

一、Gazebo仿真环境搭建

1.1 创建一个工作空间或打开一个现成的工作空间

1.2 在workspace/src下创建gazebo_pkg功能包

- 打开一个终端 (Ctrl + Alt + T) , 输入以下命令

```
cd workspace/src          #进入代码目录

catkin_create_pkg gazebo_pkg  #创建功能包

cd ..                     #返回上级目录

catkin_make               #编译
```

1.3 将材料中的urdf、world、meshes文件夹复制到gazebo_pkg功能包下

urdf、meshes文件夹中存放的是机器人的模型文件

world文件夹存放的是Gezebo的世界模型文件

1.4 创建launch启动文件

- 在gazebo_pkg功能包下创建launch文件夹
- 在launch文件夹下创建race.launch文件
- 将以下代码复制至race.launch文件中

```
<launch>
  <!-- these are the arguments you can pass this launch file, for example
  paused:=true -->
  <arg name="paused" default="true"/>
  <!--   <arg name="use_sim_time" default="true"/> -->
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <!-- <remap from="robot/laser/scan" to="/scan"/> -->

  <!--模型车的起点放置位置-->
  <arg name="x_pos" default="0.18"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>
  <arg name="R_pos" default="0.0"/>
  <arg name="P_pos" default="0.0"/>
```

```

<arg name="Y_pos" default="3.1416"/>

<include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="world_name" value="$(find gazebo_pkg)/world/map_model.world"/>
  <arg name="debug" value="$(arg debug)" />
  <arg name="gui" value="$(arg gui)" />
</include>

<!-- Load the URDF into the ROS Parameter Server -->
<arg name="model" default="$(find gazebo_pkg)/urdf/racecar.xacro" />

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

<!-- Run a python script to send a service call the gazebo_ros to spawn a URDF
robot -->
<node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
output="screen"
  args="-urdf -model racecar -param robot_description -x $(arg x_pos) -y $(arg
y_pos) -z $(arg z_pos) -R $(arg R_pos) -P $(arg P_pos) -Y $(arg Y_pos)"/>

<!-- ros_control racecar launch file -->
<!--<include file="$(find racecar_control)/launch/racecar_control.launch"
ns="/" />-->

<!--Launch the simulation joystick control -->
<!--<rosparam command="load" file="$(find
racecar_control)/config/keyboard_teleop.yaml" />
<node pkg="racecar_control" type="keyboard_teleop.py" name="keyboard_teleop"
/> -->

<node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
<node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"/>

</launch>

```

1.5 将路径添加进.bashrc文件中

Ubuntu默认使用的终端是bash，就是（Ctrl + Alt + T）召唤出来的黑框框，**bash 在每次启动时都会加载 .bashrc 文件的内容**，.bashrc文件的作用是用来**存储并加载你的终端配置和环境变量**。

例如，ROS默认安装在/opt路径下的，我们来看看.bashrc文件中是如何帮我们找到ROS的；.bashrc文件**位于主目录下**，并且是隐藏文件，打开你的主目录，按（Ctrl + H）恢复隐藏文件，你就可以看到.bashrc文件，如果你已经成功地将小乌龟跑起来了，就可以在最后一行看到

```
source /opt/ros/ros版本/setup.bash
```

同理，想要运行自己的程序，也要告诉bash你自己的文件路径，否则的话，当你想要运行launch文件或执行roslaunch命令时，就会**报找不到文件路径的错误**：

```
atim@atim-virtual-machine:~$ roslaunch gazebo_pkg race.launch
RLEException: [race.launch] is neither a launch file in package [gazebo_pkg]
nor is [gazebo_pkg] a launch file name
The traceback for the exception was written to the log file
```

而添加环境有以下两种方法：

1. 打开一个终端，进入你的workspace目录下，运行

```
source devel/setup.bash
```

此时再次运行roslaunch、roslaunch等命令即可正常运行，但这种办法每次重新打开一个终端、添加新的功能包及进行编译都要再次运行，所以下面介绍另一种方法：

2. 手动添加命令到.bashrc

- 打开终端输入：

```
gedit ~/.bashrc
```

打开.bashrc文件，当然你也可以在主目录（home）下唤出隐藏文件（**Ctrl + H**），直接打开.bashrc文件

- 在.bashrc文件最后一行添加命令：

```
source ~/workspace/devel/setup.bash
# workspace 为你自己工作空间的名字
```

- 在终端输入：

```
source ~/.bashrc
```

使环境变量生效

- 前两步也可以用以下命令替代：

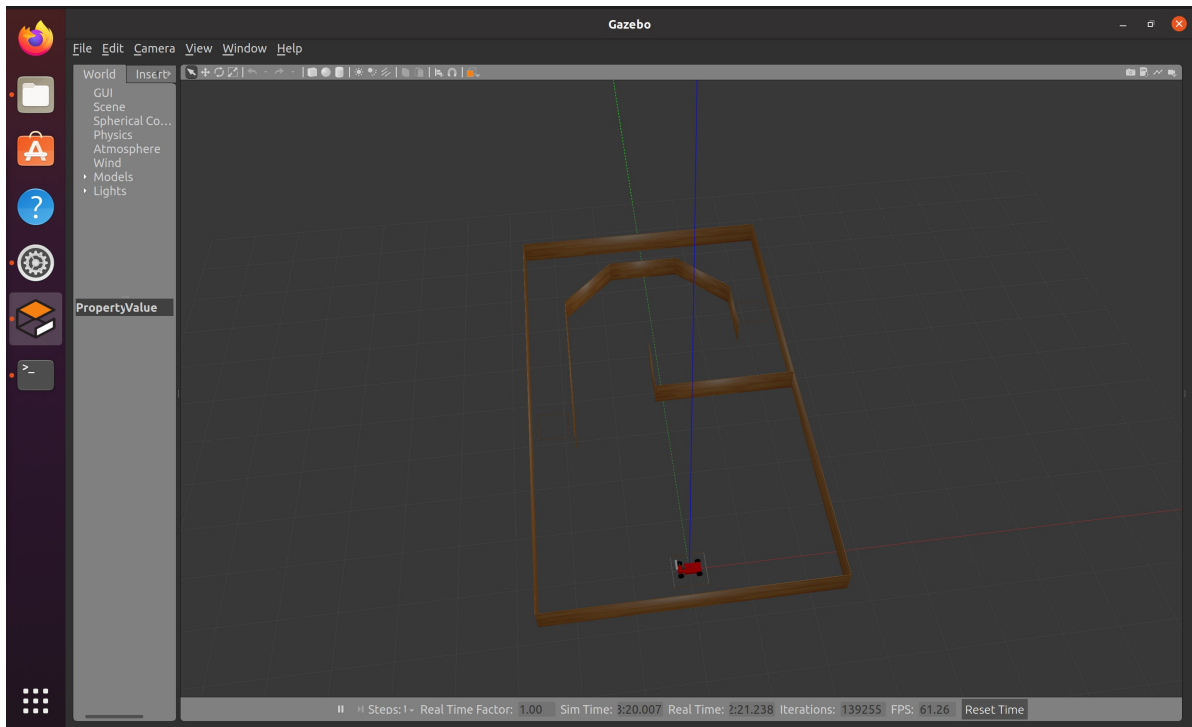
```
echo "source ~/catkin_ws/devel/setup.sh" >> ~/.bashrc
```

1.6 运行Gazebo仿真环境

打开终端，运行：

```
roslaunch gazebo_pkg race.launch
```

如果前面的步骤都没有问题，那么**恭喜你**！你已经成功运行起Gazebo的仿真环境了，你将会看到以下的画面：



1.7 控制机器人运动

在安装ROS时，相信你已经体验过了怎么使用键盘来控制小乌龟来进行运动；同样的，我们也可以使用键盘来控制我们自己的机器人进行运动，为后面的建图做准备：

- 安装依赖，终端中输入：

```
sudo apt-get install ros-版本-effort-controllers
sudo apt-get install ros-版本-ackermann-msgs
```

