LOAM: Lidar Odometry and Mapping in Real-time

郑洁豪

摘要

本文旨在复现一种基于两轴激光雷达六维度测量数据的实时测距和建图方法。实时建图的难点在于距离测量数据是在不同的时间接收的，而运动估计中的错误会导致所得到的点云的错误配准。迄今为止，3D 地图可以通过离线批处理方法构建，通常使用闭环来校正随时间推移的漂移。该论文提出的方法实现了低漂移和低计算复杂度，不需要高精度测距或惯性测量。获得这种性能水平的关键是对同时定位和建图这一复杂问题的划分，通过两种算法同时优化大量变量。其中，一种算法在高频率低保真的情况下进行测程来估计激光雷达的速度。另一种算法以低一个数量级的频率运行，用于点云的精细匹配和配准。结合这两种算法，可以实现实时建图。

关键词：激光雷达；点云；点云匹配；实时建图；

1 引言

3D 建图仍然是一项流行的技术。使用激光雷达进行测绘是常见的，因为激光雷达可以提供高频范围测量，其中误差相对恒定，与测量的距离无关。在激光雷达的唯一运动是旋转激光束的情况下，点云的配准很简单。然而，如果激光雷达本身是移动的，精确的测绘需要连续激光测距过程中激光雷达姿态的变化信息。

解决这个问题的一种常见方法是使用独立的位置估计 (例如通过 GPS/INS) 将激光点注册到一个固定的坐标系中。另一组方法使用里程测量，如车轮编码器或视觉里程测量系统来记录激光点。由于里程计在一段时间内集成了小的增量运动，因此它必然会漂移，所以需要使用一些方法用于减少漂移 (例如使用闭环)。

在这里，我们考虑使用在 6 自由度移动的 2 轴激光雷达创建低漂移里程计地图的情况。激光雷达的一个关键优势是它对环境光线和场景中的光学纹理不敏感。随着技术发展，激光雷达已经减小了它们的尺寸和重量。激光雷达可以由穿越环境的人手持，甚至可以安装到微型飞行器上。该方法实现了低漂移和低计算复杂度，而不需要高精度测距或惯性测量。获得这种性能表现的关键是对同时定位和映射 (SLAM) 的典型复杂问题的划分，该方法通过两个算法同时优化大量变量。一种算法在高频率低精度的情况下进行测程来估计激光雷达的速度。另一种算法以低一个数量级的频率运行，用于点云的精细匹配和配准。具体而言，这两种算法分别提取位于尖锐边缘和平面表面的特征点，并将特征点与边缘线段和平面表面块进行匹配。在测程算法中，通过快速计算，找到特征点之间的对应关系。在建图算法中，通过相关的特征值和特征向量，通过检查局部点簇的几何分布来确定对应关系。

通过对原始问题的分解，首先解决一个相对简单的问题，即在线运动估计。之后，通过批量优化 (类似于迭代最接近点 (ICP) 方法) 进行映射，得到高精度的运动估计和映射。并行算法结构保证了问题实时求解的可行性。此外，由于运动估计是在更高的频率下进行的，因此映射有足够的时间来提高精度。当以较低的频率运行时，映射算法能够合并大量的特征点，并使用足够多的迭代进行收敛。

2 相关工作

激光雷达已成为机器人导航中非常有用的距离传感器。对于定位和配准，大多数应用程序使用 2D 激光雷达。当激光雷达的扫描速率相对于其外部运动较高时，扫描过程中的运动畸变往往可以忽略不计。在这种情况下，可以使用标准 ICP 方法[1]来匹配不同扫描之间的激光点云。与复现论文中方法最接近的是 Bosse 和 Zlot 提出的方法[2]，他们使用 2 轴激光雷达获取点云，通过匹配局部点簇的几何结构进行配准。此外，他们使用多个 2 轴激光雷达绘制地下矿井。这种方法包含了一个 IMU，并使用闭环来创建大型映射。获取数据集后，通过批处理优化方法恢复轨迹，该方法处理分段数据集，在分段之间添加边界约束。该方法利用激光测量单元的测量值来进行激光点的标定，并利用优化算法来修正激光测量单元的偏差。从本质上讲，Bosse 和 Zlot 的方法需要批处理来生成精确的地图，因此不适合需要实时建图。相比之下，复现论文所提出的方法[3]实时生成的地图在定性上与 Bosse 和 Zlot 的地图相似。区别在于目标论文的方法可以为自动驾驶汽车的引导提供运动估计。此外，还利用了激光雷达的扫描模式和点云分布。在测程算法和匹配算法中分别实现了特征匹配，保证了计算速度和精度。

