

在之前的课程中我们学习了 ROS 的基本概念，包括：节点、话题、消息、功能包、工作空间等。通过小乌龟仿真器和 TT 无人机我们学习了使用 ROS 命令发布和查看话题消息，以及使用 rviz 和 rqt 工具箱，还编写了两个功能包节点，实现了两只小乌龟的转圈，以及 TT 无人机人脸和 QR code 二维码跟踪。

本节课我们学习 ROS 中的坐标变换。通过坐标变换，我们会在 turtlesim 中实现两只小乌龟的相互跟随，在之后的课程中将通过坐标变换实现 TT 无人机对 apriltag 二维码的追踪。坐标变换所涉及的 tf/tf2 库，对后续课程 mini 无人车建图导航任务也至关重要。

准备工作

更新 rmtt_ros 系列功能包，请保存好之前实现的节点避免覆盖：

从 ~/tt_ws/src/（自行创建文件夹）打开一个 terminal：

```
git clone https://github.com/cavayangtao/rmtt_ros.git
```

或者：

```
git clone https://gitee.com/cavayangtao/rmtt_ros.git
```

在该工作空间（~/tt_ws/）下打开终端：

编译工作空间：

```
catkin_make_isolated;
```

运行功能包中的节点前，请在该终端及新终端中输入：

```
source devel_isolated/setup.bash;
```

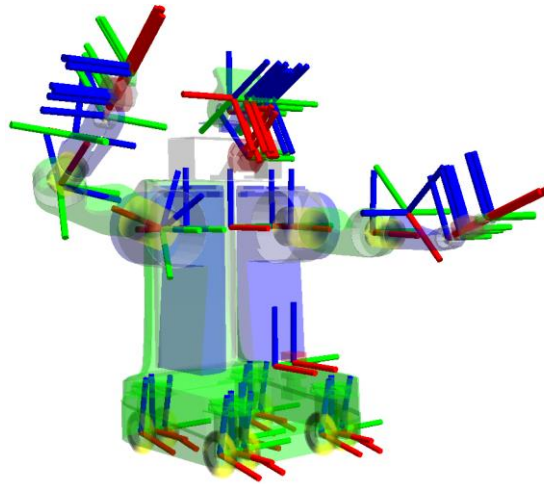
或在 home 中打开终端，输入：gedit ~/.bashrc，将以上命令粘贴入文档并保存；

关闭所有终端，再次打开新终端可直接运行功能包中的节点。

注意我们今天使用了 catkin_make_isolated 而非 catkin_make 进行编译，catkin_make_isolated 会对工作空间中各个功能包共同的依赖库分别单独编译。

tf 变换基础知识

坐标变换是机器人学中一个非常基础同时也是非常重要的概念。机器人本体和机器人的工作环境中往往存在大量的组件元素，在机器人设计和机器人应用中都会涉及不同组件的位置和姿态，这就需要引入坐标系以及坐标变换的概念。



只要我们能够知道当前坐标系在参考坐标系的描述(平移和旋转)，我们就可以把当前坐标系里任何一个点的坐标变换成参考坐标系里的坐标。刚体的任一姿态（刚体坐标系相对于参考坐标系）可以经由三次基本旋转得到，用三个角来描述，这就是欧拉角。

ROS 中使用 `tf` 软件库进行坐标转换，现在已经有 `tf2` 库了，而且比 `tf` 库更加功能强大，建议优先选用 `tf2` 进行开发。在坐标变换中，两个坐标轴的关系（也就是转换信息）用一个 6 自由度的相对位姿表示：平移量（translation）+ 旋转量（rotation）。其中平移量就是一个三维向量，旋转量可以用一个旋转量矩阵表示，`tf` 中没有表示旋转量矩阵的类型，而是通过四元数类型 `tf::Quaternion` 来表示。四元数是刚体姿态的另一种描述方式，理论基础是，刚体姿态可以经过某一特定轴经一次旋转一定角度得到，等价于欧拉角，等价于旋转矩阵。

