ROS 自主清洁机器人

一、 设计思路

清洁机器人是一种家用服务型移动机器人。完成移动机器人的自主智能导航，通常需要解决建图，定位，导航以及运动控制四大问题，来实现机器人的对环境智能感知以及自主决策。

由于测试环境较为狭小，这里选用 turtlebot3 系列机器人中的个头最小的 burger 机器人作为测试用机器人。如图 1 所示。

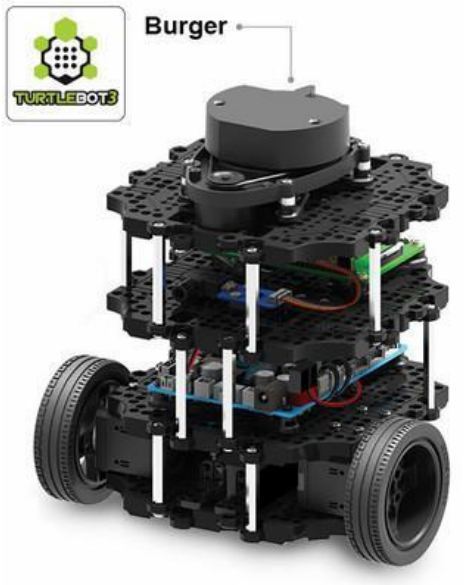


图 1 burger 机器人

在定位系统上选择 AMCL（粒子滤波）作为定位方法。

在建图上选择 karto slam 算法作为初次地图构建方法。

在导航规划上采用全覆盖路径规划算法（Complete Coverage Path Planning）作为第一次规划结果，之后将其规划的路径点按照距离依次发给机器人导航系统，控制机器人执行。

二、 文件结构与运行方法

我的 clean_robot 包核心整体结构中 src 为源程序

CleaningPathPlanner.cpp/h 为路径规划核心程序，

next_goal.cpp 为发布下一个目标点的源程序，

PathPlanningNode.cpp 为对路径规划的封装。

整个包会形成两个节点 path_planning_node 与 next_goal 分表示全覆盖路径规划器与目标点发送程序。

Launch 文件中

clean_work.launch 表示启动清洁工作，

auto_slam.launch 表示全自主探索建图（全自动不需要人干预）

nav_slam.launch 表示导航建图（也可以使用键盘控制建图），

gazebo.launch 表示启动仿真环境，

move_base.launch 表示机器人的导航栈配置，

amcl.launch 表示粒子滤波定位系统，

turtlebot3_navigation.launch，表示机器人导航系统

Config 为路径规划所使用参数的 yaml 文件，param 为 move_base 所采用的参数文件，maps 表示地图文件，worlds 为仿真环境。

运行方法

启动清洁　　roslaunch clean_robot clean_work.launch

启动全自主探索建图 roslaunch clean_robot auto_slam.launch

启动导航建图 roslaunch clean_robot nav_slam.launch

三、 在 Gazebo 中搭建仿真环境

由于在家环境受到限制，采用 Gazebo 仿真的形式来实现清洁机器人的构建与测试。按照题中提供的图 2a 的平面图。按要求在 gazebo 的 building edit 功能中搭建 6m*4m 的仿真环境，包含墙板和障碍物，搭建如下图 2b 的仿真环境。

