【古月居】古月·ROS 入门 21 讲 | 一学就会的 ROS 机器人入门教程_哔哩哔哩_bilibili

首先推荐大家先学习这门课程，对 ROS 系统有一个简要的认识，课程时间很短，有助快速了解。

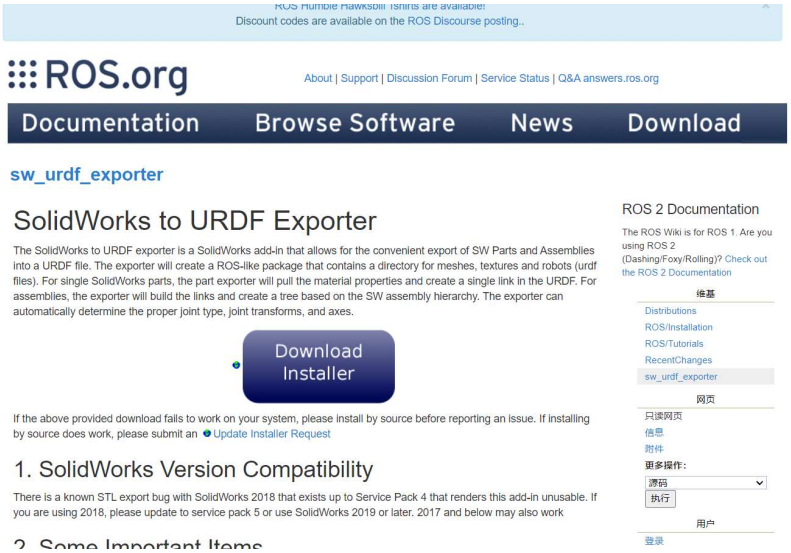
所需设备：Ubuntu18.04 以及配置好的 ROS 系统

第1章　利用 SW 建立机器人模型

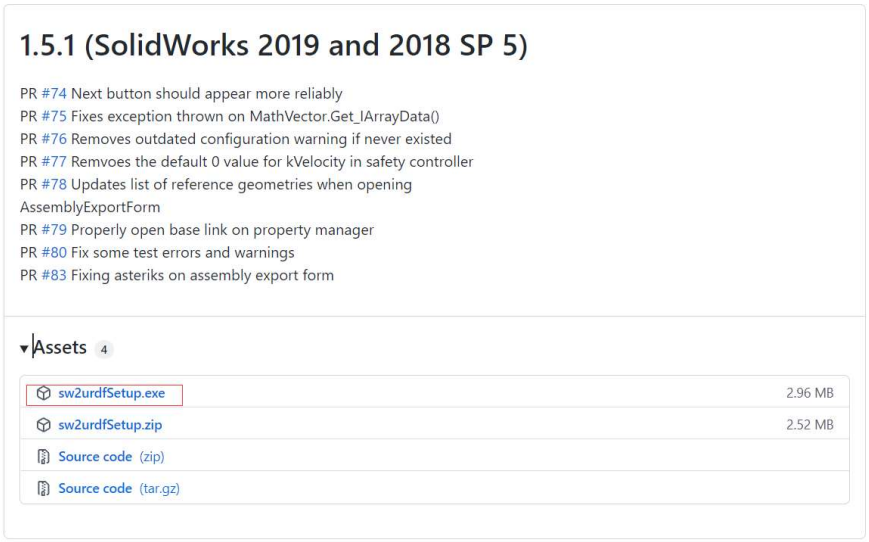
推荐使用 SW2016 版本，18 版本会出现模型导出错位现象等 BUG。

wiki.ros.org/sw_urdf_exporter

进入网址下载 SW TO URDF 插件

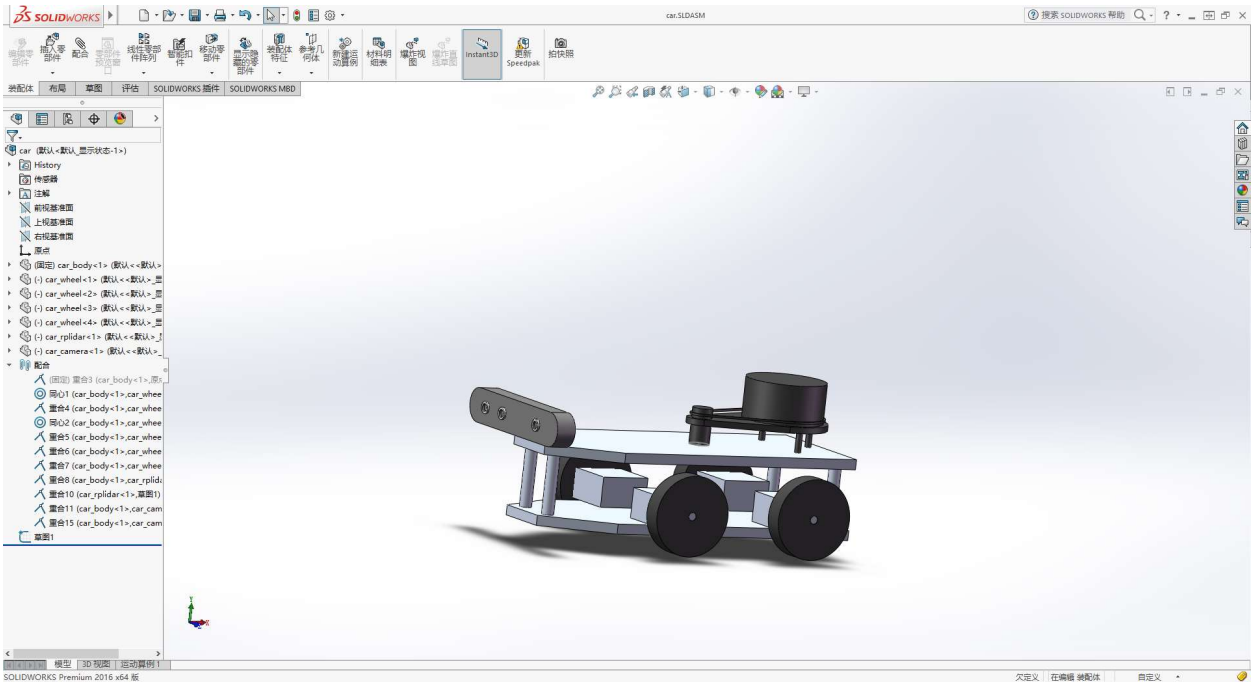


点击 download installer 下载并安装



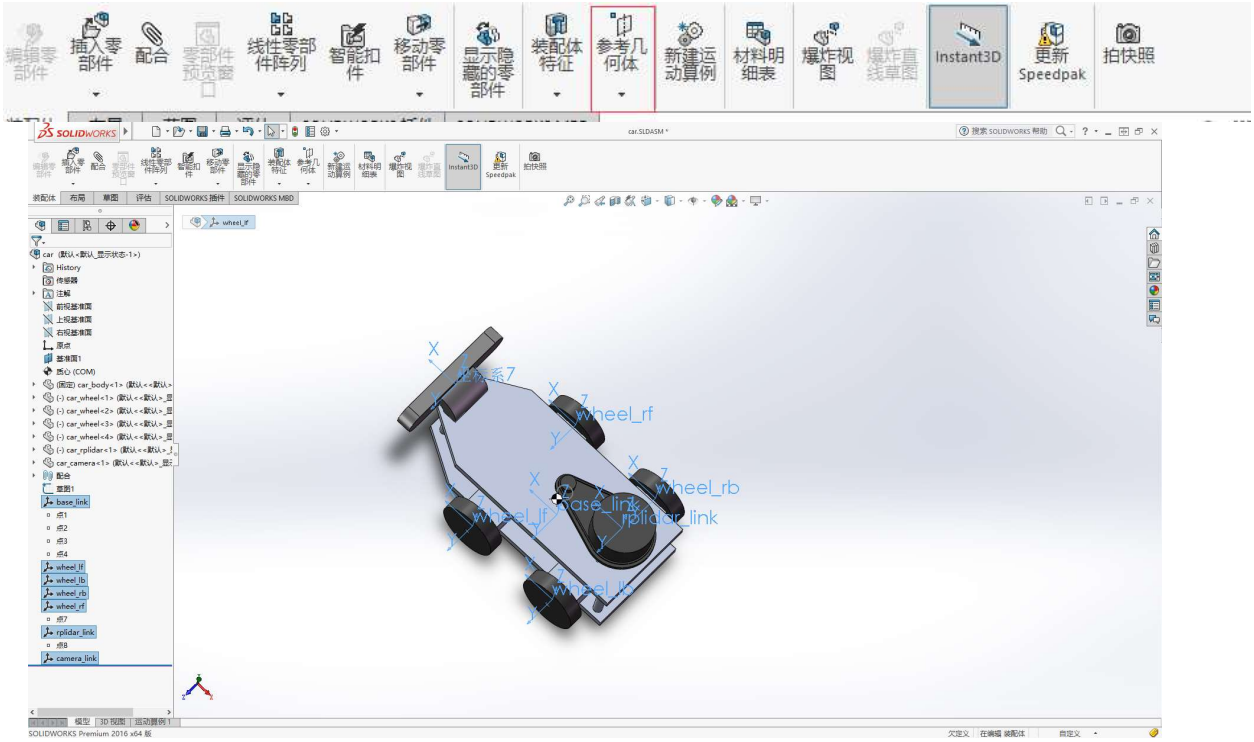
安装这个版本即可完美适配 SW2016。

安装好后进入 SW 建立机器人模型

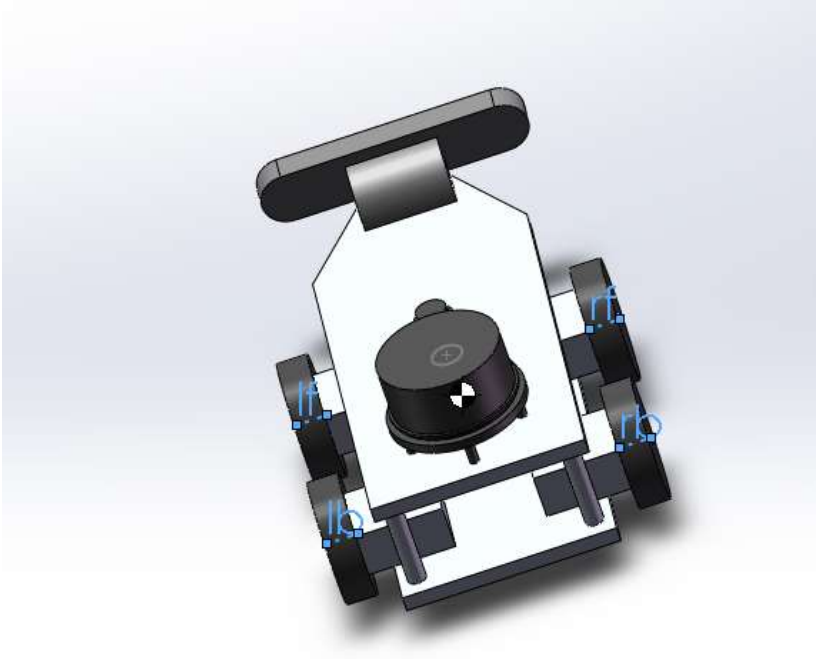


建模过程省略，建立一个四轮差速模型即可，无需一摸一样（激光雷达建立圆柱即可，无需像我一样）

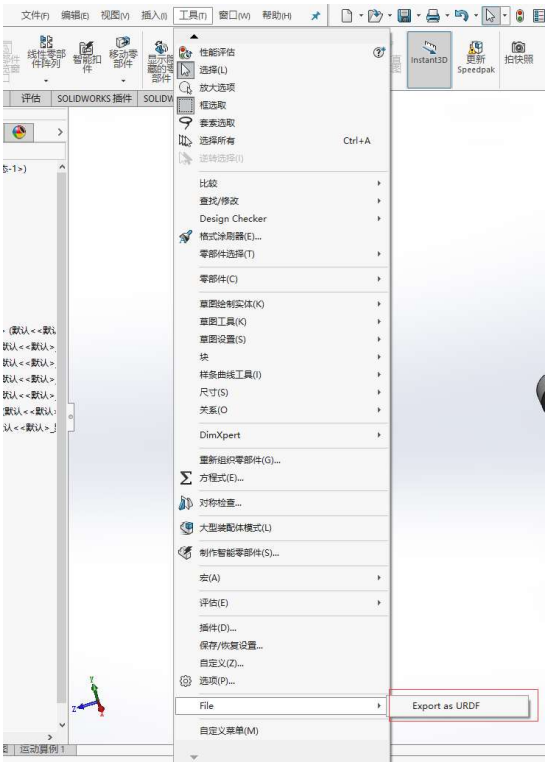
建立好模型后为每一个 link（零件）建立独立坐标系，X 轴方向指向机器人前进方向,Z 轴向上，右手定则确定 Y 轴（指向左转方向）

