移动机器人开发技术（激光SLAM版）配套教学PPT

序　章

第1课 移动机器人的过去、现在及未来

机器人操作系统

第2课 初识ROS

第3课 ROS编程初步

第4课 机器人的坐标变换

第5课 机器人仿真环境

机器人硬件平台

第6课 TurtleBot3仿真环境实战

第7课 自主搭建机器人小车

机器人核心技术

感知	建图与定位	路径规划与导航
第08课 环境感知基础	第10课 机器人的移动控制	第13课 导航基础
第09课 感知数据融合	第11课 SLAM基础	第14课 ROS中的导航包
	第12课 SLAM实战	第15课 ROS导航实战

机器人应用实战

送餐	物流（专题讲座）	图书盘点（专题讲座）
1 送餐机器人结构设计	1 物流机器人结构设计	1 图书盘点机器人结构
2 送餐机器人环境搭建	2 物流机器人环境模拟	2 图书盘点机器人环境
3 送餐机器人建图	3 物流机器人关键技术	3 图书盘点机器人工作模式
4 送餐机器人导航	4 大规模多机器人调度	4 图书盘点中的视觉分析

移动机器人开发技术（激光SLAM版）配套教学PPT

第五课 机器人仿真环境

北京郵電大學
Beijing University of Posts and Telecommunications

移动机器人与智能技术实验室编

宋桂岭 明安龙 2021.10
expsong@qq.com

第5课 机器人仿真环境

北邮移动机器人与智能技术实验室 编

1. 统一机器人描述格式URDF
2. 机器人可视化工具RViz
3. ROS仿真工具Gazebo
4. Launch文件

第5课 机器人仿真环境

北邮移动机器人与智能技术实验室 编

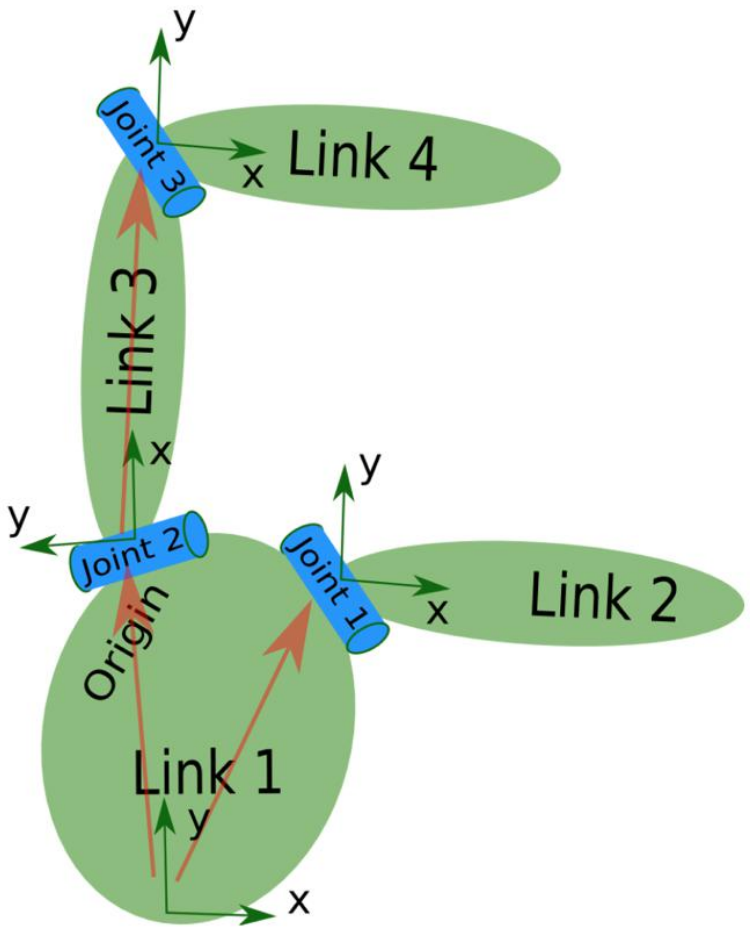
1 统一机器人描述格式URDF

1.1 机器人结构回顾

机器人的基本结构由“**关节**”和“**连杆**”组成。

关节：关节（joint）即运动副(kinematic pairs)，是允许机器人手臂各零件之间发生相对运动的机构，是两构件直接接触并能产生相对运动的活动连接，A、B两部件可以做互动连接。

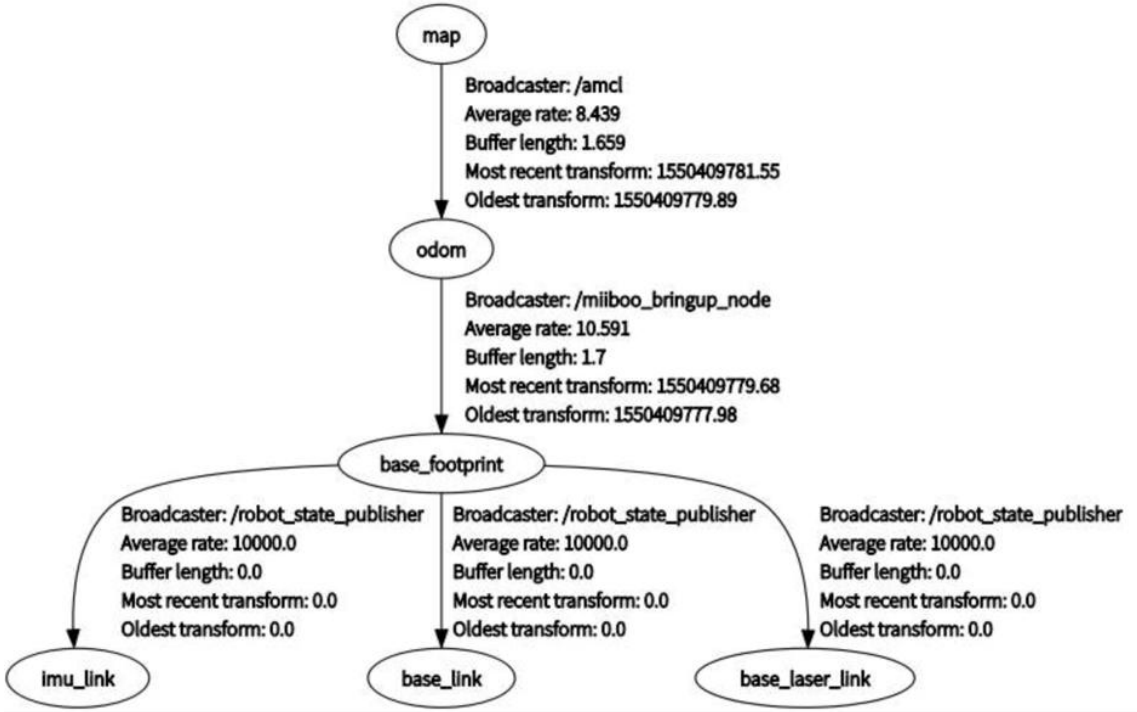
连杆：连杆（link）指机器人手臂上被相邻两关节分开的部分，是保持各关节间固定关系的刚体，是机械连杆机构中，两端分别与主动和从动构件铰接，以传递运动和力的杆件。连杆是机器人中的重要部件，它连接着关节，其作用是将一种运动形式转变为另一种运动形式，并把作用在主动构件上的力传给从动构件，以输出功率。