利用 `tf` 库管理坐标系主要要做的就是两件事：**监听 `tf` 变换**和**广播 `tf` 变换**。

监听 `tf` 变换：接收并缓存系统中发布的所有参考系变换，并从中查询所需要的参考系变换。

广播 `tf` 变换：向系统中广播参考系之间的坐标变换关系。

`tf` 变换描述某个坐标系(`child_frame_id`)相对于另一个参考坐标系(`frame_id`)在某个时刻的位姿关系（平移+旋转矩阵）。`tf` 维护的坐标系关系其实是将有关坐标系话题发布在 `tf::tfMessage` 类型消息中，该消息定义为 `geometry_msgs/TransformStamped` 类型的向量：

```
geometry_msgs/TransformStamped[] transforms
```

```
std_msgs/Header header
```

```
uint32 seq
```

```
time stamp
```

```
string frame_id
```

```
string child_frame_id

geometry_msgs/Transform transform

geometry_msgs/Vector3 translation

float64 x

float64 y

float64 z

geometry_msgs/Quaternion rotation

float64 x

float64 y

float64 z

float64 w
```

通过 tf 变换实现小乌龟跟踪

tf 变换比较抽象，我们将通过小乌龟跟踪的例程来理解 tf 工具及其使用。

首先确保已经安装了该例程：

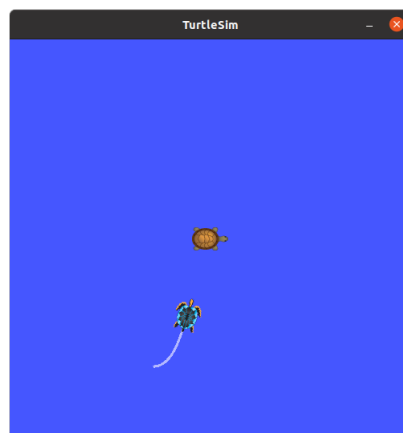
```
sudo apt-get install ros-noetic-turtle-tf2
```

由于 ROS noetic 已经全面支持 python3，需要解决代码兼容性问题，在终端输入：

```
sudo apt-get install python-is-python3
```

现在，可以运行例程了：

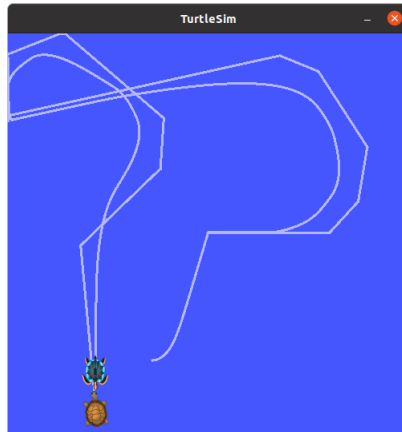
```
roslaunch turtle_tf2 turtle_tf2_demo.launch
```



应该能看到两只小乌龟。

打开键盘控制节点，通过方向键进行控制，看看会发生什么：

```
roslaunch turtlesim turtle_teleop_key
```



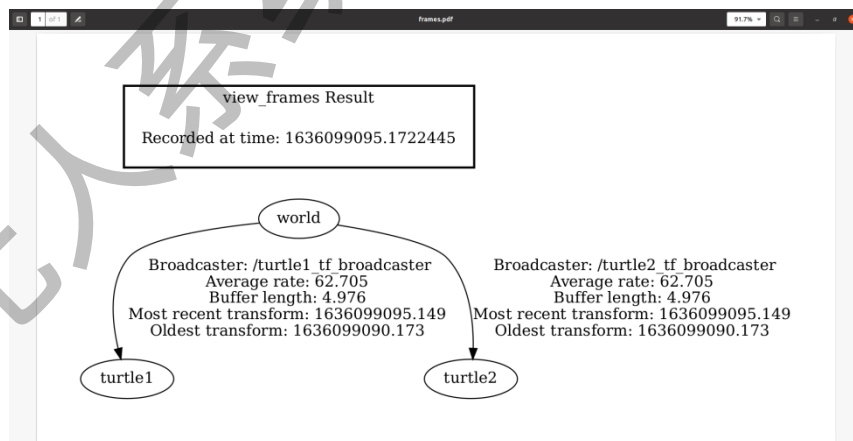
该例程使用 tf2 库创建了三个坐标系：世界坐标系、turtle1 坐标系和 turtle2 坐标系。使用 tf2 广播器发布 turtle 标系，并使用 tf2 监听并计算 turtle 标系中的差异并移动一只乌龟跟随另一只乌龟。

