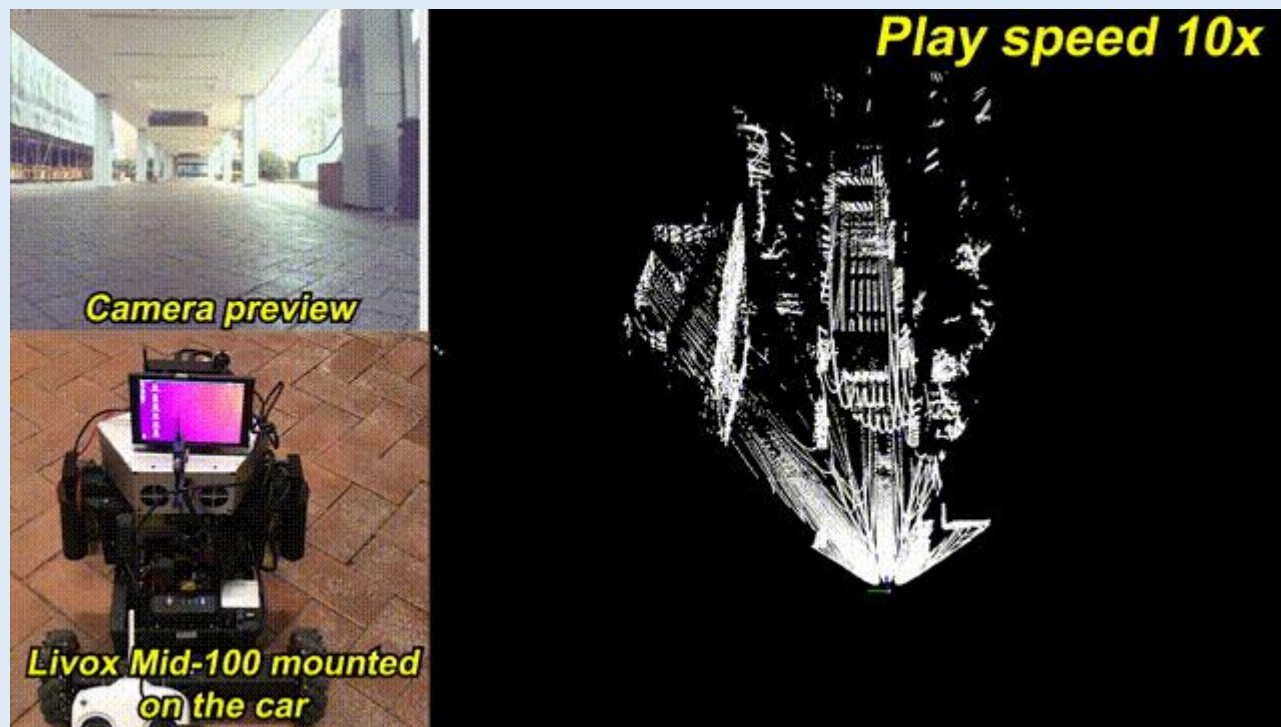


上一节LIO-SAM

雷达惯性框架（LIS）基本上在LIO-SAM的基础上做的改进：

- 在最开始，使用IMU的积分作为先验（假设IMU的bias、noise和速度都是0）；之后就采用视觉惯性框架VIS的值作为初始值优化。
- 在LIO-SAM的预积分因子、雷达里程计因子的基础上，变为预积分因子、雷达里程计因子、视觉惯性里程计因子三项，回环检测因子则由视觉部分提供。
- 失效检测：对两个关键帧进行匹配的最小二乘方程中，系数矩阵和其转置矩阵相乘的特征值小于一定阈值，认为失效，丢弃本次约束；

## 扩展——固态激光雷达livox-loam



本节目录:

- 介绍
- 固态激光雷达和机械雷达的区别
- 筛掉不好的点
- 位姿的迭代估计
- 外点和动态点的过滤
- 回环检测

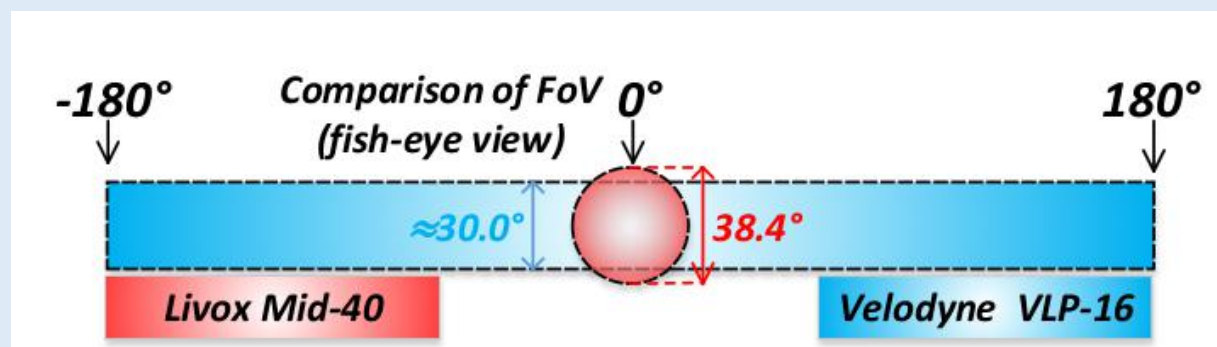
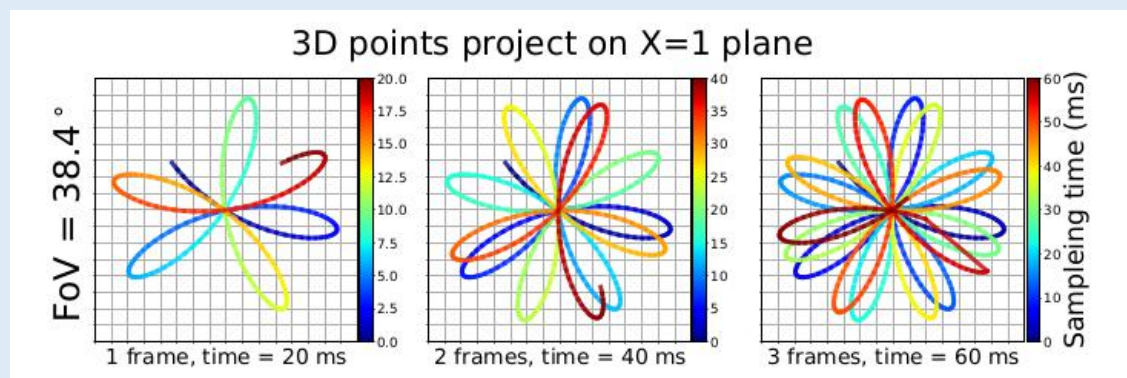
# Livox-loam

- Livox-loam为香港大学于2019年开源的工作，其主要由两篇论文组成：
- Loam livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV 这篇主要讲算法前端和后端，针对于LOAM的改进方法；
- A fast, complete, point cloud based loop closure for LiDAR odometry and mapping 这篇主要讲算法使用的回环检测方法；
- 两篇文章共享同一个github开源链接。

# livox-LOAM

## 固态雷达和机械雷达的区别

- 机械雷达：通过机械方式改变扫描方向；类似于一个激光笔，放在电机上旋转；十六线雷达，就是依次改变扫描的角度和高度，进行旋转。
- 固态雷达：通过阵列干涉改变扫描方向。类似于一个激光笔自己不动，改变激光射出的方向和角度。



# livox-LOAM

筛掉不好的点

- livox-loam依然采用和LOAM一样的方式，选取边缘点和平面点。但是它和LOAM最大区别就是，在选取候选的特征点时，变得更加“小心”了。
- 四大原则：

1.接近视角边缘的点不要。说白了就是射线和x轴夹角大于17度的就不选。

$$\phi(\mathbf{P}) = \tan^{-1} \left( \sqrt{(y^2 + z^2)/x^2} \right)$$

2.反射强度太大或者太小的点不要。

$$D(\mathbf{P}) = \sqrt{x^2 + y^2 + z^2}$$

$$I(\mathbf{P}) = R/D(\mathbf{P})^2$$

R是目标反射率，D是点云到光心的距离。

目标反射率可以由雷达直接获得。

- 反射率高，距离近，则认为反射强度大：会对雷达硬件的信号接收电路产生冲击，从而影响这个点的精度
- 反射率低，距离远，则认为反射强度小：信噪比低，不可信。



# livox-LOAM

3.和平面夹角很小的点不要

$$\theta(\mathbf{P}_b) = \cos^{-1} \left( \frac{(\mathbf{P}_a - \mathbf{P}_c) \cdot \mathbf{P}_b}{|\mathbf{P}_a - \mathbf{P}_c| |\mathbf{P}_b|} \right)$$