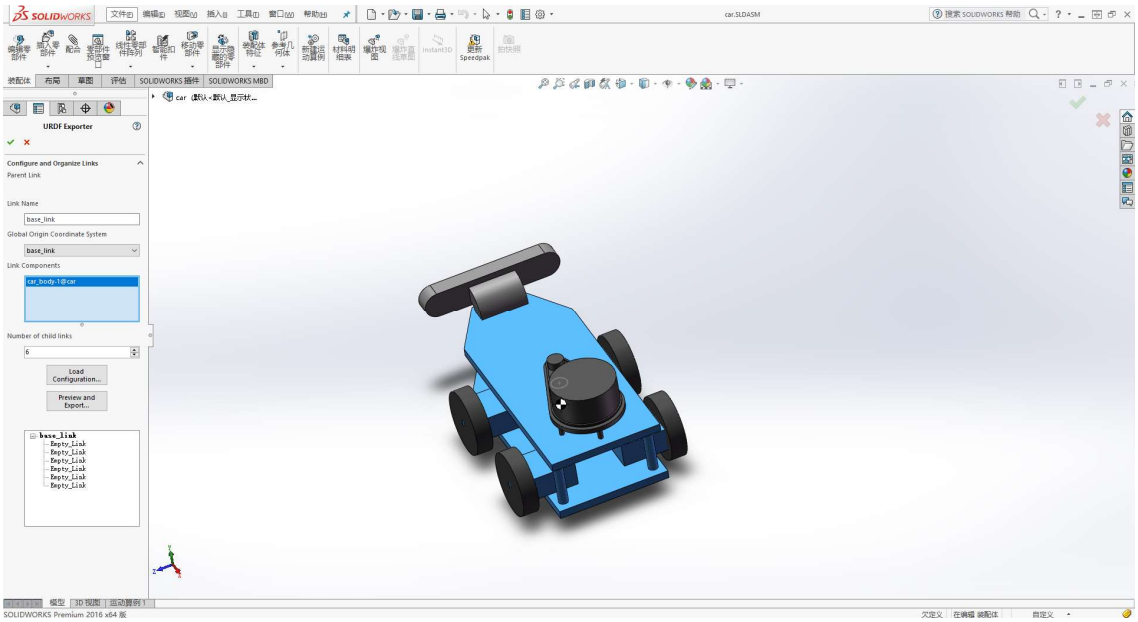


并为每个车轮设置旋转轴



点击如图选项



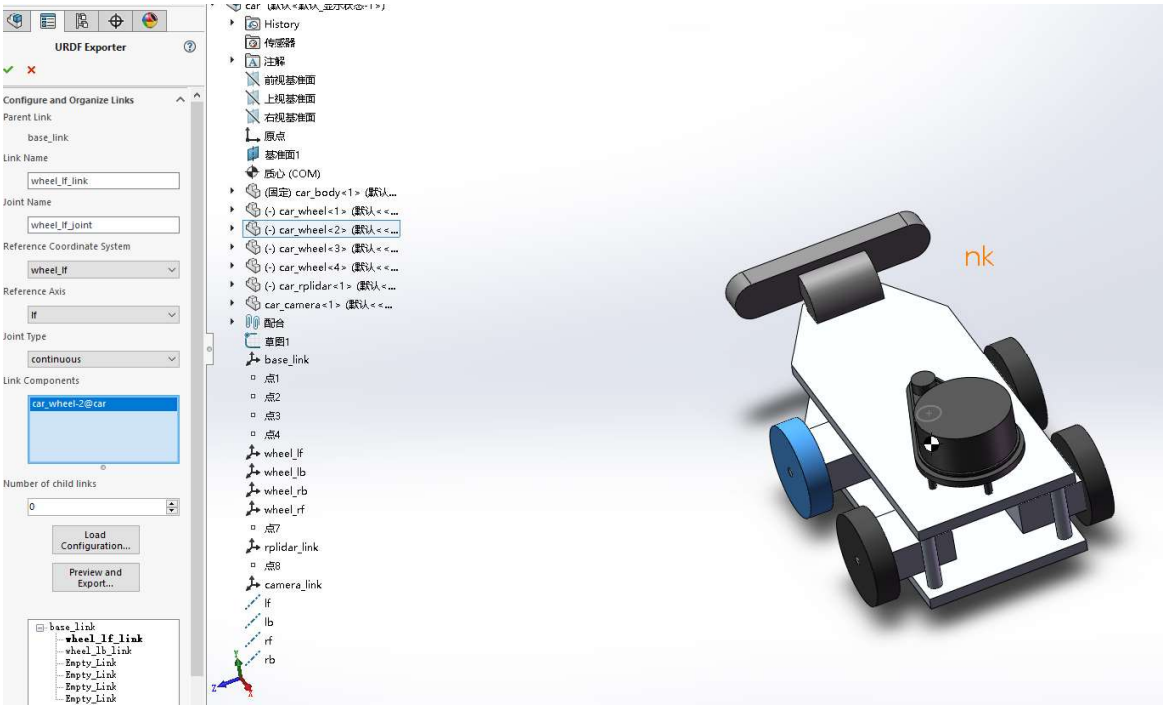


Link_name：自己设置，为 urdf 模型文件代码中 link 名

Global Origin Coordinate System：选择相应坐标系

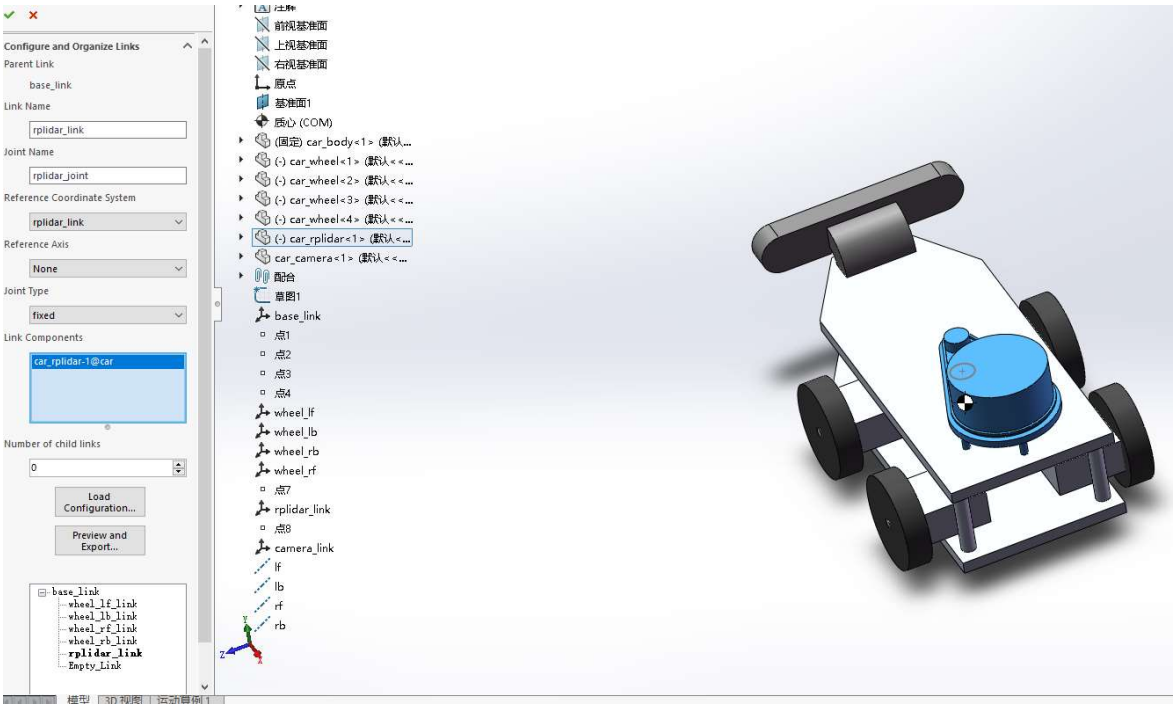
Link Components：选择对应零件

然后设置有 6 个子 link：四个车轮，相机，激光雷达

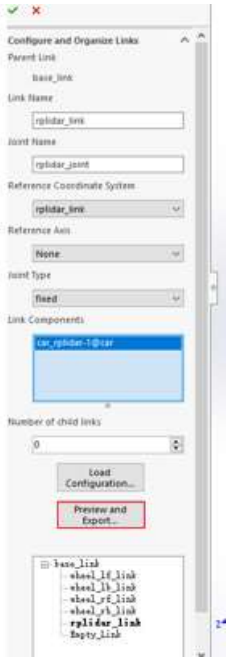


车轮如图设置

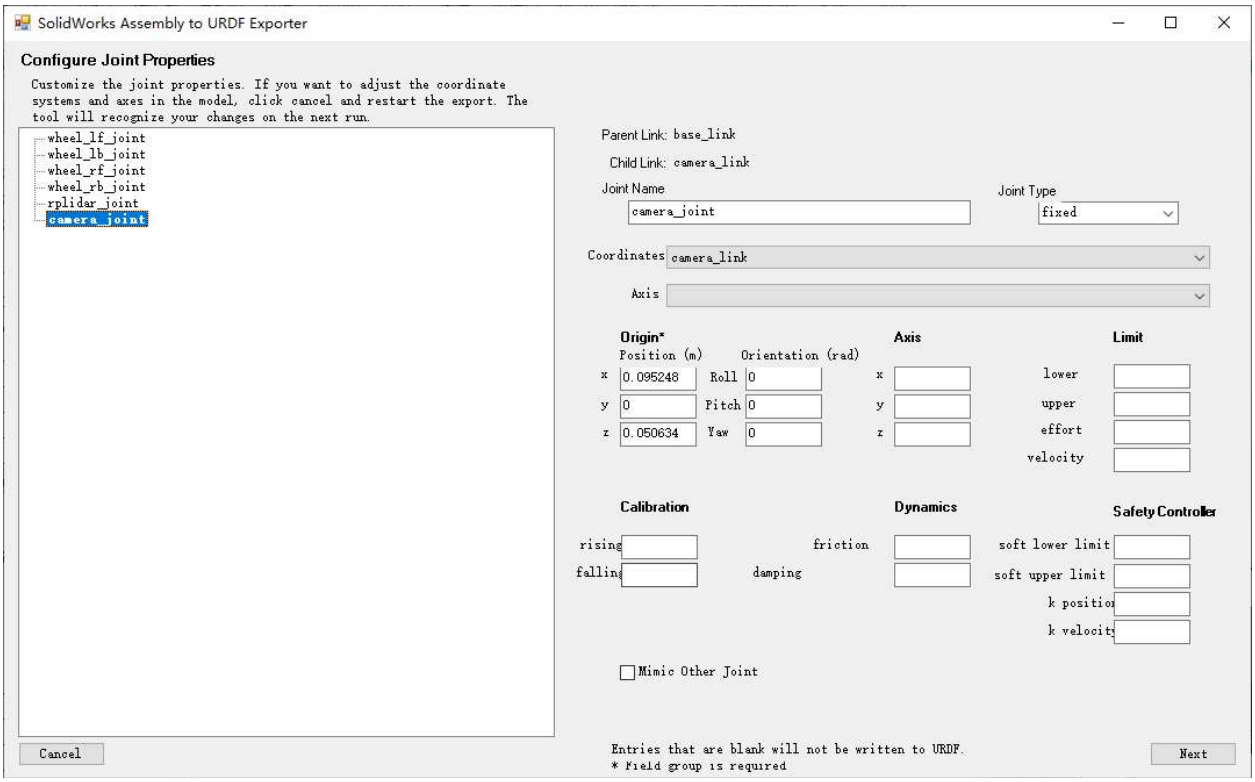
Joint 为关节：自己设置名字，然后选择坐标系，旋转轴，关节类型设置为 countinuous，无限位旋转，四个轮子依次设置。



