机器人操作系统

机电工程学院

机器人智能制造研究团队

2025 · 秋

课程安排

01 第一章 ROS概述与环境搭建

02 第二章 ROS通信机制

03 第三章 ROS架构与运行管理

04 第四章 ROS常用组件

05 第五章 机器人建模与仿真

06 第六章 ROS进阶功能

第 四 章

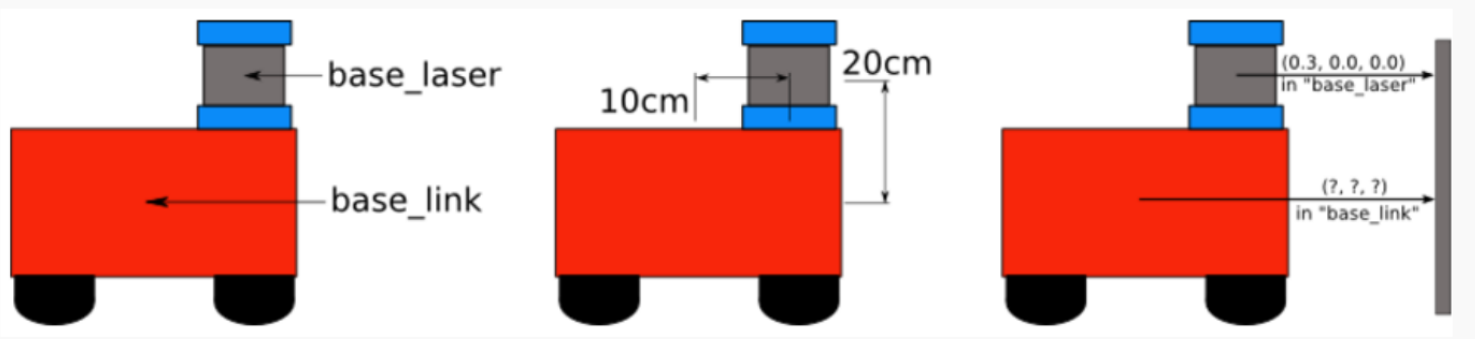
ROS常用组件

目录

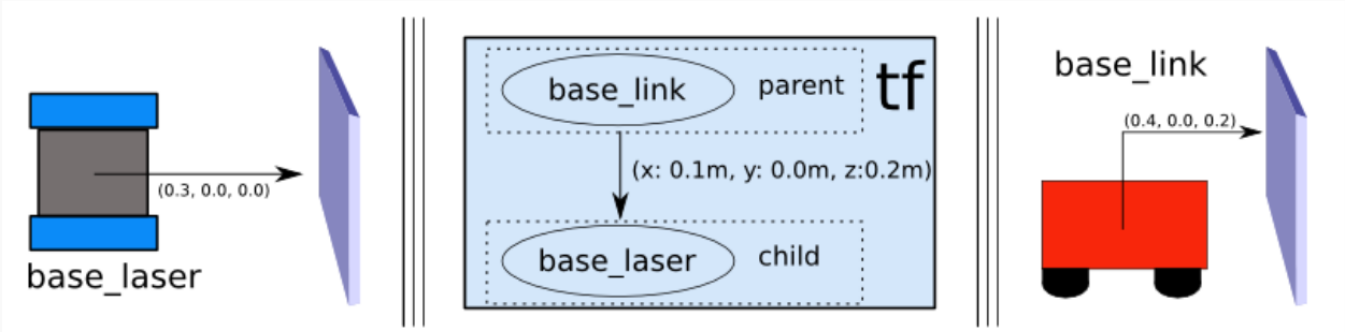
CONTENTS

TF坐标变换

现有一移动式机器人底盘，在底盘上安装了一雷达，雷达相对于底盘的偏移量已知，现雷达检测到一障碍物信息，获取到坐标分别为(x,y,z)，该坐标是以雷达为参考系的，如何将这个坐标转换成以小车为参考系的坐标呢?

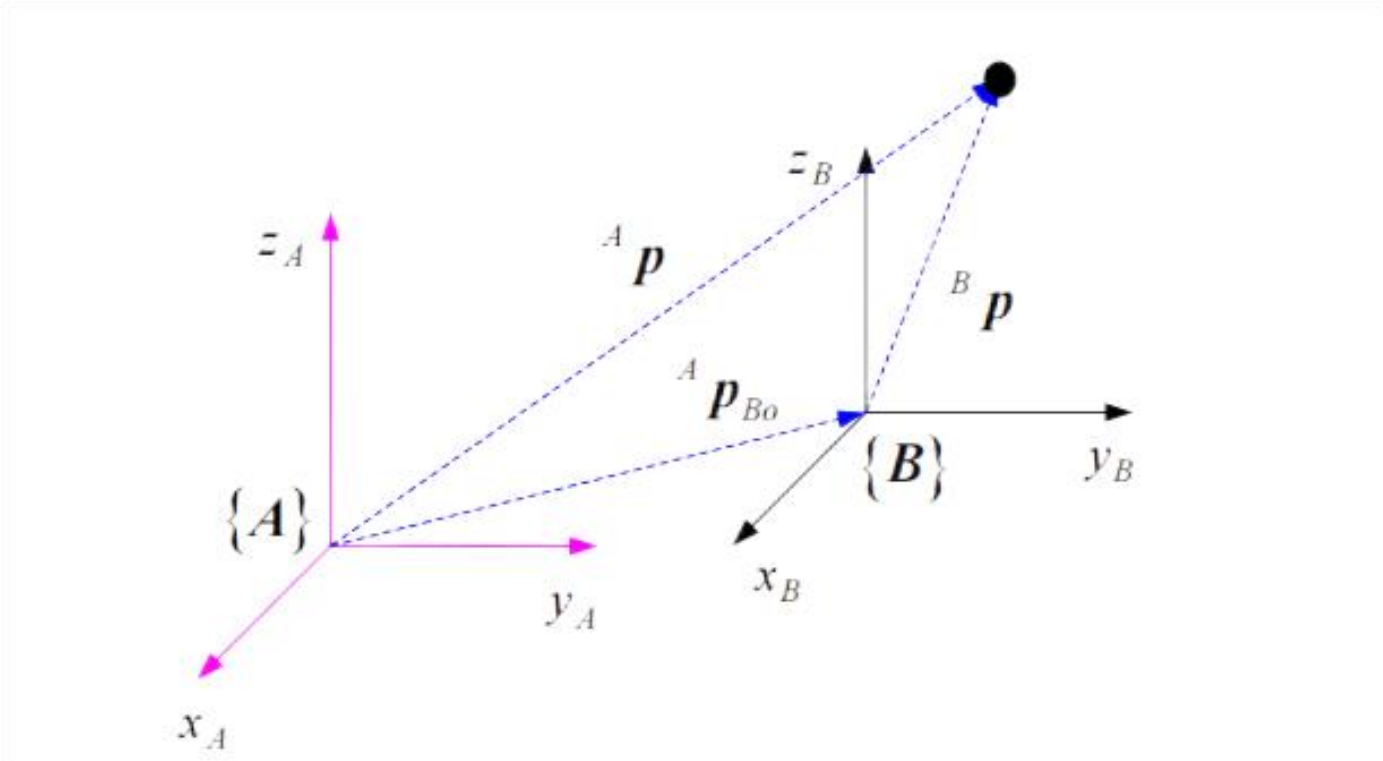


移动机器人的本体坐标系与雷达坐标系



坐标系之间的数据变换

两个坐标系之间的关系由6自由度的相对位姿表示。
坐标系B在坐标系A中的相对位姿应包含两部分：
① 由A向B的平移（原点的平移）
② A的各轴在B中的旋转
上述两部分可理解为：先平移（原点重合），再旋转（各轴重合）

