项目使用手册

1 使用流程

1.1 环境配置

项目运行的虚拟机环境为 ubuntu16.04，ros 版本为 kinetic。

1.2 项目运行

将源代码 racecar_ws 拷贝到虚拟机的 home 目录下。

初始化工作空间。

```
cd ~/racecar_ws/src
catkin_init_workspace
```

安装相关依赖文件。

```
sudo apt-get install ros-kinetic-driver-base
sudo apt-get install ros-kinetic-ackermann-msgs
sudo apt-get install ros-kinetic-controller-manager
sudo apt-get install ros-kinetic-gazebo-ros-control
sudo apt-get install ros-kinetic-effort-controllers
sudo apt-get install ros-kinetic-joint-state-controller
sudo apt-get install ros-kinetic-rtabmap-ros
sudo apt-get install ros-kinetic-move-base
sudo apt-get install ros-kinetic-map-server
sudo apt-get install ros-kinetic-teb-local-planner
sudo apt-get install ros-kinetic-global-planner
sudo apt-get install ros-kinetic-dwa-local-planner
```

编译源代码。

```
cd ~/racecar_ws
catkin_make
```

设置环境变量。

```
echo ”source ~/racecar_ws/devel/setup.bash” >> ~/.bashrc
```

打开新的终端启动 ros 环境。

```
roscore
```

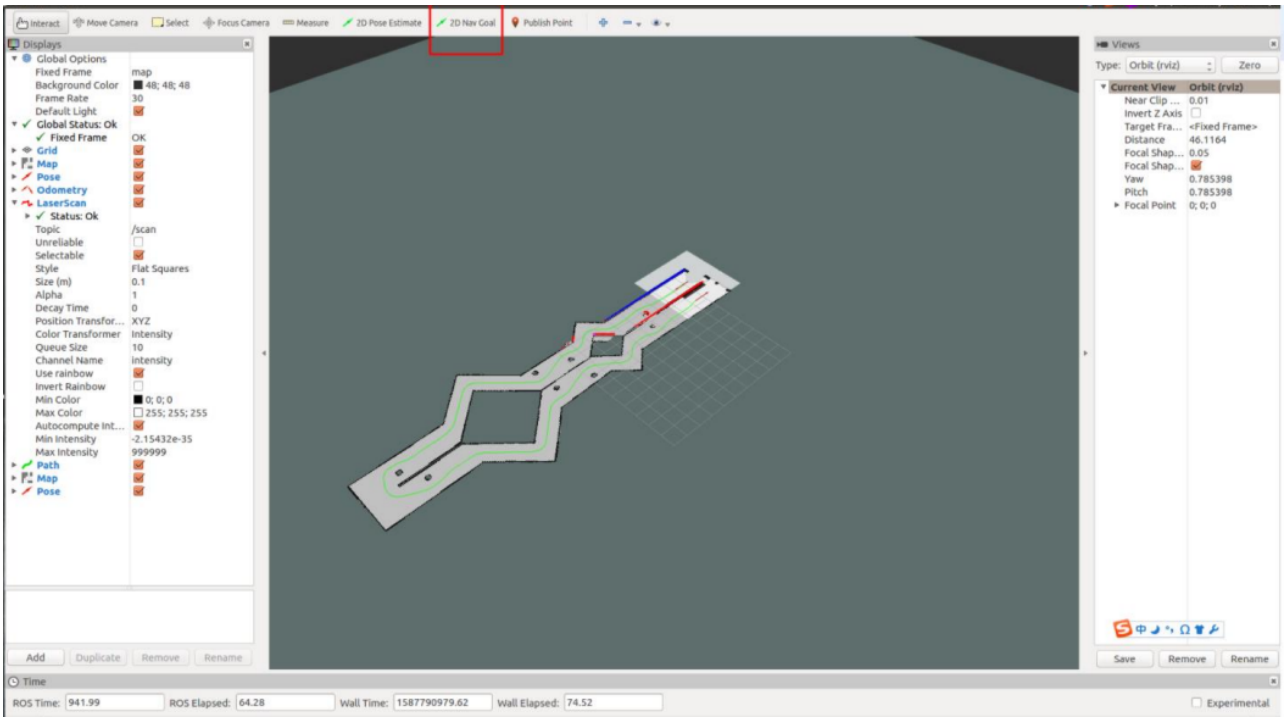
打开另一个终端启动 gazebo，gazebo 启动可能需要一小会，此时不要进行其他操作，不然 gazebo 容易崩溃

```
cd ~/racecar_ws/
roslaunch racecar_gazebo racecar_runway_navigation.launch
```

