# **M05算法技术报告**

## **环境**

* **操作系统:Ubuntu22.04**
* **ROS2 version:Humble**

## **用到的技术概述**

* **Navigation2导航框架**
* **Gmapping建图算法**
* **nav2\_amcl定位算法**
* **dwb局部路径规划算法**
* **ROS2主从机**
* **ldlidar\_sl\_ros2雷达驱动**
* **rf2o\_laser\_odometry\_ros2激光里程计**
* **串口发送速度信息**

## **源代码地址**

1. **Navigation2**[**https://github.com/ros-planning/navigation2.git**](https://github.com/ros-planning/navigation2.git)
2. **slam\_gmapping**[**https://github.com/Project-MANAS/slam\_gmapping.git**](https://github.com/Project-MANAS/slam_gmapping.git)
3. **ldlidar\_sl\_ros2**[**https://github.com/ldrobotSensorTeam/ldlidar\_sl\_ros2.git**](https://github.com/ldrobotSensorTeam/ldlidar_sl_ros2.git)
4. **rf2o\_laser\_odometry\_ros2**[**https://github.com/linuxsen/rf2o\_laser\_odometry\_ros2.git**](https://github.com/linuxsen/rf2o_laser_odometry_ros2.git)

## **参数配置**

**#提前说明：只会涉及到有改动的以及关键部分（可能会有遗漏）**

1. **机器人的各坐标关系urdf  
   放在{workspace}/src/navigation2/nav2\_bringup/urdf下，根据实际机器人各部位的测量的相对位置进行撰写，因代码过长就不放出来了。**

**本次写到的主要link有base\_link, base\_footprint, laser\_link, left\_front\_wheel\_link, left\_rear\_wheel\_link, right\_front\_wheel\_link, right\_rear\_wheel\_link.**

|  |
| --- |
| **其关系如下图所示** |
| 图1 |

1. **gmapping配置  
   #{workspace}/src/slam\_gmapping/slam\_gmapping/launch/slam\_gmapping.launch.py**

**use\_sim\_time的值改为false #不只是gmapping中为false，所有用到的包的use\_sim\_time都要改为false，因为目的不是跑仿真**

|  |
| --- |
| {'base\_frame\_': 'base\_link'}, # 基座坐标系  {'map\_frame\_': 'map'}, # 地图坐标系  {'odom\_frame\_': 'odom'}, # 里程计坐标系 |

**#所有用到的包的坐标系都要以图1为基准设置**

**建完图保存后会得到 .pgm 和 .yaml 后缀的两个文件，其中.pgm后缀的是地图，.yaml后缀的是地图信息文件。建议放到{workspace}/src/navigation2/nav2\_bringup/map下**

1. **ldlidar\_sl\_ros2配置**

**#根据具体雷达型号选择合适的launch文件，本次选用的是ld14.launch.py**

|  |
| --- |
| {'laser\_scan\_topic\_name': 'scan'}, #雷达发布的话题名称  {'frame\_id': 'laser\_link'}, #以图1为基准  {'port\_name': '/dev/ttyUSB0'}, #串口名称，可能会根据插入的设备顺序有所不同 |

**因为已经在urdf文件中发布了base\_link和laser\_link的关系，所以需要将launch文件中发布这两个link静态关系的部分注释掉，  
即# base\_link to base\_laser tf node部分**

**在运行此程序之前，需要先给串口权限，命令如下  
sudo chmod 777 /dev/ttyUSB0  
#注意串口名称**

1. **rf2o\_laser\_odometry\_ros2配置**

**#rf2o\_laser\_odometry.launch.py**

|  |
| --- |
| {  'laser\_scan\_topic' : '/scan', #scan话题，用于接收雷达数据，与驱动中发布的话题一致  'odom\_topic' : '/odom', #里程计话题，发布里程计信息  'publish\_tf' : True,  'base\_frame\_id' : 'base\_footprint', # 发布base\_footprint坐标系，代表机器人的足迹  'odom\_frame\_id' : 'odom', #发布odom坐标系，代表里程计  'init\_pose\_from\_topic' : '',  'freq' : 20.0} #发布频率，不能设置太低，太低会导致无法发布局部代价地图 |

1. **navigation2下nav2\_bringup的配置**

**#bringup\_launch.py**

**启动时会读取{workspace}/src/navigation2/nav2\_bringup/params/nav2\_params.yaml参数文件**

**在启动该launch文件时 需要在命令的末尾加上地图的.yaml路径,例如**

**ros2 launch nav2\_bringup bringup\_launch.py map:=src/navigation2/nav2\_bringup/maps/turtlebot3\_world.yaml**

**也可以直接在源代码中添加，但每次变动都需要重新编译，例如**

|  |
| --- |
| declare\_map\_yaml\_cmd = DeclareLaunchArgument(  'map',  default\_value='/home/zllc02/navigation2\_ws/map/map\_9021.yaml', #地图路径  description='Full path to map yaml file to load') |