由于 tf 工具箱和 ROS noetic 已经部分不兼容了，先安装 tf2 工具箱，然后查看一下当前有哪几个坐标系，它们之间存在怎样的坐标变换。tf2 工具箱已经更新在了 rmtt_ros 文件夹中，也可以自行安装：

```
sudo apt-get install ros-noetic-tf2-tools
```

```
roslaunch tf2_tools view_frames.py
```

以上命令会将 tf 树的 pdf 文件保存在当前目录。



在上图中，可以看到 tf2 广播的三个坐标系：世界坐标系，turtle1 坐标系和 turtle2 坐标系，并且世界坐标系是 turtle1 和 turtle2 坐标系的父级。view_frames 还报告一些诊断信息，这些信息有关何时接收到最旧和最新的坐标系转换，以及发布的速度。

现在要让 turtle2 跟随 turtle1 运动，也就是让 turtle2 的本体坐标系需要向 turtle1 的本体坐标系移动，这就需要知道 turtle2 和 turtle1 之间的坐标变换关系。使用 tf_echo 工具在 tf 树中查找小乌龟坐标系间的转换关系，指令格式：

```
roslaunch tf_echo [reference_frame] [target_frame]
```

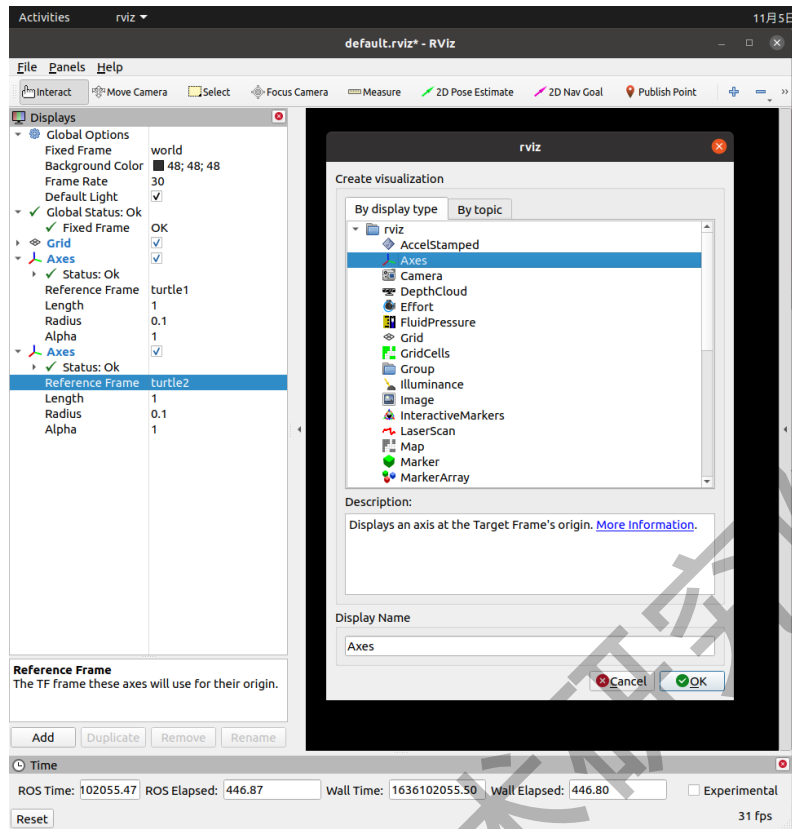
看一下 turtle2 坐标系相对于 turtle1 坐标系的变换：

```
roslaunch tf_echo turtle1 turtle2
```

