

***Package na***

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

Gazebo相关基础

https://blog.csdn.net/kevin\_chan04/article/details/78467218

<https://blog.csdn.net/gongdiwudu/article/details/124539319> gazebo基本操作案例

urdf转sdf

(1) 利用cd命令切换到 xxx.urdf 目录下

(2) 执行下面命令即可

gz sdf -p xxx.urdf > xxx.sdf

gz sdf -p marm.urdf > marm.sdf

虽然URDF在ROS中是一个非常有用的标准格式，但是它缺少很多特性并且不能满足机器人不断发展的需求。URDF只能单独指定一个单个机器人的运动学和动力学特性，但是它不能指定这个机器人自身在世界坐标里的姿态，并且它也缺少摩擦力等其他特性。除此之外，它不能描述一些东西，比如灯光，高度图等。

在实现方面，URDF大量使用XML属性打破了proper formatting，这恰恰***使得URDF更加不灵活***。同时URDF也没有向后兼容的机制。

为了解决这个问题，***一个新的格式SDF被用在Gazebo以此解决URDF的不足***。SDF对从世界到机器人都有完整的描述，使得添加和修改gazebo中的元素很容易。

## gazebo加载机器人模型的方式

### 加载xacro模型

我们自己的启动文件，是一种启动的参考方式

|  |
| --- |
| <launch>  <!-- 设置launch文件的参数 -->  <arg name="world\_name" value="$(find mbot\_gazebo\_demo)/worlds/cloister.world"/>  <arg name="paused" default="false"/>  <arg name="use\_sim\_time" default="true"/>  <arg name="gui" default="true"/>  <arg name="headless" default="false"/>  <arg name="debug" default="false"/>  <!-- 运行gazebo仿真环境 -->  <include file="$(find gazebo\_ros)/launch/empty\_world.launch">  <arg name="world\_name" value="$(arg world\_name)" />  <arg name="debug" value="$(arg debug)" />  <arg name="gui" value="$(arg gui)" />  <arg name="paused" value="$(arg paused)"/>  <arg name="use\_sim\_time" value="$(arg use\_sim\_time)"/>  <arg name="headless" value="$(arg headless)"/>  </include>  <!-- 加载机器人模型参数 -->  <arg name="model" default="$(find xacro)/xacro --inorder '$(find mbot\_gazebo\_demo)/urdf/xacro/mbot\_gazebo.xacro'" />  <param name="robot\_description" command="$(arg model)" />    <!-- 运行joint\_state\_publisher节点，发布机器人的关节状态 -->  <node name="joint\_state\_publisher" pkg="joint\_state\_publisher" type="joint\_state\_publisher" />  <!-- 运行robot\_state\_publisher节点，发布tf -->  <node pkg="robot\_state\_publisher" type="robot\_state\_publisher" name="state\_publisher">  <param name="publish\_frequency" type="double" value="5.0" />  </node>  <!-- 在gazebo中加载机器人模型-->  <node name="urdf\_spawner" pkg="gazebo\_ros" type="spawn\_model" respawn="false" output="screen"  args="-urdf -model mbot -param robot\_description"/>  <!-- 运行rviz可视化界面 -->  <!-- <node pkg="rviz" type="rviz" name="rviz" args="-d $(find mbot\_gazebo\_demo)/rviz/mbot.rviz"/> -->  </launch> |

下面是别人的启动文件和对应的解释

|  |
| --- |
| <launch>  <param name="robot\_description"  command="$(find xacro)/xacro --inorder '$(find rrbot\_description)/urdf/rrbot.xacro'" />  <!-- send fake joint values -->  <node name="joint\_state\_publisher" pkg="joint\_state\_publisher" type="joint\_state\_publisher">  <param name="use\_gui" value="true"/>  </node>  <!-- Combine joint values -->  <node name="robot\_state\_publisher" pkg="robot\_state\_publisher" type="robot\_state\_publisher"/>  <!-- Show in Rviz -->  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find rrbot\_description)/launch/rrbot.rviz"/>  </launch> |

首先加载机器人的模型文件rrbot.urdf，***此处会调用xacro解析器，自动将xacro转为urdf***，然后启动节点joint\_state\_publisher，同时use\_gui=true表示打开joint\_state\_publisher用户界面，随后启动robot\_state\_publisher和rviz。在启动rviz时，会传入rrbot.rviz配置文件，这个配置文件里保存的有用户设置，比如rviz界面中左侧的一些displays设置。

在***rrbot.xacro***中，里面主要是一些< link >和< joint>，同时也包含两个文件：

|  |
| --- |
| rrbot.gazebo：gazebo下的颜色指定，以及一些控制器插件  materials.xacro：rviz下的颜色指定  rviz下指定的颜色并不能在gazebo中显示，因为gazebo接受OGRE的材料脚本 |

在rrbot\_world.launch文件中主要是加载world以及启动节点spawn\_model，这个节点是向gazebo\_ros请求生成一个机器人，加载该节点内容如下：

|  |
| --- |
| <!-- Run a python script to the send a service call to gazebo\_ros to spawn a URDF robot -->  <node name="urdf\_spawner" pkg="gazebo\_ros" type="spawn\_model" respawn="false" output="screen"  args="-urdf -model rrbot -param robot\_description"/> |

**ROS中spawn\_model方式加载xacro文件，设置模型的位置(xyz,rpy)**

当设置xyz时，如下：

|  |
| --- |
| <node name="spawn\_turtlebot\_model" pkg="gazebo\_ros" type="spawn\_model"  args="$(optenv ROBOT\_INITIAL\_POSE) -unpause -urdf -param robot\_description -model mobile\_base -x ${arg value1} -y ${arg value2} -z ${arg value3}"/> |

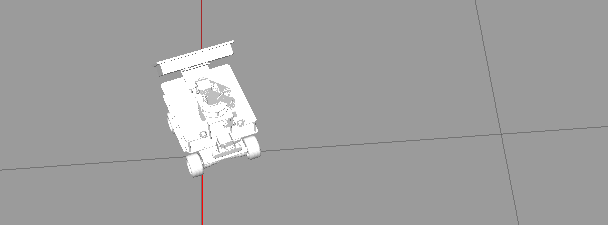
当设置rpy(roll/pitch/yaw)时,如下：

|  |
| --- |
| <node name="spawn\_turtlebot\_model" pkg="gazebo\_ros" type="spawn\_model"  args="$(optenv ROBOT\_INITIAL\_POSE) -unpause -urdf -param robot\_description -model mobile\_base -R ${arg value1} -P ${arg value2} -Y ${arg value3}"/> |

respawn=“false”表示节点挂掉不再重启；args里面的内容：-urdf表示加载的是urdf文件，-model rrbot表示加载的模型的名字是rrbot，这个对应rrbot.xacro中的robot name：

### 加载urdf模型

|  |
| --- |
| <launch>  <!-- 设置launch文件的参数 -->  <arg name="world\_name" value="$(find mbot\_gazebo\_demo)/worlds/cloister.world"/>  <arg name="paused" default="false"/>  <arg name="use\_sim\_time" default="true"/>  <arg name="gui" default="true"/>  <arg name="headless" default="false"/>  <arg name="debug" default="false"/>  <!-- 运行gazebo仿真环境 -->  <include file="$(find gazebo\_ros)/launch/empty\_world.launch">  <arg name="world\_name" value="$(arg world\_name)" />  <arg name="debug" value="$(arg debug)" />  <arg name="gui" value="$(arg gui)" />  <arg name="paused" value="$(arg paused)"/>  <arg name="use\_sim\_time" value="$(arg use\_sim\_time)"/>  <arg name="headless" value="$(arg headless)"/>  </include>  <!-- 加载机器人模型参数 -->    <param  name="robot\_description"  textfile="$(find marm\_base)/urdf/marm.urdf" />  <!-- 运行joint\_state\_publisher节点，发布机器人的关节状态 -->  <node name="joint\_state\_publisher" pkg="joint\_state\_publisher" type="joint\_state\_publisher" />  <!-- 运行robot\_state\_publisher节点，发布tf -->  <node pkg="robot\_state\_publisher" type="robot\_state\_publisher" name="state\_publisher">  <param name="publish\_frequency" type="double" value="5.0" />  </node>  <!-- 在gazebo中加载机器人模型-->  <node name="urdf\_spawner" pkg="gazebo\_ros" type="spawn\_model" respawn="false" output="screen"  args="-urdf -model mbot -param robot\_description"/>  <!-- 运行rviz可视化界面 -->  <!-- <node pkg="rviz" type="rviz" name="rviz" args="-d $(find mbot\_gazebo\_demo)/rviz/mbot.rviz"/> -->  </launch> |



## Xacro基本语法

### 汉字添加

<?xml version="1.0" encoding="UTF-8"?>

<https://blog.csdn.net/chishuideyu/article/details/53695392>

### 1.属性与算数运算

1、属性是可以插入到XML文档中的任何位置的值。

2、属性块是XML的名称片段，可以插入允许XML的任何位置

3、属性标记不能在xacro：宏中声明。

属性是可以插入到XML文档中的任何位置的值。 属性块是XML的名称片段，可以插入允许XML的任何位置。 两者都使用属性标签来定义值。属性标记不能在xacro：宏中声明。以下示例将显示如何声明和使用属性：

|  |
| --- |
| <xacro:property name="the\_radius" value="2.1" />  <xacro:property name="the\_length" value="4.5" />  <geometry type="cylinder" radius="${the\_radius}" length="${the\_length}" /> |

