# ROS坐标变换

# 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。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，需要引入坐标系以及坐标变换的概念。

# TF坐标变换

机器人系统上，有多个传感器，如激光雷达、摄像头等，有的传感器是可以感知机器人周边的物体方位(或者称之为:坐标，横向、纵向、高度的距离信息)的，以协助机器人定位障碍物，是否可以直接将物体相对该传感器的方位信息视作物体相对于机器人系统或机器人其它组件的方位信息呢？显示是不行的，这中间需要一个转换过程。更具体描述如下:

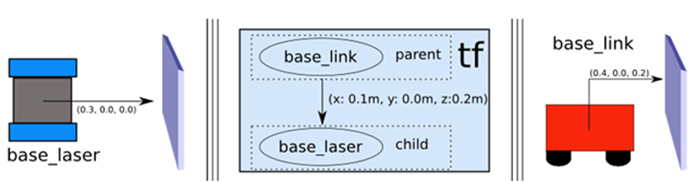


具体来说，我们假设有一个机器人，包括一个机器人移动平台和一个安装在平台之上的激光雷达。以这个机器人为例，定义两个坐标系，一个坐标系是以机器人移动平台的中心为原点，称为base\_link参考系；另一个坐标系以激光雷达的中心为原点，称为base\_laser参考系。

假设在机器人运行过程中，激光雷达可以采集到距离前方障碍物的数据，这些数据是以激光雷达为原点的测量值，也就是base\_laser参考系下的测量值。现在，如果我们想使用这些数据帮助机器人完成避障功能，当然，由于激光雷达在机器人之上，直接使用这些数据不会产生太大的问题，但是激光雷达并不在机器人的中心之上，在精度要求较高的系统中，会始终存在一个雷达与机器人中心的偏差值。

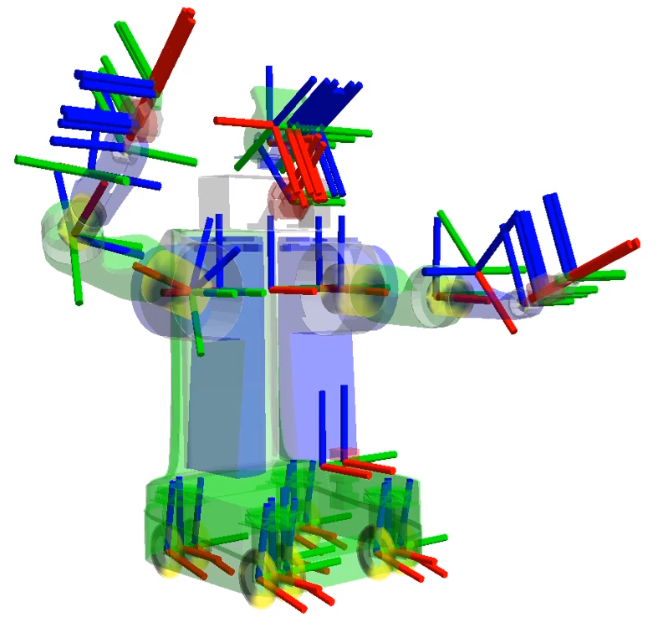
如果我们采用一种坐标变换，将激光数据从base\_laser参考系变换到base\_link参考下，问题就可以解决。这里我们就需要定义这两个坐标系之间的相对位姿关系。

   为了定义这个变换关系，假设我们已知激光雷达安装的位置在机器人的中心点上方20cm，前方10cm处。这就根据这些数据，就足以定义这两个参考系之间的变换关系：当我们获取激光数据后，采用(x: -0.1m, y: 0.0m, z: -0.2m)的坐标变换，就可以将数据从base\_laser参考系变换到base\_link参考系了。