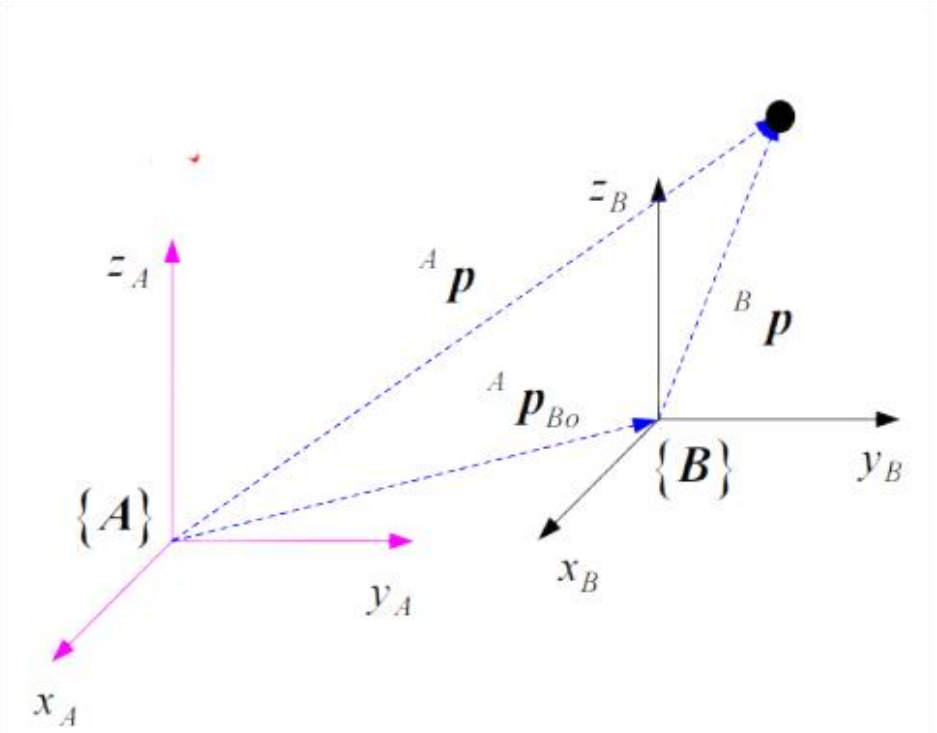


在坐标系｛A｝中，空间任意一点可表示为列矢量 ${}^{A}\boldsymbol{p}$

$$ {}^{A}\boldsymbol{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} $$

- **平移变换方程—｛A，B｝方位相同**

$$ {}^{A}\boldsymbol{p} = {}^{B}\boldsymbol{p} + {}^{A}\boldsymbol{p}_{B_0} $$



- 旋转变换方程—{A，B}

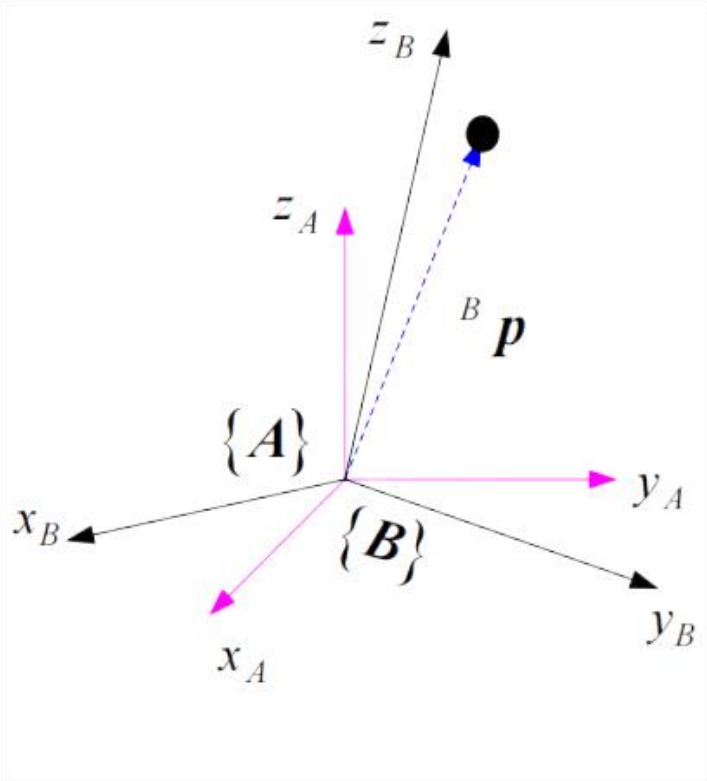
$$^{A}\boldsymbol{p} = {}^{A}_{B}R\,{}^{B}\boldsymbol{p}$$

- 正交矩阵：

$${}^{B}_{A}\boldsymbol{R} = {}^{A}_{B}\boldsymbol{R}^{-1} = {}^{A}_{B}\boldsymbol{R}^{\mathrm{T}}$$

- 一般变换方程—{A，B}方位和原点均不同

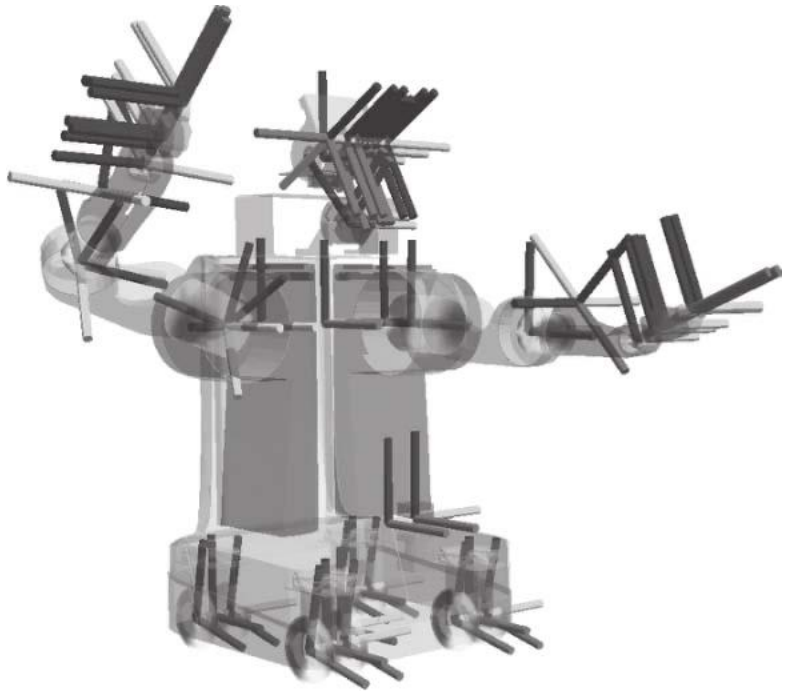
$$^{A}\boldsymbol{p} = {}^{A}_{B}\boldsymbol{R}\,{}^{B}\boldsymbol{p} + {}^{A}\boldsymbol{p}_{B_0}$$



坐标系B相对于A方位矩阵

坐标系B的坐标原点相对于A的位置

- TF是一个让用户随时间跟踪多个坐标系的功能包，它使用树形数据结构，根据时间缓冲并维护多个坐标系之间的坐标变换关系，可以帮助开发者在任意时间、在坐标系间完成点、向量等坐标的变换。



一个机器人系统通常有很多三维坐标系，而且会随着时间的推移发生变化，如世界坐标系（WorldFrame）、基坐标系（Base Frame）、机械夹爪坐标系（Gripper Frame）、机器人头部坐标系（Head Frame）等。

TF可以时间为轴跟踪这些坐标系（默认10s之内）。

想要使用TF功能包，总体来说需要以下两个步骤。

1）监听TF变换

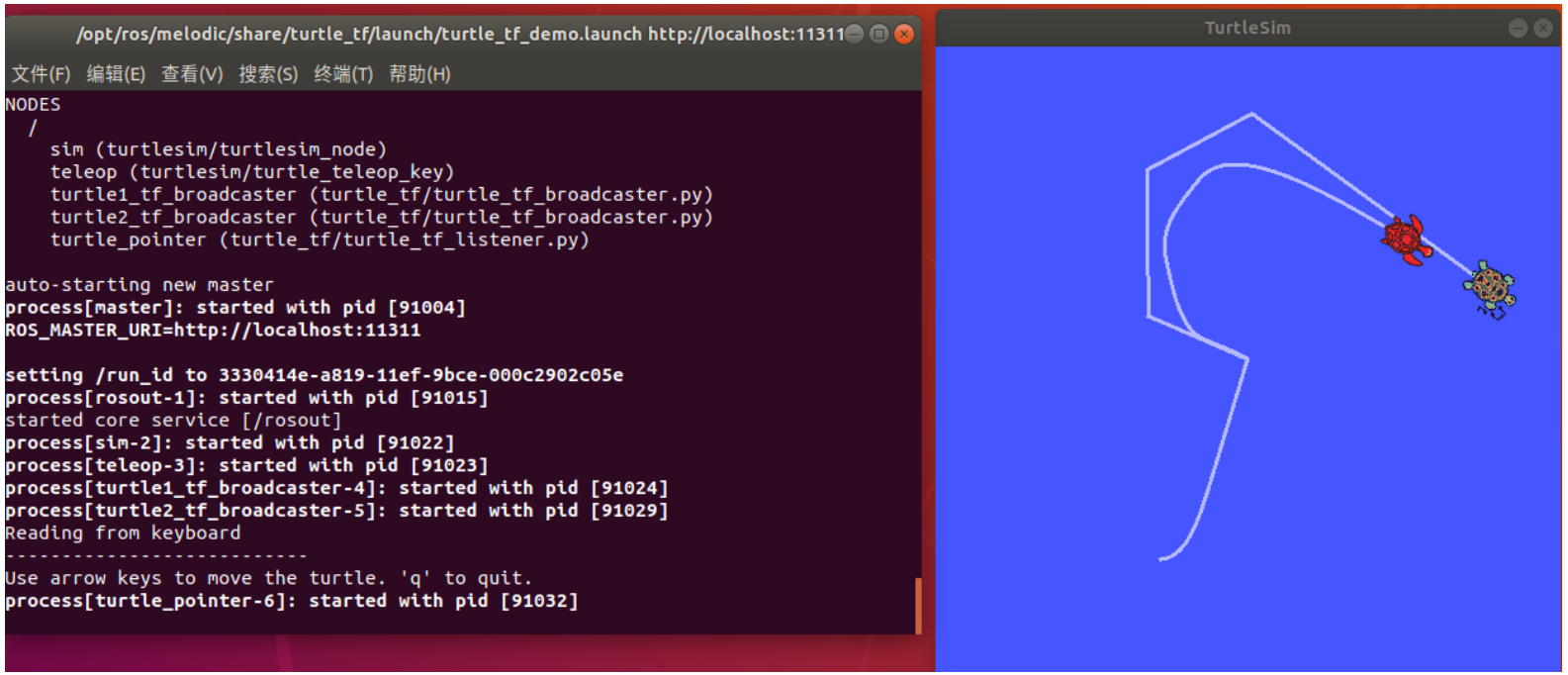
- 接收并缓存系统中发布的所有坐标变换数据，并从中查询所需要的坐标变换关系。

2）广播TF变换

- 向系统中广播坐标系之间的坐标变换关系。系统中可能会存在多个不同部分的TF变换广播，每个广播都可以直接将坐标变换关系插入TF树中，不需要再进行同步。

- sudo apt-get install ros-melodic-ros-tutorials ros-melodic-geometry-tutorials ros-melodic-rviz ros-melodic-rosbash ros-melodic-rqt-tf-tree
- roslaunch turtle_tf turtle_tf_demo.launch