通过将名称放在dollared-braces（）中，你可以将两个属性值插入到几何表达式中。如果你想要一个文本“ {”，你应该将其转义为“$$ {”。

下面是使用属性块的示例：

|  |
| --- |
| <xacro:property name="front\_left\_origin">  <origin xyz="0.3 0 0" rpy="0 0 0" />  </xacro:property>  <pr2\_wheel name="front\_left\_wheel">  <xacro:insert\_block name="front\_left\_origin" />  </pr2\_wheel> |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
| 属性的定义：<xacro:property name="xxxx" value="yyyy" />  属性调用： ${属性名称}  算数运算： ${数学表达式}  例： <xacro:property name=“PI" value=“3.14" />  ${PI}  ${PI/2}  ————————————————  版权声明：本文为CSDN博主「robotyang1223」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。  原文链接：https://blog.csdn.net/robotyang1223/article/details/123757898 |

### 2.宏

xacro的主要特性是它对宏的支持。

使用宏标签定义宏，并指定宏名称和参数列表。

参数列表应以空格分隔。 它们变成宏观本地属性。

|  |
| --- |
| <xacro:macro name="pr2\_caster" params="suffix \*origin \*\*content \*\*anothercontent">  <joint name="caster\_${suffix}\_joint">  <axis xyz="0 0 1" />  </joint>  <link name="caster\_${suffix}">  <xacro:insert\_block name="origin" />  <xacro:insert\_block name="content" />  <xacro:insert\_block name="anothercontent" />  </link>  </xacro:macro>  <xacro:pr2\_caster suffix="front\_left">  <pose xyz="0 1 0" rpy="0 0 0" />  <container>  <color name="yellow"/>  <mass>0.1</mass>  </container>  <another>  <inertial>  <origin xyz="0 0 0.5" rpy="0 0 0"/>  <mass value="1"/>  <inertia ixx="100" ixy="0" ixz="0" iyy="100" iyz="0" izz="100" />  </inertial>  </another>  </xacro:pr2\_caster> |

该示例声明了一个宏“pr2\_caster”，它有两个参数：suffix和origin。

请注意，“origin”已加星标。 这表明origin是一个块参数，而不是一个简单的文本参数。

向前看pr2\_caster的使用。 后缀属性在pr2\_caster标记中定义为属性，但没有定义origin属性。

相反，origin指的是第一个元素（在这种情况下是“pose”块）。

双星号版本（“content”，“anothercontent”）允许插入在随后可用的元素（在上面的示例中分别是“container”，“another”）中传递的任意数量的元素。 此示例扩展为以下内容：

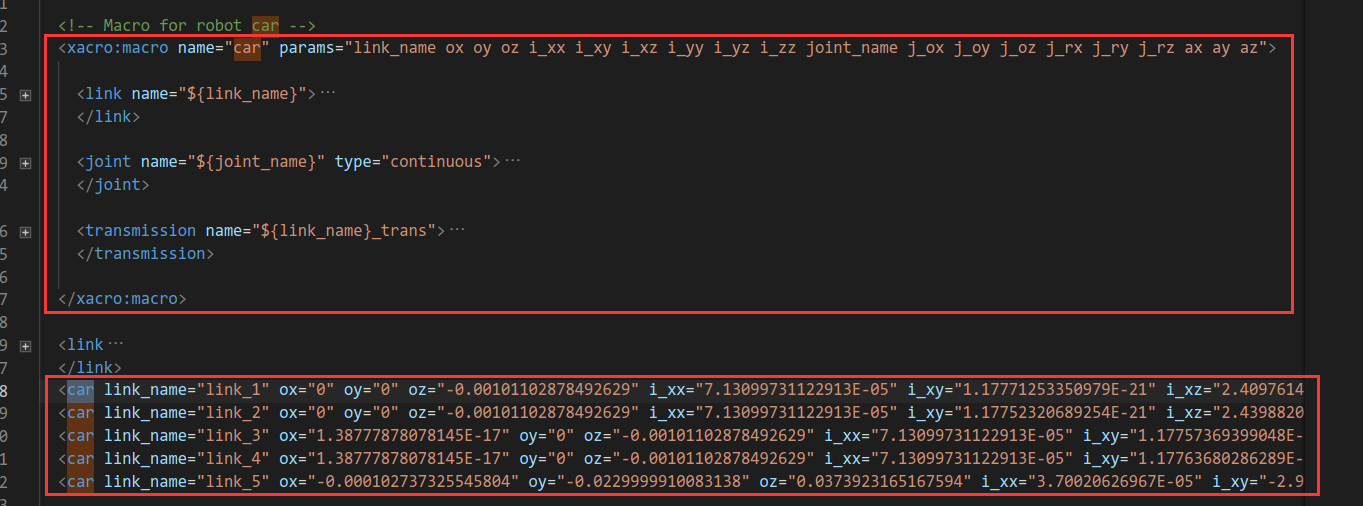
|  |
| --- |
| <xacro:macro name="reorder" params="\*first \*second">  <xacro:insert\_block name="second"/>  <xacro:insert\_block name="first"/>  </xacro:macro>  <reorder>  <first/>  <second/>  </reorder> |

|  |
| --- |
|  |

|  |
| --- |
| <!--定义-->  <xacro:macro name="func" params="v1 v2">  <!--代码块-->  </xacro:macro>  <!--调用-->  <xacro:func v1="" v2="" /> |

使用宏定义的方式，可以使用一行代码，带入参数，就可以完成代码块的编写

|  |
| --- |
| 宏的定义：  <xacro:macro name="宏名称" params="参数列表(多参数之间使用空格分隔)">  参数调用格式:  </xacro:macro>  宏的调用：  <xacro:宏名称 参数1=“xxx“ 参数2=“xxx“/> |



通过这样的宏定义，就可以减少代码重复的部分，通过如下的car宏使用就可以产生过个描述信息

|  |
| --- |
| <!-- Macro for robot car -->  <xacro:macro name="car" params="link\_name ox oy oz i\_xx i\_xy i\_xz i\_yy i\_yz i\_zz joint\_name j\_ox j\_oy j\_oz j\_rx j\_ry j\_rz ax ay az">  <link name="${link\_name}">  <inertial>  <origin xyz="${ox} ${oy} ${oz}" rpy="0 0 0" />  <mass  value="${m\_mass}" />  <inertia  ixx="${i\_xx}"  ixy="${i\_xy}"  ixz="${i\_xz}"  iyy="${i\_yy}"  iyz="${i\_yz}"  izz="${i\_zz}" />  </inertial>  <visual>  <origin  xyz="0 0 0"  rpy="0 0 0" />  <geometry>  <mesh  filename="package://mbot\_gazebo\_smartcity/meshes/${link\_name}.STL" />  </geometry>  <material name='gray' />  </visual>  <collision>  <origin  xyz="0 0 0"  rpy="0 0 0" />  <geometry>  <mesh  filename="package://mbot\_gazebo\_smartcity/meshes/${link\_name}.STL" />  </geometry>  </collision>  </link>  <joint name="${joint\_name}" type="continuous">  <origin xyz="${j\_ox} ${j\_oy} ${j\_oz}" rpy="${j\_rx} ${j\_ry} ${j\_rz}" />  <parent link="base\_link" />  <child link="${link\_name}" />  <axis xyz="${ax} ${ay} ${az}" />  </joint>    <transmission name="${link\_name}\_trans">  <type>transmission\_interface/SimpleTransmission</type>  <joint name="${joint\_name}" >  <hardwareInterface>hardware\_interface/VelocityJointInterface</hardwareInterface>  </joint>  <actuator name="${joint\_name}\_motor">  <hardwareInterface>hardware\_interface/VelocityJointInterface</hardwareInterface>  <mechanicalReduction>1</mechanicalReduction>  </actuator>  </transmission>  </xacro:macro> |

#### 默认参数

Indigo中更新

宏参数可以有默认值：

|  |
| --- |
| <xacro:macro name="foo" params="x:=${x} y:=${2\*y} z:=0"/> |

|  |
| --- |
|  |

|  |
| --- |
|  |

### 3.文件包含

机器人由多部件组成，不同部件可封装为单独的 xacro 文件，最后再将不同的文件集成，组合为完整机器人，可以使用文件包含实现。

|  |
| --- |
| 例如：  my\_robot.xacro = my\_base.xacro + my\_camera.xacro + my\_laser.xacro  <robot name=“my\_robot" xmlns:xacro="http://wiki.ros.org/xacro">  <xacro:include filename="my\_base.xacro" />  <xacro:include filename="my\_camera.xacro" />  <xacro:include filename="my\_laser.xacro" />  ....  </robot> |

|  |
| --- |
|  |

transmission标签：用于描述关节与驱动器之间的关系  
7.model\_state标签：用于描述模型当前状态

transmission标签：用于描述关节与驱动器之间的关系

7.model\_state标签：用于描述模型当前状态

## sdf语法

<http://sdformat.org/spec?ver=1.8&elem=material>

## Gazebo插件

Gazebo plugins in ROS

<https://classic.gazebosim.org/tutorials?tut=ros_gzplugins>

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

## spawn\_model发布模型

1)插入指令

方式1:

rosrun gazebo\_ros spawn\_model -database coke\_can -gazebo -model coke\_can1 -y 1

1

分析:rosrun gazebo\_ros spawn\_model这个是基本指令,目的是告诉ros需要加载一个模型;database 表示该模型是位于gazebo官方的模型库里面的;该模型在模型库里面名称叫coke\_can;插入的是model,而且,在gazebo里面名称叫coke\_can1;该模型在gazebo坐标系位置,y=1m的方向,其他为定义默认为0,所以插入的位置坐标是(0,1).

