**内部资料，注意保密**

# ROS机器人开发技术

# 中国大学MOOC

2018年7月V1.0



**中科重德智能有限公司**

目录

[ROS机器人开发技术 1](#_Toc115827007)

[中国大学MOOC 1](#_Toc1237032044)

[前言 2](#_Toc82072756)

[子任务1——创建一个tf listener 2](#_Toc869601212)

[步骤 2](#_Toc1749688641)

[子任务2——创建DEMO中的tf listener 3](#_Toc1483187787)

[步骤： 4](#_Toc1094460048)

[检验结果 5](#_Toc8700696)

[核心代码解释 5](#_Toc492720233)

[子任务3——增加一个frame 6](#_Toc1099540302)

[要点补充 6](#_Toc1733860177)

[步骤： 6](#_Toc2046334360)

[检验结果 7](#_Toc602209840)

[Fixed\_tf\_broadcaster中的核心代码解释 9](#_Toc946261869)

[作业 9](#_Toc1944679620)

前言

在之前的实训课中，我们创建了一个tf broadcaster去发布turtle的位置信息到tf，现在我们要创建一个tf listener来用tf。

这个listener的程序做了3件事情：

1. 放个 turtle2乌龟到舞台上去，用的turtlesim/Spawn服务；

2. 用tf.TransformListener类下的lookupTransform方法，查询turtle2 相对于 turtle1 的位置。

3. 用查到的相对位置来发布cmd速度和角速度信息，引导 turtle2 运动

# 子任务1——创建一个tf listener

步骤：

NO.1

在与之前的路径相同的文件夹下创建一个名为py\_tf\_listener.py的文件，例子中路径为/home/your\_PC/catkin\_ws/src/ROS-Academy-for-beginner/tf\_demo/script

NO.2

文件中输入头文件库并注册初始化节点

import rospy

import math

import tf

if \_\_name\_\_ == '\_\_main\_\_':

rospy.init\_node('py\_tf\_turtle')

NO.3

创建TransformListener的类对象listener,并设置循环比例

listener = tf.TransformListener() #TransformListener创建后就开始接受tf广播信息，最多可以缓存10s

rate = rospy.Rate(1.0)

NO.4

使用waitForTransform等待广播信息

#1. 阻塞直到frame相通

print '1. 阻塞直到frame相通'

listener.waitForTransform("/base\_link", "/link1", rospy.Time(), rospy.Duration(4.0))

NO.5

构建循环结构，不断地监听两个frame之间的变换

while not rospy.is\_shutdown():

try:

#2. 监听对应的tf,返回平移和旋转

print '2. 监听对应的tf,返回平移和旋转'

(trans,rot) = listener.lookupTransform('/base\_link', '/link1', rospy.Time(0)) #rospy.Time(0)不表示0时刻的tf，而是指最近一帧tf

except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):

Continue

NO.6

判断两个frame之间是否相通

C rospy.loginfo('距离原点的位置: x=%f ,y= %f，z=%f \n 旋转四元数: w=%f ,x= %f，y=%f z=%f ',trans[0],trans[1],trans[2],rot[0],rot[1],rot[2],rot[3])

#3. 判断两个frame是否相通

print '3. 判断两个frame是否相通'

if listener.canTransform('/link1','/base\_link',rospy.Time(0)) :

print 'true'

else :

print 'false'

rate.sleep()

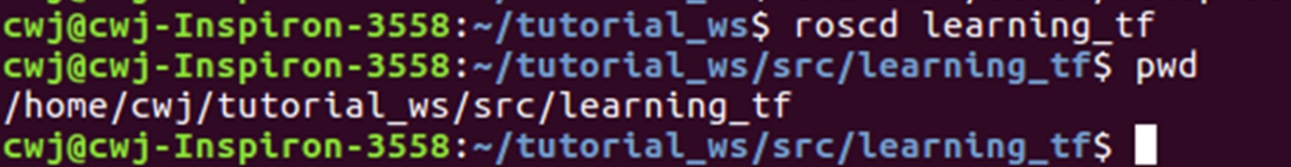
子任务2——创建DEMO中的tf listener

## **步骤**：

NO.1 进入之前创建的package路径，命令cd %YOUR\_PACKAGE\_ROUTE%

创建源文件，命令：$ roscd learning\_tf

结果显示：



可以看到执行完命令后，文件路径发生了变化。如果忘记了文件的路径可以通过PWD命令查询。

NO.2 在相同的路径下打开nodes文件夹，创建名为turtle\_tf\_listener.py的文件，并将下面代码粘贴，保存。

Codes：

#!/usr/bin/env python

import roslib

roslib.load\_manifest('learning\_tf')