- 一旦启动了turtlesim，你可以使用键盘上的箭头键来控制中心的乌龟在turtlesim中移动。选择roslaunch的终端窗口，这样可以捕捉你的按键来控制乌龟的运动。



TF Demo

- 本演示使用 tf 库创建了三个坐标系：一个**world坐标系**、一个**turtle1坐标系**和一个**turtle2坐标系**。
- **使用 tf 广播器来发布乌龟的坐标系**
- **使用 tf 监听器来计算乌龟之间的坐标差异**
- 使一只乌龟跟随另一只乌龟
- tf/Tutorials/Introduction to tf - ROS Wiki

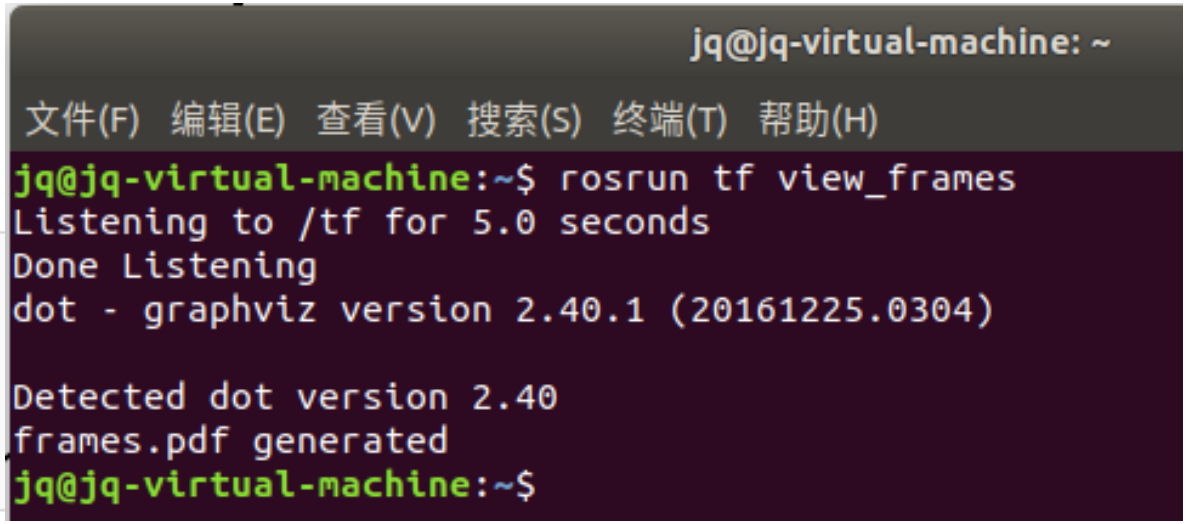
- 现在让我们看看 tf 是如何用来创建这个演示的。我们可以使用 tf 工具来观察 tf 在幕后做了什么。

`view_frames` 会创建一个图表，显示 tf 通过 ROS 广播的各个坐标系。

运行以下命令可以生成当前通过 tf 广播的坐标系图：

```bash
rosrun tf view_frames
```

注意在另外一个终端中输入命令行



执行该命令后，它将在当前工作目录中生成一个名为 `frames.pdf` 的文件，其中包含所有广播的坐标系以及它们之间的关系。你可以使用 PDF 查看工具打开该文件来查看坐标系图。

1. `Listening to /tf for 5.0 seconds`

- 表示工具正在监听 `/tf` 主题（topic）的数据流，时间为 **5秒钟**。
- `/tf` 是 ROS 系统中用于传输坐标变换（transforms）数据的主题，`view_frames` 工具会收集这段时间内广播的所有坐标系信息。

2. `Done Listening`

- 表示工具完成了对 `/tf` 主题的监听。
- 此时，工具已经收集到所有需要的数据。

```
jq@jq-virtual-machine:~$ rosrun tf view_frames
Listening to /tf for 5.0 seconds
Done Listening
dot - graphviz version 2.40.1 (20161225.0304)
```

3. `dot - graphviz version 2.40.1 (20161225.0304)`

- 表示 `view_frames` 使用的图形绘制工具 **Graphviz** 的版本信息。
- `Graphviz` 是用来生成图表的工具，`view_frames` 利用它生成 PDF 格式的坐标系图。

4. `Detected dot version 2.40`

- 再次确认 `dot`（Graphviz 的核心工具）版本为 **2.40**。
- 这个工具负责根据采集的数据绘制坐标系的关系图。

```
Detected dot version 2.40
frames.pdf generated
jq@jq-virtual-machine:~$
```

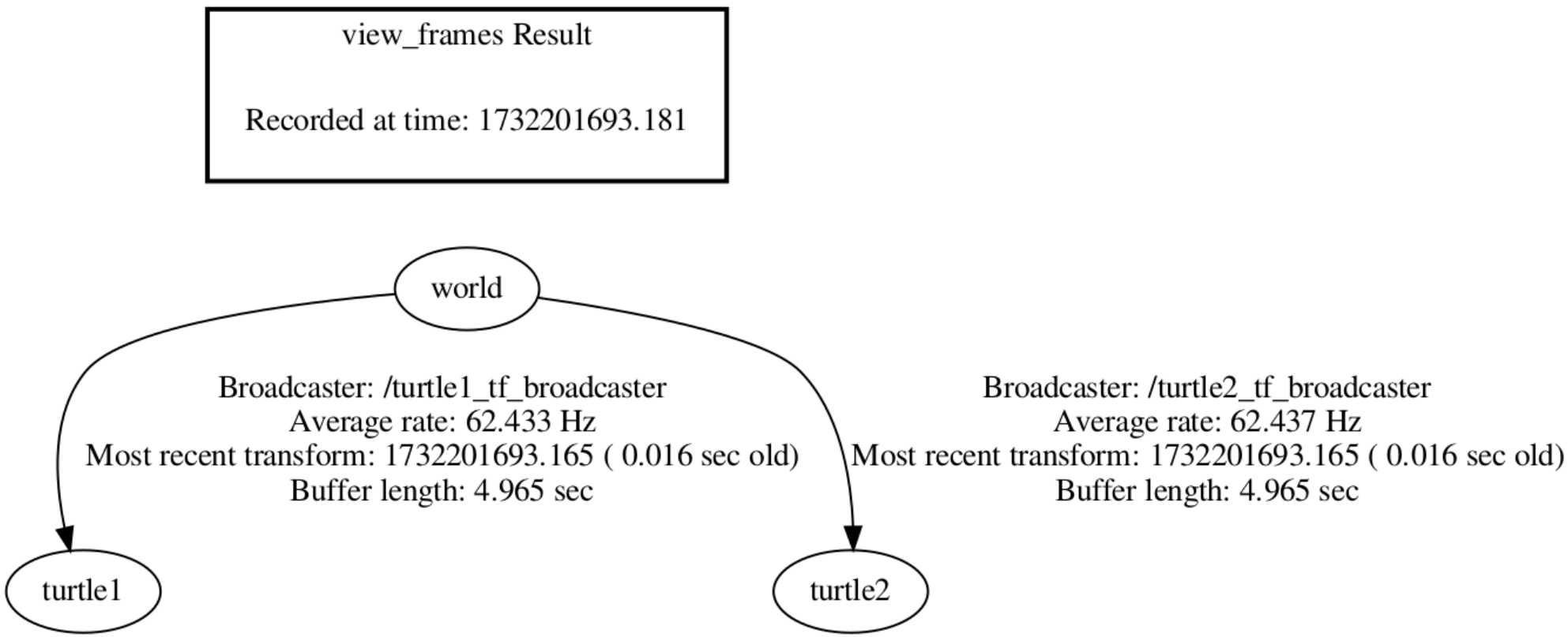
5. `frames.pdf generated`

- 表示 `view_frames` 已成功生成 PDF 文件，文件名为 `frames.pdf`。
- 该文件包含所有在监听时间内广播的坐标系，以及它们之间的关系和转换信息。