所以,这里要注意一点,这条指令需要联网的,而且链接到gazebo官方库是没问题的.(有时候联网了,链接那个数据库有可能存在问题)

方式2:

rosrun gazebo\_ros spawn\_model -file ~/.gazebo/models/box/model.sdf -sdf -model box2 -y 0 -x -1

1

分析:这个指令基本含义和上面的基本一样,但是不同点在于,启动的模型路径是本地的,这里的路径是~/.gazebo/models/box/model.sdf;然后插入模型在gazebo中的坐标是y=0,x=-1.

这种方式便利在于,不需要联网到官方库,只要告诉该gazebo模型的sdf文件在哪里就ok了.个人是推荐使用方式2.

————————————————

版权声明：本文为CSDN博主「鸿\_H」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_45701501/article/details/108197614

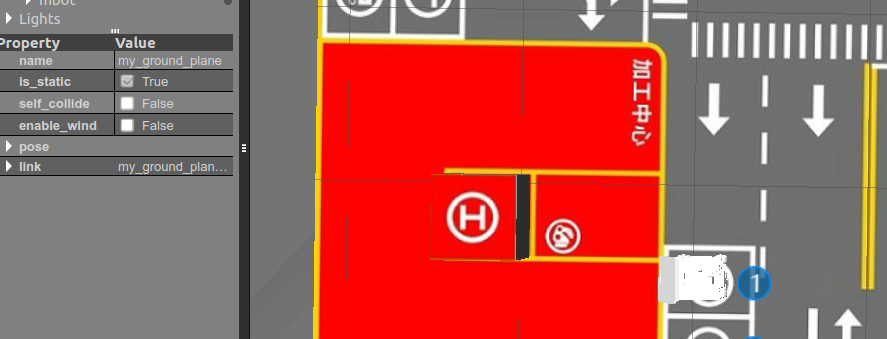
## 智慧城市world



#### 欧拉角和四元数转换工具

<https://quaternions.online/>

#### 设置gazebo小车初始位置为1号停车位



|  |
| --- |
| zonesion@zonesion:~/catkin\_ws/src/aicar/script/map$ rosservice call /gazebo/set\_model\_state "model\_state:  model\_name: 'mbot'  pose:  position:  x: -2.525  y: 0.95  z: 0.0  orientation:  x: 0.0  y: 0.0  z: -0.707  w: 0.707  twist:  linear:  x: 0.0  y: 0.0  z: 0.0  angular:  x: 0.0  y: 0.0  z: 0.0  reference\_frame: 'world'"  success: True  status\_message: "SetModelState: set model state done" |

使用启动文件

|  |
| --- |
| <!-- 在gazebo中加载机器人模型 x -2.525 y 0.95 yew -1.57-->  <node name="urdf\_spawner" pkg="gazebo\_ros" type="spawn\_model" respawn="false" output="screen"  args="-unpause -urdf -param robot\_description -model mbot -x -2.525 -y 0.95 -z 0 -R 0 -P 0 -Y -1.57"/> |

#### linux下编译脚本xxx.sh时，直接报错：bash: ./xxx.sh: /bin/bash^M: 解释器错误: 没有那个文件或目录

|  |
| --- |
| sed -i 's/\r$//' xcar.sh |

|  |
| --- |
| zonesion@zonesion:~/catkin\_ws/src/aicar/bin$ **sed -i 's/\r$//' xcar.sh**  zonesion@zonesion:~/catkin\_ws/src/aicar/bin$ ./xcar.sh  WARNING: topic [/gps/fix] does not appear to be published yet  WARNING: topic [/demo/acar/plate] does not appear to be published yet |

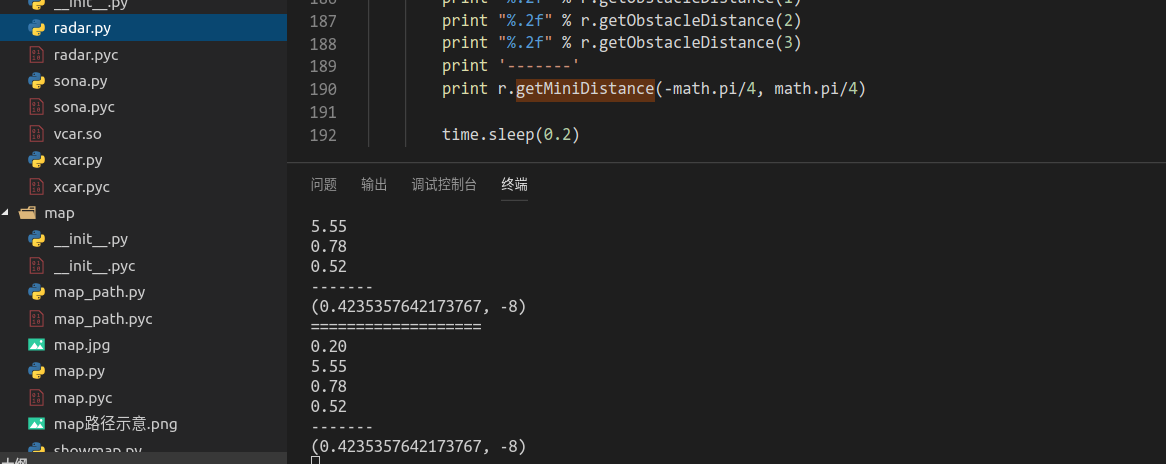
|  |
| --- |
|  |

#### 雷达参数的调整

lidar\_gazebo.xacro

|  |
| --- |
| <gazebo reference="${prefix}\_link">  <sensor type="ray" name="rplidar">  <pose>0 0 0 0 0 0</pose>  <visualize>false</visualize>  <update\_rate>5.5</update\_rate>  <ray>  <scan>  <horizontal>  <samples>360</samples>  <resolution>1</resolution>  <min\_angle>0</min\_angle>  <max\_angle>${M\_PI\*2}</max\_angle> 之前雷达的角度范围是-180 到180，这样在上层应用中获取的数据偏差180度，所以这里要将范围修正  </horizontal>  </scan>  <range>  <min>0.14</min>  <max>6.0</max>  <resolution>0.01</resolution>  </range>  <noise>  <type>gaussian</type>  <mean>0.0</mean>  <stddev>0.01</stddev>  </noise>  </ray>  <plugin name="gazebo\_rplidar" filename="libgazebo\_ros\_laser.so">  <topicName>/scan</topicName>  <frameName>laser\_link</frameName>  </plugin>  </sensor>  </gazebo>  </xacro:macro> |





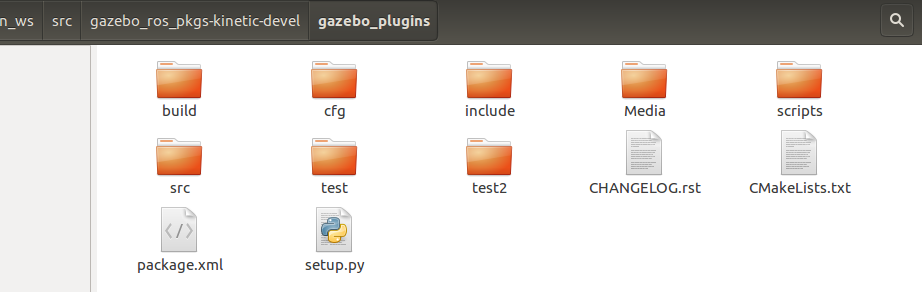
这样获取的数据就是以正前方的数据，这样与aicar的数据对应上，不用修改代码。

|  |
| --- |
|  |

#### 调整gazebo\_ros\_planar\_move.so中的odom的速度发布

由于这里是动态库，所以需要去下载源码

|  |
| --- |
| <!--Planar Move Plugin -->  <gazebo>  <plugin name="planar\_move\_controller" filename="libgazebo\_ros\_planar\_move.so">  <commandTopic>cmd\_vel</commandTopic>  <odometryTopic>odom</odometryTopic>  <odometryFrame>odom</odometryFrame>  <robotBaseFrame>base\_link</robotBaseFrame>  <leftFrontJoint>joint\_2</leftFrontJoint>  <rightFrontJoint>joint\_1</rightFrontJoint>  <leftRearJoint>joint\_4</leftRearJoint>  <rightRearJoint>joint\_3</rightRearJoint>    <odometryRate>100</odometryRate>  <broadcastTF>1</broadcastTF>  </plugin>  </gazebo>  </robot> |



下载完成后在如上目录进行编译，编译时候出现错误，

zonesion@zonesion:~$ protoc --version

libprotoc 3.9.0

原因是虚拟机的这个版本太高了。需要降级

|  |
| --- |
| 查找原因  查看/usr/include/gazebo-7/gazebo/msgs其中一个文件说明  #if GOOGLE\_PROTOBUF\_VERSION < 2006000  #error This file was generated by a newer version of protoc which is  #error incompatible with your Protocol Buffer headers. Please update  #error your headers.  #endif  #if 2006001 < GOOGLE\_PROTOBUF\_MIN\_PROTOC\_VERSION  #error This file was generated by an older version of protoc which is  #error incompatible with your Protocol Buffer headers. Please  #error regenerate this file with a newer version of protoc.  #endif  1  2  3  4  5  6  7  8  9  10  Protocol Buffer 需要2.6.1版本  下载2.6.1版本后执行  ./configure  make  make check  sudo make install  再次执行catkin\_make  ————————————————  版权声明：本文为CSDN博主「西西弗Sisyphus」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。  原文链接：https://blog.csdn.net/flyfish1986/article/details/85103865 |

