TF的python学习样例

参考教程：https://www.ncnynl.com/archives/201611/1076.html

1.个人对使用TF涉及概念的一些理解

TF也就是transform，按照我看到的教材，翻译为“坐标变换”。也就是说每当机器人的位置信息发生变换，就会触发一次TF，或者也可以理解为向ros系统更新机器人当前的位置信息。

值得简单一提的是，TF中包含的位置信息包括两部分，即位置(position)和方向(direction)，统称为姿态(pose)。位置(position)包含三维空间坐标信息x、y、z。方向(direction)使用四元数表达，即x,y,z,w。

了解了TF的信息组成成分，理解TF就相对简单了。TF，实际上可以类比于之前学到过的Topic，需要我们在机器人位姿发生变化时，在程序中设置更新机器人的位置，包含“广播”和“监听”两个基本使用流程。

此外，既然涉及到位置坐标，为了精确的表示，不可避免的要确定参考坐标系。因此对TF还可以有更细致的理解，即它是一个让用户岁时间跟踪机器人在多个参考系下坐标的功能包，可以帮助用户在任意时间，将点、向量等数据的坐标，在两个参考系中完成坐标变换。也是为了保证位姿信息的精确性，TF使用的4元组来表示方向信息(direction)，从而能够避免我们常用的3元组（roll、pitch、yaw）表示方向会面临的“万向节死锁(gimballock)”和速度等问题。

2.使用TF的简单例子

正如上面提到的，使用TF设计两个基本流程：“广播”和“监听”。下面例子也将遵循这两个流程。实现的程序运行成果是：demo中的小乌龟1号通过方向键盘接收控制信息，小乌龟1运动，同时发布位姿信息。于此同时，小乌龟2监听小乌龟1发布的信息，进行模仿跟随。

2.1 广播小乌龟1位姿信息

- 创建包

```
$ cd catkin_ws/src
$ catkin_create_pgk learning_tf roscpp rospy turtlesim
```

- 编译

```
$ cd ..
$ catkin_make
$ source ./devel/setup.bash
```

- 创建运行的文件

```
$ roscd learning_tf
$ mkdir nodes
$ touch nodes/turtle_tf_broadcaster.py
$ chmod +x nodes/turtle_tf_broadcaster.py
$ gedit nodes/turtle_tf_broadcaster.py
```

- 手动输入代码（去掉中文注释！！！否则不能运行）

```
#!/usr/bin/env python
import roslib
roslib.load_manifest('learning_tf')
import rospy
import tf
import turtlesim.msg

def handle_turtle_pose(msg, turtlename):
    br = tf.TransformBroadcaster()
    # 参数分别为：当前空间坐标、当前朝向、广播时间、机器人名、世界
    # 最后两个参数代表的含义为：发布的位置信息是名为“turtlename”的机器人与“世界”之间的
位置关系
    # 作为变换，从"world" 坐标系到"turtleX"坐标系进行发布。
    br.sendTransform((msg.x, msg.y, 0),
                     tf.transformations.quaternion_from_euler(0, 0,
msg.theta),
                     rospy.Time.now(),
                     turtlename,
                     "world")
if __name__ == '__main__':
    rospy.init_node('turtle_tf_broadcaster')
    turtlename = rospy.get_param('~turtle')
    # 参数含义分别为：指定接收主题为/turtleX/pose(“X”为不确定，填数字区别不同机器人
名)、
    # 消息格式文件为turtlesim.msg.Pose、消息处理（回调）函数为
handle_turtle_pose、
    # 回调函数传参为turtlename
    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

- 设置.launch文件 在命令行输入一下命令，创建start_demo.launch文件,并用编辑器打开

```
$ roscd learning_tf
$ mkdir launch
$ touch launch/start_demo.launch
$ gedit launch/start_demo.launch
```

在编辑器内输入一下代码

```
<launch>
    <!-- Turtlesim Node-->
    <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
    <node pkg="turtlesim" type="turtle_teleop_key" name="teleop"
output="screen"/>