运行以下命令即可打开 `frames.pdf` 文件：

```bash
evince frames.pdf
```

- `evince` 是一个常见的 PDF 查看工具，用于在 GUI 界面中打开 PDF 文件。
- `frames.pdf` 是由 `rosrun tf view_frames` 命令生成的文件，包含广播的坐标系及其关系的图表。

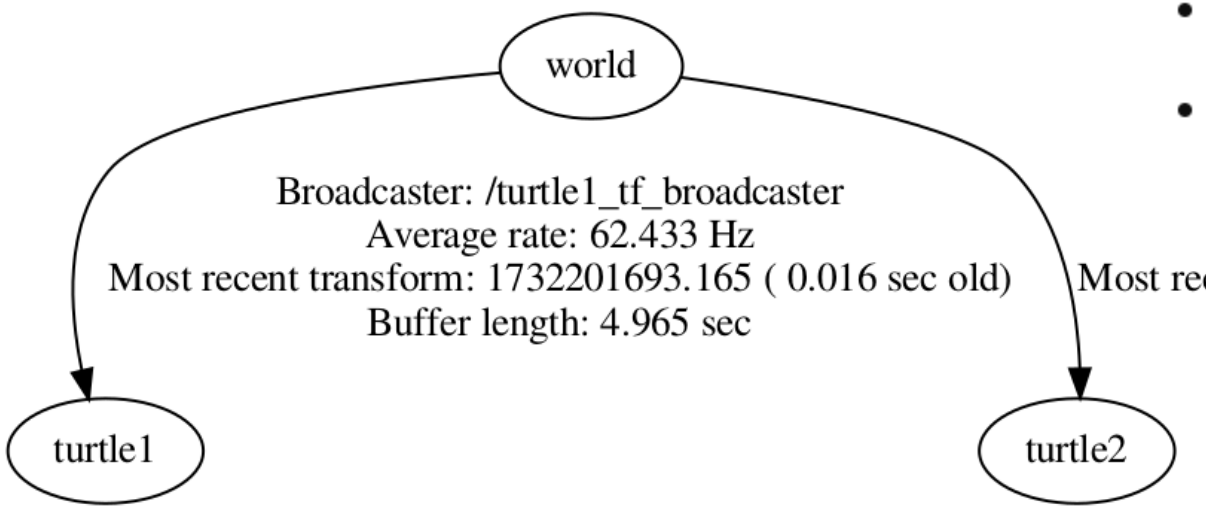


view_frames Result

Recorded at time: 1732201693.181

- `view_frames Result`：表示这是通过 `rosrun tf view_frames` 命令生成的结果。
- `Recorded at time: 1732201693.181`：显示记录的时间戳（单位为秒，通常是ROS中的系统时间）。

每条从 `world` 到子坐标系的箭头表示一个 TF 广播器。



- `world` **坐标系**：
  - 这是场景的全局坐标系，是所有其他坐标系的父坐标系。
  - 两个子坐标系 `turtle1` 和 `turtle2` 都从 `world` 坐标系派生。

- `turtle1` **和** `turtle2` **坐标系**：
  - 这些坐标系分别由 `turtle1` 和 `turtle2` 的位置和方向定义。
  - `turtle1` 和 `turtle2` 坐标系各自由一个 TF 广播器发布。

- `/turtle1_tf_broadcaster`（左侧箭头）：
  - `Average rate: 62.433 Hz`：表示 `turtle1` 坐标系的变换以约 62.433Hz 的频率广播。
  - `Most recent transform: 1732201693.165 (0.016 sec old)`：
    - 最近一次广播的时间戳是 `1732201693.165`。
    - 变换延迟为 `0.016 秒`。
  - `Buffer length: 4.965 sec`：缓存了最近 4.965 秒的变换信息。

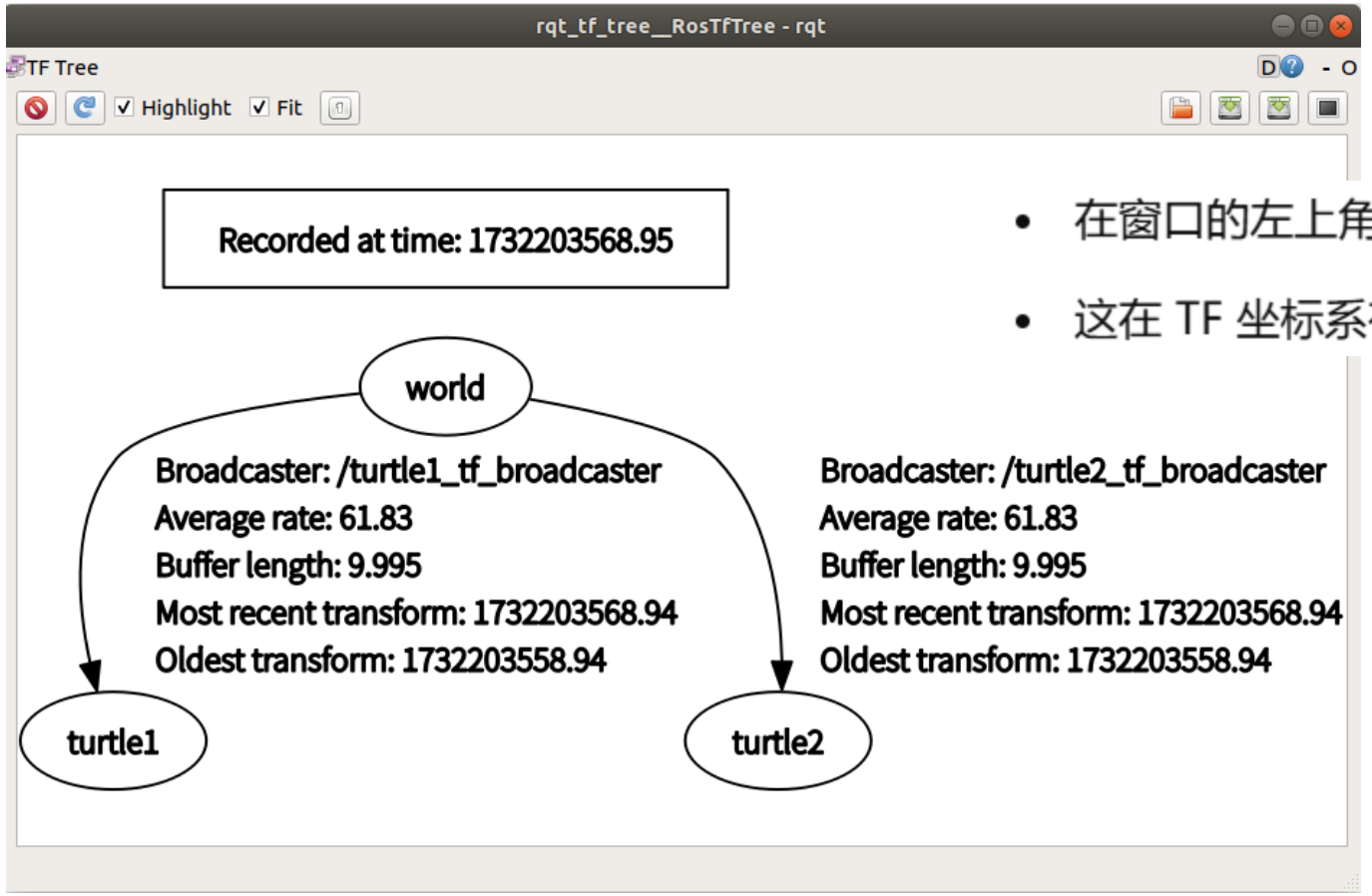
运行以下命令启动 `rqt_tf_tree` 工具，用于实时查看 TF 坐标系的树状结构：

bash

注意在另外一个终端中输入命令行

```
rosrun rqt_tf_tree rqt_tf_tree
```

- 可以实时监控 TF 广播的坐标系，分析系统的坐标变换层次结构。
- 帮助调试坐标系之间的关系是否正确，或者是否有丢失的坐标系。



- 在窗口的左上角有一个**刷新按钮**，点击它可以手动刷新坐标系的显示。
- 这在 TF 坐标系有动态变化时非常有用。

命令格式：

```bash
rosrun tf tf_echo [reference_frame] [target_frame]
```

`tf_echo` 用于报告 ROS 中任意两个坐标系之间的变换关系。

- `reference_frame`：参考坐标系（源坐标系）。
- `target_frame`：目标坐标系。

查看 `turtle2` 坐标系相对于 `turtle1` 坐标系的变换：

思考：怎么确定哪知乌龟是turtle1？

- 示例命令：

```bash
rosrun tf tf_echo turtle1 turtle2
```

- 解释： 这个命令会计算 `turtle2` 坐标系相对于 `turtle1` 坐标系的变换关系。其结果等价于：
  - `turtle1` 到 `world` 的变换乘以 `world` 到 `turtle2` 的变换。

