

Mapping 3.0 系统文档

<https://idriverplus.com>

智行者定位组

摘要

本文档描述了智行者建图算法库 Mapping 3.0 的结构与功能。Mapping 3.0 是智行者云端、现场建图使用的建图算法库，具备高度自动化能力。本文档从 3.0 的框架和基本概念出发，介绍了 Mapping 3.0 的各模块的原理与具体功能实现。

目录

1 Mapping 3.0 概述	4
2 Mapping 3.0 框架与结构	4
2.1 3.0 基本概念	4
2.1.1 建图流水线	4
2.1.2 数据准备	6
2.1.3 数据预处理	6
2.1.4 前端处理	7
2.1.5 后端处理	8
2.1.6 仿真验证	8
2.1.7 CLI 与 GUI	8
2.2 3.0 代码概述	9
2.2.1 代码下载	9
2.2.2 依赖项安装	9
2.2.3 代码编译、安装	10
2.3 代码目录结构	10
3 服务器	12
3.1 服务器指令解析	12
3.2 流水线引擎的调度	12
3.2.1 服务线程	13
3.2.2 定期查询线程	14
3.3 服务器的监视、反爬虫等稳定程序	14

3.3.1	服务器守护程序	14
3.3.2	黑名单机制	15
3.3.3	输入输出文件	15
4	前端模块	15
4.1	DR 的计算	16
4.1.1	初始化	16
4.1.2	四元数与速度更新	17
4.1.3	位置更新	17
4.1.4	误差修正	17
4.2	GPS 的计算	18
4.3	关键帧的创建	18
4.3.1	筛选关键帧	18
4.3.2	构造关键帧	18
4.3.3	设定噪声	18
4.4	激光里程计的计算	19
4.4.1	Loam 匹配算法	19
4.4.2	匹配退化处理	21
5	后端模块	22
5.1	后端整体流程	22
5.2	全局轨迹优化	22
5.2.1	Pose Graph 模型	22
5.2.2	两轮优化算法	24
5.3	GPS 跳变检测	25
5.4	层级分析与水平面优化	26
5.4.1	层级检测	26
5.4.2	水平面优化	27
5.5	GPS/DR/Lidar 的 SE2 对齐	27
5.6	回环检测与回环检测约束	28
5.7	全局姿态约束	28
5.8	点云去重影	29
5.8.1	重复轨迹检测	29
5.8.2	重叠点剔除及重复点云微调	29
6	仿真、准入与准出	31
6.1	仿真	31
6.1.1	仿真过程	31
6.2	准入	33

6.3	准出	34
7	图形界面、调试工具	36
7.1	mapping-ui	36
7.1.1	mapping-ui 的使用	36
7.1.2	现场建图	36
7.2	debug-ui	37
7.2.1	debug-ui 的用途	37
7.2.2	使用 debug-ui	38
7.3	报告的生成与发送	40
8	已知的建图问题与方案	42
8.1	相关文档	42

1 Mapping 3.0 概述

Mapping 3.0（以下简称 3.0）是智行者定位组的工作成果之一，是整个地图采集、构建、制图、定位环节中的重要一环。它负责将采集到的 3D 场景数据转换为真实的轨迹与点云地图，作为下游制图、定位的输入环节。它包含基础的建图功能与配套的工具集，且支持云端或现场部署。Mapping 3.0 的主要特性如下：

1. **自动化建图。** 3.0 对多项建图算法进行了性能提升，消除 2.0 存在的建图困难问题。2.0 在许多建图能力边界附近会产生一些需要人工反复调试的场景，例如 GPS 缺失、跳变、多 Bag 包拼接、闭环检查、地图重影，等等。同时，其整体操作流程较为复杂，对地图专员要求较高，操作过程以人工调用算法为主，无法实现 24 小时在线计算。在 3.0 中，我们希望对上述问题进行消解或简化，以达到自动化建图，且使用同一套参数适配绝大多数场景的目的。
2. **24 小时云端建图。** 3.0 将使用基于建图服务器的全自动建图流水线。在默认情况下，绝大多数建图请求，从数据上传到建图结果的准出都无需人工参与。服务器将实时接管上传完毕的数据并启动流水线，而维护人员可以在建图结果完成后收到建图质量报告，决定通过与否。
3. **现场建图。** 我们将允许现场人员采集数据之后，直接在现场进行建图制图。为此 3.0 提供了一个简单易用的可视化界面，实现一键式建图操作。
4. **ODD 拓展。** 3.0 在原有建图基础上实现了 ODD 的拓展。它允许采集时起点无 GPS 信号（全程也可以没有），在一定约束条件下实现室内外混合建图。

本文档主要从代码层面描述 Mapping 3.0 的结构与功能。关于算法具体原理与实验结果，请参见 2019 年 8 月与 11 月的阶段性报告，此处不再赘述。同时，本文档主要解释工作原理而非代码接口格式，读者可以参考[代码文档](#)来查看各个类的函数接口。

本文档主要面向 Mapping 3.0 的开发人员与交接人员。

2 Mapping 3.0 框架与结构

2.1 3.0 基本概念

在介绍代码结构之前，我们首先详细描述 3.0 的基本概念，给读者一个大致的理解。

2.1.1 建图流水线

3.0 最重要的功能是自动化建图功能。在自动化建图中，每次建图任务具有固定的执行流程，抽象成一条**建图流水线**（pipeline）。每条流水线中含有各自的**流水线内容**（pipeline context）。根据任务的需求不同，流水线的内容会有一些区别。建图任务主要分为三大类：

1. **新建地图。** 目标场景还没有地图的，由采集人员采集数据，新建该场景的地图；
2. **拓展地图。** 目标场景已有部分地图，希望拓展新的地区。那么采集人员会采集新增区域的地图，然后与已有地图进行合并；
3. **更新地图。** 目标场景由于结构变换导致旧地图失效的，由采集人员采集更新区域的地图，然后覆盖旧地图的变化部分。

不同类型的任务会使用不同的流水线，不过目前绝大多数任务属于新建地图。因此，默认的流水线也就是新建地图的流水线，它含有五个内容，它们都直接继承自 Pipeline Context 类，其继承关系见图 2：

1. 数据准备 (Data Fetching)。
2. 数据预处理 (Data Preparing)。
3. 前端处理 (Lidar Odometry)。
4. 后端优化 (Backend Optimization)。
5. 仿真验证 (Verification)。

各种流水线内容具有共有基类：`Pipeline Context` 类，它定义了各步骤的统一接口（见图 1）。对于任何一个步骤，可以对它进行初始化、开始执行、生成报告等操作。由 Pipeline Engine 负责对这些内容执行操作，然后收集它们返回的结果。如果用户需要自定义新的 context，也必须实现对应的接口。

Public Types	
<pre>enum ContextStatus { PENDING = 0, WORKING, SUCCEED, FAILED, OTHER }</pre>	每一步的状态

Public Member Functions	
<code>ContextStatus Status ()</code>	获取当前步骤的状态
<code>void SetStatus (ContextStatus status)</code>	设定自身的状态
<code>std::string GetName () const</code>	获取当前步骤的名称
<code>virtual bool Init ()=0</code>	
<code>virtual bool Start ()=0</code>	
<code>virtual bool GenerateReport (std::string &report, bool verbose=true)=0</code>	
<code>virtual bool Save ()</code>	缓存中间结果
<code>virtual bool Load ()</code>	读取中间结果
<code>virtual bool FillTaskInfo (TaskInfo &info)</code>	填写任务信息

图 1：流水线内容接口

拓展任务与更新任务的流水线与默认的会稍有不同，主要是带有各自的**拓展步骤**和**更新步骤**。在流水线执行完毕后，无论建图成功与否，服务器会生成建图报告并向组内成员群体发送。报告首页的样例参见图 3。

从流程上来看，建图任务是由现场采集人员发起的。云控部的地图管理软件会生成地图对应的 ID 及任务信息，该信息目前以一个 config.yaml 文件与建图任务相关联¹。建图任务是多地并发，互不关联的，这就要求建图服务具有并发属性。在 3.0 中，由**流水线引擎** (pipeline engine) 执行每条流水线内容，在可能出问题的时候报告问题。同时，由**建图服务器** (mapping server) 实现各条流水线引擎的管理、调度工作。服务器的管理策略会决定它的运行情况。例如，在限制最大并发数策略下，不到最大并发数时，服务器会同时运行各个引擎；而超出并发数时，服务器对各引擎进行排队。而在限制最大内存的策略下，则会视各引擎的运行情况，动态分配任务数量。流水线模型示意图见图 4，详细的调度方法见第 3 章。

¹但是早期采集的历史数据不具备该格式，所以 3.0 也可以直接从数据包开始构建地图。

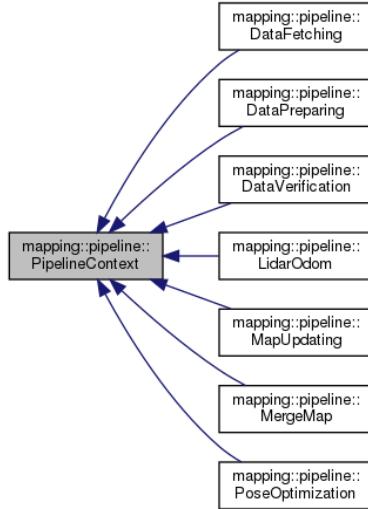


图 2: Pipeline context 继承关系：常见的五个内容都继承自 Pipeline Context 接口；

流水线引擎可以部署在云端，也可以部署在本地。目前，组内拥有一台阿里云建图服务器，24 小时响应来自 AVCP 的建图任务。此外，3.0 的现场建图功能也可以执行单次的建图任务。通过单元测试，开发人员也可以自定义流水线内容，或者单独执行流水线的一个步骤。总体而言，云端机器主顶向并发建图需求，具备高频率、大内存条件；现场建图主要面向快速演示需求，只运行单条流水线，不可并发。

以下简单说明流水线内各步骤的工作内容，具体代码实现参见后文各自的小节。

2.1.2 数据准备

数据准备阶段，流水线引擎将从预订位置下载建图采集的数据包。在云端建图时，数据包位于云控部门的 OSS (Object Storage Server) 服务器上，由云控触发器生成数据的下载链接并发送到 3.0 的服务器。在本地建图时，数据包由用户指定本地存储目录。无论以哪种方式下载数据，3.0 都会把地图数据转移到`/home/idriver/地图名称`目录下，该目录称为**地图数据目录**。

3.0 支持 zip 形式或 rar 形式的数据包。在下载完毕后，把数据解压至该目录下，然后寻找 calibration.launch 文件，并认为采集数据位于同目录下。calibration.launch 文件含有车辆传感器的标定信息，而实际采集的数据包为若干个 rosbag 数据包。如果解压失败或找不到标定参数，流水线引擎会提前结束并给出错误信息。

2.1.3 数据预处理

数据预处理阶段，优先对准备阶段提供的数据进行准入检查。数据准入检查通过后，数据预处理会解析 GPS 数据得到地图原点信息、设定传感器参数、计算 DR 轨迹、计算 GPS 轨迹并根据数据状态位和搜星数等确定其不确定性、创建关键帧、存储点云 db 文件等。同时为了考虑自动分包等情况，会针对性处理分包处 DR 初始化的问题。

beijing qinghuazhanshi 建图报告

建图软件版本 3.1.5, 联系方式: mapping@idriverplus.com

November 21, 2019

1 简报

Table 1: 综合信息

地图名称 (ID)	beijing qinghuazhanshi					
用时	1 分 21.7 秒	有效包数	2		数据量	292MB
场景类型	Outdoor	ODD	ODD 内		难度	★
长度	83.9m	推算面积	839 平米	是否成功	是	关键帧数 227
准出	失败	失败理由 (若有)				
地图数据	http://127.0.0.1:8000/beijing_qinghuazhanshi/map.db					
视频链接	http://127.0.0.1:8000/beijing_qinghuazhanshi/simulation.flv					
GMM 地图链接	http://127.0.0.1:8000/beijing_qinghuazhanshi/gmm.zip					

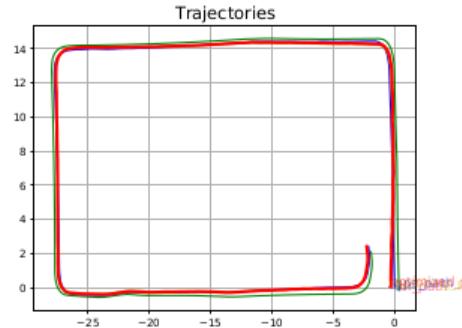


Figure 1: 轨迹

2 详细报告

data fetching: file already exist, skip downloading.

1

图 3: 中文报告示例

2.1.4 前端处理

在前端处理中，3.0 会计算激光点云里程计算法，由激光数据判断车辆的前进轨迹。激光匹配会根据数据预处理阶段计算的 DR 轨迹，进行位姿预测，然后利用匹配算法进行匹配。针对不同的激光雷达或者不同的场景特点，前端可以选择 Loam 和 NDT 两种激光点云匹配算法进行激光里程计算。大部分场景下，16 线的 velodyne 激光雷达，默认选择 Loam 方法，对于广场等容易退化的场景，建议选择 NDT 方法。目前对 velodyne 的 32 线和 64 线激光，3.0 暂时只支持 NDT 方法。前端会处理场景退化问题，并将退化区域结果传递给后端，用于后端优化时激光权重调整的依据。

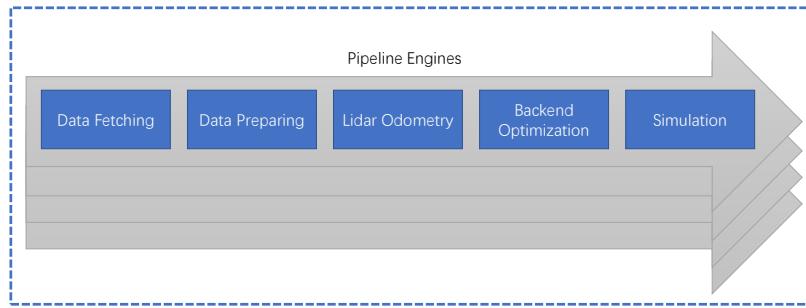


图 4: 流水线模型示意图

2.1.5 后端处理

后端将处理所有轨迹数据源，综合考虑各个传感器数据，给出优化后的轨迹。在经过前端处理后，轨迹的数据来源主要有三处：GPS 轨迹、DR 轨迹与激光轨迹。三个数据来源有各自的特点：GPS 给出全局位置姿态信息，然而受信号影响较大，可能存在大片信号不良的情况。DR 轨迹局部比较准确，但有明显的累计误差，且轮子打滑时估计不准；激光主要由环境结构影响，局部准确，但可能存在若干退化情形。后端优化算法会综合考虑各传感器的运行情况，构建最优化问题，去除不准确的信息，给出最优的估计。为了能够将多个轨迹拼接起来、减少高度上的累计误差等，后端还包括了轨迹拼接、层级检测等算法模块，处理一些建图中会遇到的实际问题。

2.1.6 仿真验证

仿真验证环节主要包括仿真测试与准出部分。按照蜗小白的采集规范，采集人员除了采集建图数据包以外，还需要采集对应的贴边数据包以适配贴边环境。在仿真阶段，会按照事先建好的点云地图，利用 z 包来验证地图能否成功定位。如果定位发生问题，验证环节将给出提示信息，并写在报告中。

2.1.7 CLI 与 GUI

3.0 具有命令行界面（command line interface）与图形界面（graphical user interfation）。命令行界面是指 3.0 编译得到的二进制程序 mapping，用户可以填写一定的子参数，完成流水线引擎的创建、管理等工作。常见的建图指令有：

1. **mapping server** 启动建图服务器。服务器参数从 config/server.yaml 中读取，包括 IP、端口、最大并发数量等配置。
2. **mapping start 地图名称地图数据链接** . 新建一个建图任务，数据将自动从链接处下载。如果当前正在运行的任务数量超过最大并发数，服务器将此任务放入待办队列。如果 config/server.yaml 中的 ip 填了远程的机器，那么实际流水线也会跑在远程机器上。
3. **mapping list**。查看流水线运行情况。可以配合 watch 命令实现自动刷新：watch -t -c bin/mapping list。同样，也可以看别人机器上的流水线数量和状态。
4. **mapping show 地图名称** 查看指定地图的详细报告，会打印此图的执行阶段和报告信息。