1.2 ROS TF概念回顾

```
std_mags/Header header
        uint32 seq
        time stamp
        string frame_id
string child_frame_id
geometry_msgs/Transform transform
        geometry_msgs/Vector3 translation
                float64 x
                float64 y
                float64 z
        geometry_msgs/Quaternion rotation
                float64 x
                float64 y
                flaot64 z
                float64 w
```

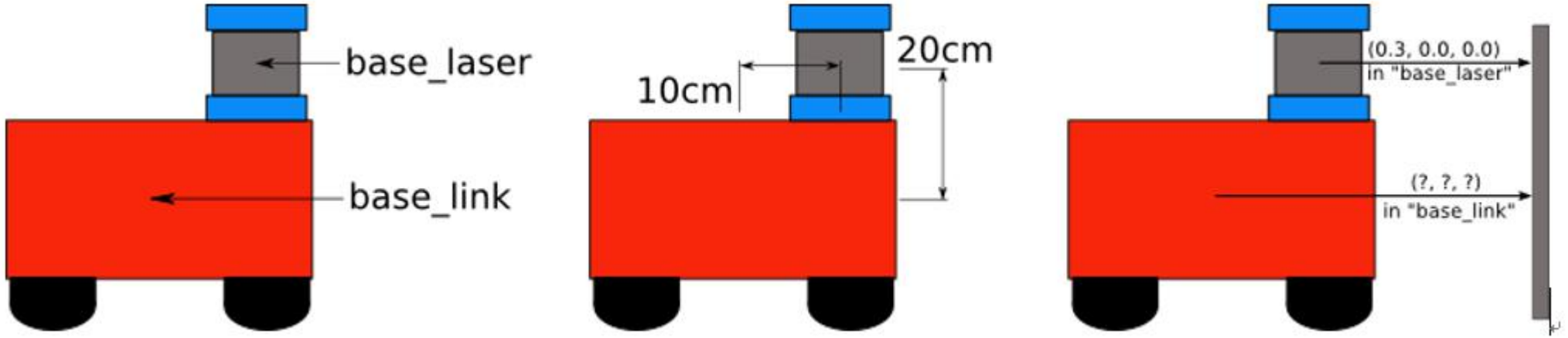
TF数据类型TransformStamped.msg



每一个Link部件拥有一个坐标系（Frame），构成了TF树

1.3 机器人结构的话题发布

定义两个坐标系：
base_link参考坐标系：以机器人移动平台的中心为原点
base_laser参考坐标系：以激光雷达的中心为原点

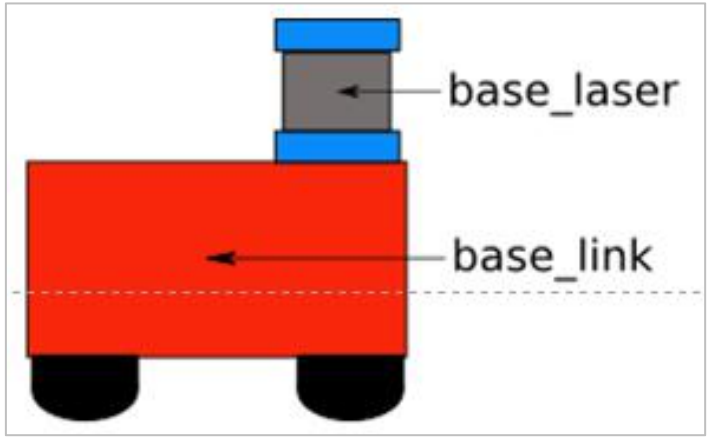


在ROS的tf变换树中定义了不同坐标系之间的平移与旋转变换关系，tf功能包提供了存储、计算不同数据在不同参考系之间变换的功能，因此只需要告诉tf树这些参考系之间的变换公式即可。

```
tf::Transform transform;   // 初始化tf数据
transform.setOrigin( tf::Vector3(0.1, 0.0, 0.2) )  //定义了两个坐标系之间的变换关系
transform.setRotation( tf::Quaternion(0,0,0,1) ); //定义了两个坐标系之间的旋转关系
static tf::TransformBroadcaster br;    // 广播坐标关系
br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "base_link", "base_laser"));
```

1.4 统一机器人描述格式URDF

以上关于机器人结构的程序代码，可以通过配置文件来表述，从而提高机器人结构的复用性



底盘描述

```
<link name="base_link">
  <visual>
    <geometry>
      <box size=".2 .3 .3"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0" />
    <material name="red">
      <color rgba=".8 .3 0 1" />
    </material>
  </visual>
</link>
```

雷达描述

```
<link name="base_laser">
<visual>
  <geometry>
    <cylinder length="0.25"
radius="0.05" />
  </geometry>
  <origin rpy="0 0 0" xyz="0 0 0" />
  <material name="blue">
    <color rgba="0.5 .8 1" />
  </material>
</visual>
</link>
```

关节描述

```
<joint name="laser_joint" type="fixed">
<axis xyz="0 1 0" />
<origin xyz="0.1 0 0.2" rpy="0 0 0"/>
<parent link="base_link"/>
<child link="laser_link"/>
</joint>
```

1.4 统一机器人描述格式URDF

URDF的基础标签

<robot>标签

<robot>标签是完整机器人模型的最顶层标签，其他所有标签都必须包含在<robot>标签内

<link>标签

<link>标签用于描述机器人某个刚体部分的外观和物理属性，包括尺寸（size）、颜色（color）、形状（shape）、惯性矩阵（inertial matrix）、碰撞参数（collision properties）等