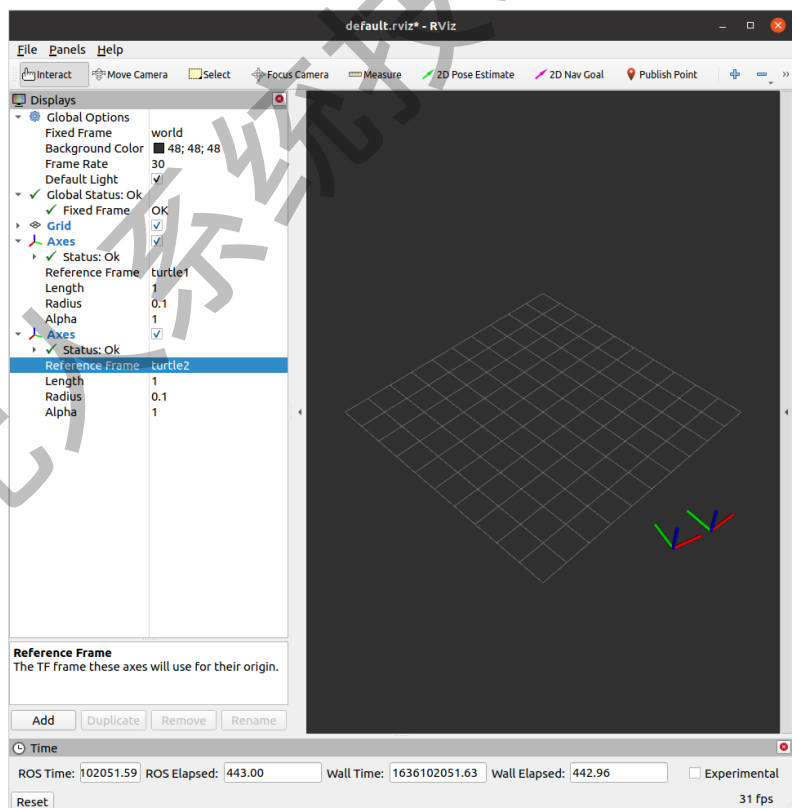
```
- Rotation: in Quaternion [0.000, 0.000, -0.261, 0.965]
           in RPY (radian) [0.000, 0.000, -0.528]
           in RPY (degree) [0.000, 0.000, -30.258]
At time 1636100724.570
- Translation: [-3.010, 0.541, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.110, 0.994]
           in RPY (radian) [0.000, 0.000, -0.220]
           in RPY (degree) [0.000, 0.000, -12.591]
At time 1636100725.562
- Translation: [-3.409, 0.298, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.052, 0.999]
           in RPY (radian) [0.000, 0.000, -0.105]
           in RPY (degree) [0.000, 0.000, -6.013]
At time 1636100726.557
- Translation: [-3.416, 0.171, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.028, 1.000]
           in RPY (radian) [0.000, 0.000, -0.056]
           in RPY (degree) [0.000, 0.000, -3.206]
At time 1636100727.564
- Translation: [-2.037, 0.100, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.025, 1.000]
           in RPY (radian) [0.000, 0.000, -0.049]
           in RPY (degree) [0.000, 0.000, -2.828]
```

其中 turtle1 是 parent 坐标系，turtle2 是 child 坐标系。当移动乌龟时，会看到随着两只乌龟相对移动，坐标变换发生了变化。

在新的终端输入：rviz。在 rviz 中查看两只小乌龟本体坐标系的运动。



如图添加两个 Axes 然后选择正确的 Fixed Frame 和 Reference Frame。



通过键盘控制小乌龟运动，会看到两个坐标系也动了，并且一个跟随一个。

编写 tf 广播和监听节点

本节我们将学习以上过程是如何实现的，learning_tf2 的功能包已经更新在了 rmtt_ros 文件夹中，也可以自行创建一个名为 learning_tf2 的功能包。这功能包依赖 tf2、tf2_ros、roscpp、rospy 和 turtlesim 库。若自行创建，在 ~tt_ws/src 目录下打开终端，输入以下命令：

```
catkin_create_pkg learning_tf2 tf2 tf2_ros roscpp rospy turtlesim
```

启动 ROS master:

```
roscore
```

启动一个小乌龟仿真节点:

```
roslaunch turtlesim turtlesim_node
```

还记不记得我们如何通过 rosservice call /spawn 服务命令调出一只新的小乌龟？我们也可以写一个节点实现此功能。参考，在 /learning_tf2/src 文件夹下的 turtle_spawn.py 节点：



```
1#!/usr/bin/env python3
2
3import rospy
4from turtlesim.srv import Spawn, SpawnRequest, SpawnResponse
5
6if __name__ == "__main__":
7
8    rospy.init_node("turtle_spawn")
9    # 创建服务客户端
10    client = rospy.ServiceProxy("/spawn", Spawn)
11    # 等待服务启动
12    client.wait_for_service()
13    # 创建请求数据
14    req = SpawnRequest()
15    req.x = 1.0
16    req.y = 1.0
17    req.theta = 3.14
18    req.name = "turtle2"
19    # 发送请求并处理响应
20    try:
21        response = client.call(req)
22        rospy.loginfo("乌龟创建成功, 名字是:%s", response.name)
23    except Exception as e:
24        rospy.loginfo(e)
```