3 本文方法

3.1 本文方法概述

Lidar 接收原始点云数据 $\hat{P}$，首先进行点云配准 Point Cloud Registration（分为特征点提取和关联）配准后点云数据为 P_k，接下来两个算法：一部分通过 Lidar Odometry 输出连续两帧之间 10Hz 的低精度姿态估计，另一部分将畸变矫正后的点云输入 Lidar mapping 并输出 1Hz 的高精度点云地图。最后估计的位姿和全局地图进行匹配优化位姿，生成基于地图的 10Hz 激光雷达姿态数据。如图 1

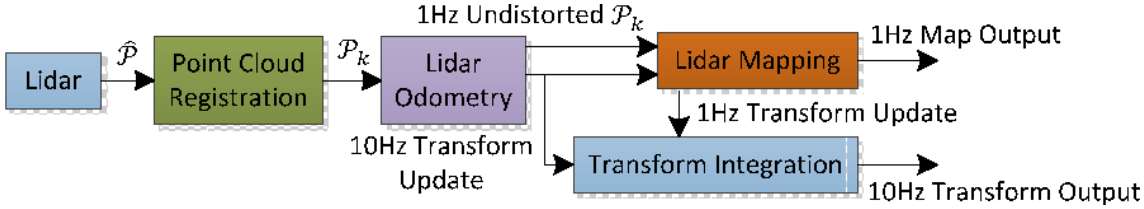

图 1: 实时建图软件系统流程图

3.2 特征点提取

对于激光点云，提取的特征点为两类：平面点（Planar Point）和边缘点（Edge Point）。两者的选取依据为曲率 c。在具体实现中，一般通过每个点周围最近五个点进行计算，得到平面光滑度，即曲率 c。

$$c = \frac{1}{|S| \cdot \left\| X^L_{(k,i)} \right\|} \left\| \sum_{j \in S, j \neq i} \left(X^L_{(k,i)} - X^L_{(k,j)} \right) \right\|$$

根据求得的 c 值进行排序，然后选择 c 值最大的特征点即边缘点，c 值最小的特征点即平面点。为了在环境中均匀分布特征点，我们将扫描分为四个相同的子区域。每个子区域最多可以提供 2 个边缘点和 4 个平面点。当点 i 的 c 值大于或小于阈值，且所选点个数不超过最大值时，点 i 才能被选为边点或平面点。

在选择特征点时，为了保证特征点分布均匀，要避免选择周围点被选中的点，或者局部平面上与激光束大致平行的点 (图 2中的 B 点)。这些点通常被认为是不可靠的。

同样，我们希望避免在被遮挡区域边界上的点。如图 2(b) 所示。点 A 是激光雷达云中的一个边缘点，因为它的连接表面 (虚线段) 被另一个物体挡住了。

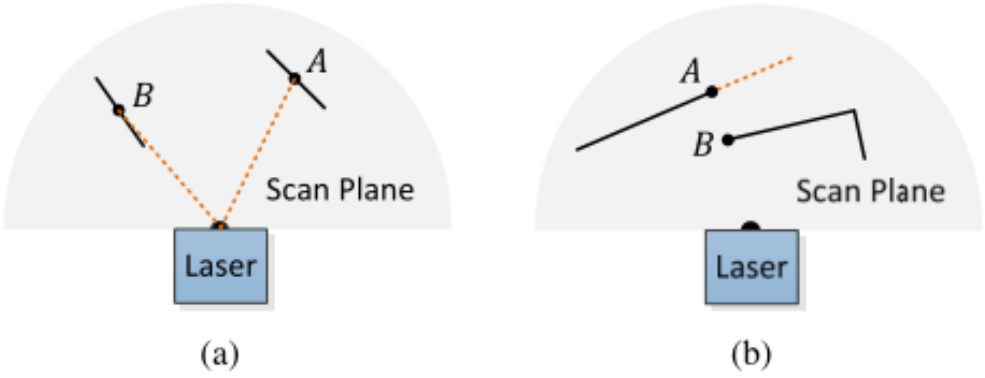

图 2: 不可靠的特征点

综上，根据曲率 c 选取特征点依照以下三个原则：

1. 每个子区域选择边缘点或平面点的数量不能超过最大值，其中边缘点 2 个，平面点 4 个；
2. 特征点周围的点不能再被选作特征点；
3. 平行于激光束或被遮挡的点不能选作特征点。

