移动机器人系列课程

机器人定位与导航系列

文档版本 V1.0（2022/6/20）

深圳市史河机器人科技有限公司

Cartographer 算法建图

1. SLAM 激光雷达建图

机器人技术的迅猛发展，促使机器人逐渐走进了人们的日常生活，服务型室内移动机器人更是获得了广泛的关注。室内机器人的定位与导航是其中的关键问题之一。在这类问题的研究中，需要把握三个重点：一是地图精确建模；二是机器人准确定位；三是路径实时规划。

SLAM（Simultaneous Localization and Mapping，即时定位与地图构建）最早于1988 年提出。作为一种基础技术，SLAM 从最早的军事用途到今天的扫地机器人，吸引了一大批研究者和爱好者，同时也使这项技术逐步走入普通消费者的视野。

使用 ROS 实现机器人的 SLAM 和自主导航等功能是非常方便的，因为有较多现成的功能包可供开发者使用，如 gmapping、hector_slam、cartographer、rgbdslam、ORB_SLAM、move_base、amcl 等。

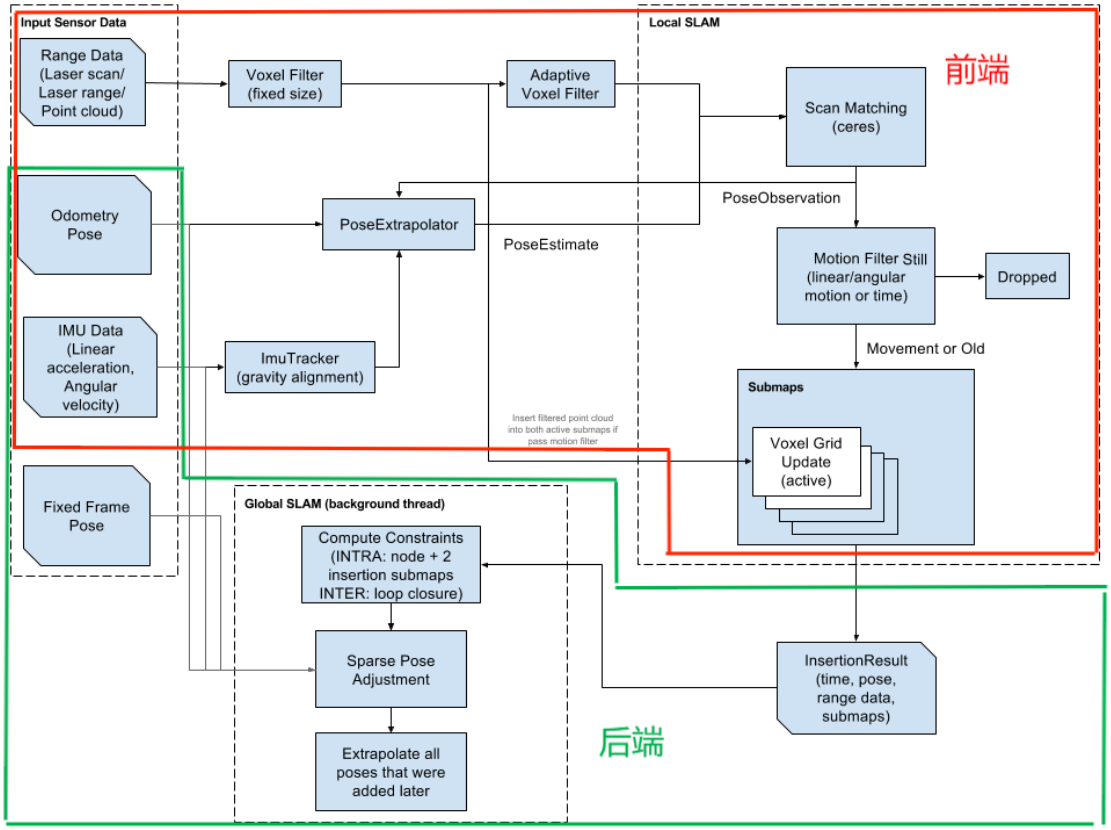
本节将学习 Cartographer 算法建图。

2. Cartographer 介绍

Cartographer 是谷歌在 2016 年开源的一个可以在多传感器配置下实现低计算资源消耗的 SLAM 算法框架。Cartographer 是一个实时的室内建图算法，能生成分辨率 r=5cm 的栅格地图。在前端将最新的激光雷达扫描数据在相邻的子图上(整个地图的一小块)完成扫描匹配，得到一个在短时间内准确的最佳插入位置(位姿)后，将扫描插入到子图中。扫描匹

配中，位姿估计的误差会在整个地图中随时间逐渐累积，在后端中，通过回环检测加约束进行优化消除误差。下图为 Cartographer 算法框架图。整个框架分为数据源、局部 SLAM 以及全局 SLAM 以及数据融合等部分。