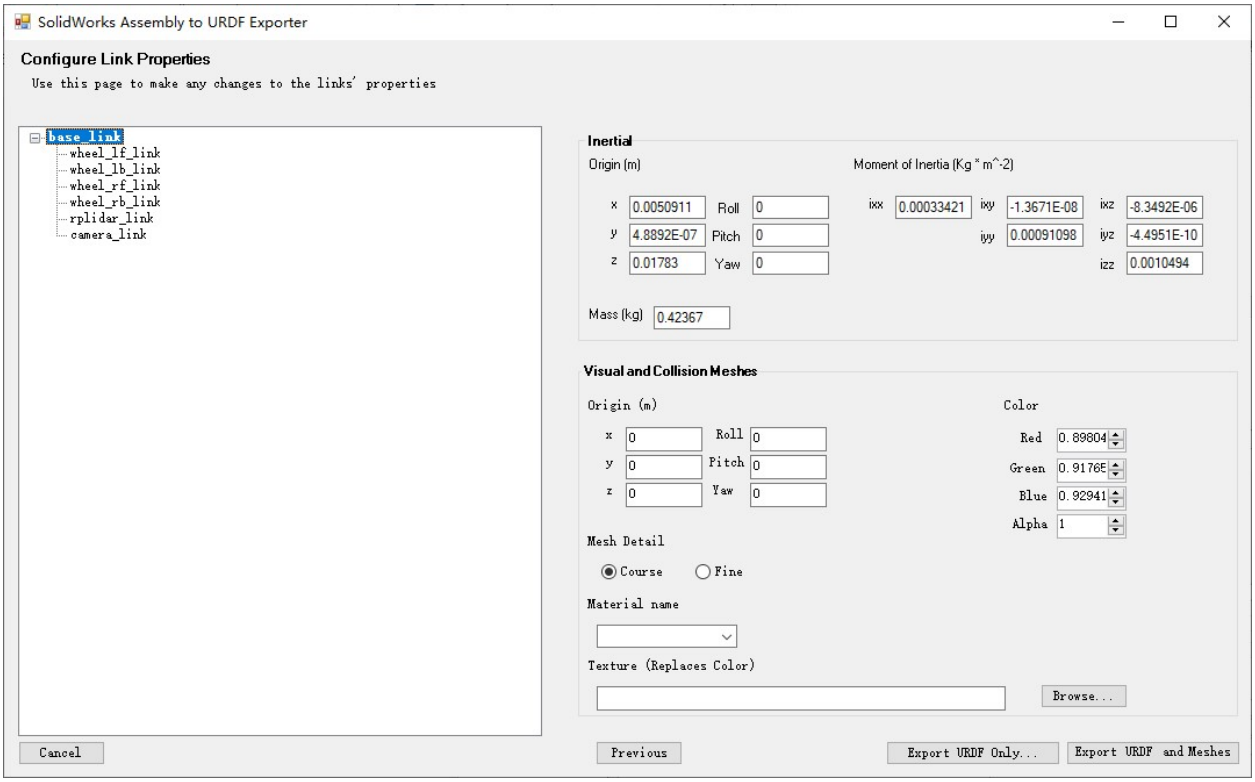
雷达和相机如图设置，选择 fixed 固定。



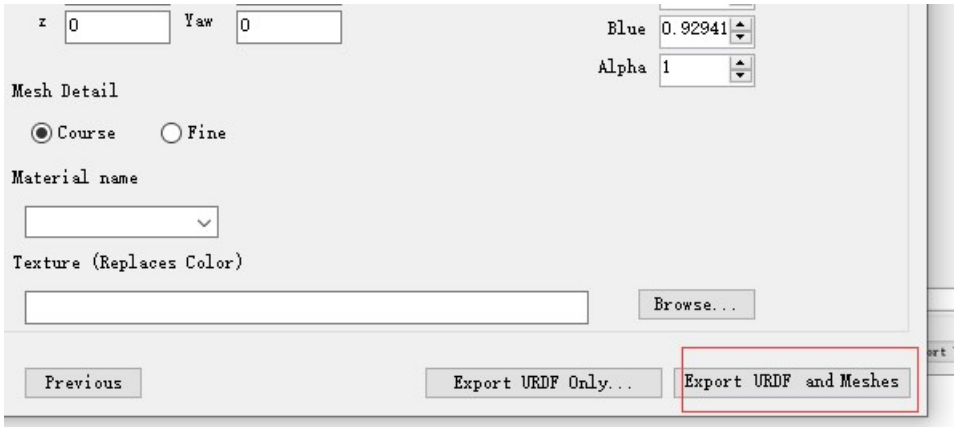
全部设置好后点击 Preview and export 导出功能包



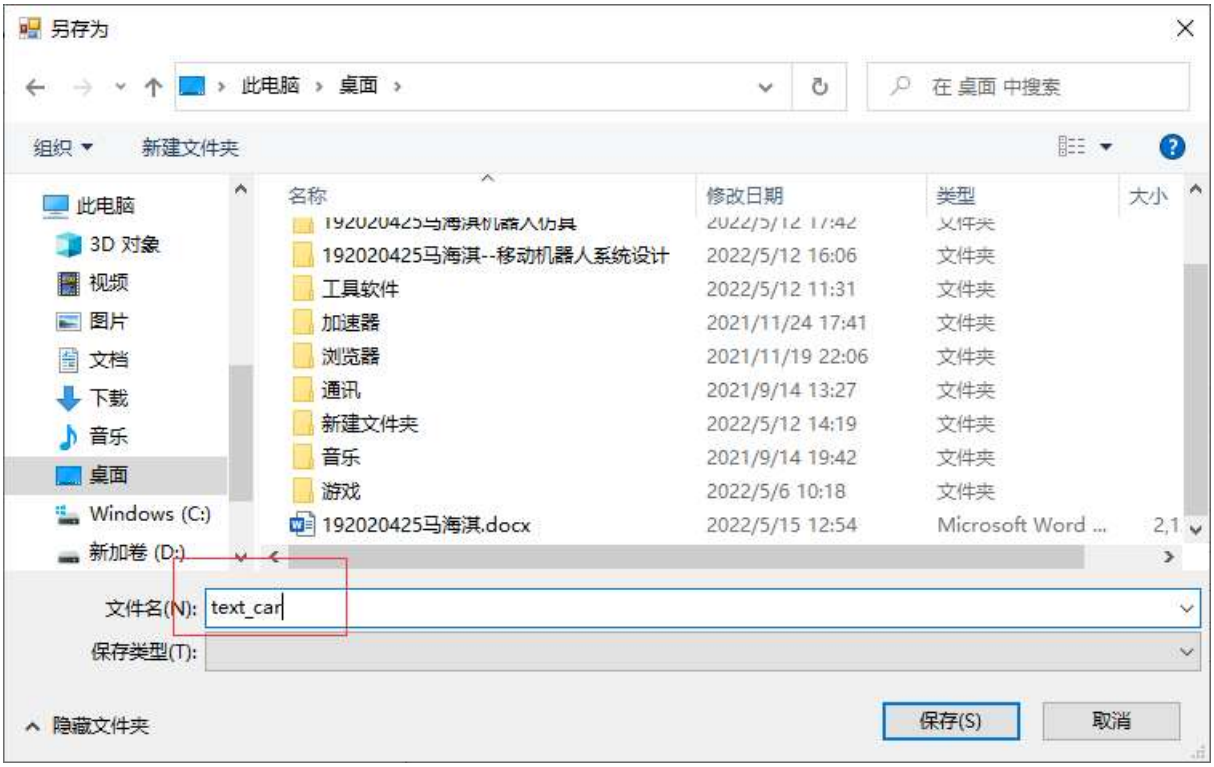
检查关节设置是否正确



检车 link 设置是否正确