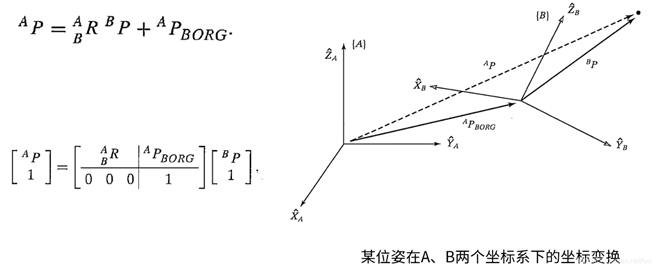




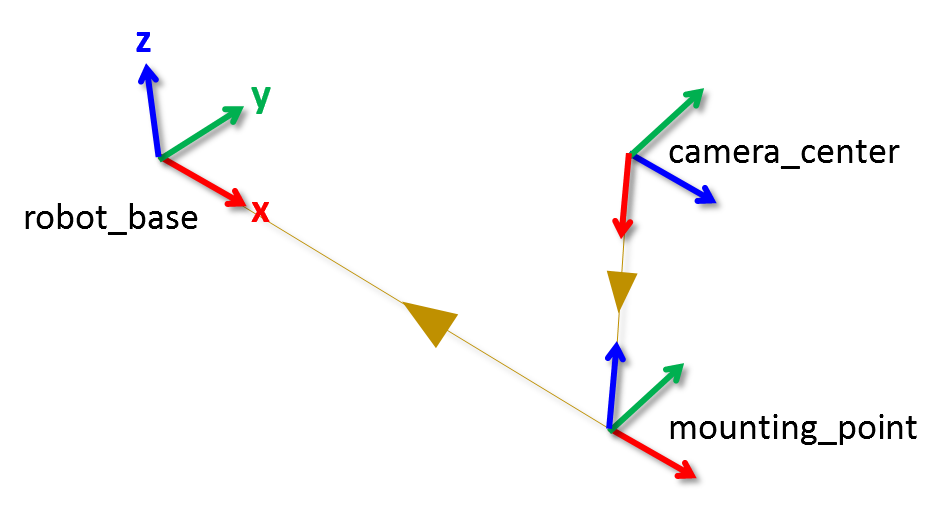
从上边的示例看来，参考系之间的坐标变换好像并不复杂，但是在复杂的系统中，存在的参考系可能远远大于两个，如果我们都使用这种手动的方式进行变换，估计很快你就会被繁杂的坐标关系搞蒙了。



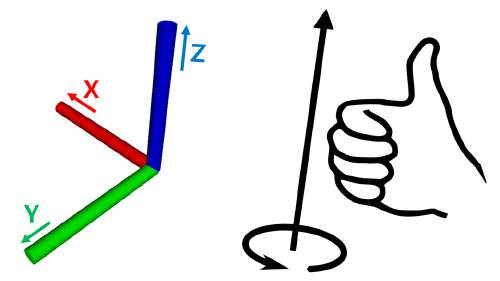
ROS提供的tf变换就是为解决这个问题而生的，tf功能包提供了存储、计算不同数据在不同坐标系之间变换的功能，我们只需要告诉tf树这些参考系之间的变换公式即可，这棵tf树就可以“树”的数据结构管理所需要的参考系变换。



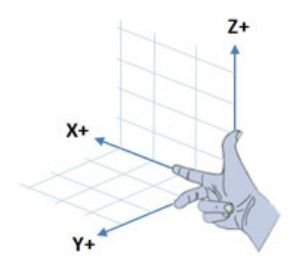
坐标系A可以通过一系列的旋转和平移变换成坐标系B，平移和旋转可以通过4x4的变换矩阵来描述。



ROS中的旋转轴使用x，y和z轴表示。正面是x轴的正方向，轴是红色（R）。左边是y轴的正方向，轴用绿色（G）表示。最后，上方是z轴的正方向，轴用蓝色（B）表示。



这种坐标表示法在ROS编程中经常使用，必须以x：forward，y：left，z：up的形式进行编程。

为了便于记忆，采用右手坐标系来表示，将x轴视为食指，将y轴视为中指，将z轴视为拇指。顺序是x、y、z，且颜色是RGB颜色顺序。  


* REP-03 标准测量单位和坐标变换的相关约定

https://www.ros.org/reps/rep-0103.html

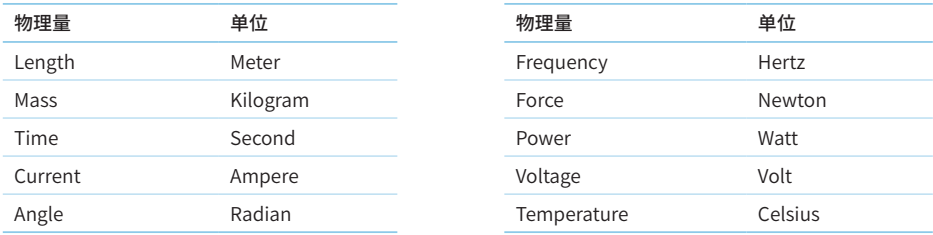
* [REP 105](https://link.zhihu.com/?target=http%3A//www.ros.org/reps/rep-0105.html" \t "_blank)涉及到移动底座坐标系的标准名称约定；

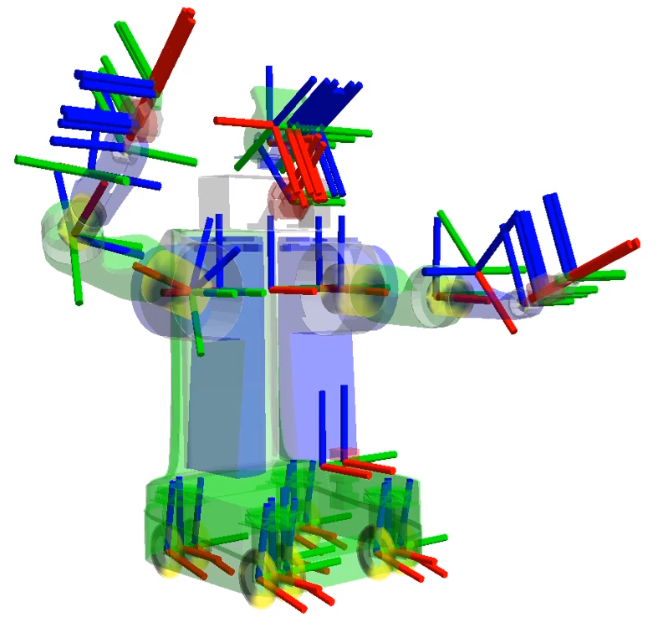
https://www.ros.org/reps/rep-0103.html

* [REP 120](https://link.zhihu.com/?target=http%3A//www.ros.org/reps/rep-0120.html)包含类人机器人的标准坐标系约定；