- 将材料中的racecar_control功能包复制至workspace/src下
- 进行编译：

```
cd workspace

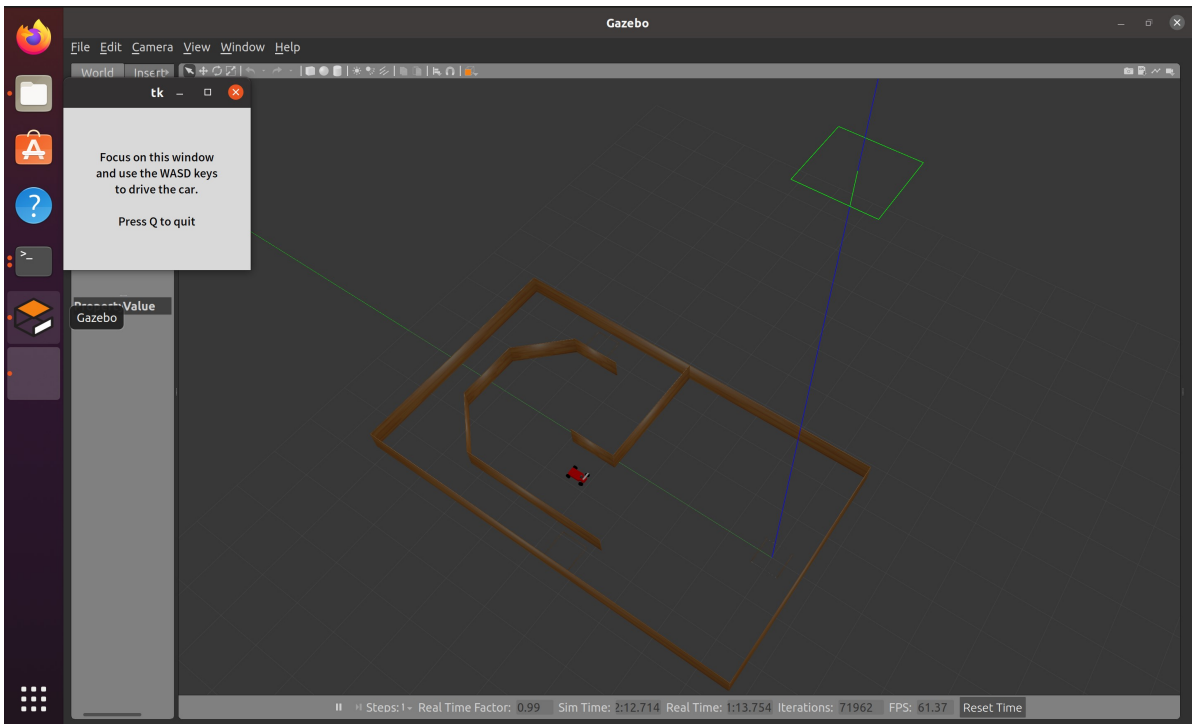
catkin_make
```

- 打开终端运行：

```
roslaunch gazebo_pkg race.launch

roslaunch racecar_control racecar_control.launch
```

- 此时你已经可以控制机器人进行运动啦！试试控制它跑一圈叭



二、建图功能包创建

在Linux平台下，软件包的类型可以划分为两类：源码包、二进制包；

一个软件要在Linux上执行，必须是二进制文件；

源码包：即程序软件的源代码（一般也叫Tarball，即将软件的源码以tar打包后再压缩的资源包）；源码包是作者直接将源程序发布在网上，我们直接下载源文件，自己编译成二进制程序使用；

二进制包：如 Red Hat发行版的.rpm包，Debian发行版的.deb包；我们可以在ROS的安装路径/opt下找到各个依赖包对应的库文件(lib)、头文件(include)、可执行文件等

区别	二进制包	源代码
下载方式	apt-get install /下载deb	git clone/github等网站直接下载源码
编译方式	无需编译	通过make/cmake/catkin
来源	官方apt软件源	开源项目/第三方开发者
扩展性	无法修改	通过源代码修改
可读性	无法查看源代码	方便阅读、学习源代码
优点	下载简单、安装方便	源码可修改、卸载
缺点	无法修改	安装、编译复杂

- ros中当我们没有学习、修改功能包源码的需求时，我们可以选择二进制安装
- 一般我们只需配置好必要的参数即可使用功能包

2.1 在workspace/src下创建gazebo_map功能包

- 打开终端，输入：

```
cd workspace/src                #进入代码目录

catkin_create_pkg gazebo_map    #创建功能包

cd ..                            #返回上级目录

catkin_make                     #编译
```

2.2 在gazebo_map功能包下创建cfg、launch、map三个文件夹

cfg文件夹存放的是参数配置文件

launch文件夹存放的是建图节点启动文件

map文件夹存放的是建图完成后的地图图片(.pgm)和配置文件(.yaml)

2.3 安装建图功能包

这里给出几种slam方法的源码地址可供学习、安装：

- 激光SLAM：

- gmapping: https://github.com/ros-perception/openslam_gmapping
https://github.com/ros-perception/slam_gmapping.git
- hector: https://github.com/tu-darmstadt-ros-pkg/hector_slam
- karto: https://github.com/ros-perception/open_karto.git
https://github.com/ros-perception/slam_karto.git
- cartographer: <https://github.com/googlecartographer/cartographer.git>
https://github.com/googlecartographer/cartographer_ros.git
- LOAM: https://github.com/laboshin/loam_velodyne.git

- 视觉SLAM：

- ORB_SLAM: https://github.com/raulmur/ORB_SLAM
https://github.com/raulmur/ORB_SLAM2
https://github.com/UZ-SLAMLab/ORB_SLAM3.git
- LSD_SLAM: https://github.com/tum-vision/lsd_slam (https://github.com/zouyuelin/LSD_SLAM_Changed.git)

- 其它：

- RTABMAP: https://github.com/introlab/rtabmap_ros
- VINS: <https://github.com/HKUST-Aerial-Robotics/VINS-Mono.git>

你可以在以下几种方法中任选其一，也可都安装，它们并不冲突，甚至你可以比较用以下几种方法建立的地图谁更好。当你完成2.3.x中的任意一个你就可以跳到2.4开始建图啦！

2.3.1 gmapping安装

- 采用二进制安装，终端中输入：

```
sudo apt-get install ros-版本-slam-gmapping
sudo apt-get install ros-版本-gmapping
```

- 于launch文件夹中创建slam_gmapping.launch文件
- 将以下代码复制至slam_gmapping.launch文件中

```
<launch>
  <param name="use_sim_time" value="true"/>
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
output="screen">
    <remap from="scan" to="scan"/>
    <param name="base_frame" value="base_link"/>
    <param name="map_frame" value="map"/>
    <param name="odom_frame" value="odom"/>

    <param name="map_update_interval" value="5.0"/>
    <param name="maxUrange" value="16.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="srr" value="0.1"/>
    <param name="srt" value="0.2"/>
    <param name="str" value="0.1"/>
    <param name="stt" value="0.2"/>
    <param name="linearUpdate" value="1.0"/>
    <param name="angularUpdate" value="0.5"/>
    <param name="temporalUpdate" value="3.0"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="30"/>
    <param name="xmin" value="-50.0"/>
    <param name="ymin" value="-50.0"/>
    <param name="xmax" value="50.0"/>
    <param name="ymax" value="50.0"/>
    <param name="delta" value="0.05"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>
  </node>
</launch>
```

2.3.2 hector安装

- 采用二进制安装，终端中输入：

```
sudo apt-get install ros-版本-hector-slam
```

- 于launch文件夹中创建slam_hector文件夹，同时创建slam_hector.launch、mapping_default.launch、geotiff_mapper.launch三个文件
- 将以下代码复制至slam_hector.launch文件中

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

<launch>

  <arg name="geotiff_map_file_path" default="$(find gazebo_map)/map"/>

  <param name="/use_sim_time" value="true"/>

  <!-- <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/> -->

  <include file="$(find
gazebo_map)/launch/slam_hector/mapping_default.launch"/>

  <include file="$(find
gazebo_map)/launch/slam_hector/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
  </include>