点击导出功能包。



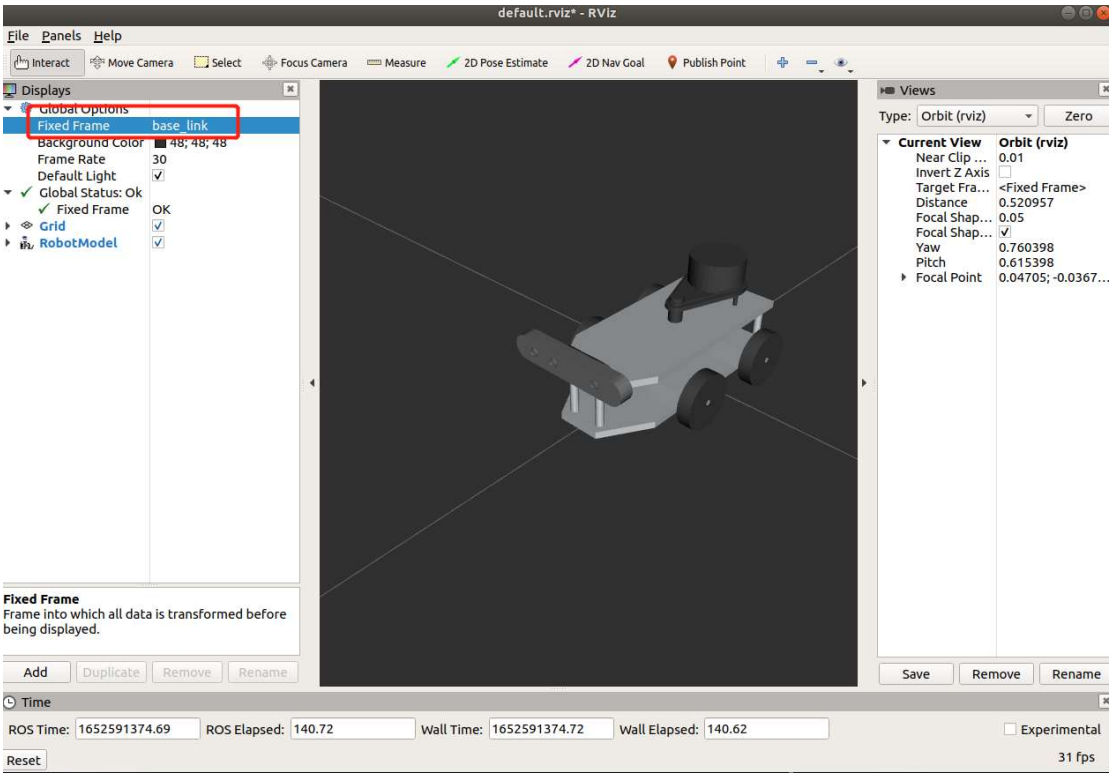
去掉后面的后缀。这样，我们第一步建立机器人基本模型就完成了，但是到这里我们只是建立了一个模型，没有任何系统。

第2章　ROS 系统中显示以及添加控制插件

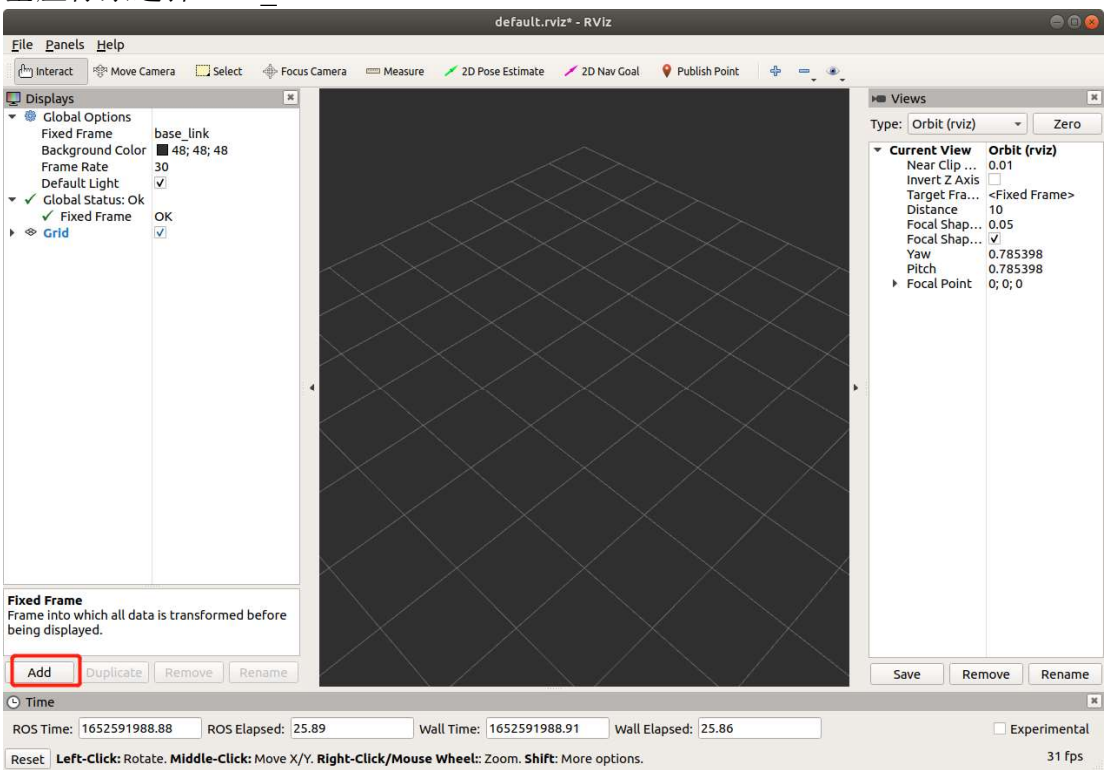
将打出的功能包复制到 Ubuntu 系统中的工作空间 src 路径下。（先看 ROS21 讲）