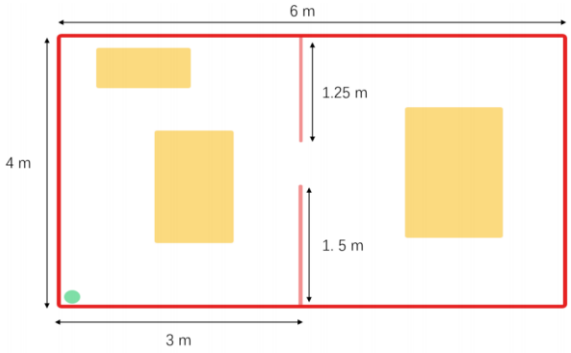


图 2a 平面图

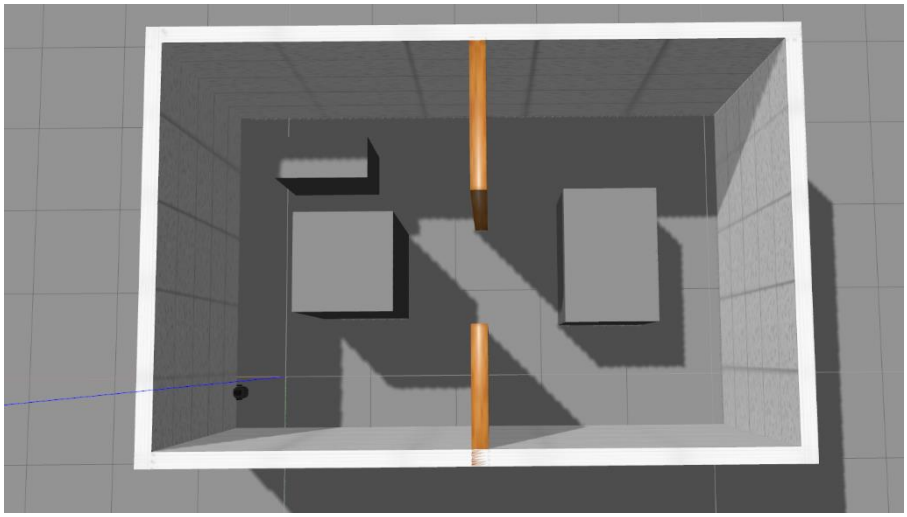


图 2b 仿真环境展示

之后将文件保存于我创建的包下 worlds 文件夹。并编写仿真环境启动 launch 文件，命名为 gazebo.launch。其主要流程为启动 gazebo，并加载 world 文件，之后加载机器人描述文件 xacro，最后将机器人放入 gazebo 仿真环境中。其中位置由 x_pos, y_pos 决定。

```
<launch>
  <arg name="model" default="burger" doc="model type [burger, waffle, waffle_pi]"/
>
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find clean_robot)/worlds/clean_room.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --
inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -
model turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -
param robot_description" />
</launch>
```

四、 基于局部已知环境导航的激光 SLAM 建图

地图是机器人预先知道的基本环境先验信息。机器人要想在后续环境中精准的定位需要有一个准确的环境地图，同时路径规划系统也要基于栅格地图进行规划。因此我们需要提前构建一张精准的环境地图。

此处我们选用 kartoSLAM 系统，该系统是基于图优化的 SLAM 算法利用图的均值表示地图，每个节点表示机器人轨迹的一个位置点和传感器测量数据集，箭头的指向的连接表示连续机器人位置点的运动，每个新节点加入，地图就会依据空间中的节点箭头的约束进行计算更新。相较于基于粒子滤波的 gmapping 算法，其构建地图过程中能对整个 graph 进行优化，减小滤波算法中由于一次的观测误差导致后续的误差积累。由于键盘操作控制机器人建图效果不佳。因此我配置了导航节点，可以实现通过设定目标点（目标点不能超出当前的地图），来实现自动运行建图。

为了方便地图构建，我将所有的系统启动的节点写入一个名为 nav_slam.launch 文件。

```
 <?xml version="1.0"?>
<launch>

  <!-
- <arg name="slam_methods" default="karto" doc="slam type [gmapping, cartographer,
 hector, karto, frontier_exploration]"/> -->
  <arg name="model" value="burger" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="open_rviz" default="true"/>
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>
  <!-- 启动仿真环境 -->
  <include file="$(find clean_robot)/launch/gazebo.launch"/>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration, RTAB-
Map -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_karto.launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="$(arg configuration_basename)"/>
  </include>

  <!-- 启动 rviz 的标签 -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true" args="-
d $(find clean_robot)/rviz/turtlebot3_karto.rviz"/>
  </group>
  <!-- 通过导航来实现自动建图 -->
  <include file="$(find turtlebot3_navigation)/launch/move_base.launch"/>
  <node pkg="turtlebot3_teleop" type="turtlebot3_teleop_key" name="turtlebot3_tele
op_key" launch-prefix="xterm -e" output="screen"/>

</launch>
```