不同的是我这里是gazebo8，但是解决方法是一样的。

##### Linux中安装protobuf（详细操作截图）

<https://blog.csdn.net/m0_53636439/article/details/119035091>

<https://github.com/protocolbuffers/protobuf/releases?page=11>

下载发行版，



以下开始进行安装，1、首先下载protobuf-2.6.1.tar.gz

2、解压tar -zxf protobuf-2.6.1.tar.gz

3、以下为安装步骤 ./configure

make

make check 这里可以不需要

make install 直接安装

————————————————

版权声明：本文为CSDN博主「昵称字符数」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/Ssxysxy123/article/details/50481304>

zonesion@zonesion:~/protobuf-2.6.1$ protoc --version

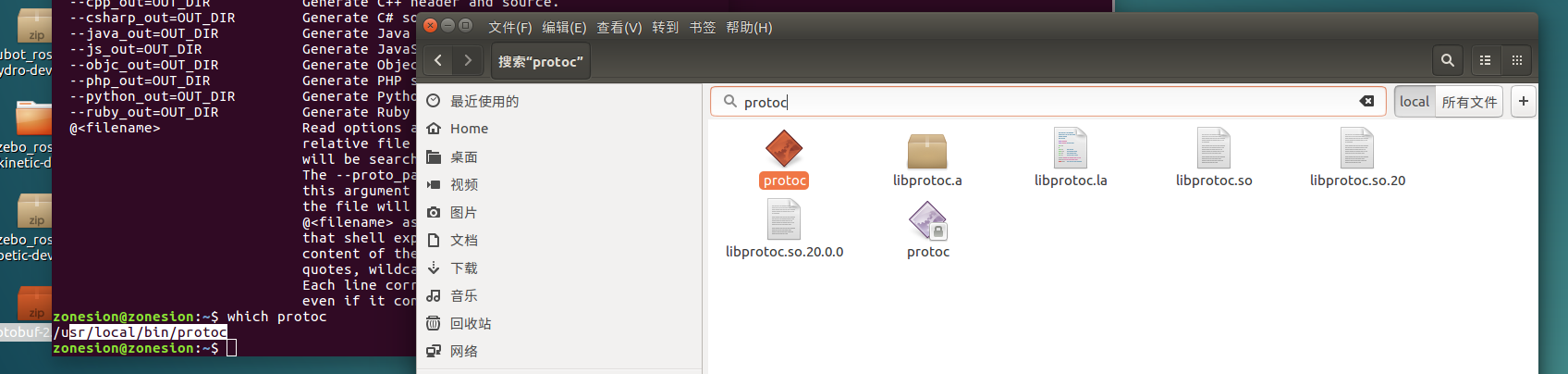
libprotoc 2.6.1

zonesion@zonesion

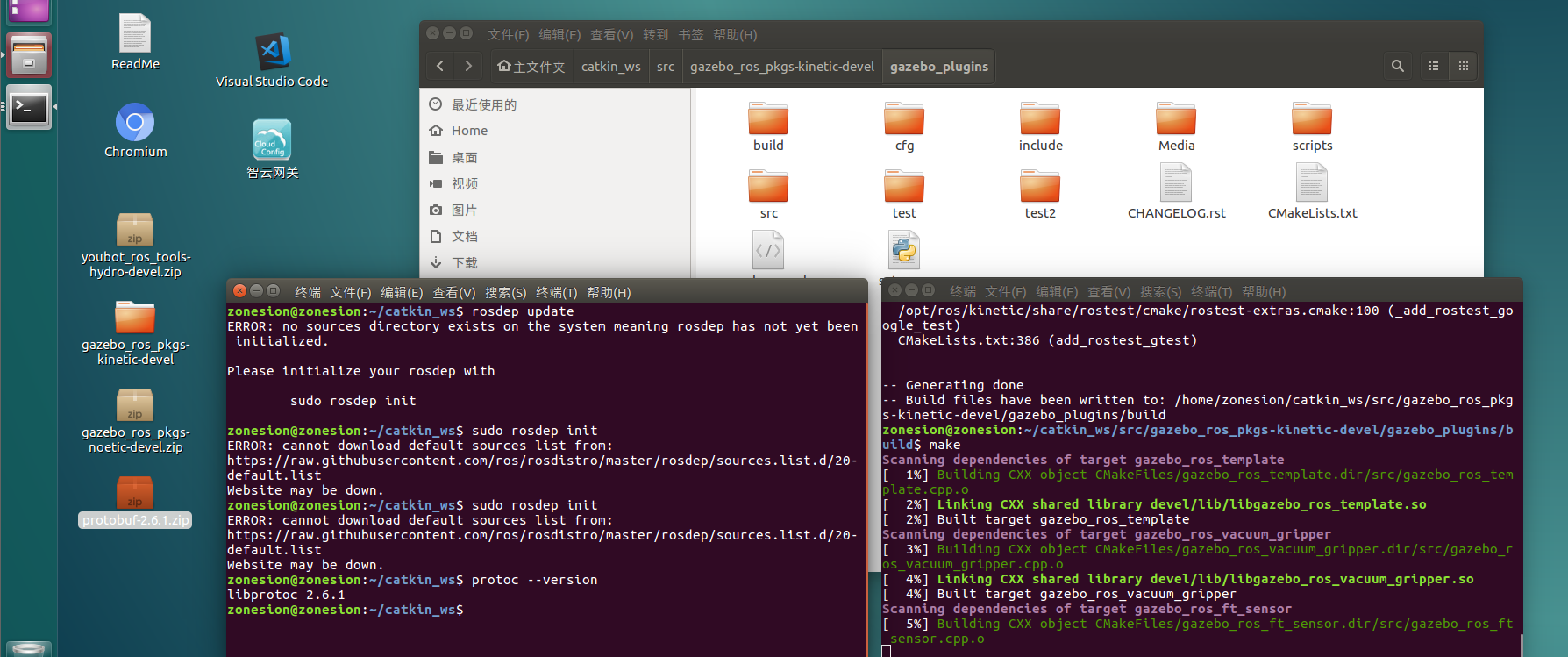
保证版本为2.6,1

grep "register\_blkdev" -r ./

###### 虚拟机原始版本



|  |
| --- |
| zonesion@zonesion:~$ protoc --version  libprotoc 3.9.0 |



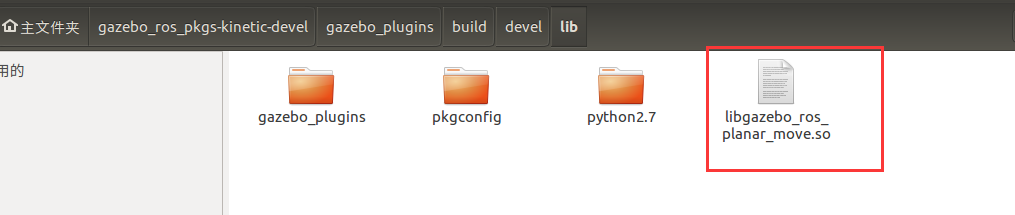
|  |
| --- |
|  |

##### 只编译gazebo\_ros\_planar\_move.cpp

|  |
| --- |
| cmake\_minimum\_required(VERSION 2.8.3)  project(gazebo\_plugins)  option(ENABLE\_DISPLAY\_TESTS "Enable the building of tests that requires a display" OFF)  find\_package(catkin REQUIRED COMPONENTS  gazebo\_dev  message\_generation  gazebo\_msgs  roscpp  rospy  nodelet  angles  std\_srvs  geometry\_msgs  sensor\_msgs  nav\_msgs  urdf  tf  tf2\_ros  dynamic\_reconfigure  rosgraph\_msgs  trajectory\_msgs  image\_transport  rosconsole  cv\_bridge  polled\_camera  diagnostic\_updater  camera\_info\_manager  std\_msgs  moveit\_msgs  )  include (FindPkgConfig)  if (PKG\_CONFIG\_FOUND)  pkg\_check\_modules(XML libxml-2.0)  pkg\_check\_modules(OGRE OGRE)  pkg\_check\_modules(OGRE-Terrain OGRE-Terrain)  pkg\_check\_modules(OGRE-Paging OGRE-Paging)  else()  message(FATAL\_ERROR "pkg-config is required; please install it")  endif()  find\_package(Boost REQUIRED COMPONENTS thread)  execute\_process(COMMAND  pkg-config --variable=plugindir OGRE  OUTPUT\_VARIABLE OGRE\_PLUGIN\_PATH  OUTPUT\_STRIP\_TRAILING\_WHITESPACE  )  catkin\_python\_setup()  generate\_dynamic\_reconfigure\_options(  cfg/CameraSynchronizer.cfg  cfg/GazeboRosCamera.cfg  cfg/GazeboRosOpenniKinect.cfg  cfg/Hokuyo.cfg  )  include\_directories(include  ${Boost\_INCLUDE\_DIRS}  ${catkin\_INCLUDE\_DIRS}  ${OGRE\_INCLUDE\_DIRS}  ${OGRE-Terrain\_INCLUDE\_DIRS}  ${OGRE-Paging\_INCLUDE\_DIRS}  )  link\_directories(  ${catkin\_LIBRARY\_DIRS}  ${OGRE\_LIBRARY\_DIRS}  ${OGRE-Terrain\_LIBRARY\_DIRS}  ${OGRE-Paging\_LIBRARY\_DIRS}  )  if (NOT GAZEBO\_VERSION VERSION\_LESS 6.0)  catkin\_package( INCLUDE\_DIRS include LIBRARIES gazebo\_ros\_elevator)  endif()  if (NOT GAZEBO\_VERSION VERSION\_LESS 7.3)  catkin\_package(INCLUDE\_DIRS include LIBRARIES gazebo\_ros\_harness)  endif()  catkin\_package(  INCLUDE\_DIRS include  LIBRARIES  vision\_reconfigure  gazebo\_ros\_planar\_move  CATKIN\_DEPENDS  message\_runtime  gazebo\_msgs  roscpp  rospy  nodelet  angles  std\_srvs  geometry\_msgs  sensor\_msgs  nav\_msgs  urdf  tf  tf2\_ros  dynamic\_reconfigure  rosgraph\_msgs  trajectory\_msgs  image\_transport  rosconsole  camera\_info\_manager  std\_msgs  moveit\_msgs  )  add\_dependencies(${PROJECT\_NAME}\_gencfg ${catkin\_EXPORTED\_TARGETS})  ## Executables  add\_executable(hokuyo\_node src/hokuyo\_node.cpp)  add\_dependencies(hokuyo\_node ${PROJECT\_NAME}\_gencfg)  target\_link\_libraries(hokuyo\_node  ${catkin\_LIBRARIES}  )  ## Plugins  add\_library(gazebo\_ros\_planar\_move src/gazebo\_ros\_planar\_move.cpp)  target\_link\_libraries(gazebo\_ros\_planar\_move ${catkin\_LIBRARIES} ${Boost\_LIBRARIES})  ##  ## Add your new plugin here  ##  ## Template  install(TARGETS  hokuyo\_node  gazebo\_ros\_planar\_move  DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}  LIBRARY DESTINATION ${CATKIN\_PACKAGE\_LIB\_DESTINATION}  )  # Tests  # These need to be run with -j1 flag because gazebo can't be run  # in parallel.  