·有关使用到的某些数学术语定义，请参阅[/Terminology](https://link.zhihu.com/?target=https%3A//wiki.ros.org/tf2/Terminology)页面。

对ROS中所使用的消息（message），推荐使用世界上最广泛运用的标准单位SI。为了确保这一点，REP-01031也明确了各物理量的单位。例如，长度（Length）使用米（merter）、质量（Mass）使用千克（Kilogram）、时间（Time）使用秒（Second）、电流（Current）使用安培（Ampere）、角度（Angle）使用弧度（Radian）、频率（Frequency）使用赫兹（Hertz）、力（Force）使用牛顿（Newton）、功率（Power）使用瓦（Watt）、电压（Voltage）使用伏特（Volt）、温度（Temperature）使用摄氏度（Celsius）。其他所有单位都是这些单位的组合。  
例如，**平移速度**以米/秒表示，**旋转速度以弧度/秒**表示。消息鼓励重用ROS提供的方式，但也可以根据需要使用用户全新定义的新的类型的消息。然而，消息用到的单位却必须要遵守使用SI单位，这是为了让其他用户使用这种消息的时候不需要转换单位。



一个机器人系统通常有很多三维坐标——机器人本体坐标系，头部坐标系、机械夹具的坐标系等，而且这些坐标系会随时间的推移发生变化。

ROS中的TF功能包会**跟踪记录一段时间内**（默认为10s之内）这些坐标系，并且允许ROS节点请求相关数据。

例如：5秒钟之前，机器人头部坐标系相对于机器人本体坐标系的变换关系是什么？机器人本体坐标系相对于全局坐标系的位置是多少？

需要说明的是：TF功能包可以在分布式系统使用，一个机器人系统中所有的坐标变换，对于所有节点都是可用的，所有订阅了TF消息的节点都会保存一份所有坐标变换的数据，所以，不需要使用中心服务器来存储任何数据。

### 说明：

在ROS中坐标变换最初对应的是tf，不过在 hydro 版本开始, tf 被弃用，迁移到 **tf2**,**后者更为简洁高效**。

tf2对应的常用功能包有两大类：

**数据类型支持包**

* **tf2\_geometry\_msgs**: 用于处理[geometry\_msgs](https://link.zhihu.com/?target=https%3A//wiki.ros.org/geometry_msgs)数据类型的tf2方法。

tf2\_geometry\_msgs包含用于在各种geometry\_msgs数据类型之间转换的函数。

* [**tf2\_bullet**](https://link.zhihu.com/?target=https%3A//wiki.ros.org/tf2_bullet)**软件包：用于处理**[**bullet**](https://link.zhihu.com/?target=https%3A//wiki.ros.org/bullet)**数据类型的tf2方法，为C++软件包；**

* **[tf2\_eigen](https://link.zhihu.com/?target=https%3A//wiki.ros.org/tf2_eigen" \t "_blank)软件包：用于本地处理Eigen数据类型的tf2方法，为C++软件包；**
* [**tf2\_kdl**](https://link.zhihu.com/?target=https%3A//wiki.ros.org/tf2_kdl)**软件包：用于本地处理**[**kdl**](https://link.zhihu.com/?target=https%3A//wiki.ros.org/kdl)**数据类型的tf2方法，为C++或者Python软件包；**
* [**tf2\_sensor\_msgs**](https://link.zhihu.com/?target=https%3A//wiki.ros.org/tf2_sensor_msgs)**软件包：用于本地处理**[**sensor\_msgs**](https://link.zhihu.com/?target=https%3A//wiki.ros.org/sensor_msgs)**数据类型的tf2方法，为C++或者Python软件包。**

**工具**

* **tf2**: 封装了坐标变换的常用消息。
* **tf2\_ros**: 为tf2提供了roscpp和rospy绑定，封装了坐标变换常用的API。

# 1 坐标msg消息

订阅发布模型中数据载体 msg 是一个重要实现，首先需要了解一下，在坐标变换实现中常用的 msg：**geometry\_msgs**/Transform**Stamped**和geometry\_msgs/Point**Stamped**。

前者用于传输与坐标系相对位置的信息，后者用于传输某个坐标系内坐标点的信息。在坐标变换中，频繁的需要使用到坐标系的相对关系以及坐标点信息。

**1.geometry\_msgs/TransformStamped**

命令行键入: rosmsg info geometry\_msgs/TransformStamped

**std\_msgs/Header header**  **#头信息**

uint32 seq #|-- 序号

time stamp #|-- 时间戳

string frame\_id #|-- 坐标 ID

**string child\_frame\_id** **#子坐标系的 id**

**geometry\_msgs/Transform transform** **#坐标信息**

**geometry\_msgs/Vector3 translation** #偏移量

float64 x #|-- X 方向的偏移量

float64 y #|-- Y 方向的偏移量

float64 z #|-- Z 方向上的偏移量

**geometry\_msgs/Quaternion rotation** #四元数

float64 x

float64 y

float64 z

float64 w

四元数的一个很好的视频教程：

https://eater.net/quaternions

[Visualizing quaternions, an explorable video series](https://eater.net/quaternions)