为了避免共享控制台界面导致看不到键盘操作提示界面，使用 `launch-prefix="xterm -e"`在单独终端里面打开控制器。

使用导航方式建图，绿色为全局路径，蓝色为局部路径（也可使用键盘控制建图）

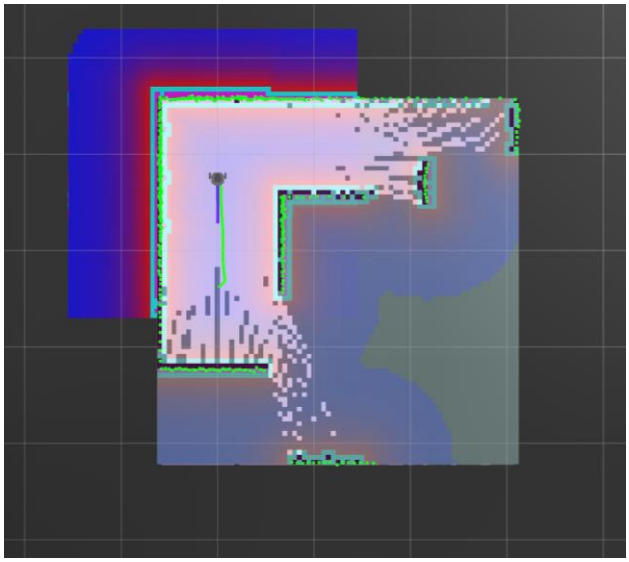


图 3 导航建图过程

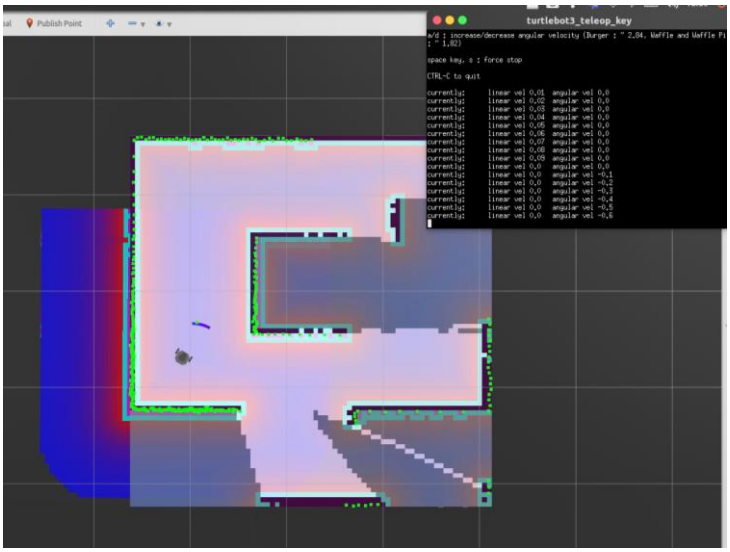


图 4 键盘控制建图

五、 基于全覆盖路径规划的自主清洁机器人

这一部分我主要采用全覆盖路径规划（Complete Coverage Path Planning）的思想，首先将原有的栅格地图无障碍物区域按照机器人大小进行分割，分割后直接对所有分割区域进行遍历即可。（按照一定规则直接进行遍历，不能保证路径最优，可以实现相对较优，完成基本功能）。