图 5: 3.0 建图图形界面

5. **mapping restart 地图名称开始阶段** 若建图任务失败，可以用此命令重开某个地图，并指定最初的运行阶段。开始阶段为一个 1-5 的整数，对行流水线的五个阶段。1 为数据下载，2 为数据准备，3 为 Lidar Odom，4 为后端优化，5 为仿真。
6. **mapping close 关闭服务器**。注意该指令不会马上关闭服务器，而是等所有任务执行完毕后才会实际关闭。

命令行界面允许用户以比较灵活的方式操作 3.0。这些命令可以由用户手工输入，也可以用脚本自动执行，或者通过网络发送。这给自动测试、云端自动触发带来了很多便利之处。

在现场建图任务中，对于不熟悉 linux 操作的建图人员，3.0 也具备一键式的图形界面，如图 5 所示。图形界面仅需指定地图 config.yaml 所在位置即可，其他部分完全自动化进行，无需手工输入任何信息。图形界面程序随 3.0 一起发布，名称为 **mapping-ui**。

后文将详细介绍各步骤的工作原理与实现细节。

2.2 3.0 代码概述

2.2.1 代码下载

Mapping 3.0 代码目录位于内网 gitlab 的 [/map/mapping3.0.git](#) 下，通过内网即可下载源代码。3.0 主要为 C++ 代码和部分 python 运行脚本，支持环境为 Ubuntu 16.04 与 18.04。

2.2.2 依赖项安装

3.0 需要环境中以下软件包：

- ROS，与 Ubuntu 16.04 或 18.04 对应的 ROS。
- libboost-all-dev，Boost 库。
- Eigen3，代数与矩阵运算。
- OpenCV，图像运算与生成。

- libpcl-dev, pcl-tools, 点云库。
- texlive-full, pdf 报告生成。
- libsuitesparse-dev, libcholmod3, 稀疏代数运算库。
- ffmpeg, 仿真视频生成。
- libgoogle-glog-dev, libgflags-dev, LOG 和 flags。
- libgtest-dev, 单元测试。
- libsqlite3-dev, 数据库界面。
- python-matplotlib, 绘图。
- sendemail, 邮件发送。

大部分小型包都可以通过 apt-get 安装。用户也可以执行 scripts/install_dependency.sh 完成小型依赖项的安装。如果仅使用 3.0 而不编译，则只需从智行者持续集成系统下载编译好的软件包即可，下载地址见 3.0 的使用手册。

2.2.3 代码编译、安装

3.0 是一个 cmake 工程，可以直接使用 cmake 的默认方式编译。或者，执行 3.0 代码根目录下的 build.sh 可以一键编译它的三方库依赖项与 3.0 工程本身。编译完成后，bin/ 目录下会生成可执行文件，其中重要的文件有：

- bin/mapping, 3.0 的命令行界面。
- bin/mapping-ui, 3.0 的图形界面。
- bin/debug-ui, 3.0 的修图工具。

此外还会生成一些单元测试文件，对于普通用户可以忽略。

在编译之后，执行 install.sh 脚本，可以把 3.0 的编译产出打包。打包之后，代码目录下生成一个 install_mapping.tar.gz 软件包，可用于软件的分发。只要目标机器上具备相同的环境，则可以直接使用编译好的软件包运行建图服务。

以上编译、安装脚本已部署于智行者持续集成系统中。3.0 的每个发布版本均会由 gitlab 自动执行编译和打包，用户可在持续集成系统中直接获取 3.0 各版本软件包。开发者需要发布某个版本时，只需在 git commit 时提交信息为'release for 3.0.0'，即可发布对应版本的软件。

2.3 代码目录结构

3.0 的目录结构由以下几个文件夹组成：

- **bin**. 编译后的二进制文件。
- **cmake**. cmake 的各种依赖项。
- **config**. 配置文件目录。主要配置文件为 config/server.yaml 和 config/mapping.yaml。前者配置服务器的内容，后者是建图任务的默认参数模板。
- **doc**. 代码文档、阶段性实验报告、使用手册、准入准出标准等文档。
- **scripts**. 一些 python 脚本，用于画图、自动执行程序、守护线程等。
- **src**. 源代码。
- **test**. 单元测试。

- **thirdparty**. 源代码形式的三方依赖库。

其中，源代码目录进一步分解如下。

- **app**. 可执行程序的主要内容。包括文件: mapping_client.h, mapping_client.cc, mapping_command.cc, mapping_server.h, mapping_server.cc。这些文件用于构建 3.0 的服务器与命令行界面。
- **common**. 3.0 的基础数学结构、共享数据等，包括文件: angle.h, common.h, pointcloud_constants.h, vector3.h, circular_buffer.h, common_struct.h, point_types.h, version.h, key_frame.h, quaternion.h, common.cc, twist.h。common/msg/下含有车端 ROS message 的具体定义。
- **core**. 核心算法代码。主要包含：基本矩阵运算、插值、数学常量、四元数处理、卡尔曼滤波。core 下层文件目录还包括：
 - **msf**. 多传感器融合算法。主要为惯性导航算法及对应的数据结构文件。
 - **layer_detector**. 层级分析算法。
 - **lidar_matching**. 激光 SLAM 算法。主要包含了 Loam 和 NDT 两种算法。
 - **initialization**. 定位初始化的实现。
 - **resolution_matching**. 闭环检测和多分辨率匹配算法。主要用于轨迹间和轨迹内的闭环检测模块。
- **io**. 数据输入输出，主要包括：
 - db_io.h, db_io.cc。与 SQL 数据库文件相关的读写程序。
 - oss_io.h, oss_io.cc。与阿里云 OSS 相关的上传下载功能。
 - xml_io.h, xml_io.cc。与 XML 相关的读写，主要用于 calibration.launch 文件的解析。
 - yaml_io.h, yaml_io.cc。YAML 文件的读写，用于读写参数配置文件和地图 config.yaml 文件。
- **pipeline**. 流水线引擎和各个流水线内容的定义。它的下层目录还包括：
 - **backend**. 后端优化的具体实现。包括优化算法的定义、多 bag 间回环检测算法，等等；
 - **frontend**. 前端激光里程计的定义与实现。包括 Loam、NDT 两种算法的选择调用、匹配和退化结果的返回。
 - **checking**. 准入与准出算法的实现。
 - **preprocessing**. 数据包预处理相关代码，包括传感器消息格式统一和轨迹预解算，以及关键帧的构造等。
 - **refine_pose**. 点云去重影的实现。
 - **simulation**. 仿真的实现。
 - **merge_map**. 拓展地图的实现。包括地图基本信息的读取、拓展地图和目标地图的拼接算法等。
 - **updating_maps**. 地图更新的实现。
- **tools**. 各种中间项与小工具。包括：
 - **bag_convert**. 数据包格式转换。
 - **gmm_map_generator**. gmm 地图生成。
 - **pointcloud_convert**. 点云驱动。包括 packets 向 pointcloud 的转换、点云运动补偿。
 - **bag_loader**. 数据包加载。将 bag 数据加载为系统所需数据类型。
 - **intensity_calib**. 点云反射率标定。
 - **save_file**. 读写关键帧和地图文件的接口。文件类型包括 pcd、txt 和 db 等。
 - **pcd_to_png**. 俯视投影 pcd 为 png 文件的实现。

- **file_filter**. 文件搜索。
- **perception_interface**. 点云高度提取接口。用于调用 thirdparty 中的提取点云高度算法。
- **transform**. 数据类型转换和 GPS 坐标转换。
- **ui**. 图形界面相关实现。包括:
 - **debug-ui**. 修图软件的实现。
 - **mapping-ui**. 建图软件的实现。

3 服务器

3.0 服务器主要为自定义协议的 Socket 服务。此外，为了方便报告与成果数据的下载，3.0 也支持使用 Simple Python HTTP 或 Apache HTTP 服务来管理文件下载。服务器部分代码位于 `src/app/mapping_server` 中，分为服务端 `server` 与客户端 `client`。不管是服务端还是客户端，都使用统一的命令行接口 `mapping_command`。命令行接口通过解析其参数来获得指令是服务端还是客户端的信息。

3.1 服务器指令解析

服务器的命令行界面由 `mapping_command.cc` 编译而成，用户使用的 mapping 二进制文件即它编译的结果。该文件会尝试解析用户命令的子参数合法性，若合法时，调用对应的功能；非法时，直接退出。因此，通过 mapping 二进制发送出去的指令必定是合法的。目前能够接收的指令有：

1. `server` 启动服务器；
2. `list` 列出运行任务；
3. `start` 新建一个任务，需要后两个参数为地图名和链接；
4. `show` 显示任务详情，需要后一个参数为地图名；
5. `close` 退出服务器；
6. `restart` 重启一个任务，需要后两个参数为地图名和重启步骤；
7. `version` 打印软件版本；
8. `merge` 合并两个地图任务，需要后两个参数为输入地图名与合并地图名；

在这些指令中，`mapping server` 指令会新建一个服务器端，`version` 指令会直接打印头文件中的版本信息（位于 `src/common/version.h` 中），其他指令都会启动一个客户端，然后通过客户端向服务器端发送对应指令和参数。客户端逻辑较为简单，以字符串的形式向服务器端转发所有参数信息即可，而服务器端的流程则复杂一些。

3.2 流水线引擎的调度

如果用户使用 `mapping server` 创建服务器，那么服务器会首先从 `config/server.yaml` 中读取服务器端信息（见图 6）。该文件定义了服务器要绑定的 IP 与端口，通常情况下，使用这两者即可实现服务器与客户端的访问。而在阿里云上，由于阿里云定义了内外网不同的 IP，内网绑定 0.0.0.0 而外网绑定指定 IP，因此需要打开 `server.use_internal_ip` 项。

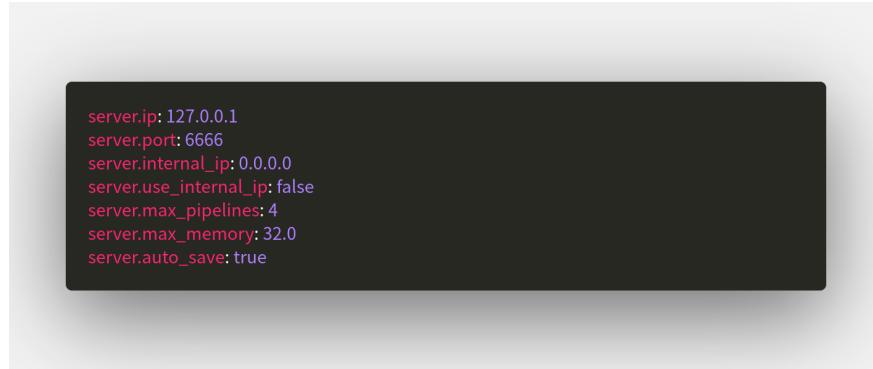


图 6: server.yaml 中的内容

服务器的流水线分配有两种策略：最大并发数策略与最大内存策略。在设定最大并发数策略时，服务器会限定内部同时运行的流水线数量；而在最大内存策略中，服务器会估计每条流水线的使用内存，动态分配并发数量。不过，由于新建流水线时只有一个任务名和数据链接，无从得知流水线对应的地图大小，基于最大内存的策略会使用预定的内存容量（默认一条流水线占据 4GB 内存），可能与实际用量有出入。因此，目前在云端使用的还是最大流水线数量策略。

服务器内部主要由两个线程组成：服务线程与定期查询线程。

3.2.1 服务线程

服务线程监听指定端口（默认为 6666）的数据输入。如果有计算机从该端口进行 TCP 连接，则会开始接收客户端发过来的指令。如果该指令由 mapping 3.0 的命令行界面调用，那么指令本身已经通过客户端检查，必定是合法的。此时，服务线程根据指令内容，执行相应操作，然后以字符串形式返回一个回答内容给客户端。对应函数见 mapping_server.cc 的 RunCommand 函数。

服务端通过不同的流水线引擎容器来管理各种各样的流水线（见图 7）。主要的容器有：正在运行的流水线，运行结束已经归档的流水线，未开始执行的流水线。所有的流水线有内部分配的 ID，与用户指定的地图名称构成映射关系。在接收到 start 指令时，服务线程会查看正在运行的流水线是否超过最大限制。如果是，则把新的流水线放到 TODO 类中，否则直接开始执行新的流水线。

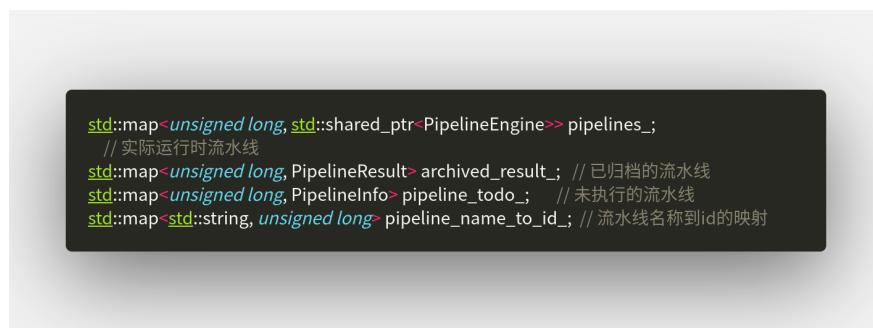


图 7: 服务器中的各种流水线

3.2.2 定期查询线程

除了服务线程之间，服务器还有一个每 5 秒运行一次的定期查询线程。该线程完成的工作为：

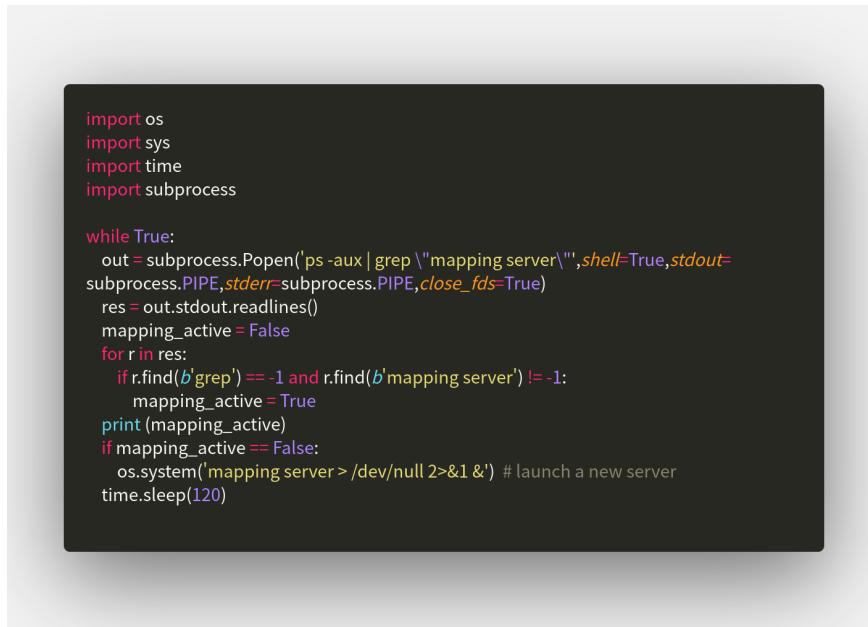
1. 遍历正在运行的流水线。如果该流水线已经完成，不轮成功与否，都放入归档流水线中；
2. 如果有流水线已经结束且待办流水线不空，那么把待办的第一个流水线放入运行流水线中，并启动它；
3. 备份所有流水线状态至 happy 文件中。

如果 server.yaml 中的备份功能被打开，那么服务器会维护一个名为 happy 的文本文件，该文件存储了当前各流水线的状态，包括已归档的和未执行的。如果服务器因为各种原因退出，在下次启动时就可以从 happy 文件中恢复自身的运行状态。