if (CATKIN\_ENABLE\_TESTING)  find\_package(rostest REQUIRED)  add\_rostest\_gtest(set\_model\_state-test  test/set\_model\_state\_test/set\_model\_state\_test.test  test/set\_model\_state\_test/set\_model\_state\_test.cpp)  target\_link\_libraries(set\_model\_state-test ${catkin\_LIBRARIES})  add\_rostest(test/range/range\_plugin.test)  if (ENABLE\_DISPLAY\_TESTS)  add\_rostest\_gtest(depth\_camera-test  test/camera/depth\_camera.test  test/camera/depth\_camera.cpp)  target\_link\_libraries(depth\_camera-test ${catkin\_LIBRARIES})  add\_rostest\_gtest(multicamera-test  test/camera/multicamera.test  test/camera/multicamera.cpp)  target\_link\_libraries(multicamera-test ${catkin\_LIBRARIES})  add\_rostest\_gtest(camera-test  test/camera/camera.test  test/camera/camera.cpp)  target\_link\_libraries(camera-test ${catkin\_LIBRARIES})  add\_rostest\_gtest(camera16bit-test  test/camera/camera16bit.test  test/camera/camera16bit.cpp)  target\_link\_libraries(camera16bit-test ${catkin\_LIBRARIES})  add\_rostest\_gtest(distortion\_barrel\_test  test/camera/distortion\_barrel.test  test/camera/distortion\_barrel.cpp)  target\_link\_libraries(distortion\_barrel\_test ${catkin\_LIBRARIES})  add\_rostest\_gtest(distortion\_pincushion\_test  test/camera/distortion\_pincushion.test  test/camera/distortion\_pincushion.cpp)  target\_link\_libraries(distortion\_pincushion\_test ${catkin\_LIBRARIES})  add\_rostest\_gtest(triggered-camera-test  test/camera/triggered\_camera.test  test/camera/triggered\_camera.cpp)  target\_link\_libraries(triggered-camera-test ${catkin\_LIBRARIES})  endif()  endif() |

删减后的cmake 文件。

##### 修改gazebo\_ros\_planar\_move.cpp



这样编译速度提升，因为这里我们只需要修改这里的so文件

调整这里的gazebo\_ros\_planar\_move.cpp

|  |
| --- |
| x\_parameter\_ = 1.0;  if (!sdf->HasElement("xparameter"))  {  ROS\_WARN\_NAMED("planar\_move", "PlanarMovePlugin (ns = %s) missing <xparameter>, "  "defaults to %f",  robot\_namespace\_.c\_str(), x\_parameter\_);  }  else  {  x\_parameter\_ = sdf->GetElement("xparameter")->Get<double>();  }  y\_parameter\_ = 1.0;  if (!sdf->HasElement("yparameter"))  {  ROS\_WARN\_NAMED("planar\_move", "PlanarMovePlugin (ns = %s) missing <yparameter>, "  "defaults to %f",  robot\_namespace\_.c\_str(), y\_parameter\_);  }  else  {  y\_parameter\_ = sdf->GetElement("yparameter")->Get<double>();  }  rot\_parameter\_ = 1.0;  if (!sdf->HasElement("rotparameter"))  {  ROS\_WARN\_NAMED("planar\_move", "PlanarMovePlugin (ns = %s) missing <rotparameter>, "  "defaults to %f",  robot\_namespace\_.c\_str(), rot\_parameter\_);  }  else  {  rot\_parameter\_ = sdf->GetElement("rotparameter")->Get<double>();  } |

增加这些默认参数

|  |
| --- |
| // Update the controller  void GazeboRosPlanarMove::UpdateChild()  {  boost::mutex::scoped\_lock scoped\_lock(lock);  #if GAZEBO\_MAJOR\_VERSION >= 8  ignition::math::Pose3d pose = parent\_->WorldPose();  #else  ignition::math::Pose3d pose = parent\_->GetWorldPose().Ign();  #endif  float yaw = pose.Rot().Yaw();  parent\_->SetLinearVel(ignition::math::Vector3d(  (x\_ \* cosf(yaw) - y\_ \* sinf(yaw))\*x\_parameter\_,  (y\_ \* cosf(yaw) + x\_ \* sinf(yaw))\*y\_parameter\_,  0));  parent\_->SetAngularVel(ignition::math::Vector3d(0, 0, rot\_\*rot\_parameter\_)); |

主要在这些函数中优化代码，这样外部就有参数调整这里的速度发布了

重新编译后替换

|  |
| --- |
| zonesion@zonesion:~$ ls /opt/ros/kinetic/lib/libgazebo\_ros\_planar\_move.so\*  /opt/ros/kinetic/lib/libgazebo\_ros\_planar\_move.so  /opt/ros/kinetic/lib/libgazebo\_ros\_planar\_move.so.bk.v1.0 原始的备份了 |

##### 应用新的gazebo\_ros\_planar\_move.cpp文件

|  |
| --- |
| <!--Planar Move Plugin -->  <gazebo>  <plugin name="planar\_move\_controller" filename="libgazebo\_ros\_planar\_move.so">  <commandTopic>cmd\_vel</commandTopic>  <odometryTopic>odom</odometryTopic>  <odometryFrame>odom</odometryFrame>  <robotBaseFrame>base\_link</robotBaseFrame>  <leftFrontJoint>joint\_2</leftFrontJoint>  <rightFrontJoint>joint\_1</rightFrontJoint>  <leftRearJoint>joint\_4</leftRearJoint>  <rightRearJoint>joint\_3</rightRearJoint>  <!-- -->  <xparameter>2.0</xparameter>  <yparameter>2.0</yparameter>  <rotparameter>1.0</rotparameter>  <odometryRate>100</odometryRate>  <broadcastTF>1</broadcastTF>  </plugin>  </gazebo>  </robot> |

这样同等单位的速度就能够产生不一样的移动结果，能够显得更加人性化，经

##### 总结

1、这里的速度控制跟关节的控制器没有关系，如下其实都是可以屏蔽掉

|  |
| --- |
| <!-- <transmission name="${link\_name}\_trans">  <type>transmission\_interface/SimpleTransmission</type>  <joint name="${joint\_name}" >  <hardwareInterface>hardware\_interface/VelocityJointInterface</hardwareInterface>  </joint>  <actuator name="${joint\_name}\_motor">  <hardwareInterface>hardware\_interface/VelocityJointInterface</hardwareInterface>  <mechanicalReduction>1</mechanicalReduction>  </actuator>  </transmission> -->  <!-- <gazebo reference="${link\_name}">  <mu1>10000000 </mu1>  <mu2>10000000 </mu2>  <kp>10000000 </kp>  <kd>1 </kd>  </gazebo> --> |

|  |
| --- |
|  |

|  |
| --- |
|  |

## 局部规划器

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

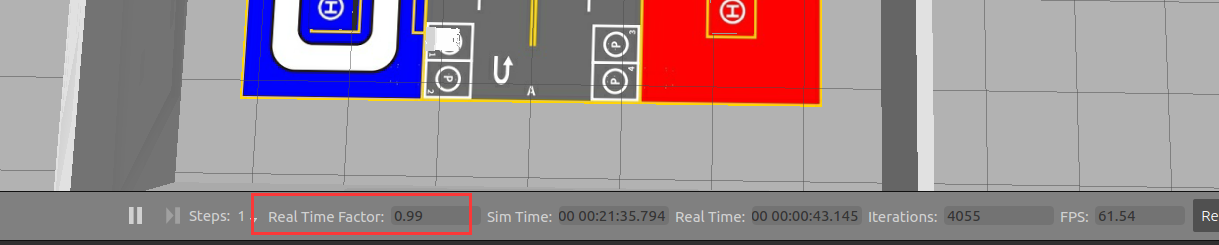
|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