gazebo 启动成功并显示模型后， 打开新的终端窗口启动 rviz

```
cd ~/racecar_ws/
roslaunch racecar_gazebo racecar_rviz.launch
```

用 2D Nav Goal 发布目标，点击如下图按钮，并在 rviz 地图上点击从而发布目标



此时打开新的终端窗口启动导航脚本，便可看见小车开始自主导航

```
rosrun racecar_gazebo path_pursuit.py
```

2 文件结构

1. 工程目录树状图

```
WAVES
└──src
    ├──racecar_control
    ├──racecar_description
    │   ├──meshes
    │   ├──models
    │   └──urdf
    ├──racecar_gazebo
    │   ├──config
    │   ├──launch
    │   │   └──.vscode
    │   ├──map
    │   ├──model
    │   ├──scripts
    │   │   └──.vscode
    │   ├──src
    │   └──worlds
    └──system
```

```
├──ackermann_msgs
├──hokuyo_node
├──joystick_drivers
├──racecar
├──serial
├──vesc
└──waypoint_logger
```

2. 重点文件夹说明

racecar_control

运动控制器，主要是配置了车辆的关节运动，这里设计 racecar 转向其实和机械臂的关节设计是一样的，他们的类型都是 joint_state_controller。

racecar_contrl/config

关节参数的具体配置文件，里面有转向 PID 的参数配置，仿真其实就实际一样的。常用的舵机转向相当于是自带了位置环所以我们一般不去设计他的 PID；但是仿真的转向没有位置环，因为我们是仿真世界的主人，我们可以知道关节转动了多少角度，也能控制转向速度，所以用角度作为反馈来控制转向速度从而让关节快速的转到我们所需要的角度，所以这里就必须配置关节的 PID 了，具体实现 gazebo 已经帮我们做好了，只要自己调参数就行。

racecar_contrl/launch

关节控制的启动文件。

racecar_contrl/scripts

控制的脚本。

racecar_description

这个是整个车模的描述文件夹，含有小车的 xacro 模型文件，和场地的模型文件，还有场地外观的材料文件。

racecar_description/urdf

在模型文件racecar中可修改传感器模型的相关参数。

<!-- Add Hokuyo laser scanner -->注释下修改激光传感器。

<!-- zed camera -->注释下修改摄像头参数。

racecar_gazebo

gazebo有关的文件。

racecar_gazebo/config

里面是 slam 的配置文件，以及 rviz 启动的文件 。

ROS 中常用 amcl 来实现定位。AMCL 是 ROS/ROS2 系统中最官方的定位模块，是导航模块中唯一指定的定位算法。

ROS 有一套完整的导航功能框架，框架中大部分功能都可以使用ROS中原有的包，我们只需要封装好传感器信息，就可以像搭积木一样，把整个系统给搭建起来了。

想实现 ROS 小车的自主导航，用 ROS 中路径规划的功能包 —— move_base 就可以搞定，它由两大规划器组成：

1. 全局路径规划（global planner）

根据给定的目标位置和全局地图进行总体路径的规划。在导航中，使用 Dijkstra 或 A* 算法进行全局路径的规划，计算出小车到目标位置的最优路线，作为机器人的全局路线。

1．本地实时规划（local planner）

在实际情况中，小车往往无法严格按照全局路线行驶，所以需要针对地图信息和机器人附近随时可能出现的障碍物，规划每个周期内应该行驶的路线，使之尽量符合全局最优路径。

本地的实时规划由 local_planner 模块实现，使用 Dynamic Window Approaches 算法搜索躲避和行进的多条路经，综合各评价标准（是否会撞击障碍物，所需要的时间等）选取最优路径，并且计算行驶周期内的线速度和角速度，避免与动态出现的障碍物发生碰撞。