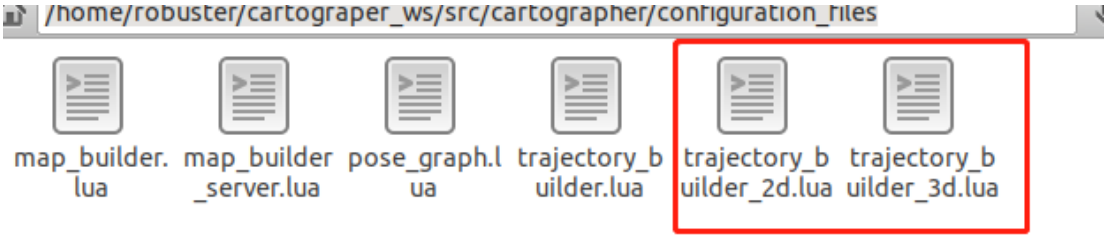
2.1 Cartographer 算法框架理解



Submaps可以理解为:地图是一片一片拼接逐渐形成的,一片地图便是一个 submap。

Cartographer 是整体可以分为两个部分:

一是 Local SLAM 部分，也被称之为前端（Frontend）；该部分的主要任务是建立维护子图（Submaps），但问题是该部分建图误差会随着时间累积。该部分相关的参数定义在/home/robuster/cartograper_ws/src/cartographer/configuration_files 文件夹中。



二是 Global SLAM 部分，也称为后端（Backend）。该部分的主要任务是进行回环检测（Loop Closure）。回环检测本质上也是一个优化问题，它在后台线程中运行，它的主要工作是找到闭环约束。它通过针对子图的扫描匹配扫描（主要是通过节点 Node）来做到这一点。将位姿推移类传感器和前端结果按时间顺序进行串联起来形成一条轨迹，寻找闭环关系，最终采用优化求解获取位姿总体误差最小时的位姿轨迹即 submap 位姿。在 3D 中，它还试图找到重力的方向。它的大部分选项都可以在 /home/robuster/cartograper_ws/install_isolated/share/cartographer/configuration_files/pose_graph.lua 中找到。

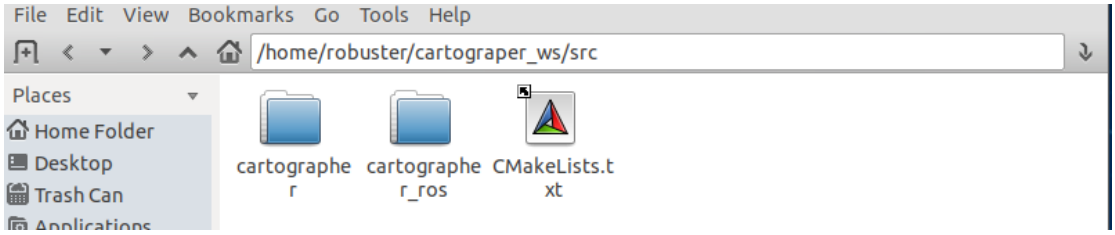
更抽象的说，局部 SLAM 的工作是生成更好的子图，全局 SLAM 的工作是更好的将子图约束到一起。