3.3 特征点匹配

设 t_k 为第 k 次扫描的开始时间，每次扫描结束时，扫描过程中获得的点云集，记为 P_k，将会重新映射到 t_{k+1} 时间戳，如图 3。将重新投影的点云标记为 $\overline{P}_k$。接下来需要找到两个点云集的映射关系。也就是特征点之间的映射关系。

设第 k 次扫描的点云集为 P_k，E_k 为边缘点集合，H_k 为平面点集合，第 k+1 次扫描的点云集为 P_{k+1}，E_{k+1} 为边缘点集合，H_{k+1} 为平面点集合。

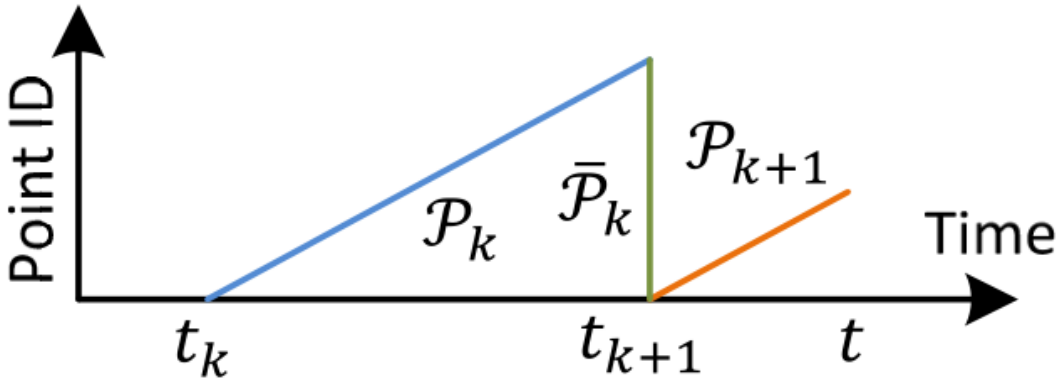

图 3: 点云映射

3.3.1 边缘点匹配

边缘点和边缘线的对应。对于当前时刻的边缘角点找到上一时刻对应的边缘线，首先通过当前时刻点 i 对应的上一时刻最近的两个点，然后判断这两个点是不是边缘点，j 和 l 必须是不同线上的点，因为一次一个线在某一段中只能有一个边缘点。

从 $\tilde{E}_{k+1}$ 选取一个点 i，在 P_k 中选取与 i 最近的点 j，以及在 P_k 中选取和点 j 相邻扫描线中最近的点 l，这样的目的是防止 i，j，k 三点共线而无法构成三角形。l，j 关系如图 4