3.3 服务器的监视、反爬虫等稳定程序

3.3.1 服务器守护程序

服务器可能由于各种原因而退出，例如：内存不足、硬盘不足、配置文件出错、数据包格式出错、机器断网断电等等。为了防止服务器退出导致建图服务中断的情况，通常使用守护程序来启动服务器。守护程序的脚本为 scripts/server_daemon.py（图8）。守护程序定期从系统中获取 mapping server 是否在运行的状态。如果发现建图服务器未在运行，则尝试重新启动系统服务器。如果自动备份文件 happy 存在，那么服务器将恢复到退出前的状态。



```
import os
import sys
import time
import subprocess

while True:
    out = subprocess.Popen('ps -aux | grep \'mapping server\'', shell=True, stdout=
subprocess.PIPE, stderr=subprocess.PIPE, close_fds=True)
    res = out.stdout.readlines()
    mapping_active = False
    for r in res:
        if r.find(b'grep') == -1 and r.find(b'mapping server') != -1:
            mapping_active = True
    print(mapping_active)
    if mapping_active == False:
        os.system('mapping server > /dev/null 2>&1 &') # launch a new server
    time.sleep(120)
```

图 8：守护程序脚本

注意守护线程可以解决偶发的内存不足导致的建图服务器退出问题，但如果由于系统 bug 使得建图失败，那么即使重启，也会在同样的地方失败，导致系统进入死循环。此时需要人工干预进行故障的排查。

3.3.2 黑名单机制

由于建图服务器是对外开放端口的，除了建图客户端给服务器发送建图指令之外，也可能收到其他软件的 TCP 连接。建图服务器与云控端就是直接使用地址和端口来发送数据的。不过，大部分这样的连接来自外网的爬虫程序，它们使用 HTTP 协议爬取网页信息。但是建图服务器使用的是自定义的 socket 协议，故 HTTP 协议会在指令检查时被排除。同时，我们会把这些爬虫的 IP 记进黑名单，被记入黑名单的地址会被屏蔽。如果使用建图的命令行界面进行连接，因为建图的客户端已经做了指令检查，不必担心发送错误指令而被记入黑名单的问题。

建图服务器的维护人员可以通过/tmp/mapping.INFO 查看运行日期。可以从日志中检查黑名单的 IP 信息。

3.3.3 输入输出文件

服务器默认把地图数据存储在/**home/idriver/data/地图名称**下方。在新建地图时，AVCP 发送的数据链接中会含有数据包的名称信息。建图服务器通过解析此名称，获取数据包是 zip 还是 rar 格式的信息。然后，服务器将数据下载至本地并重命名为 data.rar 或 data.zip，然后将它解压至同名目录。

如果用户发送的数据链接不是一个合法的 URL（不能直接通过 wget 下载），那么服务器会在地图数据目录里寻找是否有现成的 data.rar 或 data.zip。若有，就会认为这是一个正确的数据包，从而跳过下载步骤，直接解压进行建图。由于这个逻辑的存在，用户在本地使用建图服务时，可以手动把数据包拷贝到地图目录并重命名，然后直接启动建图流水线，这种方式会跳过数据下载环节，可以用于开发者调试。事实上，现场建图也使用了同样的逻辑。

在建图完成后，mapping 3.0 把结果文件存储于/**home/idriver/results/地图名**目录下。在云端，我们配置了 apache 服务器，用户可通过 http 下载所有位于/**home/idriver/results/**处的文件（需要登入，图 9）。在本地使用时，mapping 3.0 会在成果目录下启动 simple python http 服务器，使用户可以点击报告中的数据链接来下载结果数据。当然，也可以直接到成果目录中拷贝数据。

4 前端模块

前端模块主要目标是通过 DR 算法、GPS 坐标转换算法和 3D 激光里程计算法解算各传感器数据，创建关键帧，计算出关键帧的 DR 位姿、GPS 位姿和激光匹配的位姿，以及各个位姿对应的噪声。前端模块计算完的关键帧的各个位姿信息最终用于后端模块的融合优化中，计算出最终的优化位姿。代码实现包括 data_preparing 和 front 两个 pipeline。data_preparing 中实现 DR 的位姿计算和 GPS 的坐标转换，根据 DR 轨迹筛选关键帧，同时进行时间对齐找出对应的 GPS 位姿以及激光点云数据，作为关键帧的成员。包含了 DR 位姿、GPS 位姿、激光点云数据的关键帧是激光里程计的数据输入源。front 主要是根据输入的关键帧的点云数据进行激光里程计的计算，计算完后，关键帧将会被赋予匹配位姿和相应的噪声。包含了时间对齐的 DR 位姿、GPS 位姿、匹配位姿以及各位姿相应噪声的关键帧将再作为后端优化的输入源，进行后续的位姿优化计算。

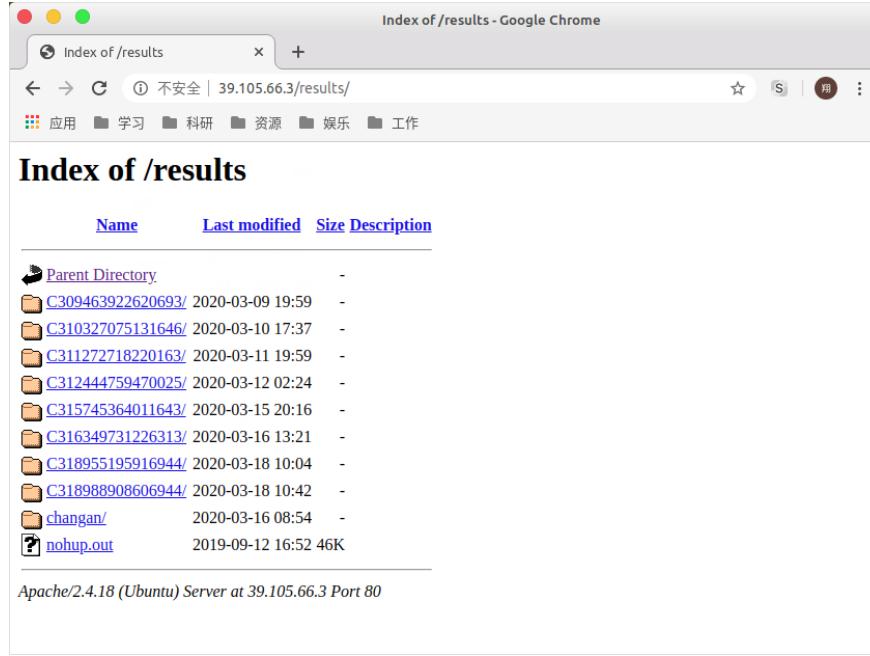


图 9: 云端的 Apache 服务器提供所有运行结果文件的下载

4.1 DR 的计算

3.0 中的 Dead Reckoning 计算 (简称 DR) 的主要目标是通过卡尔曼滤波器解算车辆搭载的轮速计和 IMU 传感器数据, 计算车辆行驶的 3D 轨迹。代码实现在 src/core/msf 中, 对应 dead_reckoning.cc, 被 preprocessing 中的 trajectory_builder/path_solver 调用, 用于筛选关键帧, 并定义自身为关键帧的 DR 位姿, 作为后续激光里程计匹配计算时的预测以及后端优化时顶点间的边。以下是对 DR 状态估计和递推的数学描述。

4.1.1 初始化

本文档中所有的计算都是基于东-北-天坐标系, IMU 输出三轴角速度和三轴加速度数据, 当车辆处于静止状态时, 根据 IMU 输出的加速度可以计算得到车辆当前的水平姿态角, 为了获得比较准确地初始水平姿态, 一般取一段时间内的加速度均值 (记为 $\mathbf{f}_m^b = [f_{mx}, f_{my}, f_{mz}]$) 来计算水平姿态的初始值, 如下式所示:

$$\begin{cases} \text{roll} = \arctan(-f_{mx}, f_{mz}) \\ \text{pitch} = \arcsin(f_{my}/g) \\ \text{yaw} = 0 \end{cases} \quad (1)$$

其中 g 为当地重力加速度, roll 为车辆横滚角, pitch 为俯仰角, yaw 为航向角, 初始时置零。

由上式可获得车辆当前的水平姿态和航向角, 然后采用下式完成四元数的初始化。

$$\begin{cases} q_0 = \cos\left(\frac{\text{yaw}}{2}\right) \cos\left(\frac{\text{pitch}}{2}\right) \cos\left(\frac{\text{roll}}{2}\right) + \sin\left(\frac{\text{yaw}}{2}\right) \sin\left(\frac{\text{pitch}}{2}\right) \sin\left(\frac{\text{roll}}{2}\right) \\ q_1 = \cos\left(\frac{\text{yaw}}{2}\right) \sin\left(\frac{\text{pitch}}{2}\right) \cos\left(\frac{\text{roll}}{2}\right) + \sin\left(\frac{\text{yaw}}{2}\right) \cos\left(\frac{\text{pitch}}{2}\right) \sin\left(\frac{\text{roll}}{2}\right) \\ q_2 = \cos\left(\frac{\text{yaw}}{2}\right) \cos\left(\frac{\text{pitch}}{2}\right) \sin\left(\frac{\text{roll}}{2}\right) - \sin\left(\frac{\text{yaw}}{2}\right) \sin\left(\frac{\text{pitch}}{2}\right) \cos\left(\frac{\text{roll}}{2}\right) \\ q_3 = \cos\left(\frac{\text{yaw}}{2}\right) \sin\left(\frac{\text{pitch}}{2}\right) \sin\left(\frac{\text{roll}}{2}\right) - \sin\left(\frac{\text{yaw}}{2}\right) \cos\left(\frac{\text{pitch}}{2}\right) \cos\left(\frac{\text{roll}}{2}\right) \end{cases} \quad (2)$$

假设初始化时车辆处于静止状态，采用下式完成位置和速度初始化：

$$\begin{cases} v_e = v_n = v_u = 0 \\ p_e = p_n = p_u = 0 \end{cases} \quad (3)$$

4.1.2 四元数与速度更新

由上式可得四元数的初始值。记三轴陀螺仪输出的角速度 $\omega^b = [\omega_x^b, \omega_y^b, \omega_z^b]$ ，记 IMU 采集时间间隔为 ΔT ，那么可由下式计算一个采样时间内的角度增量：

$$\Delta\theta = \omega^b \Delta T = [\theta_x, \theta_y, \theta_z] \quad (4)$$

四元数更新由下式描述：

$$\begin{cases} \Delta\theta = \sqrt{\theta_x^2 + \theta_y^2 + \theta_z^2} \\ \mathbf{Q}_v = \left[\cos \frac{\Delta\theta}{2}, \frac{\sin(\Delta\theta/2)}{\Delta\theta} \theta_x, \frac{\sin(\Delta\theta/2)}{\Delta\theta} \theta_y, \frac{\sin(\Delta\theta/2)}{\Delta\theta} \theta_z \right] \\ \mathbf{Q}_k = \mathbf{Q}_{k-1} \otimes \mathbf{Q}_v \end{cases} \quad (5)$$

根据上一时刻的四元数 \mathbf{Q}_{k-1} 和上一时刻到当前时刻的角度增量，更新得到当前时刻的四元数 \mathbf{Q}_k ，再根据当前时刻的四元数获得当前时刻的姿态矩阵 \mathbf{C}_n^b ，记三轴加速度计输出的加速度 $\mathbf{f}^b = [f_x^b, f_y^b, f_z^b]$ ，由下式完成速度更新：

$$\mathbf{V}_k = \mathbf{V}_{k-1} + (\mathbf{C}_n^b \mathbf{f}^b + \mathbf{g}) \quad (6)$$

其中 \mathbf{V}_{k-1} 为上一时刻速度矢量， \mathbf{V}_k 为当前时刻速度矢量， \mathbf{g} 为当地重力加速度矢量。

4.1.3 位置更新

采用轮速计输出的车速 v^b 和当前时刻的姿态矩阵 \mathbf{C}_n^b ，完成位置更新：

$$\mathbf{P}_k = \mathbf{P}_{k-1} + \mathbf{C}_n^b [0, v^b, 0]^T \Delta T \quad (7)$$

4.1.4 误差修正

选取状态量 $\mathbf{X} = [\phi_e, \phi_n, \phi_u, \delta v_e, \delta v_n, \delta v_u]$ ，其中 $[\phi_e, \phi_n, \phi_u]$ 为失准角， $[\delta v_e, \delta v_n, \delta v_u]$ 为东北天向速度误差，建立误差模型：

$$\begin{cases} \dot{\mathbf{X}} = \mathbf{A}\mathbf{X} + \mathbf{G}\mathbf{W} \\ \mathbf{Z} = \mathbf{H}\mathbf{X} + \mathbf{V} \end{cases} \quad (8)$$

上式中：

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\ (\mathbf{C}_n^b \mathbf{f}^b) \times & \mathbf{0}_{3 \times 3} \end{bmatrix}, \mathbf{G} = \begin{bmatrix} -\mathbf{C}_n^b & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{3 \times 3} & \mathbf{C}_n^b \end{bmatrix},$$

\mathbf{W} 为 IMU 噪声矩阵， $\mathbf{H} = [\mathbf{0}_{3 \times 3}, I_{3 \times 3}]$ ， \mathbf{V} 为观测噪声，即轮速计速度测量噪声。

建立如上所示的误差模型，以轮速计输出的当前车速 v^b 作为当前速度观测，与式 (6) 更新得到当前车速，做差：

$$\delta\mathbf{V} = \mathbf{V}_k - \mathbf{C}_n^b [0, v^b, 0]^T \quad (9)$$

根据(8),(9)进行卡尔曼滤波完成误差量的估计,失准角用来修正式(5)更新得到的四元数,速度误差用来修正式(6)更新得到的速度,再由修正后的四元数根据式(10)得到当前准确的姿态角。

$$\begin{cases} \text{roll} = \arctan(-2(q_1q_3 - q_0q_2), q_0^2 - q_1^2 - q_2^2 + q_3^2) \\ \text{pitch} = \arcsin(2(q_0q_1 + q_2q_3)) \\ \text{yaw} = \arctan(2(q_1q_2 - q_0q_3), q_0^2 - q_1^2 + q_2^2 - q_3^2) \end{cases} \quad (10)$$

4.2 GPS 的计算

GPS 计算主要实现经纬度坐标向 UTM 坐标的转换,并根据天线安装角度,计算出车辆的航向角。代码实现在 src/tools/transform 中,被 data_preparing 中的 traject_builder/path_solver 调用,对应着 gps_transform.cc,在计算之前需要通过解析 calibration.launch 文件中得到 GPS 天线安装角度等标定参数。

在建图任务起初阶段,data_preparing 会优先根据建图数据计算起始处的 UTM 坐标,并将其设定为即将建的地图的原点信息,后续计算的 DR 位姿和激光里程计都会相对于该原点进行轨迹解算。

4.3 关键帧的创建

在计算完 DR 轨迹和 GPS 轨迹后,3.0 根据 DR 轨迹筛选出关键帧,并通过时间对齐的方式找出与关键帧的 DR 位姿时间接近的 GPS 位姿和激光点云数据。关键帧作为激光里程计的输入源,利用关键帧的点云数据进行激光匹配计算,并在激光里程计的计算结束后,赋予关键帧匹配位姿。关键帧具体数据类型见 src/common/key_frame.h 文件。创建关键帧主要包括筛选关键帧、构造关键帧和设定噪声。

4.3.1 筛选关键帧

筛选关键帧根据 DR 轨迹的时间和空间距离进行判断,3.0 里依据以下条件进行关键帧的筛选:

1. 水平面每隔 0.5 米;
2. 水平面每隔 5 度;
3. 时间每隔 30 秒;

只要满足以上三个筛选条件的任意一条,将会开始构造关键帧。

4.3.2 构造关键帧

构造关键帧主要是进行 DR 位姿、GPS 位姿和激光点云数据的时间同步。根据当前 DR 位姿查找时间上最近的点云数据,将点云时间戳作为关键帧的时间戳,由于 DR 位姿频率比点云高很多,所以当前 DR 位姿和已确定的点云数据不一定是时间上最接近的,因此需要根据点云时间戳,反向查找是时间上最接近的 DR 位姿和 GPS 位姿。3.0 没有进行插值计算来实现精准的时间对齐,由于 DR 频率较高, GPS 本身作为后端的先验信息,针对蜗系列产品,精确度可以在接受范围内。