可以看出，从传感器出发，Laser 的数据经过 VoxelFilter（体素滤波，点击链接）和 Adaptive Voxel Filter（自适应体素滤波），用来构建子图（submap），而新的 Lsaer Scan 进来后也会插入到已经维护着的子图的适当位姿。如何决定插入的最优位姿，就是通过 Ceres Scan Matching 来实现的。估计出来的最优位姿会与里程计和 IMU 数据融合，用来估计下一时刻的位姿。

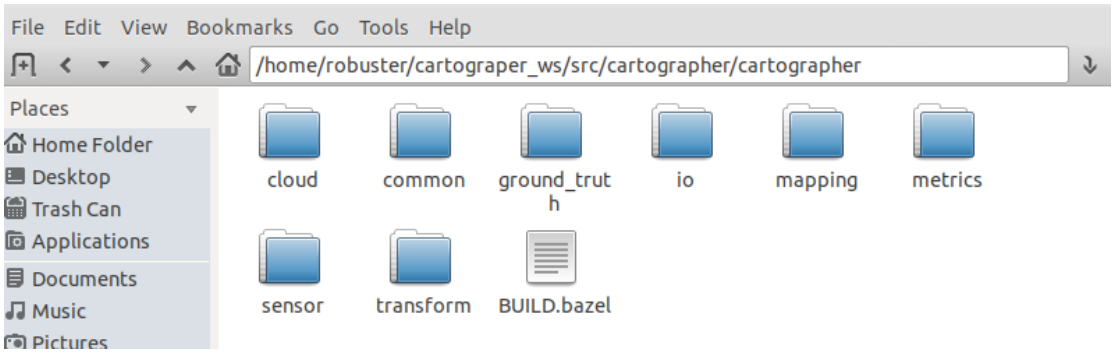
2.2 Cartographer 软件包的 ROS 框架

BR280 的 Cartographer 软件在名为 cartographer_ws 的工作空间中，在 src 目录下

它的代码主要分为两个部分：cartographer 和 cartographer_ros。



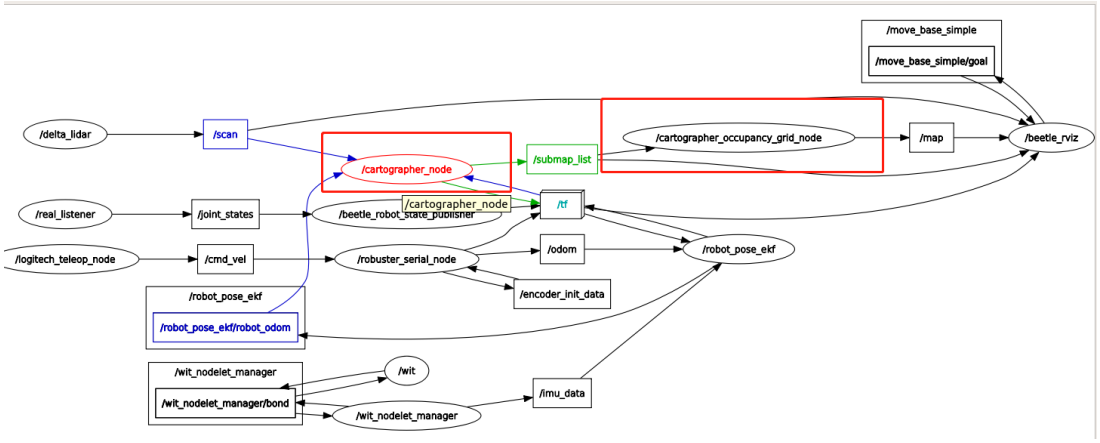
cartographer 主要负责处理来自雷达、IMU 和里程计的数据并基于这些数据进行地图构建，是 cartographer 理论的底层实现。cartographer_ros 则是基于 ROS 的通信机制获取传感器的数据并将它们转换成 cartographer 中定义的格式传递给 cartographer 处理，与此同时也将 cartographer 的处理结果发布用于显示或保存，是基于 cartographer 的上层应用。