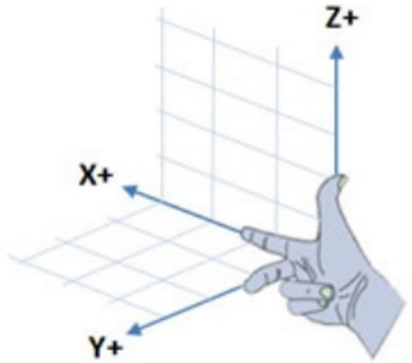
```
jq@jq-virtual-machine: ~
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
At time 1732204136.294
- Translation: [-0.002, -0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.029, 1.000]
            in RPY (radian) [0.000, -0.000, 0.058]
            in RPY (degree) [0.000, -0.000, 3.311]
At time 1732204137.280
- Translation: [-0.001, -0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.029, 1.000]
            in RPY (radian) [0.000, -0.000, 0.058]
            in RPY (degree) [0.000, -0.000, 3.311]
At time 1732204138.287
- Translation: [-0.001, -0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.029, 1.000]
            in RPY (radian) [0.000, -0.000, 0.058]
            in RPY (degree) [0.000, -0.000, 3.314]
At time 1732204139.297
- Translation: [-0.000, -0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.029, 1.000]
            in RPY (radian) [0.000, -0.000, 0.058]
            in RPY (degree) [0.000, -0.000, 3.317]
```

```
At time 1732204136.294
```

- 表示变换数据的时间戳，单位为秒。它表明这一条变换信息是在 ROS 时间 `1732204136.294` 采集的。

```
Translation: [-0.002, -0.000, 0.000]
```

- 表示 `turtle2` 相对于 `turtle1` 在 `x`，`y`，`z` 三个方向上的平移量。
- 这里的平移值为：
  - `x`：-0.002
  - `y`：-0.000
  - `z`：0.000



当你控制乌龟移动时，你会看到坐标变换随着两只乌龟相对位置的变化而更新。

```
Rotation: in Quaternion [0.000, 0.000, 0.029, 1.000]
```

- 四元数 `[x, y, z, w]` 用来表示 `turtle2` 相对于 `turtle1` 的旋转。
  - `x`，`y`，`z`，`w` 分别是四元数的分量。

```
in RPY (radian) [0.000, -0.000, 0.058]
in RPY (degree) [0.000, -0.000, 3.311]
```

- RPY (绕轴旋转角) 是另一种表示旋转的形式：
  - **弧度 (radian)**：[0.000, -0.000, 0.058] 以数学中标准的弧度单位表示旋转角度。
  - **角度 (degree)**：[0.000, -0.000, 3.311] 以常用的角度单位表示旋转角度。

各个分量代表：

- `Roll (翻滚)`：绕 `x` 轴旋转。
- `Pitch (俯仰)`：绕 `y` 轴旋转。
- `Yaw (偏航)`：绕 `z` 轴旋转。

旋转可以通过以下几种方式描述：

1. **欧拉角 (Euler angles)**：
   - 通过绕 `x、y、z` 三个轴的旋转角度来描述。
   - 存在万向节锁死问题，不适合连续计算。
2. **旋转矩阵**：
   - 使用 3×3 的矩阵描述旋转。
   - 存在冗余信息，容易导致数值误差积累。

$$\boldsymbol{R}(x,\theta)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{bmatrix}$$

$$\boldsymbol{R}(y,\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}$$

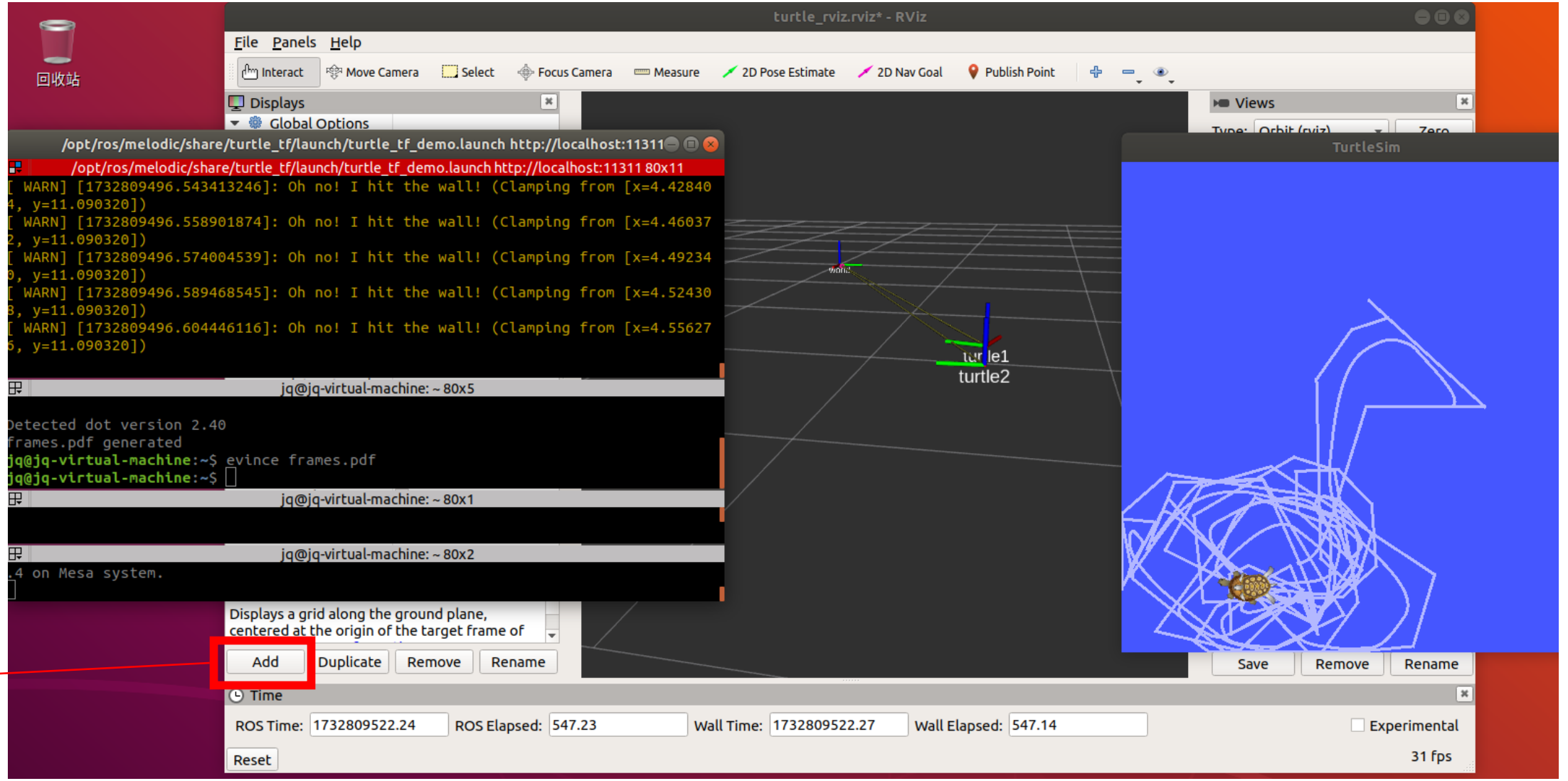
$$\boldsymbol{R}(z,\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

3. **四元数 (Quaternion)**：
   - 是一种紧凑的旋转描述方法，避免了欧拉角的奇异性和旋转矩阵的冗余。
   - 四元数由 4 个值组成：`x，y，z，w`。

- rviz是一款三维可视化工具，很好地兼容了各种基于ROS软件框架的机器人平台。
- 在rviz中，可以使用XML对机器人、周围物体等任何实物进行尺寸、质量、位置、材质、关节等属性的描述，并且在界面中呈现出来。同时，rviz还可以通过图形化方式，实时显示机器人传感器的信息、机器人的运动状态、周围环境的变化等。
- 总而言之，rviz可以帮助开发者实现所有可监测信息的图形化显示，开发者也可以在rviz的控制界面下，通过按钮、滑动条、数值等方式控制机器人的行为。

```
rosrun rviz rviz -d `rospack find turtle_tf`/rviz/turtle_rviz.rviz
```

rosrun rviz rviz -d `rospack find turtle_tf`/rviz/turtle_rviz.rviz



```
rosrun rviz rviz -d `rospack find turtle_tf`/rviz/turtle_rviz.rviz
```

- 这里运行的是 `rviz` 包中的 `rviz` 节点（即 RViz 可视化工具）。

`-d`：

- 这是 RViz 的参数，用于指定要加载的配置文件（.rviz 文件）。
- 配置文件定义了 RViz 的视图设置（如显示的坐标系、网格、主题等）。

`` `rospack find turtle_tf` ``：

- 使用反引号执行嵌套命令。
- `rospack find turtle_tf`：查找 `turtle_tf` 包的路径。
- 这会返回 `turtle_tf` 包在工作空间中的绝对路径。

`/rviz/turtle_rviz.rviz`：

- 指定相对于 `turtle_tf` 包路径的配置文件位置。
- 这里指的是 `rviz/turtle_rviz.rviz` 文件，用于配置 RViz 的显示设置。