4.3.3 设定噪声

构造的关键帧需要给 DR 位姿和 GPS 位姿和待计算的激光匹配位姿设定合理的噪声(3.0 里在创建关键帧阶段就设定激光匹配位姿的噪声,假设其为一个固定的噪声,在面对退化问题时,会额外维护一个退化标

志数组, 用于解决位姿优化时匹配失效的问题), 用于后端优化。在融合各种传感器输入时, 必须十分注意它们的原始数据噪声大小, 否则可能使融合失效。按照现实中的传感器精度, 例如单点 GPS, 在信号有效时, 取:

$$\Sigma_{GPS} = \begin{bmatrix} \text{diag}(\underbrace{0.15, 0.15, 0.15}_{\text{平移部分}}, \underbrace{0.08, 0.08, 0.08}_{\text{角度部分}}) \end{bmatrix}^2 \quad (11)$$

表明单点 GPS 标准差的平移为 15cm, 旋转为 5°, 这是符合现实传感器的。若输入时判断 GPS 信号不好, 则会以倍率扩大该协方差矩阵。

对于激光匹配和 DR, 它们在局部比较准确, 目前取:

$$\begin{aligned} \Sigma_{DR} &= [\text{diag}(0.05, 0.05, 0.05, 0.008, 0.008, 0.008)]^2 \\ \Sigma_{LIDAR} &= [\text{diag}(0.05, 0.05, 0.05, 0.008, 0.008, 0.008)]^2 \end{aligned} \quad (12)$$

即它们在两个关键帧之间的标准差为 5cm 和 0.5 度。这是一个比较小的噪声, 它保证我们的优化结果局部与 Lidar 和 DR 更加相似。

相比 DR 和激光匹配位姿, GPS 受场景影响更大, 因此 GPS 位姿的噪声不能简单设定为一个固定值, 而是需要根据数据状态位、搜星数和精度因子以及航向标志位进行噪声计算。在 GPS 的噪声计算中, 当 GPS 位姿与其前后的 4 个 GPS 位姿对应的传感器数据中状态位为 4(有效)、搜星数大于 10、精度因子小于 3、基线长度大于 0.01 且航向标志位有效同时满足时, 该 GPS 位姿才会被认为是“GOOD”, 当以上条件其余都满足而航向角失效时, 该 GPS 位姿会被认为是“GENERAL”, 剩下的则全部为“BAD”。被判断为“GOOD”的 GPS 位置噪声设定为 0.15, 航向噪声 3×10^{-9} , 代码实现 data_preparing 中的 traject_builder 中。

4.4 激光里程计的计算

激光里程计主要进行关键帧的点云连续匹配, 计算匹配后的位姿, 并根据算法精度设定匹配位姿的噪声。目前 3.0 支持 Loam 和 NDT 两种匹配算法, 蜗系列产品默认为 16 线 velodyne 的 Loam 匹配算法。在特征较少的空旷地上, Loam 会发生较为严重的匹配退化问题, 此时可以选择 NDT 进行匹配, NDT 的表现会好于 Loam。对于 32 线的 velodyne, 目前 3.0 只支持 NDT 匹配算法。因此本文该部分主要介绍一下 Loam 匹配算法。激光里程计作为单独的 pipeline, 其代码在 src/pipeline/front 中, 对应着 lidar_odom.cc, Loam 和 NDT 匹配算法在 src/core/lidar_matching 中, 通过修改 config/mapping.yaml 的 matching_type 参数, 可以选择激光里程计所使用的匹配方法。

4.4.1 Loam 匹配算法

Loam 通过对原始点云进行点云分割、线面特征提取、帧间匹配和局部地图优化以及闭环检测五个步骤实现激光里程计的计算。通过提取特征再做匹配的方法加快整体计算的速度; 通过对原始点云的分割, 去除杂草和树叶等特征噪声以及地面平面的提取, 提高特征的精度从而提升激光里程计的整体精度; 通过实时的闭环检测和闭环校正, 尽可能的去除全局的畸变, 在为后端优化提供全局一致性较好的激光匹配位姿轨迹。代码实现在 src/core/lidar_matching 中, 点云分割部分对应 range_image_projection.cc, 特征提取对应 feature_extractor.cc, 帧间匹配对应 feature_tracker.cc, 局部地图优化和闭环检测对应 local_mapper.cc。

点云分割 先将原始点云投影成距离图像，距离图像每一行代表了多线激光每一条线的激光点云，每一列代表了 0 到 360 度之间水平旋转对应的分辨率角度，图像中的每个像素代表了原始点云中的每个点，其值是对应的点到传感器中心的距离；然后对扫描到地面的点云进行地面拟合，筛选出属于地面点云，将拟合后的点云打上“地面”标签；通过基于图像的分割方法对分离地面后剩下的点云进行基于距离图像的聚类，每簇点云中点的个数必须大于一定数量，并且被打上唯一的标签，树叶等微小物体在点云分割的过程中将被剔除，分割后的点云基本表示了较大的物体，如树干、建筑物和地面等，作为后续特征提取的输入。

线面特征提取 为了均匀地提取特征点，先将距离图像水平上划分成 6 个子图像，每个子图像对应了多线激光旋转 60 度区域内的原始点云；再根据点云分割时各个点打标签的情况，剔除没有打标签的杂草等噪音，对每个子图像中打有效标签的点选取其左右各 5 个点进行曲率计算；设定曲率阈值进行子图像中特征候选点的提取，将曲率大于设定阈值且标签不是地面的点提取为边缘点候选点，将曲率小于设定阈值的点提取为平面点候选点；从边缘点的候选点中选择出子图像每行曲率最大的 2 个点作为子图像的边缘点特征点，从平面点的候选点中选择子图像每行曲率最小且标签为地面的 4 个点作为子图像的平面点特征点。

帧间匹配 两步匹配求解。首先依据点云分割后每个特征点的标签进行两帧特征点云之间的对应关系的搜索，当前帧中平面点特征的标签为地面，所以其只需在前一帧标签也是地面的特征点云中进行最近邻搜索，找到距离最近的特征点组成平面点特征匹配对，而对于边缘点特征，其只需在前一帧标签不是地面的特征点云中进行最近邻搜索，找到距离最近的特征点组成边缘点特征匹配对，通过点云分割时打的标签确定匹配对的方式减少了误匹配的概率，准确率得以提升；然后利用两步匹配算法求解两帧点云之间 6 自由度的位姿变换，第一步优先计算平面点特征匹配对之间的相似性，利用列文伯格-马夸尔的非线性优化算法求解两帧点云之间 6 自由度位姿变换中的 z 轴方向的位移、横滚角和俯仰角这 3 个自由度的状态变量，由于平面点特征描述的是地面信息，所以其在要求解的 3 个自由度变量上具有较强的约束；第二步再计算边缘点特征匹配对之间的相似性，再次利用列文伯格-马夸尔优化算法求解位姿变换中另外 3 个自由度变量，即航向角、x 轴和 y 轴方向的位移。两步匹配优化求解通过分割后更加准确的特征匹配对将 6 个自由度一次性优化问题拆分成两个 3 自由度优化问题，在保证精度的同时减少了计算时间。

局部地图优化 当前帧位置附近的 N 帧特征点云作为局部地图，当前帧的特征点云和局部地图进行特征匹配对搜索，然后同样利用列文伯格-马夸尔优化算法进行求解，更新状态变量，计算方法同帧间匹配。

闭环检测 当前帧会对历史轨迹所有关键帧进行闭环检测，闭环检测依据是已经计算的匹配位姿，当前帧如果与历史轨迹中的某一帧关键帧距离较近，则将该关键帧作为闭环候选帧。计算当前帧和候选帧与其前后的一些关键帧构成的局部地图进行 ICP 匹配。如果匹配结果分数较高，则认为两者之间存在闭环。闭环约束加入全局优化器中，进行全局优化，校正闭环。由于激光里程计本身存在一定的累计误差，尤其是当场景规模较大，长时间的连续匹配会发生全局畸变，尤其是俯仰角的误差，会导致长时间累计误差后高度上无法正常闭环。因此激光里程计中的闭环检测适用与小范围的闭环校正。对于全局大范围的闭环问题，由后端模块基于多分辨率地图的闭环算法进行解决。该闭环模块的可以通过调整 config/mapping.yaml 中 feature_matching_params 里的几个闭环参数来适应场景的特殊情况，目前默认的参数可以适应大部分场景。

4.4.2 匹配退化处理

Loam 正常匹配算法可以应对特征较为丰富的场景，当遇到广场和走廊这种特征较少的特殊场景时，Loam 将会发生匹配退化问题。激光里程计针对这个问题进行了专门的处理。在容易发生匹配退化的场景下，点云分割去除噪声点的处理会导致有效特征点较少，影响帧间匹配的稳定性和精确度，因此该环节可以跳过，所以用户可以通过设定 config/mapping.yaml 中 feature_matching_params 里的 segment_enabled 标志位禁用该模块。对于一些特别空旷的场景，禁用该点云分割模块依然无法提取有效的特征，此时应对匹配退化的方法为：激光里程计自身不去解决退化问题，而是有效的识别退化区域，并将识别的结果传递给后端，后端位姿优化在面对广场区域时采取“局部更相信 DR”的策略，由后端统一解决。识别匹配退化的方法如下：

LOAM 的帧间匹配是一个状态估计问题，同样也是一个最小二乘问题，采用高斯牛顿法去求解这个问题的本质就是求解增量方程：

$$\mathbf{H}\Delta\mathbf{x} = \mathbf{g} \quad (13)$$

其中 $\mathbf{H} = \mathbf{J}^T \mathbf{J}$, \mathbf{J} 为雅可比矩阵。为了求解增量方程，通常需要 \mathbf{H} 是可逆的，也就是满秩的，假设 \mathbf{H} 是 n 维的，则 $\text{rank}(\mathbf{H}) = n$ ，此时：

$$\Delta\mathbf{x} = \mathbf{H}^{-1} \mathbf{g} \quad (14)$$

然而在一些情况下， $\mathbf{J}^T \mathbf{J}$ 是半正定的， \mathbf{H} 不满秩，那么增量方程便会存在一系列通解：

$$\Delta\mathbf{x} = \underbrace{\mathbf{H}^{-1} \mathbf{g}}_{\text{特解}} + \underbrace{k_1 \Delta\mathbf{x}_1 + k_2 \Delta\mathbf{x}_2 + \dots + k_r \Delta\mathbf{x}_r}_{\text{通解}}. \quad (15)$$

公式 (15) 中的 $\Delta\mathbf{x}_1, \Delta\mathbf{x}_2, \dots, \Delta\mathbf{x}_r$ 属于增量方程的零空间，此时 $\text{rank}(\mathbf{H}) = n - r$ ，在这种情况下， \mathbf{H} 为奇异矩阵，增量的稳定性较差，导致算法不收敛，此时系统便发生了匹配退化。

通过判断 \mathbf{H} 是否满秩即可判断系统是否退化，在建图的后端优化里便可针对性的处理退化场景。然而实际应用中，在一些容易使得系统退化的场景中（如广场区域），由于各种精度误差的存在， \mathbf{H} 往往都是满秩的，需要通过对 \mathbf{H} 进行特征值分解或者对 \mathbf{J} 进行奇异值分解的手段来判断系统是否退化。对 \mathbf{H} 进行特征值分解如下所示：

$$\begin{aligned} \mathbf{H} &= \mathbf{V} \Lambda \mathbf{V}^T \\ \Lambda &= [\text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)], \quad \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \end{aligned} \quad (16)$$

当 \mathbf{H} 最小特征值 λ_n 与最大特征值 λ_1 的比值非常小（实际取 10^{-4} 量级）的时候，可认为 \mathbf{H} 接近奇异矩阵，此时即使数值上是满秩的 \mathbf{H} 同样会使得整个系统是病态的，增量的稳定性较差。因此对 \mathbf{H} 特征值分解，通过判断最小特征值是否非常小判别系统是否退化的有效手段，对 \mathbf{J} 进行奇异值分解同理。代码实现 src/core/lidar_matching/feature_matching/local_mapper.cc 里，这里判断当最 \mathbf{H} 最小特征值小于 100 时则此次匹配发生退化问题。

判断为退化的激光匹配位姿，将会被赋予非常大的噪声，后端优化便更相信 DR 递推结果，使得退化区域的轨迹整体较为平滑，不受激光里程计发生匹配退化问题的影响。

5 后端模块

后端模块负责综合各传感器信息，给出轨迹的最优估计。在实现当中，为了让建图算法具有高度自动化运行能力，我们通常使用统一的默认参数来适配复杂的现场环境，这就导致后端优化在流程上显得比较复杂。

5.1 后端整体流程

后端整体流程实现于 **Pose Optimization** 类中，我们简要描述其流程：

1. 初始化过程中，从配置文件读入各种优化参数；
2. 在有多个包时，使用轨迹拼接算法计算包与包之间的拼接关系，并划分轨迹 ID；
3. 若打开了层级分析，则进行层级分析；
4. 进行第一轮优化，存储第一轮中间结果；
5. 进行自动闭环检测；
6. 进行第二轮优化，存储第二轮中间结果；
7. 执行点云去重影算法；
8. 输出结果文件；

其中层级分析、去重影等算法将在后文描述。

5.2 全局轨迹优化

5.2.1 Pose Graph 模型

后端轨迹优化是一个状态估计问题，它从若干种数据来源中计算最优的轨迹。数据来源主要有以下七类：

1. **GPS 定位**：由前端处理之后的 GPS 信息，同时根据 GPS 的状态，已设定其噪声矩阵；
2. **IMU+ 轮速积分**，在 Mapping 3.0 中称为 Dead Reckoning (DR)；DR 的噪声是固定的，但是由于标定、打滑等因素可能导致估计不准确；
3. **激光前后帧匹配 (Lidar Odom)**：给出相对运动约束，但可能退化；
4. **回环检测给出的相对位姿**：可能出现错误的匹配；
5. **层级分析的高度约束**：将同层的轨迹约束在同一个水平面；
6. **全局姿态约束**：车辆通常是水平向上放置的，此类约束将车辆的 Z 轴与世界的 Z 轴对齐；
7. **修图软件给出的人为约束**：在修图软件中，可以人工调整两个关键帧之间的相对位姿；

其中前三种观测量是由真实传感器给出的，而后四种约束则来自人为的环境假设或者算法的估计值。在前三者中，仅有 GPS 能够提供绝对定位信息（相对于真实世界），其他两种定位来源均为相对定位（相对于上个时刻或之前某个时刻）。所以原则上说，若全部区域没有 GPS，那我们无法给出带物理世界绝对位置信息的地图；若长时间缺少 GPS 信号，那么在缺失 GPS 的区域无法保证绝对位置的准确性。

首先来描述前面三种测量值。我们以图优化形式描述上述问题，那么整个问题可抽象为一个 Pose Graph [1, 2]。以 $\mathbf{x}_k \in \text{SE}(3)$ 表示第 k 时刻位姿，那么各种观测表达为：

1. GPS 位姿： $\mathbf{x}_{k,\text{GPS}}$ ；
2. DR 相对位姿（从 $k-1$ 至 k ）： $\mathbf{x}_{k-1,k,\text{DR}}$ ；
3. Lidar 相对位姿（从 $k-1$ 至 k ）： $\mathbf{x}_{k-1,k,\text{LIDAR}}$ 。

每个数据对应真实位姿的一次观测：

$$\begin{aligned}\mathbf{x}_{k,\text{GPS}} &= \mathbf{x}_k \oplus \mathbf{n}_{\text{GPS}}, \\ \mathbf{x}_{k-1,k,\text{DR}} &= \mathbf{x}_{k-1}^{-1} \mathbf{x}_k \oplus \mathbf{n}_{\text{DR}} \\ \mathbf{x}_{k-1,k,\text{LIDAR}} &= \mathbf{x}_{k-1}^{-1} \mathbf{x}_k \oplus \mathbf{n}_{\text{LIDAR}}\end{aligned}\tag{17}$$

