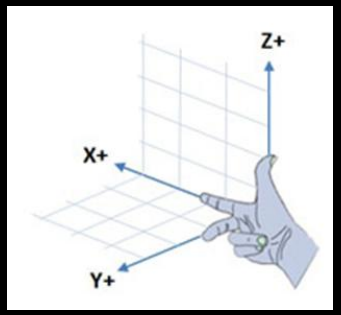
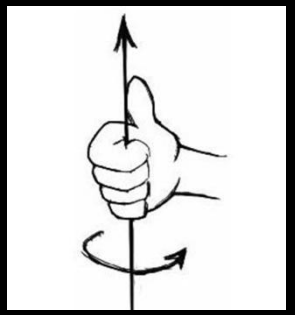
ROS组件

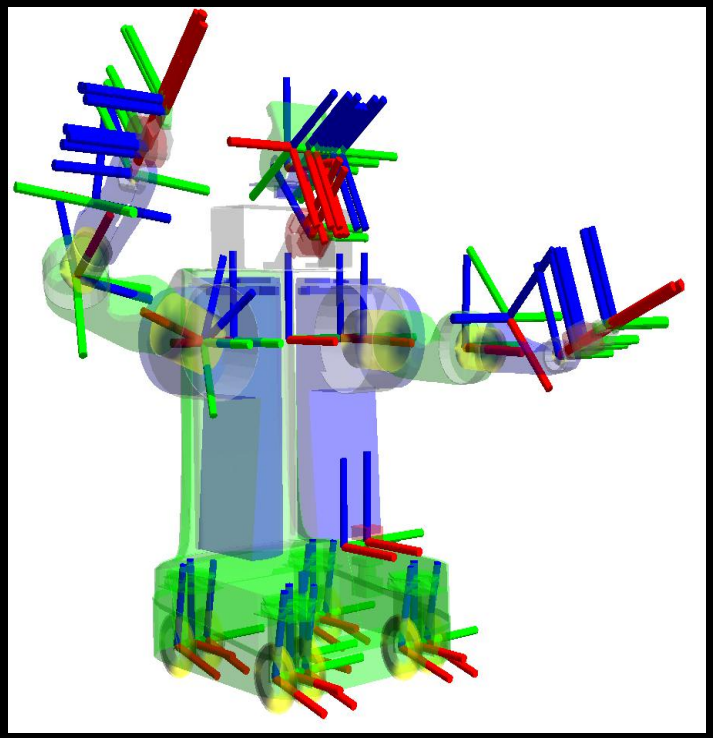
----TF



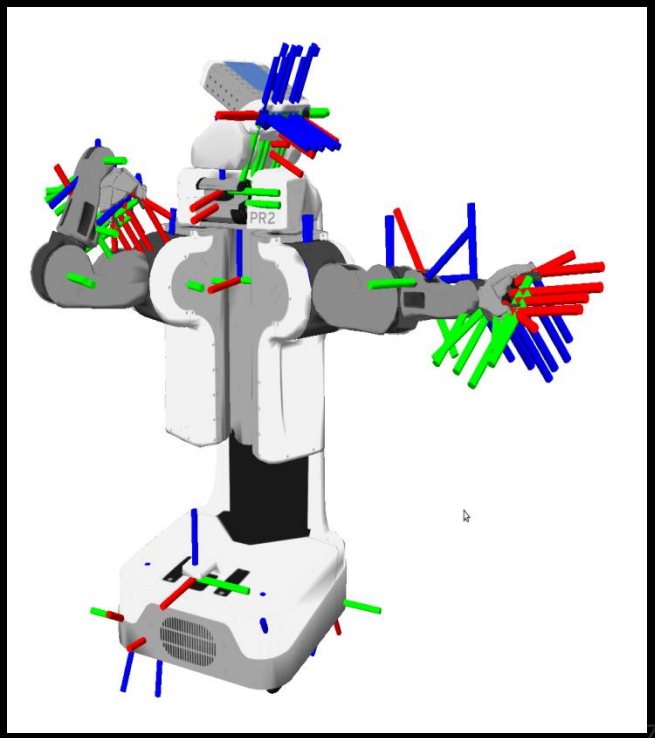


内容概要

- TF概念
- TF应用



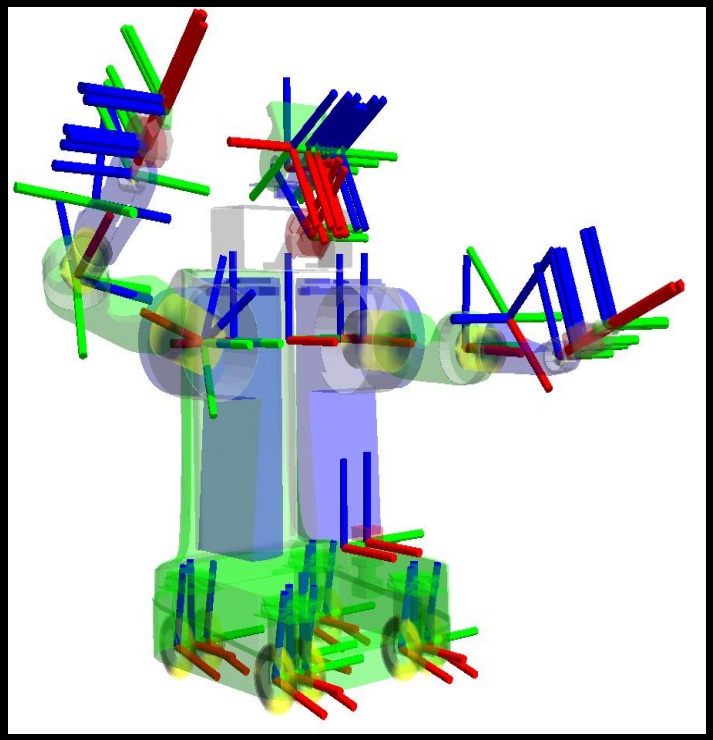
http://wiki.ros.org/tf

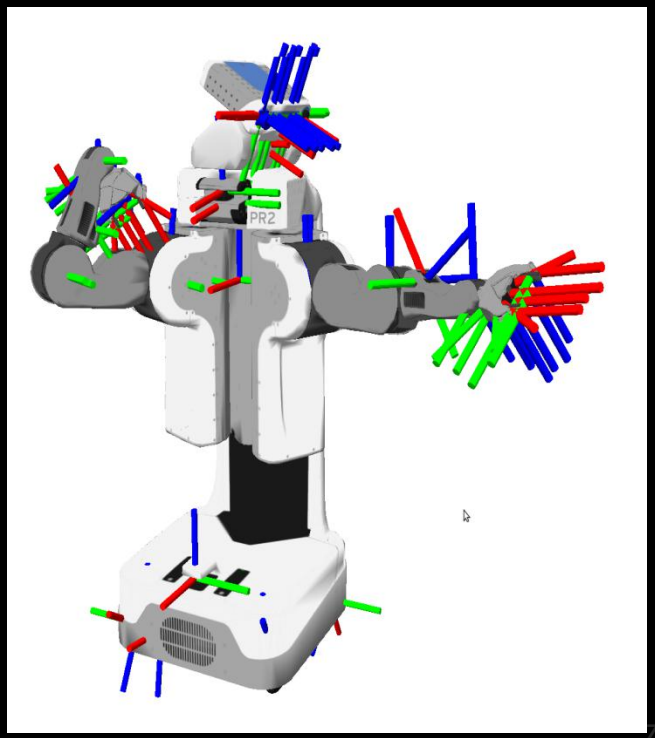


http://wiki.ros.org/tf2

什么是TF

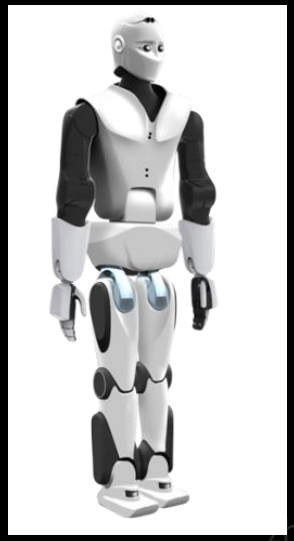
机器人系统通常具有随着时间的推移而改变的许多3D 坐标系，例如大地坐标，基坐标，工具坐标（夹持器），工件坐标（夹持目标）等。tf2随时间跟踪所有这些坐标。