相关算法实现的参数配置在 src/racecar_gazebo/config文件夹中，可根据配置文件名寻找需要修改的参数。

racecar_gazebo/launch

gmapping 功能包需要订阅机器人的深度数据、 IMU 信息和里程计信息，同时完成一些必要参数的配置，即可创建并输出基于概率的二维栅格地图。

Gmapping是一个基于 2D 激光雷达使用 RBPF（Rao-Blackwellized Particle Filters）算法完成二维栅格地图构建的 SLAM 算法。

启动算法的文件 src/racecar_gazebo/launch/gmapping.launch，这里需要根据各自的情况，对代码文件中注释的部分进行更改，更改内容根据自己的情况。

racecar_gazebo/scripts

这里面存放了一些控制的脚本，且代码文件中已有相关注释。

system

这里面主要用上的就是阿克曼消息的转换，文件中还有一些与硬件有关的驱动。

3 算法替换

在项目中我们使用了两种全局路径规划算法和两种局部路径规划算法，分别为 A*、Dijkstra和Teb、DWA。

提交项目代码中默认配置使用的是 Dijkstra 和 Teb 算法，配置算法参数的文件为src/racecar_gazebo/launch/racecar_runway_navigation.launch。

如果需要切换其他算法，需要在racecar_runway_navigation.launch文件中修改参数：

- A* 算法：

```
    <param name="base_global_planner" value="global_planner/GlobalPlanner" />
<param name="planner_frequency" value="0.01" />
<param name="planner_patience" value="5.0" />
<!--param name="use_dijkstra" value="false" /-->
```

- Dijkstra 算法：

```
    <param name="base_global_planner" value="global_planner/GlobalPlanner" />
<param name="planner_frequency" value="0.01" />
<param name="planner_patience" value="5.0" />
<param name="use_dijkstra" value="true" />
```

- Teb 算法：

```
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find racecar_gazebo)/config/costmap_common_params.yaml" command="load" ns="
global_costmap" />
    <rosparam file="$(find racecar_gazebo)/config/costmap_common_params.yaml" command="load" ns="
local_costmap" />
    <rosparam file="$(find racecar_gazebo)/config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find racecar_gazebo)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find racecar_gazebo)/config/teb_local_planner_params.yaml" command="load" />

    <param name="base_global_planner" value="global_planner/GlobalPlanner" />
    <param name="planner_frequency" value="0.01" />
    <param name="planner_patience" value="5.0" />
    <!--param name="use_dijkstra" value="false" /-->

    <param name="base_local_planner" value="teb_local_planner/TebLocalPlannerROS" />
    <param name="controller_frequency" value="3.0" />
    <param name="controller_patience" value="15.0" />

    <param name="clearing_rotation_allowed" value="false" />
  </node>
```

并且还需要在racecar_gazebo/scripts/path_pursuit.py中修改第18行：

```
self.path_pose = rospy.Subscriber('/move_base/TebLocalPlannerROS/global_plan', Path, self.callback_read_path,
queue_size=1)
```

- DWA算法：

```
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find racecar_gazebo)/config/costmap_common_params.yaml" command="load" ns="
global_costmap" />
    <rosparam file="$(find racecar_gazebo)/config/costmap_common_params.yaml" command="load" ns="
local_costmap" />
    <rosparam file="$(find racecar_gazebo)/config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find racecar_gazebo)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find racecar_gazebo)/config/dwa_local_planner_params.yaml" command="load" />

    <param name="base_global_planner" value="global_planner/GlobalPlanner" />
    <param name="planner_frequency" value="0.01" />
    <param name="planner_patience" value="5.0" />
    <!--param name="use_dijkstra" value="false" /-->

    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    <param name="controller_frequency" value="3.0" />
    <param name="controller_patience" value="15.0" />

    <param name="clearing_rotation_allowed" value="false" />
  </node>
```

并且还需要在racecar_gazebo/scripts/path_pursuit.py中修改第18行：

```
self.path_pose = rospy.Subscriber('/move_base/DWAPlannerROS/global_plan', Path, self.callback_read_path,
queue_size=1)
```