然后 catkin_make 编译工作空间

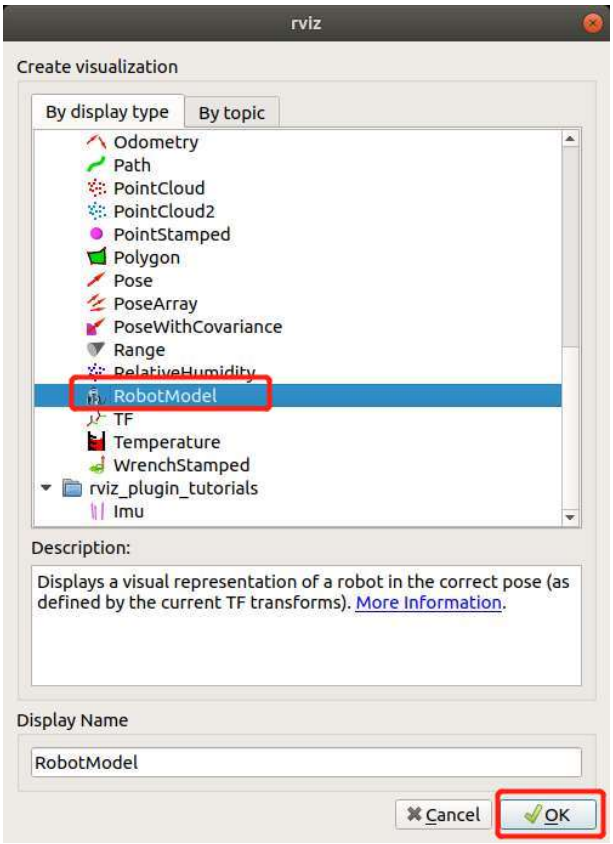
在终端输入 roslaunch text_car(功能包名) display.launch



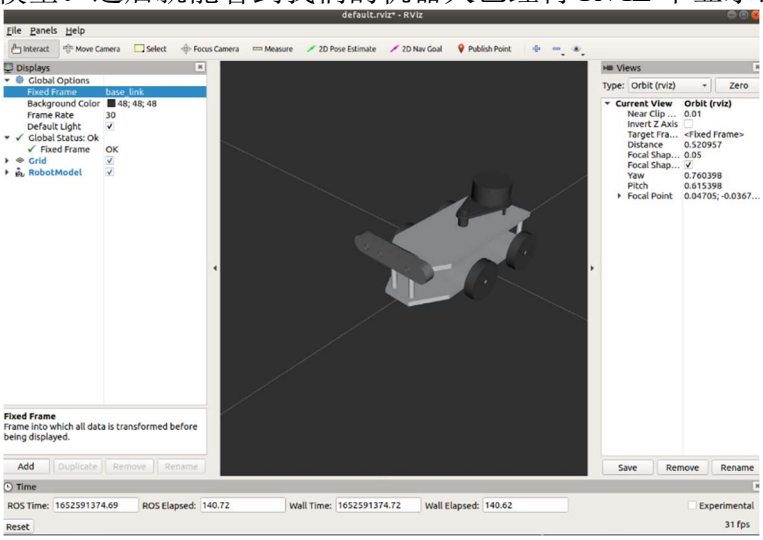
基座标系选择 base_link



选择 add

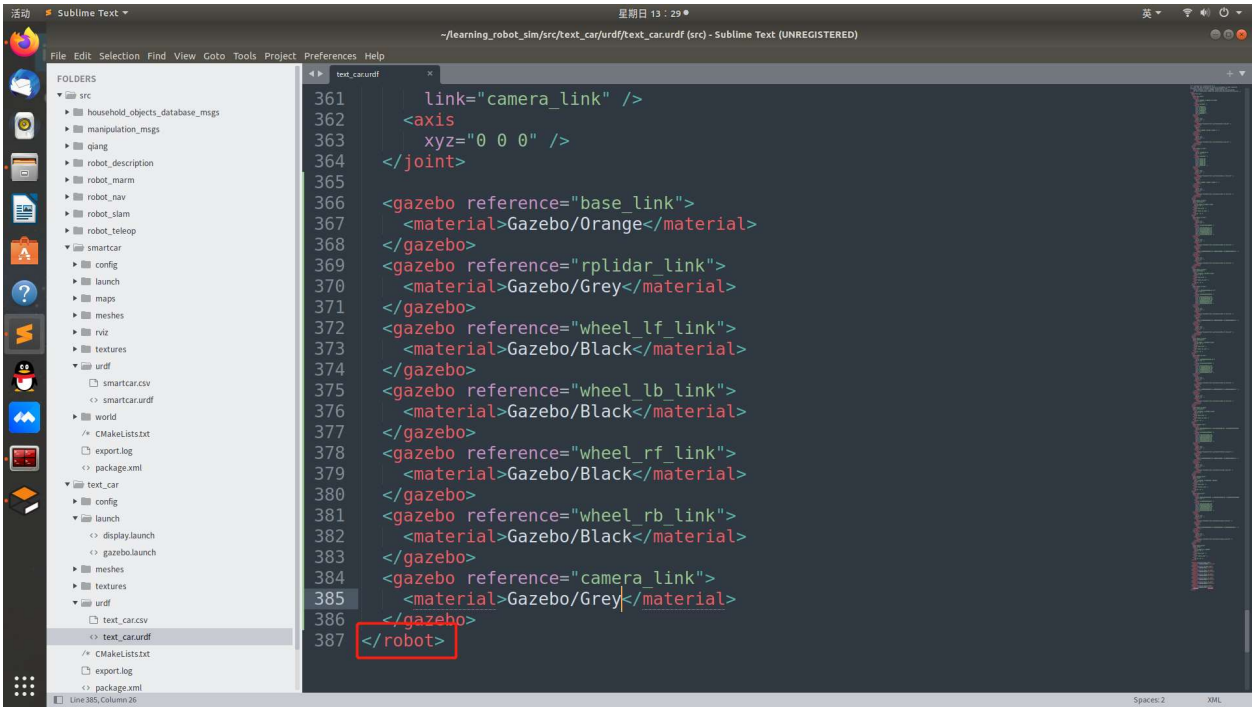


选择加入机器人模型。之后就能看到我们的机器人已经再 RVIZ 中显示出来了。



接着输入 roslaunch text_car gazebo.launch 就可以看到机器人模型再 gaebo 仿真环境中显示出来了，但是是纯灰色，接下来我们就来添加 gazebo 颜色标签。

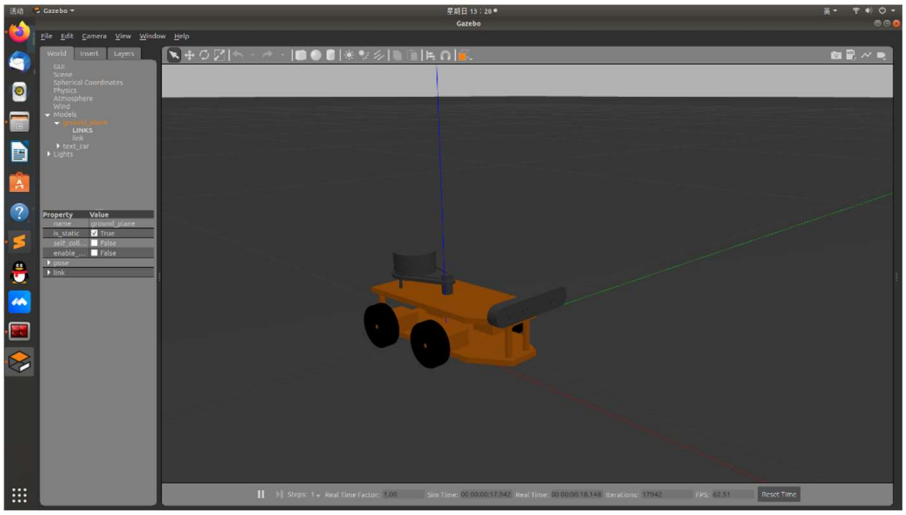
打开功能包 urdf 文件夹下的 urdf 文件，在其中插入（注意插入位置，）



要在 robot 标签以内位置插入如下代码

```
<gazebo reference="base_link">
  <material>Gazebo/Orange</material>
</gazebo>
<gazebo reference="rplidar_link">
  <material>Gazebo/Grey</material>
</gazebo>
<gazebo reference="wheel_lf_link">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="wheel_lb_link">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="wheel_rf_link">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="wheel_rb_link">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="camera_link">
  <material>Gazebo/Grey</material>
</gazebo>
```

