**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](#_Toc519493252)

[中国大学MOOC 1](#_Toc519493253)

[前言 2](#_Toc519493254)

[子任务1——tf & time 2](#_Toc519493255)

[引入 2](#_Toc519493256)

[检验结果 4](#_Toc519493257)

[子任务2——时间迁移 4](#_Toc519493258)

[原理解释 5](#_Toc519493259)

# 前言

我们已经知道tf可以维护坐标系树，也就是tf tree间的关系。这个树会随着时间而改变而且tf可以缓存一小段时间的转换关系（默认为最长10s）。直到我们使用了lookupTransform函数/命令，从而可以得到tf tree中，最近的可被获得的转换，而不必知道什么时间这个转换被记录下来。 我们尝试寻求一种可以在特定时间点转换的方法。

任务规划

演示和演练tf&time中的两个实例，明白如何设置waitfortransform的参数，从而得到特定时间点的转换，另一方面明白日常使用的时间函数，着重对常用函数理解，对比的目的是为了加深对time(0)的理解。

如果时间允许的话，开始进行第一个机器人的tf设置，牵扯到具体的变换摄者含义和方法。

# 子任务1——tf & time

## 引入

我们还记得之前的listener中的lookuptransform()设置如下图



这里的时间函数是Time(0)，含义是得到最近的坐标变换关系，也是最常用的方法。如果我们将这段代码替换成这样，会有什么样的结果呢？

Codes：

try:

now = rospy.Time.now()

(trans,rot) = listener.lookupTransform("/turtle2", "/carrot1", now)

except (tf.LookupException, tf.ConnectivityException):

continue

运行后会出现错误，并提示：

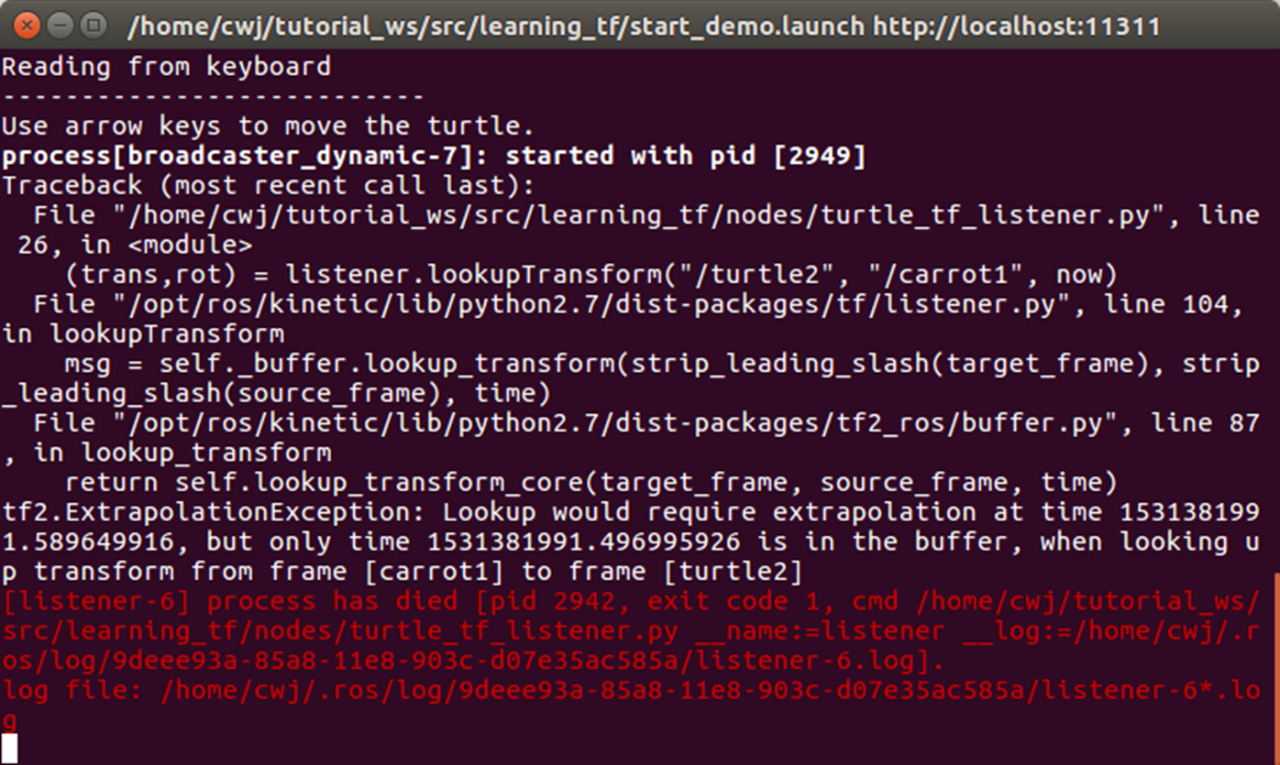


图1-1 报错提示图

信息是这样：

Traceback (most recent call last):

File "~/ros/pkgs/wg-ros-pkg-trunk/sandbox/learning\_tf/nodes/turtle\_tf\_listener.py", line 25, in <module>

(trans,rot) = listener.lookupTransform('/turtle2', '/carrot1', now)

tf.ExtrapolationException: Extrapolation Too Far in the future: target\_time is 1253830476.460, but the closest tf data is at 1253830476.435 which is 0.024 seconds away.Extrapolation Too Far in the future: target\_time is 1253830476.460, *but the closest tf data is at 1253830476.459 which is 0.001 seconds away.Extrapolation* Too Far from single value: target\_time is 1253830476.460, but the closest tf data is at 1253830476.459 which is 0.001 seconds away.

