四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：

fuji_mecanum: (https://github.com/DaiGuard/fuji_mecanum)

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 mecanum_robot

cd mecanum_robot

创建五个目录：

mkdir mesh urdf rviz launch src config

1.2 编写 URDF 模型文件

四轮全向机器人采用了四个麦克纳姆轮，其 URDF 模型包括：

底盘：底盘被定义成一个基础的连杆：base_link 以及连接四个轮辋的连杆：front_right_wheel_link、front_left_wheel_link、rear_right_wheel_link、rear_left_wheel_link。

urdf 文件从 fuji_mecanum 下载，将 roller.xacro，wheel.xacro,mecanum_wheel_macro.xacro，test_robot.urdf.xacro 文件放入 urdf 目录。

对于这个机器人的 URDF 文件，我们需要对四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>
    <transmission name="${name}_wheel_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="${name}_wheel_joint">
        <hardwareInterface>${interface}</hardwareInterface>
      </joint>
      <actuator name="${name}_wheel_motor">
        <hardwareInterface>${interface}</hardwareInterface>
        <mechanismReduction>1</mechanismReduction>
      </actuator>
    </transmission>
```

name 在 test_robot.urdf.xacro 文件中定义

```
  <xacro:mecanum_wheel name="front_right" side="1"
interface="hardware_interface/EffortJointInterface"/>
  <xacro:mecanum_wheel name="front_left"  side="-1"
interface="hardware_interface/EffortJointInterface"/>
  <xacro:mecanum_wheel name="rear_right"  side="-1"
```

```
interface="hardware_interface/EffortJointInterface"/>
  <xacro:mecanum_wheel name="rear_left"   side="1"
interface="hardware_interface/EffortJointInterface"/>
```

加载 gazebo_ros_control 插件：

```
  <gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
      <robotNamespace>/</robotNamespace>
      <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
      <legacyModeNS>true</legacyModeNS>
    </plugin>
  </gazebo>
```

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。

简单定义一下四个驱动轮的 yaml 文件，把它命名为 controller.yaml:

```
joint_state_controller:
  type: "joint_state_controller/JointStateController"
  publish_rate: 50

front_right_controller:
  type: effort_controllers/JointVelocityController
  joint: front_right_wheel_joint
  pid: {p: 10.0, i: 0.01, d: 0.0}

front_left_controller:
  type: effort_controllers/JointVelocityController
  joint: front_left_wheel_joint
  pid: {p: 10.0, i: 0.01, d: 0.0}

rear_right_controller:
  type: effort_controllers/JointVelocityController
  joint: rear_right_wheel_joint
  pid: {p: 10.0, i: 0.01, d: 0.0}

rear_left_controller:
  type: effort_controllers/JointVelocityController
  joint: rear_left_wheel_joint
  pid: {p: 10.0, i: 0.01, 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 view_test_robot.launch:

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

  <!-- robot spawn parameters -->
  <arg name="robot_namespace" default="/" />
  <arg name="model" default="$(find mecanum_robot)/urdf/test_robot.urdf.xacro"/>
  <arg name="gui" default="false" />
  <arg name="rvizconfig" default="$(find mecanum_robot)/rviz/view_test_robot.rviz"
/>

  <!-- load robot urdf -->
  <param name="robot_description"
    command="$(find xacro)/xacro $(arg model) --inorder" />
  <param name="use_gui" value="$(arg gui)" />

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

  <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 $(arg rvizconfig)"
    respawn="false" output="screen" />

</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 mecanum_robot view_test_robot.launch

加载模型结果如图 1 所示。

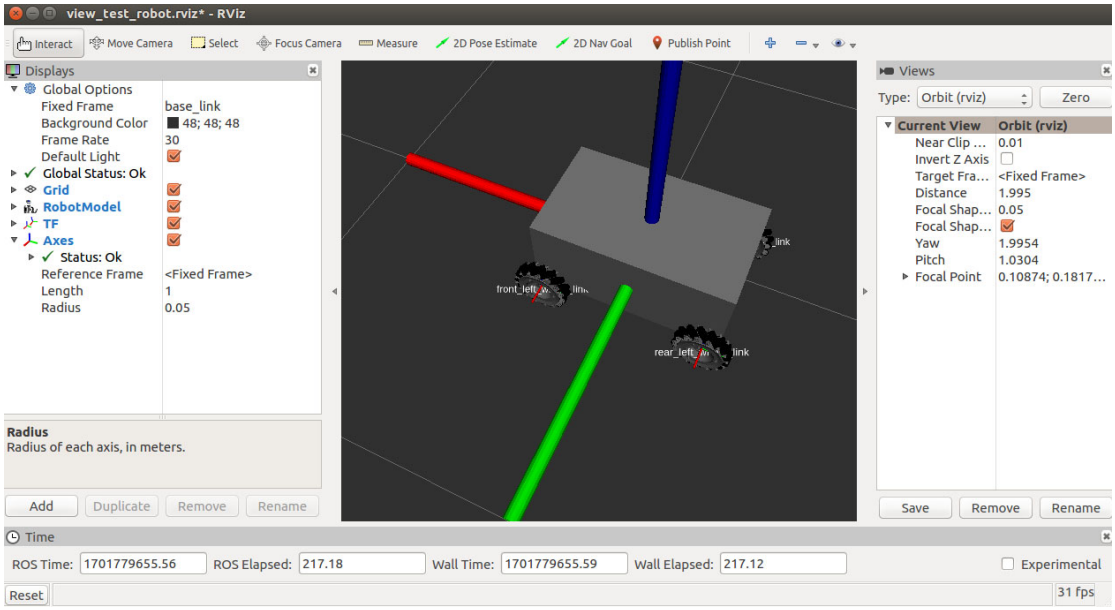

图 1 四轮全向移动机器人 URDF 模型在 RVIZ 中的加载

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控制器的 launch 文件，gazebo_test_robot.launch:

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

<launch>

  <!-- Launch empty world Gazebo -->
  <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>

  <!-- robot urdf load -->
  <param name="robot_description"
    command="$(find xacro)/xacro '$(find mecanum_robot)/urdf/test_robot.urdf.xacro'
--inorder" />

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

  <!-- Load controller configuration -->
  <rosparam command="load" file="$(find mecanum_robot)/config/controller.yaml" />
```

```
  <!-- Spawn controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner"
        args="joint_state_controller front_right_controller front_left_controller
rear_right_controller rear_left_controller" />

  <!-- Spawn robot in gazebo -->
  <node name="spawn_model" pkg="gazebo_ros" type="spawn_model"
      args="-z 1.0
            -unpause
            -urdf
            -model robot
            -param robot_description"
      respawn="false"
      output="screen" />

</launch>
```

至此，我们编写了四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 2 所示。

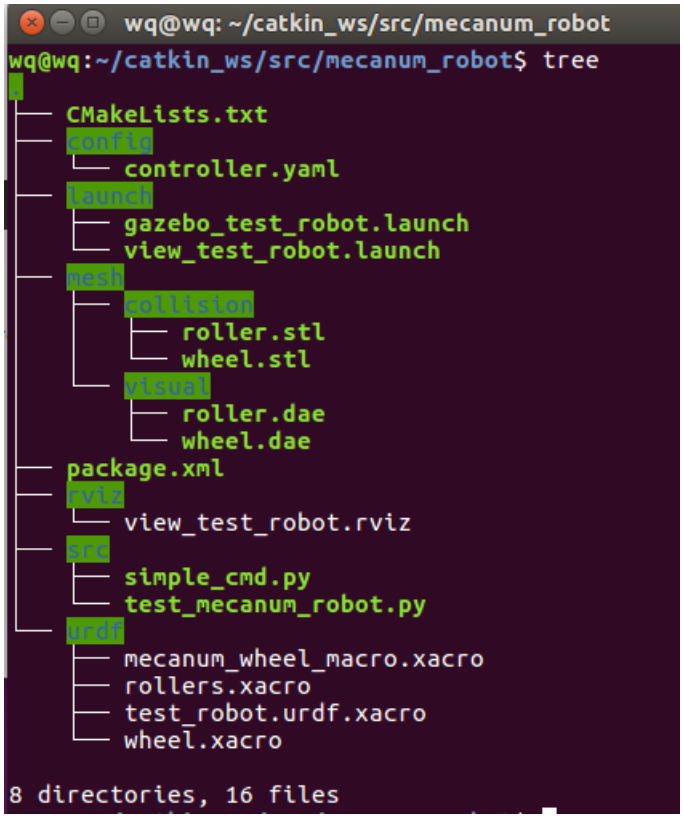

图 2 四轮全向移动机器人工程文件目录树

1.5 测试加载模型

运行： roslaunch mecanum_robot gazebo_test_robot.launch

加载模型结果如图 3 所示。

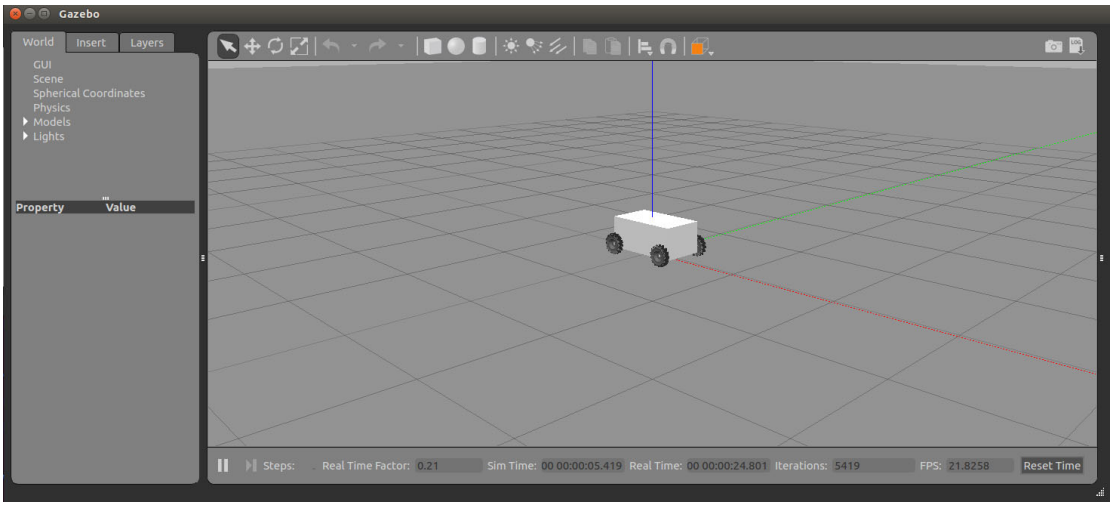

图 3　gazebo 加载界面

2 运动学测试

四轮全向移动机器人的运动学方程见教材公式(3-152)，可由机器人的底盘速度得到四个轮子的速度。由此公式可以推出移动机器人任意方向的运动控制。这里我们简单测试几种典型的运动控制模式，如表 1 所示，注意四个轮子的方向（rviz 中前轮向后关节为正，后轮向前关节为正，如图 4 所示）。

$$U=\begin{bmatrix}u_1\\u_2\\u_3\\u_4\end{bmatrix}=H(0)V_b=\frac{1}{r}\begin{bmatrix}-l-w & 1 & -1\\ l+w & 1 & 1\\ l+w & 1 & -1\\ -l-w & 1 & 1\end{bmatrix}\begin{bmatrix}\omega_{bz}\\v_{bx}\\v_{by}\end{bmatrix} \tag{3-152}$$

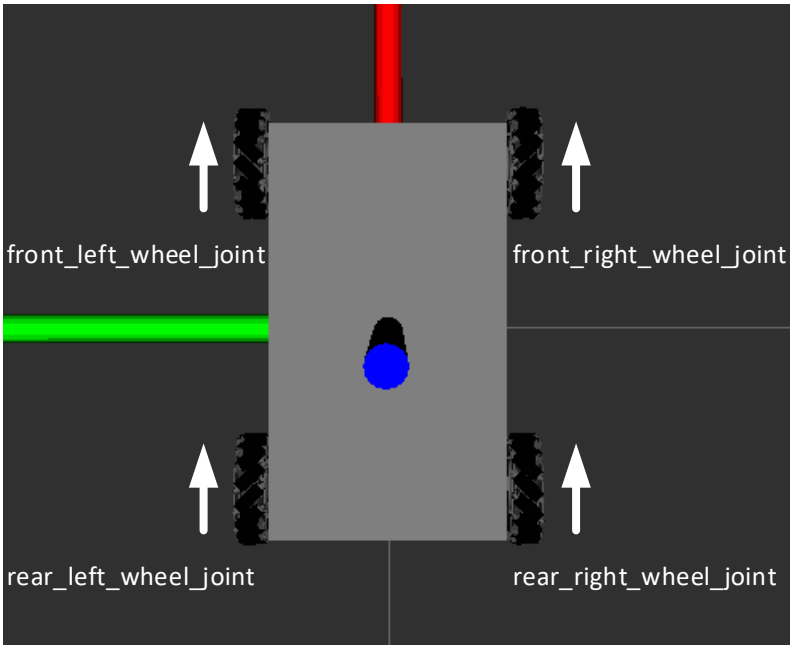

图 4 rviz 中轮子的方向

表 1 运动模式控制

方向 速度	fl_wheel_joint	fr_wheel_joint	rl_wheel_joint	rr_wheel_joint
向前	+10	+10	+10	+10
向后	-10	-10	-10	-10
向左	-10	+10	+10	-10
向右	+10	-10	-10	+10
原地向左旋转	-10	+10	-10	+10
原地向右旋转	+10	-10	+10	-10
concerning	10	0	+10	0
turn of fl_fr axis	10	-10	0	0
停止	0	0	0	0

简单用 python 编一个发布四个轮子速度的程序测试一下，测试 simple_cmd.py 代码如下：

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

import rospy
from std_msgs.msg import Float64

#wheel_vel
vel = 10

if __name__=="__main__":

    rospy.init_node('Simple_Vel_Cmd')
    fl_pub = rospy.Publisher('/front_left_controller/command', Float64,
queue_size=1)
    fr_pub = rospy.Publisher('/front_right_controller/command', Float64,
queue_size=1)
    bl_pub = rospy.Publisher('/rear_left_controller/command', Float64, queue_size=1)
    br_pub = rospy.Publisher('/rear_right_controller/command', Float64,
queue_size=1)

    rate=rospy.Rate(2)

    fl_vel = Float64()
    fr_vel = Float64()
    bl_vel = Float64()
    br_vel = Float64()


    fl_vel.data = 0
    fr_vel.data = 0
```

```
bl_vel.data = 0
br_vel.data = 0

while not rospy.is_shutdown():
    key = raw_input()
    ## forward line
    if key == 'w':
        fl_vel = vel
        fr_vel = vel
        bl_vel = vel
        br_vel = vel

    ## back line
    elif key == 's':
        fl_vel = -vel
        fr_vel = -vel
        bl_vel = -vel
        br_vel = -vel

    ## left line
    elif key == 'a':
        fl_vel = -vel
        fr_vel = vel
        bl_vel = vel
        br_vel = -vel

    ## right line
    elif key == 'd':
        fl_vel = vel
        fr_vel = -vel
        bl_vel = -vel
        br_vel = vel

    ## turn left round
    elif key == 'u':
        fl_vel = -vel
        fr_vel = vel
        bl_vel = -vel
        br_vel = vel

    ## turn right round
    elif key == 'i':
        fl_vel = vel
        fr_vel = -vel
```

```
            bl_vel = vel
            br_vel = -vel

        ## concerning
        elif key == 'j':
            fl_vel = vel
            fr_vel = 0
            bl_vel = vel
            br_vel = 0

        ## turn of fl_fr axis
        elif key == 'k':
            fl_vel = vel
            fr_vel = -vel
            bl_vel = 0
            br_vel = 0

        ## stop
        elif key == 'l':
            fl_vel = 0
            fr_vel = 0
            bl_vel = 0
            br_vel = 0

        ## exit
        else:
            fl_vel = 0
            fr_vel = 0
            bl_vel = 0
            br_vel = 0
            exit()

        fl_pub.publish(fl_vel)
        fr_pub.publish(fr_vel)
        bl_pub.publish(bl_vel)
        br_pub.publish(br_vel)

        rate.sleep()
```

简单用 python 编一个发布机器人速度的程序测试一下，测试 test_mecanum_robot.py 代码如下：

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

```
import traceback

import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64

import numpy as np
import math

width = 0.275
length = 0.575

fr_pub = None
fl_pub = None
rr_pub = None
rl_pub = None

def cmdVelCB(data):

  global fr_pub, fl_pub, rr_pub, rl_pub

  mat = np.matrix([[ 1,  1,  (width + length)],
                   [ 1, -1, -(width + length)],
                   [ 1, -1,  (width + length)],
                   [ 1,  1, -(width + length)]])

  cmd_vel = np.matrix([data.linear.x, data.linear.y, data.angular.z])

  wheel_vel = (np.dot(mat, cmd_vel.T).A1).tolist()
  print("wheel_vel=[fr,fl,rr,rl]",wheel_vel)

  wv = Float64()

  wv.data = wheel_vel[0]
  fr_pub.publish(wv)

  wv.data = wheel_vel[1]
  fl_pub.publish(wv)

  wv.data = wheel_vel[2]
  rr_pub.publish(wv)

  wv.data = wheel_vel[3]
```

```
  rl_pub.publish(wv)


def process():

  global fr_pub, fl_pub, rr_pub, rl_pub

  rospy.init_node('test_mecanum_robot', anonymous=False)

  loop_rate = rospy.Rate(10)

  fr_pub = rospy.Publisher('/front_right_controller/command', Float64, queue_size=10)
  fl_pub = rospy.Publisher('/front_left_controller/command', Float64, queue_size=10)
  rr_pub = rospy.Publisher('/rear_right_controller/command', Float64, queue_size=10)
  rl_pub = rospy.Publisher('/rear_left_controller/command', Float64, queue_size=10)

  mouse_sub = rospy.Subscriber('/cmd_vel', Twist, cmdVelCB, queue_size=10)

  while not rospy.is_shutdown():

    loop_rate.sleep()


if __name__ == '__main__':

  try:

    process()

  except Exception as ex:
    print(traceback.print_exc())
```

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

chmod +x simple_cmd.py

chmod +x test_mecanum_robot.py

运行：

rosrun mecanum_robot simple_cmd.py

运行结果如图 5 所示。

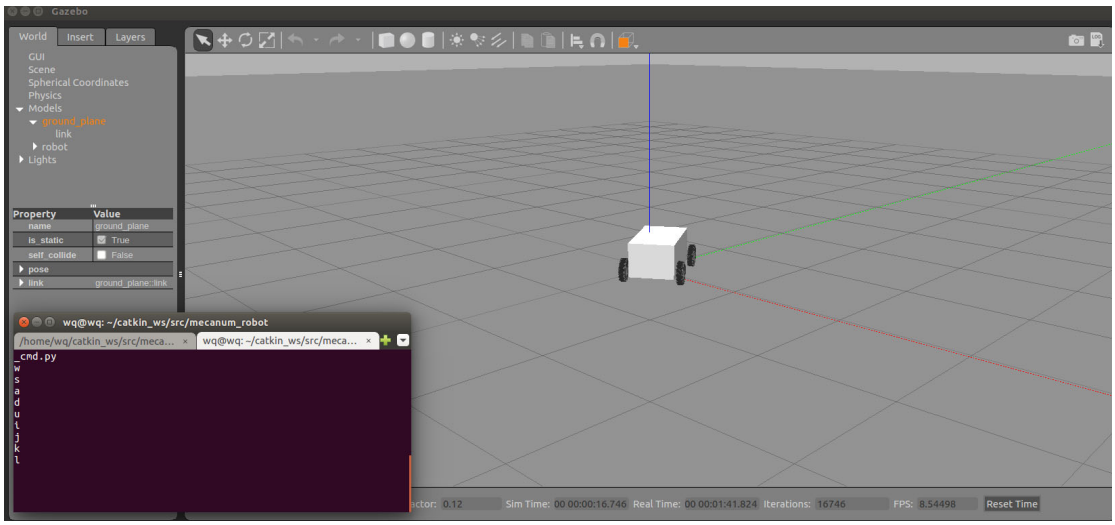

图 5 简易运动控制

对四轮机器人进行速度控制，运行：

rosrun mecanum_robot test_mecanum_robot.py

rostopic pub /cmd_vel geometry_msgs/Twist "linear:

x: 1.0

y: 0.0

z: 0.0

angular:

x: 0.0

y: 0.0

z: 0.0"

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。