### 2.geometry\_msgs/PointStamped

命令行键入: rosmsg info geometry\_msgs/PointStamped

**std\_msgs/Header header** #头

uint32 seq #|-- 序号

time stamp #|-- 时间戳

string frame\_id #|-- 所属坐标系的 id

**geometry\_msgs/Point point** #点坐标

float64 x #|-- x y z 坐标

float64 y

float64 z

# 2 静态坐标变换

所谓静态坐标变换，是指两个坐标系之间的相对位置是固定的。

具体示例：

现有一机器人模型，由机器人本体和雷达组成，各对应一坐标系，坐标系的原点分别位于主体与雷达的物理中心，已知雷达原点相对于主体原点位移关系如下: x：0.2/y：0.0/z：0.5。当前雷达检测到一障碍物，在雷达坐标系中障碍物的坐标为 (2.0 3.0 5.0),请问，该障碍物相对于主体的坐标是多少？

**实现分析:**

1. 坐标系相对关系，可以通过发布方发布
2. 订阅方，订阅到发布的坐标系相对关系，再传入坐标点信息，然后借助于 tf 实现坐标变换，并将结果输出

**实现流程:**

1. 新建功能包，添加依赖
2. 编写发布方实现
3. 编写订阅方实现
4. 执行并查看结果

**实现：**

### 创建功能包

创建项目功能包依赖于 **tf2、tf2\_ros、tf2\_geometry\_msgs**、roscpp rospy std\_msgs geometry\_msgs

### 2.发布方

*/\**

*静态坐标变换发布方:*

*发布关于 laser 坐标系的位置信息*

*实现流程:*

*1.包含头文件*

*2.初始化 ROS 节点*

*3.创建静态坐标转换广播器*

*4.创建坐标系信息*

*5.广播器发布坐标系信息*

*6.spin()*

*\*/*

*// 1.包含头文件*

#include "ros/ros.h"

#include "tf2\_ros/**static\_transform\_broadcaster.h**"

#include "geometry\_msgs/TransformStamped.h"

#include "tf2/LinearMath/Quaternion.h"

**int** **main**(**int** argc, **char** **\***argv[])

{

setlocale(LC\_ALL,"");

*// 2.初始化 ROS 节点*

ros**::**init(argc,argv,"static\_brocast");

*// 3.创建静态坐标转换广播器*

tf2\_ros**::StaticTransformBroadcaster** broadcaster;

*// 4.创建坐标系信息*

geometry\_msgs**::**TransformStamped ts;

*//----设置头信息*

ts.header.seq **=** 100;

ts.header.stamp **=** ros**::**Time**::**now();

ts.header.frame\_id **=** "base\_link";

*//----设置子级坐标系*

ts.child\_frame\_id **=** "base\_laser";

*//----设置子级相对于父级的偏移量*

ts.transform.translation.x **=** 0.2;

ts.transform.translation.y **=** 0.0;

ts.transform.translation.z **=** 0.5;

*//----设置四元数:将 欧拉角数据转换成四元数*

tf2**::**Quaternion qtn;

qtn.setRPY(0,0,0);

ts.transform.rotation.x **=** qtn.getX();

ts.transform.rotation.y **=** qtn.getY();

ts.transform.rotation.z **=** qtn.getZ();

ts.transform.rotation.w **=** qtn.getW();

*// 5.广播器发布坐标系信息*

broadcaster.sendTransform(ts);

ros**::**spin();

**return** 0;

}

### 3.订阅方

*/\**

*订阅坐标系信息，生成一个相对于 子级坐标系的坐标点数据，转换成父级坐标系中的坐标点*

*实现流程:*

*1.包含头文件*

*2.初始化 ROS 节点*

*3.创建 TF 订阅节点*

*4.生成一个坐标点(相对于子级坐标系)*

*5.转换坐标点(相对于父级坐标系)*

*6.spin()*

*\*/*

*//1.包含头文件*

#include "ros/ros.h"

#include "tf2\_ros/transform\_listener.h"

#include "tf2\_ros/buffer.h"

#include "geometry\_msgs/PointStamped.h"

#include "tf2\_geometry\_msgs/tf2\_geometry\_msgs.h"

//注意: 调用 transform 必须包含该头文件

**int** **main**(**int** argc, **char** **\***argv[])

{

setlocale(LC\_ALL,"");

*// 2.初始化 ROS 节点*

ros**::**init(argc,argv,"tf\_sub");

ros**::**NodeHandle nh;

*// 3.创建 TF 订阅节点*

tf2\_ros**::**Buffer buffer;

tf2\_ros**::**TransformListener listener(buffer);

ros**::**Rate r(1);

**while** (ros**::**ok())

{

*// 4.生成一个坐标点(相对于子级坐标系)*

geometry\_msgs**::**PointStamped point\_laser;

point\_laser.header.frame\_id **=** "laser";

point\_laser.header.stamp **=** ros**::**Time**::**now();

point\_laser.point.x **=** 1;

point\_laser.point.y **=** 2;

point\_laser.point.z **=** 7.3;