<joint>标签

<joint>标签用于描述机器人关节的运动学和动力学属性，包括关节运动的位置和速度限制

关节类型	描述
Continuous	旋转关节，可以围绕单轴无限旋转
Revolute	旋转关节，类似于continuous，但是有旋转角度限制
Prismatic	滑动关节，延某一轴线移动的关节，带有位置极限
Planar	平面关节，允许在平面正交方向上平移或旋转
Floating	浮动关节，允许进行平移、旋转运动
Fixed	固定关节，不允许运动的特殊关节

1.5 URDF示例：第一个机器人littlecar

1.5.1 创建工作空间

1、创建ROS工作空间，命名为littlecar，并在该工作空间中创建src文件。

```
mkdir -p littlecar/src
```

2、初始化工作空间：

```
cd littlecar/
catkin_make
```

3、创建ROS机器人小车描述包：

```
cd src
catkin_create_pkg littlecar_description urdf roscpp rospy
```

1.5 第一个机器人littlecar

1.5.2 创建机器人格式描述文件

初始化工作空间后，在littlecar_description文件夹下创建urdf文件夹，并创建小车的描述文件littlecar.urdf。

```
<?xml version="1.0" encoding="utf-8"?>
<robot name="smartcar">
    <link name="base_link">
        <visual>
            <geometry>
                <cylinder length="0.02" radius="0.25" />
            </geometry>
            <origin rpy="0 0 0" xyz="0 0 0" />
            <material name="blue">
                <color rgba="0 .5 .8 1" />
            </material>
        </visual>
    </link>
    <link name ="front_wheel">
        <visual>
            <!-- <origin xyz="0.2 0 -0.03" rpy="0 0 0"/> -->
            <geometry>
                <sphere radius="0.025"/>
            </geometry>
            <material name="Cyan">
                <color rgba="0 1 1 1"/>
            </material>
        </visual>
    </link>
    <joint name="front_wheel_joint" type="continuous">
        <axis xyz="0 0 1" />
        <parent link="base_link" />
        <child link="front_wheel" />
        <!-- <origin rpy="0 0 0" xyz="0 0 0" /> -->
        <origin rpy="0 0 0" xyz="0.2 0 0" />
        <limit effort="100" velocity="100" />
        <joint_properties damping="0.0" friction="0.0" />
    </joint>
```

→ 底盘刚体，这里定义为圆柱体，其他各个模块都以base_link为中心分布。每个模块的内容主要包括了模块的形状、大小、位置、颜色等信息

→ 前轮刚体，这里定义为一个球体

→ 前轮与底盘关节，连续转动

北邮移动机器人与智能技术实验室 编

2 ROS可视化工具RViz

2.1 Rviz概述

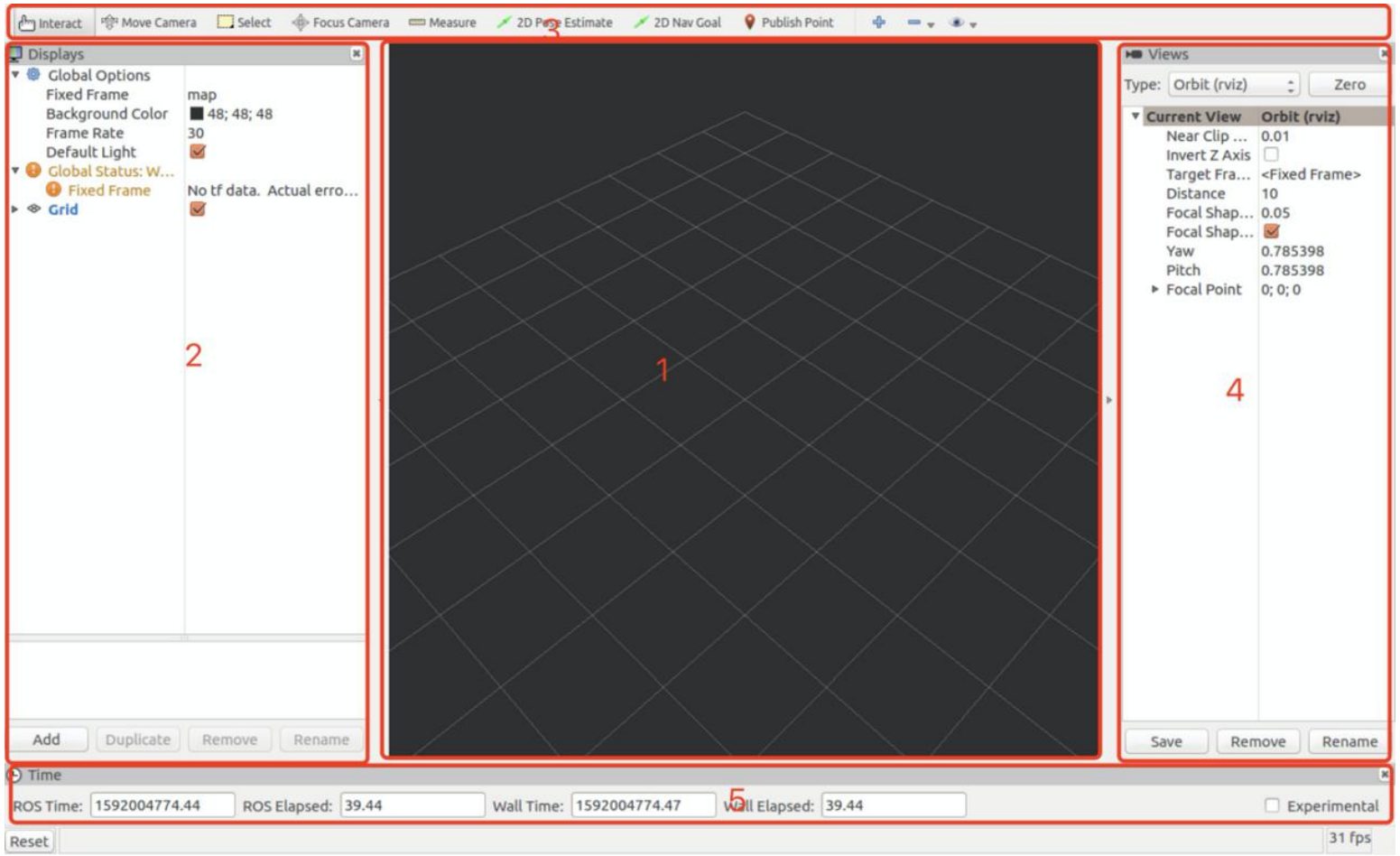
文献阅读：rviz - ROS Wiki

三维可视化工具
可以显示：
机器人结构
传感器数据
运动轨迹
环境地图

可以交互：
数值方式
按钮方式
滑动条方式

ROS下的运行：

```
roscore
rviz
```



1、3D视图区，用于可视化显示数据，目前没有任何数据，所以显示黑色。
2、显示项列表，可以配置每个插件属性，控制试图区显示内容。
3、工具栏，提供了视角控制、目标设置、发布地点等快捷工具。
4、视角设置区，用于选择多种观测视角。
5、时间显示区，用于显示当前的系统时间和ROS时间。