## 将Gazebo的仿真速度加快10倍！！！

如果gazebo仿真这里的真实时间系数小的话，可以将如下max\_step\_size和real\_time\_update\_rate参数进行修改，适当的增大max\_step\_size，并保证时间乘积为1



<physics name='default\_physics' default='0' type='ode'>

<max\_step\_size>0.01</max\_step\_size>

<real\_time\_factor>1</real\_time\_factor>

<real\_time\_update\_rate>100</real\_time\_update\_rate>

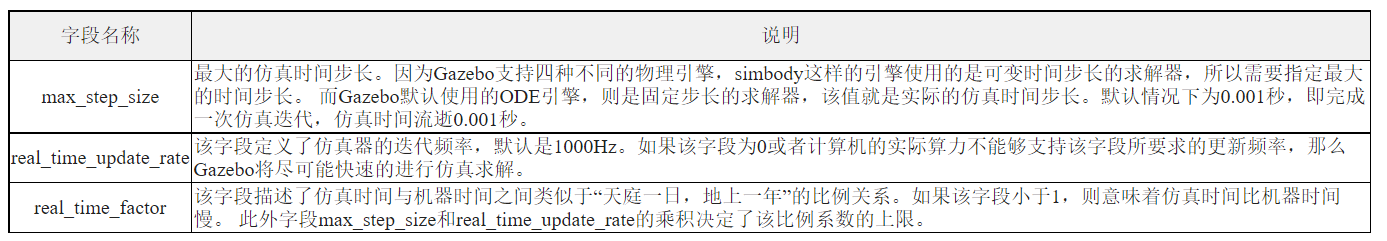
</physics>

The max step size specifies the time duration in seconds of each physics update step.

<https://classic.gazebosim.org/tutorials?tut=modifying_world&cat=build_world#PhysicsProperties>

<https://gaoyichao.com/Xiaotu/?book=Gazebo&title=Gazebo%E7%9A%84%E7%B3%BB%E7%BB%9F%E6%8F%92%E4%BB%B6%E4%B8%8EROS%E7%9A%84%E4%BB%BF%E7%9C%9F%E6%97%B6%E9%97%B4>

Gazebo的系统插件与ROS的仿真时间



|  |
| --- |
|  |

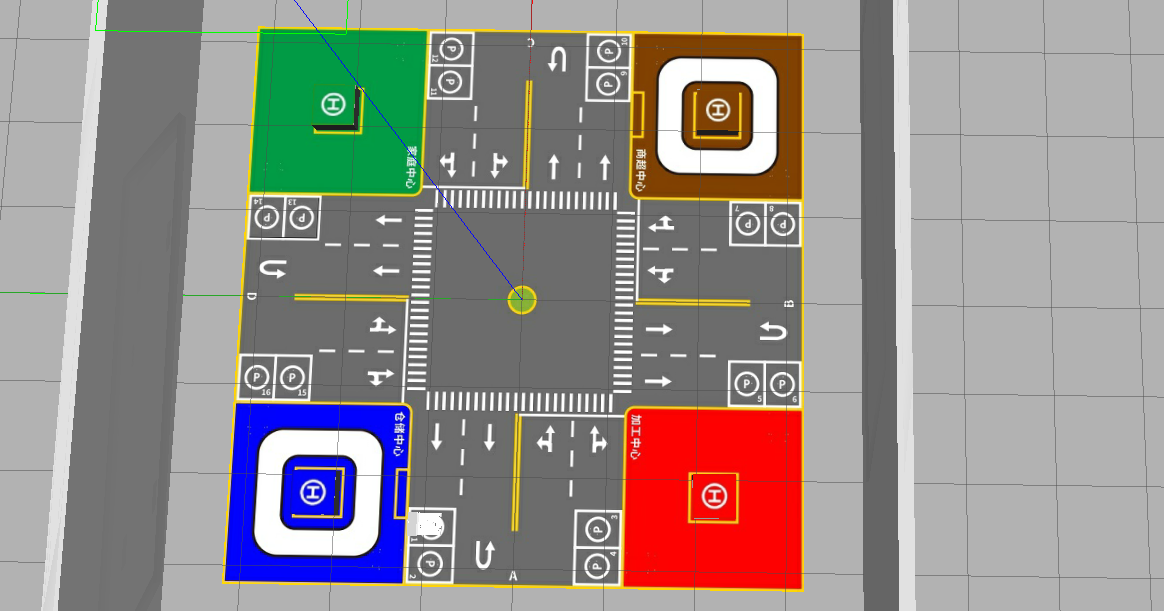
|  |
| --- |
|  |

## 基本操作步骤

### 1、手动建图

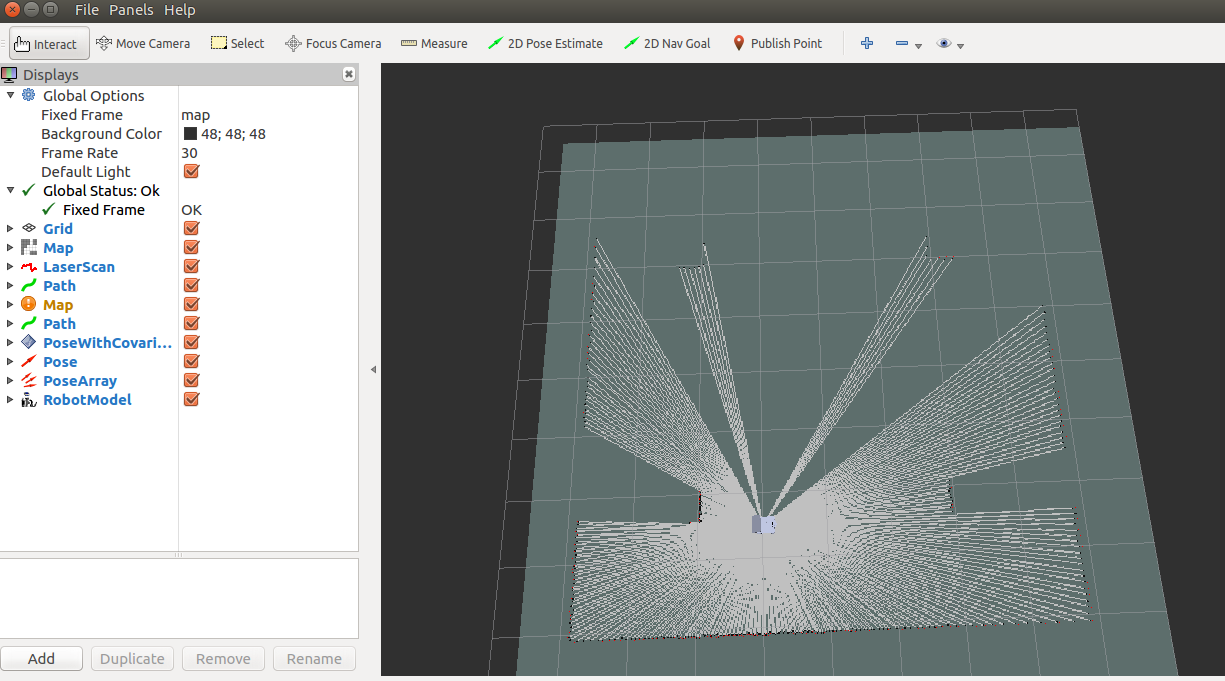
1、启动gazebo沙盘

|  |
| --- |
| zonesion@zonesion:~$ roslaunch mbot\_gazebo\_smartcity mbot\_gazebo\_smartcity.launch |