*// 5.转换坐标点(相对于父级坐标系)*

*//新建一个坐标点，用于接收转换结果*

*//使用 try 语句或休眠，否则可能由于缓存接收延迟而导致坐标转换失败*

**try**

{

geometry\_msgs**::**PointStamped point\_base;

point\_base **=** buffer.transform(point\_laser,"base\_link");

ROS\_INFO("转换后的数据:(%.2f,%.2f,%.2f),参考的坐标系是:%s",

point\_base.point.x,

point\_base.point.y,

point\_base.point.z,

point\_base.header.frame\_id.c\_str());

}

**catch**(**const** std**::**exception**&** e)

{

*// std::cerr << e.what() << '\n';*

ROS\_INFO("程序异常.....");

}

r.sleep();

ros**::**spinOnce();

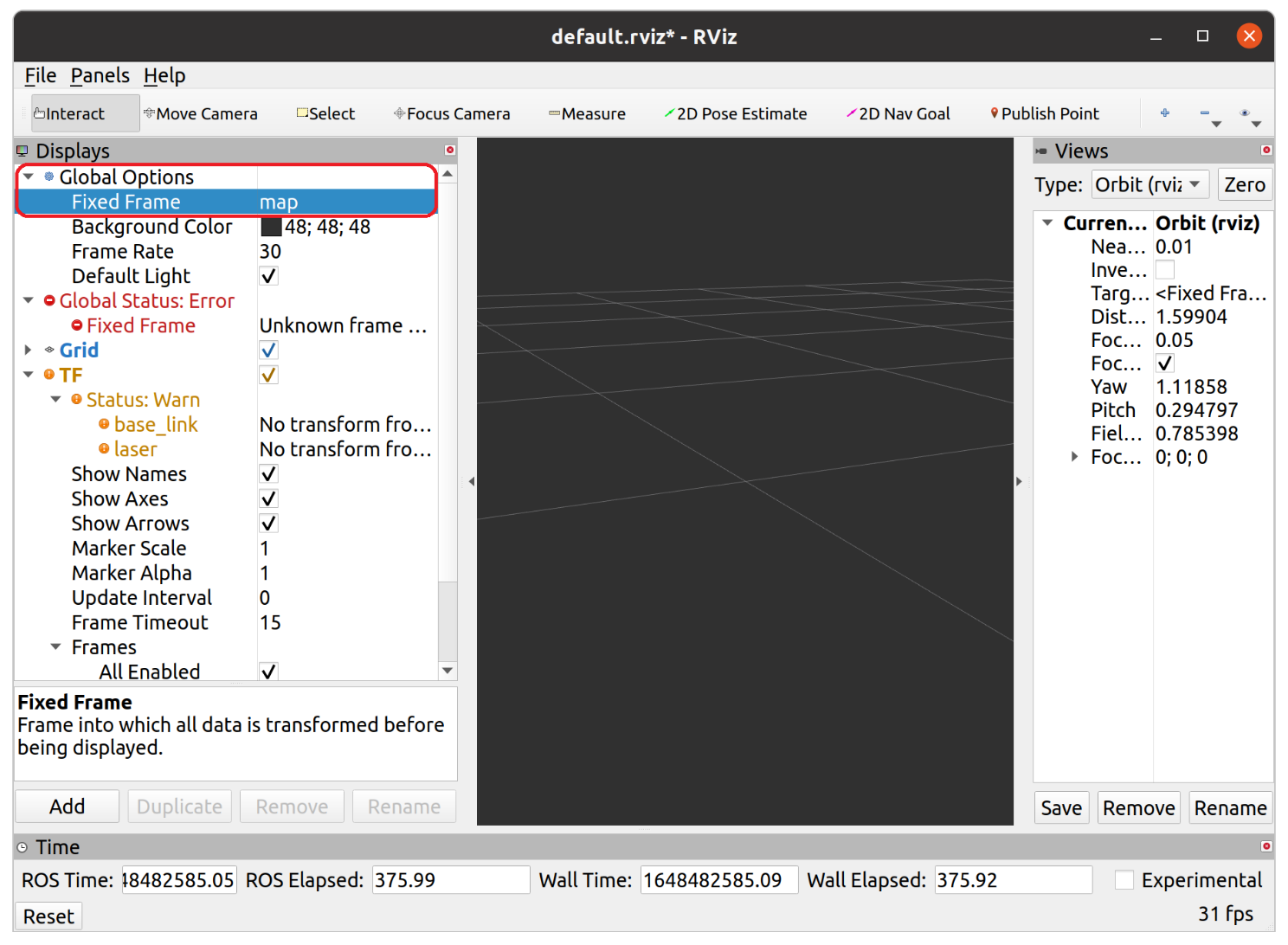
}

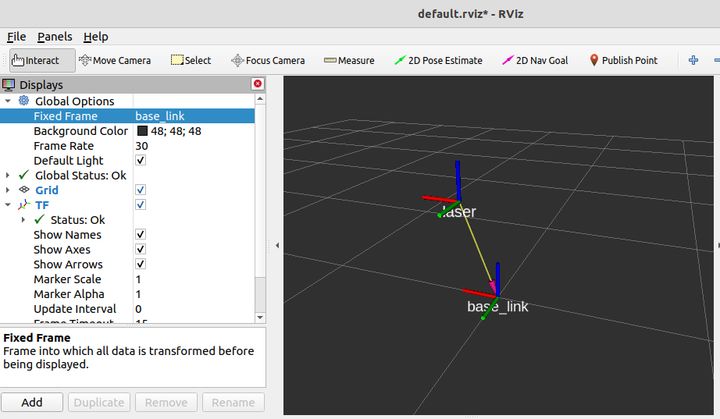
**return** 0;