</launch>
```

- 将以下代码复制至mapping_default.launch文件中

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

<launch>
  <arg name="tf_map_scanmatch_transform_frame_name"
default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_link"/>
  <arg name="odom_frame" default="odom"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>

  <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping"
output="screen">

    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />

    <!-- Tf use -->
```



```

<param name="use_tf_scan_transformation" value="true"/>
<param name="use_tf_pose_start_estimate" value="false"/>
<param name="pub_map_odom_transform" value="$(arg
pub_map_odom_transform)"/>

<!-- Map size / start point -->
<param name="map_resolution" value="0.050"/>
<param name="map_size" value="$(arg map_size)"/>
<param name="map_start_x" value="0.5"/>
<param name="map_start_y" value="0.5" />
<param name="map_multi_res_levels" value="2" />

<!-- Map update parameters -->
<param name="update_factor_free" value="0.4"/>
<param name="update_factor_occupied" value="0.9" />
<param name="map_update_distance_thresh" value="0.4"/>
<param name="map_update_angle_thresh" value="0.9" />
<param name="laser_z_min_value" value = "-1.0" />
<param name="laser_z_max_value" value = "1.0" />

<!-- Advertising config -->
<param name="advertise_map_service" value="true"/>

<param name="scan_subscriber_queue_size" value="$(arg
scan_subscriber_queue_size)"/>
<param name="scan_topic" value="$(arg scan_topic)"/>

<!-- Debug parameters -->
<!--
  <param name="output_timing" value="false"/>
  <param name="pub_drawings" value="true"/>
  <param name="pub_debug_output" value="true"/>
-->
<param name="tf_map_scanmatch_transform_frame_name" value="$(arg
tf_map_scanmatch_transform_frame_name)" />
</node>

<!-- <node pkg="tf" type="static_transform_publisher"
name="map_odom_broadcaster" args="0 0 0 0 0 map $(arg odom_frame) 100"/> -
->
</launch>

```

- 将以下代码复制至geotiff_mapper.launch文件中

```

<?xml version="1.0"?>

<launch>
  <arg name="trajectory_source_frame_name" default="/base_link"/>
  <arg name="trajectory_update_rate" default="4"/>
  <arg name="trajectory_publish_rate" default="0.25"/>
  <arg name="map_file_path" default="$(find gazebo_map)/map"/>
  <arg name="map_file_base_name" default="hector_map"/>

  <node pkg="hector_trajectory_server" type="hector_trajectory_server"
name="hector_trajectory_server" output="screen">
    <param name="target_frame_name" type="string" value="/map" />

```

```

    <param name="source_frame_name" type="string" value="$(arg
trajectory_source_frame_name)" />
    <param name="trajectory_update_rate" type="double" value="$(arg
trajectory_update_rate)" />
    <param name="trajectory_publish_rate" type="double" value="$(arg
trajectory_publish_rate)" />
  </node>

  <node pkg="hector_geotiff" type="geotiff_node" name="hector_geotiff_node"
output="screen" launch-prefix="nice -n 15">
    <remap from="map" to="/dynamic_map" />
    <param name="map_file_path" type="string" value="$(arg map_file_path)"
/>
    <param name="map_file_base_name" type="string" value="$(arg
map_file_base_name)" />
    <param name="geotiff_save_period" type="double" value="0" />
    <param name="draw_background_checkerboard" type="bool" value="true" />
    <param name="draw_free_space_grid" type="bool" value="true" />
    <param name="plugins" type="string"
value="hector_geotiff_plugins/TrajectoryMapWriter" />
  </node>

</launch>

```

2.3.3 karto安装

- 采用二进制安装，终端中输入：

```
sudo apt-get install ros-版本-slam-karto
```

- 于launch文件夹中创建slam_karto.launch文件
- 将以下代码复制至slam_karto.launch文件中

```

<launch>
  <node pkg = "slam_karto" type = "slam_karto" name = "slam_karto" output
= "screen" >
    <!-- <remap from = "scan" to = "scan"/> -->
    <param name = "odom_frame" value = "odom"/>
    <param name="base_frame" value="base_link"/>
    <param name="map_frame" value="map"/>
    <param name = "map_update_interval" value = "25"/>
    <param name = "resolution" value = "0.025"/>
  </node>
</launch>

```

2.3.4 cartographer安装

1. 安装依赖

- 终端中输入：

```
sudo apt-get install -y \
```

```
cmake \  
g++ \  
git \  
google-mock \  
libboost-all-dev \  
libcairo2-dev \  
libeigen3-dev \  
libgflags-dev \  
libgoogle-glog-dev \  
liblua5.2-dev \  
libsuitesparse-dev \  
libwebp-dev \  
ninja-build \  
protobuf-compiler \  
python-sphinx
```

2. 修复rosdep

通过rosdep install命令可以给当前工作空间自动安装依赖，后面我们将使用它来帮助我们安装ceres库

当然，如果你在安装ROS时已经解决了rosdep的问题，你便可以跳过这一步了

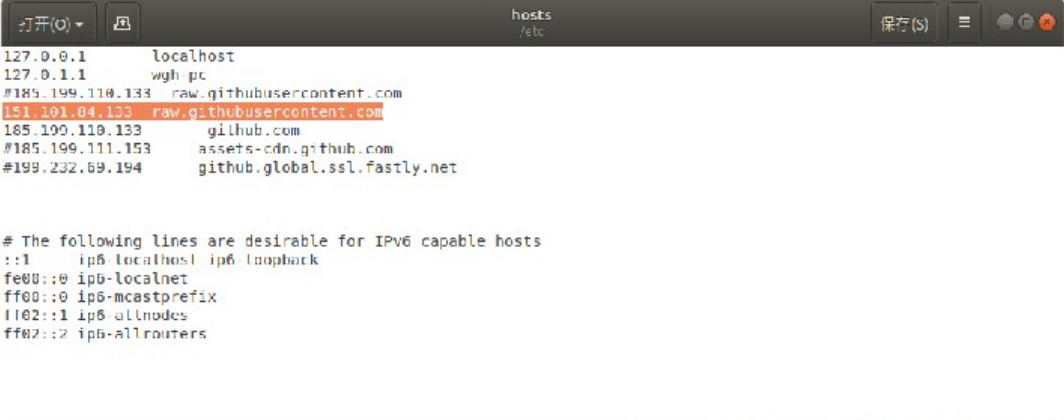
参考链接：[rosdep update 超时失败2021最新解决方法](#)

常规的方法是修改“/etc/hosts”文件，把“raw.githubusercontent.com”服务器的ip地址注册到里边，之前的话，通过此方法基本能解决 rosdep 问题，基本百试百灵。“raw.githubusercontent.com”的服务器 ip 可能会变化，大家可以通过<https://www.ipaddress.com>这个网站来查询当前的 ip。

- 终端中输入：

```
sudo gedit /etc/hosts
```

- 根据你查到的ip输入：



```
127.0.0.1        localhost  
127.0.1.1        wgh.pc  
#185.199.110.133 raw.githubusercontent.com  
151.101.04.133  raw.githubusercontent.com  
185.199.110.133 github.com  
#185.199.111.153 assets-cdn.github.com  
#199.232.69.194 github.global.ssl.fastly.net  
  
# The following lines are desirable for IPv6 capable hosts  
::1            ip6-localhost ip6-loopback  
fe00::0        ip6-localnet  
ff00::0        ip6-mcastprefix  
ff02::1        ip6-allnodes  
ff02::2        ip6-allrouters
```

纯文本 制表符宽度: 8 第 4 行, 第 1 列 插入

- 终端运行：

```
sudo rosdep init  
rosdep update
```

如果还是不行，尝试参考链接中的第二个方法

或尝试使用“rosdep”: [本文之后, 世上再无rosdep更新失败问题!](#)