2.2 Rviz下显示littecar

2.2.1 创建launch文件

在littlecar_description文件夹下新建文件夹名为“launch”，并创建launch启动文件littlecar.urdf.rviz.launch

```
<launch>
    <arg name="littlecar"/>
    <arg name="gui" default="False"/>
    <!-- 加载机器人模型 -->
    <param name="robot_description" textfile="$(find littlecar_description)/urdf/littlecar.urdf"/>
    <param name="use_gui" value="$(arg gui)"/>
    <!--启动arbotix模拟器-->
    <node name="arbotix" pkg="arbotix_python" type="arbotix_driver" output = "screen">
        <rosparam file="$(find littlecar_description)/config/littlecar_arbotix.yaml" command="load" />
        <param name="sim" value="true"/>
    </node>
    <!--运行joint_state_publisher节点，发布机器人关节状态-->
    <node name="joint_state_publisher" pkg="joint_state_publisher" type = "joint_state_publisher" >
    </node>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"/>

    <!-- 在rviz中加载机器人模型-->
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find littlecar_description)/rviz/urdf.rviz" />
</launch>
```

launch文件的作用

从 ROS 角度来看，机器人系统就是一堆 node 和 topic （再添加一些 parameter, service 等）构成的网络（rosgraph），其中每个 node 都可以完成一定的功能。通常一个机器人运行时要开启很多 node，如果一个 node 一个 node 的启动，比较麻烦。通过 launch文件以及 roslaunch 命令可以一次性启动多个 node，并且可以设置丰富的参数。

2.2 Rviz下显示littecar

2.2.2 导入arbotix插件

arbotix是一款控制电机、舵机的控制板，并提供了ROS功能包。他可以驱动真实的arbotix控制板，也可以在rviz中仿真，arbotix节点接受 /cmd_vel topic ,并驱动小车运动

sudo apt-get install ros-noetic-arbotix

在littlecar_description下新建文件夹config，并新建文件littlecar_arbotix.yaml， 内容如下，键值之间一定要有空格

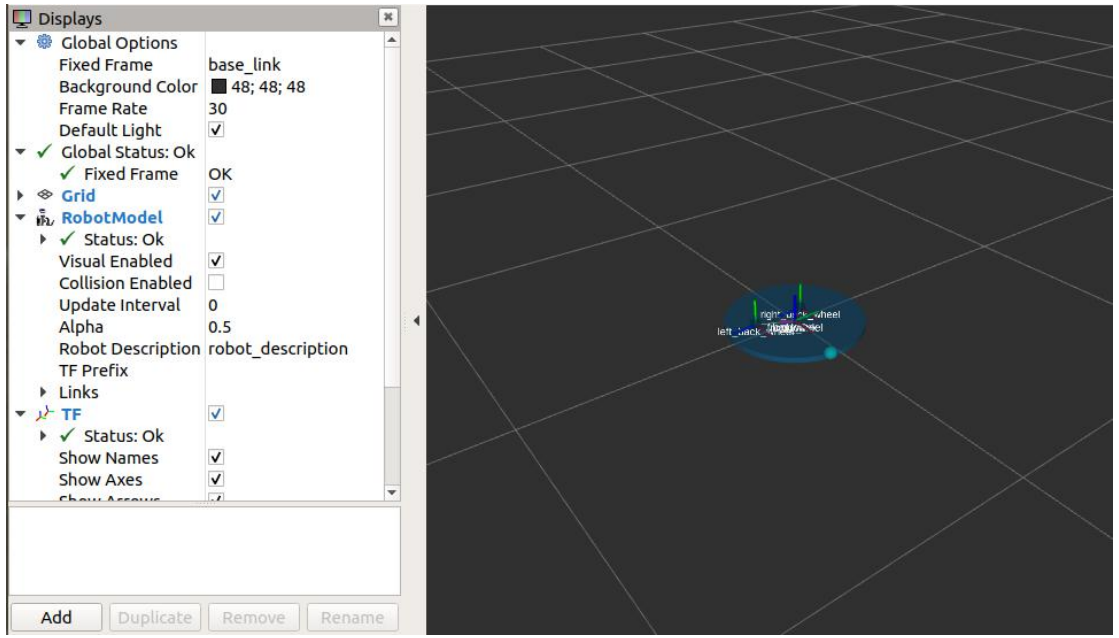
```
port: /dev/ttyUSB0
baud: 115200
rate: 20
sync_write: True
sync_read: True
read_rate: 20
write_rate: 20
controllers: {
  base_controller: {type: diff_controller, base_frame_id: base_link, base_width: 0.26, ticks_meter: 4100, Kp: 12, Kd: 12, Ki: 0, Ko: 50, accel_limit: 1.0}
}
```

2.2 Rviz下显示littecar

2.2.3 显示littlecar

从ROS给出的示例urdf_tutorial中拷贝urdf.rviz到littlecar工作目录下:

```
sudo apt-get install ros-noetic-urdf-tutorial
cd littlecar_description
mkdir -p rviz
cp /opt/ros/noetic/share/urdf_tutorial/rviz/urdf.rviz rviz
```



运行:

```
source devel/setup.bash
roslaunch littlecar_description littlecar.urdf.rviz.launch
```

发送运动指令:

```
rostopic pub -r 10 /cmd_vel geometry_msgs/Twist '{linear: {x: 0.5, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0.5}}'
```

2.3 用键盘控制小车移动

RViz的机器人仿真主要用“Twist{}结构”控制机器人的移动。用户可以通过发布Topic话题的方式，将速度指令发布到littlecar的速度话题中，进而控制机器人。

1、在命令行中进入littlecar的工作空间，并输入命令：

```
cd ~/littlecar/src/littlecar_description
mkdir scripts
```

2、在scripts下创建脚本python文件teleop.py

3、注意，在创建完teleopy.py文件后需要对其赋予权限，脚本才能运行：

```
chmod +x teleopy.py
```

4、用rosrun来执行python文件。在程序中，用键控制小车的移动。

```
rosrun littlecar_description teleopy.py
```