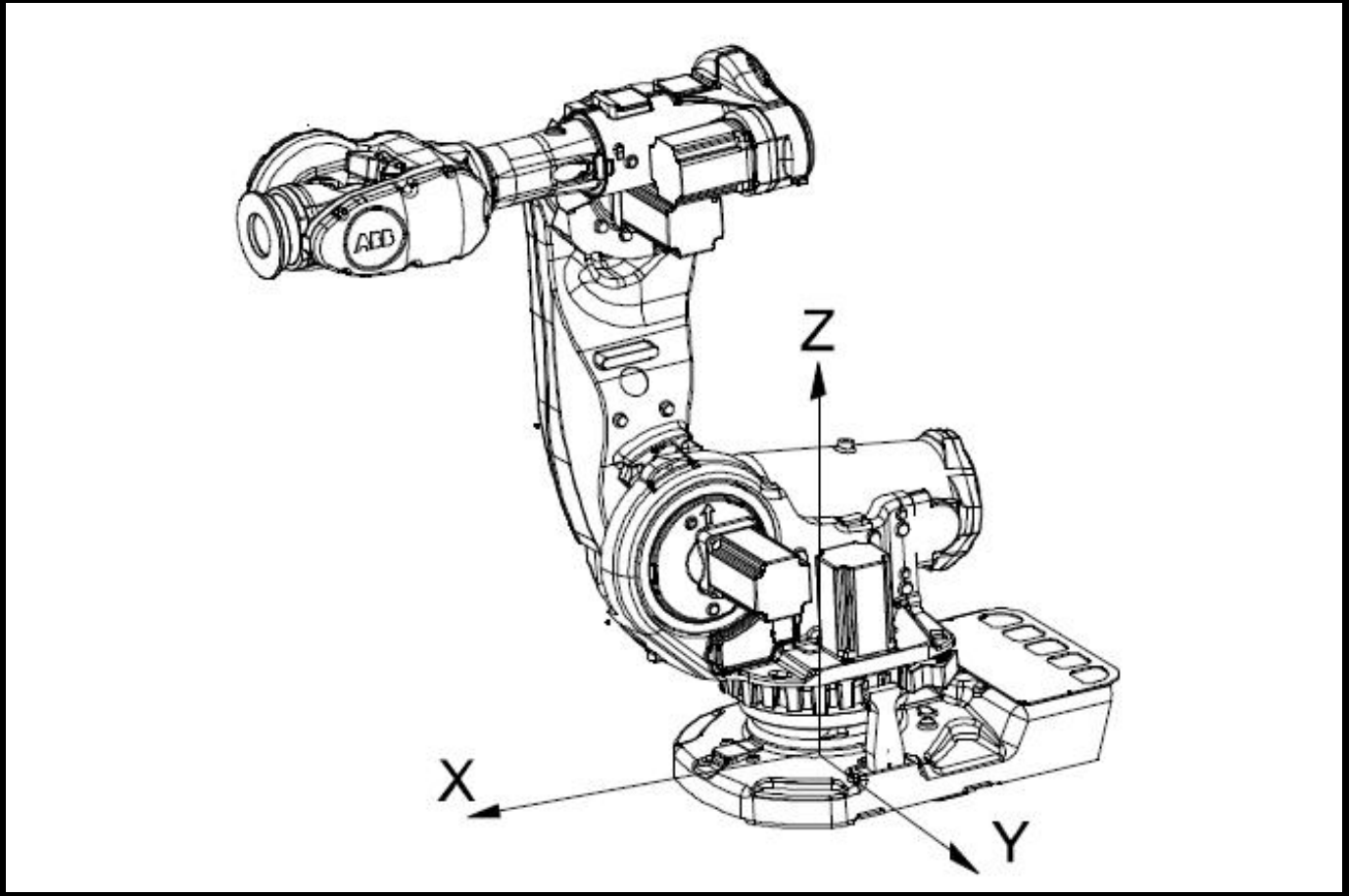
两个小示例

- 工业机器人
- 移动机器人



基坐标系

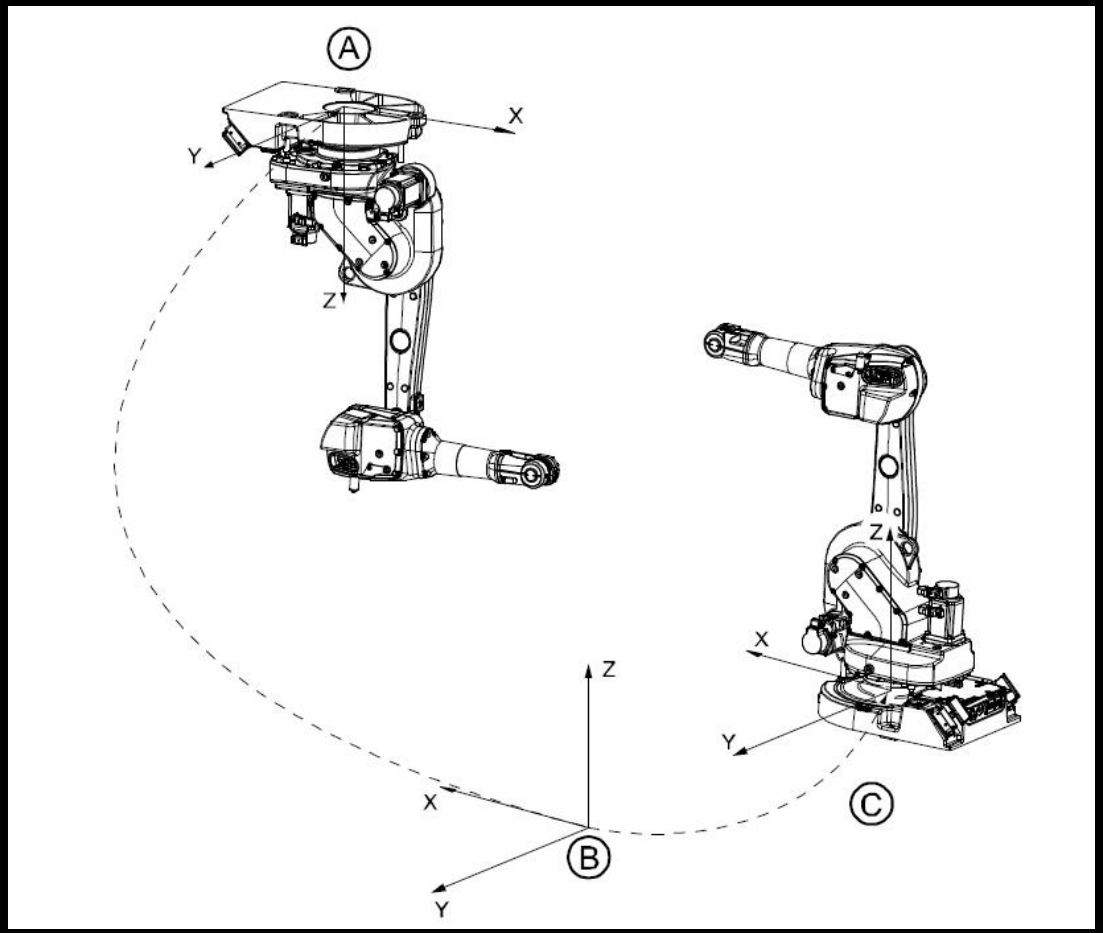
基坐标系在机器人基座中有相应的零点，这使固定安装的机器人的移动具有可预测性。因此它对于将机器人从一个位置移动到另一个位置很有帮助。