看到如下画面则你的rosdep修复成功啦!

```
reading in sources list data from /etc/ros/rosdep/sources.list.d
Warning: running 'rosdep update' as root is not recommended.
You should run 'sudo rosdep fix-permissions' and invoke 'rosdep update' again without sudo.
Hit https://mirrors.tuna.tsinghua.edu.cn/github-raw/ros/rosdistro/master/rosdep/osx-homebrew.yaml
Hit https://mirrors.tuna.tsinghua.edu.cn/github-raw/ros/rosdistro/master/rosdep/base.yaml
Hit https://mirrors.tuna.tsinghua.edu.cn/github-raw/ros/rosdistro/master/rosdep/python.yaml
Hit https://mirrors.tuna.tsinghua.edu.cn/github-raw/ros/rosdistro/master/rosdep/ruby.yaml
Query rosdistro index https://mirrors.tuna.tsinghua.edu.cn/rosdistro/index-v4.yaml
Skip end-of-life distro "ardent"
Skip end-of-life distro "bouncy"
Skip end-of-life distro "crystal"
Skip end-of-life distro "dashing"
Skip end-of-life distro "eloquent"
Add distro "foxy"
Add distro "galactic"
Skip end-of-life distro "groovy"
Add distro "humble"
Skip end-of-life distro "hydro"
Skip end-of-life distro "indigo"
Skip end-of-life distro "jade"
Skip end-of-life distro "kinetic"
Skip end-of-life distro "lunar"
Add distro "melodic"
Add distro "noetic"
Add distro "rolling"
updated cache in /root/.ros/rosdep/sources.cache
```

3. 安装protobuf3.6.0

参考链接: [Cartographer安装教程及踩坑实录](#)

- 安装protobuf依赖, 终端输入:

```
sudo apt-get install autoconf automake libtool curl make g++ unzip
```

- 克隆源码, 终端输入:

```
git clone -b v3.6.0 https://github.com/protocolbuffers/protobuf.git

cd protobuf

git submodule update --init --recursive
```

- 编译安装, 终端输入:

```
./autogen.sh

./configure

make //此处编译会很长时间, 耐心等待

make check

sudo make install

sudo ldconfig

// 输出 protobuf 版本信息则表示安装成功
protoc --version
```

- 查看安装位置, 终端输入:

```
which protoc
```

- 如果不是“/usr/bin/protoc”需要更正安装位置，终端输入：

```
sudo cp /usr/local/bin/protoc /usr/bin
```

4. 安装abseil-cpp

- 终端依次输入：

```
git clone https://github.com/abseil/abseil-cpp.git

cd abseil-cpp

mkdir build

cd build

cmake -G Ninja \
-DMAKE_BUILD_TYPE=Release \
-DMAKE_POSITION_INDEPENDENT_CODE=ON \
-DMAKE_INSTALL_PREFIX=/usr/local/stow/abs1 \
..

ninja //编译需要一定时间

sudo ninja install

cd /usr/local/stow

sudo stow abs1
```

5. 安装ceres

我们这里选择在最后使用rosdep进行二进制安装，当然你也可以选择源码安装，注意是**1.14版本！ 1.14版本！ 1.14版本！**

手动安装参考链接：[Ubuntu20.04安装Ceres和g2o库](#)

[Ceres Solver官方教程](#)

6. 编译安装cartographer与cartographer_ros

- 安装依赖，终端输入：

```
sudo apt-get install -y python3-wstool python3-rosdep ninja-build stow
```

- 源码下载，终端输入：

```
git clone https://github.com/googlecartographer/cartographer.git

git clone https://github.com/googlecartographer/cartographer_ros.git
```

- 终端输入：

```
mkdir cartographer_ws  
  
cd cartographer_ws  
  
wstool init src
```

- 将 cartographer_ros 文件夹放入刚刚创建好的工作空间 src 中
- rosdep 自动安装依赖，仔细可以看到有安装 ceres1.14，终端输入：

```
sudo rosdep init  
  
rosdep update  
  
rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -  
y
```

- 编译 cartographer，返回主目录，终端输入：

```
cd cartographer  
  
mkdir build  
  
cd build  
  
cmake .. -G Ninja  
  
ninja //编译需要一定时间，耐心等待  
  
CTEST_OUTPUT_ON_FAILURE=1 ninja test  
  
sudo ninja install
```

编译过程中完全无报错无异常证明已将 cartographer 成功安装到系统中，报错的需要看看问题是不是 protobuf 版本不对，或者是 ceres 版本不对，查看方法就是直接看报错的文件路径。

- 编译工作空间，回到 cartographer_ws 工作空间目录下运行：

```
catkin_make
```

看到如下画面，**恭喜你!** cartographer 安装完成

```
atim@atim-virtual-machine: ~/cartographer_ws
[ 95%] Linking CXX executable /home/atim/cartographer_ws/devel/lib/cartographer_
ros/cartographer_dev_trajectory_comparison
[ 95%] Built target cartographer_rviz_autogen
Scanning dependencies of target cartographer_rviz
[ 95%] Built target cartographer_node
[ 97%] Building CXX object cartographer_ros/cartographer_rviz/CMakeFiles/cartogr
apher_rviz.dir/cartographer_rviz/ogre_slice.cc.o
[ 97%] Building CXX object cartographer_ros/cartographer_rviz/CMakeFiles/cartogr
apher_rviz.dir/cartographer_rviz_autogen/mocs_compilation.cpp.o
[ 97%] Building CXX object cartographer_ros/cartographer_rviz/CMakeFiles/cartogr
apher_rviz.dir/cartographer_rviz/drawable_submap.cc.o
[ 98%] Building CXX object cartographer_ros/cartographer_rviz/CMakeFiles/cartogr
apher_rviz.dir/cartographer_rviz/submaps_display.cc.o
[ 98%] Built target cartographer_dev_trajectory_comparison
[100%] Linking CXX executable /home/atim/cartographer_ws/devel/lib/cartographer_
ros/cartographer_dev_rosbag_publisher
[100%] Built target cartographer_dev_rosbag_publisher
[100%] Linking CXX executable /home/atim/cartographer_ws/devel/lib/cartographer_
ros/cartographer_rosbag_validate
[100%] Built target cartographer_rosbag_validate
[100%] Linking CXX shared library /home/atim/cartographer_ws/devel/lib/libcartog
rapher_rviz.so
[100%] Built target cartographer_rviz
atim@atim-virtual-machine: ~/cartographer_ws$
```

7. 配置参数配置、节点启动文件

- 将cartographer_ros功能包复制至你的工作空间src目录下，并编译：

```
cd workspace
catkin_make
```

- 于cfg文件夹中创建my_revo_lds.lua文件
- 将以下代码复制至my_revo_lds.lua文件中

```
-- Copyright 2016 The Cartographer Authors
--
-- Licensed under the Apache License, Version 2.0 (the "License");
-- you may not use this file except in compliance with the License.
-- You may obtain a copy of the License at
--
--     http://www.apache.org/licenses/LICENSE-2.0
--
-- Unless required by applicable law or agreed to in writing, software
-- distributed under the License is distributed on an "AS IS" BASIS,
-- WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
-- See the License for the specific language governing permissions and
-- limitations under the License.

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

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_link",
  published_frame = "base_link",
  odom_frame = "odom",
  provide_odom_frame = true,
```

```

publish_frame_projected_to_2d = false,
use_pose_extrapolator = true,
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_2D.submaps.num_range_data = 35
TRAJECTORY_BUILDER_2D.min_range = 0.3
TRAJECTORY_BUILDER_2D.max_range = 8.
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 1.
TRAJECTORY_BUILDER_2D.use_imu_data = false
TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = true
TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.linear_search_w
indow = 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

```

- 于launch文件夹中创建slam_cartographer.launch文件
- 将以下代码复制至slam_cartographer.launch文件中

```

<launch>
  <param name="/use_sim_time" value="true" />

  <node name="cartographer_node" pkg="cartographer_ros"
    type="cartographer_node" args="
      -configuration_directory $(find gazebo_map)/cfg
      -configuration_basename my_revo_lds.lua"
    output="screen">
    <remap from="scan" to="scan" />
  </node>

  <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"

```



```

type="cartographer_occupancy_grid_node" args="-resolution 0.05" />

<!-- <node name="rviz" pkg="rviz" type="rviz" required="true"
      args="-d $(find gazebo_with_map)/cfg/demo.rviz" /> -->
</launch>