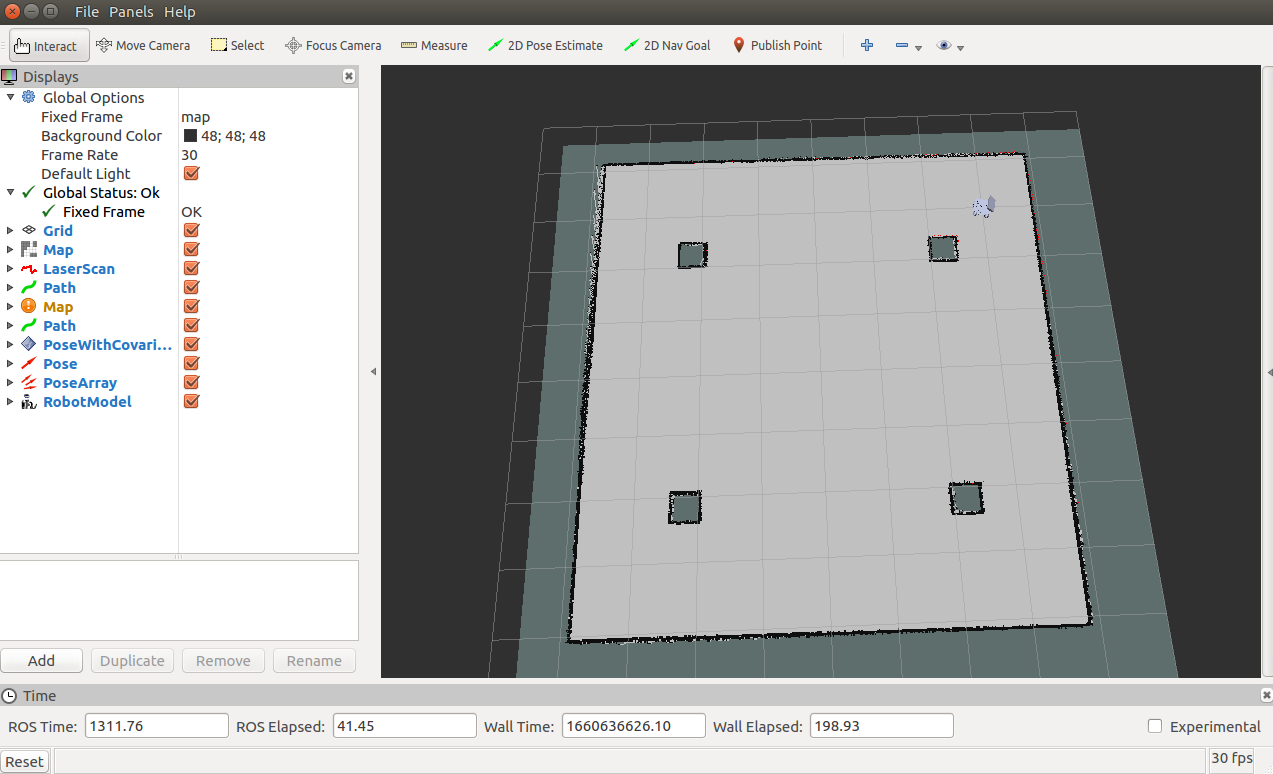


2、启动手动建图服务

|  |
| --- |
| zonesion@zonesion:~$ roslaunch mbot\_vcar gmapping\_demo.launch |

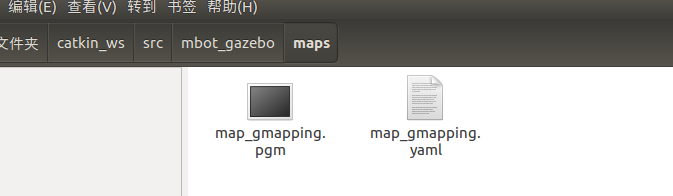


启动完成后如上



手动控制建图完成。输入命令保存地图

|  |
| --- |
| zonesion@zonesion:~$ **rosrun map\_server map\_saver -f map\_gmapping**  [ INFO] [1660636686.463674490]: Waiting for the map  [ INFO] [1660636686.681019142]: Received a 480 X 480 map @ 0.020 m/pix  [ INFO] [1660636686.681516833]: Writing map occupancy data to map\_gmapping.pgm  [ INFO] [1660636686.693338204]: Writing map occupancy data to map\_gmapping.yaml  [ INFO] [1660636686.695060438]: Done  zonesion@zonesion:~$ |



建图完成后替换如上文件，。

|  |
| --- |
|  |

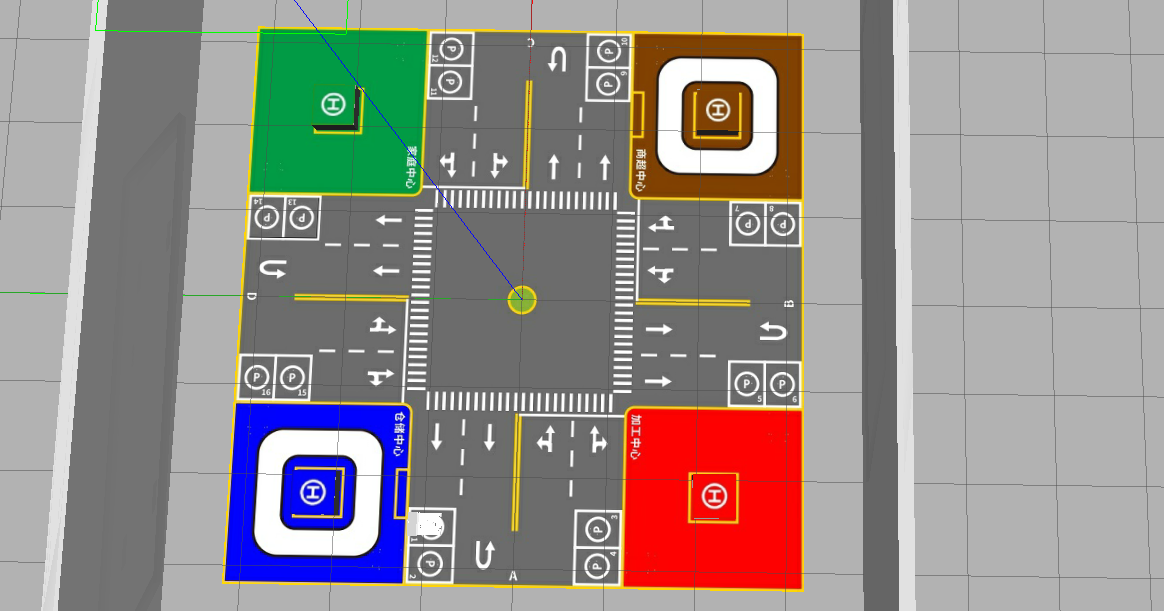
### 2、自动探索建图

|  |
| --- |
|  |

|  |
| --- |
|  |

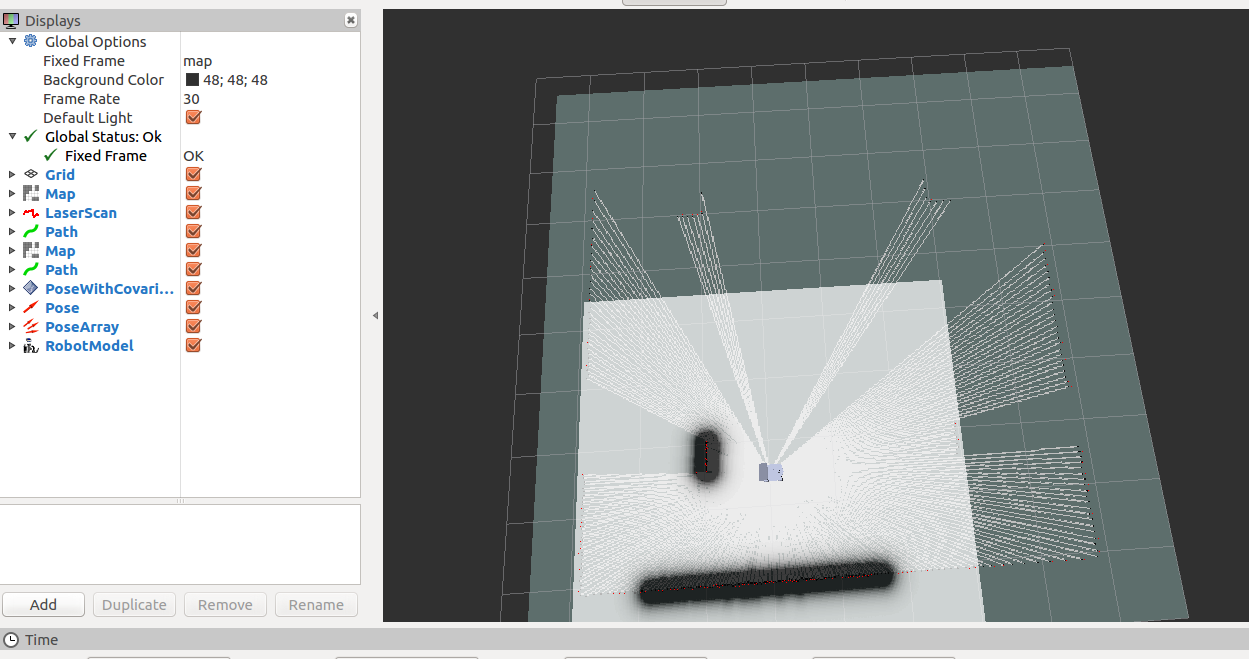
1、启动gazebo沙盘

|  |
| --- |
| zonesion@zonesion:~$ roslaunch mbot\_gazebo\_smartcity mbot\_gazebo\_smartcity.launch |



2、启动探索建图

|  |
| --- |
| zonesion@zonesion:~$ roslaunch mbot\_vcar exploring\_slam\_demo.launch |



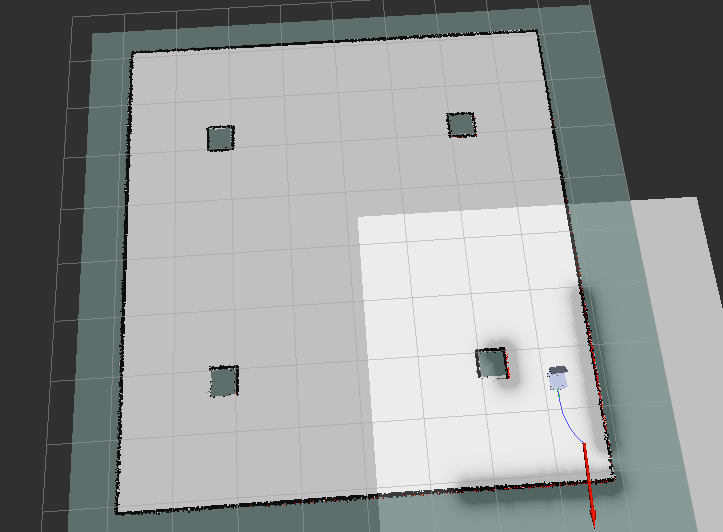
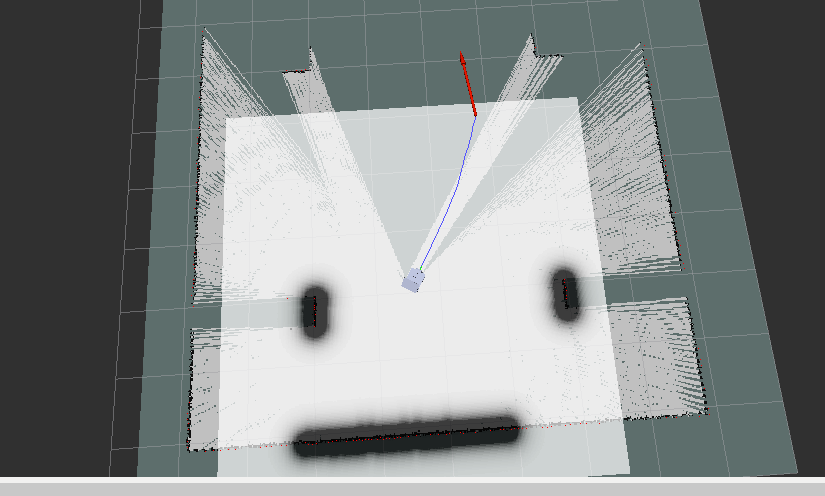
如果出现错误