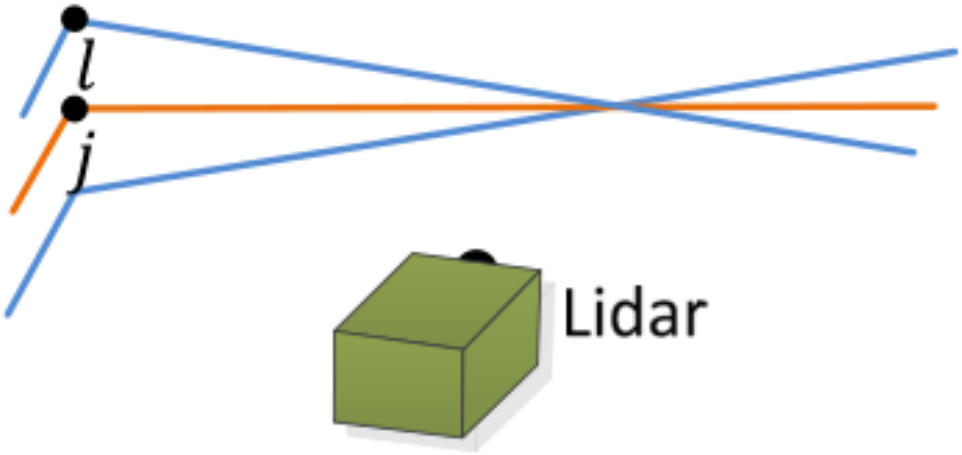

图 4: 边缘点匹配

这样我们就选取了 3 个点，姿态变换问题转化为求点 i 到 j，l 两点连线的最短距离 d_ε。

$$d_\varepsilon = \frac{\left|\left(\tilde{X}^L_{(k+1,i)} - \overline{X}^L_{(k,j)}\right) \times \left(\tilde{X}^L_{(k+1,i)} - \overline{X}^L_{(k,l)}\right)\right|}{\left(\overline{X}^L_{(k+1,i)} - \overline{X}^L_{(k,j)}\right)}$$

3.3.2 平面点匹配

与前面边缘点匹配类似，从 $\tilde{H}_{k+1}$ 中取一个点 i，从 P_k 中找到与 i 最近的点 j，然后再找 j 同一相邻的点 l 和相邻激光束上最近的点 m，这样能保证 j，l，m 非共线且在同一个平面上。j，l，m 关系如图 5。

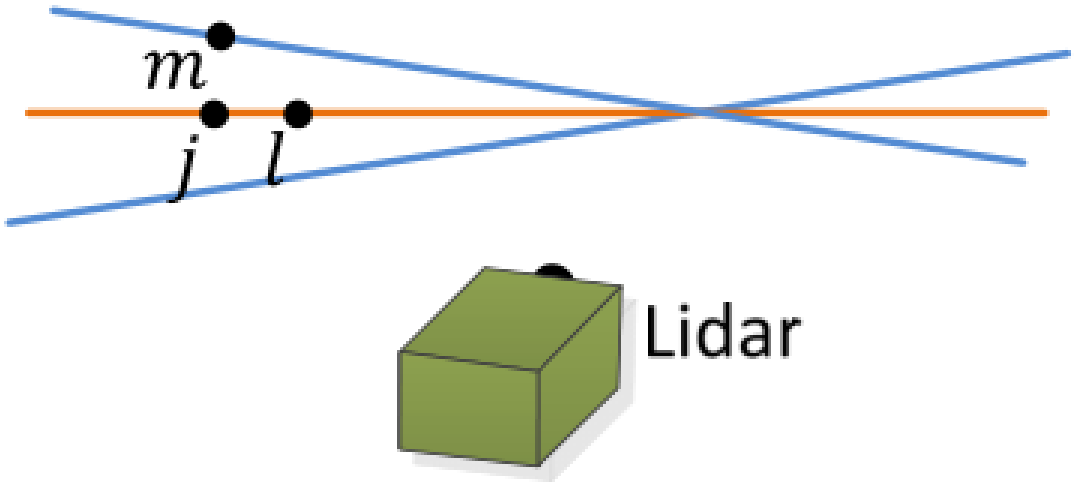

图 5: 边缘点匹配

这样姿态变换变成求点到面的最短距离 d_H。

$$d_H = \frac{\left|\begin{array}{c}\tilde{X}^L_{(k+1,i)} - \overline{X}^L_{(k,j)} \\ \left(\left(\tilde{X}^L_{(k,j)} - \overline{X}^L_{(k,l)}\right) \times \left(\tilde{X}^L_{(k,j)} - \overline{X}^L_{(k,m)}\right)\right)\end{array}\right|}{\left|\left(\tilde{X}^L_{(k,j)} - \overline{X}^L_{(k,l)}\right) \times \left(\tilde{X}^L_{(k,j)} - \overline{X}^L_{(k,m)}\right)\right|}$$