```

2.4 开始建图

1. 保存Rviz配置

- 于终端中依次打开以下节点启动文件（这里以gmapping为例）

```
roslaunch gazebo_pkg race.launch
```

```
roslaunch racecar_control racecar_control.launch
```

```
roslaunch gazebo_map slam_gmapping.launch
```

选用其它slam算法则运行其对应的launch文件，即slam_XXXXXXX.launch

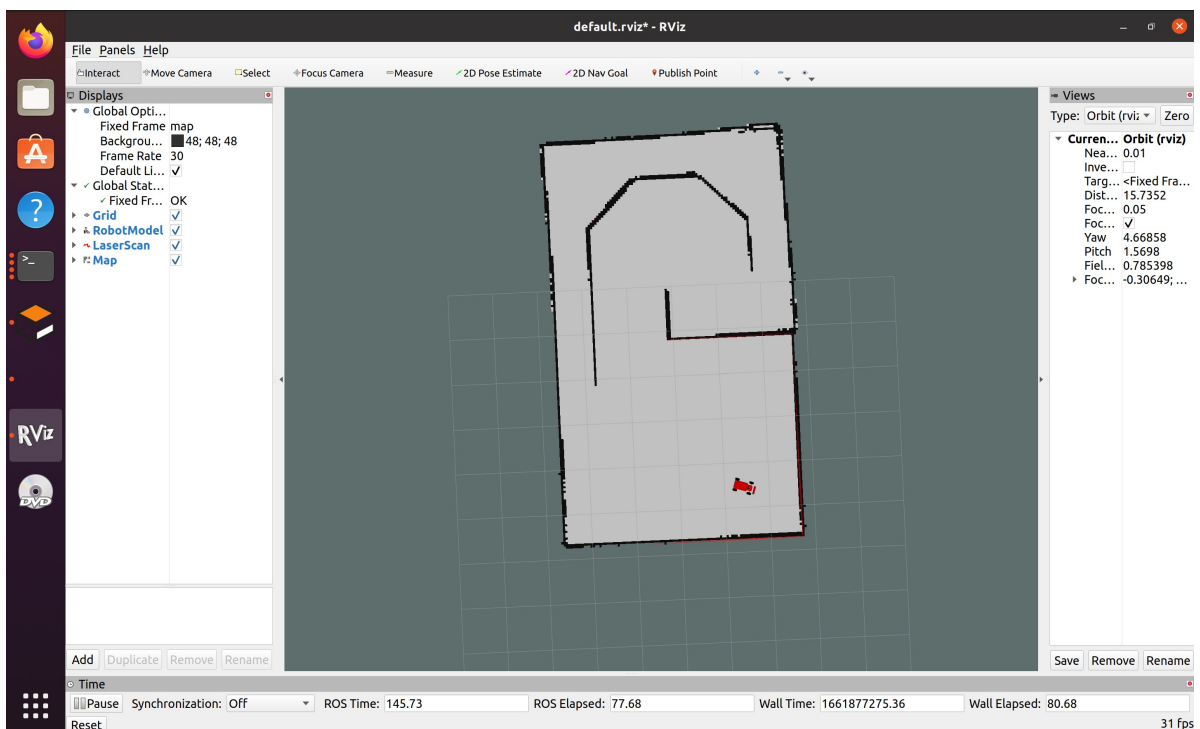
- 打开Rviz

点击左下角**Add**，于By display type中添加**RobotModel**，于By topic中添加**Map**、**LaserScan**

- 点击右上角file—>save config as，将Rviz配置文件命名为demo.rviz存放于cfg文件夹中

2. 完成建图

- 控制机器人运动，进行建图，当你看到如下画面，恭喜你！完成了一张二维栅格地图的构建



2.5 保存地图

- 终端运行以下命令保存地图

```
roslaunch map_server map_saver -f mapname
```

若所建地图不够好，可在保存地图时调整阈值参数。-occ为无法通行的阈值，-free为可通行的阈值，命令为：

```
roslaunch map_server map_saver --occ 70 --free 30 -f mapname
```

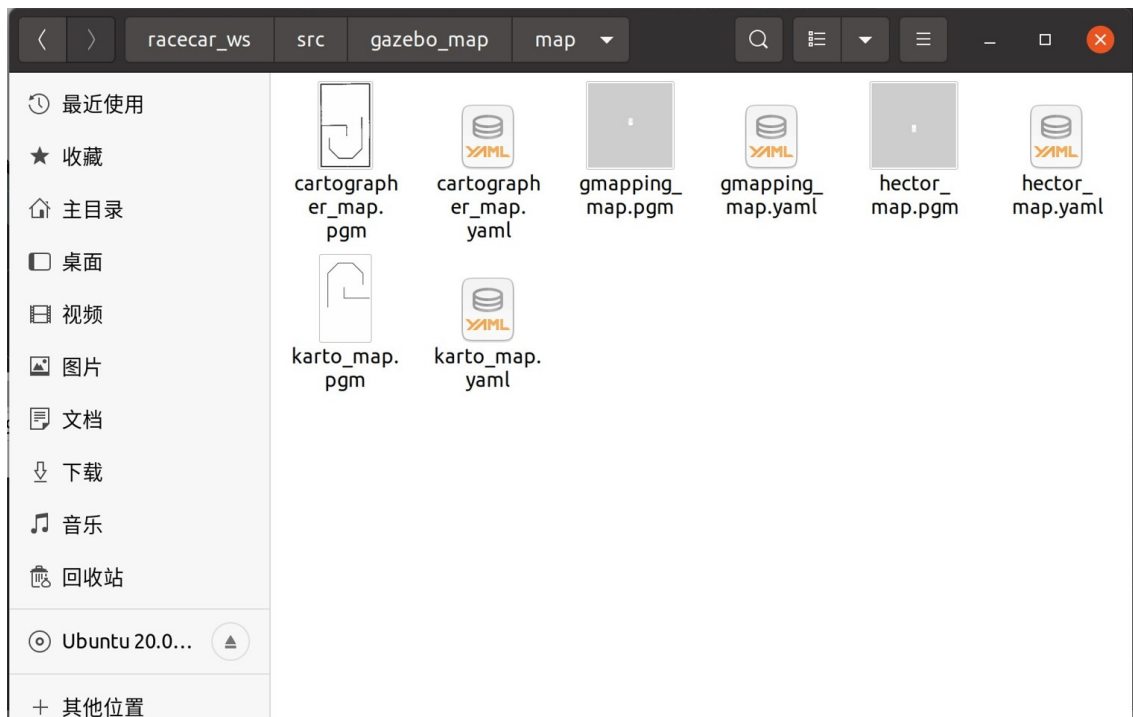
若-occ过高，则可能出现墙体边缘出现灰色未探索地块；若-occ过低，则无墙体部分可能会被凭空建墙。

若-free过高，则可能出现墙体被认为是自由区域；若-free过低，则自由区域可能出现灰色未探索地块。

- 若无法保存地图则可能为没有安装map_server，终端中输入：

```
sudo apt-get install ros-版本-map-server
```

- 地图默认保存于主目录下，将其移至**map**文件夹下完成建图



三、导航功能包创建

3.1 安装导航功能包

- 打开终端输入：

```
sudo apt-get install ros-版本-navigation
```

3.2 在workspace/src下创建gazebo_nav功能包

- 打开终端依次输入

```
cd workspace/src  
  
catkin_create_pkg gazebo_nav  
  
cd ..  
  
catkin_make
```

3.3 于gazebo_nav功能包下创建cfg、launch、map文件夹

cfg文件夹存放的是参数配置文件

launch文件夹存放的是导航节点启动文件

map文件夹存放的是导航使用的地图图片和配置文件(yaml)