ABB工业机器人

大地坐标系

大地坐标系在工作单元或工作站中的固定位置有其相应的零点。这有助于处理若干个机器人或由外轴移动的机器人.在默认情况下，大地坐标系与基坐标系是一致的

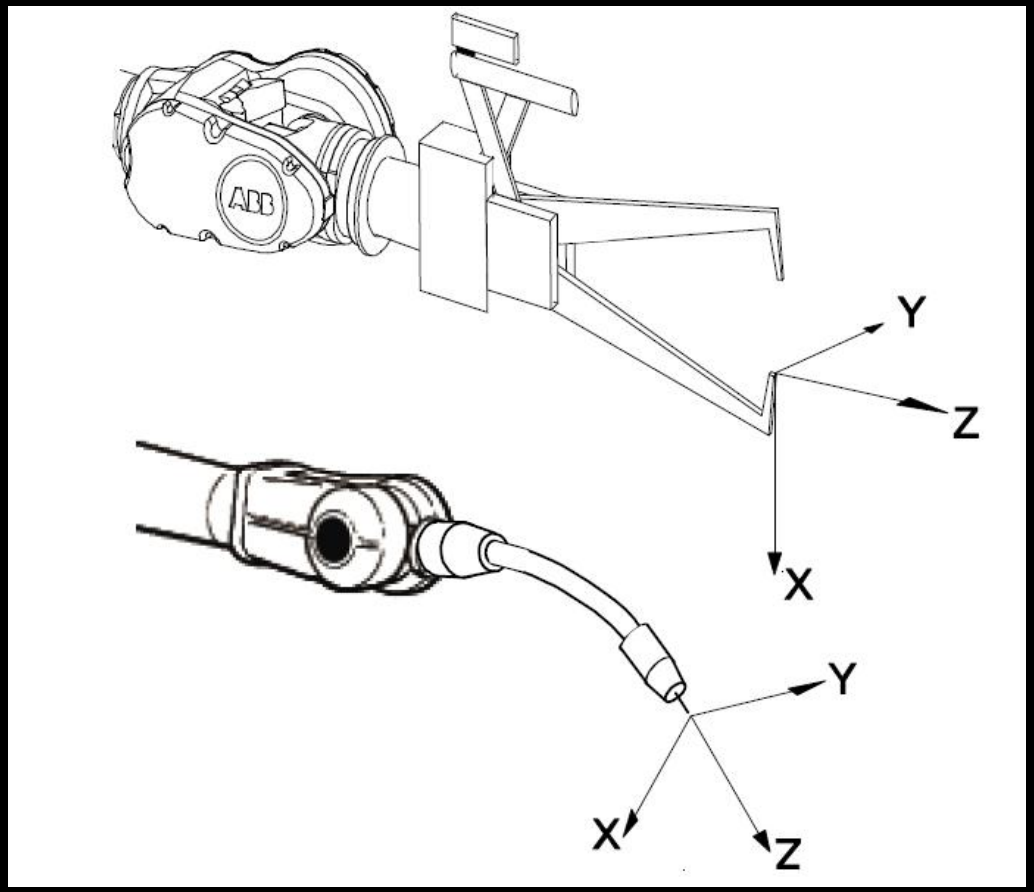


A:机器人1的基坐标系
B:大地坐标系
C:机器人2的基坐标系

ABB工业机器人

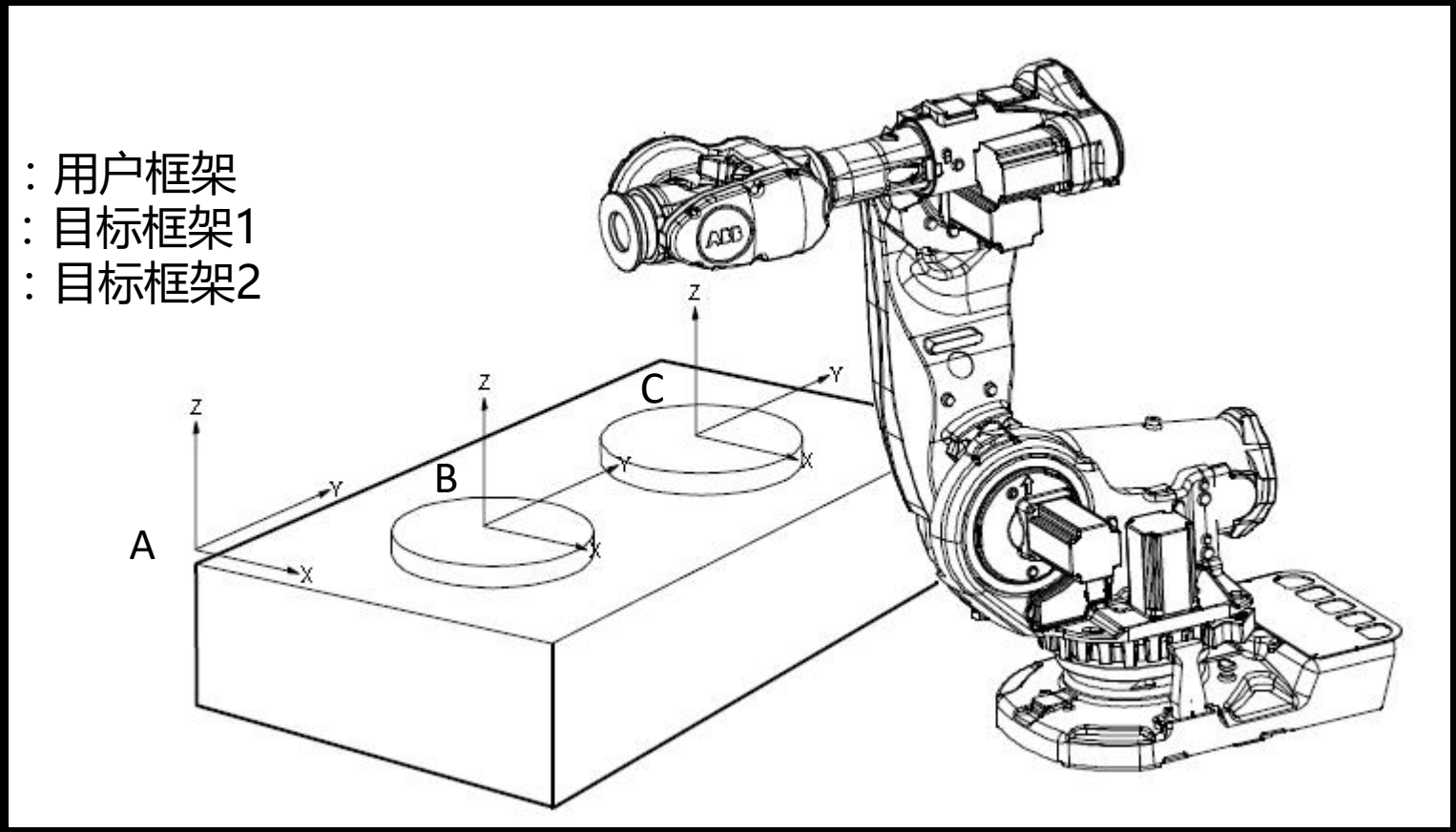
工具坐标系

工具坐标系将工具中心点设为零位，由此定义工具的位置和方向，工具坐标系中心缩写为TCP (Tool Center Point)。



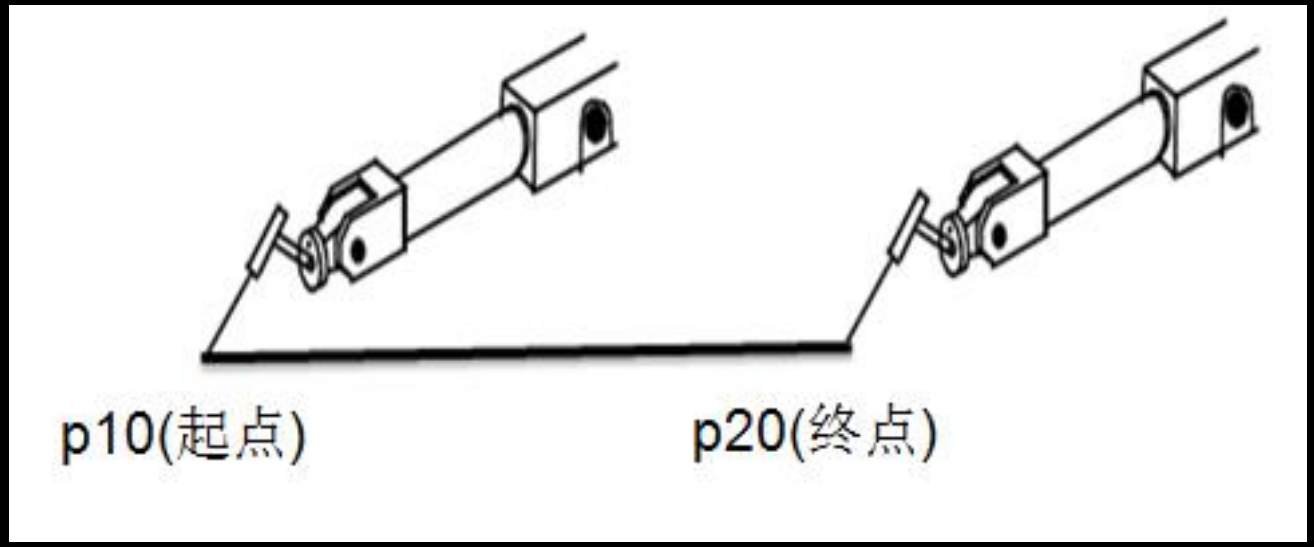
工件坐标系

工件坐标系是拥有特定附加属性的坐标系。它主要用于简化编程，工件坐标系拥有两个框架：用户框架（与大地基座相关）和工件框架（与用户框架相关）。



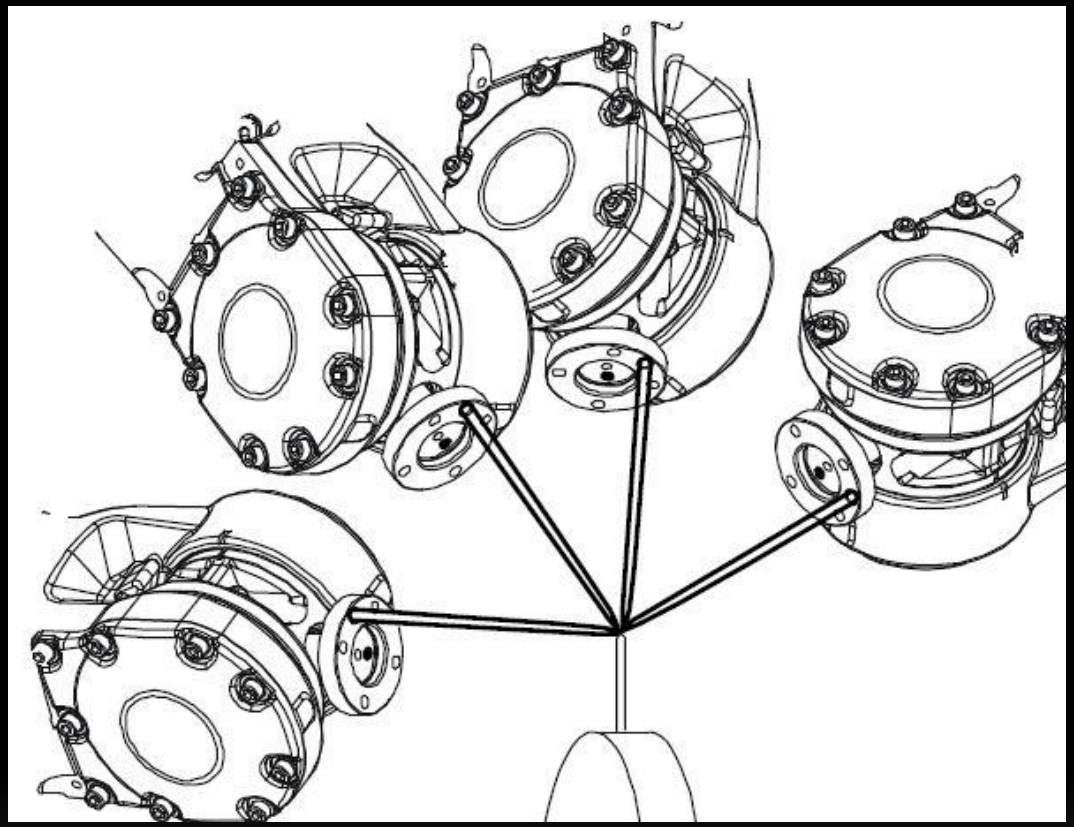
线性运动模式

线性运动即控制机器人TCP沿着指定的参考坐标系的坐标轴方向进行移动，在运动过程中工具的姿态不变，常用于空间范围内移动机器人TCP位置。

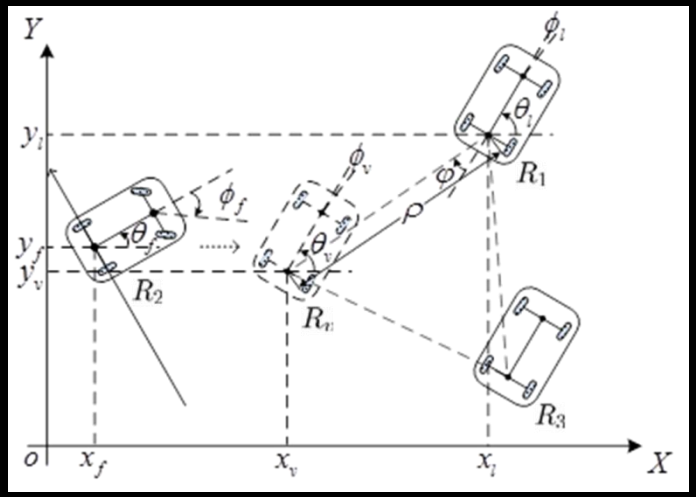


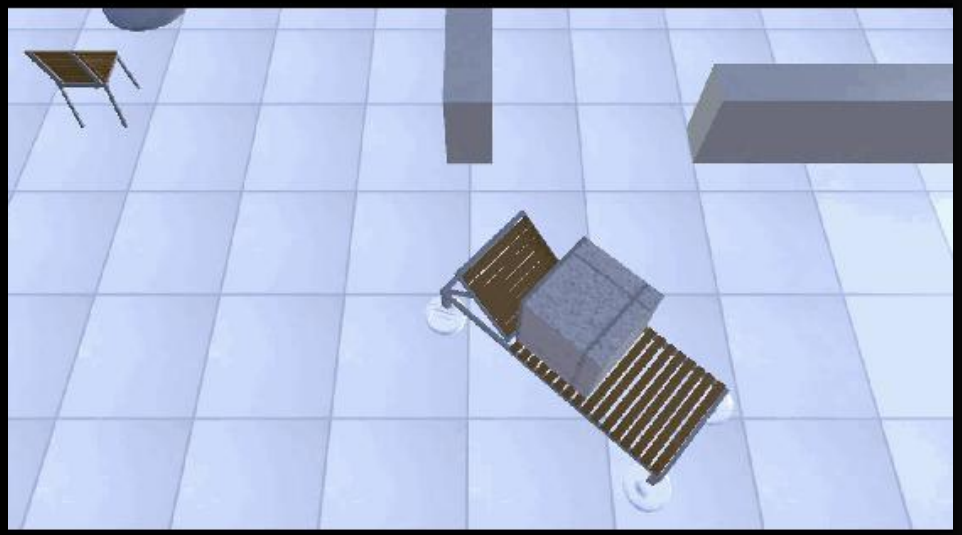
重定位运动模式

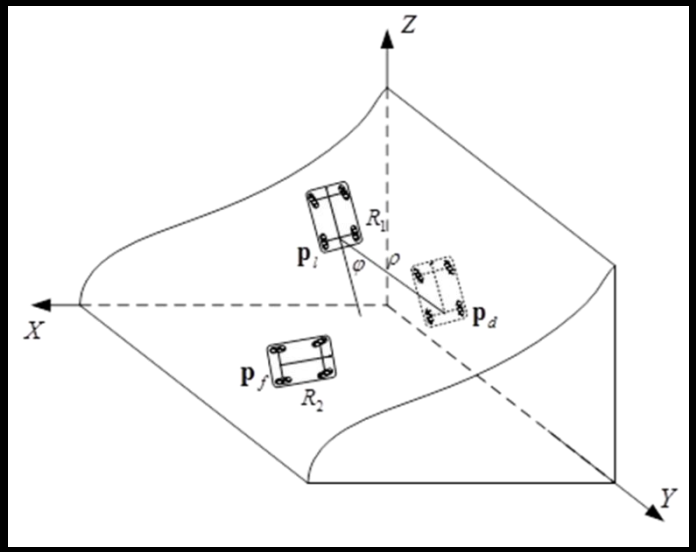
一些特定情况下我们需要重新定位工具方向，使其与工件保持特定的角度，以便获得最佳效果，例如在焊接、切割、铣削等应用。当将工具中心点微调至特定位置后，在大多数情况下需要重新定位工具方向，定位完成后，将继续以线性动作进行微动控制，以完成路径和所需操作。