1.如果机器人前方已经被覆盖过了或者是障碍,那么左转或者右转旋转 180 度接着走，实现 S 型走，如图 5 所示。

2.如果机器人的周围都被覆盖过了，以当前机器人的位置为起点，未清洁区域作为目标点，用 A*算法找出路径。重复 1，2
3.如果没有未清洁区域，算法结束

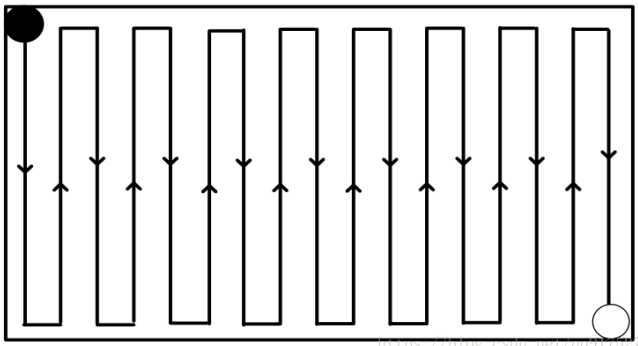


图 5 全覆盖路径规划示意图

为了求取更优的解，也可以将这个问题视作一个商旅问题，即将所有的分割区域视作待访问节点，求取经过所有节点的最短路径，这种方法规划的路径更优，能够实现最短路径，通常采用回溯方法来实现求解。
为了实现动态障碍物避障与突变点的路径规划，我采用 move_base 导航栈作为控制机器人追踪全覆盖路径的模块。系统结构图如图 6 所示。

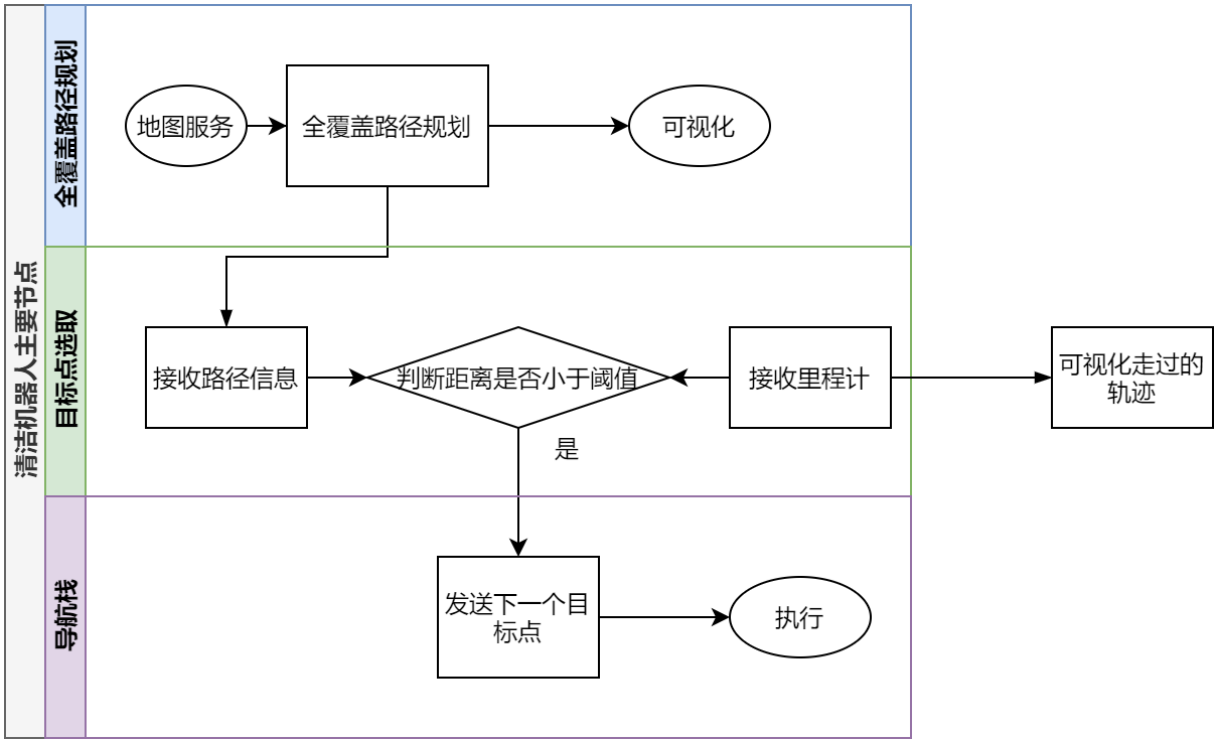


图 6 系统框架结构图

为实现全覆盖路径规划，我参考现有基本的全覆盖路径规划算法[1]，实现基本的路径规划节点（path_planning_node），首先先对地图按照机器人大小进行分割[2]，然后按照固定规则进行遍历。但是由于规划完之后路径是地图分割后的中心点位置，每两个点仅相隔一个机器人的大小，且只有到达时候才会发送下一个路径规划点，导致机器人容易走走停停，清扫效率较低。