- 于cfg文件夹下创建amcl、move_base、rviz三个文件夹

amcl文件夹存放的是AMCL(adaptive Monte Carlo Localization)自适应蒙特卡洛**定位功能包**所需的参数配置文件

move_base文件夹存放的是move_base**导航与路径规划功能包**所需的参数配置文件

rviz文件夹存放的是rviz的配置文件

3.4 配置机器人定位启动文件

- 于cfg/amcl文件夹下创建amcl.launch文件
- 将以下代码复制至amcl.launch文件中

```
<launch>  
  <arg name="scan_topic" default="scan"/>  
  <arg name="initial_pose_x" default="0.18"/>  
  <arg name="initial_pose_y" default="0.0"/>  
  <arg name="initial_pose_a" default="3.1415926"/>  
  
  <node pkg="amcl" type="amcl" name="amcl" clear_params="true">  
    <param name="min_particles" value="500"/>  
    <param name="max_particles" value="3000"/>  
  </node>  
</launch>
```

```

    <param name="kld_err" value="0.02"/>
    <param name="update_min_d" value="0.20"/>
    <param name="update_min_a" value="0.20"/>
    <param name="resample_interval" value="1"/>
    <param name="transform_tolerance" value="0.5"/>
    <param name="recovery_alpha_slow" value="0.00"/>
    <param name="recovery_alpha_fast" value="0.00"/>
    <param name="initial_pose_x" value="$(arg
initial_pose_x)"/>
    <param name="initial_pose_y" value="$(arg
initial_pose_y)"/>
    <param name="initial_pose_a" value="$(arg
initial_pose_a)"/>
    <param name="gui_publish_rate" value="50.0"/>

    <remap from="scan" to="$(arg scan_topic)"/>
    <param name="laser_max_range" value="3.5"/>
    <param name="laser_max_beams" value="180"/>
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="laser_model_type" value="likelihood_field"/>

    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha1" value="0.1"/>
    <param name="odom_alpha2" value="0.1"/>
    <param name="odom_alpha3" value="0.1"/>
    <param name="odom_alpha4" value="0.1"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="base_frame_id" value="base_link"/>
  </node>
</launch>

```

3.5 配置导航参数文件

- 于cfg/move_base文件夹下分别创建move_base_params.yaml、costmap_common_params.yaml、global_costmap_params.yaml、global_planner_params.yaml、local_costmap_params.yaml、dwa_local_planner_params.yaml、teb_local_planner_params.yaml文件：

1. 将以下代码复制至move_base_params.yaml文件

```

shutdown_costmaps: false #当move_base在不活动状态时,是否关掉costmap

controller_frequency: 5.0 #向底盘控制移动话题cmd_vel发送命令的频率.
controller_patience: 3.0

planner_frequency: 0.5

```

```

planner_patience: 5.0

#机器人必须移动多远（以米计）才能被视为不摆动。
#如果出现摆动则说明全局规划失败，那么将在超时后执行恢复模块
oscillation_timeout: 10.0
oscillation_distance: 0.1

conservative_reset_dist: 0.1

```

2. 将以下代码复制至costmap_common_params.yaml文件

```

#Description:
# 代价地图通用参数配置文件,就是全局代价地图和局部代价地图
# 共同都需要配置的参数,各参数意义如下:
# robot_radius: 机器人的半径
# "obstacle_range" 参数确定最大范围传感器读数,这将导致障碍物被放入代价地图中。
# 在这里,我们把它设置在3米,这意味着机器人只会更新其地图包含距离移动基座3米以内的障碍物的信息。
# "raytrace_range"参数确定了用于清除指定范围外的空间。
# 将其设置为3.0米,这意味着机器人将尝试清除3米外的空间,在代价地图中清除3米外的障碍物。

footprint: [[0.10, 0.10], [0.10, -0.10], [-0.10, -0.10], [-0.10, 0.10]]

obstacle_range: 2.5
raytrace_range: 3.0

static_layer:
  enabled: true

obstacle_layer:
  enabled: true
  track_unknown_space: true
  combination_method: 1

obstacle_range: 2.5
raytrace_range: 3.0

observation_sources: scan
scan: {
  data_type: LaserScan,
  topic: /scan,
  marking: true,
  clearing: true
}

```

3. 将以下代码复制至global_costmap_params.yaml文件

```

#Description:
# 全局代价地图参数配置文件,各参数的意义如下:
# global_frame:在全局代价地图中的全局坐标系;
# robot_base_frame:机器人的基坐标系;
#
global_costmap:
  global_frame: map
  robot_base_frame: base_link # base_footprint

```

```

transform_tolerance: 0.5

update_frequency: 15.0
publish_frequency: 10.0

plugins:
  - {name: static_layer,    type: "costmap_2d::StaticLayer"}
  - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
  - {name: inflation_layer, type: "costmap_2d::InflationLayer"}

inflation_layer:
  enabled: true
  cost_scaling_factor: 10.0 # exponential rate at which the obstacle
cost drops off (default: 10)
  inflation_radius: 0.4 # max. distance from an obstacle at which
costs are incurred for planning paths.

```

4. 将以下代码复制至global_planner_params.yaml文件

```

#Description:
# * allow_unknown: 是否允许规划器规划穿过未知区域的路径,
#   只设计该参数为true还不行, 还要在costmap_commons_params.yaml
#   中设置track_unknown_space参数也为true才行.
# * default_tolerance: 当设置的目的地被障碍物占据时, 需要以该参数
#   为半径寻找到最近的点作为新目的地.
# * visualize_potential: 是否显示从PointCloud2计算得到的势区域.
# * use_dijkstra: 设置为true, 将使用dijkstra算法, 否则使用A*算法.
# * use_quadratic: 设置为true, 将使用二次函数近似函数, 否则使用更加
#   简单的计算方式, 这样节省硬件计算资源.
# * use_grid_path: 如果设置为true, 则会规划一条沿着网格边界的路径,
#   偏向于直线穿越网格, 否则将使用梯度下降算法, 路径更为光滑点.
# * old_navfn_behavior: 若在某些情况下, 想让global_planner完全复制navfn
#   的功能, 那就设置为true, 但是需要注意navfn是非常旧的ROS系统中使用的,
#   现在已经都用global_planner代替navfn了, 所以不建议设置为true.
# * lethal_cost: 致命代价值, 默认是设置为253, 可以动态来配置该参数.
# * neutral_cost: 中等代价值, 默认设置是50, 可以动态配置该参数.
# * cost_factor: 代价地图与每个代价值相乘的因子.
# * publish_potential: 是否发布costmap的势函数.
# * orientation_mode: 如何设置每个点的方向 (None = 0, Forward = 1,
#   Interpolate = 2, ForwardThenInterpolate = 3, Backward = 4,
#   Leftward = 5, Rightward = 6) (可动态重新配置)
# * orientation_window_size: 根据orientation_mode指定的位置积分来得到
#   使用窗口的方向. 默认值1, 可以动态重新配置.
GlobalPlanner:
  allow_unknown: true
  default_tolerance: 0.05
  use_dijkstra: true
  use_quadratic: true
  use_grid_path: false
  outline_map: false
  old_navfn_behavior: false

  visualize_potential: false
  publish_potential: true

  lethal_cost: 253

```

```

neutral_cost: 50
cost_factor: 3.0

orientation_mode: 0
orientation_window_size: 1

```

5. 将以下代码复制至local_costmap_params.yaml文件

```

#Description:
# 本地代价地图需要配置的参数，各参数意义如下：
# global_frame:在本地代价地图中的全局坐标系；
# robot_base_frame:机器人本体的基坐标系；

local_costmap:
  # global_frame: odom
  global_frame: map
  robot_base_frame: base_link      # base_footprint
  transform_tolerance: 0.5

  # update_frequency: 10.0
  # publish_frequency: 10.0
  update_frequency: 15.0
  publish_frequency: 10.0
  rolling_window: true
  width: 15
  height: 15
  resolution: 0.05

  plugins:
    - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
    - {name: inflation_layer, type: "costmap_2d::InflationLayer"}

  inflation_layer:
    enabled: true
    cost_scaling_factor: 10.0 # exponential rate at which the obstacle
    cost drops off (default: 10)
    inflation_radius: 0.2 # max. distance from an obstacle at which
    costs are incurred for planning paths.