}

### 4.执行

可以使用命令行或launch文件的方式分别启动发布节点与订阅节点，如果程序无异常，控制台将输出，坐标转换后的结果。





### 说明1:

当坐标系之间的相对位置固定时，那么所需参数也是固定的: 父系坐标名称、子级坐标系名称、x偏移量、y偏移量、z偏移量、x 翻滚角度、y俯仰角度、z偏航角度，实现逻辑相同，参数不同，那么 ROS 系统就已经封装好了专门的节点，使用方式如下:

rosrun tf2\_ros static\_transform\_publisher x偏移量 y偏移量 z偏移量 z偏航角度 y俯仰角度 x翻滚角度 父级坐标系 子级坐标系

示例:

rosrun tf2\_ros static\_transform\_publisher 0.2 0 0.5 0 0 0 /baselink /laser

也建议使用该种方式直接实现静态坐标系相对信息发布。

### 说明2:

可以借助于rviz显示坐标系关系，具体操作:

* 新建窗口输入命令:rviz;
* 在启动的 rviz 中设置Fixed Frame 为 base\_link;
* 点击左下的 add 按钮，在弹出的窗口中选择 TF 组件，即可显示坐标关系。

# 3 动态坐标变换

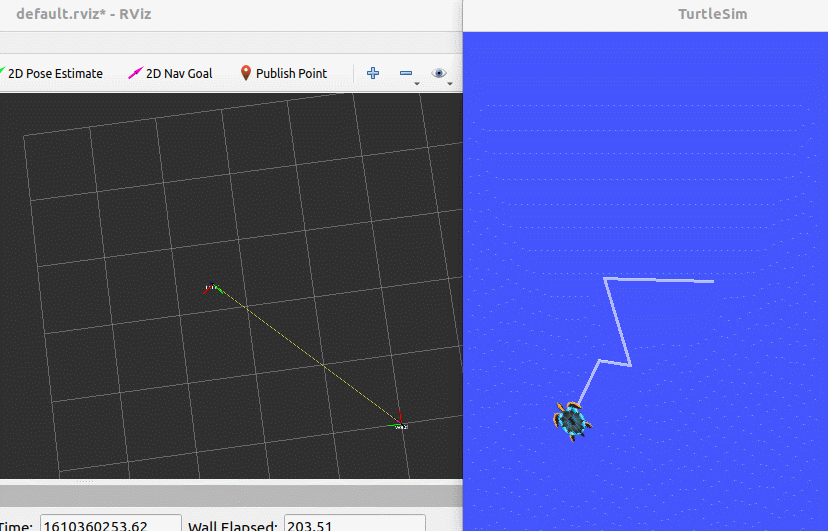
所谓动态坐标变换，是指两个坐标系之间的相对位置是变化的。

**示例:**

启动 turtlesim\_node,该节点中窗体有一个世界坐标系(左下角为坐标系原点)，乌龟是另一个坐标系，键盘控制乌龟运动，将两个坐标系的相对位置动态发布。



**结果演示:**



**实现分析:**

1. 乌龟本身不但可以看作坐标系，也是世界坐标系中的一个坐标点
2. 订阅 turtle1/pose,可以获取乌龟在世界坐标系的 x坐标、y坐标、偏转量以及线速度和角速度
3. 将 pose 信息转换成坐标系相对信息并发布

可以使用如下命令行语句来查看乌龟节点发布的话题和消息类型：

* rostopic list
* rosttopic info /turtle1/**pose**
* rosmsg info turtlesim/**Pose**

float32 x

float32 y

float32 theta

float32 linear\_velocity

float32 angular\_velocity

**实现流程:**

1. 新建功能包，添加依赖
2. 创建坐标相对关系发布方(同时需要订阅乌龟位姿信息)
3. 创建坐标相对关系订阅方
4. 执行

**具体实现：**

### 1.创建功能包

创建项目功能包依赖于 roscpp rospy std\_msgs geometry\_msgs turtlesim tf2 tf2\_ros tf2\_geometry\_msgs