cartographer 部分源码主要分为如上图所示几个文件加，其中一些文件的功能如下：

1. common：定义了基本数据结构和一些工具的使用接口。例如，四舍五入取整的函数、时间转化相关的一些函数、数值计算的函数、互斥锁工具等。
2. sensor：定义了传感器数据的相关数据结构。
3. transform：定义了位姿的数据结构及其相关的转化。如 2d\3d 的刚体变化等。
4. mapping：定义了上层应用的调用接口及局部 submap 构建和基于回环检测的位姿优化等接口。这个文件也是算法的核心。其中 mapping_2d 和 mapping_3d 是对 mapping 接口的不同实现。

5. io：定义了一些与 IO 相关的工具，用于存读取数据、记录日志等。详情请参考 io 部分源码分析。



在运行 cartographer_ros 建图算法时，需要启动两个 cartographer 的节点，第一个是 cartographer_node 节点，该节点为 cartographer 算法核心节点，订阅了 scan 和 odom 数据并输出 submap_list 数据，第二个节点是 cartographer_occupancy_gird_node 节点，将 cartographer 输出的 submap_list 融合成 ROS 标准的 map 格栅地图数据话题并发布。

2.2.1 cartographer_node 订阅的话题内容

- echoes(sensor_msgs/MultiEchoLaserScan)

  多回波雷达数据

- scan(sensor_msgs/LaserScan)

  用于创建地图的激光雷达数据

- points2(sensor_msgs/PointCloud2)

  点云传感器数据

- imu（sensor_msgs/Imu）

  IMU 传感器数据

- odom（nav_msgs/Odometry）

里程计传感器数据

2.2.2 cartographer_node 发布的话题内容

- scan_matched_points（sensor_msgs/PointCloud2）

  匹配好的 2D 点云数据，用来 scan-to-submap matching

- submap_list（cartographer_ros_msgs/SubmapList）

  发布构建好的 submap

2.2.3 cartographer_node 提供的 Service

- submap_query（cartographer_ros_msgs/SubmapQuery）

  提供查询 submap 的服务，获取到 request 的 submap

- star_trajectory（cartographer_ros_msgs/StartTrajectory）

  开始维护一条轨迹

- finish_tracjectory（cartographer_ros_msgs/FinishTrajectory）

  结束一个给定 ID 的轨迹

- write_state(cartographer_ros_msgs/WriteState)

  将当前状态写入文件保存

- get_trajectory_states（cartographer_ros_msgs/GetTrajectoryStates）

  获取指定 trajectory 的状态

2.2.4 cartographer_occupancy_grid_node 订阅的话题内容

- submap_list（cartographer_ros_msgs/Submaplist）

  由 cartographer_node 提供的 submap 数据

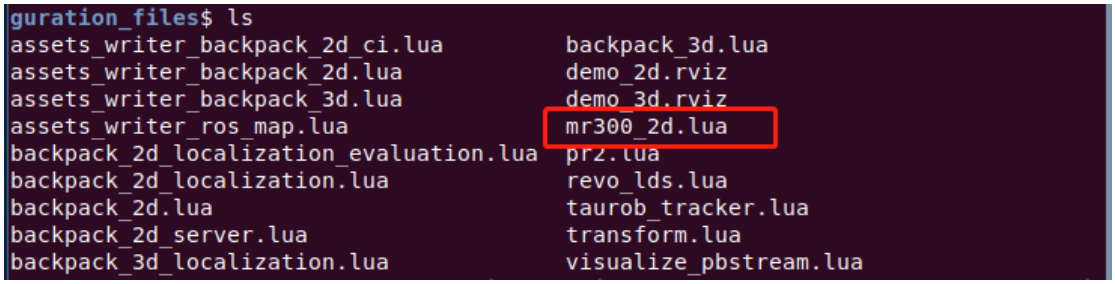
2.2.5 cartographer_occupancy_grid_node 发布的话题内容