3.4 位姿估计

在扫描过程中，激光雷达的运动为恒定的角速度和线速度。这允许我们在扫描中对不同时间接收到的点进行线性插值。给定点 $i, i \in P_{k+1}$，设 t_i 为其时间戳，设 $T^L_{(k+1,i)}$ 为 $[t_{k+1}, t_i]$ 之间的位姿变换。$T^L_{(k+1,i)}$ 可以通过 T^L_{k+1} 的线性插值得到：

$$T^L_{(k+1,i)} = \frac{t_i - t_{k+1}}{t - t_{k+1}} T^L_{k+1}$$

为了解算出激光雷达运动位姿，根据上述方程得到 $T^L_{(k+1,i)}$，可建立如下位姿变换方程：

$$X^L_{(k+1,i)} = R\tilde{X}^L_{(k+1,i)} + T^L_{(k+1,i)}$$

其中，R 是旋转矩阵，可以有罗德里格斯公式计算得到：

$$R = e^{\hat{\omega}\theta} = I + \hat{\omega}\sin\theta + \hat{\omega}^2(1 - \cos\theta)$$

其中，θ 是旋转的角度，ω 是旋转方向的单位向量，$\hat{\omega}$ 是 ω 的斜对称矩阵。

根据 ε_{k+1} 边缘点与边缘线的匹配关系，可得到如下约束方程：

$$f_\varepsilon\left(X^L_{(k+1,i)}, T^L_{k+1}\right) = d_\varepsilon, i \in \varepsilon_{k+1}$$

同理，可得到平面点映射的约束方程：

$$f_H\left(X^L_{(k+1,i)}, T^L_{k+1}\right) = d_H, i \in H_{k+1}$$

合并后，可得：

$$f_H\left(X^L_{(k+1,i)}, T^L_{k+1}\right) = d_H, i \in H_{k+1}$$

然后使用 LM（列文伯格-马夸特法）进行迭代求解。

4 复现细节

4.1 与已有开源代码对比

特征点提取模块和匹配模块使用的是开放的源代码模块。考虑到实际应用场景中，一些地面点，如被风吹动的草地等，会对点云扫描进行干扰，形成噪声，故在特征点提取前对图像进行去除噪声点。部分实现代码如图 6和图 7

```
//去除地面无效点
void groundRemoval(){
    size_t lowerInd, upperInd;
    float diffX, diffY, diffZ, angle;

    for (size_t j = 0; j < Horizon_SCAN; ++j){
        for (size_t i = 0; i < groundScanInd; ++i){

            lowerInd = j + ( i )*Horizon_SCAN;
            upperInd = j + (i+1)*Horizon_SCAN;
            // 都是-1 证明是空点nanPoint
            if (fullCloud->points[lowerInd].intensity == -1 ||
                fullCloud->points[upperInd].intensity == -1){
                groundMat.at<int8_t>(i,j) = -1;
                continue;
            }
            diffX = fullCloud->points[upperInd].x - fullCloud->points[lowerInd].x;
            diffY = fullCloud->points[upperInd].y - fullCloud->points[lowerInd].y;
            diffZ = fullCloud->points[upperInd].z - fullCloud->points[lowerInd].z;

            angle = atan2(diffZ, sqrt(diffX*diffX + diffY*diffY) ) * 180 / M_PI;

            if (abs(angle - sensorMountAngle) <= 10){
                groundMat.at<int8_t>(i,j) = 1;
                groundMat.at<int8_t>(i+1,j) = 1;
            }
        }
    }
```

图 6: 部分实现代码

```
    // 找到所有点中的地面点或者距离为FLT_MAX(rangeMat的初始值)的点，并将他们标记为-1
    // rangeMat[i][j]==FLT_MAX代表无效点
    for (size_t i = 0; i < N_SCAN; ++i){
        for (size_t j = 0; j < Horizon_SCAN; ++j){
            if (groundMat.at<int8_t>(i,j) == 1 || rangeMat.at<float>(i,j) == FLT_MAX){
                labelMat.at<int>(i,j) = -1;
            }
        }
    }

    if (pubGroundCloud.getNumSubscribers() != 0){
        for (size_t i = 0; i <= groundScanInd; ++i){
            for (size_t j = 0; j < Horizon_SCAN; ++j){
                if (groundMat.at<int8_t>(i,j) == 1)
                    groundCloud->push_back(fullCloud->points[j + i*Horizon_SCAN]);
            }
        }
    }
}
```