第5课 机器人仿真环境

北邮移动机器人与智能技术实验室 编

2 ROS仿真工具Gazebo

3.1 Gazebo概述

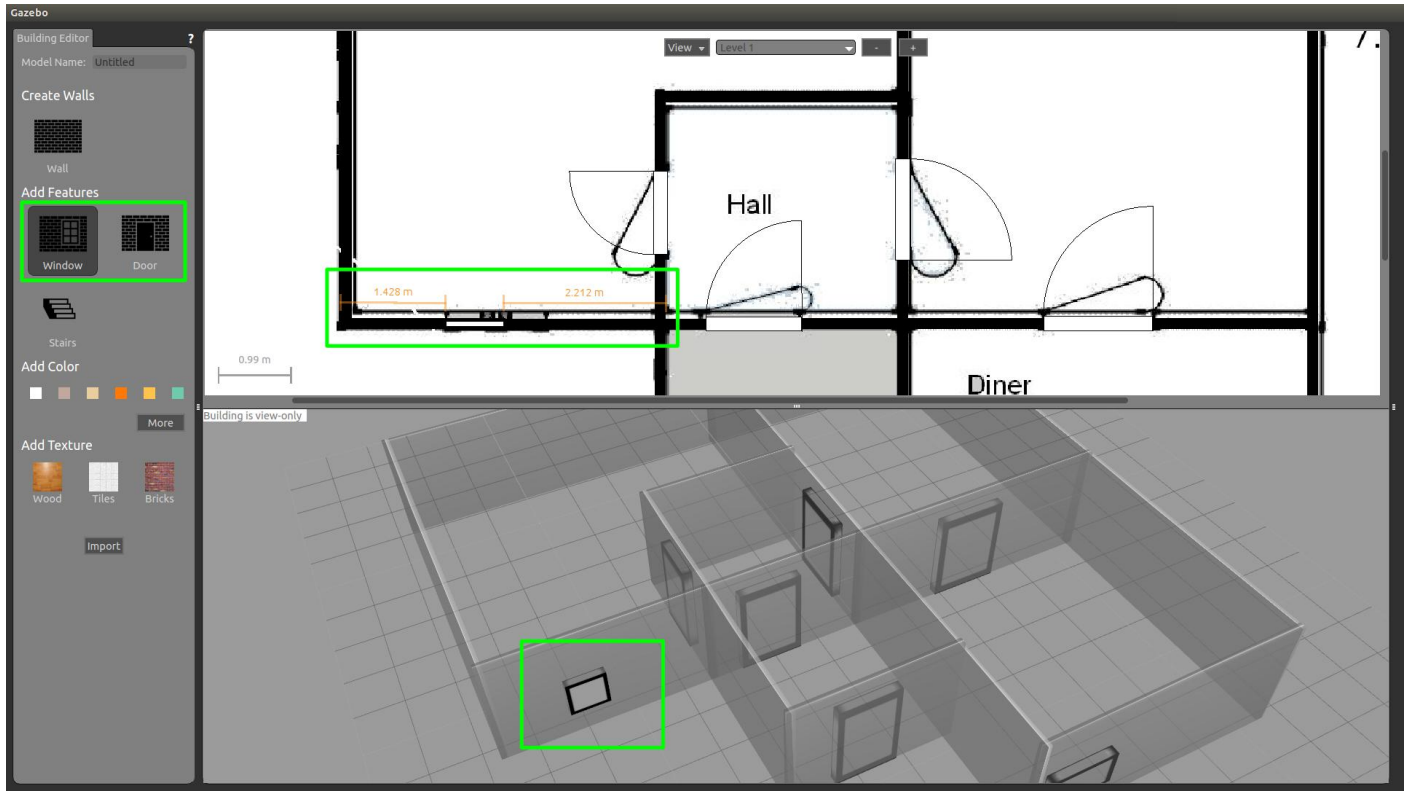
三维物理仿真平台

- 具备强大的物理引擎；
- 高质量的图形渲染；
- 方便的编程与图形接口；
- 开源免费

典型应用场景

- 测试机器人算法；
- 机器人的设计；
- 现实情况的回溯实验

官网：Gazebo (gazebosim.org)



学习资料：使用gazebo中的building editor创建一个建筑环境并用于仿真

3.2 在Gazebo上进行仿真

3.2.1 创建littlecar.xacro

Xacro宏语言提供了Constants(常量)，Macro(宏)和Include(引用)等高级编码方式来描述机器人，从而实现代码的复用。Xacro的详细介绍可以参考http://wiki.ros.org/xacro。

```
<xacro:property name="width" value="0.2" />
<xacro:property name="bodylen" value="0.6" />
<link name="base_link">
    <visual>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <geometry>
            <cylinder radius="${width}" length="${bodylen}"/>
        </geometry>
    </collision>
</link>
```

Constants(常量)使用

```
<xacro:macro name="leg" params="prefix reflect">
    <link name="${prefix}_leg">
        <visual>
            <geometry>
                <box size="${leglen} 0.1 0.2"/>
            </geometry>
            <origin xyz="0 0 -${leglen/2}" rpy="0 ${pi/2} 0"/>
            <material name="white"/>
        </visual>
        <collision>
            <geometry>
                <box size="${leglen} 0.1 0.2"/>
            </geometry>
            <origin xyz="0 0 -${leglen/2}" rpy="0 ${pi/2} 0"/>
        </collision>
        <xacro:default_inertial mass="10"/>
    </link>

    <joint name="base_to_${prefix}_leg" type="fixed">
        <parent link="base_link"/>
        <child link="${prefix}_leg"/>
        <origin xyz="0 ${reflect*(width+.02)} 0.25" />
    </joint>
    <!-- A bunch of stuff cut -->
</xacro:macro>
<xacro:leg prefix="right" reflect="1" />
<xacro:leg prefix="left" reflect="-1" />
```