### 发布方

1. */\**
2. *动态的坐标系相对姿态发布(一个坐标系相对于另一个坐标系的相对姿态是不断变动的)*
3. *需求: 启动 turtlesim\_node,该节点中窗体有一个世界坐标系(左下角为坐标系原点)，乌龟是另一个坐标系，键盘*
4. *控制乌龟运动，将两个坐标系的相对位置动态发布*
5. *实现分析:*
6. *1.乌龟本身不但可以看作坐标系，也是世界坐标系中的一个坐标点*
7. *2.订阅 turtle1/pose,可以获取乌龟在世界坐标系的 x坐标、y坐标、偏移量以及线速度和角速度*
8. *3.将 pose 信息转换成 坐标系相对信息并发布*
9. *实现流程:*
10. *1.包含头文件*
11. *2.初始化 ROS 节点*
12. *3.创建 ROS 句柄*
13. *4.创建订阅对象*
14. *5.回调函数处理订阅到的数据(实现TF广播)*
15. *5-1.创建 TF 广播器*
16. *5-2.创建 广播的数据(通过 pose 设置)*
17. *5-3.广播器发布数据*
18. *6.spin*
19. *\*/*
20. *// 1.包含头文件*
21. #include "ros/ros.h"
22. #include "turtlesim/Pose.h"
23. #include "tf2\_ros/**transform\_broadcaster.h**"
24. #include "geometry\_msgs/TransformStamped.h"
25. #include "tf2/LinearMath/Quaternion.h"
26. **void** **doPose**(**const** turtlesim**::**Pose**::**ConstPtr**&** pose){
27. *// 5-1.创建 TF 广播器*
28. **static** tf2\_ros**::**TransformBroadcaster broadcaster;
29. *// 5-2.创建 广播的数据(通过 pose 设置)*
30. geometry\_msgs**::**TransformStamped tfs;
31. *// |----头设置*
32. tfs.header.frame\_id **=** "world";
33. tfs.header.stamp **=** ros**::**Time**::**now();
34. *// |----坐标系 ID*
35. tfs.child\_frame\_id **=** "turtle1";
36. *// |----坐标系相对信息设置*
37. tfs.transform.translation.x **=** pose**->**x;
38. tfs.transform.translation.y **=** pose**->**y;
39. tfs.transform.translation.z **=** 0.0; *// 二维实现，pose 中没有z，z 是 0*
40. *// |--------- 四元数设置*
41. tf2**::**Quaternion qtn;
42. qtn.setRPY(0,0,pose**->**theta);
43. tfs.transform.rotation.x **=** qtn.getX();
44. tfs.transform.rotation.y **=** qtn.getY();
45. tfs.transform.rotation.z **=** qtn.getZ();
46. tfs.transform.rotation.w **=** qtn.getW();
47. *// 5-3.广播器发布数据*
48. broadcaster.sendTransform(tfs);
49. }
50. **int** **main**(**int** argc, **char** **\***argv[])
51. {
52. setlocale(LC\_ALL,"");
53. *// 2.初始化 ROS 节点*
54. ros**::**init(argc,argv,"dynamic\_tf\_pub");
55. *// 3.创建 ROS 句柄*
56. ros**::**NodeHandle nh;
57. *// 4.创建订阅对象*
58. ros**::**Subscriber sub **=**

nh.subscribe**<**turtlesim**::**Pose**>**("/turtle1/pose",1000,doPose);

1. *// 5.回调函数处理订阅到的数据(实现TF广播)*
2. *//*
3. *// 6.spin*
4. ros**::**spin();
5. **return** 0;
6. }

### 3.订阅方

1. //1.包含头文件
2. #include "ros/ros.h"
3. #include "tf2\_ros/transform\_listener.h"
4. #include "tf2\_ros/buffer.h"
5. #include "geometry\_msgs/PointStamped.h"
6. #include "tf2\_geometry\_msgs/tf2\_geometry\_msgs.h" //注意: 调用 transform 必须包含该头文件
7. int main(int argc, char \*argv[])
8. {
9. setlocale(LC\_ALL,"");
10. // 2.初始化 ROS 节点
11. ros::init(argc,argv,"dynamic\_tf\_sub");
12. ros::NodeHandle nh;
13. // 3.创建 TF 订阅节点
14. tf2\_ros::Buffer buffer;
15. tf2\_ros::TransformListener listener(buffer);
16. ros::Rate r(1);
17. while (ros::ok())
18. {
19. // 4.生成一个坐标点(相对于子级坐标系)
20. geometry\_msgs::PointStamped point\_laser;
21. point\_laser.header.frame\_id = **"turtle1";**
22. point\_laser.header.stamp = **ros::Time(0.0)**;
23. point\_laser.point.x = 1;
24. point\_laser.point.y = 1;
25. point\_laser.point.z = 0;
26. // 5.转换坐标点(相对于父级坐标系)
27. //新建一个坐标点，用于接收转换结果
28. //使用 try 语句或休眠，否则可能由于缓存接收延迟而导致坐标转换失败
29. try
30. {
31. geometry\_msgs::PointStamped point\_base;
32. point\_base = buffer.transform(point\_laser,"world");
33. ROS\_INFO("坐标点相对于 world 的坐标为:(%.2f,%.2f,%.2f)",point\_base.point.x,point\_base.point.y,point\_base.point.z);
34. }
35. catch(const std::exception& e)
36. {
37. // std::cerr << e.what() << '\n';
38. ROS\_INFO("程序异常:%s",e.what());
39. }
40. r.sleep();
41. ros::spinOnce();
42. }
43. return 0;
44. }