图 7: 部分实现代码

4.2 实验环境搭建

4.2.1 Ceres Solver

Ceres Solver 是一个开源 C++ 库，用于建模和解决大型复杂的优化问题。它可以用于解决具有边界约束和一般无约束优化问题的非线性最小二乘问题。它是一个成熟，功能丰富且高性能的库。Ceres Solver 是谷歌 2010 就开始用于解决优化问题的 C++ 库，2014 年开源。在 Google 地图，Tango 项目的优化模块中均使用了 Ceres Solver。这里我们使用的是 Ceres Solver2.0.0 版本，在 ubuntu 系统中可使用 git clone https://ceres-solver.googlesource.com/ceres-solver，获取源码压缩包，解压后进行编译安装。

4.2.2 PCL

PCL（Point Cloud Library）是在吸收了前人点云相关研究基础上建立起来的大型跨平台开源 C++ 编程库，它实现了大量点云相关的通用算法和高效数据结构，涉及到点云获取、滤波、分割、配准、

检索、特征提取、识别、追踪、曲面重建、可视化等。在这里我们使用的是 1.7 版本，在 ubuntu 系统下，可使用命令行 sudo apt-get install libboost-dev 进行安装。

4.2.3 ROS

ROS 是一个适用于机器人的开源的元操作系统。它提供了操作系统应有的服务，包括硬件抽象，底层设备控制，常用函数的实现，进程间消息传递，以及包管理。它也提供用于获取、编译、编写、和跨计算机运行代码所需的工具和库函数。在 ubuntu18.04 系统下，可使用 sudo apt install ros-melodic-desktop-full 命令行进行安装。

4.3 界面分析与使用说明

rviz 是 ROS 自带的一款三维可视化平台，一方面能够实现对外部信息的图形化显示，另外还可以通过 rviz 给对象发布控制信息，从而实现对机器人的监测与控制。操作界面主要分为如下几块：

1. 中间部分为 3D 视图显示区，能够显示外部信息;
2. 上部为工具栏，包括视角控制、目标设置、地点发布等，还可以添加自定义的一些插件;
3. 左侧为显示项目，显示当前选择的插件，并且能够对插件的属性进行设置;
4. 下侧为时间显示区域，包括系统时间和 ROS 时间等;
5. 右侧为观测视角设置区域，可以设置不同的观测视角。