- map（nav_msgs/OccupancyGrid）

  ROS 标准格式的格栅地图

2.3 cartographer 建图算法参数

在 BR280 机器人中，cartographer 的配置参数文件为 mr300_2d.lua，可以使用 roscd cartographer_ros/configuration_files/ 命令直接进入该文件夹。



配置文件内容如下图：

```
include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "imu_link",
  published_frame = "base_link",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = false,
  use_pose_extrapolator = on,
  use_odometry = false,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}

MAP_BUILDER.use_trajectory_builder_2d = true

TRAJECTORY_BUILDER.collate_landmarks = on

TRAJECTORY_BUILDER_2D.submaps.num_range_data = 35
TRAJECTORY_BUILDER_2D.min_range = 0.15
TRAJECTORY_BUILDER_2D.max_range = 8.
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 1.
TRAJECTORY_BUILDER_2D.use_imu_data = true
TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = true
TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.linear_search_window = 0.1
TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.translation_delta_cost_weight = 10.
TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.rotation_delta_cost_weight = 1e-1

POSE_GRAPH.optimization_problem.huber_scale = 1e2
POSE_GRAPH.optimize_every_n_nodes = 35
POSE_GRAPH.constraint_builder.min_score = 0.65

return options
```

重点参数如下：

map_frame = "map", //地图坐标系名称

tracking_frame = "imu_link", //路径追踪的坐标系的名称

published_frame = "base_link", //发布坐标系变换的坐标系名称

odom_frame = "odom", //里程计坐标系名称

provide_odom_frame = false, //是否发布里程计坐标系变换

publish_frame_projected_to_2d = false, //如果启用，发布纯 2d 姿势

use_pose_extrapolator = on, //是否启用位姿推测器

use_odometry = false, //是否使用 odom 数据

use_nav_sat = false, //如果启用，需要在话题“fix”上订阅 sensor_msgs/NavSatFix。在这种情况下必修提供 Navigation data，并且信息将包含到全局 SLAM 中

use_landmarks = false, //如果启用，则在主题 Landmarks 上订阅 cartographer_ros_msgs/LandmarkList 数据，在这种情况下必修提供 Landmarks，并且信息将包含在 SLAM 中。