运行:

```
roslaunch learning_tf2 turtle_spawn.py
```

注意运行先要开启 python 文件权限，之后将不再赘述。

参考在 /learning_tf2/src 文件夹下名为 turtle1_tf2_broadcaster.py 的节点，该文件订阅 turtle1 的 pose，然后广播相对 world 的坐标系信息：

```
Open  turtle1_tf2_broadcaster.py  Save  -  x
~/tt_ws/src/learning_tf2/src

1 #!/usr/bin/env python3
2
3 import rospy
4 import tf2_ros
5 from tf.transformations import quaternion_from_euler, euler_from_quaternion
6 import turtlesim.msg
7 from geometry_msgs.msg import TransformStamped
8
9 def handle_turtle_pose(msg, turtlename):
10     br = tf2_ros.TransformBroadcaster()
11
12     t = TransformStamped()
13     # Read message content and assign it to
14     # corresponding tf variables
15     t.header.stamp = rospy.Time.now()
16     t.header.frame_id = 'world'
17     t.child_frame_id = turtlename
18
19     # Turtle only exists in 2D, thus we get x and y translation
20     # coordinates from the message and set the z coordinate to 0
21     t.transform.translation.x = msg.x
22     t.transform.translation.y = msg.y
23     t.transform.translation.z = 0.0
24
25     # For the same reason, turtle can only rotate around one axis
26     # and this why we set rotation in x and y to 0 and obtain
27     # rotation in z axis from the message
28     q = quaternion_from_euler(0, 0, msg.theta)
29     t.transform.rotation.x = q[0]
30     t.transform.rotation.y = q[1]
31     t.transform.rotation.z = q[2]
32     t.transform.rotation.w = q[3]
33
34     br.sendTransform(t)
35
36 if __name__ == '__main__':
37     rospy.init_node('turtle1_tf2_broadcaster')
38     turtlename = 'turtle1'
39     rospy.Subscriber('/%s/pose' % turtlename,
40                     turtlesim.msg.Pose,
41                     handle_turtle_pose,
42                     turtlename)
43     rospy.spin()

Python 3  Tab Width: 8  Ln 1, Col 23  INS
```

实现流程：1. 导包；2. 初始化 ros 节点；3. 创建订阅对象；4. 回调函数处理订阅的 pose 信息；5. 回调函数中创建 tf 广播器；6. 将 pose 信息转换成 TransformStamped；7. 发布。

完成后运行：


```
roslaunch learning_tf2 turtle1_tf2_broadcaster.py
```

请自行写一个节点发布 turtle2 相对 world 的坐标系信息，在 /learning_tf2/src 文件夹下创建名为 turtle2_tf2_broadcaster.py 的 python 文件，修改以上代码的节点名和 turtlename。

完成后运行：

```
roslaunch learning_tf2 turtle2_tf2_broadcaster.py
```


现在创建最后一个节点，在 /learning_tf2/src 文件夹下创建名为 turtle_tf2_listener.py 的 python 文件。该节点订阅 turtle1 和 turtle2 的 tf 广播信息，查找并转换时间最近的 TF 信息，将 turtle1 转换成相对 turtle2 的坐标，计算线速度和角速度并发布。



```
1 #! /usr/bin/env python3
2
3 import rospy
4 import tf2_ros
5 from geometry_msgs.msg import TransformStamped, Twist
6 import math
7
8 if __name__ == "__main__":
9
10     rospy.init_node("sub_tfs_p")
11
12     # 创建 TF 订阅对象
13     buffer = tf2_ros.Buffer()
14     listener = tf2_ros.TransformListener(buffer)
15     # 处理订阅到的 TF
16     rate = rospy.Rate(10)
17     # 创建速度发布对象
18     pub = rospy.Publisher("/turtle2/cmd_vel", Twist, queue_size=1000)
19     while not rospy.is_shutdown():
20
21         rate.sleep()
22         try:
23             # def lookup_transform(self, target_frame, source_frame, time,
24             # timeout=rospy.Duration(0.0)):
25             trans = buffer.lookup_transform("turtle2", "turtle1", rospy.Time(0))
26             # rospy.loginfo("相对坐标: (%.2f, %.2f, %.2f)",
27             #             trans.transform.translation.x,
28             #             trans.transform.translation.y,
29             #             trans.transform.translation.z
30             #             )
31             # 根据转变后的坐标计算出速度和角速度信息
32             twist = Twist()
33             # 间距 = x^2 + y^2 然后开方
34             twist.linear.x = 0.5 *
35             math.sqrt(math.pow(trans.transform.translation.x, 2) +
36             math.pow(trans.transform.translation.y, 2))
37             twist.angular.z = 4 * math.atan2(trans.transform.translation.y,
38             trans.transform.translation.x)
39             pub.publish(twist)
40         except Exception as e:
41             rospy.logwarn(e)
```