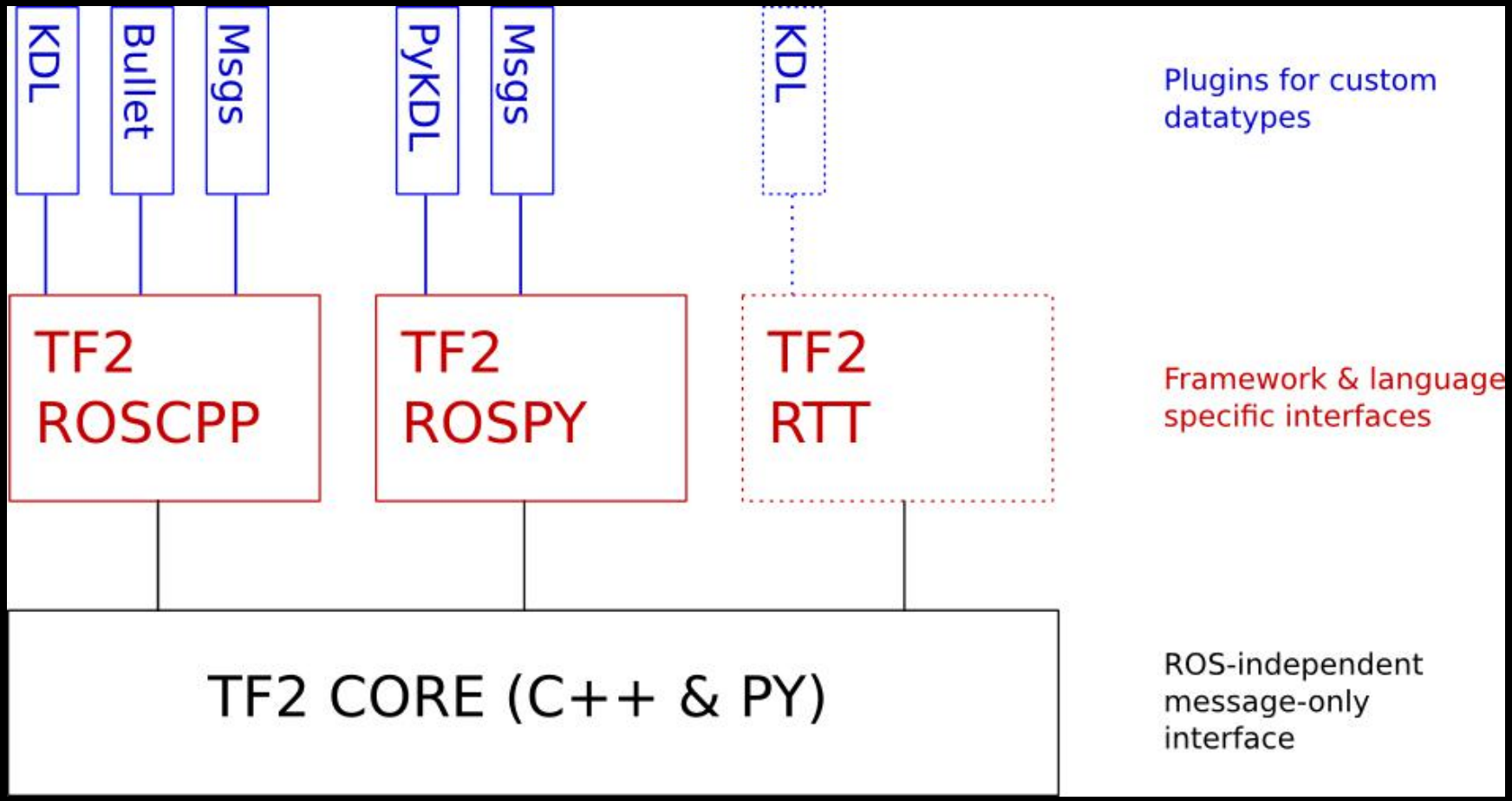
多机器人系统编队


Y
X
o
x_f
x_v
x_l
y_l
y_f
y_v
R_1
R_2
R_3
R_v
ϕ_l
ϕ_f
ϕ_v
θ_l
θ_f
θ_v
ρ




Z
X
Y
$\mathbf{p}_l$
R_1
$\mathbf{p}_d$
$\mathbf{p}_f$
R_2
ρ
α

TF2



TF2

- 增加/tf_static主题，提高效率
- 独立核心，不依赖ROS通信
- 新API接口
- 更好的支持Python，核心库仍是C++
- 响应查询模式
- 不再支持tf_prefix

http://wiki.ros.org/tf2/Migration

思考

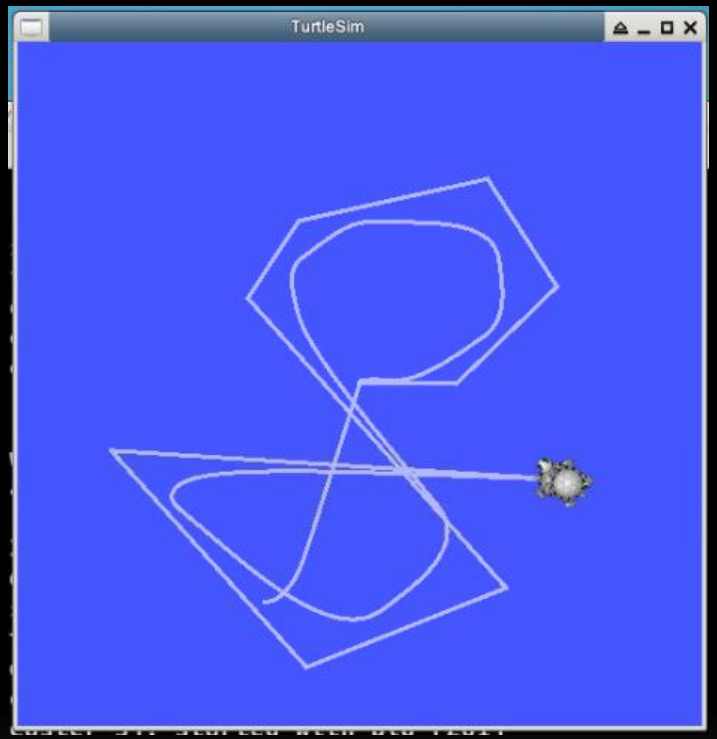
- 1. 如何让一个机器人及时跟随另一个机器人的轨迹？
- 2. 如何让一个机器人及时跟随另一个机器人的轨迹并　且保持 一段距离？静态或动态
- 3.如何让一个机器人跟随另一个机器人前几秒走过的轨迹？

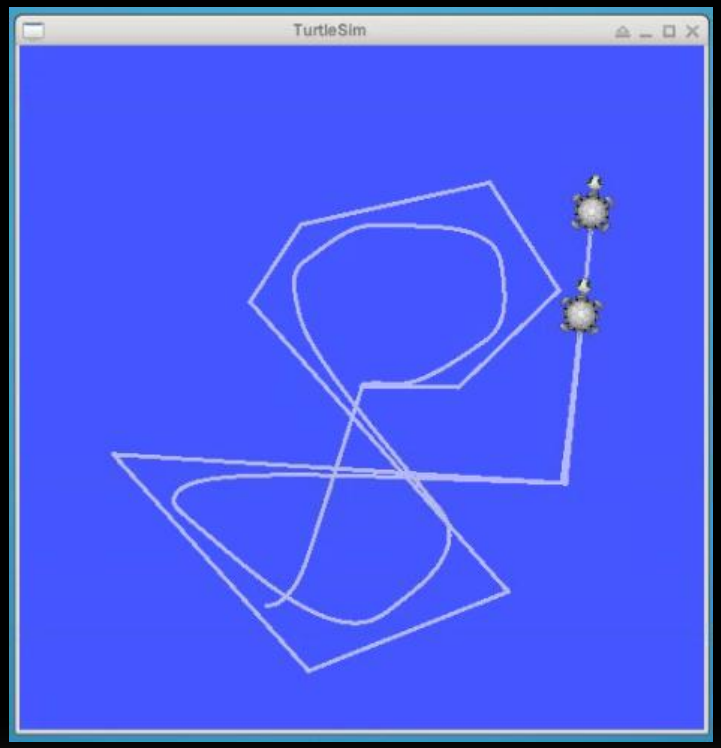
运行示例

在终端输入下面命令：

$ roslaunch turtle_tf2 turtle_tf2_demo.launch

选择当前终端，并通过键盘上下左右键控制运动





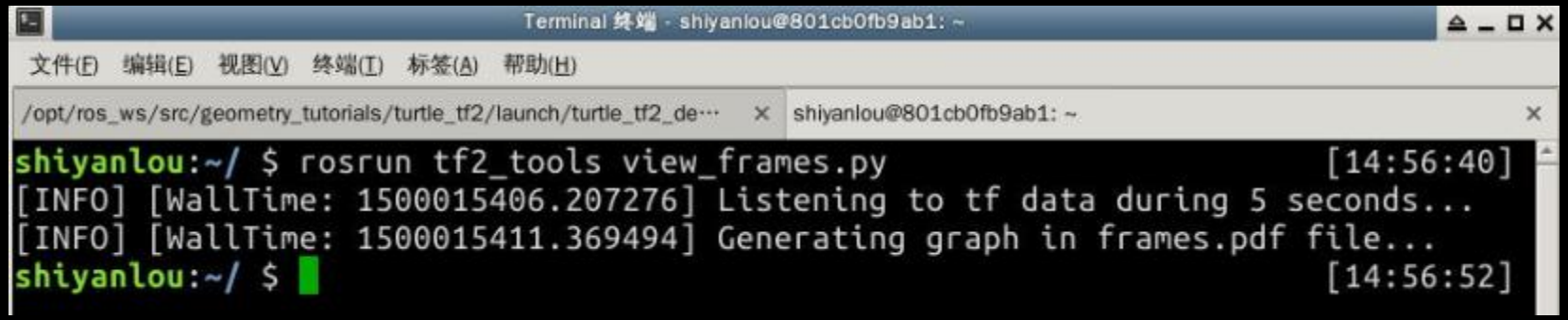
TF2工具

view_frames是一个图形调试工具，可以创建当前tf2变换的PDF图形。

```
$ rosrun tf2_tools view_frames.py
```