```

6. 将以下代码复制至dwa_local_planner_params.yaml文件

```

#Description:
# dwa_local_planner提供一个能够驱动底座的控制器,该控制器连接了路径规划器和机器人.
# 使用地图,规划器产生从起点到目标点的运动轨迹,在移动时,规划器在机器人周围产生一个函数,
# 用网格地图表示.控制器的工作就是利用这个函数来确定发送给机器人的速度dx, dy, dtheta
#
#    >> DWA算法的基本思想 <<
# 1. 在机器人控制空间离散采样(dx, dy, dtheta)
# 2. 对每一个采样的速度进行前向模拟,看看在当前状态下,使用该采样速度移动一小段时间后会发生什么.
# 3. 评价前向模拟得到的每个轨迹,是否接近障碍物,是否接近目标,是否接近全局路径以及速度等等.
# 舍弃非法路径
# 4. 选择得分最高的路径,发送对应的速度给底座
#

```

```

# DWA与Trajectory Rollout的区别主要是在机器人的控制空间采样差异.Trajectory Rollout
采样点来源于整个
# 前向模拟阶段所有可用速度集合,而DWA采样点仅仅来源于一个模拟步骤中的可用速度集合.这意味着
相比之下
# DWA是一种更加有效算法,因为其使用了更小采样空间;然而对于低加速度的机器人来说可能
Trajectory Rollout更好,
# 因为DWA不能对常加速度做前向模拟。
#
# 下面来依次介绍下每个参数的意义:
# * acc_lim_x:x方向的加速度绝对值
# * acc_lim_y:y方向的加速度绝对值,该值只有全向移动的机器人才需配置.
# * acc_lim_th:旋转加速度的绝对值.
#
# * max_trans_vel:平移速度最大值绝对值
# * min_trans_vel:平移速度最小值的绝对值
#
# * max_vel_x:x方向最大速度的绝对值
# * min_vel_x:x方向最小值绝对值,如果为负值表示可以后退.
# * max_vel_y:y方向最大速度的绝对值.
# * min_vel_y:y方向最小速度的绝对值.
#
# *trans_stopped_vel:停止的时候,最大的平移速度
# *theta_stopped_vel:停止的时候,最大角速度
#
# * max_rot_vel:最大旋转速度的绝对值.
# * min_rot_vel:最小旋转速度的绝对值.
# *
# * yaw_goal_tolerance:到达目标点时偏行角允许的误差,单位弧度.
# * xy_goal_tolerance:到达目标点时,在xy平面内与目标点的距离误差.
# * latch_xy_goal_tolerance:设置为true,如果到达容错距离内,机器人就会原地
# 旋转,即使转动是会跑出容错距离外.
#
# * sim_time:向前仿真轨迹的时间.
# * sim_granularity:步长,轨迹上采样点之间的距离,轨迹上点的密集程度.
# * vx_samples:x方向速度空间的采样点数.
# * vy_samples:y方向速度空间采样点数.
# * vth_samples:旋转方向的速度空间采样点数.
# * controller_frequency:发送给底盘控制移动指令的频率.
#
# * path_distance_bias:定义控制器与给定路径接近程度.
# * goal_distance_bias:定义控制器与局部目标点的接近程度
# * occdist_scale:定义控制器躲避障碍物的程度.
# * stop_time_buffer:为防止碰撞,机器人必须提前停止的时间长度.
# * scaling_speed:启动机器人底盘的速度.
# * max_scaling_factor:最大缩放参数.
#
# * publish_cost_grid:是否发布规划器在规划路径时的代价网格.如果设置为true,
# 那么就会在~/cost_cloud话题上发布sensor_msgs/PointCloud2类型消息.
# 每个点云代表代价网格,并且每个单独的评价函数都有一个字段及其每个单元
# 的总代价,并考虑评分参数.
#
# * oscillation_reset_dist:机器人运动多远距离才会重置振荡标记.
#
# * prune_plan:机器人前进是否清楚身后1m外的轨迹.
latch_xy_goal_tolerance: true

```

DWAPlannerROS:

Robot Configuration Parameters - stdr robot


```

acc_lim_x: 0.3
acc_lim_y: 0.3
acc_lim_th: 0.6

# max_trans_vel: 0.4#choose slightly less than the base's capability
# min_trans_vel: 0.1 #this is the min trans velocity when there is
negligible rotational velocity

max_vel_trans: 1.0 #choose slightly less than the base's capability
min_vel_trans: -0.1 #this is the min trans velocity when there is
negligible rotational velocity

trans_stopped_vel: 0.1
theta_stopped_vel: 0.1

max_vel_x: 1.5 # 0.8
min_vel_x: -0.8
max_vel_y: 1.8 # 0.2 diff drive robot,don't need set vel_y
min_vel_y: -0.8

# max_rot_vel: 0.5 #choose slightly less than the base's capability
# min_rot_vel: 0.1 #this is the min angular velocity when there is
negligible translational velocity

max_vel_theta: 1.9 #choose slightly less than the base's capability
min_vel_theta: -0.9 #this is the min angular velocity when there is
negligible translational velocity

# Goal Tolerance Parameters
yaw_goal_tolerance: 0.5 # 0.1 rad = 5.7 degree
xy_goal_tolerance: 0.6
latch_xy_goal_tolerance: true

# Forward Simulation Parameters
sim_time: 1.0 # 1.7
sim_granularity: 0.025
vx_samples: 3 # default 3
vy_samples: 3 # diff drive robot, there is only one sample
vth_samples: 7 # 20
controller_frequency: 5.0

# Trajectory Scoring Parameters
path_distance_bias: 32.0 # 32.0 -weighting for how much it should stick
to the global path plan
goal_distance_bias: 24.0 # 24.0 -weighting for how much it should attempt
to reach its goal
occdist_scale: 0.08 # 0.05 -weighting for how much the controller
should avoid obstacles
forward_point_distance: 0.125 # 0.325 -how far along to place an
additional scoring point
stop_time_buffer: 0.6 # 0.2 -amount of time a robot must stop in
before colliding for a valid traj.
scaling_speed: 0.20 # 0.25 -absolute velocity at which to start
scaling the robot's footprint
max_scaling_factor: 0.2 # 0.2 -how much to scale the robot's footprint
when at speed.
publish_cost_grid: false

```

```
# Oscillation Prevention Parameters
oscillation_reset_dist: 0.05 # default 0.05

hdiff_scale: 1.0 #全局和局部角度判断
heading_points: 1

# Global Plan Parameters
prune_plan: true

publish_traj_pc : false
publish_cost_grid_pc: false
global_frame_id: map
```