Macro(宏)使用

3.2 在Gazebo上进行仿真

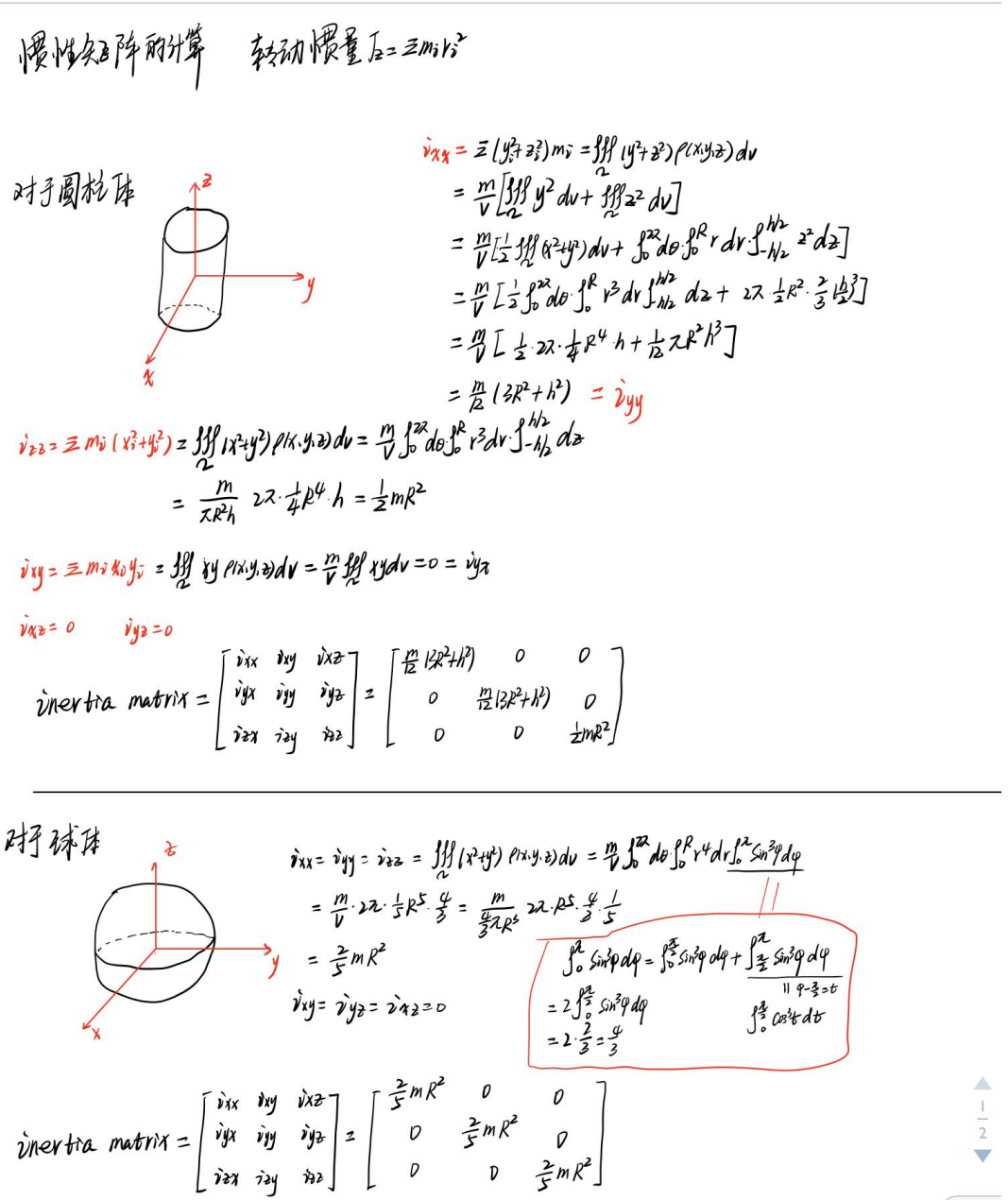
3.2.1 创建littlecar.xacro

小车代码位于：ch2/littlecar/src/littlecar_description/urdf/littlecar.xacro

```
<link name="base_link">
  <visual>
    <geometry>
      <cylinder length="0.02" radius="0.25" />
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0" />
    <material name="blue">
      <color rgba="0 .5 .8 1" />
    </material>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.02" radius="0.25" />
    </geometry>
 </collision>
 <inertial>
  <mass value="1.0" />
  <inertia ixx="0.0157" iyy="0.0157" izz="0.003125" ixy="0" ixz="0"
iyz="0" />
  </inertial>
</link>
…
<gazebo reference="base_link">
<material>
Gazebo/Blue
</material>
</gazebo>
```

增加了防碰撞和物理特性
Mass为质量单位千克
其他形状单位为米

仿真引擎材质描述

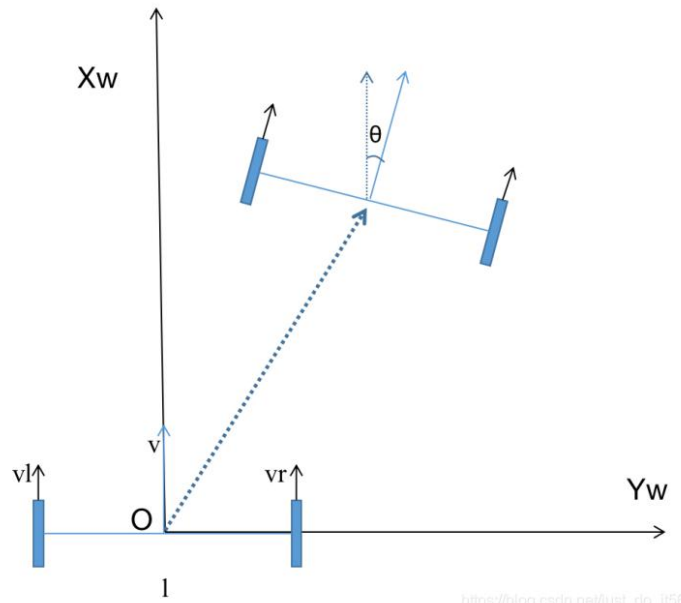


3.2 在Gazebo上进行仿真

3.2.2 导入物理模型

```
<gazebo>
  <plugin name=“differential_drive_controller”
filename="libgazebo_ros_diff_drive.so">
    <leftJoint>left_back_wheel_joint</leftJoint>
    <rightJoint>right_back_wheel_joint</rightJoint>
    <robotBaseFrame>base_link</robotBaseFrame>
    <wheelSeparation>0.45</wheelSeparation>
    <wheelDiameter>0.05</wheelDiameter>
   <legacyMode>true</legacyMode>
   <publishWheelJointState>true</publishWheelJointState>
  </plugin>
</gazebo>
```

预设的两轮差分模型



转向角速度 $$W = \frac{v_l}{r - \frac{l}{2}} = \frac{v_r}{r - \frac{l}{2}} \Rightarrow W = \frac{v_r - v_l}{l}$$