其中 \oplus 指 SE(3) 上广义加法， \mathbf{n} 为各种噪声，假设服从高斯分布：

$$\begin{aligned}\mathbf{n}_{\text{GPS}} &\sim \mathcal{N}(0, \Sigma_{\text{GPS}}), \\ \mathbf{n}_{\text{DR}} &\sim \mathcal{N}(0, \Sigma_{\text{DR}}), \\ \mathbf{n}_{\text{LIDAR}} &\sim \mathcal{N}(0, \Sigma_{\text{LIDAR}})\end{aligned}\tag{18}$$

那么整个 Pose Graph 问题由最小二乘问题

$$\{\mathbf{x}\}^* = \arg \min_{\mathbf{x}_k, k=1, \dots, n} \sum \{\rho(\mathbf{e}_{\text{GPS}}^T \Sigma_{\text{GPS}}^{-1} \mathbf{e}_{\text{GPS}}) + \rho(\mathbf{e}_{\text{DR}}^T \Sigma_{\text{DR}}^{-1} \mathbf{e}_{\text{DR}}) + \rho(\mathbf{e}_{\text{LIDAR}}^T \Sigma_{\text{LIDAR}}^{-1} \mathbf{e}_{\text{LIDAR}})\}\tag{19}$$

描述，其中 ρ 表示鲁棒核函数（实用中使用 Cauchy 核，理论证明 [3] 加权 Cauchy 核等价于自适应估计，因此 Cauchy 核在噪声大小不确定时比较好用）[4]，不同下标的 \mathbf{e} 表示不同种类的误差，整个优化问题为各类误差在高斯协方差意义下的 Σ 范数。

Pose Graph 在理论上是简单易用的，但实际处理中存在几个重要的问题：

- 如何构建合适的 Pose Graph？虽然各类观测的数学定义是明确的，但实际上对于 Lidar 和 DR 的测量，仅添加 $k-1$ 到 k 时刻的测量信息并不足以保证局部轨迹形状不发生畸变；而局部轨迹形状一旦和测量不符，点云拼接时即可能造成重影。因此，除了 GPS 信息仍然为对单次定位的测量之外，我们会在 Pose Graph 中添加 $k-l, k-l+1, \dots, k-1$ 至 k 时刻的相对测量，其中 l 为参与局部测量的关键帧个数。在实用中，取 $l_{\text{LIDAR}} = 5, l_{\text{DR}} = 2$ 。这两个参数在 mapping.yaml 中的 optimization_params.dr_continuous_num 和 lidar_continuous_num 中配置。
- 如何制定优化的流程？为了让建图结果更加完善，我们并不是简单地构建一个 Pose Graph 并求解，而是设置了一种两轮优化的方式，在后文描述。
- 如何确定各种测量的协方差矩阵？协方差矩阵描述了单次测量的精度信息。在融合各种传感器输入时，必须十分注意它们的原始数据噪声大小，否则可能使融合失效。按照现实中的传感器精度，例如单点 GPS，在信号有效时，我们取：

$$\Sigma_{\text{GPS}} = \left[\underbrace{\text{diag}(0.15, 0.15, 0.15)}_{\text{平移部分}}, \underbrace{0.08, 0.08, 0.08}_{\text{角度部分}} \right]^2\tag{20}$$

表明单点 GPS 标准差的平移为 15cm，旋转为 5°，这是符合现实传感器的。若输入时判断 GPS 信号不好，则会以倍率扩大该协方差矩阵。

对于激光匹配和 DR，它们在局部比较准确，目前取：

$$\begin{aligned}\Sigma_{\text{DR}} &= [\text{diag}(0.05, 0.05, 0.05, 0.008, 0.008, 0.008)]^2 \\ \Sigma_{\text{LIDAR}} &= [\text{diag}(0.05, 0.05, 0.05, 0.008, 0.008, 0.008)]^2\end{aligned}\tag{21}$$

即它们在两个关键帧之间的标准差为 5cm 和 0.5 度。这是一个比较小的噪声，它保证我们的优化结果局部与 Lidar 和 DR 更加相似。

- 如何判断测量信息是否有效？根据前面所述，GPS 即使在输入时判定为有效，实际也可能出现各种形式的跳变。因此，在优化时，我们检查各类传感器数据的代价函数是否超出阈值 δ ，若超过，则会将该测量信息在第 2 遍优化时移除。实用中的取值为²：

$$\delta_{\text{GPS}} = 0.535, \delta_{\text{DR}} = 1.566, \delta_{\text{LIDAR}} = 1.437. \quad (22)$$

针对 GPS 部分，如果超过 90% 的数据无效，我们会增大 δ_{GPS} 至两倍，以适应 GPS 不好的场景，这种增大最多运行三次，即 GPS 外点的阈值最多为初始值的 8 倍。

总体而言，上述的算法保证我们在信息源存在部分异常值时，算法仍能给出良好的轨迹估计。

5.2.2 两轮优化算法

由于我们使用基于位置的回环检测算法（而非基于相似度），回环检测需要依赖一个大致正确的轨迹估计，于是我们提出了两轮优化的方式。算法 1 描述了后端优化的整体流程。

Algorithm 1 后端优化算法流程

- 1: 第一轮
 - 2: 第 1 遍优化
 - 3: 对激光、DR、GPS 轨迹进行 SE(2) 对齐；
 - 4: 添加所有传感器数据，添加 Robust Kernel；
 - 5: 得到优化结果；
 - 6: 判断 outlier，调整 outlier 阈值，去除 outlier，去除 Robust Kernel
 - 7: 第 2 遍优化
 - 8: 得到仅使用 inlier，无 Robust Kernel 的结果；
 - 9: 第二轮
 - 10: 根据第一轮优化结果计算回环检测
 - 11: 带入回环检测信息，执行 2-7 步骤
 - 12: 输出最终结果
-

相比于单次优化，它的主要改进在于：第 1 轮优化后我们已得到一个大致准确的轨迹，它们可能来自多个包，所以直接拼接可能会有明显的重影。但是，第 1 轮轨迹可以为回环检测提供初始位姿，使我们可以在限定区域内搜索有效的回环信息。这些回环信息加入之后，第 2 轮优化会使得全局轨迹更加准确，拼接处重影更少。

两轮轨迹优化会分别把中间结果存储于地图目录的 map_first_optimization 文件夹与 map_second_optimization 文件夹，开发人员可以直接查看这两个文件夹来确认优化是否正确。轨迹优化的详细信息会输出的报告中，供用户查看（图 10）。

²这些取值是根据实验中的误差值分布确定的，例如某张 GPS 信号良好的区域，误差的 90% 分位约在 0.29，而最大值为 0.657。这种分布可以辅助我们确定各类测量的外点阈值。

```

GPS inliers/outlier: 468/15, in/out=0.968944
GPS: num: 468, Ave: 0.085701, mean: 0.061520, 0.1: 0.016159, 0.9: 0.199823, max: 0.438219, th: 0.535000
Matching: num: 2400, Ave: 0.013145, mean: 0.009235, 0.1: 0.002662, 0.9: 0.027209, max: 0.121879, th:
1.437000
DR: num: 482, Ave: 0.372727, mean: 0.192768, 0.1: 0.056185, 0.9: 0.877512, max: 7.302232, th: 1.566000
Loop inlier/outlier: 0/0
pose optimization: ok
Pose_Optimization time usage: 81.069 seconds.

```

图 10: 报告中输出的优化信息：GPS 内点外点与占比、各种优化项误差总和与统计分位、回环检测结果的内外点占比、优化运行时间

5.3 GPS 跳变检测

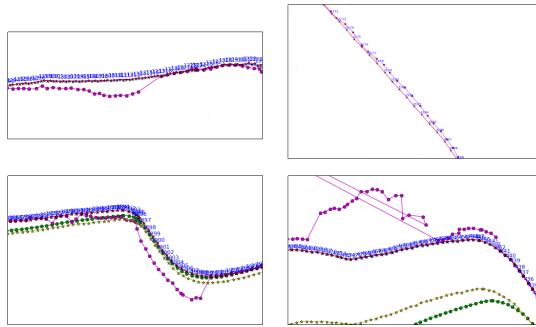


图 11: GPS 跳变的实际例子：紫色线：GPS 轨迹；红色线 + 蓝色标记：优化轨迹

GPS 信号跳变是 2.0 难以处理大规模地图的一个主要原因。由于 GPS 多径效应的存在，我们拿到的 GPS 信号值可能与真实情况不符，但 GPS 接收机认为信号有高置信度。详细来说，GPS 跳变可以分为以下四类：

- GPS 跳变。指 GPS 信号短时间内跳至别处，一段时间内跳回正确值的情况。
- GPS 阶跃。指 GPS 跳至错误值并持续较长时间，表现为阶跃特征。
- GPS 演变。指 GPS 缓慢变化至一个错误的值。
- GPS 失效。指 GPS 信号变化至零或极远处值。

以上四种情况在实际当中均可能出现。由于实际行驶轨迹的不可预测性，GPS 信号跳变也可能是各种情况的混合，而非属于单一类别。以图 11 为例³，左上角 GPS 信号出现了渐变，然后跳回正确位置；右上角 GPS 出现随机跳动；左下角 GPS 跳变至错误值，然后返回；右下角 GPS 跳至极远处无效值，然后返回。由此可见，对 GPS 跳变进行归类，是一种人为的，仅为了方便讨论的做法，实际对每次跳变进行分类是比较困难的。

为了保证 GPS 异常值能够被正确检测到，mapping 3.0 采取了以下策略：

1. 在前端处理时，区分对待各种情况的 GPS。对信号测量不好的情况给予较大的噪声。那样，测量值的信息矩阵就会较小，对整体优化的影响也较小；
2. 同理，对 GPS 的位置与角度的噪声是分别判定的，存在某些时刻 GPS 位置较好但角度不好的情况；同样，在位置当中，高度的不确定性较大，因此高度值的信息较小；
3. 在两轮优化中，动态调整 GPS 的 outlier 判定阈值；

³本例以优化轨迹，即红色轨迹/蓝色数字作为参考。

4. 在强制采图中，存在起始区域 GPS 不好，但后续又有 GPS 较好的路段。这种情况下，前端给出的起点无效，那么利用 SE(2) 对齐 GPS 轨迹与激光轨迹；

这些策略保证了 GPS 测量能够正确作用于轨迹优化，同时对于 GPS 不好的强制采图数据也有正确的结果。实际情况中，GPS 信号不佳是非常常见的（图 12 为广东的例子），因此对 GPS 的处理也必须仔细小心。

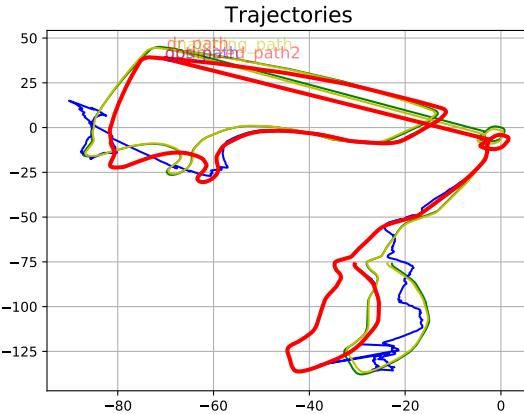


图 12：实际数据中的 GPS 轨迹（蓝色线）与优化轨迹（红色线）

5.4 层级分析与水平面优化

层级分析以及水平面优化设计之初的主要目的是为了支持“多层结构”场景的建图任务。在实际的运营情况下，目前大部分建图任务的场景主要是室外坡度和高度混合的同层结构、室内平面同层结构。因此 3.0 暂时不应对多层结构场景的建图任务。因此层级分析当前的主要目的就是正确检测出“同一高度”的区域，并将该区域内的所有关键帧认为是同一水平面的，将水平面高度约束加入后端，可以有效的解决“GPS 数据不可用同时激光里程存在俯仰角的累计误差从而导致地图高度发生畸变”的问题。该功能主要对室内场景和 GPS 大规模不可用的室外场景起到较好的平面约束。对于规模较大的室外场景，尤其是存在一定不可忽视的坡度的场景，由于 DR 位姿的长时间累计误差和外参标定的误差，加之不可忽略的缓坡难以准确识别，会导致层级检测准确度降低，给到后端的水平面约束是错误的，以至于地图高度上发生“断层”的问题。因此，目前 3.0 默认关闭层级检测功能，当室内场景和 GPS 大部分不可用的室外场景建图需要高度约束解决高度畸变问题时，可在 config/mapping.yaml 中修改 optimization_params/with_layer_detection 参数，打开层级检测和水平面优化。

5.4.1 层级检测

层级检测当前主要目的是检测出同一水平面区域内的关键帧，因此根据 DR 轨迹可以根据其俯仰角、横滚角和高度变化识别出平面区域对应的关键帧。代码实现在 src/pipeline/backend/trajectory_connector.cc 中。具体算法如下：

- 将轨迹在水平面上切分成 3 米长的线段，计算每条线段的高度、俯仰角和横滚角的变化；

- 通过以下公式计算每条线段的变化率：

$$\text{ratio} = |\Delta\text{roll}| + |\Delta\text{pitch}| + 5 * |\Delta\text{height}|, \quad (23)$$

- 将低于设定阈值的变化率对应的线段作为同一水平面的候选线段；
- 根据空间连续和时间连续的特点，剔除不属于同一水平面的候选线段，并填充空间隔断的相邻候选线段，计算出最终同一水平面的关键帧。

5.4.2 水平面优化

在分析完层次结构后，为保持每层的地面水平，我们将统计每层的平均高度，然后约束该层内关键帧的高度值。设关键帧 k 的位姿为 $\mathbf{T}_k \in \text{SE}(3)$ ，其平均高度约束为 h_k ，那么施加在上方的约束为：

$$e_{\text{height}} = \mathbf{t}_{k,3} - h_k, \quad (24)$$

其中 $\mathbf{t}_{k,3}$ 为 \mathbf{T} 平移部分取第 3 个维度（即高度）。由于 g2o 对位姿使用 $\text{SE}(3)$ 上的右乘更新，设更新方程为 $\mathbf{T}_k \leftarrow \mathbf{T}_k \Delta \mathbf{T}_k$ ，其中 $\Delta \mathbf{T}_k \in \text{SE}(3)$ ，于是该约束项的雅可比矩阵为：

$$\frac{\partial e_{\text{height}}}{\partial \Delta \mathbf{T}_k} = [\underbrace{\mathbf{0}}_{\Delta \mathbf{R} \text{部分}}, \underbrace{\mathbf{R}_{k,3}^T}_{\Delta \mathbf{t} \text{部分}}], \quad (25)$$

其中 $\mathbf{R}_{k,3}^T$ 为 \mathbf{T} 的旋转矩阵第 3 行的元素。

施加该约束后，同层的位姿将变得更加水平，高度统一。并且，由于高度被水面约束，某些 GPS 高度不准确的地方将出现较大的 GPS 误差，导致 GPS 信息被认作异常值（但此时水平位置是有效的）。为了防止这种情况出现，我们同时增大了 GPS 高度值噪声，使 GPS 自带高度信息减弱。代码实现在 src/pipeline/backend/optimizer.cc 中。

5.5 GPS/DR/Lidar 的 SE2 对齐

由于激光里程计和 DR 的轨迹都存在各自的累积误差，当场景规模大到一定程度或者半室内场景起点位置无 GPS 信号时，两者的轨迹和 GPS 真值之间存在较大的偏差，这个偏差在进行后端优化时会使得一些可用的约束被判定为 outlier，导致最终地图在全局和局部上产生一定的冲突。在进行优化之前，先将 DR 和激光里程计的轨迹与 GPS 轨迹进行对齐，基于对齐后的 DR 和激光里程计的优化会更加合理的判断异常值和正确值，进一步提升优化的合理性，以保证在部分区域有 GPS 时能够满足此区域的 GPS 位置要求。代码实现在 src/pipeline/backend/optimizer.cc 中。