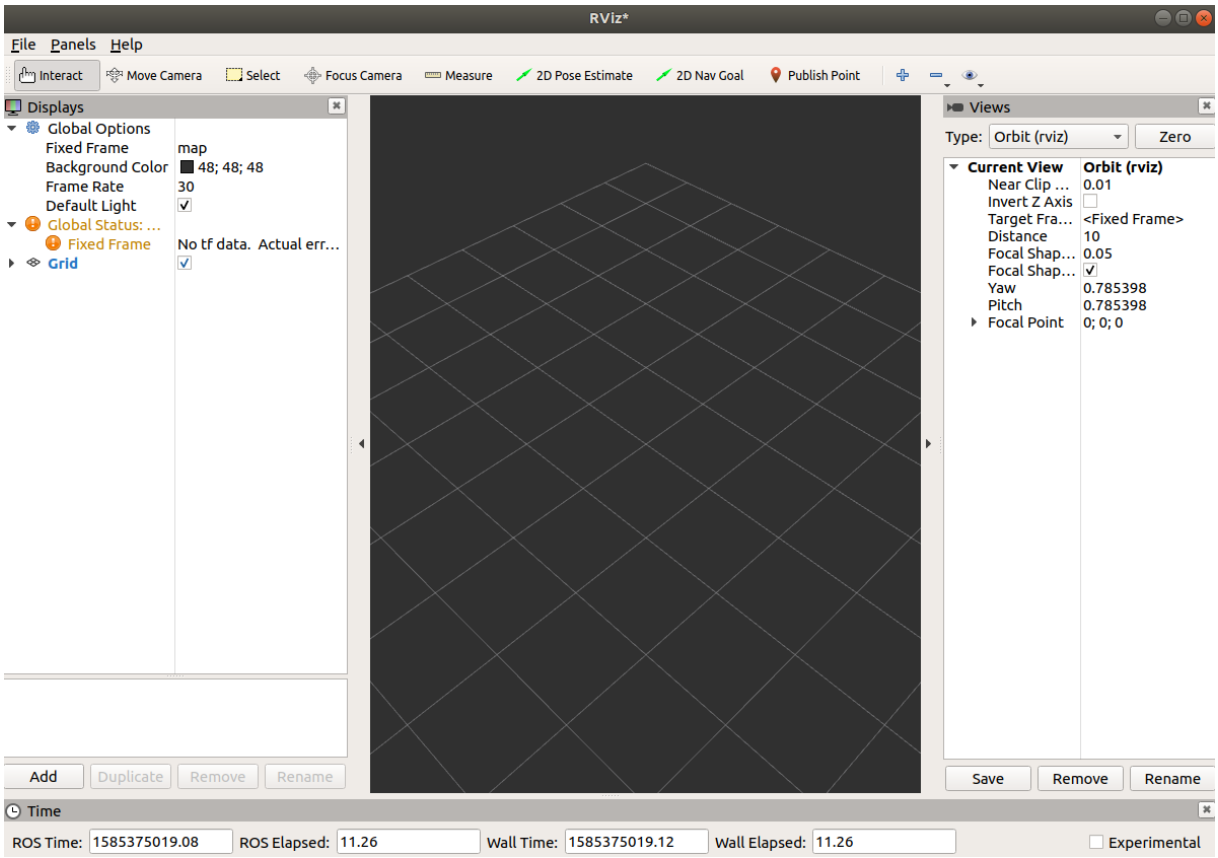

图 8: 操作界面示意

4.4 创新点

由于 VLP-16 有 16 根扫描线，其竖直方向距离为 16，因此通过判断其竖直维度的特性，可以很好的标记出地面点和非地面点。本文对地面噪声点进行进行剔除后，可以使点云数据集更好处理。