    <node name="turtle1_tf_broadcaster" pkg="learning_tf"
type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
      <param name="turtle" type="string" value="turtle1" />
    </node>
    <node name="turtle2_tf_broadcaster" pkg="learning_tf"
type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
      <param name="turtle" type="string" value="turtle2" />
    </node>
</launch>
```

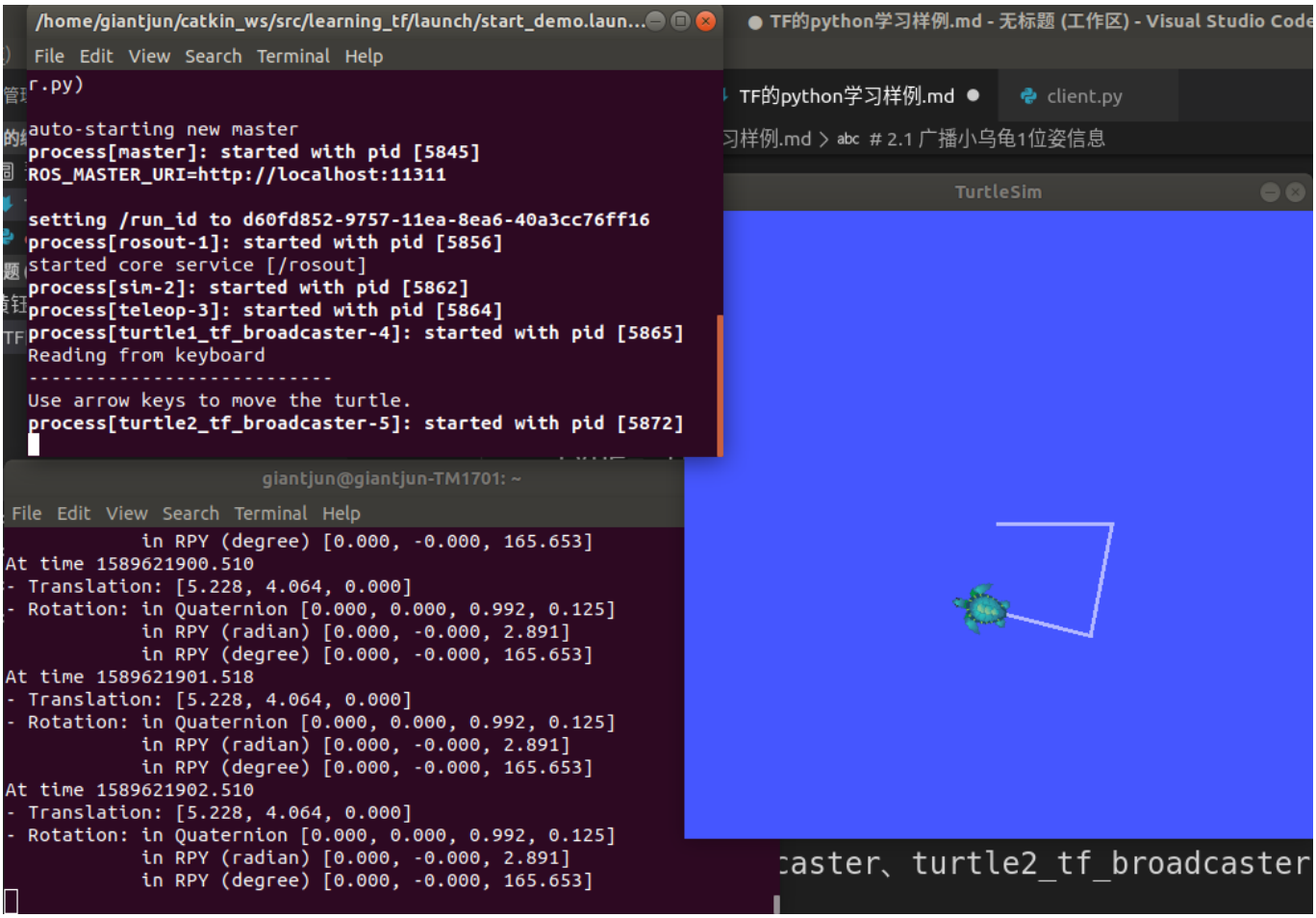
start_demo.launch文件声明了将要运行的节点（实际映射的节点名）sim、teleop、turtle1_tf_broadcaster、turtle2_tf_broadcaster的信息，各字段含义比较简单。

- 运行程序

```
$ roslaunch learning_tf start_demo.launch
```

在另一个终端中输入

```
$ rosrun tf tf_echo /world /turtle1
```



如上图所示，左上角的命令行是第一条命令运行结果，只用选中了该窗口，并使用方向键才能才能控制小乌龟运行。右侧为小乌龟的图形化显示界面（只是显示！！！不能控制）。右下角命令行，为第二条命令运行结果，显示小乌龟位姿信息。

2.2 实现对小乌龟1位姿的监听

- 创建监听运行文件

```
$ roscd learning_tf
$ touch nodes/turtle_tf_listener.py
$ chmod +x nodes/turtle_tf_listener.py
$ gedit nodes/turtle_tf_listener.py
```

- 输入代码

```
#!/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('tf_turtle')

    listener = tf.TransformListener()

    # 等待服务“spawn”，直至其不被其他进程调用时，继续往下运行
    rospy.wait_for_service('spawn')
    # 向master节点注册本节点为“spawn”的客户节点，turtlesim.srv.Spawn为消息数据结构
    spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
    # 创建请求信息，spwaner是turtlesim中的一个服务！
    # 作用是再生成一直小乌龟！
    # 从下面的参数我们可以知道，另一只小乌龟的坐标是（4,2），初始方向交是0度
    # 小乌龟名字是“turtle2”
    spawner(4, 2, 0, 'turtle2')