再次 roslaunch text_car gazebo.launch 就可以看待机器人已经显示出我们所设置 的颜色了。



输入 rostopic list 我们可以看到，其中没有/cmd_vel 运动控制指令，也就是说，小车并不能进行运动控制，接下来我们就为小车添加运动控制插件。

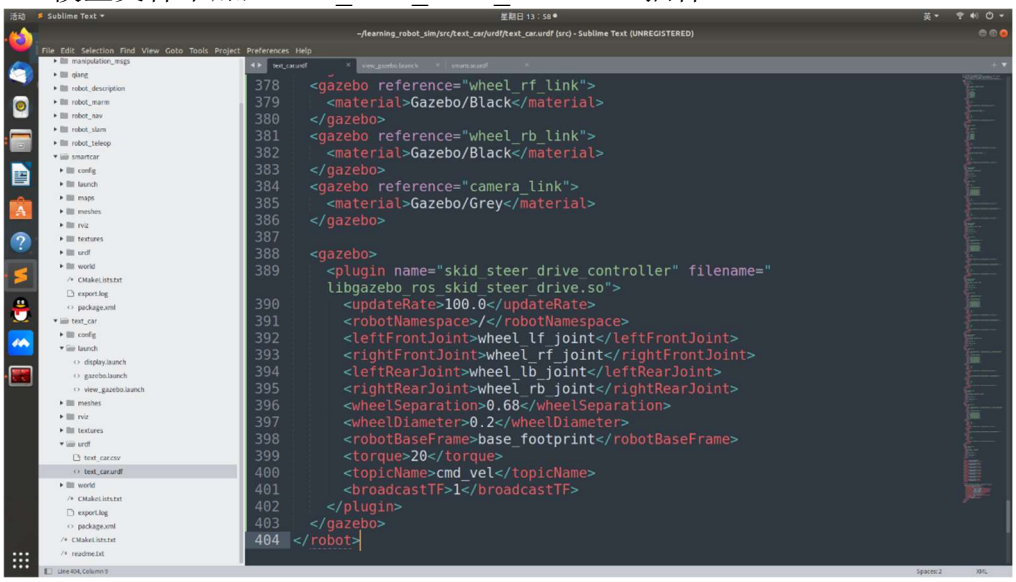
Gazebo 插件有很多

differential_drive_controller：这是一个差速控制插件，使用与两个驱动轮的机器人（很明显不适用于我们的机器人）

skid_steer_drive_controller：滑动转向插件，适用于四轮驱动，选用这款插件。

还有激光雷达插件，深度相机插件，具体到后面代码实现，我们不在这里赘述。

继续在 urdf 模型文件中加入 skid_steer_drive_controller 插件：



```
<gazebo reference="wheel_rf_link">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="wheel_rb_link">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="camera_link">
  <material>Gazebo/Grey</material>
</gazebo>

<gazebo>
  <plugin name="skid_steer_drive_controller" filename="
  libgazebo_ros_skid_steer_drive.so">
    <updateRate>100.0</updateRate>
    <robotNamespace>/</robotNamespace>
    <leftFrontJoint>wheel_lf_joint</leftFrontJoint>
    <rightFrontJoint>wheel_rf_joint</rightFrontJoint>
    <leftRearJoint>wheel_lb_joint</leftRearJoint>
    <rightRearJoint>wheel_rb_joint</rightRearJoint>
    <wheelSeparation>0.68</wheelSeparation>
    <wheelDiameter>0.2</wheelDiameter>
    <robotBaseFrame>base_footprint</robotBaseFrame>
    <torque>20</torque>
    <topicName>cmd_vel</topicName>
    <broadcastTF>1</broadcastTF>
  </plugin>
</gazebo>
</robot>
```

```xml
<gazebo>
  <plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.so">
    <updateRate>100.0</updateRate>
    <robotNamespace>/</robotNamespace>
    <leftFrontJoint>wheel_lf_joint</leftFrontJoint>
    <rightFrontJoint>wheel_rf_joint</rightFrontJoint>
    <leftRearJoint>wheel_lb_joint</leftRearJoint>
    <rightRearJoint>wheel_rb_joint</rightRearJoint>
    <wheelSeparation>0.68</wheelSeparation>
    <wheelDiameter>0.2</wheelDiameter>
    <robotBaseFrame>base_footprint</robotBaseFrame>
    <torque>20</torque>
    <topicName>cmd_vel</topicName>
    <broadcastTF>1</broadcastTF>
  </plugin>
</gazebo>
```

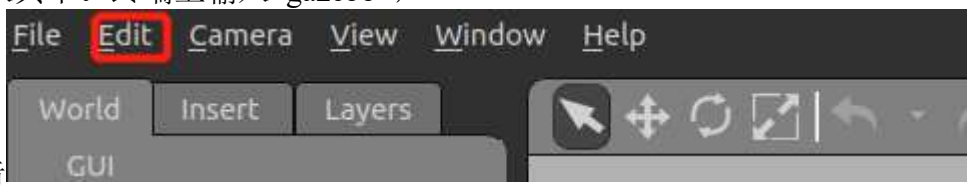
将如图所示代码加入 urdf 文件，此时还不能直接运行 gazebo.luanch 文件，因为其卵巢文件缺少很多东西，所以我们重写一份，在功能包路径下 luanch 文件夹内新建 view_gazebo.launch

```xml
<launch>

    <!-- 设置 launch 文件的参数 -->
    <arg name="world_name" value="$(find text_car)/world/smartcar_text.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>

    <!-- 加载机器人模型描述参数 -->
    <param
```

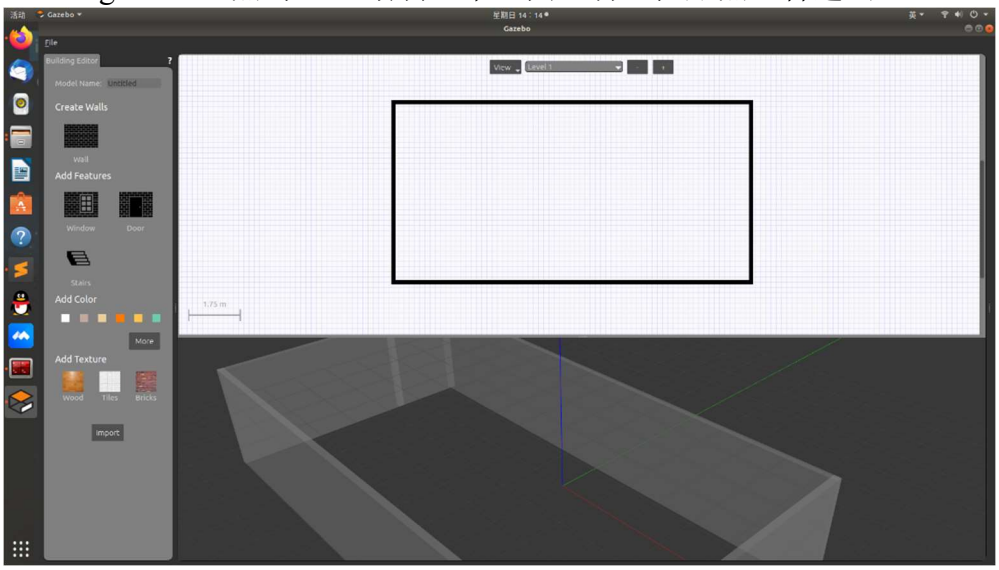
```
23.     name="robot_description"
24.     textfile="$(find text_car)/urdf/text_car.urdf" />
25.
26.     <!-- 运行 joint_state_publisher 节点，发布机器人的关节状态  -->
27.     <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint
    _state_publisher" ></node>
28.
29.     <!-- 运行 robot_state_publisher 节点，发布 tf  -->
30.     <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot
    _state_publisher"  output="screen" >
31.         <param name="publish_frequency" type="double" value="50.0" />
32.     </node>
33.
34.     <!-- 在 gazebo 中加载机器人模型-->
35.     <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="fal
    se" output="screen"
36.           args="-file $(find text_car)/urdf/text_car.urdf -urdf -
    model text_car"/>
37.     <node name="rviz" pkg="rviz" type="rviz" args="-
    d $(find text_car)/rviz/smart.rviz" required="true" />
38.
39. </launch>
```

其中代码主要修改位置为：

第 4 行：所创建世界路径为本功能包下 world 文件夹，没有就创建一个，然后将世界文件放入其中。终端上输入 gazebo ，

点击 

在点击 building editor，点击 wall 绘制一个地图（切记把原点包含进去）

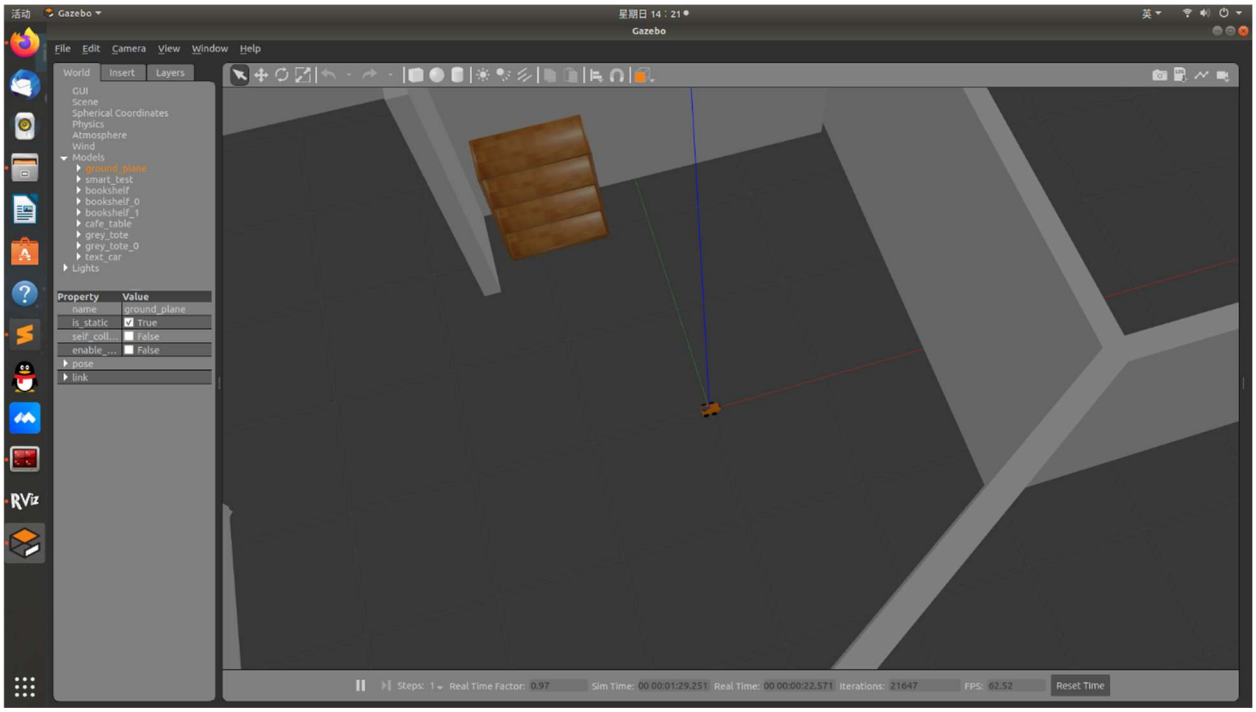


然后 Ctrl + S 保存一下，点击左上角 file，exit building editor ，就可以看到地图已经建立成功，保存地图为 launch 文件中 `smartcar_text.world`，也可自定义，但记得修改 launch 文件，最后将该地图文件复制到功能包 world 文件夹下。