4.部分被遮挡的点不要

$$|\mathbf{P}_e - \mathbf{P}_d| \geq 0.1 |\mathbf{P}_e|, \text{ and } |\mathbf{P}_e| > |\mathbf{P}_d|$$

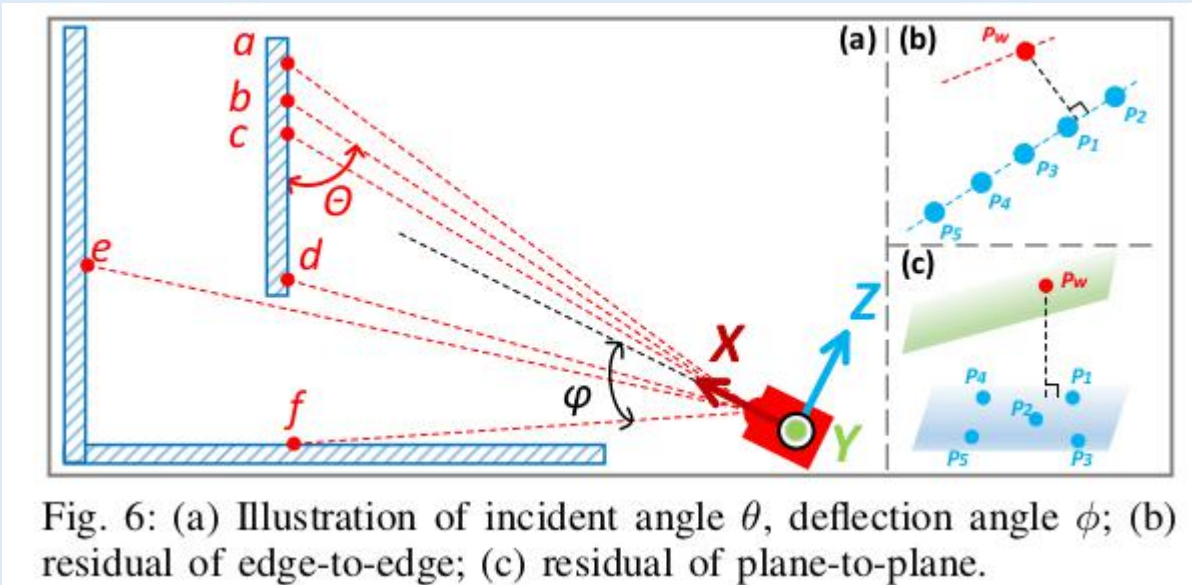


Fig. 6: (a) Illustration of incident angle  $\theta$ , deflection angle  $\phi$ ; (b) residual of edge-to-edge; (c) residual of plane-to-plane.

# livox-LOAM

- 选取特征点（边缘点，平面点）的方式和LOAM中一致（根据曲率），还另外多一种选取方式：
- 利用反射强度信息：
- 如果相邻点的反射强度区别很大，也认为是一个边缘点。