num_laser_scans = 1, //订阅的激光雷达主题数量

num_multi_echo_laser_scans = 0, //订阅的多回波雷达主题的数量

num_subdivisions_per_laser_scan = 1, //将每个接收到的（多回波）激光扫描分成的点云数

num_point_clouds = 0, //订阅的点云主题数量

lookup_transform_timeout_sec = 0.2, //使用 TF 查找变换的超时时间

submap_publish_period_sec = 0.3, //发布子图的时间间隔

pose_publish_period_sec = 5e-3, //发布 pose 的时间间隔，5e-3 为 200Hz

trajectory_publish_period_sec = 30e-3, //发布轨迹标记的间隔，30e-3 为 30 毫秒

rangefinder_sampling_ratio = 1., //激光雷达消息的固定采样频率

odometry_sampling_ratio = 1., //里程计消息的固定采样频率

fixed_frame_pose_sampling_ratio = 1., //固定坐标系消息的固定采用频率

imu_sampling_ratio = 1., //IMU 消息的固定采样频率

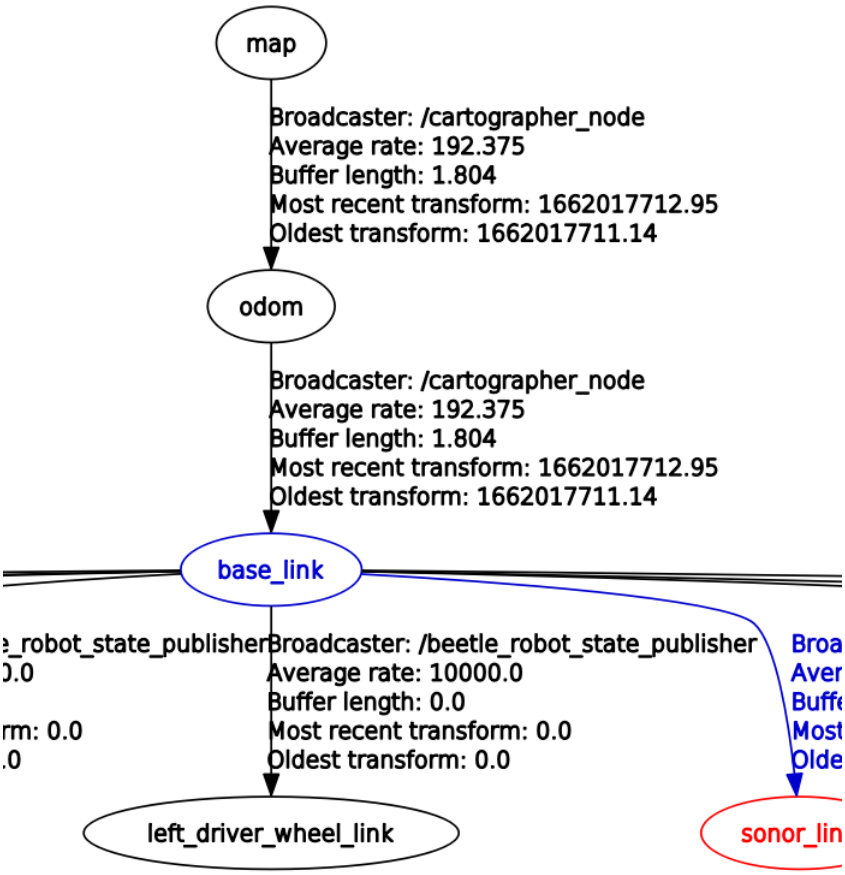
landmarks_sampling_ratio = 1.， //路标消息的固定采样频率

Cartographer 可配置的参数比较多，在此不一一列举，如果要详细学习 cartographer 算法，可以参阅下面链接，通过官方文档学习：

Cartographer Github

Cartographer 官方 Doc

2.4 Cartographer 的 TF 坐标发布



由图中可以看到，cartographer 节点发布了从 map 到 odom 的坐标变换，这就意味着 cartographer 节点提供了机器人在 map 坐标系中的定位。