解释结果的意思：目标坐标系的时间和tf传递过来的时间不一致，tf有0.0xx或者0.001s的延时，这个延时太大了，完全不是now（实时的）

原因：每个监听器（listener）都有一个缓存区，用来储存来自全部坐标系转换的tf 广播（broadcasters）。而broadcaster发布了转换，需要一小段时间才能由listener监听并放到缓存里，就像人听到发出的指令，需要想一下该怎么做，而listener是需要把这些信息放到缓存里，这一段时间通常是0.002s，所以做不到完全的“now”，需要等上这么几微秒。

所以获取实时的坐标转换是不现实的，但是TF还有一个工具可以等待直至坐标变换可以获得，具体的代码如下：

Codes：

listener.waitForTransform("/turtle2", "/carrot1", rospy.Time(), rospy.Duration(4.0))

while not rospy.is\_shutdown():

try:

now = rospy.Time.now()

listener.waitForTransform("/turtle2", "/carrot1", now, rospy.Duration(4.0))

(trans,rot) = listener.lookupTransform("/turtle2", "/carrot1", now)

这里要强调的四个注意的地方

1. 等待的是哪个坐标系
2. 向哪个坐标系转换
3. 开始的时间
4. 等待的时间：最长不会超过这个时间值

所以waitransform()的执行是这样一个过程，阻塞，等待着变换可以获得，或者超过设定的最大时长还没有获得变换。

## 检验结果

再次运行结果，会发现第二个乌龟将会接着跟随第一个乌龟，这两种情况之间没有明显的差异，是因为时间的差值非常小只有几微秒，而举出time(0)和now()也是为了区别tf缓存和时间延时之间的区别和联系，在实际的情况中，通常更多的使用Time(0),这一点很重要。

# 子任务2——时间迁移

这次的任务不再是实时的让turtle2实时的追turtle1，而是让turtle2去追5秒前的turtle1，步骤：

NO.1 进入之间的文件路径，创建源文件

命令：$ roscd learning\_tf

NO.2 再次进入nodes文件下，编辑turtle\_tf\_listener.py文件，替换对应的代码段落如下：

Codes：

try:

now = rospy.Time.now() - rospy.Duration(5.0)

listener.waitForTransform("/turtle2", "/turtle1", now, rospy.Duration(1.0))

(trans, rot) = listener.lookupTransform("/turtle2", "/turtle1", now)

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

看到代码后，可以猜测乌龟可能的运动，在运行的前五秒，显然第二只turtle不知道去哪儿，因为不清楚第一只乌龟前五秒的历史位置，但是五秒钟后，turtle2该如何行动呢？

NO.3 运行launch文件

命令：$ catkin\_make （or make）

$ roslaunch learning\_tf start\_demo.launch

结果展示：



图2-1 过去的状态控制现在的姿态结果图

可以看到turtle2无法控制了，那么导致这个结果的原因是什么呢？

实际上，我们让tf做的事是：五秒前，turtle2相对于turtle1的位置，然后根据这个位置去控制现在的turtle2去追赶turtle1，可想而知，这个结果就变得无法预测了。

然而，我们想要达成的目标是让现在的turtle2去追赶5秒前的turtle1，需要的是现在的turtle2相对于5秒前的turtle1，那么这该如何实现呢？

示例codes如下：

try:

now = rospy.Time.now()

past = now - rospy.Duration(5.0)

listener.waitForTransformFull("/turtle2", now,

"/turtle1", past,

"/world", rospy.Duration(1.0))

(trans, rot) = listener.lookupTransformFull("/turtle2", now,

"/turtle1", past, "/world")

可以看到这里的参数多了now, past和world,命令的含义是：

给定的转换是从哪个坐标系（第一个frame位置），在哪个时间（第一个时间位置），到哪个坐标系（第二个frame位置），第二个坐标系的时间，制定的固定的坐标系（第三个frame位置）和储存结果的变量。

原理解释：

图片包含 红色, 浅色, 物体

已生成高可信度的说明

图2-2 迁移原理图

Turtle1在过去的5秒前将转换信息发送给world坐标系，到了5秒后的现在，由world frame向turtle2转换，着就实现了时间的迁移，从过去到现在，计算的结果是过去的turtle1和现在的turtle2的转换。

检验结果

再次运行代码，得到更深刻的印象。

作业：将实训2中的carrot2设置为跟随5秒前的turtle2