import rospy

import math

import tf

import geometry\_msgs.msg

import turtlesim.srv

if \_\_name\_\_ == '\_\_main\_\_':

rospy.init\_node('turtle\_tf\_listener')

listener = tf.TransformListener()

rospy.wait\_for\_service('spawn')

spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)

spawner(4, 2, 0, 'turtle2')

turtle\_vel = rospy.Publisher('turtle2/cmd\_vel', geometry\_msgs.msg.Twist,queue\_size=1)

rate = rospy.Rate(10.0)

while not rospy.is\_shutdown():

try:

(trans,rot) = listener.lookupTransform('/turtle2', '/turtle1', rospy.Time(0))

except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):

continue

angular = 4 \* math.atan2(trans[1], trans[0])

linear = 0.5 \* math.sqrt(trans[0] \*\* 2 + trans[1] \*\* 2)

cmd = geometry\_msgs.msg.Twist()

cmd.linear.x = linear

cmd.angular.z = angular

turtle\_vel.publish(cmd)

rate.sleep()

注：同样不要遗忘执行chmod +x nodes/turtle\_tf\_listener.py命令，使文件可执行。

NO.3 创建执行文件，在上一次的launch文件夹下，找到名为start\_demo.launch的文件，用编辑器打开，并在文件中删除末端的</launch>文件，同时将如下代码添加到末端。

Codes：

<node pkg="learning\_tf" type="turtle\_tf\_listener.py"

name="listener" />

</launch>

NO.4 同样的执行launch文件

命令：$ roslaunch learning\_tf start\_demo.launch

检验结果

选中终端窗口后用方向键驱动乌龟，如果另一只乌龟跟随，则证明无误！

如果出现下面的报错信息：

图片包含 屏幕截图, 人员

已生成极高可信度的说明

是因为listener在尝试计算turtle2还未接收到的转换信息，因为这个转换需要些许的时间并开始广播坐标系。

核心代码解释

import tf

解释：加载tf工具，tf package提供的tf.TransformLIstener执行工具，可以简化接受坐标转换。

listener = tf.TransformListener()

解释：listener是创建的一个tf.TransformListener 对象，它从创建起开始接受坐标转换，并缓存这些转换最长10秒钟。

rate = rospy.Rate(10.0)

while not rospy.is\_shutdown():

try:

(trans,rot) = listener.lookupTransform('/turtle2', '/turtle1', rospy.Time(0))

except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):

continue

解释：这里是核心工作，通过lookupTransform查询listener的特定转换，这里有三个注意的参数：从哪个frame转换到哪个frame，即turtle2和turtle1的位置，还有起始时间，这里表示从最近的可获得时间开始。所有这些都包含在try-except块中以捕获可能的异常。

子任务3——增加一个frame

要点补充

1. 添加frame的必要性

在授课中已经展示过关于PR2的坐标系模型，可以看到它的坐标系之多远不止三个，而在实际的情况中，坐标系的数量也远不止3个，比如工业的6自由度机械臂，至少需要7个frame，因此这节课的内容是十分必要的。

1. 添加frame的位置

从授课中，就已经提到tf形成和维护的是tf tree，即树形的坐标系结构，所以坐标系的整体呈现是一个从父到子的关系，或者说从1到多的关系，不允许出现封闭的坐标系结构，如环形。这意味着每一个坐标系只能有一个上层的父坐标系，而这个父坐标系可以有多个子坐标系。在我们的DEMO实例中，一共有三个坐标系，其中world frame是父坐标系，而turtle1和turtle2是子坐标系。