**#rviz\_launch.py**

**启动该程序以使用rviz2发布定位和目标点**

**#nav2\_params.yaml**

**注意该参数文件下个坐标系和话题的名称一定要与图1相匹配，各use\_sim\_time的值改为false #以下并非全部代码，仅为部分**

|  |
| --- |
| amcl: #amcl定位参数  ros\_\_parameters:  use\_sim\_time: False  base\_frame\_id: "base\_link"  global\_frame\_id: "map"  odom\_frame\_id: "odom"  scan\_topic: scan  bt\_navigator: #行为树  ros\_\_parameters:  use\_sim\_time: false  global\_frame: map  robot\_base\_frame: base\_link  odom\_topic: /odom  controller\_server:  ros\_\_parameters:  # DWB parameters  FollowPath:  plugin: "dwb\_core::DWBLocalPlanner"  debug\_trajectory\_details: True  min\_vel\_x: 0.0  min\_vel\_y: 0.0  max\_vel\_x: 0.26 #最大线速度0.26  max\_vel\_y: 0.0  max\_vel\_theta: 1.0 #最大角速度1.0  local\_costmap: #局部代价地图  local\_costmap:  ros\_\_parameters:  update\_frequency: 5.0  publish\_frequency: 2.0  global\_frame: odom  robot\_base\_frame: base\_link  use\_sim\_time: False  inflation\_layer: #膨胀层  plugin: "nav2\_costmap\_2d::InflationLayer"  cost\_scaling\_factor: 3.0  inflation\_radius: 0.3 #膨胀半径，用于避障，根据实际情况设置，此处为0.3m  voxel\_layer:  plugin: "nav2\_costmap\_2d::VoxelLayer"  enabled: True  publish\_voxel\_map: True  origin\_z: 0.0  z\_resolution: 0.05  z\_voxels: 40  min\_obstacle\_height: 0.01  max\_obstacle\_height: 2.0  mark\_threshold: 0  observation\_sources: scan  scan:  topic: /scan #接收的话题名称  min\_obstacle\_height: 0.01  max\_obstacle\_height: 2.0  clearing: True  marking: True  data\_type: "LaserScan"  raytrace\_max\_range: 3.0  raytrace\_min\_range: 0.0  obstacle\_max\_range: 2.5  obstacle\_min\_range: 0.0  global\_costmap: #全局代价地图  global\_costmap:  ros\_\_parameters:  update\_frequency: 1.0  publish\_frequency: 1.0  global\_frame: map  robot\_base\_frame: base\_link  use\_sim\_time: False  robot\_radius: 0.22 #机器人半径，根据实际需求设置  resolution: 0.05  track\_unknown\_space: true  behavior\_server:  ros\_\_parameters:  global\_frame: odom  robot\_base\_frame: base\_link  velocity\_smoother:  ros\_\_parameters:  use\_sim\_time: False  odom\_topic: "/odom" |

## **为什么使用主从机**

* **校内赛所使用的miniPC为n3700开发板，内存为4g。原计划使用nomachine进行远程桌面操控，但由于所启动的程序+nomachine远程服务内存消耗太大，经常出现nomachine服务挂掉而远程无法连接的问题，导致无法操控miniPC，尤其在程序运行时具有很大的不确定性，这是比赛中我们所不希望看见的。**
* **利用主从机技术可以减少miniPC的运算量，大大减少与miniPC断连的可能性。  
  将主机设置为miniPC，自己的电脑设置为从机。miniPC可以节省掉图形计算这一部分的算力，甚至可以不用插hdmi诱骗器(完全不需要图象输出，连桌面都不需要)，因为图形计算这部分被转移到从机上了。只需在从机上ssh连接主机，并利用命令行控制主机即可。既减少了内存开销，又节省了图形计算开销，又减小了断连的可能性，这主从机多是一件美事啊！**

## **主从机具体配置方法 #主机和从机在统一局域网下**

* **主机部分(miniPC)：  
   1.控制台输入hostname获取hostname  
   输入ifconfig获取本机ip  
   2.打开~/.bashrc  
   gedit ~/.bashrc  
   3.在末尾添加**

|  |
| --- |
| export ROS\_HOSTNAME=主机hostname  export ROS\_MASTER\_URI=[http://](http://192.168.9.163:11311/)主机ip:11311  export ROS\_IP=主机ip |

**4.控制台输入sudo gedit /etc/hosts  
 在末尾添加**

|  |
| --- |
| 从机IP 从机hostname |

* **从机部分(自己电脑)：  
   1.控制台输入hostname获取hostname  
   输入ifconfig获取本机ip  
   2.打开~/.bashrc  
   gedit ~/.bashrc  
   3.在末尾添加**

|  |
| --- |
| export ROS\_HOSTNAME=从机hostname  export ROS\_MASTER\_URI=[http://](http://192.168.9.163:11311/)主机ip:11311  export ROS\_IP=主机ip |

**4.控制台输入sudo gedit /etc/hosts  
 在末尾添加**

|  |
| --- |
| 主机IP 主机hostname |

* **测试**

**主机运行**

**ros2 run turtlesim turtlesim\_node**

**从机运行  
ros2 run turtlesim turtle\_teleop\_key**

**在从机上若能成功操作主机的turtle则证明配置成功**

## **串口发送数据程序**

**主要思路：**

* **创建一个节点接收nav2发出的/cmd\_vel话题，其中为速度信息，再将角速度和线速度信息发送给下位机。**
* **在发送数据时，对速度作了处理，因为从/cmd\_vel话题接收到的速度值有正负，为了便于发送，所以在发送数据时对速度加上了个基准值使发送的数据都为正数，下位机接收到数据时再减去基准值即可。**

**遇到的问题：**

* **在插上串口时使用命令  
  ls /*dev/*tty\*  
  会出现找不到串口的问题**
* **解决方法：  
  sudo apt remove brltty**

## **小技巧**

* **在bashrc中添加source命令可省掉每启动一个终端都要source的步骤**
* **给串口永久权限可以省掉启动雷达驱动和串口发送程序前给权限的步骤**