三轮全向移动机器人运动学实验

1 准备工作

本实验教程的模型及相关代码部分借鉴 OpenBase：
(https://github.com/GuiRitter/OpenBase)，以及：
https://github.com/YugAjmera/omni3ros_pkg
如果要实现键盘遥控，也可以参考后面网站的程序。本实验教程仅从最基本的原理出发，一步一步讲解如何编程仿真控制三轮全向移动机器人。

1.1 创建 ROS 工程

在以前创建的 ROS project 下进入到 src 目录，然后执行包(package)创建命令：
catkin_create_pkg omni_robot
cd omni_robot
创建五个目录：
mkdir src urdf launch meshes config

1.2 编写 URDF 模型文件

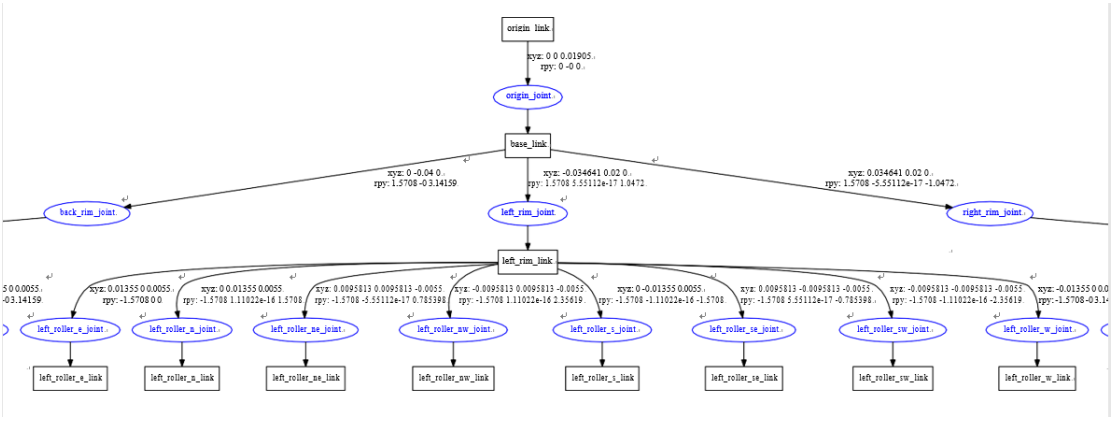
前面我们编写的两轮差分驱动机器人 URDF 文件比较简单，link 和 joint 不多，但三轮全向机器人采用了三个麦克纳姆轮，每个麦克纳姆轮有一个轮辋(rim，三个轮一般被定义成：left_rim，back_rim，right_rim)，每个轮辋的外圆上都有若干小滚轮(roller)，以 OpenBase(https://github.com/GuiRitter/OpenBase)里的开源三轮全向机器人为例，其 URDF 模型包括：

底盘：底盘被定义成一个基础的连杆：base_link 以及连接三个轮辋的连杆：rim_left_link、rim_back_link、rim_right_link。

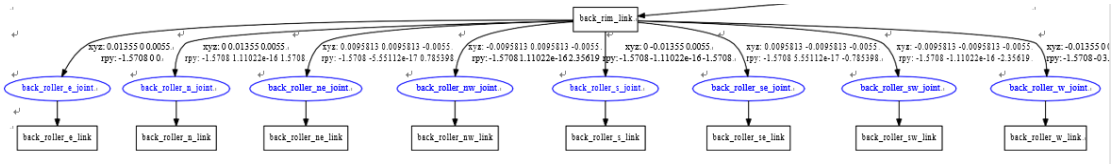
轮辋：轮辋即三个主驱动轮，被定义为(joint)：rim_left_joint、rim_back_joint、rim_right_joint。这三个关节的父连杆都是 base_link，子连杆分别是 rim_${name}_link，如果嫌麻烦，可以定义一个 name 宏，分别为"left"、"back"、"back"。这三个子连杆再连接各自小滚轮。

根据 OpenBase 的 URDF 模型，我们可以使用下面的一条命令直接生成这个机器人的模型树结构（包括两个文件 omni_robot.gv，omni_robot.pdf）：urdf_to_graphiz description.urdf

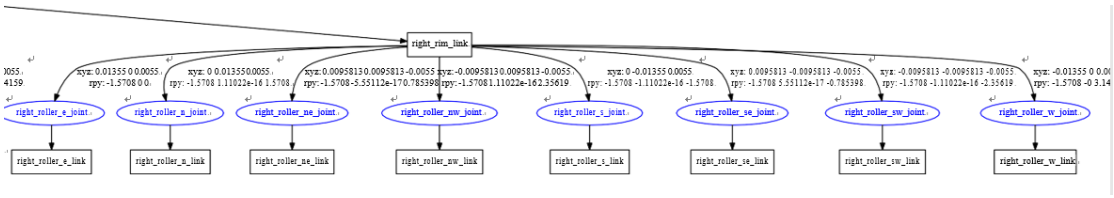
OpenBase 的模型树如图 1 所示。图 1 太大了，需要放大才能看清楚，图 1 为局部放大后的左中右的树结构细节图。



(a) 中间部分(左轮往下的连接)



(b) 左边部分(后轮往下的连接)



(c) 右边部分(右轮往下的连接)

图 1 使用 urdf_to_graphiz 命令生成的模型树结构图

这个 urdf 文件包括 CAD 文件可以从 OpenBase 下载。将 description.urdf 文件放入 urdf 目录，base.stl、rim.stl、roller.stl 文件放入 meshes 目录。

注意，对于这个机器人的 URDF 文件，我们需要对三个驱动轮设置传动标记 transmission，它用于描述致动器和关节之间的关系。

```
<transmission name="left_transmission">

    <type>transmission_interface/SimpleTransmission</type>
    <joint name="left_rim_joint">

        <hardwareInterface>EffortJointInterface</hardwareInterface>

    </joint>
    <actuator name="left_motor">

        <hardwareInterface>EffortJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>

    </actuator>

</transmission>
<transmission name="back_transmission">
```

```
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="back_rim_joint">

            <hardwareInterface>EffortJointInterface</hardwareInterface>

        </joint>
        <actuator name="back_motor">

            <hardwareInterface>EffortJointInterface</hardwareInterface>
            <mechanicalReduction>1</mechanicalReduction>

        </actuator>

    </transmission>
    <transmission name="right_transmission">

        <type>transmission_interface/SimpleTransmission</type>
        <joint name="right_rim_joint">

            <hardwareInterface>EffortJointInterface</hardwareInterface>

        </joint>
        <actuator name="rim_motor">

            <hardwareInterface>EffortJointInterface</hardwareInterface>
            <mechanicalReduction>1</mechanicalReduction>

        </actuator>

    </transmission>
```

加载 **gazebo_ros_control 插件：**

```
    <gazebo>

        <plugin name="gazebo_ros_control"
filename="libgazebo_ros_control.so">

            <robotNamespace>/omni_robot</robotNamespace>

        </plugin>

    </gazebo>
```

注意这里名字命名空间改成我们的 pkg 名：omni_robot，这里可能会由于 ROS 及 gazebo 的版本问题出现 GazeboControlPlugin missing 之类的问题，一般加上下面一句即可：

<legacyModeNS>true</legacyModeNS>

1.3 创建用于机器人控制器的 yaml 配置文件

对于轮式移动机器人，在 ROS 里控制，一般需要两种控制器：轮子关节的控制器以及底盘的控制器。这次我们只在 config 目录编写轮子关节的速度控制。

简单定义一下三个驱动轮的 yaml 文件，把它命名为 joint_vel_control.yaml:

```
omni_robot:

    joint_state_controller:
        type: joint_state_controller/JointStateController
        publish_rate: 100

    left_joint_velocity_controller:
        type: velocity_controllers/JointVelocityController
        joint: rim_left_joint
        pid: {p: 0.001, i: 0.0, d: 0.0}

    back_joint_velocity_controller:
        type: velocity_controllers/JointVelocityController
        joint: rim_back_joint
        pid: {p: 0.001, i: 0.0, d: 0.0}

    right_joint_velocity_controller:
        type: velocity_controllers/JointVelocityController
        joint: rim_right_joint
        pid: {p: 0.001, i: 0.0, d: 0.0}
```

如果以前没装过 JointVelocityController，需要安装：sudo apt install ros-kinetic-velocity-controllers。

1.4 创建 launch 文件加载模型

先测试一下这个 URDF 模型在 rviz 下的加载，简单写一个 launch 文件 display_rviz.launch:

```
<launch>

  <arg name="model" />
  <arg name="gui" default="true" />

  <!-- Load the URDF to the parameter server -->
  <param name="robot_description" command="$(find xacro)/xacro --
inorder '$(find omni_robot)/urdf/description.urdf'" />
  <!--param name="use_gui" value="$(arg gui)" /-->
```

```
  <node name="joint_state_publisher_gui"
pkg="joint_state_publisher_gui" type="joint_state_publisher_gui" />

  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="false" output="screen" />

  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
omni_robot)/rviz/description.rviz" required="true" />

</launch>
```

回到 ros 工程 workspace 的一级目录下，执行 catkin_make，如果没有报错，那么运行：

source devel/setup.bash

roslaunch omni_robot display_rviz.launch

加载模型结果如图 2 所示。

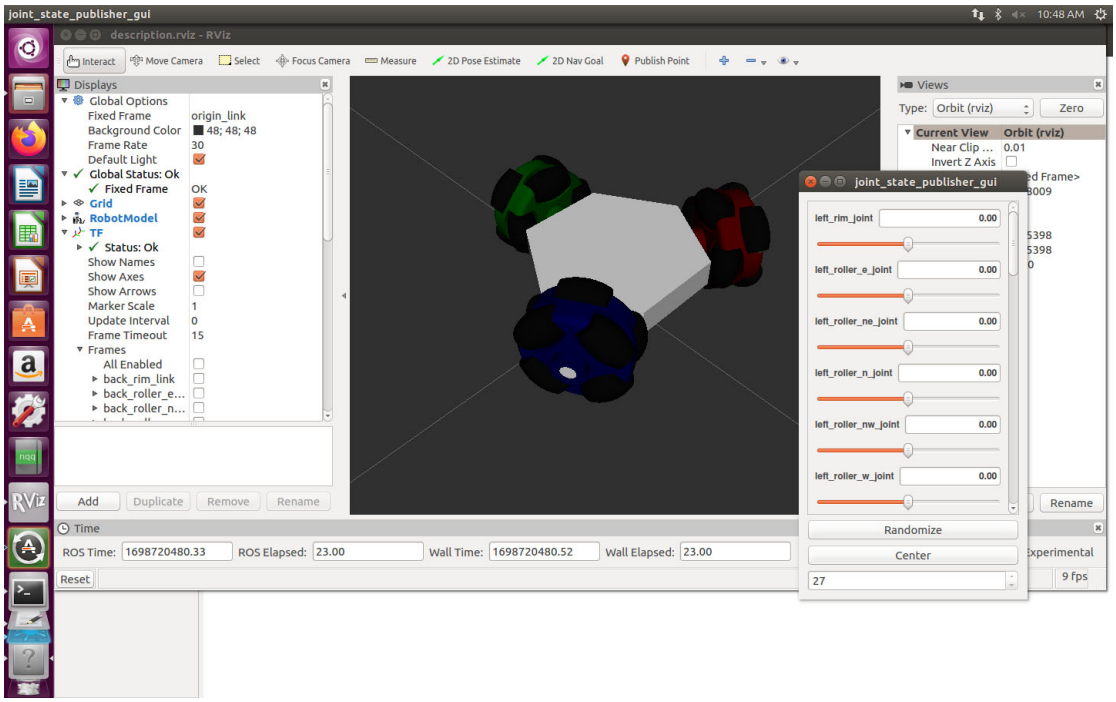


图 2 三轮全向移动机器人 URDF 模型在 RVIZ 中的加载

接下来我们来看看如何通过 **gazebo_ros_control** 插件去控制机器人移动。

在 gazobo 里加载机器人模型及参数服务器，先写个加载模型的 launch 文件 gaz.launch:

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

    <!-- Launch empty world Gazebo -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" default="worlds/empty.world"/>
        <arg name="paused" value="false"/>
        <arg name="gui" value="true"/>
    </include>
```

```
    <!-- Spawn the urdf model -->
    <param name="robot_description" command="$(find xacro)/xacro --
inorder '$(find omni_robot)/urdf/main.urdf.xacro'" />


    <arg name="x" default="0.0" />
    <arg name="y" default="0.0" />
    <arg name="z" default="0.0" />

    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
output="screen" args="-urdf -x $(arg x) -y $(arg y) -z $(arg z) -
model omni_robot -param robot_description"/>

</launch>
```

注意，由于 gazebo 的控制器用到一些参数设置，所以这个模型文件没有用 urdf 文件，而是其网站上下载的三个 xacro 文件，我把原来的 name 和 namespace 都改成了 omni_robot。

接下来，编写在 gazebo 中加载模型的参数及控制器的 launch 文件 test.launch：

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

    <include file="$(find omni_robot)/launch/gaz.launch" />

    <!-- Load controllers -->
     <rosparam file="$(find omni_robot)/config/joint_velocity.yaml"
command="load"/>

    <!-- Controllers-->
    <node name="controller_spawner" pkg="controller_manager"
type="spawner" respawn="false" output="screen" ns="/omni_robot"
    args="--namespace=/omni_robot
    joint_state_controller
    left_joint_velocity_controller
    back_joint_velocity_controller
    right_joint_velocity_controller
    "/>

</launch>
```

至此，我们编写了三轮全向移动机器人的模型以及控制器对应的参数服务器文件。整个工程文件目录树如图 1 所示。

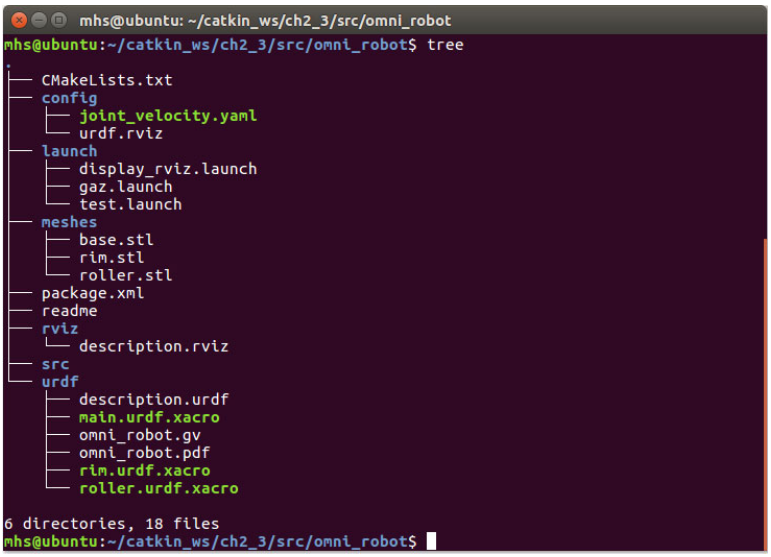


图 3 三轮全向移动机器人工程文件目录树

1.5 测试加载模型

实际目前还是没有编写控制程序，但还是假装编译一下：

回到 ros 工程 workspace 的一级目录下，比如 catkin_ws（在图 3 中，我的 workspace 一级目录是 ch2_3）。执行 catkin_make，如果没有报错，那么运行：

source devel/setup.bash

roslaunch omni_robot test.launch

加载模型结果如图 4 所示。

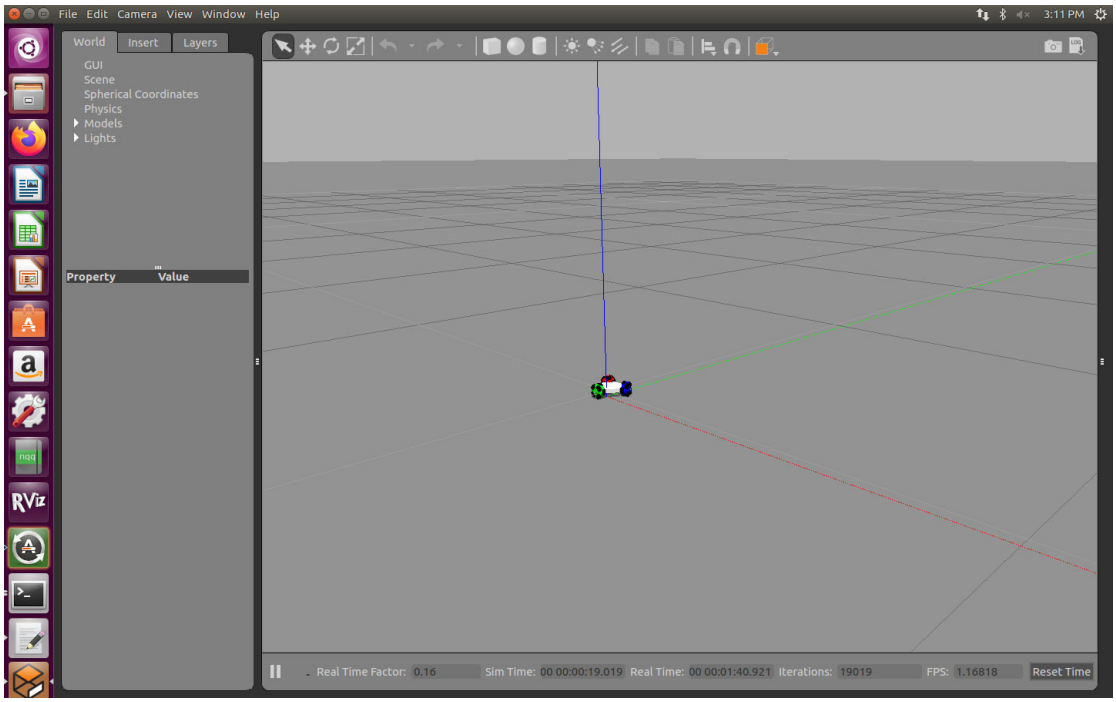


图 4 gazebo 加载界面

2 运动学测试

三轮全向移动机器人的运动学方程见教材公式(3-151)，可由机器人的底盘速度得到三个轮子的速度。由此公式可以推出移动机器人任意方向的运动控制。这里我们简单测试三种运动方向控制：向前、向左、原地旋转，停止，如表 1 所示，请大家课后查阅资料完成复杂的运动控制模式设计。

表 1 简易运动模式控制

方向 \ 速度	Rim_back_wheel	Rim_left_wheel	Rim_Right_wheel
前向	0	-10	+10
左向	+10	-10	-10
原地转圈	+10	+10	+10
停止	0	0	0

简单用 python 编一个发布三个轮子速度的程序测试一下：(在 URDF 中左轮为红色，后轮是绿色，右轮是蓝色)，测试 simple_cmd.py 代码如下：

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import Float64


if __name__=="__main__":

    rospy.init_node('Simple_Vel_Cmd')
    pub0 =
rospy.Publisher('/omni_robot/back_joint_velocity_controller/command',
Float64, queue_size=1)
    pub1 =
rospy.Publisher('/omni_robot/left_joint_velocity_controller/command',
Float64, queue_size=1)
    pub2 =
rospy.Publisher('/omni_robot/right_joint_velocity_controller/command'
, Float64, queue_size=1)

    rate=rospy.Rate(2)

    vel0 = Float64()
    vel1 = Float64()
    vel2 = Float64()

    vel0.data = 0
    vel1.data = 0
    vel2.data = 0
```

```
while not rospy.is_shutdown():
key = raw_input()
    ## forward line
if key == 'f':
   vel0 = 0
   vel1 = -10
   vel2 = 10
    ## left line
elif key == 'g':
   vel0 = 10
   vel1 = -10
   vel2 = -10
    ## rotate
elif key == 'h':
   vel0 = 10
   vel1 = 10
   vel2 = 10
    ## stop
elif key == 'j':
   vel0 = 0
   vel1 = 0
   vel2 = 0
    ## exit
else:
   vel0 = 0
   vel1 = 0
   vel2 = 0
   exit()

pub0.publish(vel0)
pub1.publish(vel1)
pub2.publish(vel2)

rate.sleep()
```

将 simple_cmd.py 存放在 src 目录下，然后给它加上可执行权限。

`chmod +x` simple_cmd.py

运行：

rosrun omni_robot simple_cmd.py

运行结果如图 5 所示。

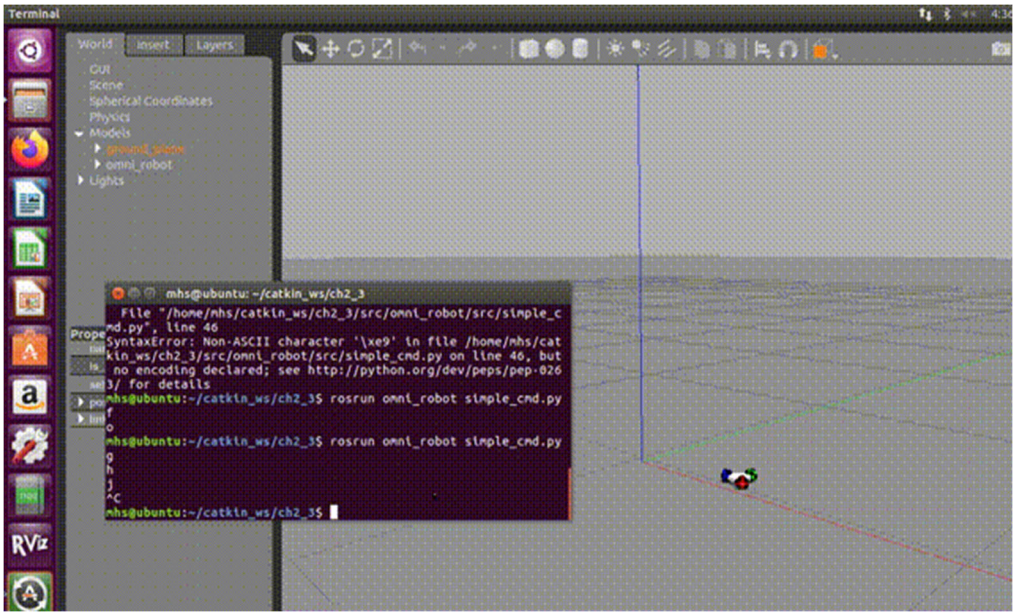


图 5 简易运动控制

本教程只演示了 step by step 的基本运动学控制原理，相关运动学的正逆解测试请同学们课后自己琢磨如何设计的更为深入一些，包括移动机器人的运动规划，在后续的章节中讲到的时候，也可以采用本章的模型去仿真测试。