常用的对齐方式是 SE3 对齐，两条轨迹对齐可以认为是 3D-3D 的点集匹配位姿估计问题，可以通过 SVD 分解或者非线性优化的方式求解 ICP 问题来分别实现 DR 和激光里程计轨迹与 GPS 的对齐。但是由于实际采集数据的路线很可能是一条直线，这种路线对应的轨迹对齐将会导致在一个自由度上匹配退化的情况，这就使得对齐后的 DR 和激光里程计轨迹在横滚角上有较大的偏差，造成地图不可用。为了防止这种情况出现并结合车辆实际在地面上行驶的事实，假设在高度、横滚角和俯仰角 3 个自由度上，轨迹的性质一样，因此只对轨迹在航向角和平面 2 维位置进行对齐，即 SE2 对齐。如图 21 所示，(a) 是 SE3 对齐，(b) 是 SE2 对齐，将 GPS 轨迹、DR 轨迹和激光里程计轨迹投影到水平面上，基于 2D 轨迹进行轨迹对齐。

SE2 对齐采用非线性优化的方式进行求解，利用 G2O 优化器进行求解。定义：

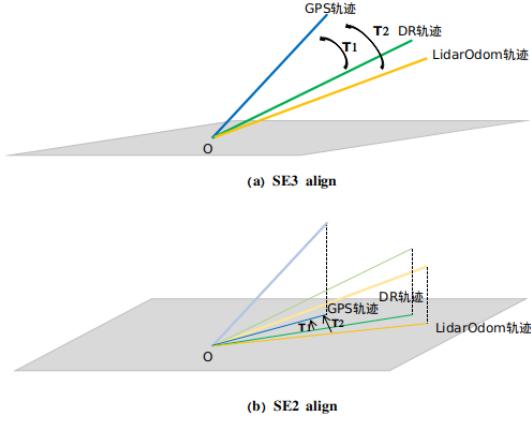


图 13: SE3 对齐 (a) 和 SE2 对齐 (b) 示意图

- 求解的 SE2 相对位姿变换作为顶点, 表示为 $\mathbf{T}_{\text{align}} \in \text{SE}(2)$;
- GPS 轨迹作为真值, DR 轨迹或激光里程计轨迹作为观测值, 两者之间的误差作为边, 这个边是由于顶点只是一个位姿变换, 因此该边是单元边。设关键帧 k 的 GPS 二维坐标为 $\mathbf{P}_k = [x_k, y_k]^T$, 对应的观测值的二维坐标为 $\hat{\mathbf{P}}_k = [\hat{x}_k, \hat{y}_k]^T$, 则误差项为:

$$\mathbf{e}_k = \mathbf{P}_k - \mathbf{T}_{\text{align}} \hat{\mathbf{P}}_k, \quad (26)$$

所以最小二乘目标函数为:

$$\min \sum_{n=0}^N \mathbf{e}_k^\top \mathbf{R}_k^{-1} \mathbf{e}_k, \quad (27)$$

进行 SE2 对齐后的 DR、激光里程计和 GPS 轨迹在水平面上, 更加全局统一, 高度和俯仰角可以结合场景的“层级关系”进一步提供相关约束, 这样的后端优化便可以在合理地处理各个传感器提供的位姿约束的同时在高度上也不会有太大的偏差。

5.6 回环检测与回环检测约束

5.7 全局姿态约束

除了上述约束之外, 由于车辆通常是水平向上运动的, 还需要加上建图时的全局姿态约束。车辆本身的 ODD 限制了它的滚转角与俯仰角范围, 而全局姿态约束体现了这一特点。

全局姿态约束计算车辆 Z 轴与世界系的 Z 轴之间夹角。设车辆位姿为 \mathbf{T}_{wb} , 那么 b 系下 Z 轴在世界系下为:

$$\mathbf{T}_{wb} \cdot [0, 0, 1, 0]^T = \mathbf{R}_{wb}(3, 1 : 3) \quad (28)$$

即旋转矩阵的第 3 行。该向量与 $[0, 0, 1]^T$ 作内积, 即得到所求的夹角余弦值。因此, 我们可以约束旋转矩阵第 (3, 3) 个元素近似于 1, 或者第 3 行前两个元素近似于零。实际当中我们使用了后者的做法, 并添加了死区限制 (即允许车辆有一定范围的滚转与俯仰):

$$\mathbf{e}_{\text{rot}} = [\mathbf{R}_{wb}(3, 1), \mathbf{R}_{wb}(3, 2)]^T \quad (29)$$

若这两个元素分量小于固定阈值（取 5 度），令残差为零，不对优化产生影响。

全局姿态约束会在 GPS 全局不准确，且未开启层级检测的情况下生效。此时，由于 GPS 无效，整个地图的全局姿态未知，可能产生一定的俯仰或滚转。而在制图软件 HAMO 中，通常假设 XY 是一个水平平面，这会导致地图在 HAMO 中看起来是倾斜的。加入全局姿态约束后，能够保证地图的 XY 平面与地面是平行的。

5.8 点云去重影

5.8.1 重复轨迹检测

由于车辆可能一直在一个地方来回运动，由于估计误差，可能导致地图同一区域被采样了多遍，导致数据冗余或重影。因此，在后端完成轨迹优化之后，还会调用重复轨迹检测与去除算法去除同样区域的点云。

点云拼接优化算法主要包含三部分：重复轨迹检测，重叠点的剔除，重复点云位姿微调。算法 2 描述了完整的拼接优化算法设计流程。

Algorithm 2 拼接优化算法流程

- 1: 输入: 优化后关键帧及对应点云
 - 2: 提取关键帧，构建建图轨迹；
 - 3: 建图轨迹分类，提取重复轨迹及参考轨迹；
 - 4: 根据建图轨迹预先构建全地图空间栅格，参考轨迹对应点云进行投影；
 - 5: 重复估计对应点云进行栅格投影，判断栅格状态，提取非重叠点；
 - 6: 提取参考轨迹点云作为配准参考；
 - 7: 重叠轨迹关键帧进行配准，校验输入状态及配准结果；
 - 8: 更新各帧点云变换矩阵及对应点云；
 - 9: 输出: 更新后的关键帧及点云向量
-

重复轨迹检测时采用空间距离作为检测标准，具体实现采用栅格进行轨迹筛选，规则如下：

1. 将整个建图轨迹进行栅格划分，栅格默认大小为 5；
2. 若两帧非相邻关键帧位于同一个栅格中，则认为两帧相交并重复；
3. 当 5*5 相邻栅格中存在两条或以上非相邻轨迹段时，认为时序靠后的轨迹段为重复路段；
4. 考虑到转弯轨迹对于整体建图影响，提取出曲率较大的轨迹段认为重复轨迹；
5. 建图轨迹的前段 10 米不允许设置为重复轨迹，其余轨迹按上述分类并对距离较近的重复轨迹进行合并

5.8.2 重叠点剔除及重复点云微调

基于检测出的重复轨迹，通过重叠点剔除及重复点云微调，实现点云拼接去重影。重叠点的剔除使用三维栅格，栅格大小设置为 1 米。首先将参考轨迹的点全部投影到对应索引的栅格中，再将重叠轨迹对应关键帧各点进行投影。某重叠轨迹对应点投影时，若对应索引中已存在点云点，则认为该点已重复，剔除该点；若对应索引中不存在点，则认为该处需由重叠轨迹填充，保留该点。由于采用三维栅格进行索引，整体计算效率较高。同时 1 米的栅格间距不会对定位造成较大影响。

重叠点的剔除优化目标在于减少重叠轨迹导致的墙面变厚问题，对于部分场景效果并不明显，故增加了点云微调步骤。点云微调实际参考了定位思路，以参考轨迹对应点云作为地图，重复轨迹作为配准目标，采

用 NDT 算法进行重复点云的姿态微调。

重复轨迹检测效果参考图 14。由图中可以看出，参考轨迹（非重复轨迹）基本覆盖完整地图区域，红色重复轨迹线均是轨迹存在重复区域或曲率较大区域。

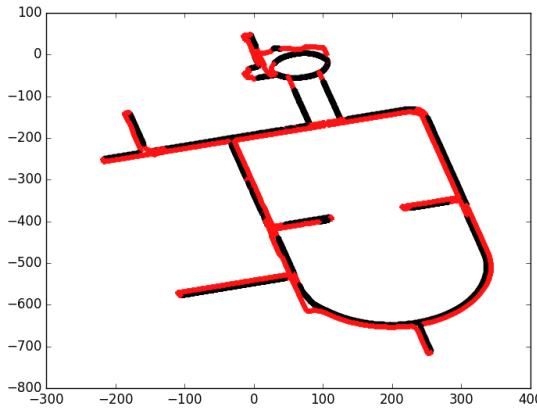


图 14：内蒙商贸轨迹分类：深蓝色：参考轨迹；红色线：重复轨迹

图 15 为点云拼接完成后效果对比。该场景为海南诺雅达景区场景，上图为原始部分点云，下图为点云拼接后效果展示，可以看到点云拼接后重影明显改善。

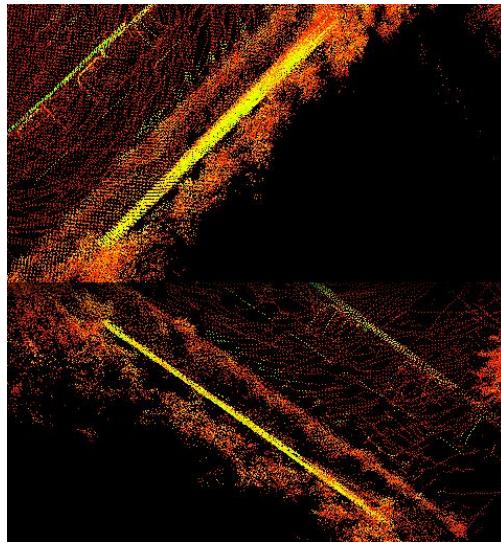


图 15：重叠点云处理效果展示：上图为原始部分点云，下图为点云拼接后效果

6 仿真、准入与准出

6.1 仿真

仿真和准出作为整个建图中最后一个环节，又称数据验证环节。该环节主要完成点云地图的激光匹配验证、准出项检测以及相关数据保存等内容，任意过程出现问题或者不满足准出检测标准，均会给出相关原因，根据原因优先级输出数据验证最终检测结果，仿真错误优先级将高于准出错误优先级。

Algorithm 3 仿真与自动准出

- 1: 仿真
 - 2: 前期数据准备；
 - 3: 获取单个激光匹配数据包并完成数据转换；
 - 4: 选择初始化定位模式，并根据模式选择对应的初始化定位预测位姿（gps、固定点、记忆点或者广域搜索）；
 - 5: 定位初始化；
 - 6: 若初始化定位成功，进行后续激光匹配；若定位初始化不成功，记录并执行 3 -6 步骤；
 - 7: 实时将激光匹配记过与 imu、odom 或者 gps，进行 MSF 融合，若出现融合定位失败，切换成传极光定位，若纯激光定位丢失，执行 3 -7 步骤；若没有定位丢失（含融合和纯激光定位），执行下一步；
 - 8: 若还存在未仿真数据包，执行 3 -7 步骤；
 - 9: 计算自适应激光匹配阈值；
 - 10: 若存在故障信息，则需要判断类型并进行相应操作，并输入至报告信息内；
 - 11: 准出检测
 - 12: 仿真结果数据输入；
 - 13: 数据完整性检测；
 - 14: 平滑性、断层、位置和角度偏差检测与统计；
 - 15: GMM 地图生成；
 - 16: 相关数据保存；
 - 17: 输出数据验证最终检测结果
-

6.1.1 仿真过程

为了更好的定义仿真过程中出现的问题类型，以及是否需要重建点云地图等，将仿真过程输出的故障（错误）等级分为 3 级：0. 正常；1. 部分不正常；2. 致命错误。详细的故障信息会在 report (pdf 文档) 中给出，结合故障等级和详细的故障信息快速定位问题类型和原因，为有效解决故障或问题提供了重要依据。

仿真过程主要包含：前期数据准备、定位初始化、激光匹配定位、MSF 融合⁴以及自适应激光匹配阈值生成。

1. **前期数据准备。** 获取需要仿真数据包相关信息和地图加载模式，若为整块地图加载模式，此阶段需要加载整张地图；若为瓦片模式，此阶段只需要获取所有瓦片地图相关信息，无需加载；若为区域性加载

⁴激光、GPS、轮速及以及 IMU 传感器数据进行的多传感器定位融合。

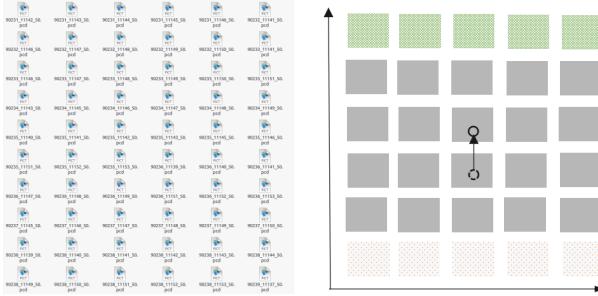


图 16: 瓦片地图加载例子: 车辆移动过程中, 绿色为新增瓦片地图, 白色为剔除以前瓦片地图, 灰色为保持不变地图

模式, 无需加载。

2. **定位初始化**。目前初始化方式有四种: GPS、固定点、记忆点及广域匹配。首先根据数据包类型, 若是 d 包, 需要判断是否为自动分包, 若为自动分包, 则使用上一个 bag 最后点为记忆点进行初始化, 若未成功, 使用 10 次固定点初始化, 若仍然定位失败, 依次进行 3 次 gps、10 次记忆点和 200 次广域搜索匹配; 若依然未成功, 则需记录并给出具体原因; 若不是自动分包, 使用 10 次固定点初始化, 若定位失败, 先后依次进行 3 次 gps、10 次记忆点和 200 次广域搜索匹配; 若依然未成功, 则需记录并给出具体原因。若数据包为 z 包, 则需要判断是否为自动分包, 若是自定分包, 则使用上一个 bag 最后点为记忆点进行初始化, 若未成功, 则依次进行 3 次 gps、10 次固定点以及 200 次广域搜索初始化, 若仍未成功, 则需记录并给出具体原因, 便于后期问题分析; 若不是自定分包, 依次进行 3 次 gps、10 次固定点以及 200 次广域搜索初始化, 若仍未成功, 则需记录并给出具体原因, 便于后期问题分析。上述任意过程定位成功后, 便可进行后续的匹配融合操作。
3. **激光匹配定位**。目前激光匹配算法为 NDT, 在初始化定位时, 激光的频率为 10Hz, 正常匹配时, 激光的频率降至 3Hz。若定位过程中出现定位丢失, 若融合定位未失效, 便根据融合提供的位置进行初始化, 若未成功, 依次进行 3 次 gps、10 次固定点和 200 次广域搜索匹配; 若成功, 记录并继续激光匹配; 若仍然失败且融合定位失效 (递推是有时间限制), 测记录失败原因并结束此数据包的激光匹配过程。
4. **MSF 融合**。融合定位频率为 100Hz, 完成单帧激光匹配后, 需将匹配结果、GPS、IMU 以及轮速计进行融合定位, 作为最终车辆位置结果, 直道所有数据包验证完, 由于融合算法优势, 根据融合定位结果的状态和返回信息可以用于检测地图畸变、断层等问题。若过程中出现融合定位丢失, 融合定位模式关闭, 同时记录融合定位丢失相关原因, 根据失败原因等级 (事先设定好的错误等级, 即严重性), 判断是否进入进入纯激光定位 (3Hz)。
5. **自适应激光阈值生成**完成所有数据包定位仿真后, 通过基于时间序列的分割聚类方法优化激光匹配阈值, 获取本场景各处最佳的定位阈值。

至此, 整个仿真过程结束, 其中图 16 为瓦片地图加载例子, 现阶段瓦片地图形状是边长为 50m 的正方形, 每次加载车辆周围 25 块, 当车辆运行至新瓦片区域时, 加载的地图需要删除远距离瓦片地图和新载入近距离瓦片地图。

由于算法迭代, 地图加载方式由之前的瓦片式升级为特定范围内局部地图模式, 即根据车辆位置, 加载方圆指定距离的关键帧数据 (150m), 合并成局部地图。如仿真过程中, 车辆出现定位丢失或者初始化定位失败, 均会在报告中生成问题指定区域范围内的点云数据, 如图 17 所示, 通过最后生成的报告可以看出采集时, 贴边数据在点云地图外。