在创建的节点中实现以上代码。

实现流程：1. 导包；2. 初始化节点；3. 创建 tf 订阅对象；4. 处理订阅到的 tf；5. 查找坐标系的相对关系；6. 生成速度信息然后发布。

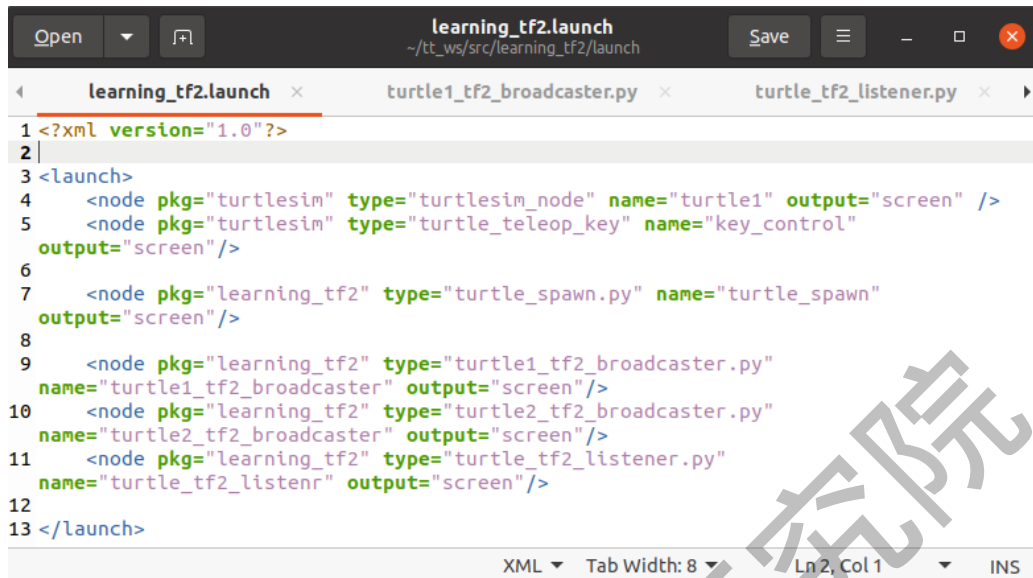
完成后运行：

roslaunch learning_tf2 turtle_tf2_listener.py

运行小乌龟键盘控制节点：

roslaunch turtlesim turtle_teleop_key

通过键盘遥控，看看有没有实现小乌龟的运动跟随。



```
1 <?xml version="1.0"?>
2
3 <launch>
4   <node pkg="turtlesim" type="turtlesim_node" name="turtle1" output="screen" />
5   <node pkg="turtlesim" type="turtle_teleop_key" name="key_control"
6     output="screen"/>
7   <node pkg="learning_tf2" type="turtle_spawn.py" name="turtle_spawn"
8     output="screen"/>
9   <node pkg="learning_tf2" type="turtle1_tf2_broadcaster.py"
10     name="turtle1_tf2_broadcaster" output="screen"/>
11   <node pkg="learning_tf2" type="turtle2_tf2_broadcaster.py"
12     name="turtle2_tf2_broadcaster" output="screen"/>
13   <node pkg="learning_tf2" type="turtle_tf2_listener.py"
14     name="turtle_tf2_listener" output="screen"/>
15 </launch>
```

关闭所有终端，在/learning_tf2/launch 文件夹下，创建名为 learning_tf2.launch 的 launch 文件组织需要运行的节点。完成后运行：

```
roslaunch learning_tf2 learning_tf2.launch
```