创建一个新的ROS包 learning_tf

在终端依次执行如下的命令行:

注意替换成自己的

- cd %YOUR_CATKIN_WORKSPACE_HOME%/src
- catkin_create_pkg learning_tf tf roscpp rospy turtlesim
- $ cd %YOUR_CATKIN_WORKSPACE_HOME%/
- $ catkin_make
- $ source ./devel/setup.bash

- tf/Tutorials/Writing a tf broadcaster (Python) - ROS Wiki

创建一个新的ROS包 learning_tf和脚本文件

在终端依次执行如下的命令行：

- roscd learning_tf
  - `roscd` 是 ROS 提供的一个快捷命令，用于快速导航到指定 ROS 包的目录。
  - 在运行 `roscd learning_tf` 之前，确保 `learning_tf` 包已经正确创建并编译。

命令	功能
`cd ..`	返回上一级目录
`cd -`	返回上一次访问的目录
`pwd`	显示当前路径
`cd ~` 或 `cd`	返回用户主目录（`/home/<username>`）
`cd /`	返回根目录（`/`）

- mkdir nodes

  在 ROS 包的目录中运行 `mkdir nodes` 的作用是创建一个名为 `nodes` 的文件夹。

- touch nodes/turtle_tf_broadcaster.py

  通过ROS的TF系统，广播某个 turtle（海龟）在 turtlesim 模拟环境中的位置和朝向（pose），并将其发布到ROS的TF树中。

  - 如果目录 `nodes` 存在，命令会在 `nodes` 目录中创建一个空的 `turtle_tf_broadcaster.py` 文件。

- chmod +x nodes/turtle_tf_broadcaster.py

  为文件赋予执行权限：

  - `chmod` 是改变文件权限的命令。
  - `+x` 表示赋予文件"可执行权限"。
  - 通过此命令后，`turtle_tf_broadcaster.py` 将被标记为一个可执行脚本，可以直接运行，而无需指定解释器。

在turtle_tf_broadcaster.py脚本文件，粘贴如下代码

```python
#!/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()
    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')
    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

- 指定该脚本使用 Python 解释器运行。
- 加载 ROS 包的依赖（仅适用于 ROS 旧版本如 ROS Fuerte 或 Groovy）。
- 在 ROS Indigo 及之后的版本中，这一行可以省略。
- ROS 的 Python 接口库。
- 用于处理坐标变换的 ROS 库。
- 导入 `turtlesim` **的消息类型** 的模块。
- **定义函数** `handle_turtle_pose`
  - 该函数是一个回调函数，用于处理订阅到的乌龟位姿（`Pose`）消息。
  - 每当订阅的 `/turtleX/pose` 话题收到新的 `Pose` 消息时，这个函数会被调用。
  - **参数说明**：
    - `msg`：乌龟的位姿消息（`turtlesim.msg.Pose` 类型），包含乌龟的 `x`、`y` 坐标以及方向 `theta`。
    - `turtlename`：当前乌龟的坐标系名称，例如 `"turtle1"`。
- 创建一个 `TransformBroadcaster` 对象，用于广播坐标系之间的变换。
- 在 ROS 中，`tf` 库允许我们定义和广播不同坐标系之间的相对位置关系（平移 + 旋转）。
- 这里，`br` 将用来广播从 `"world"` 坐标系到 `turtlename`（如 `"turtle1"`）坐标系的变换。

```
 9 def handle_turtle_pose(msg, turtlename):
10     br = tf.TransformBroadcaster()
11     br.sendTransform((msg.x, msg.y, 0),
12                      tf.transformations.quaternion_from_euler(0, 0, msg.theta),
13                      rospy.Time.now(),
14                      turtlename,
15                      "world")
```

- 第一个参数 `(msg.x, msg.y, 0)`：
  - 表示乌龟的位置，其中 `msg.x` 和 `msg.y` 是乌龟的平面坐标，`0` 是 Z 坐标（假设在平面上为 0）。
- `tf.transformations.quaternion_from_euler` 将欧拉角（roll、pitch、yaw）转换为四元数。
- 输入 `(0, 0, msg.theta)`：
  - 表示绕 Z 轴旋转 `msg.theta`（乌龟在平面上的旋转角度）。
- 四元数是用于表示三维旋转的标准格式。

调用 sendTransform 方法广播坐标变换

- **功能**：
  - 接收乌龟的位置信息（`x, y, theta`），并通过 TF 广播器将其发布为从 `"world"` 到 `"turtleX"` 的坐标变换。
- **流程**：
  1. 从 `/turtleX/pose` 获取乌龟的位姿。
  2. 使用 TF 广播器将位置信息（`x, y`）和平面方向（`theta`）转换为 TF 格式。
  3. 广播到其他节点，可供其他功能使用。

- `rospy.Time.now()` 获取当前时间。
- 这个时间戳会与发布的变换一起传递，以便其他节点能够同步数据。
- `turtlename` 表示变换的子坐标系名称（即乌龟的名字）。
- 这个名称将在 `tf` 树中标识该坐标系。
- `"world"` 是父坐标系的名称，表示乌龟的坐标系相对于全局坐标系（世界坐标系）。

```
17 if __name__ == '__main__':
18     rospy.init_node('turtle_tf_broadcaster')
19     turtlename = rospy.get_param('~turtle')
20     rospy.Subscriber('/%s/pose' % turtlename,
21                      turtlesim.msg.Pose,
22                      handle_turtle_pose,
23                      turtlename)
24     rospy.spin()
```

主程序入口：

- 确保这段代码只在直接运行该脚本时执行，而不是作为模块被导入时执行。

- `rospy.init_node` 用于初始化一个 ROS 节点。
- 节点名称为 `'turtle_tf_broadcaster'`，表示该节点的功能是广播乌龟的坐标变换。
- 使用 `rospy.get_param` 从 ROS 参数服务器中获取参数。
- `'~turtle'` 表示获取私有参数 `turtle`，这是通过命令行或参数文件传递给节点的参数。
- 返回的值被存储在变量 `turtlename` 中，用于指定乌龟的名字。

- `rospy.Subscriber` 用于订阅 ROS 中的某个话题。
- `'/%s/pose' % turtlename`：
  - 根据乌龟名字动态生成话题名称，例如 `/turtle1/pose`。
  - 该话题提供乌龟的位置信息。

- `turtlesim.msg.Pose`：
  - 订阅的消息类型是 `Pose`，定义了乌龟的位置和方向。
- `handle_turtle_pose`：
  - 回调函数，当接收到消息时触发，用于处理位置信息。
- `turtlename`：
  - 传递额外参数 `turtlename` 给回调函数，标识是哪只乌龟。

创建launch文件夹并添加launch文件

在终端依次执行如下的命令行:

- roscd learning_tf
  - `roscd` 是 ROS 提供的一个快捷命令，用于快速导航到指定 ROS 包的目录。
  - 在运行 `roscd learning_tf` 之前，确保 `learning_tf` 包已经正确创建并编译。

- mkdir launch

- touch launch/start_demo.launch

命令	功能
`cd ..`	返回上一级目录
`cd -`	返回上一次访问的目录
`pwd`	显示当前路径
`cd ~` 或 `cd`	返回用户主目录（`/home/<username>`）
`cd /`	返回根目录（`/`）

```
<node pkg="turtlesim" type="turtlesim_node" name="sim"/>
<node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
```

启动 Turtlesim 模拟节点：

- `pkg="turtlesim"`：指定节点所在的 ROS 包为 `turtlesim`。
- `type="turtlesim_node"`：指定要启动的节点类型。
- `name="sim"`：为该节点命名为 `sim`。
- `output="screen"`：
  - 节点的输出会打印到终端屏幕。

启动键盘控制节点：

- `pkg="turtlesim"`：同样来自 `turtlesim` 包。
- `type="turtle_teleop_key"`：节点类型，用于键盘控制乌龟的移动。
- `name="teleop"`：命名为 `teleop`。
- `output="screen"`：将节点的输出信息显示在终端屏幕上。