任务说明：在现在的坐标树中添加一个carrot坐标系

图片包含 红色, 户外

已生成高可信度的说明

步骤：

NO.1 同样进入之前的package路径，创建源文件

命令：$ roscd learning\_tf

NO.2 找到路径下的nodes文件夹，创建名为fixed\_tf\_broadcaster.py的文件，将下面代码粘贴，保存。

Codes：

#!/usr/bin/env python

import roslib

roslib.load\_manifest('learning\_tf')

import rospy

import tf

if \_\_name\_\_ == '\_\_main\_\_':

rospy.init\_node('fixed\_tf\_broadcaster')

br = tf.TransformBroadcaster()

rate = rospy.Rate(10.0)

while not rospy.is\_shutdown():

br.sendTransform((0.0, 2.0, 0.0),

(0.0, 0.0, 0.0, 1.0),

rospy.Time.now(),

"carrot1",

"turtle1")

rate.sleep()

注：同样需要使这个文件可执行，命令相同，我们会发现这些代码非常相似与broadcaster中的代码，区别是转换不再是实时的。

NO.3 修改launch文件

同样打开上次的launch文件（start\_demo.launch），删掉末端的</launch>，然后添加下面一行代码：

<node pkg="learning\_tf" type="fixed\_tf\_broadcaster.py"

name="broadcaster\_fixed" />

</launch>

NO.4 运行这个文件

命令：$ roslaunch learning\_tf start\_demo.launch

注：这里需要注意，在运行前需要停止之前的运行文件，方法是使用Ctrl+C，在执行前停止以前的执行文件。

检验结果：

初始的运行结果并没有发生改变，是因为turtle2并没有跟随carrot1，所以新添的frame没有对结果产生影响，修改nodes文件下的turtle\_tf\_listener.py文件，将turtle1替换为carrot1，如下：

(trans,rot) = listener.lookupTransform("/turtle2", "/carrot1", rospy.Time(0))

再次运行launch，可以发现第二只乌龟跟胡萝卜而不是第一只乌龟！ 请记住，胡萝卜位于turtle1左侧2米处。 胡萝卜没有可视化表示，但你应该看到第二只乌龟移动到那一点。

广播一个移动的坐标系carrot1

上面我们提到的frame是一个固定的坐标系，相对于父坐标系不会随时间而改变。如果想要公布一个移动的坐标系，可以编写代码去使坐标系随时间而改变，我们尝试修改坐标系carrot1相对于turtle1随时间而改变。

在nodes下创建一个名为dynamic\_tf\_broadcaster.py的文件（注意使它可以执行，方法同前），同时将下面代码复制到文件中。

Codes：

#!/usr/bin/env python

import roslib

roslib.load\_manifest('learning\_tf')

import rospy

import tf

import math

if \_\_name\_\_ == '\_\_main\_\_':

rospy.init\_node('dynamic\_tf\_broadcaster')

br = tf.TransformBroadcaster()

rate = rospy.Rate(10.0)

while not rospy.is\_shutdown():

t = rospy.Time.now().to\_sec() \* math.pi

br.sendTransform((2.0 \* math.sin(t), 2.0 \* math.cos(t), 0.0),

(0.0, 0.0, 0.0, 1.0),

rospy.Time.now(),

"carrot1",

"turtle1")

rate.sleep()

注：相对于固定于turtle1的偏移量，我们使用了关于当前时间的sin和cos函数，来定义坐标系偏移量，这样就会随时间而改变。

修改launch文件，将launch文件指向新的文件即dynamic\_tf\_broadcaster，而不是原先的fixed\_tf\_turtle，具体的代码为下：

Codes：

<node pkg="learning\_tf" type="dynamic\_tf\_broadcaster.py"

name="broadcaster\_dynamic" />

</launch>

注：这里不再是单纯的末尾添加，而是替换修改。

Fixed\_tf\_broadcaster中的核心代码解释

br.sendTransform((0.0, 2.0, 0.0),

(0.0, 0.0, 0.0, 1.0),

rospy.Time.now(),

"carrot1",

"turtle1")

解释：在这里，我们创建一个新的转换，从父“turtle1”到新的子“carrot1”。 carrot1框架距离turtle1框架偏移2米。

作业：

1. 对新建的carrot1,创建listener,使turtle2可以追着carrot1，而不是turtle1。

提示：这个作业是下节实训的内容，可以提前尝试练习，不必一定实现，记住遇到的问题。

1. 创建一个tf\_follower.py的文件，可以实现turtle2追着turtle1，但是始终保持最小间距是10mm，也就是说turtle2会跟在10mm的turtle1身后。