因此我参考纯追踪算法（pure pursuit 之前我在做阿克曼模型车的路径追踪问题时候写过这个控制器）的思想，自行编写了一个目标点位发送程序。实现目标点的动态发送。实现思路为根据当前的机器人里程计反馈回来的位姿数据，计算机器人到当前目标点的欧式距离，当小于设定阈值（/NextGoal/tolerance_goal）时候，发送下一个点（/move_base_simple/goal）。该程序接收两个话题，一个是机器人的里程计数据（/odom），一个是路径规划节点规划好的路径（/path_planning_node/cleaning_plan_nodehandle/cleaning_path）。在路径回调函数中会对接受的路径进行判断，当路径长度为 0 或者长度没有发生变化时候，认为还是上个路径，不对系统中存储的路径进行更新，反之则更新路径，并从头追踪。同时为了实现在 rviz 中显示路径，我将里程计每次反馈的位姿数据添加到一条路径消息中（passed_path），然后发布出去，实现可视化。

我的系统数据传递关系如图所示

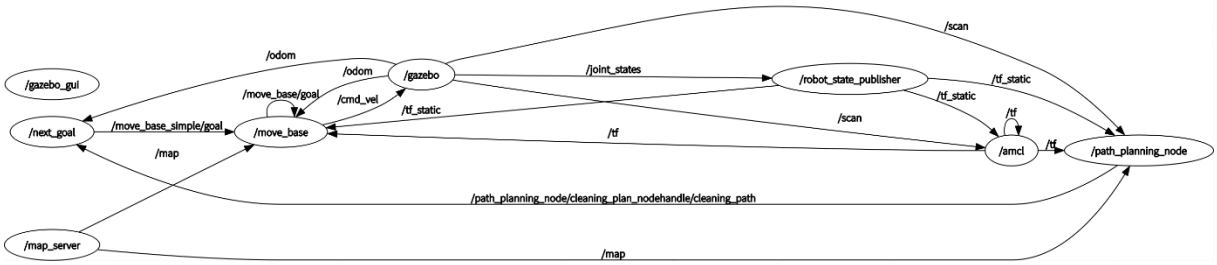


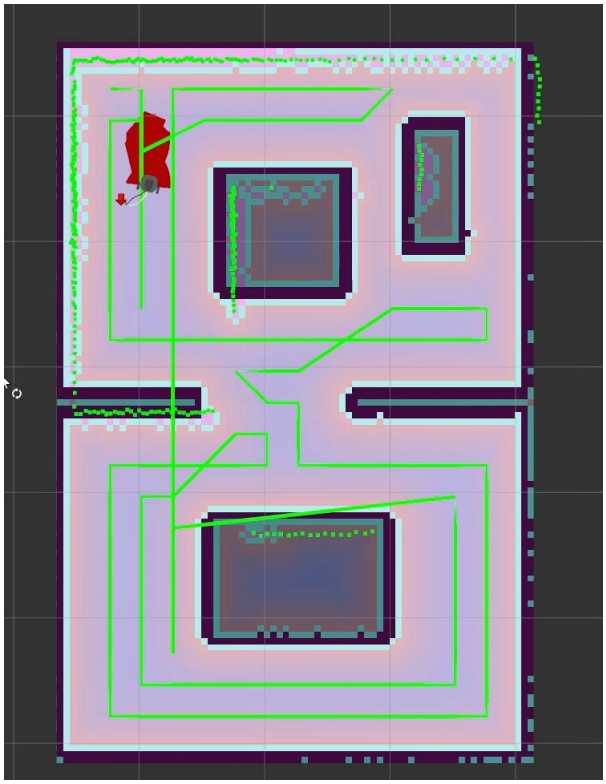
图 7 话题与节点图

六、 最终效果展示

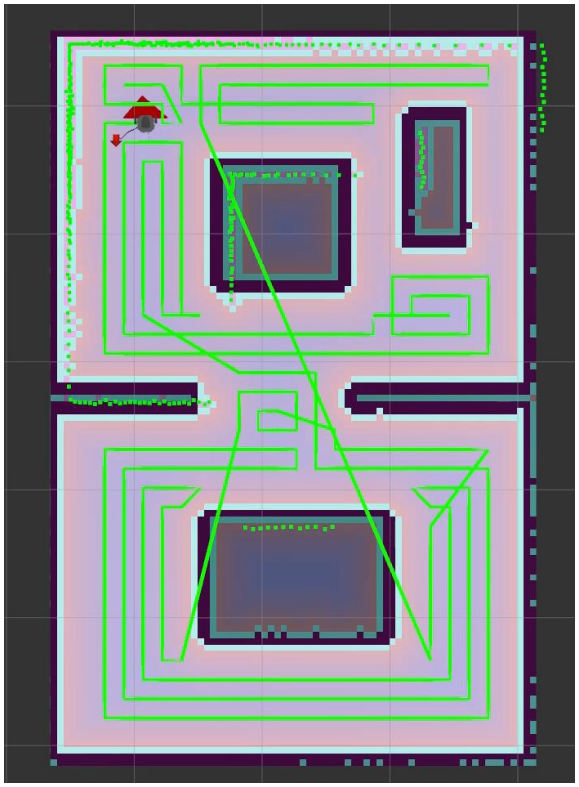
设置规划稀疏一点快速清扫（清扫开始） 设置规划密集一点细致清扫（清扫开始）

[1] https://github.com/hjr553199215/SLAM-Clean-Robot-Path-Coverage-in-ROS

[2] 通过 size_of_cell 参数可以实现控制规划的路径的密集程度，必须为奇数，越小越密，这个值代表机器人在栅格地图中的大小。