```
<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>
```

启动第一个乌龟的 TF 广播器：

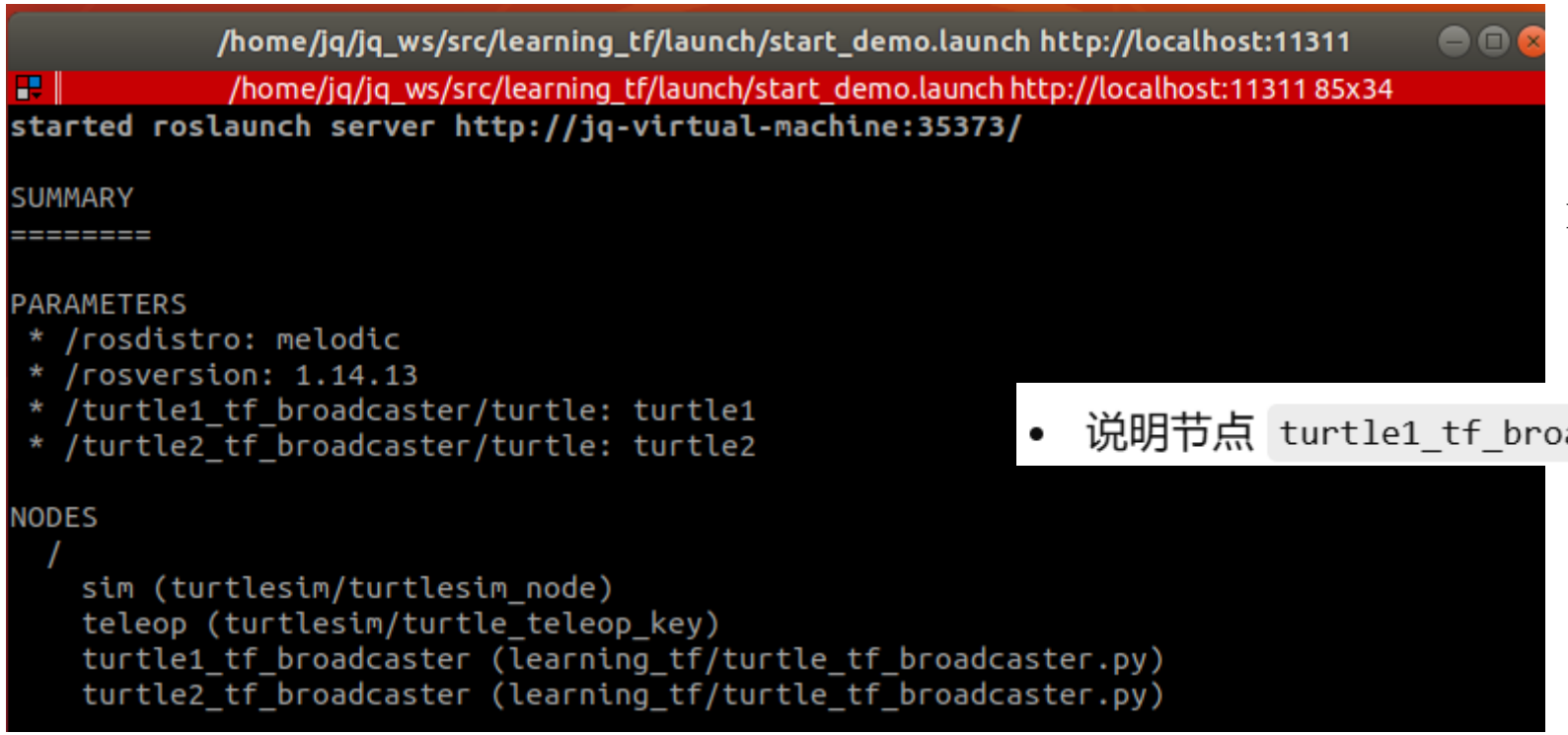
- `name="turtle1_tf_broadcaster"`：节点名称为 `turtle1_tf_broadcaster`。
- `pkg="learning_tf"`：来自 `learning_tf` 包。
- `type="turtle_tf_broadcaster.py"`：要运行的 Python 脚本。
- `respawn="false"`：如果节点崩溃，不会自动重启。
- `output="screen"`：节点输出信息显示在终端屏幕上。
- `<param name="turtle" type="string" value="turtle1" />`：
  - 为节点设置参数，定义了广播器对应的乌龟名称为 `turtle1`。

1. 节点名称（`name="turtle1_tf_broadcaster"`）

- **定义作用**：
  - 它定义了 ROS 系统中该节点的全局名称。
  - 用于标识和区分不同的节点，在 ROS 网络中必须唯一。
  - 例如，在这段代码中，`turtle1_tf_broadcaster` 是该节点的唯一标识符。
- **作用范围**：
  - 在整个 ROS 网络中，这个名称用来标记这个特定的广播器节点。
- **交互性**：
  - 其他节点可以通过该名称与它通信（例如，订阅其话题或服务）。
  - 在 ROS 工具（如 `rqt_graph` 或 `rosnode list`）中，它会以此名称出现。

2. 节点参数 (`<param name="turtle" type="string" value="turtle1" />`)

- **定义作用**:
  - 参数是节点的内部变量，用于传递配置信息给节点。
  - 在这里，`param` 定义了一个名为 `"turtle"` 的参数，其值为 `"turtle1"`，表示这个节点对应于 `turtle1` 这个乌龟。
- **作用范围**:
  - 参数是节点的私有配置，仅供该节点内部使用。
  - 参数的值（如 `turtle1`）被节点脚本读取，影响节点的具体行为。
- **交互性**:
  - 参数值不会影响节点的名称，但它会影响节点脚本的逻辑。例如:
    - 在 `turtle_tf_broadcaster.py` 中，脚本会使用 `turtle` 参数来确定广播器绑定的乌龟名称。
    - 如果参数是 `turtle1`，广播的 TF 信息会绑定到 `/turtle1/pose`。



roslaunch learning_tf start_demo.launch

- 说明节点 `turtle1_tf_broadcaster` 的参数 `turtle` 设置为 `turtle1`。

- `sim`：
  - 包 `turtlesim` 的节点 `turtlesim_node`，这是模拟乌龟运行的核心节点。
- `teleop`：
  - 包 `turtlesim` 的节点 `turtle_teleop_key`，用于通过键盘控制乌龟移动。
- `turtle1_tf_broadcaster` **和** `turtle2_tf_broadcaster`：
  - 包 `learning_tf` 的两个节点，运行的是 Python 脚本 `turtle_tf_broadcaster.py`，用于广播两个乌龟的坐标变换。

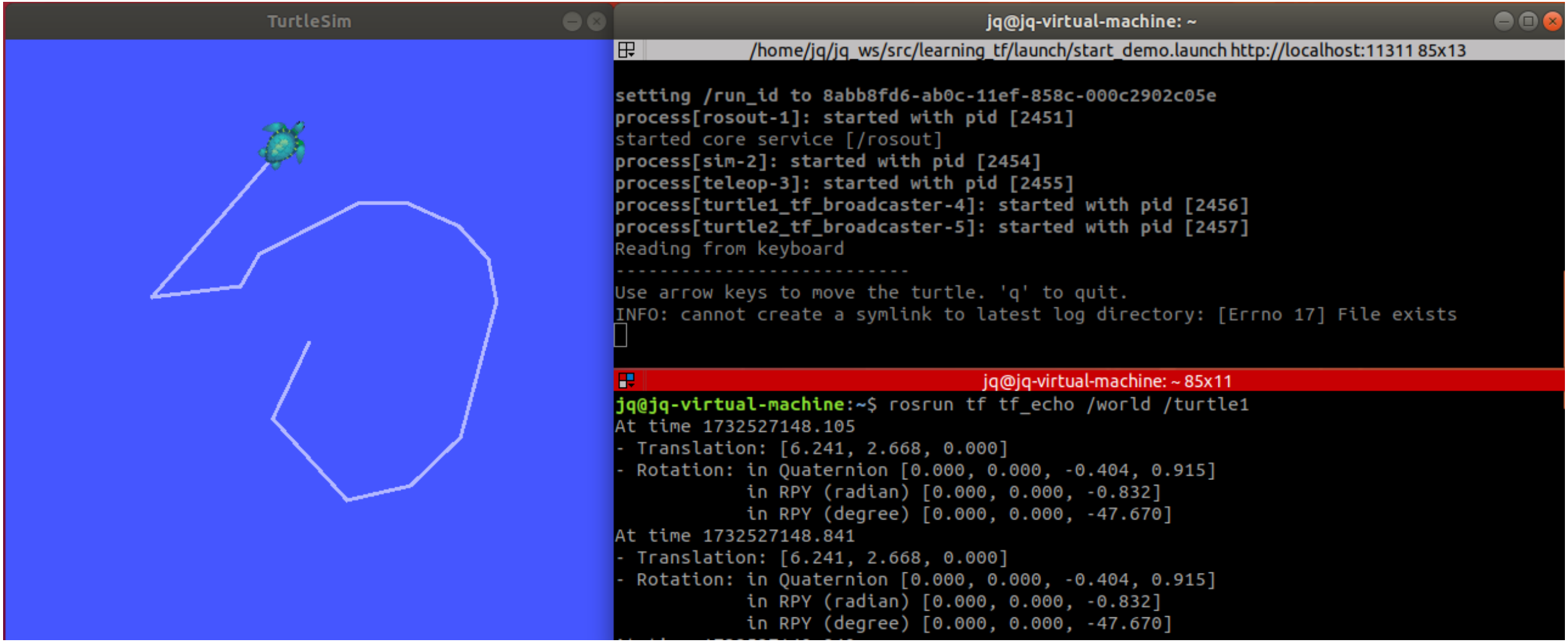
```
auto-starting new master
process[master]: started with pid [2440]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 8abb8fd6-ab0c-11ef-858c-000c2902c05e
process[rosout-1]: started with pid [2451]
started core service [/rosout]
process[sim-2]: started with pid [2454]
process[teleop-3]: started with pid [2455]
process[turtle1_tf_broadcaster-4]: started with pid [2456]
process[turtle2_tf_broadcaster-5]: started with pid [2457]
Reading from keyboard
---------------------------
Use arrow keys to move the turtle. 'q' to quit.
INFO: cannot create a symlink to latest log directory: [Errno 17] File exists
```

- `ROS_MASTER_URI=http://localhost:11311`：说明当前运行的 ROS 主节点（Master）地址是本地 `localhost`，端口号是 `11311`。

- 每个节点作为独立的进程启动，包含以下信息：
  - `process[master]: started with pid [2440]`：ROS Master 主节点启动。
  - `process[rosout-1]: started with pid [2451]`：`rosout` 节点启动，用于记录日志。
  - 其他节点（`sim`、`teleop`、`turtle1_tf_broadcaster`、`turtle2_tf_broadcaster`）依次启动，显示对应的进程 ID（`pid`）。

rosrun tf tf_echo /world /turtle1



创建turtle_tf_listener.py脚本文件

在终端依次执行如下的命令行:

- roscd learning_tf
- touch nodes/turtle_tf_listener.py
- chmod +x nodes/turtle_tf_listener.py
- tf/Tutorials/Writing a tf listener (Python) - ROS Wiki

监听 turtle2 相对于 turtle1 的变换，
并根据这个变换控制 turtle2 的运动，
使其朝着 turtle1 进行运动，
最终实现 turtle2 追踪 turtle1 的行为。

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

- 导入 Python 的内置数学运算库 `math`。
- 提供常用数学函数，例如：
  - `math.atan2()`：计算两点间的偏角。
  - `math.sqrt()`：计算平方根。
- 导入 ROS 的几何消息模块 `geometry_msgs.msg`。
- 包含常用的消息类型，例如：
  - `Twist`：用于表示速度指令（线速度和角速度）。
  - `Pose`：用于表示位置和方向。
- 导入 ROS 的 TF 库，这是 ROS 提供的一个坐标变换工具。
- 提供监听（`TransformListener`）和广播（`TransformBroadcaster`）坐标变换的功能。
- 导入 `turtlesim` 包中的服务类型模块 `turtlesim.srv`。
- 包含 Turtlesim 的服务类型，例如：
  - `Spawn`：在指定位置生成一只新的乌龟。