24 36 行：记得修改路径。

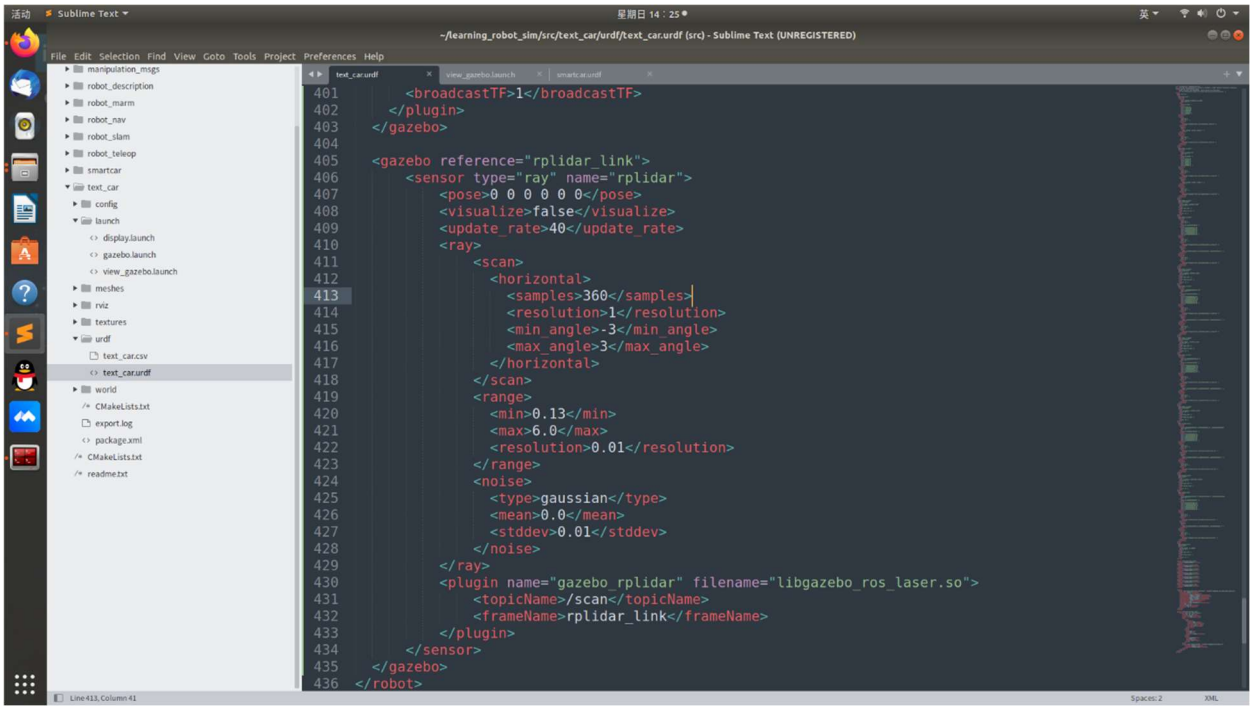
接下来 roslaunch text_car view_gazebo.launch

可以看到小车加载到了我们的世界中，



输入 rostopic list 可以看到已经有了/cmd_vel 这个 topic，说明小车已经可以运动了，通过给小车发送运动控制指令就可以控制小车运动了，这些东西自行学习我们先不在这里赘述，直接进入下一部分内容。

第3章　添加激光雷达插件

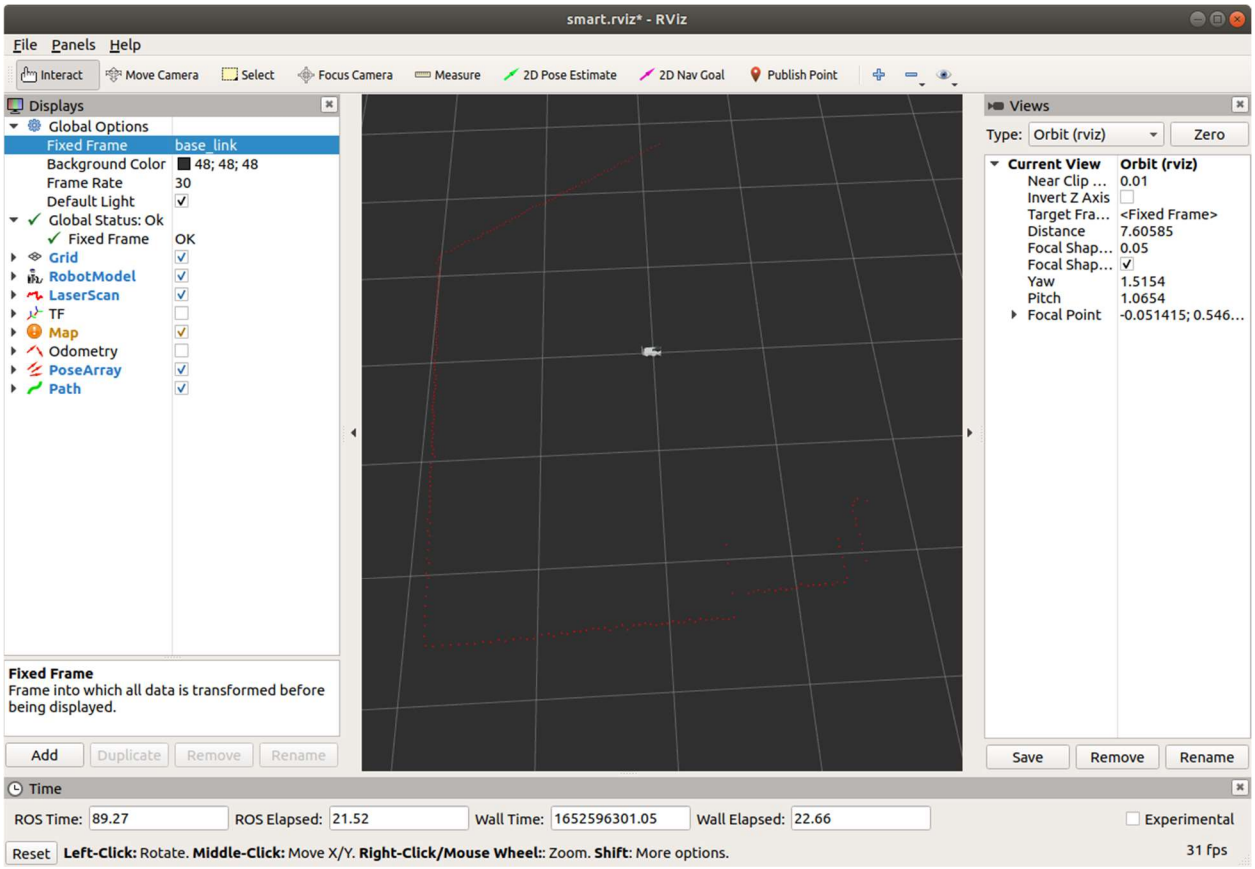


如图添加如下代码

```
<gazebo reference="rplidar_link">
    <sensor type="ray" name="rplidar">
        <pose>0 0 0 0 0 0</pose>
        <visualize>false</visualize>
        <update_rate>40</update_rate>
        <ray>
            <scan>
              <horizontal>
                <samples>360</samples>
                <resolution>1</resolution>
                <min_angle>-3</min_angle>
                <max_angle>3</max_angle>
              </horizontal>
            </scan>
            <range>
              <min>0.13</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>rplidar_link</frameName>
        </plugin>
    </sensor>
</gazebo>
```

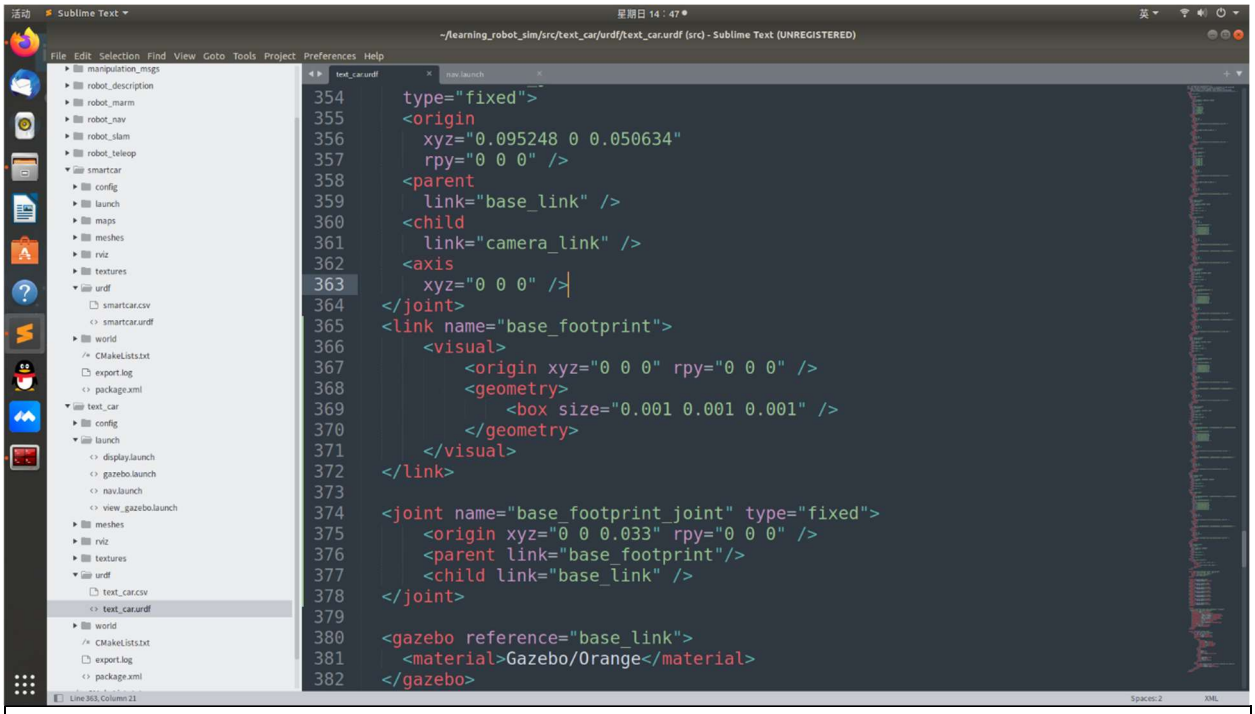
再次运动 view 文件就能在 rviz 看到激光雷达数据



第4章　SLAM 建图以及自主导航

SLAM 建图以及自主导航，这两者可以分开运用，但为了节约时间，也为了让我早点写完，这里我们一步到位，然后机器人边导航边建图。

第一步：现在 urdf 文件中加入一个 link 以及其 joint 叫做 base_footprint(具体作用自行 csdn 学习，我不在赘述)其代码如下图：