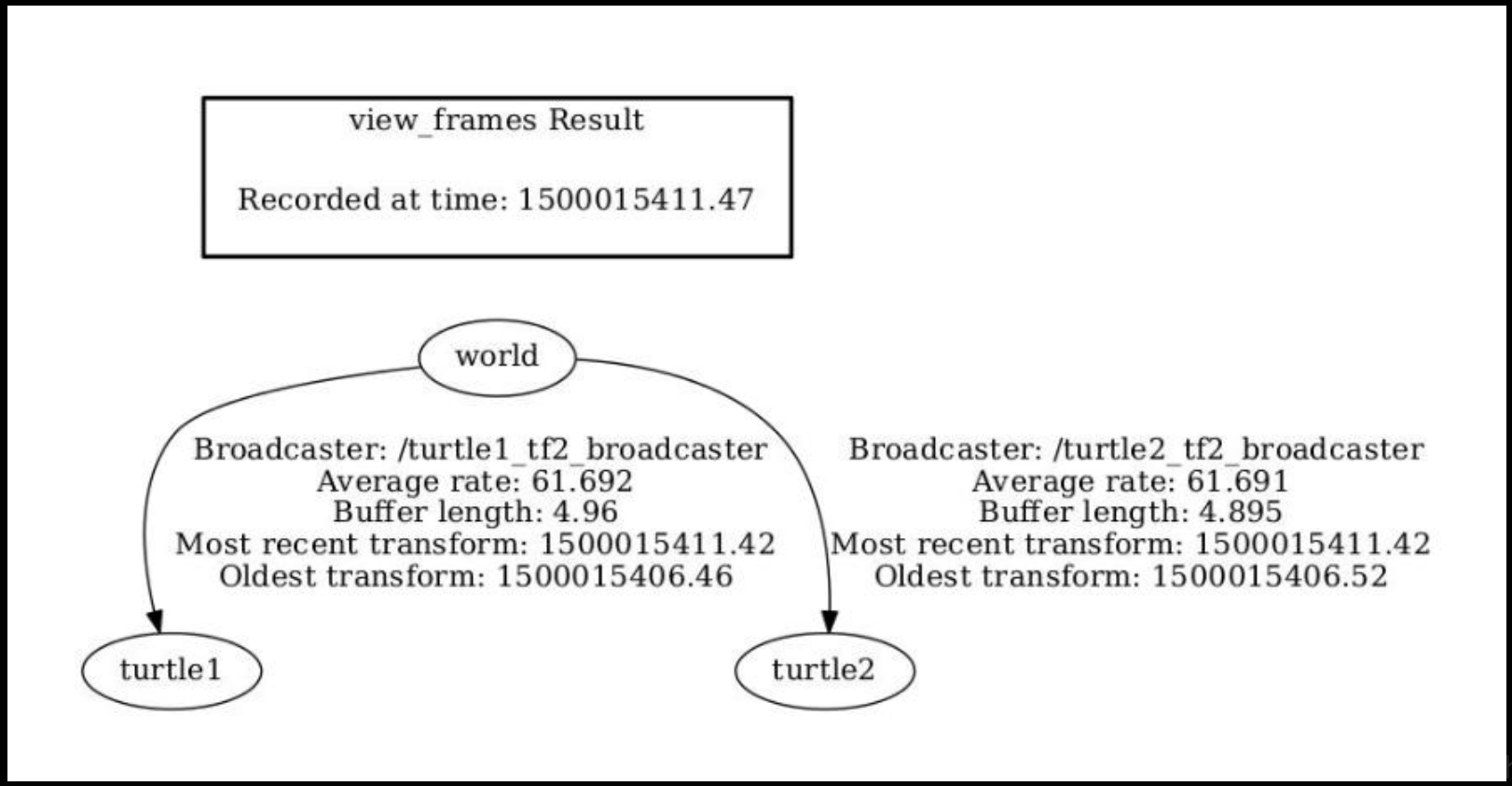
如果需要在完成后查看tf2，Ubuntu系统上的典型用法如下：

```
$ evince frames.pdf
```



TF2工具

三个坐标系：1 世界的坐标系(父节点) 2 Turtle 1的坐标系(子节点1)3 Turtle 2的坐标系(子节点2)



tf_echo

tf_echo生成通过ROS广播的任何两个坐标之间的变换关系。

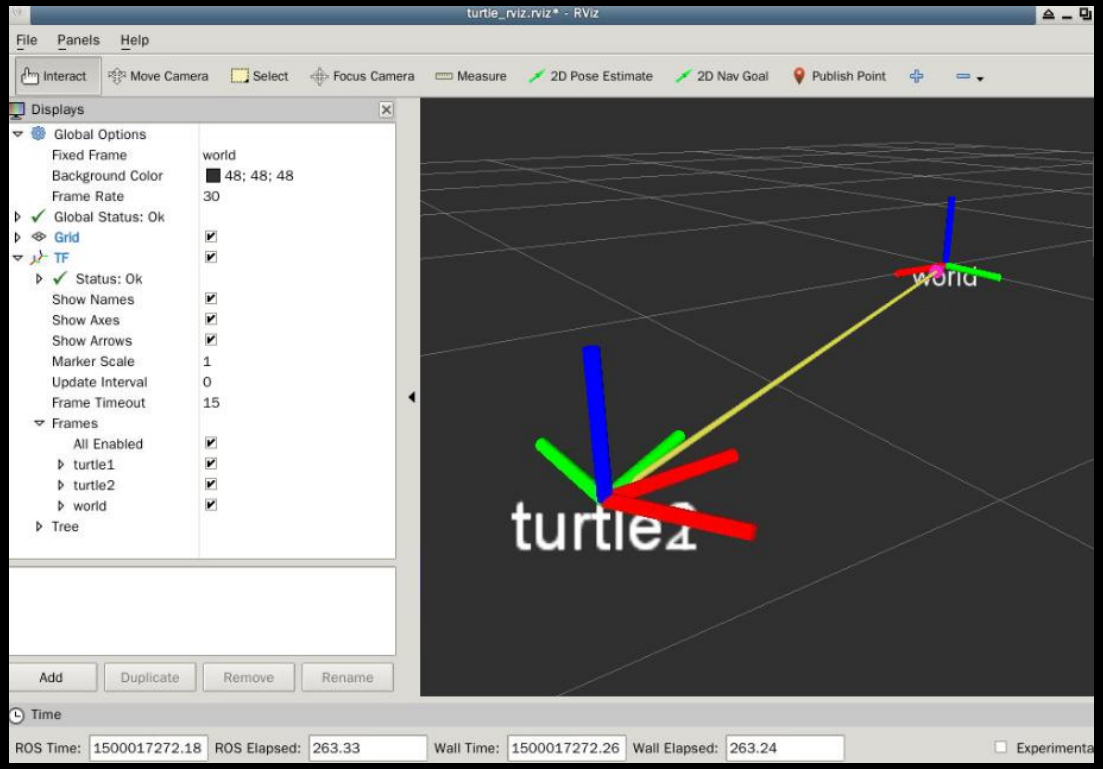
$ rosrun tf tf_echo [reference_frame] [target_frame]

$ rosrun tf tf_echo turtle1 turtle2

```
- Translation: [-0.914, 0.267, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.142, 0.990]
            in RPY (radian) [0.000, 0.000, -0.284]
            in RPY (degree) [0.000, 0.000, -16.277]
At time 1500016179.049
- Translation: [-0.527, 0.154, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.142, 0.990]
            in RPY (radian) [0.000, 0.000, -0.284]
            in RPY (degree) [0.000, 0.000, -16.277]
At time 1500016180.059
- Translation: [-0.317, 0.093, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.142, 0.990]
            in RPY (radian) [0.000, 0.000, -0.284]
            in RPY (degree) [0.000, 0.000, -16.277]
At time 1500016181.051
- Translation: [-0.194, 0.057, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.142, 0.990]
            in RPY (radian) [0.000, 0.000, -0.284]
            in RPY (degree) [0.000, 0.000, -16.277]
At time 1500016182.050
- Translation: [-0.119, 0.035, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.142, 0.990]
            in RPY (radian) [0.000, 0.000, -0.284]
            in RPY (degree) [0.000, 0.000, -16.277]
At time 1500016183.051
```

Rviz与TF2

rviz是一个可视化工具，可用于查看tf2坐标。使用rviz的turtle坐标命令如下：$ rosrun rviz rviz –d `rospack find turtle_tf2` /rviz/turtle_rviz.rviz



http://wiki.ros.org/tf2/Tutorials/Introduction%20to%20tf2

TF2原理解析

- 发布turtle的坐标 （broadcaster）
- 计算turtle发布的坐标 （listener）
- 对比现在坐标的差异
- 并移动turtle去跟随另一只

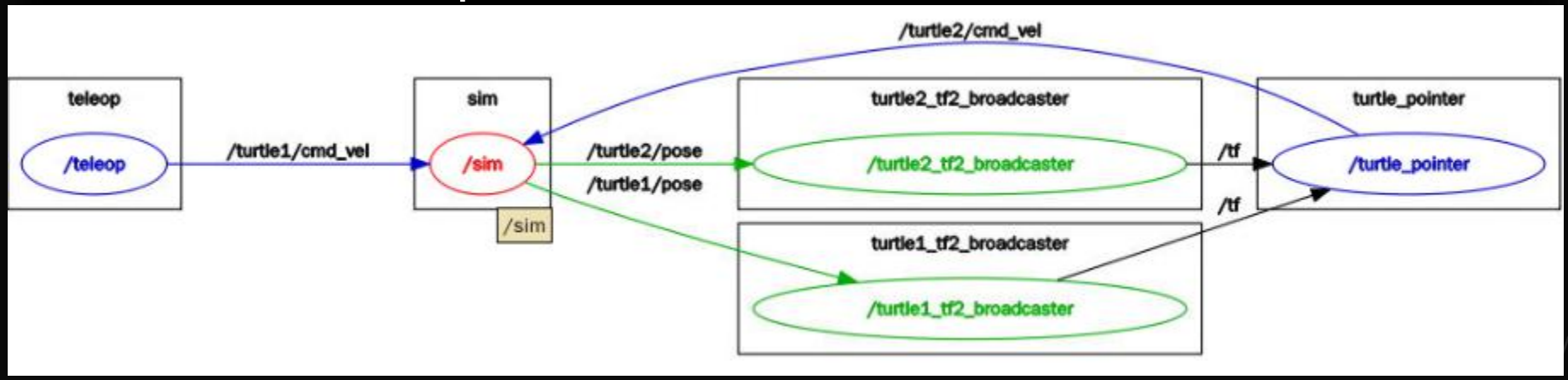
TF2原理解析

launch 启动四个节点

- turtlesim_node（/sim） 能够生成乌龟和发布pose
- teleop（/teleop） 键盘控制
- turtle1_tf2_broadcaster
- turtle2_tf2_broadcaster