```
10 if __name__ == '__main__':
11     rospy.init_node('turtle_tf_listener')
12 
13     listener = tf.TransformListener()
```

- 使用 `rospy.init_node` 初始化一个 ROS 节点，节点名为 `'turtle_tf_listener'`。
- 在 ROS 系统中，每个运行的程序需要注册为一个节点，才能参与 ROS 的通信网络。

- 创建一个 `tf.TransformListener` 实例，用于监听和查询坐标变换。
- `TransformListener` 会订阅 ROS 的 TF 数据流，记录所有发布的坐标变换信息。

```
15      rospy.wait_for_service('spawn')
16      spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
17      spawner(4, 2, 0, 'turtle2')
```

- 使用 `rospy.wait_for_service()` 等待 `spawn` 服务的可用性。
- 在 ROS 中，`spawn` 是 Turtlesim 提供的一个服务，用于在模拟环境中生成新的乌龟。
- 使用 `rospy.ServiceProxy` 创建一个服务代理对象 `spawner`。
- 服务代理是一个客户端接口，允许脚本向服务发送请求并接收响应。
- `'spawn'`：服务名称，表示目标服务是 Turtlesim 提供的 `spawn` 服务。
- `turtlesim.srv.Spawn`：服务的类型，定义了请求和响应的格式。
- 调用 `spawner` 服务代理，向 `spawn` 服务发送请求，要求生成一只新的乌龟。
    - `4`：生成乌龟的 x 坐标为 `4`。
    - `2`：生成乌龟的 y 坐标为 `2`。
    - `0`：生成乌龟的初始方向（弧度）。
    - `'turtle2'`：新生成的乌龟名称为 `turtle2`。

```
19      turtle_vel = rospy.Publisher('turtle2/cmd_vel', geometry_msgs.msg.Twist,queue_size=1)
20
21      rate = rospy.Rate(10.0)
```

- 创建一个 ROS 频率控制器（`Rate`），用于控制主循环的运行频率。

- 创建一个 ROS 发布器（`Publisher`），用于向 `turtle2/cmd_vel` 话题发布速度控制命令。

1. **话题名称**：
   - `'turtle2/cmd_vel'`：
     - `turtle2` 是乌龟的名称，`cmd_vel` 是该乌龟的速度控制话题。
     - 通过这个话题，可以控制 `turtle2` 的线速度和角速度。
2. **消息类型**：
   - `geometry_msgs.msg.Twist`：
     - 这是 ROS 中用于描述速度的标准消息类型。
     - 包含两个部分：
       - **线速度（linear）**：沿 `x`，`y`，`z` 轴的线速度。
       - **角速度（angular）**：绕 `x`，`y`，`z` 轴的角速度。
3. **队列大小**：
   - `queue_size=1`：
     - 限制发布消息队列的长度为 1。
     - 如果消息堆积（比如处理不及时），旧消息会被丢弃，确保只处理最新的控制指令。

```
22      while not rospy.is_shutdown():
23          try:
24              (trans,rot) = listener.lookupTransform('/turtle2', '/turtle1', rospy.Time(0))
25          except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
26              continue
```

- 使用 TF 的 `lookupTransform` 方法，查询 `/turtle2` 坐标系相对于 `/turtle1` 坐标系的变换。
- 返回两个值：
  1. `trans`：平移向量 `[x, y, z]`，表示两者的位置关系。
  2. `rot`：旋转四元数 `[x, y, z, w]`，表示两者的方向关系。

`rospy.Time(0)`：

- 表示查询最新的可用变换。
- 如果坐标变换是时间同步的，传入特定时间戳可以获取历史变换。

可能的异常

1. `tf.LookupException`：
   - 查询的坐标变换不存在（例如，未定义 `/turtle1` 或 `/turtle2`）。
2. `tf.ConnectivityException`：
   - 坐标变换之间的连接中断，无法获取变换。
3. `tf.ExtrapolationException`：
   - 时间戳超出缓存范围，无法获取变换（例如，请求的时间过于陈旧或过于未来）。

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

- 根据 `turtle1` 和 `turtle2` 的相对位置（`trans`）计算角速度。
- 确保 `turtle2` 的头部朝向 `turtle1` 的位置。

1. `math.atan2(y, x)`：
    - 计算两点之间的角度，结果是 `turtle2` 需要朝向 `turtle1` 的旋转角度。
    - `trans[1]` 是 `turtle1` 在 `turtle2` 坐标系下的 `y` 坐标。
    - `trans[0]` 是 `turtle1` 在 `turtle2` 坐标系下的 `x` 坐标。
2. **系数** `4`：
    - 增强角速度响应，控制旋转速度。

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

- 根据 `turtle1` 和 `turtle2` 的欧几里得距离（直线距离）计算线速度。
- 让 `turtle2` 向 `turtle1` 的方向移动，速度与距离成正比。

1. `math.sqrt(x**2 + y**2)`：
    - 计算两点之间的距离，即 `turtle1` 和 `turtle2` 的距离。
    - `trans[0]` 是 `x` 方向的距离，`trans[1]` 是 `y` 方向的距离。
2. **系数** `0.5`：
    - 限制线速度，避免 `turtle2` 移动过快。

```
30          cmd = geometry_msgs.msg.Twist()
```

- 创建一个 `Twist` 类型的消息对象 `cmd`。
- 为后续设置线速度和角速度的值提供对象。
- `Twist` 是 ROS 中用于描述移动速度的消息类型。

`Twist` 的结构

- 包含两个部分：

1. **线速度（linear）**：
   - `cmd.linear.x`：沿 `x` 轴的速度（前进或后退）。
   - `cmd.linear.y` 和 `cmd.linear.z`：沿 `y` 和 `z` 轴的速度，通常为 0（在平面运动中无需设置）。
2. **角速度（angular）**：
   - `cmd.angular.x` 和 `cmd.angular.y`：绕 `x` 和 `y` 轴的旋转速度，通常为 0。
   - `cmd.angular.z`：绕 `z` 轴的旋转速度（左右旋转）。

```
31          cmd.linear.x = linear
32          cmd.angular.z = angular
33          turtle_vel.publish(cmd)
34
35          rate.sleep()
```

- 将计算出的线速度 `linear` 赋值给 `cmd.linear.x`。
  - 设置乌龟沿 `x` 轴的前进速度，使其向目标移动。
- 将计算出的角速度 `angular` 赋值给 `cmd.angular.z`。
  - 设置乌龟绕 `z` 轴的旋转速度，使其朝向目标方向。
- 将设置好的速度控制消息 `cmd` 发布到话题 `turtle2/cmd_vel`。
  - 控制 Turtlesim 中的 `turtle2` 按照指定的速度移动。
  - 速度消息被 Turtlesim 的内置机制接收并处理，驱动乌龟执行移动。
- 调用 `rate.sleep()` 方法，按照之前设置的频率（例如 10 Hz）进行循环控制。
  - 确保主循环按照固定频率执行，避免占用过多的系统资源。
  - 在下一次迭代前，让程序休眠一段时间。