移动线速度 $$V = \frac{v_l + v_r}{2}$$

位置更新

$$Xw = Xw + d * \cos(\theta) = Xw + V * t * \cos(\theta)$$

$$Yw = Yw + d * \sin(\theta) = Yw + V * t * \sin(\theta)$$

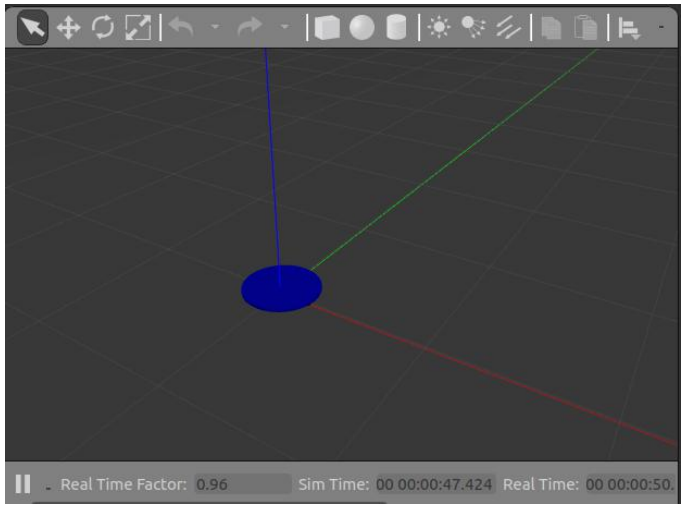
$$\theta = \theta + \alpha = \theta + W * t$$

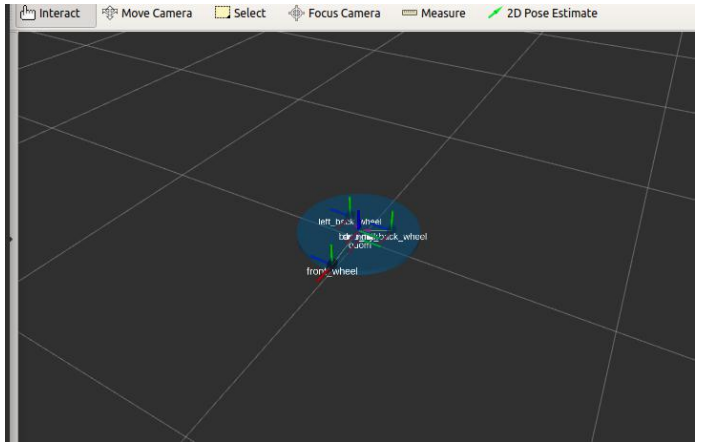
3.2 在Gazebo上进行仿真

3.2.3 编写launch文件

```
<launch>
    <!--运行gazebo仿真环境-->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
            <arg name="debug" value="false" />
            <arg name="gui" value="true" />
            <arg name="paused" value="false"/>
            <arg name="use_sim_time" value="true"/>
            <arg name="headless" value="false"/>
        </include>
    <!-- 加载机器人模型描述参数 -->
    <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find littlecar_description)/urdf/littlecar.xacro'"/>
    <!--启动arbotix模拟器-->
    <node name="arbotix" pkg="arbotix_python" type="arbotix_driver" output = "screen">
        <rosparam file="$(find littlecar_description)/config/littlecar_arbotix.yaml" command="load" />
        <param name="sim" value="true"/>
    </node>
    <!--运行joint_state_publisher节点，发布机器人关节状态-->
    <node name = "robot_state_publisher" pkg = "robot_state_publisher" type = "robot_state_publisher">
        <param name="publish_frequency" type="double" value="20.0" />
    </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 littlecar -param robot_description"/>
    <!-- 在rviz中加载机器人模型-->
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find littlecar_description)/rviz/urdf.rviz" />

    <node pkg="tf" type="static_transform_publisher" name="base_to_front" args="0.2 0 -0.03 0 0 1.57075 base_link front_wheel 100"/>
    <node pkg="tf" type="static_transform_publisher" name="base_to_dummy" args="0 0 0 0 0 0 base_link dummy 100"/>
</launch>
```






```
roslaunch littlecar_description littlecar.gazebo.launch
```

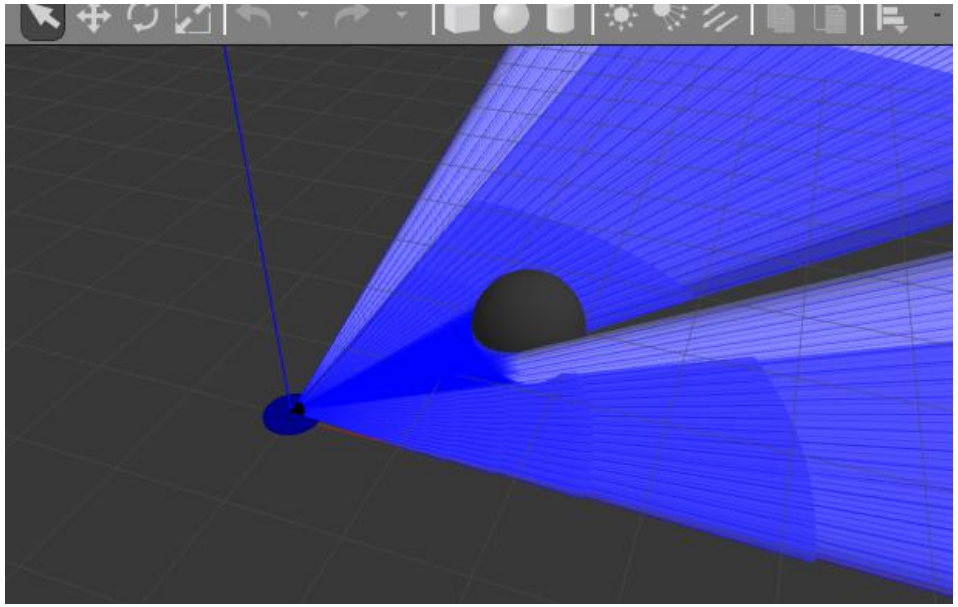
3.2 在Gazebo上进行仿真

3.2.4 激光雷达的加入

1、安装激光雷达功能包，这里用的是velodyne激光雷达

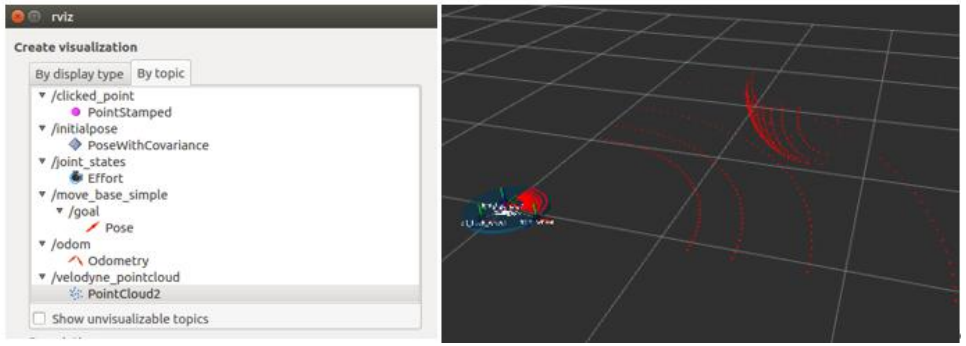
```
sudo apt-get install ros-noetic-velodyne
sudo apt-get install ros-noetic-velodyne-gazebo-plugins
```