通过主题/turtle1/cmd_vel来发布消息

通过主题/turtle*/pose来发布位置改变的消息

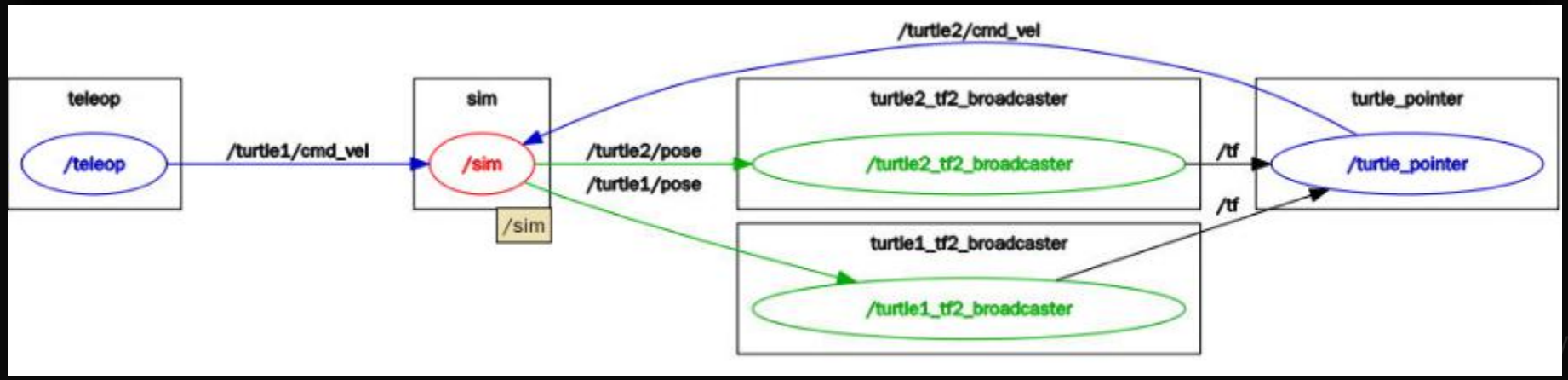


TF2原理解析

主要通过turtle_tf2_broadcaster订阅位置改变消息并通过tf2主题发布

然后listener来订阅tf2主题计算出位置改变消息并将消息通过主题/turtle2/cmd_vel发送给turtuer2。

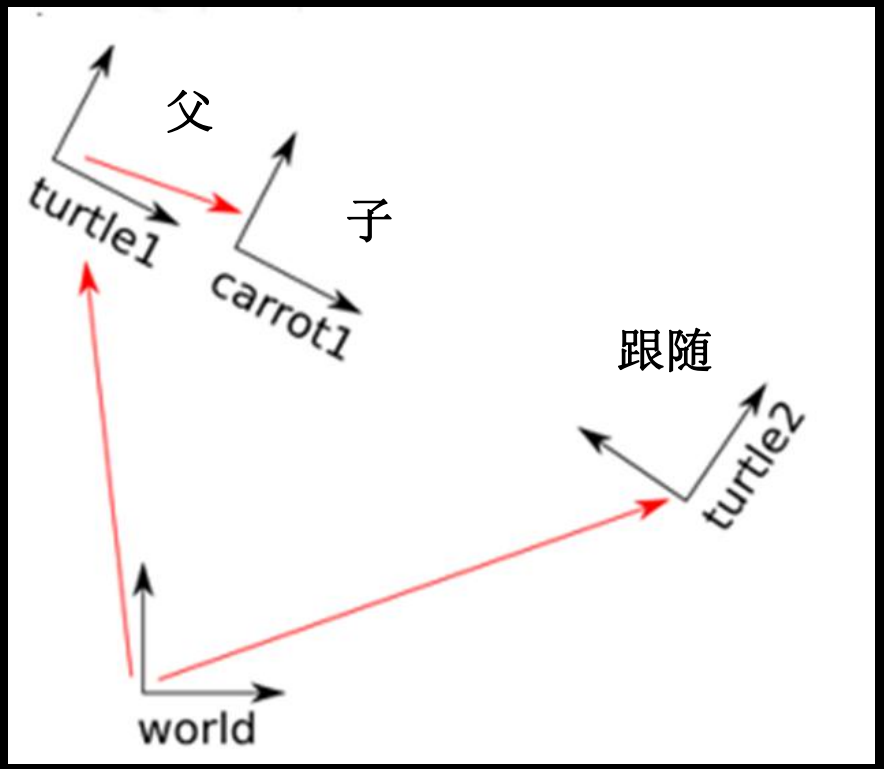
其中重点是位置转换使用tf2。



添加新坐标

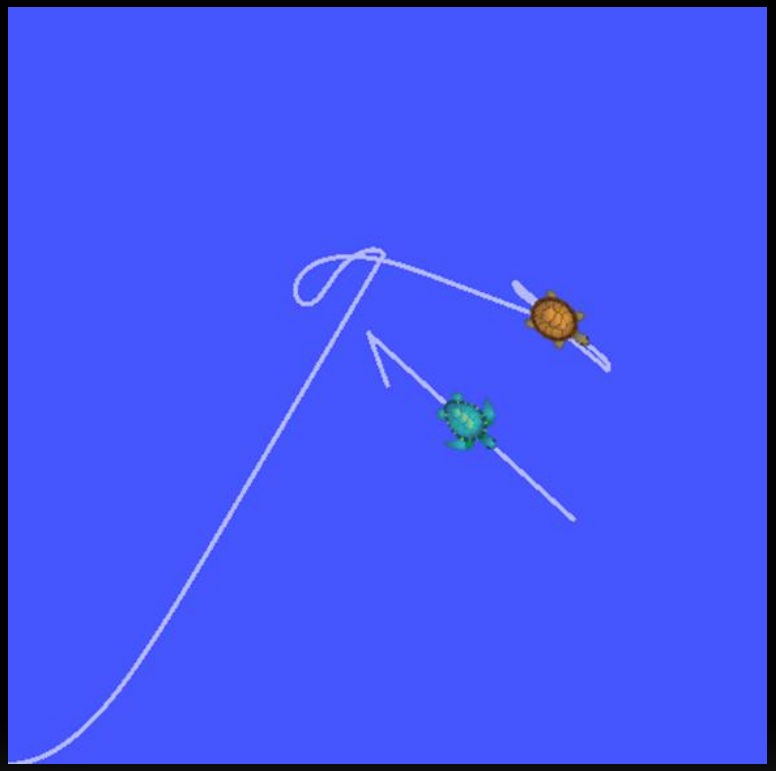
定义了一个新的frame名称为carrot1,它的父坐标是turtle1,距离它父坐标是在y轴 即在左侧2米左右。

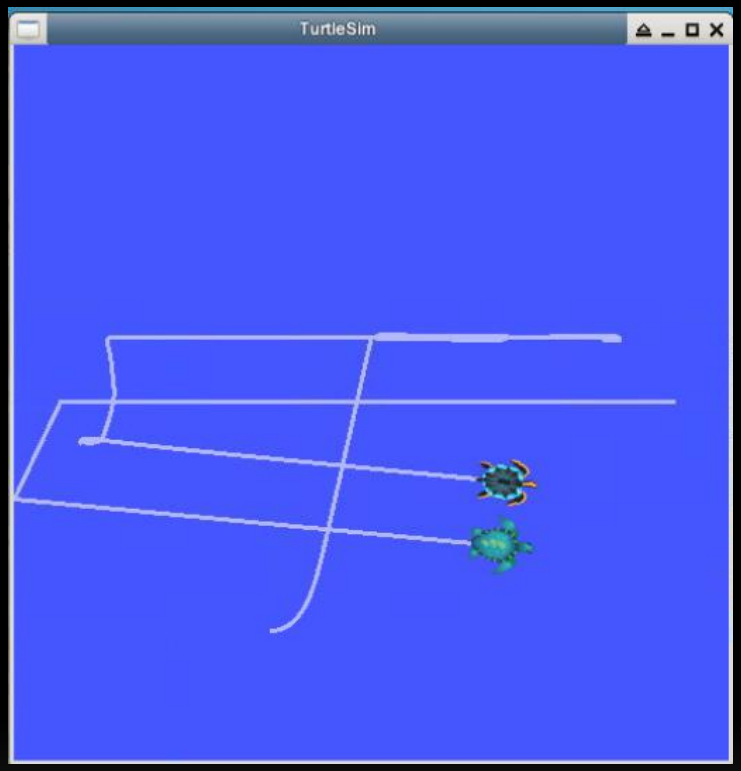
那么 turtle2跟carrot1
保持什么样的位置关系？



添加新坐标—固定

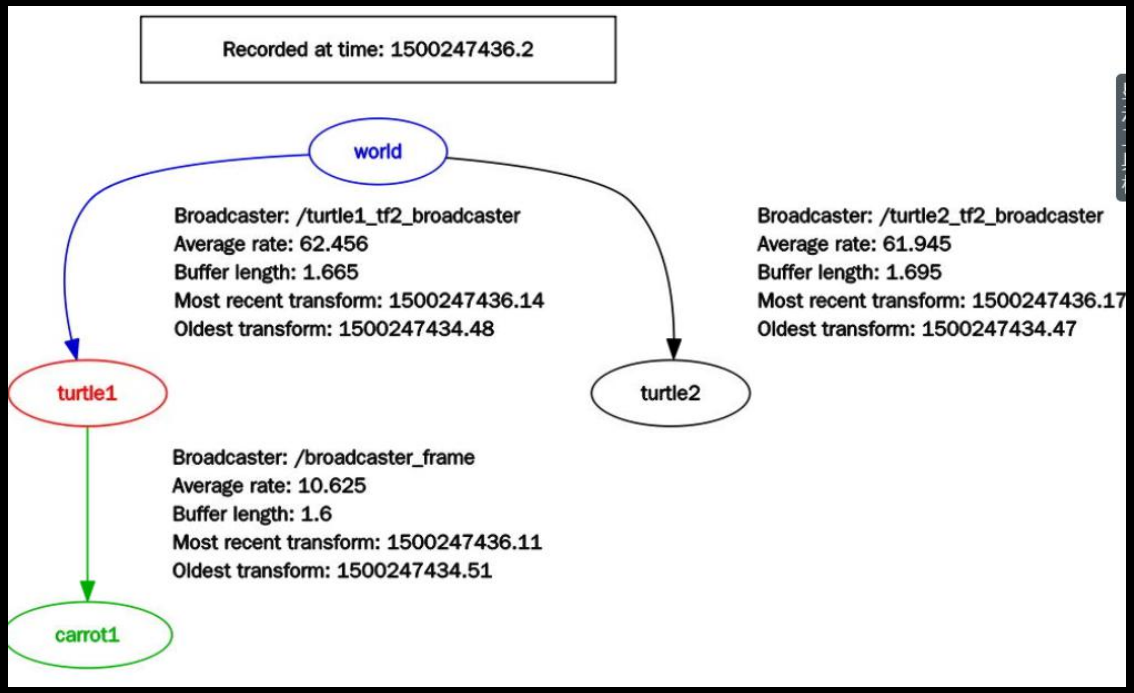
将turtle2跟随turtle1改为跟随carrot1，
这时运行后会发现， turtle2跟着carrot1改变，
但是始终在carrot1左侧n米左右