7. 将以下代码复制至teb_local_planner_params.yaml文件

```
TebLocalPlannerROS:

  odom_topic: odom
  map_frame: /map

  # Trajectory 这部分主要是用于调整轨迹

  teb_autosize: True #优化期间允许改变轨迹的时域长度
  dt_ref: 0.3 #期望的轨迹时间分辨率
  dt_hysteresis: 0.03 #根据当前时间分辨率自动调整大小的滞后现象，通常约为。建议使用dt_ref的10%

  #覆盖全局规划器提供的局部子目标的方向；规划局部路径时会覆盖掉全局路径点的方位角，
  #对于车辆的2D规划，可以设置为False，可实现对全局路径的更好跟踪。
  global_plan_overwrite_orientation: True

  #指定考虑优化的全局计划子集的最大长度，如果为0或负数：禁用；长度也受本地Costmap大小的限制
  max_global_plan_lookahead_dist: 0.8

  feasibility_check_no_poses: 1 #检测位姿可到达的时间间隔，default: 4

  #如果为true，则在目标落后于起点的情况下，可以使用向后运动来初始化基础轨迹
  #(仅在机器人配备了后部传感器的情况下才建议这样做)
  allow_init_with_backwards_motion: False

  global_plan_viapoint_sep: -1

  #参数在TebLocalPlannerROS::pruneGlobalPlan()函数中被使用
  #该参数决定了从机器人当前位置的后面一定距离开始裁剪
  #就是把机器人走过的全局路线给裁剪掉，因为已经过去了没有比较再参与计算后面的局部规划
  global_plan_prune_distance: 1

  exact_arc_length: False
  publish_feedback: False

  # Robot
  max_vel_x: 0.02
  max_vel_x_backwards: 0.02
  max_vel_theta: 0.02
  acc_lim_x: 0.02
  acc_lim_theta: 0.02
```

```

#仅适用于全向轮
# max_vel_y (double, default: 0.0)
# acc_lim_y (double, default: 0.5)

# ***** Carlake robot parameters *****
min_turning_radius: 0.36      # 最小转弯半径 注意车辆运动学中心是后轮中点
wheelbase: 0.36              # 即前后轮距离

#设置为true时, ROS话题 (rostopic) cmd_vel/angular/z 内的数据是舵机角度,
cmd_angle_instead_rotvel: True
# *****

# footprint_model: # types: "point", "circular", "two_circles", "line",
"polygon" 多边形勿重复第一个顶点, 会自动闭合
#   type: "line"
#   # radius: 0.2 # for type "circular"
#   line_start: [-0.13, 0.0] # for type "line"
#   line_end: [0.13, 0.0] # for type "line"
#   front_offset: 0.2 # for type "two_circles"
#   front_radius: 0.2 # for type "two_circles"
#   rear_offset: 0.2 # for type "two_circles"
#   rear_radius: 0.2 # for type "two_circles"
#   vertices: [ [0.25, -0.05], [0.18, -0.05], [0.18, -0.18], [-0.19,
-0.18], [-0.25, 0], [-0.19, 0.18], [0.18, 0.18], [0.18, 0.05], [0.25, 0.05]
] # for type "polygon"

# GoalTolerance
footprint_model:
  type: "polygon"
  vertices: [[0.15, 0.15], [0.15, -0.15], [-0.15, -0.15], [-0.15, 0.15]]

xy_goal_tolerance: 3.0
yaw_goal_tolerance: 5.0
#自由目标速度。设为False时, 车辆到达终点时的目标速度为0。
#TEB是时间最优规划器。缺少目标速度约束将导致车辆“全速冲线”
free_goal_vel: True

# complete_global_plan: True
# Obstacles

min_obstacle_dist: 0.05 # 与障碍的最小期望距离,
include_costmap_obstacles: True #应否考虑到局部costmap的障碍设置为True后才能规避
实时探测到的、建图时不存在的障碍物。
costmap_obstacles_behind_robot_dist: 3.0 #考虑后面n米内的障碍物2.0
obstacle_poses_affected: 30 #为了保持距离, 每个障碍物位置都与轨道上最近的位置相连。
costmap_converter_spin_thread: True
costmap_converter_rate: 5

# Optimization

no_inner_iterations: 5
no_outer_iterations: 4
optimization_activate: True
optimization_verbose: False
penalty_epsilon: 0.1
weight_max_vel_x: 2

```

```

weight_max_vel_theta: 1
weight_acc_lim_x: 1
weight_acc_lim_theta: 2 #1
weight_kinematics_nh: 1000
weight_kinematics_forward_drive: 0.1 #1
weight_kinematics_turning_radius: 1 #1
weight_optimaltime: 1
weight_obstacle: 10 #50
weight_dynamic_obstacle: 10 # not in use yet
alternative_time_cost: False # not in use yet
selection_alternative_time_cost: False
# Homotopy Class Planner

enable_homotopy_class_planning: False
enable_multithreading: False
simple_exploration: False
max_number_classes: 4
roadmap_graph_no_samples: 15
roadmap_graph_area_width: 5
h_signature_prescaler: 0.5
h_signature_threshold: 0.1
obstacle_keypoint_offset: 0.1
obstacle_heading_threshold: 0.45
visualize_hc_graph: False

# # Recovery

# shrink_horizon_backup: True
# shrink_horizon_min_duration: 10
# oscillation_recovery: True
# oscillation_v_eps: 0.1
# oscillation_omega_eps: 0.1
# oscillation_recovery_min_duration: 10
# oscillation_filter_duration: 10

```

3.6 配置导航启动文件

- 于launch文件夹下创建racecar_nav.launch文件
- 将以下代码复制至racecar_nav.launch文件

```

<launch>

    <!-- ***** Global Parameters ***** -->
    <param name="/use_sim_time" value="true"/>
    <!-- <arg name="load_state_filename"
    default="/home/hzh/HangTian_test_ws/src/gazebo_map/map/carto_map1.pbstream"/
    > -->

    <!-- ***** Maps ***** -->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find
    gazebo_nav)/map/gmapping_map.yaml" output="screen">
    <param name="frame_id" value="map" />
    </node>

```

```

<!-- <node name="cartographer_node" pkg="cartographer_ros"
      type="cartographer_node" args="
        -configuration_directory $(find gazebo_nav)/launch
        -configuration_basename
my_backpack_2d_localization.lua
        -load_state_filename $(arg load_state_filename)"
      output="screen">

  <remap from="echoes" to="scan" />
</node> -->

<!-- ***** Navigation ***** -->
<node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">

  <rosparam file="$(find
gazebo_nav)/cfg/move_base/costmap_common_params.yaml" command="load"
ns="global_costmap" />
  <rosparam file="$(find
gazebo_nav)/cfg/move_base/costmap_common_params.yaml" command="load"
ns="local_costmap" />
  <rosparam file="$(find
gazebo_nav)/cfg/move_base/global_planner_params.yaml" command="load" />
  <rosparam file="$(find
gazebo_nav)/cfg/move_base/teb_local_planner_params.yaml" command="load" />
  <rosparam file="$(find
gazebo_nav)/cfg/move_base/local_costmap_params.yaml" command="load" />
  <rosparam file="$(find
gazebo_nav)/cfg/move_base/global_costmap_params.yaml" command="load" />
  <rosparam file="$(find gazebo_nav)/cfg/move_base/move_base_params.yaml"
command="load" />

  <param name="planner_frequency" value="10" />
  <param name="planner_patience" value="15" />
  <!--param name="use_dijkstra" value="true" /-->
  <param name="base_local_planner"
value="teb_local_planner/TebLocalPlannerROS" />
  <param name="controller_frequency" value="20.0" />
  <param name="controller_patience" value="20.0" />
  <param name="clearing_rotation_allowed" value="false" />
  <param name="base_global_planner" value="global_planner/GlobalPlanner"/>

</node>

<!--<node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"
      type="cartographer_occupancy_grid_node" args="-resolution
0.05" /> -->

<!-- ***** Visualisation ***** -->

<!-- <node name="rviz" pkg="rviz" type="rviz" required="true"
      args="-d $(find gazebo_nav)/launch/config/rviz/demo1.rviz" /> --
>

</launch>

```

3.7 更改地图路径

- 将导航要使用的.pgm文件以及对应的.yaml文件复制至gazebo_nav功能包的map文件夹下
- 更改racecar_nav.launch文件中地图文件的路径改为自己的路径：

```
<node name="map_server" pkg="map_server" type="map_server" args="$(find gazebo_nav)/map/gmapping_map.yaml" output="screen">
  <param name="frame_id" value="map" />
</node>
```

3.8 开始导航

1. 保存rviz配置

- 于终端中依次打开以下节点启动文件

```
roslaunch gazebo_pkg race.launch

roslaunch racecar_control racecar_control.launch

roslaunch gazebo_nav racecar_nav.launch

roslaunch gazebo_nav amcl.launch
```

- 打开Rviz

点击左下角**Add**，于*By display type*中添加**RobotModel**、**TF**，于*By topic*中添加**Map**、**LaserScan**、**move_base/GlobalPlanner/plan/Path**、**move_base/TebLocalPlannerROS/local_plan/Path**

- 点击右上角file—>save config as，将Rviz配置文件命名为demo.rviz存放于cfg/rviz文件夹中

2. 完成导航

相信你此时已经能够看到如下画面了，**恭喜你！**终于走到了最后一步：

- 选择上方菜单栏的**2D Nav Goal**，在地图上打下一个目标点，机器人就会导航至目标点啦！