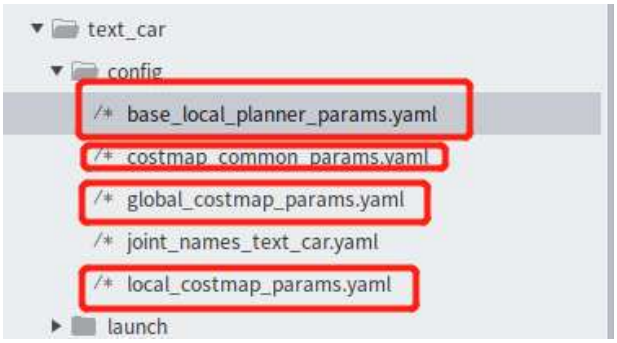



```
<link name="base_footprint">
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <box size="0.001 0.001 0.001" />
        </geometry>
    </visual>
</link>

<joint name="base_footprint_joint" type="fixed">
    <origin xyz="0 0 0.033" rpy="0 0 0" />
    <parent link="base_footprint"/>
    <child link="base_link" />
</joint>
```

第二步就要来写导航方面的配置文件了，这方面的知识不是我们这个教程能讲完的量。

(1 条消息) 机器人基于 ROS 的 move_base 导航功能的部署（代价地图）_wallEVA96 的博客-CSDN 博客_movebase 不能实时避障

给大家推荐一篇博客，但我还是推荐看书，由于现在快递不通，我这里有书，有兴趣同学来找我拿。



本地路径规划器

```
local_costmap:
  global_frame: map
  robot_base_frame: base_footprint
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 4.0
  height: 4.0
  resolution: 0.05
  origin_x: 0
  origin_y: 0
  transform_tolerance: 0.5
  plugins:
   - {name: voxel_layer,      type: "costmap_2d::VoxelLayer"}
   - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}
```

通用代价地图配置文件

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[0.06,0.04], [-0.06,0.04], [-0.06,-0.04], [0.06,-0.04]]
inflation_radius: 0.15
max_obstacle_height: 0.6
min_obstacle_height: 0.0
observation_sources: scan
scan: {data_type: LaserScan, topic: /scan, marking: true, clearing: tru
e, expected_update_rate: 0}
```

全局代价地图配置文件

```
global_costmap:
  global_frame: map
  robot_base_frame: base_footprint
  update_frequency: 5.0
  publish_frequency: 0.5
  static_map: true
  rolling_window: false
  transform_tolerance: 0.5
  plugins:
   - {name: static_layer,            type: "costmap_2d::StaticLayer"}
   - {name: voxel_layer,             type: "costmap_2d::VoxelLayer"}
```

```
  - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}
```

局部代价地图配置文件

```
local_costmap:
  global_frame: map
  robot_base_frame: base_footprint
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 4.0
  height: 4.0
  resolution: 0.05
  origin_x: 0
  origin_y: 0
  transform_tolerance: 0.5
  plugins:
   - {name: voxel_layer,      type: "costmap_2d::VoxelLayer"}
   - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}}
```

第三步编写建图导航 launch 文件

```
<launch>
   <include file="$(find text_car)/launch/view_gazebo.launch">
    </include>

     <node pkg="gmapping" type="slam_gmapping" name="simple_gmapping" output="screen">
          <param name="map_update_interval" value="1.0"/>
          <param name="maxUrange" value="5.0"/>
          <param name="maxRange" value="6.0"/>
          <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="minimumScore" value="50"/>
          <param name="srr" value="0.1"/>
          <param name="srt" value="0.2"/>
          <param name="str" value="0.1"/>
```

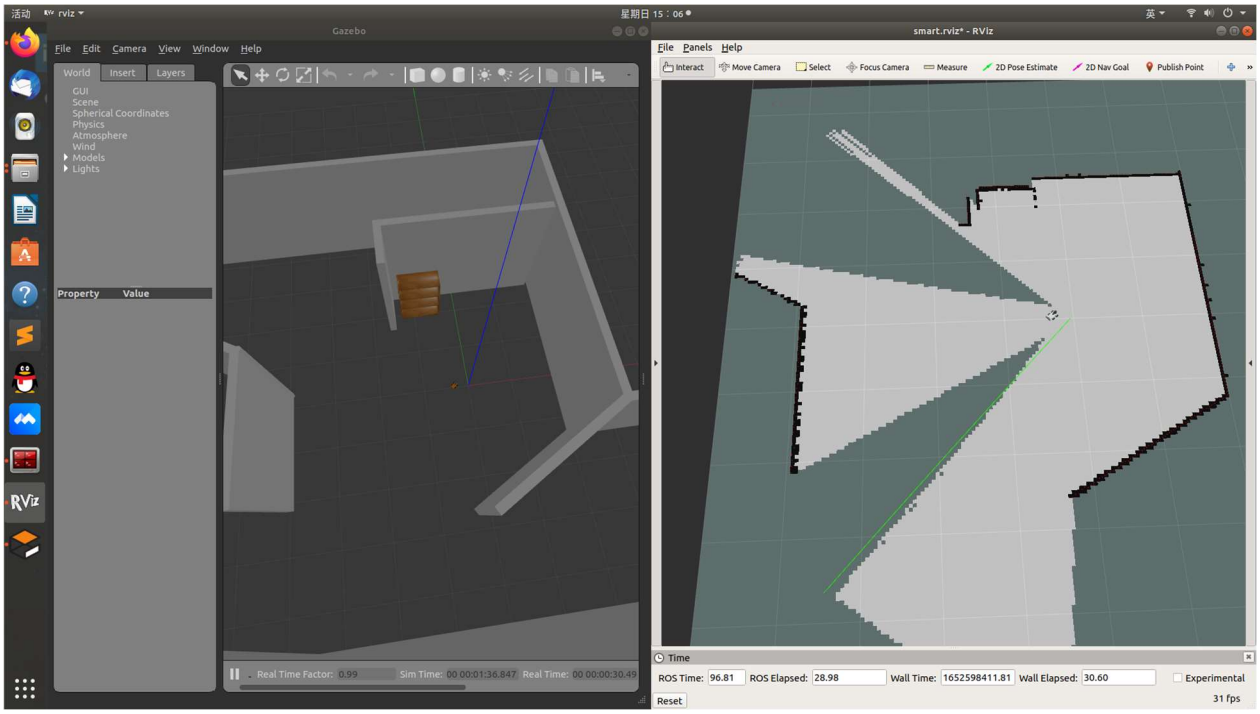
```
        <param name="stt" value="0.2"/>
        <param name="linearUpdate" value="1.0"/>
        <param name="angularUpdate" value="0.5"/>
        <param name="temporalUpdate" value="3.0"/>
        <param name="resampleThreshold" value="0.5"/>
        <param name="particles" value="50"/>
        <param name="xmin" value="-5.0"/>
        <param name="ymin" value="-5.0"/>
        <param name="xmax" value="5.0"/>
        <param name="ymax" value="5.0"/>
        <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"/>
        <param name="base_frame" value="/base_link"/>
    </node>

  <!-- 启动 move_base 节点 -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen" clear_params="true">
    <rosparam file="$(find text_car)/config/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find text_car)/config/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find text_car)/config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find text_car)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find text_car)/config/base_local_planner_params.yaml" command="load" />
  </node>

  <!-- 启动 AMCL 节点 -->
  <node pkg="amcl" type="amcl" name="amcl" clear_params="true">
    <param name="use_map_topic" value="false"/>
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha5" value="0.1"/>
    <param name="gui_publish_rate" value="10.0"/>
    <param name="laser_max_beams" value="60"/>
    <param name="laser_max_range" value="12.0"/>
    <param name="min_particles" value="500"/>
```

```
    <param name="max_particles" value="2000"/>
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.99"/>
    <param name="odom_alpha1" value="0.2"/>
    <param name="odom_alpha2" value="0.2"/>
    <param name="odom_alpha3" value="0.2"/>
    <param name="odom_alpha4" value="0.2"/>
    <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_model_type" value="likelihood_field"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d" value="0.25"/>
    <param name="update_min_a" value="0.2"/>

    <param name="odom_frame_id" value="odom"/>

    <param name="resample_interval" value="1"/>
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance" value="1.0"/>
    <param name="recovery_alpha_slow" value="0.0"/>
    <param name="recovery_alpha_fast" value="0.0"/>
    <!-- scan topic -->
    <remap from="scan" to="scan"/>
  </node>

</launch>
```

这一章节涉及到的算法非常多，例如我们运用到的 gmapping 建图算法，Dijkstra 全局路径规划算法，DWA 局部路径规划算法，以及 AMCL 定位算法。我不可能跟大家一一讲解，还是需要大家自己去理解，这些算法作用，如何运用。

到此为止，我们的机器人就可以进行建图导航了，编译功能包，然后

roslaunch text_car nav.launch



如图，我们的小车已经开始按照我们用 2D Nav Goal 设定的目标进行运动了，并且运动过程成还实时建立地图。如此，我们就完成了这次教程的全部内容，希望大家好好学习。还有什么问题直接在群里提问，就不要私聊我了，直接在群里问，这样大家都能有个参考。