5 实验结果分析

图 9为地面噪声点剔除前后对比图，可以发现大部分散乱点都被剔除了，剩下的数据点比较干净。运行点云数据集进行建图，图 10为实际建图效果。

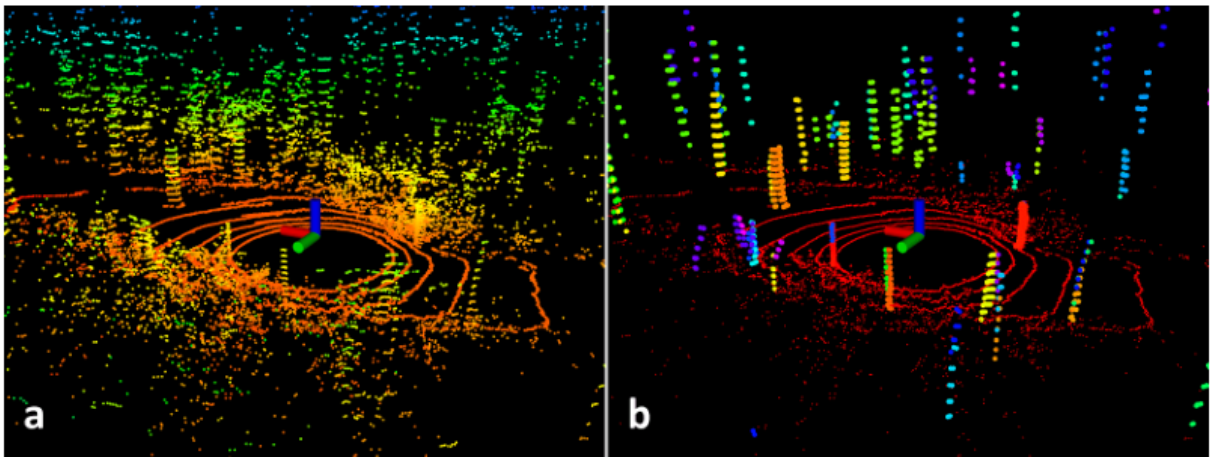

图 9: 噪声点去除效果对比。a 为原始点云图，b 为去除地面点后点云图

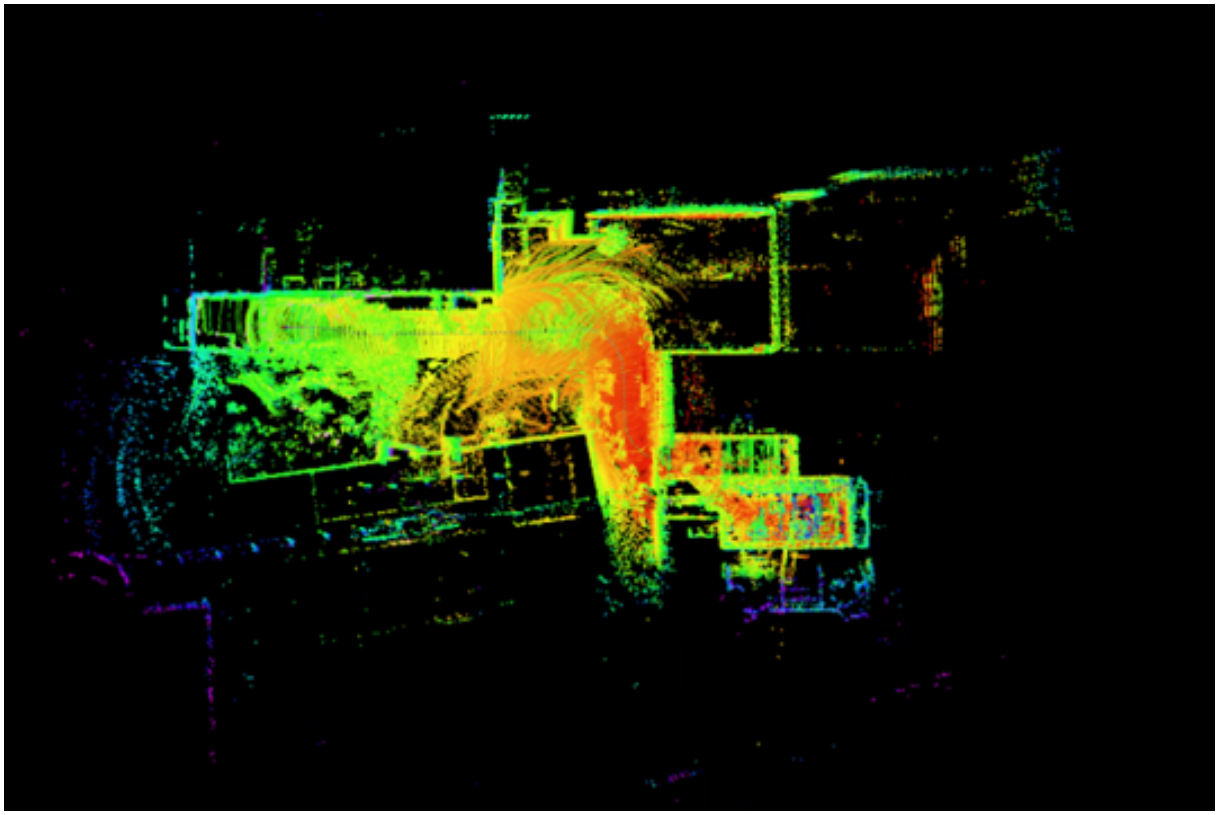

图 10: 实际建图效果

6 总结与展望

本文对 *LOAM: Lidar Odometry and Mapping in Real-time* 中提出的实时建图方法进行简要的介绍，对各个模块包括（特征点提取模块，点云匹配模块和位姿估计）进行阐述，对论文方法进行复现和创新。但从建图效果来看，并没有比原方法的效果好，其背后原因可能需要进一步的分析。未来希望能对方法进行继续优化。

参考文献

[1] POMERLEAU F, COLAS F, SIEGWART R, et al. Comparing ICP variants on real-world data sets[J]. Autonomous Robots, 2013, 34(3): 133-148.

[2] BOSSE M, ZLOT R. Continuous 3D scan-matching with a spinning 2D laser[C]//2009 IEEE International Conference on Robotics and Automation. 2009: 4312-4319.

[3] ZHANG J, SINGH S. LOAM: Lidar odometry and mapping in real-time.[C]//Robotics: Science and Systems: vol. 2: 9. 2014: 1-9.