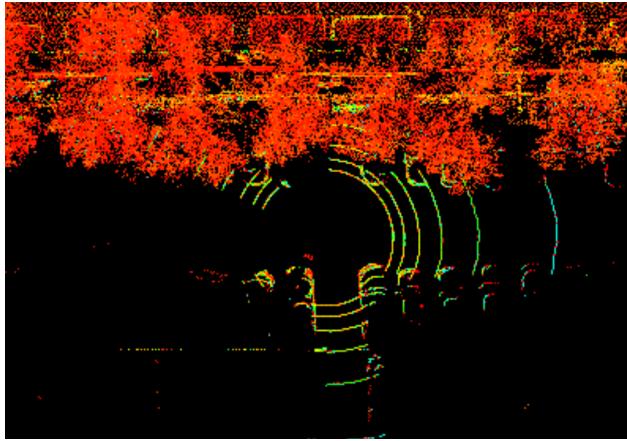


图 17: 某场地初始化定位失败事例: 黄色为车辆实际点云数据, 红色为场地图点云数据

6.2 准入

由于 mapping3.0 建图约束中, GPS 有效性是及其重要一项, 为此建图 odd 中对于运营环境 GPS 有效性有较强的约束, 要求运营环境非室内, 并且采图起始点的 GPS 位置及航向均需要超过 30 秒的有效时间。起始点 GPS 约束既对运营地图采集增加了较大工作量, 增加了适配时间, 又限制了运营场地的拓展。为拓展建图 ODD 场景, 放宽 GPS 约束, 允许室内发车, 同时对于建图准入约束, 建图算法等方面进行优化及改动。建图算法方面的优化在别的章节中叙述, 本章主要针对建图准入约束。

建图准入检测部分包含文件完整性、有效性检测, 和对 GPS 有效、弱 GPS 和无 GPS 情形进行了分类处理。下述是几项主要内容:

1. 输入数据完整性检测, 两个 launch 是否缺失, bag 包数目和类型;
2. 输入数据有效性检测, 文件格式是否损坏、内容 (话题) 是否缺失;
3. 根据 GPS 定位状态对当前场景进行分类。对当前场景所有建图数据包所有 GPS 数据进行位置有效性检测及航向有效性检测, 采用全程最优 GPS 状态进行分类。目前场景共分为 5 类: GPS 最优为位置及航向同时有效; GPS 最优为位置及航向在不同时段有效; GPS 最优为位置有效但航向始终无效; GPS 最优为浮点解或单点解; GPS 全程无解。将检测出的五类状态实时写入地图配置文件, 该状态可供后端使用。实际上, 当 GPS 全程无解时, 认为该场景为室内场景; 当 GPS 处于其他分类时, 认为该场景为半室内或全室外。
4. 根据 GPS 最优状态确定地图偏移量。当场景为半室内或全室外时, GPS 存在定位解, 直接以最优状态位置经纬度解算当前 UTM 位置作为偏移量; 当场景为全室内时, 目前偏移量为固定位置, 后续可由大数据导入。
5. 弱 GPS 数据噪声的设定。当最优 GPS 状态为浮点解或单点解时, 为保证地图与 GPS 的基本对应, 除了前面做的轨迹对齐, 修改弱 GPS 定位输入的协方差。根据 GPS 误差模型, 统一设定浮点解或单点解时 GPS 位置误差为 50.0 米, 以此施加 GPS 对最终优化的影响。

6.3 准出

为了对点云地图各项准出指标的自动检测，以及是否需要重建点云地图等，将准出过程输出的故障（错误）等级分为3级：0. 正常；1. 部分不正常；2. 致命错误，其效果和仿真过程相同。

准出过程主要包含：激光匹配（NDT）轨迹自身的跳变检测、GPS与融合轨迹和航向偏差、存储数据检测以及GMM地图生成。

1. **激光匹配（NDT）轨迹自身的跳变检测。** 激光匹配轨迹跳变分为两大类：a. 匹配曲线的不平滑；b. 轨迹点出现断裂；针对a类：使用上一帧定位结果和DR递推当前帧位置，计算当前帧和递推位置之间的欧式距离，不大于30cm则认为定位轨迹平滑，超出则认为定位轨迹不平滑，此时输出检测不合格，并输出出现不平滑轨迹编号、坐标以及跳点偏差，并记录这些点的位置坐标，将轨迹状态改成1；针对b类：直接计算前后两帧之间的欧式距离，若超过70cm（暂定），则认为该处轨迹存在断裂问题，将轨迹状态改成2；并统计不平滑的点个数以及断裂次数。将不平滑和断裂点已统计数据形式输入至报告中。若不平滑数量不满足准出标准时，会报错，并以红色预警提示。
2. **GPS与融合轨迹和航向偏差。** 首先根据GPS数据状态，将GPS数据进行分类（0-5），由于GPS本身存在误差，为了降低其噪声干扰，在比较时，只使用类型4的点，统计其角度和位置偏差的平均值和指定范围内的分布情况，根据分布情况和平均值判断该地图是否满足准出标准。
3. **存储数据检测。**
 - 数据是否为空检测，除了轨迹之外，其他相关数据缺失时只提示一个警告，而轨迹为空时，为致命错误，需要强制退出；
 - 功能点数据存储过程检测。基于需求和版本问题，功能点数据为可选的，但后期该数据为必须项；
 - 图像数据存储过程检测，图像数据目前是可选项，主要方便HAMO⁵软件制作贴边数据，但是在后期的点云地图构建中，除5G车辆外，图像数据是必需项。
 - 广域初始化原始数据存储过程检测，该数据用于车辆广域初始化，为了提高车辆初始化定位成功率，此数据为必须项。
 - 轨迹数据检测。若建图数据包轨迹为空时，检出失败并退出，若贴边数据包轨迹为空时，记录并提示一个警告，无需退出。

4. **GMM地图生成。** 随着产品段车辆定位算法不断审计，准出中提供了新算法（GMM）地图生成器，可以根据配置文件中，设置是否需要生成GMM地图，或者生成何种GMM地图（IDP和LADS）。

在整个仿真和自动准出结束后，会在报告中生成去全场景GPS状态、轨迹偏差、航向、匹配阈值以及自适应激光匹配阈值，如图18，便于直观查看整个场景定位状态。

由于准入、准出以及仿真均是对建图数据在不同阶段数据进行完整性、有效性的检测，各自存在一定的关联。因此，检测问题输出为全局性，下面对整个检测错误分布情况进行简单列举。各检测项分布如下如表1所示。

由于故障等级不同，准出输入等级以及优先级不同，可忽略的故障优先级最低，直接印象定位结果的优先级最高，其具体定义如表2所示。

⁵HAMO：制作高精度语义地图的自研软件工具。

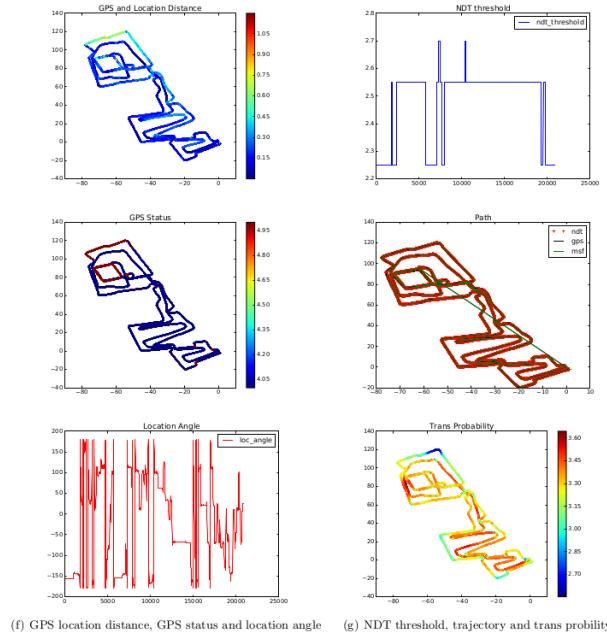


图 18: 数据验证输出的相关曲线图

表 1: 检测项说明

检测项	环节	8月版本	11月版本
硬件问题	仿真	无	有
地图问题	仿真	无	有
激光退化	仿真	无	有
定位问题	仿真	有	有
采集不规范	仿真/准出	有	有
仿真数据完整性	准出	有	有
仿真数据质量	准出	有	有
GMM 地图	准出	无	有

表 2: 检测输出状态详细变更

检测项	是否显示
硬件问题	是
地图问题	是
定位警告	是
定位问题	是
采集不规范	是
未知原因	是
轨迹平滑	是
轨迹偏差	否



图 19: 数据上传失败示意

7 图形界面、调试工具

7.1 mapping-ui

7.1.1 mapping-ui 的使用

为了在 12 月份适配现场建图人员，现在为 3.0 增加了一个图形化调用界面。该界面与软件是松耦合的，可以通过 mapping-ui 启动。界面参见图 5。

mapping-ui 是一个基于 Qt5 的应用程序。将来现场建图人员可以通过该程序输入建图数据包和地图名称，然后点击“开始建图”按钮进行建图。该程序后台和 3.0 保持一致，它会建立一个建图流水线引擎，然后和云端一样执行建图任务。mapping-ui 会随着 3.0 软件包一同发布，用户可自行选用图形界面或者命令行界面。图形界面主要是为了方便不熟悉 Linux 命令行的操作人员而设计的。

mapping-ui 目前提供建图功能分为新建和拓展，地图更新功能尚未提供，新建和拓展通过 config.yaml 配置文件任务类型进行配置。进行地图拓展时根据需要拓展的地图是否位于本地分为两种情况，若原始数据不在本地，将自动从云端拉取。地图生成后自动上传云端并提供手动重新上传功能，如图 19。

7.1.2 现场建图

Mapping 3.0 的主分支上也存在 mapping-ui，不过目前现场建图的分支主要维护于 onsite 上。onsite 分支与主分支的主要区别在于：

1. mapping-ui 使用采集地图时生成的 config.yaml 作为建图的基础配置文件，而不是数据包或者下载链接；
2. mapping-ui 生成简化版本的报告，如图 20；
3. 在启动时需要登入建图员的帐号与密码，默认登录失败 5 次退出；
4. 在现场建图过程中，mapping-ui 还会向云控制服务器实时上报建图情况与建图进度，并在完成之后自动上传成果数据；由于网络原因上传失败后，可手动点击“重新上传建图结果”重新上传数据。
5. 目前 mapping-ui 配置参数中 GMM 文件默认为 False，默认不进行 GMM 文件生成，这是由于目前 GMM 文件生成占用内存较多，在现场电脑中可能会导致内存不够。

众创 H5 手机适配测试 场地 ID:C318000608636545 建图报告

建图软件版本 3.2.3, 建图人员: huangyou

March 19, 2020

1 简报

Table 1: 综合信息						
地图名称 (ID)	C318000608636545					
用时	3 分 32.4 秒	有效包数	2	数据量	562MB	
长度	85.4m	推算面积	854 平米	是否成功	是	关键帧数 231
推出	成功	失败理由 (若有)				

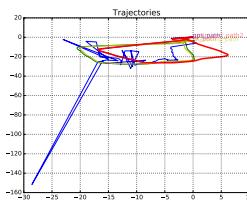


Figure 1: 轨迹

1

图 20: 简化版报告

6. onsite 分支 mapping-ui 软件的安装文件为 install_onsite.sh, 该文件将安装软件中所有 python 脚本文件转化为.pyc 文件进行加密, 软件运行时自动调用.pyc 文件保证正常运行

7.2 debug-ui

7.2.1 debug-ui 的用途

在缺少 GPS 信息时, DR 和 Lidar Odom 不可避免地会出现累计误差。对于大型室内场景, 很容易出现闭环无法正确计算, 或者出现共视重影的情况。但是, Lidar 数据特征简单, 本身并不便于自动计算回环, 所以 mapping 3.0 提出了人工闭环和修图的概念, 并实现了一套配套的程序 (debug-ui)。该程序主要用于处理回环失败的情况, 也可以用于调整关键帧位姿, 减少地图重影的效果。

debug-ui 是基于 Qt5 的可视化程序, 运行界面如图 21 所示。该程序主要有三列面板组成:

左侧面板可以指定要读取的数据库文件和对应的关键帧轨迹文件。读取后, 点云信息会在中央 3D 窗口显示。3D 窗口是一个 QVtkWidget, 集成了 PCL_Visualizer 的功能, 可以通过鼠标拖动查看点云内容, 同时会显示建图轨迹、当前帧点云等额外信息。在读取地图后, 当前关键帧会以白色高亮显示出来, 用户可以通过左侧面板逐个查看关键帧位置姿态, 而且可以直接对关键帧进行微调。

右侧面板是该程序几个主要的功能: 主动闭环和交互式优化。在主动闭环功能中, 用户可以打开回环帧高亮, 显示两个回环的候选帧。然后, 通过主动微调其中一个, 提示系统对该回环进行优化。在确认回环后,

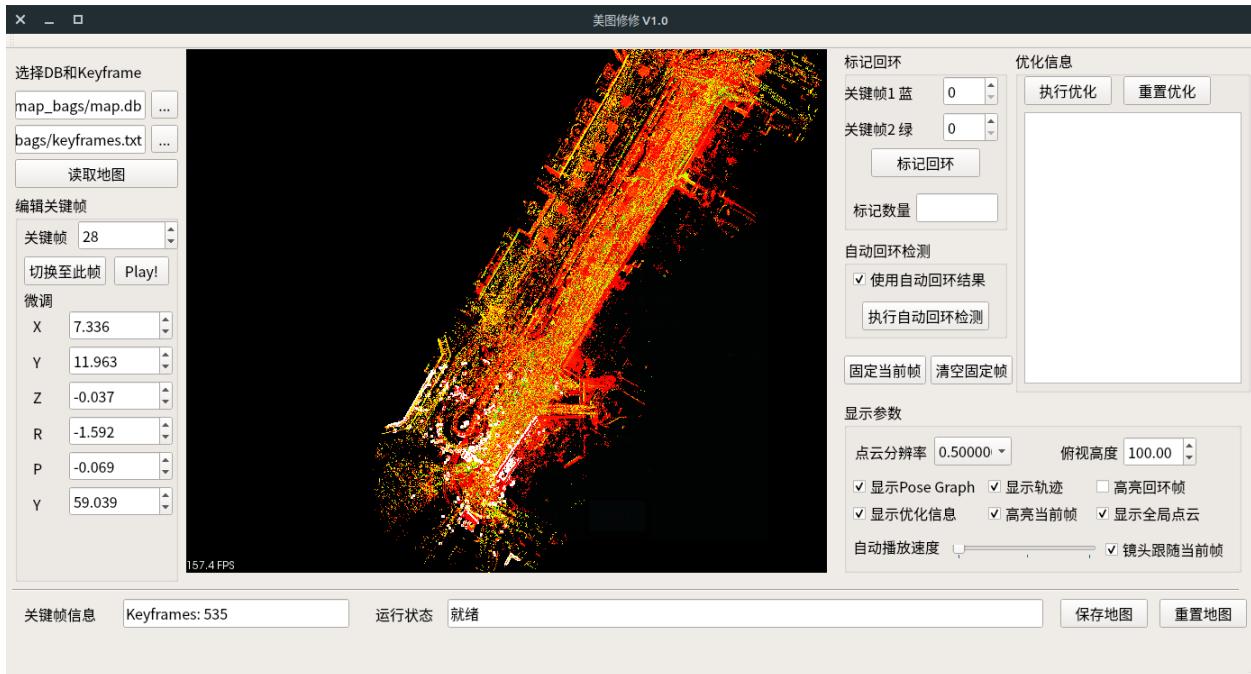


图 21: Debug-ui 图形界面

点击执行优化即可启动后端优化。优化对轨迹进行更新后，会重新渲染整个点云地图。

右下面板可以对显示参数进行调整。用户可以调整显示点云的分辨率（默认支持 0.05, 0.2, 0.5 三级分辨率）。在自动播放轨迹时，可调整镜头高度和镜头是否跟随当前关键帧。同时，还可以开关一些高亮显示部分。