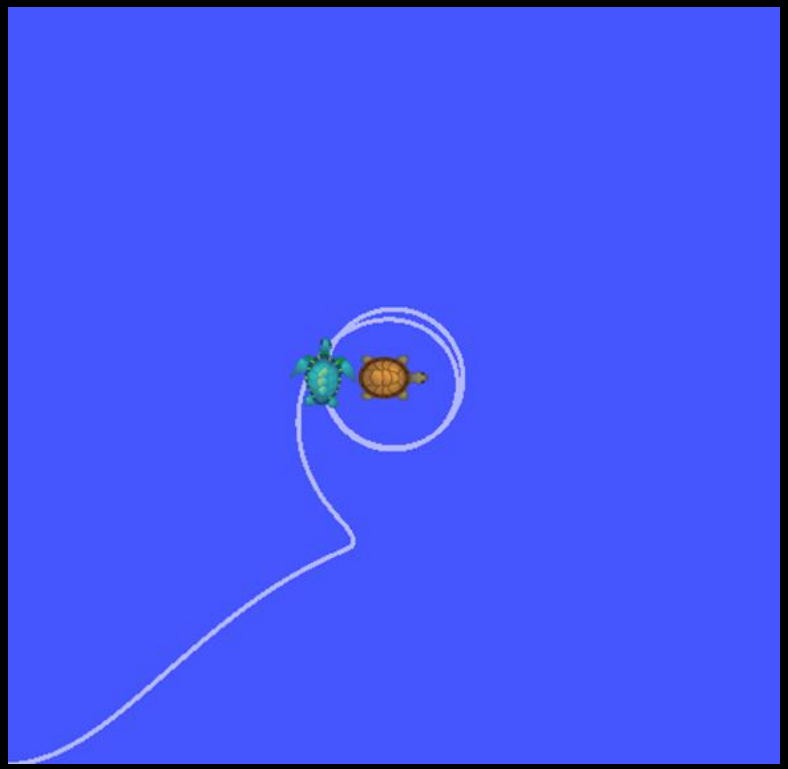
添加新坐标—固定

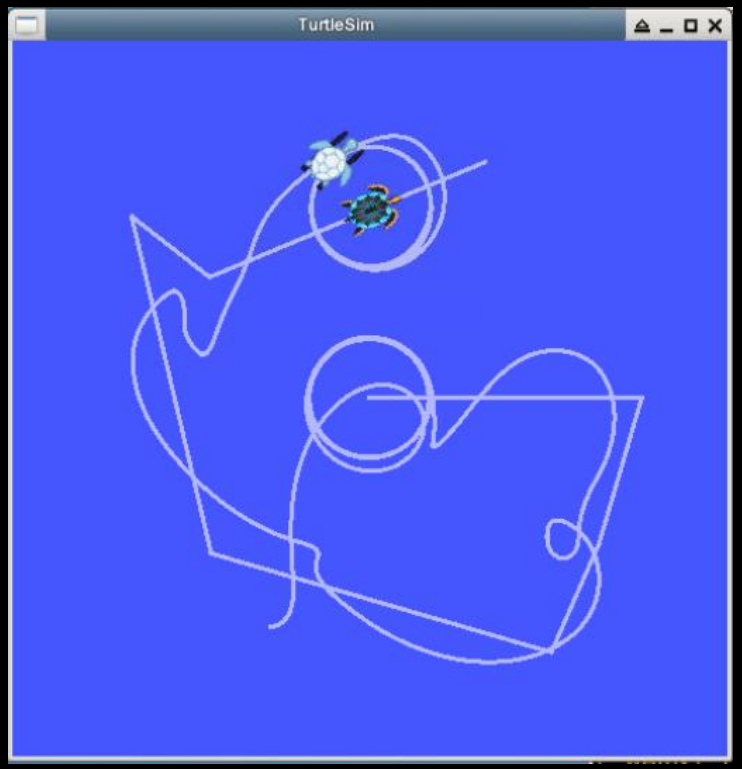
将turtle2跟随turtle1改为跟随carrot1，
这时运行后会发现， turtle2跟着carrot1改变，
但是始终在carrot1左侧n米左右



添加新坐标—移动

Carrot1的原点在以turtle1为圆心，
半径n米的圆上随时间变化。并不精确。





TF2与时间

当TF树随着时间（默认10s）变化时，tf就储存一次时间快照，记录树形结构的坐标系。使用lookupTransform()函数来查看tf树的最近可能的变化，但不知道什么时候被记录。

获取现在transform的方法：

将前面的turtle_tf_listener.cpp 中的time(0)改为Time::now()，

并添加waitForTransform()语句.

区别----**实验现象几乎一致**

time(0)是查看tf树最近的变化而不是现在的变化，改为now()后，发现turtle2的行动更自然了。

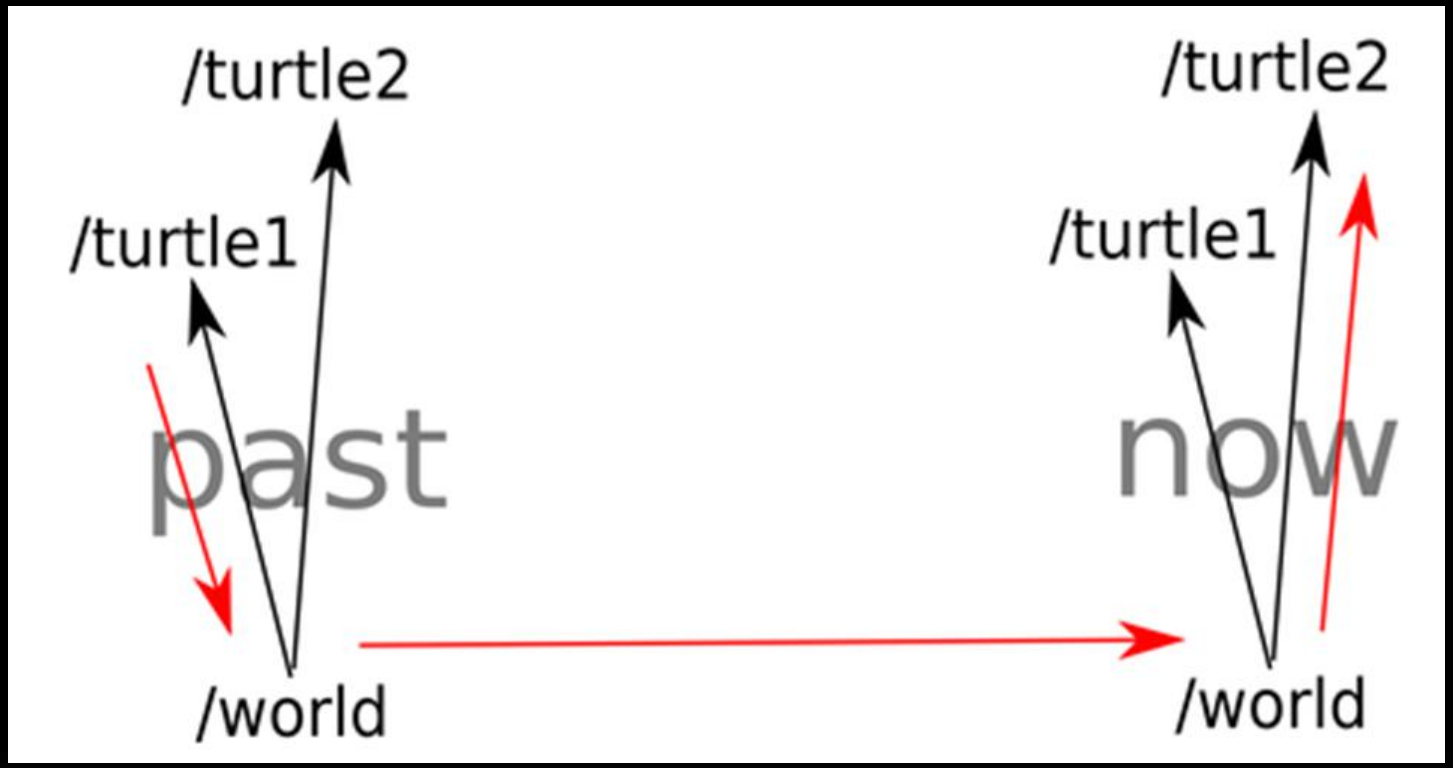
Now() 其实也不是真正意义的上的现在，而是更接近现在.

不过通常情况下，都是设为time(0)。

TF2与时间

如何让第二只乌龟跟着第一只乌龟五秒前的轨迹运动

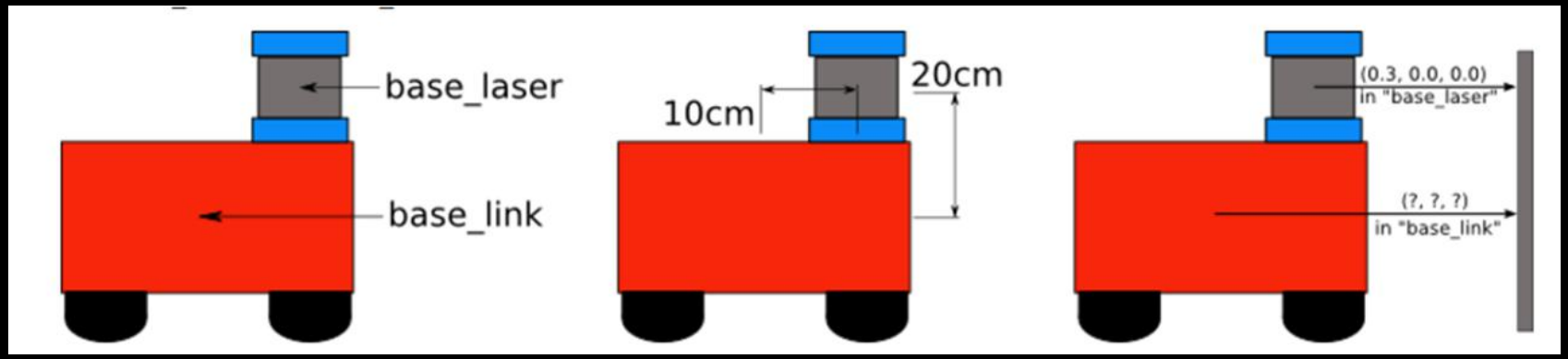
1. turtle2现在的轨迹
2. turtle1五秒以前的轨迹
3. 世界坐标不变
4. turtle2跟随turtle1五秒前轨迹