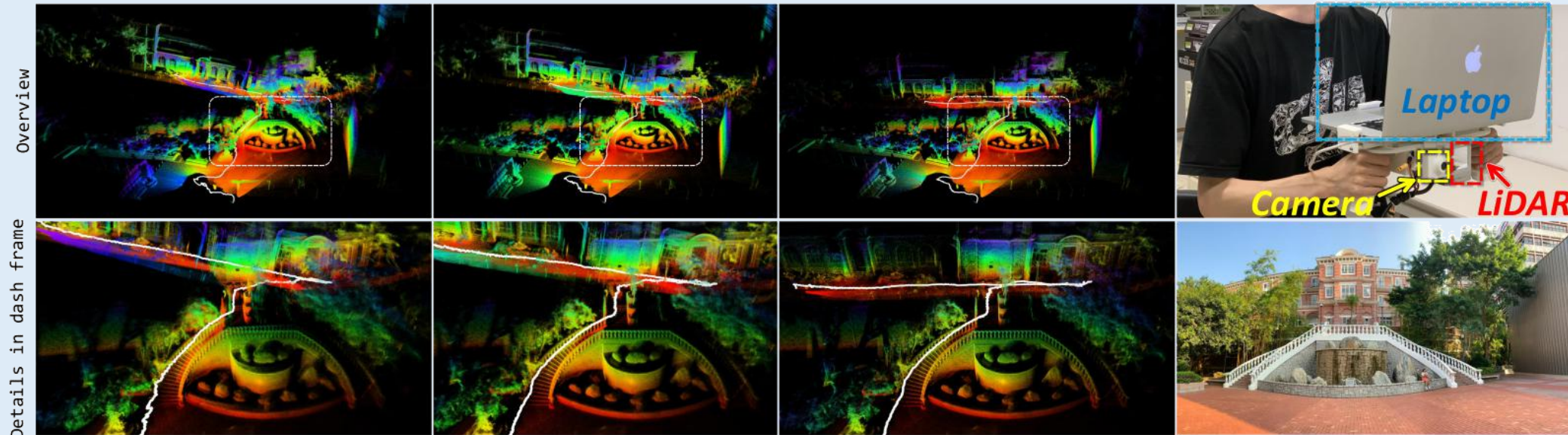
# livox-LOAM

- 计算位姿的目标函数，与LOAM中的一致，为点到线的距离（从边缘点中选点），点到面的距离（从平面点中选点），也采用协方差矩阵特征值的方式，提取边缘线和平面。
- 值得注意的是，**固态激光雷达虽然是不动的，但是绝对不意味着所有点都像相机一样是同时获取的。**
- 因此，每个点都是基于不同时间戳的。如果本身雷达自身处于运动状态，就会导致运动畸变。
- 算法采取了两种方式：线性插值和分段处理。



# livox-LOAM

- 分段处理即为把一帧数据，分成三段，分开并行匹配处理。（其实本质上也没有解决时间戳的问题）
- 线性插值为把两个时刻的位姿做一个插值，然后把每个点都插值找到位姿，投影到正确的位置。（类似LOAM的方式）



实验结果发现线性插值效果不太好，会有重影，因为遵循的是一帧激光之间是匀速运动的这个假设

# livox-LOAM

- 算法采用一种比较朴素的方式**过滤外点和动态点**:
- 先用特征直接匹配一下, 计算出相对位姿, 然后投影点云;
- 然后比较投影后的点云与地图, 去除掉太大的点, 当成外点 (outliers) 或动态点 (dynamic objects) 。
- 然后用剩下的点做继续匹配, 进行优化迭代。
- 回环检测: 类似于视觉SLAM中的词袋模型, 根据直方图进行匹配。

# LIO-mapping, LINS, Fast-LIO

- 港科大刘明老师团队的两项开源工作： **LIO-mapping, LINS**
- **LIO-Mapping**: 发表于ICRA2019。
- 文章名: **Tightly Coupled 3D Lidar Inertial Odometry and Mapping**。
- <https://github.com/hyye/lio-mapping>
- 前述lego-loam、lio-sam等工作中使用gtsam优化器，对于预积分和图优化过程，完全是黑盒处理，不利于理解细节。
- 这篇工作完全基于Vins-mono改编，基于雷达线/面特征，IMU预积分等约束放在一起进行优化；缺陷是考虑了太多的信息，算法不能实时。

# LIO-mapping, LINS, Fast-LIO

- 注意：一些工作中，把LIO-Mapping简写为LIOM，但是LIO-Mapping论文中从未将自己称为LIOM；
- 真正的LIOM应该是另一篇工作：东北大学于IROS2019发表的文章：  
A Robust Laser-Inertial Odometry and Mapping Method for Large-Scale Highway Environments——该工作使用卷积神经网络(CNN)移除动态物体，使用误差状态卡尔曼滤波器融合激光和IMU信息，解决高速环境下的运动。
- 看到论文中提到LIOM的时候，应该区分一下到底指的是哪一篇。