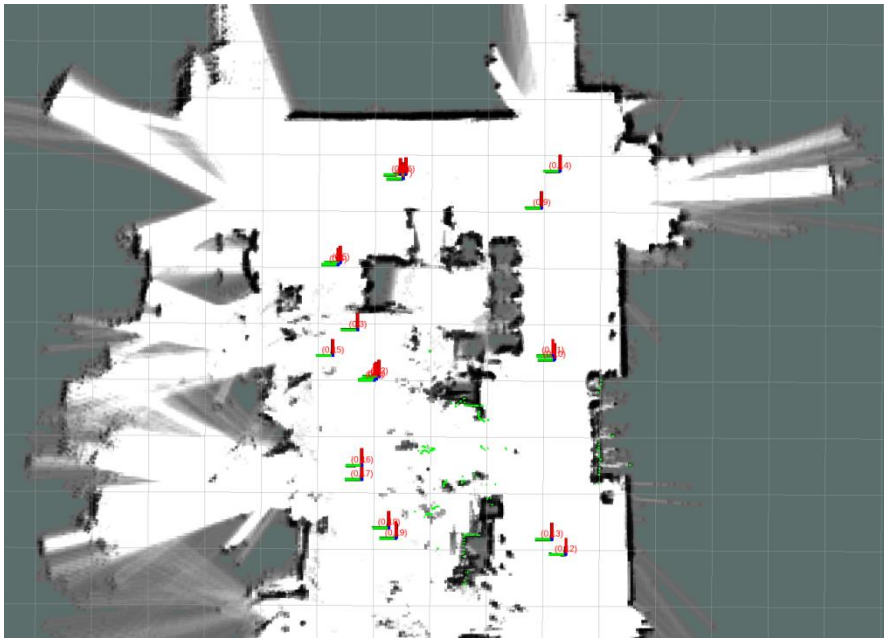
3. Cartographer 建图实践

1、输入以下指令，开始建图：

```
roslaunch robuster_mr_navigation mapping.launch
```

通过遥控器控制 BR280 移动开始建图，注意：建图时要将移动和旋转速度调至最小，否则会导致建图失败。

先原地旋转 2-3 圈，然后每隔 2 米左右旋转 2-3 圈。具体情况根据实际环境而定。如下图中红绿线相交的地方就是一个个子图，每张地图需要建立三个以上的子图才能建图成功。



2、保存地图

地图保存在/home/robuster/Downloads 目录下。

首先打开一个终端输入下面命令。

```
rosservice call /finish_trajectory 0
```

// 停止接收传感器数据

然后在输入下面命令生成 pbstream 文件。

```
rosservice call /write_state "{filename: '${HOME}/Downloads/mymap.pbstream'}"
```

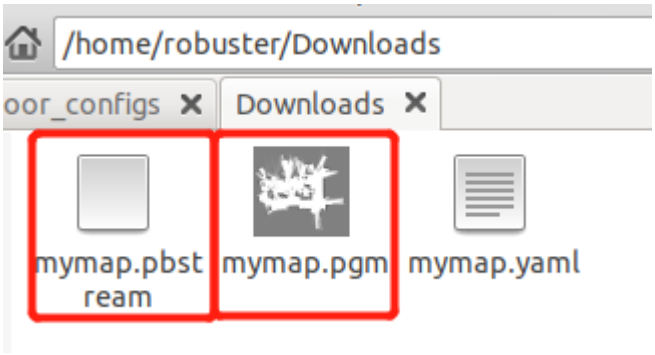
//生成.pbstream 文件

最后输入格式转换命令。

```
rosrun cartographer_ros cartographer_pbstream_to_ros_map
-map_filestem=${HOME}/Downloads/mymap
-pbstream_filename=${HOME}/Downloads/mymap.pbstream -resolution=0.05
```

//地图格式转换：由于用 cartographer_ros 提供的/write_state 方法保存的地图是.pbstream 的格式，而要在后续的自主导航中使用这个地图，我们需要将其转换为 ROS 中通用的 GridMap 格式。cartographer_ros 已经为我们提供了

cartographer_pbstream_to_ros_map 这个节点用于转换的实现。参数 map_filestem，为转换后存放结果的文件路径参数；pbstream_filename 为待转换的.pbstream 文件路径，参数 resolution 为分辨率。




```
^Crobuster@robuster:~$ rosservice call /finish_trajectory 0
status:
  code: 0
  message: "Finished trajectory 0."
robuster@robuster:~$ rosservice call /write_state "{filename: '${HOME}/Downloads
/mymap.pbstream'}"
status:
  code: 0
  message: "State written to '/home/robuster/Downloads/mymap.pbstream'."
robuster@robuster:~$ rosrun cartographer_ros cartographer_pbstream_to_ros_map -m
ap_filestem=${HOME}/Downloads/mymap -pbstream_filename=${HOME}/Downloads/mymap.p
bstream -resolution=0.05
I1223 15:31:23.052848 10112 pbstream_to_ros_map_main.cc:43] Loading submap slice
s from serialized data.
I1223 15:31:23.669100 10112 pbstream_to_ros_map_main.cc:51] Generating combined
map image from submap slices.
```

保存完成后，在建图的终端按下 Ctrl+C 键，退出建图。

4. 总结

本节主要介绍了 Cartographer 算法建图内容，关于其他的建图算法可查阅对应的 WIKI 百科。

gmapping - ROS Wiki

hector_slam - ROS Wiki

rgbdslam - ROS Wiki

深圳史河机器人科技有限公司

深圳史河机器人科技有限公司
www.robotplusplus.com.cn
深圳市龙华区龙华街道清华社区建设东路青年创业园C栋4层412-415房