机器人TF应用

一个移动机器人如何依靠激光雷达或深度视觉躲避障碍物，传感器与机器人本体坐标不完全重合？

底部的中心（base_link)，激光雷达的中心(base_laser）。对于导航包来说，base_link就是机器人的旋转中心。



坐标关系

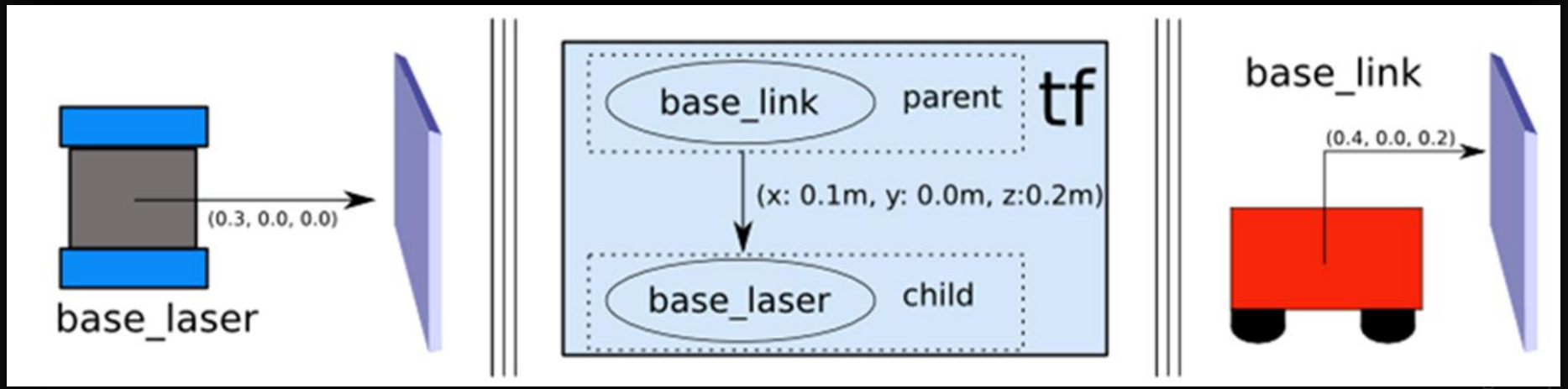
base_laser对应于base_link的数据：

(x: 0.1m, y: 0.0m, z: 0.2m)

base_link对应与base_laser的数据：

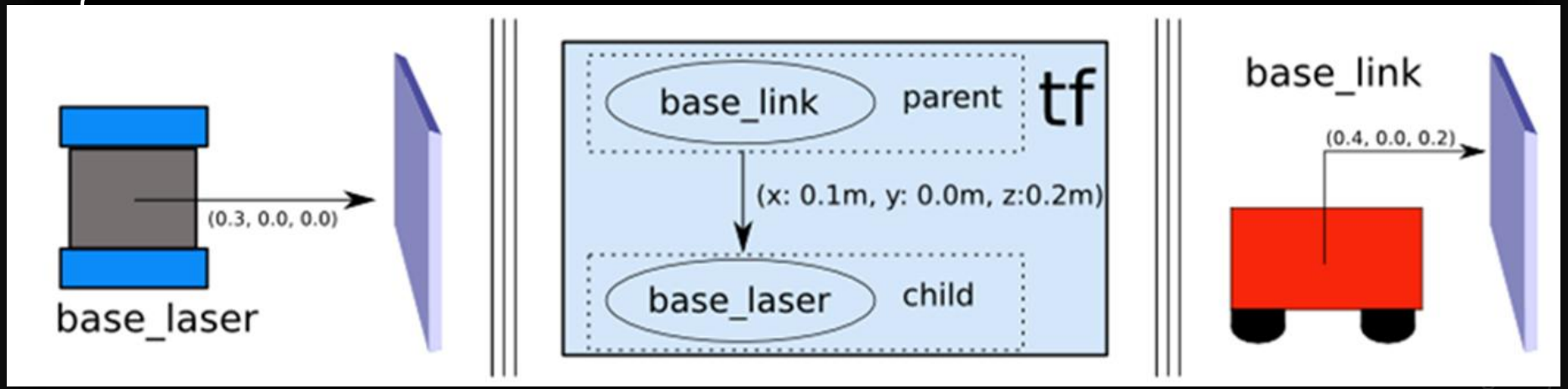
(x: -0.1m, y: 0.0m, z: -0.20)

只需要用tf将base_link和base_laser之间的关系定义一次，之后就让tf来帮我们管理这两个坐标系之间的转换



坐标关系

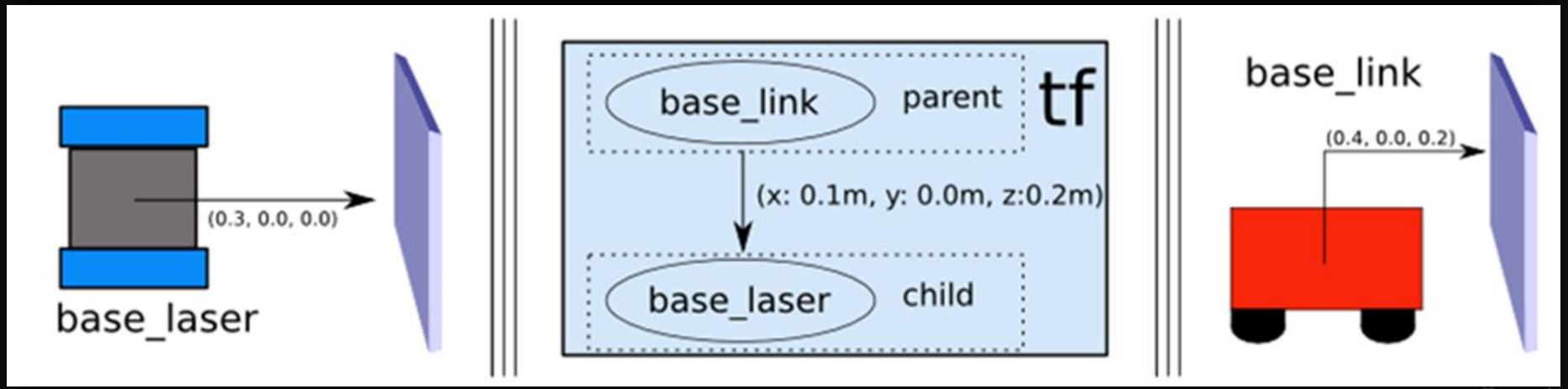
```
while(n.ok()){
        broadcaster.sendTransform(
        tf::StampedTransform(  //旋转转换
        tf::Transform(tf::Quaternion(0, 0, 0, 1), //pitch,roll,yaw
        tf::Vector3(0.1, 0.0, 0.2)), //x,y,z
        ros::Time::now(),"base_link", "base_laser"));
        r.sleep();
}
```



坐标转换

```
laser_point.point.x = 1.0;
laser_point.point.y = 0.2;
laser_point.point.z = 0.0;  //激光数据，(1.0,0.2,0.0)

listener.transformPoint("base_link", laser_point, base_point);
//转换目标、来源、基点
//base_laser的数据转到base_link下
```



测试

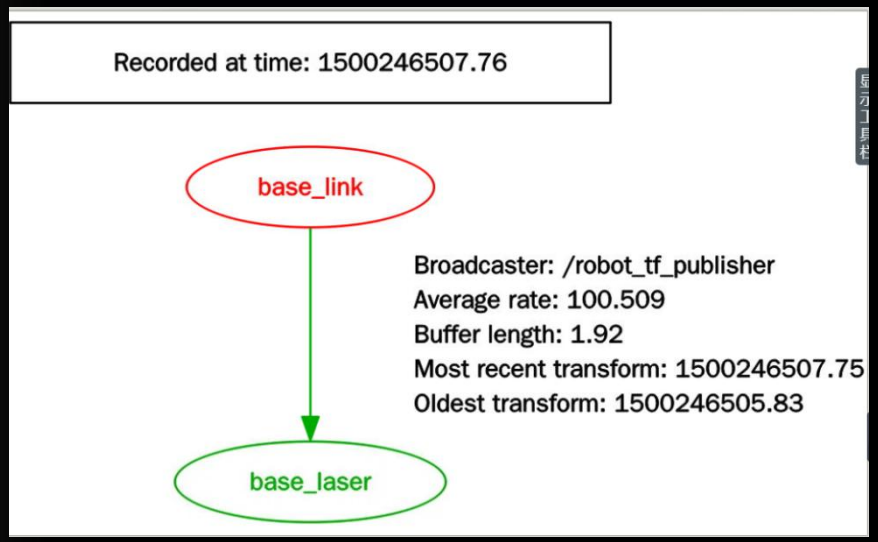
Terminal 终端 - shiyanlou@801cb0fb9ab1: ~

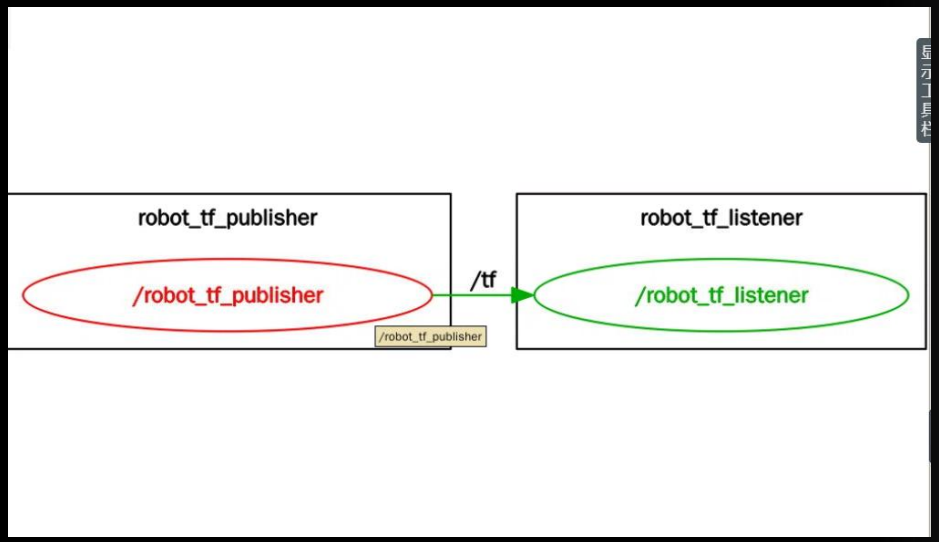
文件(F) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)

rosrun robot_setup_tf_tutorial tf_broadcaster | shiyanlou@801cb0fb9ab1: ~ | roscore http://801cb0fb9ab1:11311/

```
shiyanlou:~/ $ rosrun robot_setup_tf_tutorial tf_listener              [7:00:44]
[ INFO] [1500246051.669097760]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246051.66
[ INFO] [1500246052.673930028]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246052.67
[ INFO] [1500246053.664544679]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246053.65
[ INFO] [1500246054.665008775]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246054.66
[ INFO] [1500246055.668946425]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246055.66
[ INFO] [1500246056.668930747]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246056.66
[ INFO] [1500246057.664408164]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246057.65
[ INFO] [1500246058.664951692]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246058.66
[ INFO] [1500246059.664335075]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246059.65
[ INFO] [1500246060.664744428]: base_laser: (1.00, 0.20. 0.00) -----> base_link:
 (1.10, 0.20, 0.20) at time 1500246060.65
```

测试


Recorded at time: 1500246507.76
base_link
Broadcaster: /robot_tf_publisher
Average rate: 100.509
Buffer length: 1.92
Most recent transform: 1500246507.75
Oldest transform: 1500246505.83
base_laser


robot_tf_publisher
/robot_tf_publisher
/tf
robot_tf_listener
/robot_tf_listener
/robot_tf_publisher

发布机器人状态

当机器人有很多相关坐标系时，都发布到tf会比较繁琐。

机器人状态发布器（robot state publisher）完成这个工作。

机器人状态发布器内部有一个机器人的运动学模型，因此给定机器人的关节位置，它能计算出连杆的3D姿势(pose).

把它当做一个节点(推荐方法）也可以当做一个库。

节点：

1.一个参数服务器上的urdf xml机器人描述

2.一个发布sensor_msgs/JointState 关节位置的源

TF2总结

优点：

1. 容易上手
2. 不需要考虑数值计算细节
3. 接口简单，广播和监听

.....

缺点：

1. 笨拙
2. 维护开销大
3. 没有实时性

本课总结

1. 介绍TF的基本概念
2. 讲解tf的常用工具
3. 通过c++编写tf的常用操作的程序，如广播和监听等
4. 通过实例加深对tf的理解