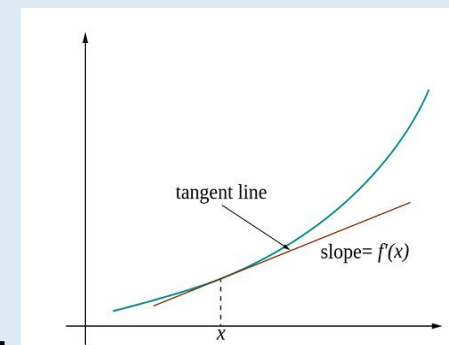
# LIO-mapping, LINS, Fast-LIO

- 最后介绍两个典型的**滤波器**方法。
- 滤波器的本质：结合**预测**和**观测**，得到精准的**后验**值。
- 什么是预测？ 只有**内在**信息：IMU, 轮式编码器 得到的运动信息
- 什么是观测？ 可以记录**外在**信息：GPS, 雷达，相机等传感器
- 后验： **融合以后的结果**。

- 之所以放在最后讲解，出于两点原因：
- 1. 涉及到一个复杂的公式推导过程
- 2. 在SLAM领域滤波方法并非主流方法，主要因为滤波器本身的套路会**限制算法对历史信息进行修正**

# LIO-mapping, LINS, Fast-LIO

- 状态方程，观测方程（根据问题来定义，一般不同）
- KF：线性（套路一般相同，预测，计算卡尔曼增益，更新）
- EKF：非线性，用一阶泰勒展开，拆成：当前时刻先验 = 一个线性化点（上一时刻后验） + 附近的线性表示，高维空间中雅可比矩阵则可理解为附近线性的“斜率”
- IEKF：非线性，迭代调整线性化点，减小误差
- 做法：先滤波得到后验，把它当作线性化点，继续迭代EKF



# LIO-mapping, LINS, Fast-LIO

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{v}_k, \mathbf{w}_k) \approx \check{\mathbf{x}}_k + \mathbf{F}_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}) + \mathbf{B}_{k-1}\mathbf{w}_k$$

$$\mathbf{y}_k = \mathbf{g}(\mathbf{x}_k, \mathbf{n}_k) \approx \check{\mathbf{y}}_k + \mathbf{G}_k(\mathbf{x}_k - \check{\mathbf{x}}_k) + \mathbf{C}_k\mathbf{n}_k$$

辅助记忆：观（**Guan**）测（**Ce**）

$$\check{\mathbf{P}}_k = \mathbf{F}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{B}_{k-1}\mathbf{Q}_k\mathbf{B}_{k-1}^T$$

$$\check{\mathbf{x}}_k = \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \mathbf{v}_k, \mathbf{0})$$

$$\mathbf{K}_k = \check{\mathbf{P}}_k\mathbf{G}_k^T (\mathbf{G}_k\check{\mathbf{P}}_k\mathbf{G}_k^T + \mathbf{C}_k\mathbf{R}_k\mathbf{C}_k^T)^{-1}$$

$$\hat{\mathbf{P}}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{G}_k)\check{\mathbf{P}}_k$$

$$\hat{\mathbf{x}}_k = \check{\mathbf{x}}_k + \mathbf{K}_k(\mathbf{y}_k - \mathbf{g}(\check{\mathbf{x}}_k, \mathbf{0}))$$

预测   卡尔曼增益   误差

固定套路：

协方差P的传递，代表对状态不确定性的传递

运动方程：代表自身的运动预测（SLAM中就是IMU）

扩展卡尔曼增益：代表对预测的**修正系数**（要考虑状态不确定性，误差不确定度）

更新协方差（状态不确定度）

真实的状态 = 预测状态 + 卡尔曼增益(**修正系数**) \* (误差)



# LIO-mapping, LINS, Fast-LIO

- **LINS:**
  - 论文名: R-LINS: A Robocentric Lidar-Inertial State Estimator for Robust and Efficient Navigation
  - 代码: <https://github.com/ChaoqinRobotics/LINS---LiDAR-inertial-SLAM>
  - 使用迭代扩展卡尔曼滤波器方式: 用IMU做状态预测, 用点-面, 点-线距离当作观测, 修正误差。
- **FAST-LIO:** 香港大学发表在RA-L 2021的工作
  - [https://github.com/hku-mars/FAST\\_LIO](https://github.com/hku-mars/FAST_LIO)
  - 使用迭代扩展卡尔曼滤波器方式, 与LINS类似, 可以支持固态激光雷达。



# Thanks for listening!

欢迎关注阿木实验室公众号！



其他问题可咨询客服（微信号jiayue199506）。