ImportError: Twisted requires zope.interface 3.6.0 or later: no module named zope.interface.

|  |
| --- |
| zonesion@zonesion:~$ pip2 install zope.interface==3.6.0  DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. pip 21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support  Defaulting to user installation because normal site-packages is not writeable  Looking in indexes: http://mirrors.aliyun.com/pypi/simple/  Collecting zope.interface==3.6.0  Downloading http://mirrors.aliyun.com/pypi/packages/fa/b2/4e60f8d0f2446d7c5f01478321ef950bf79864304471c1e195ac7dcd0bfd/zope.interface-3.6.0.tar.gz (125 kB)  |████████████████████████████████| 125 kB 191 kB/s  Requirement already satisfied: setuptools in /usr/lib/python2.7/dist-packages (from zope.interface==3.6.0) (20.7.0)  Building wheels for collected packages: zope.interface  Building wheel for zope.interface (setup.py) ... done  Created wheel for zope.interface: filename=zope.interface-3.6.0-cp27-cp27mu-linux\_x86\_64.whl size=158105 sha256=d629e544402895cff32561b91ca0d1d2a3cedfa6210429ad5bf3a5b59c5e9e32  Stored in directory: /home/zonesion/.cache/pip/wheels/33/c5/ee/303d149d29e64a19728fe98c0d5b9759f3f90622e0b2d10a45  Successfully built zope.interface  Installing collected packages: zope.interface  Successfully installed zope.interface-3.6.0 |

使用这个命令安装

3、定位导航目标

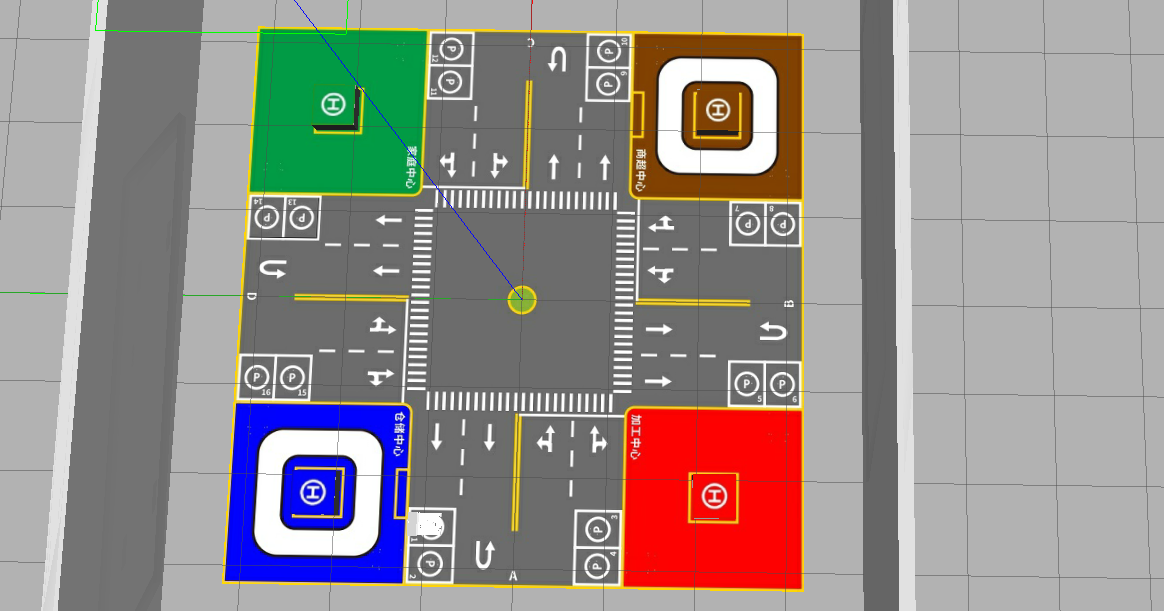


|  |
| --- |
|  |

### 3、自主导航

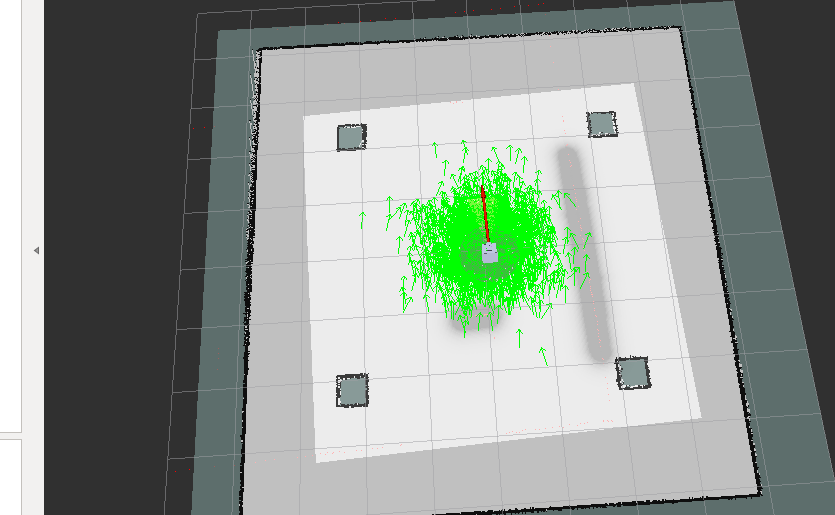
1、启动gazebo沙盘

|  |
| --- |
| zonesion@zonesion:~$ roslaunch mbot\_gazebo\_smartcity mbot\_gazebo\_smartcity.launch |

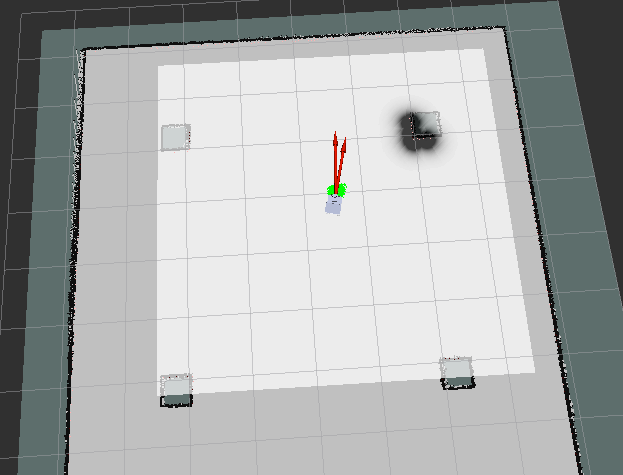
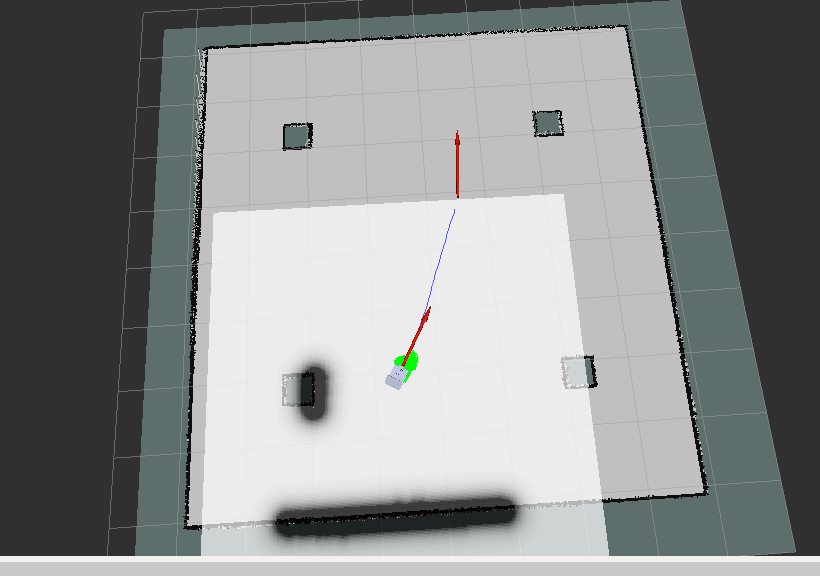
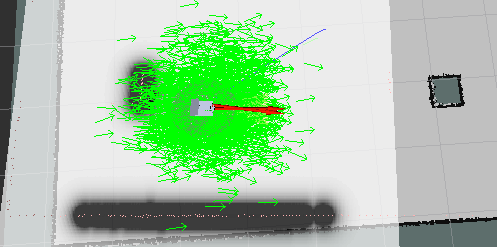


2、启动自主导航

|  |
| --- |
| zonesion@zonesion:~$ roslaunch mbot\_vcar nav\_demo.launch |



调整初始位置和启动导航



|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |