两轮差分驱动机器人运动学实验

1 准备工作(参考教程: https://medium.com/allient/create-a-simulation-of-a-wheels-mobile-robot-with-ros-a086949f2827)

1.1 创建 ROS 工程

在以前创建的 ROS project 下进入到 src 目录，然后执行包创建命令:
catkin_create_pkg differential_wmr
cd differential_wmr
创建三个目录:
mkdir src urdf launch

1.2 编写 URDF 模型文件

接下里，在 urdf 文件夹中创建 dif.xacro 文件，使用 XML 标记和 robot 标记在这个文件里描述机器人的连杆和关节模型。简单示例如下:

```
<?xml version="1.0" ?>
<robot name="Name of robot"
xmlns:xacro="https://www.ros.org/wiki/xacro" >
   <!--   Links and joints of robot  -->
</robot>
```

在这个文件我们先编写轮式移动机器人底盘:

```
<link name="link_chassis">

  <pose>0 0 0.1 0 0 0</pose>

  <inertial>
    <mass value="5"/>
    <origin rpy="0 0 0" xyz="0 0 0.1"/>
    <inertia ixx="0.0395416666667" ixy="0" ixz="0"
iyy="0.106208333333" iyz="0" izz="0.106208333333"/>
  </inertial>

  <collision name="collision_chassis">
    <geometry>
```

```
      <box size="0.5 0.3 0.07"/>
    </geometry>
  </collision>

  <visual>
   <origin rpy="0 0 0" xyz="0 0 0"/>
   <geometry>
     <box size="0.5 0.3 0.07"/>
   </geometry>
  </visual>
  <!-- caster front -->
  <collision name="caster_front_collision">
    <origin rpy=" 0 0 0" xyz="0.35 0 -0.05"/>
      <geometry>
        <sphere radius="0.05"/>
      </geometry>
    <surface>
      <friction>
        <ode>
          <mu>0</mu>
          <mu2>0</mu2>
          <slip1>1.0</slip1>
          <slip2>1.0</slip2>
        </ode>
      </friction>
    </surface>
  </collision>

  <visual name="caster_front_visual">
    <origin rpy=" 0 0 0" xyz="0.2 0 -0.05"/>
    <geometry>
      <sphere radius="0.05"/>
    </geometry>
  </visual>
</link>
```

底盘这个连杆 **link_chassis** 有一个惯性标签<inertial>，它描述了物理机器人的惯性参数，一个碰撞标签<collision name="collision_chassis">，描述碰撞的属性，还有一个可视化标签<visual>，它可以让你在 rviz 中看到这个组件。此外，还给前面的小脚轮定义了它自己的碰撞<collision name="caster_front_collision">和可视化标签<visual name="caster_front_visual">。

两个驱动轮驱动这个移动机器人运动。我们给驱动轮安装在底盘的地方也定义两个连杆：`link_right_wheel` 和 `link_left_wheel`，驱动轮定义为 joint: `joint_right_wheel` 以及 `joint_left_wheel`。

对于 joint 来说：

type="continuous"表示 360 度可往复旋转的关节，而机械臂的关节一般用 revolute，表示有限位的旋转关节，其他：fixed 表示固定关节；floating 表示浮动关节；prismatic 表示有限位的滑动关节（沿某个轴线）；planar 表述平动关节（沿平面）。

右轮的连杆及右轮关节描述如下：

```
  <!-- Create wheel right -->

  <link name="link_right_wheel">
    <inertial>
      <mass value="0.2"/>
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <inertia ixx="0.00052666666" ixy="0" ixz="0" iyy="0.00052666666"
iyz="0" izz="0.001"/>
    </inertial>

    <collision name="link_right_wheel_collision">
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0" />
      <geometry>
        <cylinder length="0.04" radius="0.1"/>
      </geometry>
    </collision>

    <visual name="link_right_wheel_visual">
      <origin rpy="0 1.5707 0" xyz="0 0.02 0"/>
      <geometry>
        <!--cylinder length="0.04" radius="0.1"/-->
        <mesh filename="package://diff_drive/meshes/tyre11.dae"
scale="1.5 1 1.5"/>
      </geometry>
    </visual>

  </link>

  <!--   Joint for right wheel  -->
  <joint name="joint_right_wheel" type="continuous">
    <origin rpy="0 0 0" xyz="-0.05 0.15 0"/>
    <child link="link_right_wheel" />
    <parent link="link_chassis"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0" />
  </joint>
```

左轮和右轮类似：

```
  <!-- Left Wheel link -->
  <link name="link_left_wheel">
```

```
    <inertial>
      <mass value="0.2"/>
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <inertia ixx="0.00052666666" ixy="0" ixz="0" iyy="0.00052666666"
iyz="0" izz="0.001"/>
    </inertial>

    <collision name="link_left_wheel_collision">
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0" />
      <geometry>
        <cylinder length="0.04" radius="0.1"/>
      </geometry>
    </collision>

    <visual name="link_left_wheel_visual">
      <origin rpy="0 1.5707 0" xyz="0 -0.02 0"/>
      <geometry>
        <!--cylinder length="0.04" radius="0.1"/-->
        <mesh filename="package://diff_drive/meshes/tyre11.dae"
scale="1.5 1 1.5"/>
      </geometry>
    </visual>

  </link>

  <!--   Joint for right wheel  -->
  <joint name="joint_left_wheel" type="continuous">
    <origin rpy="0 0 0" xyz="-0.05 -0.15 0"/>
    <child link="link_left_wheel" />
    <parent link="link_chassis"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0" />
  </joint>
```

接下来，对于这个机器人的 XACRO 文件，我们对左右轮设置传动标记 transmission，它用于描述致动器和关节之间的关系。

右轮 transmission：

```
  <!--   Joint transmission for right wheel  -->
  <transmission name="right_wheel_transmission">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint_right_wheel">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwar
eInterface>
```

```
    </joint>
    <actuator name="right_wheel_actuator">
      <mechanicalReduction>1</mechanicalReduction>
      <hardwareInterface>VelocityJointInterface</hardwareInterface>
    </actuator>
  </transmission>
```

左轮 transmission：

```
  <transmission name="left_wheel_transmission">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint_left_wheel">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwar
eInterface>
    </joint>
    <actuator name="left_wheel_actuator">
      <mechanicalReduction>1</mechanicalReduction>
      <hardwareInterface>VelocityJointInterface</hardwareInterface>
    </actuator>
    </transmission>
```

最后，为了驱动机器人，我们用到 gazebo_ros 里面的 `gazebo_ros_control` 插件，所以在这个 dif.xacro 文件里面，添加了这个插件的引用：

```
<gazebo>
  <plugin name="gazebo_ros_control" filename=
"libgazebo_ros_control.so">
   <robotNamespace>/</robotNamespace>
  </plugin>
 </gazebo>
```

需要注意这里可能会由于 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 yarm 配置文件

对于轮式移动机器人，需要两种控制器：轮子关节的控制器以及底盘的控制器。

创建一个 config 目录，在 config 目录编写轮子关节的控制 yaml 文件，joint_states.yaml:

```
joint_state_controller:
    type: "joint_state_controller/JointStateController"
    publish_rate: 50
/gazebo_ros_control:
    pid_gains:
        joint_right_wheel:
            p: 2.05
            i: 1.3
            d: 0.0
```

```
        joint_left_wheel:
            p: 2.05
            i: 1.3
            d: 0.0
```

以及底盘的控制配置文件，differential_controller.yaml：

```
mobile_base_controller:
    type        : "diff_drive_controller/DiffDriveController"
    left_wheel  : 'joint_left_wheel'
    right_wheel : 'joint_right_wheel'
    publish_rate: 50.0              # default: 50
    pose_covariance_diagonal : [0.001, .001, 1000000., 1000000.,
1000000., 1000.]
    twist_covariance_diagonal: [0.001, .001, 1000000., 1000000.,
1000000., 1000.]

    # Wheel separation and diameter. These are both optional.
    # diff_drive_controller will attempt to read either one or both
from the
    # URDF if not specified as a parameter
    wheel_separation : 0.4
    wheel_radius : 0.2

    # Wheel separation and radius multipliers
    wheel_separation_multiplier: 1.0 # default: 1.0
    wheel_radius_multiplier    : 1.0 # default: 1.0

    # Velocity commands timeout [s], default 0.25
    cmd_vel_timeout: 0.25

    # Base frame_id
    base_frame_id: link_chassis #default: base_link

    # Velocity and acceleration limits
    # Whenever a min_* is unspecified, default to -max_*
    linear:
      x:
        has_velocity_limits    : true
        max_velocity           : 1.0  # m/s
        min_velocity           : -0.5 # m/s
        has_acceleration_limits: true
        max_acceleration       : 0.8  # m/s^2
        min_acceleration       : -0.4 # m/s^2
```

```
      has_jerk_limits         : true
      max_jerk              : 5.0  # m/s^3
    angular:
      z:
      has_velocity_limits     : true
      max_velocity          : 1.7  # rad/s
      min_velocity          : -1.7  # rad/s
      has_acceleration_limits: true
      max_acceleration       : 1.5  # rad/s^2
      has_jerk_limits         : true
      max_jerk              : 2.5  # rad/s^3
```

一般来说，这两种控制器是需要先在 ROS 里安装的，比如我们前面的 joint_state_controller，底盘差分驱动的控制器为 `diff_drive_controller`，如果没有可以安装：

sudo apt install ros-kinetic-diff-drive-controller

（注意：这里的 ROS 版本是 kinetic，如果其他版本请替换）

这两个控制器的配置文件为差速驱动机器人提供了参数配置。

1.4 创建 launch 文件加载模型

先创建一个 gaz.launch，用于加载参数服务器：

```
<launch>

<!-- these are the arguments you can pass this launch file, for
example paused:=true -->
<arg name="paused" default="false"/>
<arg name="use_sim_time" default="false"/>
<arg name="gui" default="false"/>
<arg name="headless" default="false"/>
<arg name="debug" default="false"/>
<arg name="model" default="$(find diferencial)/urdf/dif.xacro"/>

<!-- We resume the logic in empty_world.launch -->
<include file="$(find gazebo_ros)/launch/empty_world.launch">
<arg name="debug" value="$(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>

<!-- Load the URDF into the ROS Parameter Server -->
<param name="robot_description" command="cat '$(arg model)'" />
```

```
<!-- Run a python script to the send a service call to gazebo_ros to
spawn a URDF robot -->
<node name="dif" pkg="gazebo_ros" type="spawn_model" respawn="false"
output="screen"
  args="-z 1.0 -urdf -model Differential_Drive -param
robot_description"/>

</launch>
```

接下来，编写在 rviz 中加载模型的 launch 文件 test.launch：

```
<launch>

<arg name="model" default="$(find differential_wmr)/urdf/dif.xacro"/>
<arg name="rvizconfig" default="$(find
differential_wmr)/config/urdf.rviz"/>

<!-- Launch gazebo -->
<include file="$(find differential_wmr)/launch/gaz.launch">
  <arg name="model" value="$(arg model)"/>
</include>

<node name="rviz" pkg="rviz" type="rviz" args="-d $(arg
rvizconfig)"/>

<!-- Load joint controller configuration from YAML file to parameter
server -->
<rosparam file="$(find
differential_wmr)/config/wheel_controller.yaml" command="load"/>
<rosparam file="$(find
differential_wmr)/config/differential_controller.yaml"
command="load"/>

<!-- load the controllers -->
<node name="rdif_control_spawner" pkg="controller_manager"
type="spawner"
  respawn="true" output="screen"
   args="joint_state_controller mobile_base_controller"/>

<!-- convert joint states to TF transforms for rviz, etc -->
<node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"
    respawn="false" output="screen">
</node>

</launch>
```

至此，我们编写了带小脚轮的两轮差分驱动轮式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。

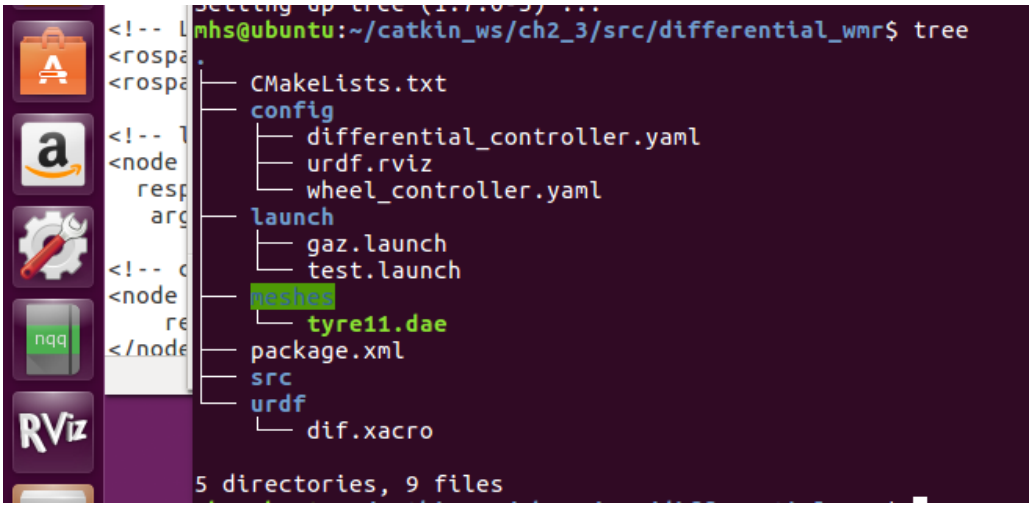

图 1 带小脚轮的两轮差分驱动轮式移动机器人工程文件目录树

1.5 测试加载模型

实际目前并没有编写控制程序，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 1 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 differential_wmr test.launch

加载模型结果如图 2 所示。

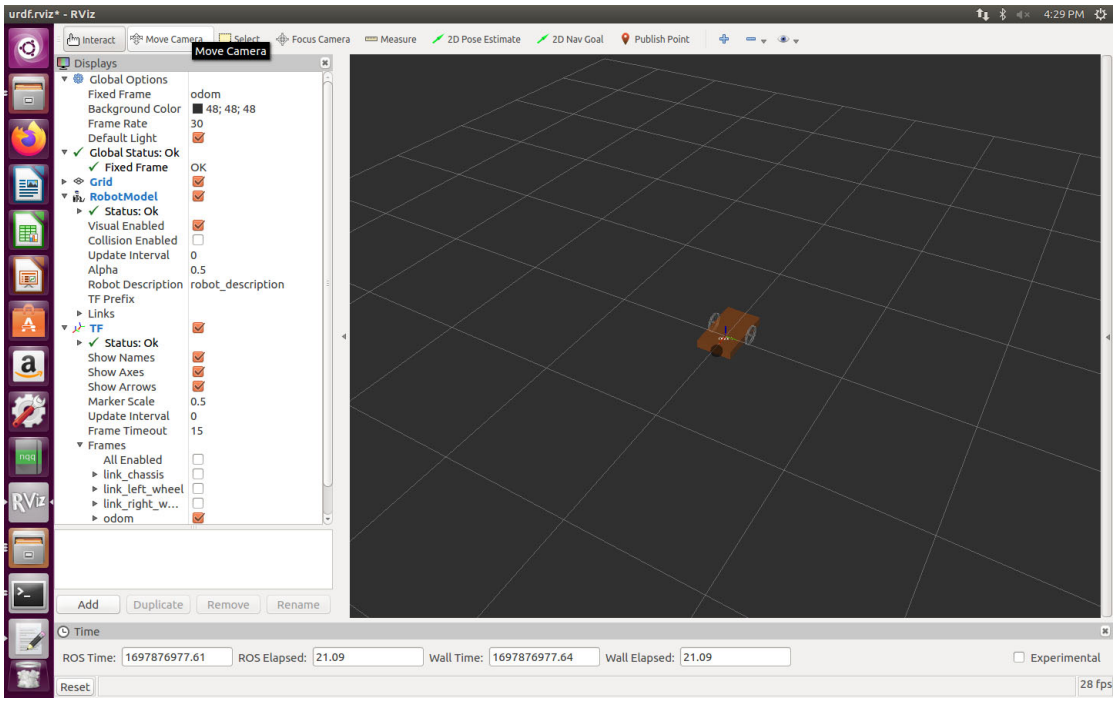

图 2 模型加载图

2 运动学测试

2.1 PID 控制器设计

模型加载成功后，我们可以看到，ROS 加载了为机器人两个控制器：

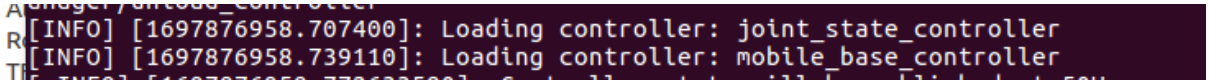

图 3 控制器

移动机器人为非完全约束机器人，其运动学见教材 3.12.5 节：

$$\dot{\xi}_I = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}cos\theta & -sin\theta & 0\\ sin\theta & cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{r\dot{\varphi}_1}{2}+\frac{r\dot{\varphi}_2}{2}\\ 0\\ \frac{r\dot{\varphi}_1}{2l}-\frac{r\dot{\varphi}_2}{2l}\end{bmatrix} \quad (3\text{-}143)$$

$$\dot{x} = \left(\frac{r\dot{\varphi}_1}{2}+\frac{r\dot{\varphi}_2}{2}\right)cos\theta$$

$$\dot{y} = \left(\frac{r\dot{\varphi}_1}{2}+\frac{r\dot{\varphi}_2}{2}\right)sin\theta$$

$$\dot{\theta} = \frac{r\dot{\varphi}_1}{2l}-\frac{r\dot{\varphi}_2}{2l}$$

这里根据教程我们实现一个 PID 控制器，完成一段轨迹跟踪控制，控制框图如图 4 所示。

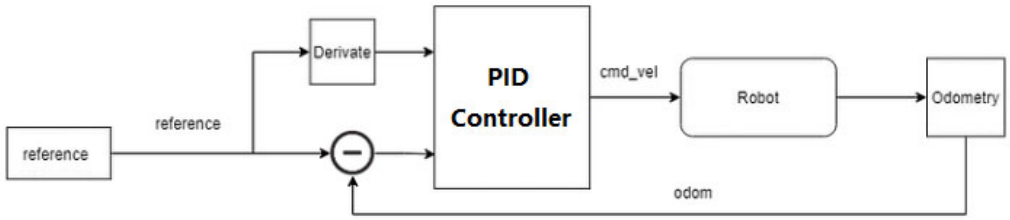

图 4 PID 控制轨迹跟踪

要在 ROS 中实现这个控制，需要订阅（subscribe）两个主题 topic：参考轨迹(reference)以及里程计(Odometry)，然后发布(publish)控制主题 topic: cmd_vel。

采用 python 编写控制代码 trajectory_control.py 如下：

```
#!/usr/bin/env python
import roslib
roslib.load_manifest('differencial')
import rospy
import math
from tf.transformations import euler_from_quaternion
from nav_msgs.msg import Odometry
import geometry_msgs.msg
```

```
import numpy as np

ESCALAR_FACTOR=70.0
p_ref_ant=np.array([[0.0],[0.0]])
p_ref=np.array([[0.0],[0.0]])
x= 0.0
y= 0.0
theta=0.0

#iNVERSE KINEMATIC PID CONTROL
def Control(angle,x_pos,y_pos,pos_before,pos_ref,K_sin):
    Jr=np.array([[math.cos(angle) ,-0.15 *
math.sin(angle)],[math.sin(angle) ,0.15 * math.cos(angle)]])
    Jr_inv=np.linalg.inv(Jr)
    K=np.array([[K_sin[0], 0],[0, K_sin[1]]])
    posp_d=pos_ref - pos_before
    pos=np.array([[x_pos],[y_pos]])
    pos_error=pos_ref - pos
    pos_error[0, 0]=math.tanh(pos_error[0, 0])
    pos_error[1, 0]=math.tanh(pos_error[1, 0])
    c=np.dot(Jr_inv, (10 * posp_d + np.dot(K,pos_error)))
    c1=c[0, 0]
    c2=c[1, 0]
    return c1,c2

#ODOMETRY DATA
def Odom(msg):
    global x
    global y
    global theta

    x=msg.pose.pose.position.x
    y=msg.pose.pose.position.y
    rot_q=msg.pose.pose.orientation
    angles=euler_from_quaternion([rot_q.x, rot_q.y, rot_q.z,
rot_q.w])
    theta=angles[2]

#CIRCLE REFERENCE
def reference(msg):
    global p_ref

    p_ref[0, 0] = msg.x
    p_ref[1, 0] = msg.y
```

```
if __name__ == '__main__':
        #NODE DEFINITION
        rospy.init_node('trajectory_control')
        sub_odom =
rospy.Subscriber('/mobile_base_controller/odom',Odometry,Odom)
        sub_reference =
rospy.Subscriber('/mobile_base_controller/reference',geometry_msgs.ms
g.Point,reference)
        diff_vel = rospy.Publisher('/mobile_base_controller/cmd_vel',
geometry_msgs.msg.Twist,queue_size=1)
        rate = rospy.Rate(10.0)
        while not rospy.is_shutdown():
            linear, angular = Control(theta, x, y, p_ref_ant, p_ref,
[0.6, 0.6])
            p_ref_ant = p_ref

            #CONTROL SIGNAL SATURED
            if linear > 1.0:
                linear = 1
            elif linear < -0.5:
                linear = -0.5
            if angular > 1.7 :
                angular = 1.7
            elif angular < -1.7:
                angular = -1.7

            print("S. Control [u w]: [%s %s]"%(linear,angular))
            #SEND VELOCITIES
            cmd = geometry_msgs.msg.Twist()
            cmd.linear.x = linear
            cmd.angular.z = angular
            diff_vel.publish(cmd)
            rate.sleep()
```

该 PID 控制器模型公式：

$$U_c = J_a^{-1}(\dot{h}_d + k\tilde{h})$$

Inverse-kinematic control

其中U_c是控制信号，J_a是机器人的雅可比，$\dot{h}_d$为微分，k是用于校准控制器的常数矩阵

2x2，$\hbar$为误差。控制框图中的微分模块采用实际轨迹与参考轨迹的差乘以频率。

主程序中首先 Subscriber 两个 topic：/mobile_base_controller/odom 以及 /mobile_base_controller/reference 。通过发布一个 topic: /mobile_base_controller/cmd_vel 给机器人控制信号。

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

```
chmod +x trajectory_control.py
```

运行：

rosrun differential_wmr trajectory_control.py

运行结果如图 5 所示。

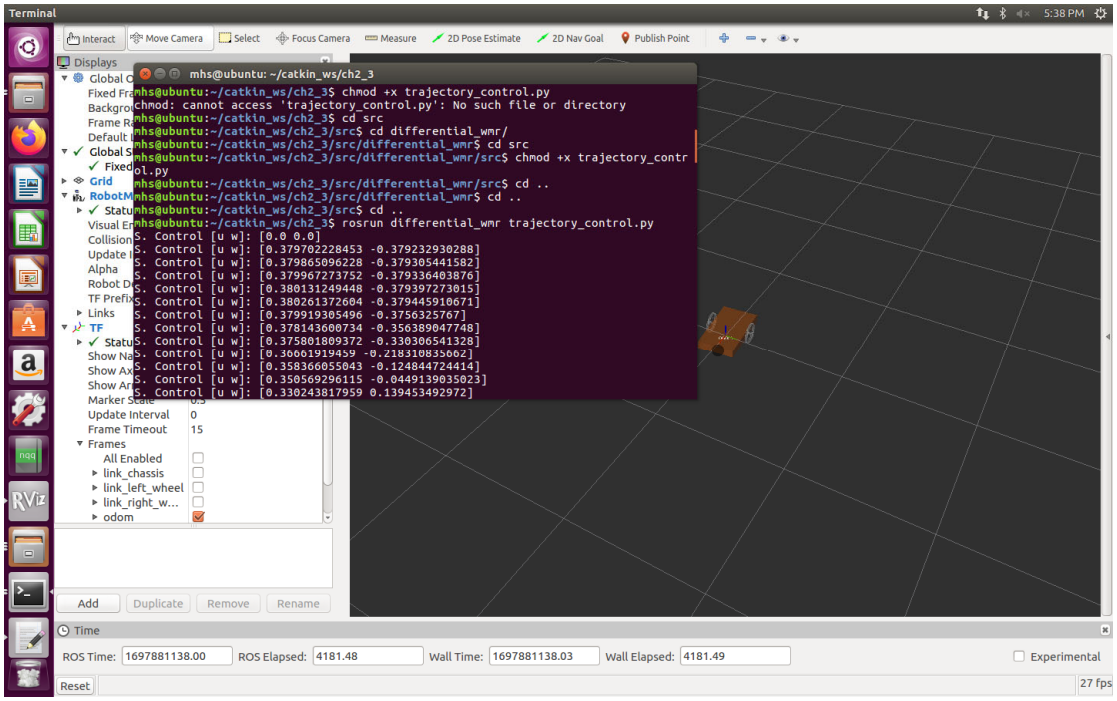

图 5 轨迹跟踪控制 python 程序

运行程序可以看到，该程序一直在订阅 odom 和 reference 主题消息，然后通过 Control 函数得到机器人底盘的线速度和角速度，发布 geometry_msgs.msg.Twist 类型的机器人底盘 cmd_vel 命令。

2.2 给出参考轨迹

在上面的控制中，需要提供 odom 和 reference 主题消息，机器人才会修改 cmd_vel 让机器人跟随参考轨迹运动。

编写第二个 python 程序，发布参考轨迹：

```
#!/usr/bin/env python
import rospy
import tf
import math
import geometry_msgs.msg

if __name__ == '__main__':
        # Init the node
```

```
        rospy.init_node('circle_reference')
         # Publisher for the reference
        ref_pub = rospy.Publisher('/mobile_base_controller/reference',
geometry_msgs.msg.Point,queue_size=1)
         # Define a transform Broadcaster
        br = tf.TransformBroadcaster()
        listener = tf.TransformListener()
         # Define the node execution frequency
        rate = rospy.Rate(10.0)
        while not rospy.is_shutdown():
            t =rospy.Time.now().to_sec() * math.pi
            x = 2.0 * math.cos(t/70)
            y = 2.0 * math.sin(t/70)
             # Create a child frame of odom for see the reference in
         RVIZ
            br.sendTransform([ x, y, 0.0],
                             [0.0, 0.0, 0.0, 1.0],
                             rospy.Time.now(),
                             "reference",
                             "odom")
             # publish the reference topic
            reference = geometry_msgs.msg.Point()
            reference.x = x
            reference.y = y
            ref_pub.publish(reference)

            rate.sleep()
```

将其存为 reference.py 存放到 src 目录，修改可执行权限：

`chmod +x` reference.py

运行：

rosrun differential_wmr rerence.py

执行该句命令，rviz 里面的 reference 坐标系(frame)将会转圈。

然后我们重新运行：

rosrun differential_wmr trajectory_control.py

那么我们就会看到，机器人刚开始会运动，经过微分控制后，运动到和 reference 的轨迹一致，机器人也开始转圆圈。

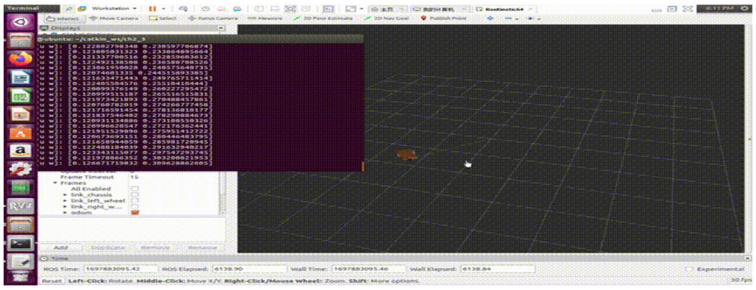

图 6 PID 控制轨迹跟随