7.2.2 使用 debug-ui

下文演示一个成功修图的案例。该案例来源于众创空间地库的一次数据采集。在这个例子中，采集员从地库入口处，先由南向北走至尽头，再由北向南行走，但未回到原点。随后，采集员又采集了南侧的通道，随后回到入口处，最后又采集了一段从入口向北的通道数据。此时由于递推轨迹较长，累计误差较大，又缺少 GPS 信号，导致轨迹无法闭合，地图出现两个明显错位的情况，可以从图 22 (a) 中看出。该图蓝色和绿色示意的结构应该为同一个，但由于轨迹偏差，出现了重影。

注意这种情况在目前 3.0 建图中是无法修复的。3.0 的回环检测在第一轮优化后进行，要求第一轮优化后轨迹应该有大致不错的结果。在室外地图中，由于 GPS 信号的存在，我们通常能够给出回环处的大致估计，但这点在室内区域是不可行的。如果没有人工调整，此类地图就无法发布。

现在我们演示如何对该图进行修复。首先，建图员可以在发现地图重影时，可以通过播放 GUI 界面播放关键帧轨迹，判断重影区域的关键帧编号。通常，一处地图重影代表至少两个轨迹片段在此处采集到了点云。在本例中，我们以关键帧 11 和 1148 为例，点击高亮回环帧后，这两帧的点云就以绿色和蓝色高亮显示出来，可以看到它们显然有重复的结构（墙角、墙边等），但轨迹出现了偏差。

现在，为了修复此处重影，我们对其中一个进行调整，使它和另一个对齐。当然，调整蓝色帧还是绿色帧是任意的，这里以调整绿色帧为例。我们把当前帧切换至绿色帧所在编号，再通过左侧面板调整其位置和

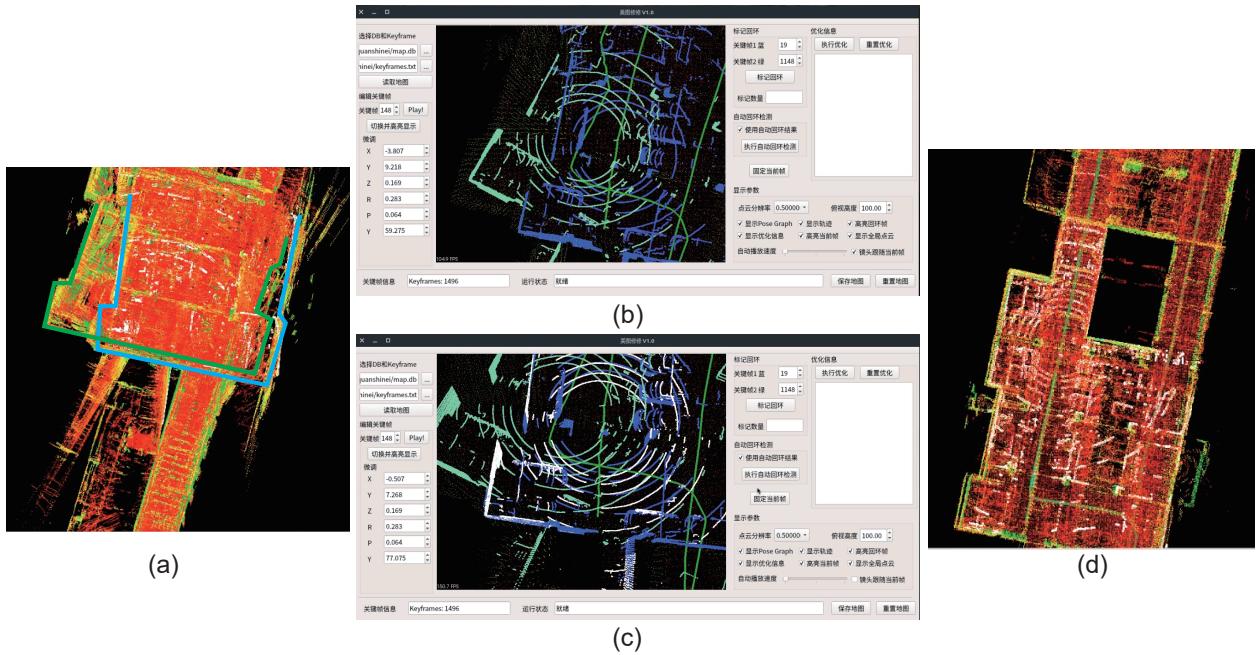


图 22: 修图示例: (a) 由于未成功闭环, 地图显示出两个明显的重复区域; (b) 我们通过对建图轨迹进行分析, 选取两个可能的回环帧 (绿色和蓝色标记的点云); (c) 调整其中一个 (本例为绿色) 至正确位置, 调整后白色轨迹和蓝色轨迹对齐; (d) 利用交互式优化, 得到正确的地图

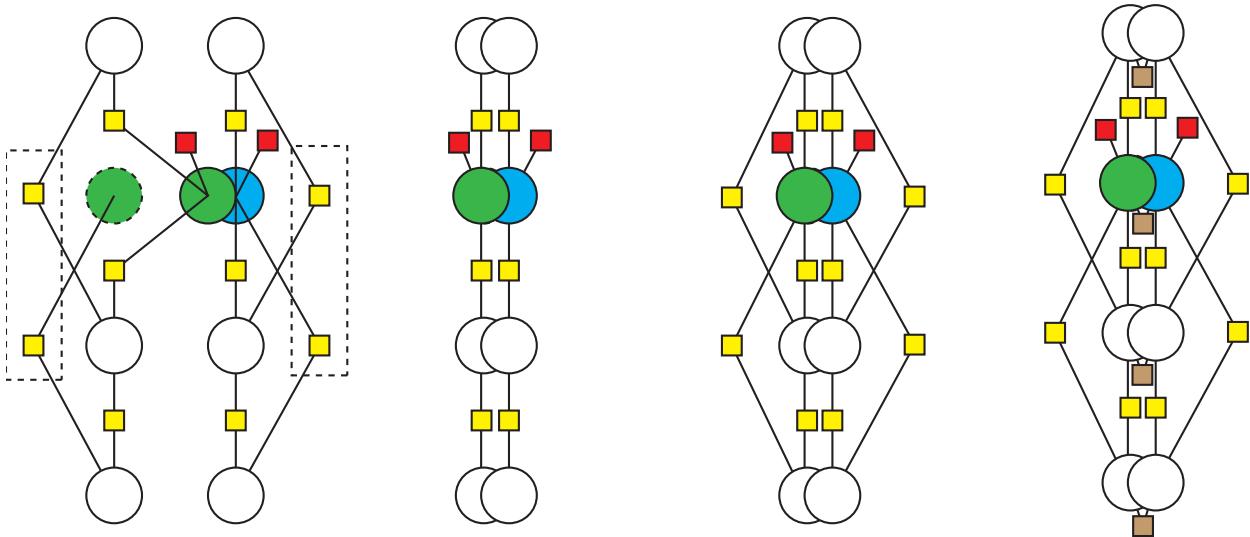


图 23: 主动闭环对应的后端逻辑示意图。圆形: 优化变量, 正方形: 因子 (黄色: 相对运动因子, 红色: 全局位姿因子, 棕色: 闭环因子)。从左至右: 第 1 步, 给主动闭环的两帧加上全局位姿约束, 然后取消非相邻帧的连接因子; 第 2 步, 优化后, 由于全局 Pose 的因子, 两个回环帧附近的整段轨迹被拉至一起; 第 3 步, 加回之前取消的非相邻帧的连接因子, 再次优化, 以保证局部结构的正确性; 第 4 步, 使用自动回环检测加上其他帧的回环约束, 再次优化

姿态。当前帧的点云会以白色显示, 所以我们只要调整白色帧, 使之与蓝色帧对准即可, 如图 22 (c) 所示。此时, 人工对准的结果相当于给出了这两个帧的期望位姿, 接下来需要把该信息放入后端优化中。UI 界

面的右侧面板是交互式优化模块。为了纠正闭环，需要完成以下操作，其对应的 Pose Graph 因子图变化示意图见 23：

1. 点击“固定关键帧”，这会给人工选择的两个回环帧加上全局位姿约束。在优化中，实际上会加入这两个顶点的 6 自由度约束，且信息矩阵取较大值（目前使用对角线为 10^6 的信息矩阵）。
2. 点击“执行优化”，以运行一次后端优化。此时后端优化的逻辑如图 23 左侧所示。首先，约束两个回环帧的全局位姿，同时删去非直接相邻的相对运动因子。这是因为，人工调整只能确定单帧的正确位姿，而难以操作它局部其他帧的位姿，所以局部看来，当前轨迹就像是“平滑轨迹上有一个异常远的关键帧”。优化算法会倾向于认为该关键帧为异常值，但现在这个异常值反而是正确的值。跨过被调整帧的那些相对运动因子倾向于保持原有的局部结构，所以我们首先把它们删掉。删掉之后，Pose Graph 变成链状的，优化算法能够将局部轨迹进行修正。优化完毕后，再把之前删除的非相邻帧因子加回至优化，以保证局部结构的最优性，如图 23 第 3 列所示。
3. 然后，建图员点击“标记回环”，此时回环检测算法根据当前轨迹状态，检测两个候选帧附近的回环情况。如果回环检测成功，就给 Pose Graph 中所有附近的回环帧加上回环约束，得到更好的回环结果。
4. 再次点击“执行优化”后，就会代入正确的回环结果，得到修正后的轨迹，如图 22 (d) 所示。

整体而言，主动回环对应的后端有一个“先减后增”的操作方法，该方法是我们在实际过程中摸索出来的一套行之有效的办法。它能够处理复杂的室内回环逻辑，也能够处理“共视重影”类问题，为重影问题提供了解决办法。

7.3 报告的生成与发送

Mapping 3.0 中，每条流水线都会维护两个基本信息：

- 流水线的 TaskInfo 结构，该结构包含了流水线的执行情况、起始时间、成功与否等关键信息；TaskInfo 由 Pipeline Engine 持有，每个 Pipeline Context 可以在执行过程中更新该结构的内容。最终，这个结构被存储至结果文件夹的 results/task_info.txt 中。
- 流水线的详细报告。详细报告是一个字符串，流水线执行过程中，算法可以根据需要，把关键信息打印在这个字符串中。这个字符串最终会生成到报告 pdf 的文本里。

在 Pipeline Engine 完成所有的 context 之后，会根据 TaskInfo 和 report 来生成输出报告，并且调用一些绘图脚本把轨迹、地图、仿真结果等信息绘制成为图片并插入报告当中。

报告的生成使用了 L^AT_EX 引擎。Mapping 3.0 会首先生成一个 report.tex 的文本文件。该文本文件使用了一个 tex 模板，mapping 3.0 只需把结果填入模板中即可。然后，调用系统当中的 L^AT_EX 指令把报告编译为 pdf（中文报告则是 xelatex）。显然，如果系统中没有安装 tex 软件，就不会生成 pdf 报告。

报告中使用的图片来自 scripts 下的绘图脚本，主要包含：地图俯视图、轨迹图、仿真过程的数据图表。此外，仿真过程会生成一个仿真视频，然后通过 ffmpeg 转换成容量较小的 simulation.flv 文件。该文件在报告中提供链接，用户可以下载查看。

报告生成后，系统调用 scripts/send_email.py 脚本，将报告群发给组内所有人。如果维护人员需要添加报告的接收人员，只需添加其邮箱至该脚本即可（图 25）。

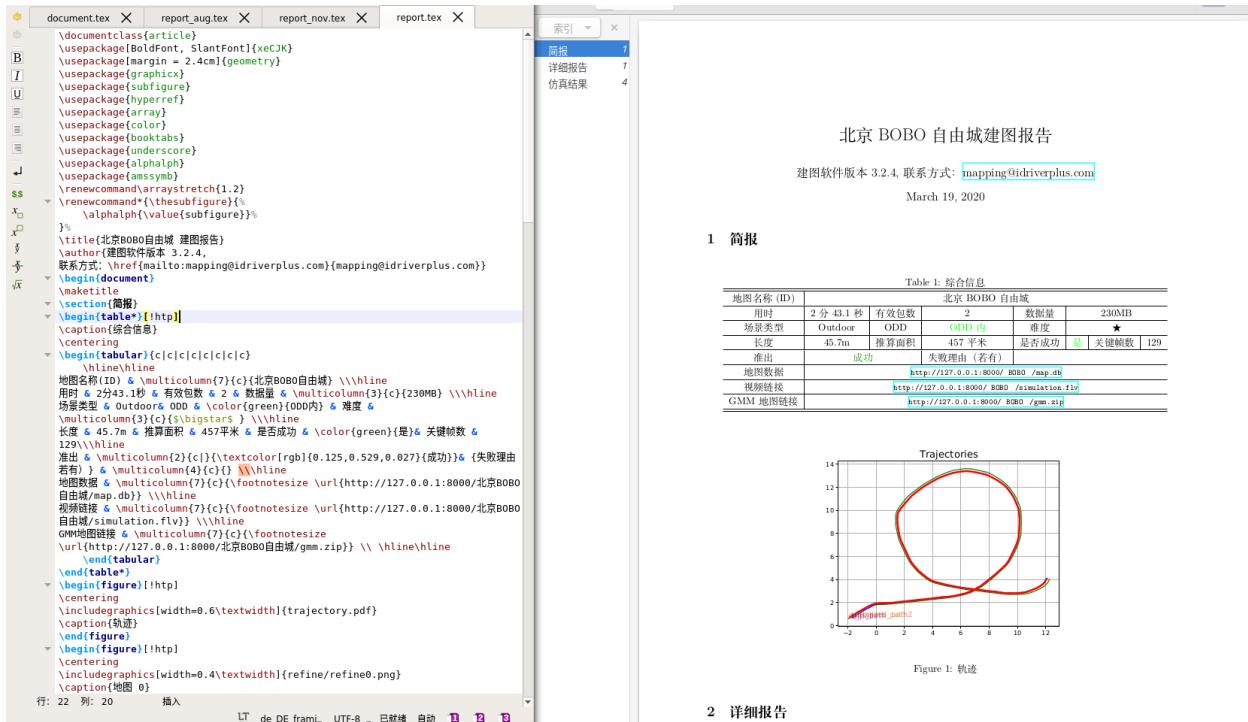


图 24: 生成的 report.tex 与编译之后的 Pdf 文件



图 25: 发送邮件的脚本与对应的邮件组

8 已知的建图问题与方案

如果下游认为建图出现问题，可以与定位组算法研发人员联系。以下列出目前已知的建图问题与解决方案。

1. 在大范围室内外混合，且有斜坡场景下，建图的层级分析可能会出错。层级分析出错时，地图可能出现上下分层的情况。如果出现此类情况，建议在配置项中关闭层级分析，重新运行建图；
2. 在 GPS 不佳且未打开层级分析或全局姿态时，地图全局位置姿态可能缺少约束，导致整体翻转、倾斜；建议打开层级分析或全局姿态其中一项；
3. GPS 约束与全局姿态约束可能出现冲突。如果地图出现扭曲、断裂等问题，可能与全局姿态约束有关。建议关闭全局状态约束并重新运行建图；
4. 由于场景相似度高导致回环出错，可能导致地图出现明显重影。建议这种情况下使用修图软件进行修复；
5. 对于某些异形建筑，激光匹配可能出现错乱，建议上报这种情况，由算法人员重新设置激光匹配参数。

参考文献

- [1] Y. Latif, C. Cadena, and J. Neira, “Robust loop closing over time for pose graph slam,” *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1611–1626, 2013.
- [2] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “g2o: A general framework for graph optimization,” in *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, IEEE, 2011.
- [3] T. D. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.
- [4] K. De Brabanter, K. Pelckmans, J. De Brabanter, M. Debruyne, J. A. Suykens, M. Hubert, and B. De Moor, “Robustness of kernel based regression: a comparison of iterative weighting schemes,” in *International Conference on Artificial Neural Networks*, pp. 100–110, Springer, 2009.