    # 直接绕开键盘，通过我们的设置，让小乌龟2听从小乌龟1的“指挥”
    turtle_vel = rospy.Publisher('turtle2/cmd_vel',
geometry_msgs.msg.Twist,queue_size=1)

    # 设置监听周期为10秒
    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        try:
            # 指明返回的变换坐标，是从“turtle2”到“turtle1”，
            # 在“rospy.Time(0)”(也就是最新的时间)转换获得的
            (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()
```

- 在 start_demo.launch 文件内添加节点信息 打开 start_demo.launch

```
$ roscd learning_tf
$ gedit launch/start_demo.launch
```

在标签内的最后一行添加如下代码

```
<node pkg="learning_tf" type="turtle_tf_listener.py" name="listener" />
```

- 运行代码

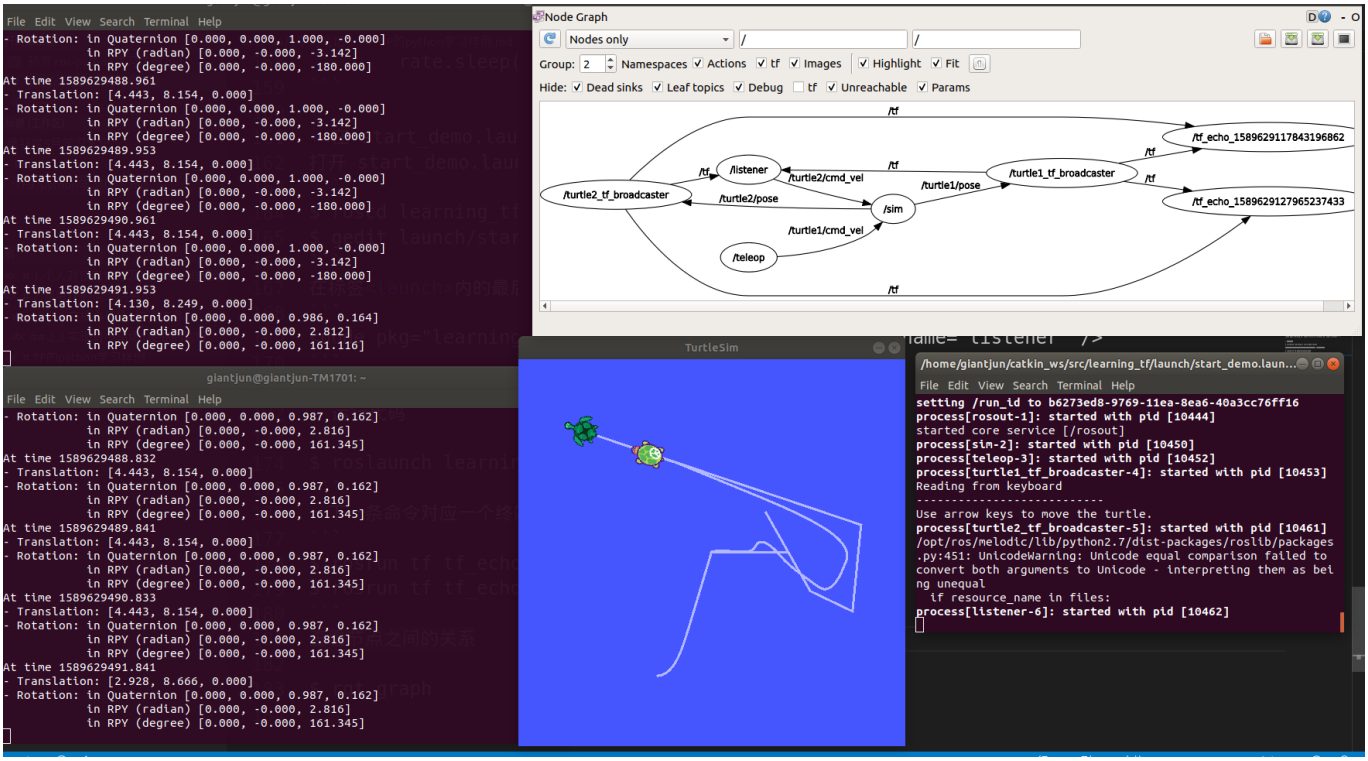
```
$ roslaunch learning_tf start_demo.launch
```

一下一条命令对应一个终端，可显示小乌龟1、2的位姿信息

```
$ rosrun tf tf_echo/world/turtle1
$ rosrun tf tf_echo/world/turtle2
```

查看节点之间的关系

```
$ rqt_graph
```



控制小乌龟1运动，小乌龟2将伴随运动到小乌龟1的位置。

结合 start_demo.launch 和节点关系图，我们可以重新对程序进行分析。图中涉及的节点较多，关系也较为复杂，但也十分精妙有趣，再重新会看 turtle_tf_broadcaster.py 和 turtle_tf_listener.py 的代码有会产生新的认识，可惜用文字描述较为冗杂难懂，所以读者可以自行分析，这里不再赘述。