2.修改ch2/littlecar/src/littlecar_description/urdf/littlecar.xacro



3、重新运行littlecar.gazebo.launch文件，我们在Gazebo中选择一个球体模型拖入地图，并放置在雷达可照射范围内。

4、在Rviz左侧点击add按钮，并选择By Topic选项卡，并选择/velodyne_pointcloud选项，这样可以在Rviz中实时查看点云。



3.3 进一步阅读

urdf/Tutorials - ROS Wiki

2. Learning URDF Step by Step

1. Building a Visual Robot Model with URDF from Scratch
   Learn how to build a visual model of a robot that you can view in Rviz
2. Building a Movable Robot Model with URDF
   Learn how to define movable joints in URDF
3. Adding Physical and Collision Properties to a URDF Model
   Learn how to add collision and inertial properties to links, and how to add joint dynamics to joints.
4. Using Xacro to Clean Up a URDF File
   Learn some tricks to reduce the amount of code in a URDF file using Xacro
5. Using a URDF in Gazebo
   Preliminary tutorial on how to spawn and control your robot in Gazebo.

3. Explaining a complete URDF file

1. Understanding the PR2 Robot Description
   This tutorial explains the layout of the top level URDF Xacro file for a complex robot such as PR2.

4. Learning URDF (including C++ API)

1. Create your own urdf file
   In this tutorial you start creating your own urdf robot description file.
2. Parse a urdf file
   This tutorial teaches you how to use the urdf parser
3. Using the robot state publisher on your own robot

URDF Tutorials — ROS 2 Documentation

URDF Tutorials

URDF (Unified Robot Description Format) is a file format for specifying the geometry and organization of robots in ROS.

- Building a Visual Robot Model with URDF from Scratch
- Building a Movable Robot Model with URDF
- Adding Physical and Collision Properties to a URDF Model
- Using Xacro to Clean Up a URDF File
- Using URDF with robot_state_publisher

第5课 机器人仿真环境

北邮移动机器人与智能技术实验室 编

4 Launch文件

4.1 Launch文件概述

启动文件（Launch File）是ROS中一种同时启动多个节点的途径，还可以自动启动ROSMaster节点管理器，而且可以实现每个节点的各种配置，为多个节点的操作提供了很大便利。

```
<launch>
   <node pkg=“turtlesim” name=“sim1” type=“turtlesim_node” />
   <node pkg="turtlesim"name="sim2" type="turtlesim_node"/>
</launch>
```

Launch文件采用XML的形式进行描述，包含一个根元素<launch>和两个节点元素<node>

1.<launch>

XML文件必须要包含一个根元素，launch文件中的根元素采用<launch>标签定义，文件中的其他内容都必须包含在这个标签之中：

```
<launch>
   ......
</launch>
```

2. <node>

启动文件的核心是启动ROS节点，采用<node>标签定义，语法如下：

```
<node pkg="package-name"type="executable-name" name="node-name" />
```

其他属性 output = "screen"：将节点的标准输出打印到终端屏幕，默认输出为日志文档； · respawn = "true"：复位属性，该节点停止时，会自动重启，默认为false； · required = "true"：必要节点，当该节点终止时，launch文件中的其他节点也被终止； · ns = "namespace"：命名空间，为节点内的相对名称添加命名空间前缀； · args = "arguments"：节点需要的输入参数

4.2 参数文件

1. <param>

parameter是ROS系统运行中的参数，存储在参数服务器中。在launch文件中通过<param>元素加载parameter；launch文件执行后，parameter就加载到ROS的参数服务器上了。每个活跃的节点都可以通过 ros::param::get()接口来获取parameter的值，用户也可以在终端中通过rosparam命令获得parameter的值。<param>的使用方法如下：

```
<param name="output_frame" value="odom"/>
```

2.<rosparam>

<rosparam>可以帮助我们将一个yaml格式文件中的参数全部加载到ROS参数服务器中，需要设置command属性为“load”，还可以选择设置命名空间“ns”。

```
rosparamfile="$(find 2dnav_pr2)/config/costmap_common_params.yaml" command="load" ns="local_costmap" />
```

3. <arg>

argument是另外一个概念，类似于launch文件内部的局部变量，仅限于launch文件使用，便于launch文件的重构，和ROS节点内部的实现没有关系。 设置argument使用<arg>标签元素，语法如下：

```
<arg name=”arg-name” default=”arg-value”/>
```

launch文件中需要使用到argument时，可以使用如下方式调用：

```
<paramname="foo" value="$(argarg-name)" /><node name="node" pkg="package" type="type "args="$(arg arg-name)" />
```

4.3 重映射

ROS的设计目标是提高代码的复用率，所以ROS社区中的很多功能包我们都可以拿来直接使用，而不需要关注功能包的内部实现。那么问题就来了，别人功能包的接口不一定和我们的系统兼容呀？

ROS提供一种重映射的机制，简单来说就是取别名，类似于C++中的别名机制，我们不需要修改别人功能包的接口，只需要将接口名称重映射一下，取个别名，我们的系统就认识了（接口的数据类型必须相同）。

launch文件中的<remap>标签可以帮我们实现这个重映射的功能。 比如turtlebot的键盘控制节点，发布的速度控制指令话题可能是/turtlebot/cmd_vel，但是我们自己的机器人订阅的速度控制话题是/cmd_vel，这个时候使用<remap>就可以轻松解决问题，将/turtlebot /cmd_vel重映射为/cmd_vel，我们的机器人就可以接收到速度控制指令了：

```
<remap from="/turtlebot/cmd_vel"to="/cmd_vel"/>
```

4.4 嵌套复用

在复杂的系统当中，launch文件往往有很多，这些launch文件之间也会存在依赖关系。如果需要直接复用一个已有launch文件中的内容，可以使用<include>标签包含其他launch文件，这和C语言中的include几乎是一样的。

```
include file="$(dirname)/other.launch" />
```

更多高级的标签元素可以访问wiki学习： http://wiki.ros.org/roslaunch/XML

移动机器人开发技术（激光SLAM版）配套教学PPT

谢 谢 观 看

北京郵電大學
Beijing University of Posts and Telecommunications

移动机器人与智能技术实验室编

宋桂岭 明安龙 2021.9
expsong@qq.com