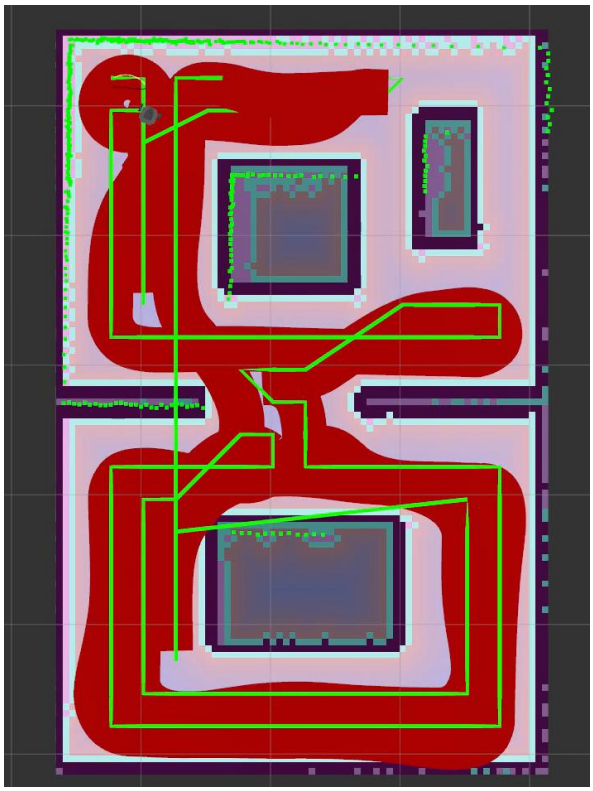


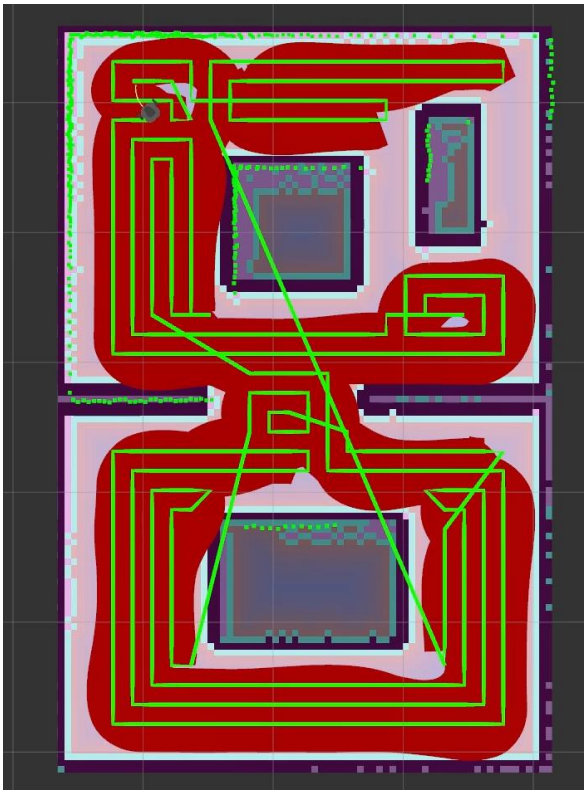
设置规划稀疏一点快速清扫（清扫开始）



设置规划密集一点细致清扫（清扫开始）

清扫完成

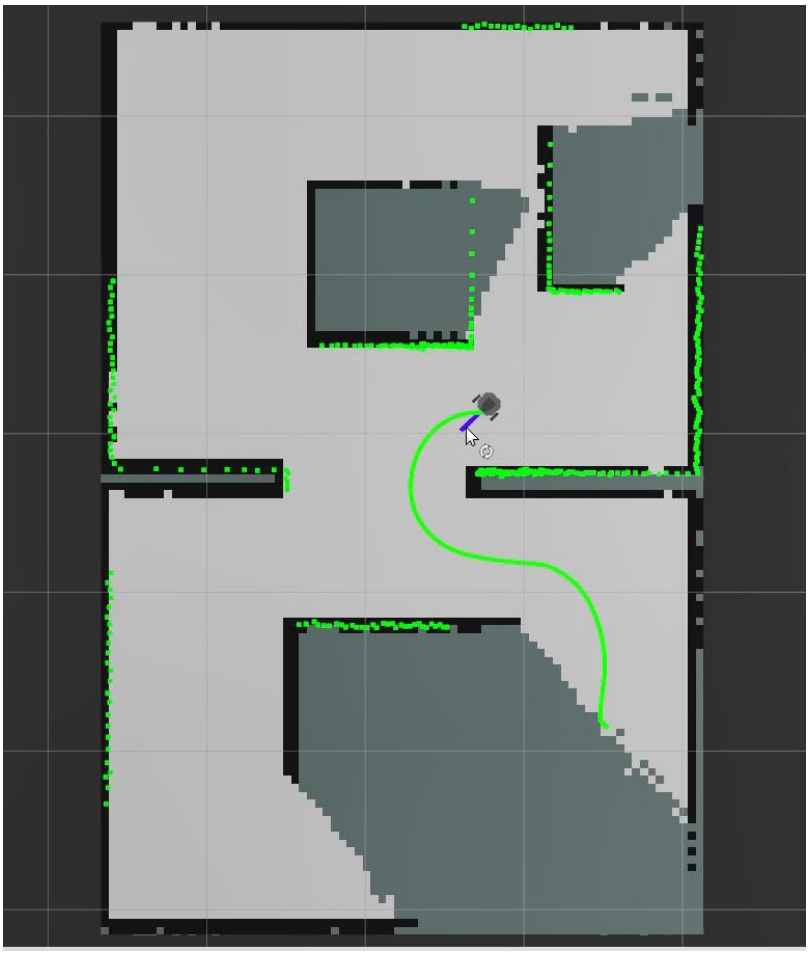




清扫完成

七、 拓展任务

考虑到真正的清洁机器人不可能让用户操作建图，因此尝试使用**自主探索建图**。采用开源的 explore_lite 包实现自主探索建图，其思路为找到可以达到的地图中的位置区域的边界中点作为导航的目标点。该方法在封闭环境中十分迅速！强烈建议使用这个建图。建图过程仅需要几分钟就好了。图中目标点的提取为全自动的方式，十分便捷。



全自主探索建图