### 4.执行

可以使用命令行或launch文件的方式分别启动发布节点与订阅节点，如果程序无异常，与演示结果类似。

1、可以使用 rviz 查看坐标系相对关系。

2、使用rostoplist list查看话题消息，多了一个坐标变换的话题消息

/tf

rostopic ech /tf

# 4 多坐标变换

**需求描述:**

现有坐标系统，父级坐标系统 world,下有两子级系统 c1，c2，c1相对于 world，以及 c2相对于 world 的关系是已知的，求 c1原点在 c2中的坐标，又已知在 c1中一点的坐标，要求求出该点在 c2中的坐标



**实现分析:**

1. 首先，需要发布 c1相对于 world，以及 c2相对于 world 的坐标系相对关系消息；
2. 然后，需要订阅坐标发布消息，并取出订阅的消息，借助于 tf2 实现 c1和 c2的转换；
3. 最后，还要实现坐标点的转换。

**实现流程:**

1. 新建功能包，添加依赖
2. 创建坐标相对关系发布方(需要发布两个坐标相对关系)
3. 创建坐标相对关系订阅方
4. 执行

**具体实现：**

### 1.创建功能包

创建项目功能包依赖于 **tf2、tf2\_ros、tf2\_geometry\_msgs**、roscpp rospy std\_msgs geometry\_msgs、turtlesim

### 2.发布方

为了方便，使用静态坐标变换发布

<launch>

<node pkg="demo\_tfs" type="static\_transform\_publisher" name="c1" args="0.2 0.8 0.3 0 0 0 /world /c1" output="screen" />

<node pkg="demo\_tfs" type="static\_transform\_publisher" name="c2" args="0.5 0 0 0 0 0 /world /c2" output="screen" />

</launch>

### 3.订阅方

*/\**

*需求:*

*现有坐标系统，父级坐标系统 world,下有两子级系统 c1，c2，*

*c1相对于 world，c2相对于 world 的关系是已知的，*

*求 c1与 c2中的坐标关系，又已知在 c1中一点的坐标，要求求出该点在 c2中的坐标*

*实现流程:*

*1.包含头文件*

*2.初始化 ros 节点*

*3.创建 ros 句柄*

*4.创建 TF 订阅对象*

*5.解析订阅信息中获取 c1坐标系原点在 c2中的坐标*

*解析 c1中的点相对于 c2的坐标*

*6.spin*

*\*/*

*//1.包含头文件*

#include "ros/ros.h"

#include "tf2\_ros/transform\_listener.h"

#include "tf2/LinearMath/Quaternion.h"

#include "tf2\_geometry\_msgs/tf2\_geometry\_msgs.h"

#include "geometry\_msgs/TransformStamped.h"

#include "geometry\_msgs/PointStamped.h"

**int** **main**(**int** argc, **char** **\***argv[])

{ setlocale(LC\_ALL,"");

*// 2.初始化 ros 节点*

ros**::**init(argc,argv,"sub\_frames");

*// 3.创建 ros 句柄*

ros**::**NodeHandle nh;

*// 4.创建 TF 订阅对象*

tf2\_ros**::**Buffer buffer;

tf2\_ros**::**TransformListener listener(buffer);

*// 5.解析订阅信息中获取 c1坐标系原点在 c2中的坐标*

ros**::**Rate r(1);

**while** (ros**::**ok())

{

**try**

{

*// 解析 c1中的点相对c2的坐标*

*/\**

*buffer的成员函数lookupTransform的原型：*

*lookupTranform(const std::string& target\_frame,*

*const std::string& source\_frame,*

*const ros::Time& Time);*

*\*/*

geometry\_msgs**::**TransformStamped tfs **=**

buffer.**lookupTransform**("c2","c1",ros**::**Time(0));

ROS\_INFO("c1相对于 c2的坐标关系:父坐标系

ID=%s",tfs.header.frame\_id.c\_str());

ROS\_INFO("c1 相对于 c2的坐标关系:子坐标系

ID=%s",tfs.child\_frame\_id.c\_str());

ROS\_INFO("c1 相对于 c2的坐标关系:x=%.2f,y=%.2f,z=%.2f",

tfs.transform.translation.x,

tfs.transform.translation.y,

tfs.transform.translation.z

);

*// 坐标点解析*

geometry\_msgs**::**PointStamped ps;

ps.header.frame\_id **=** "c1";

ps.header.stamp **=** ros**::**Time**::**now();

ps.point.x **=** 1.0;

ps.point.y **=** 2.0;

ps.point.z **=** 3.0;

geometry\_msgs**::**PointStamped psAtc2;

psAtc2 **=** buffer.transform(ps,"c2");

ROS\_INFO("在 c2 中的坐标:x=%.2f,y=%.2f,z=%.2f",

psAtc2.point.x,

psAtc2.point.y,

psAtc2.point.z

);

}

**catch**(**const** std**::**exception**&** e)

{

*// std::cerr << e.what() << '\n';*

ROS\_INFO("异常信息:%s",e.what());

}

r.sleep();

*// 6.spin*

ros**::**spinOnce();

}

**return** 0;

}

### 4.执行

可以使用命令行或launch文件的方式分别启动发布节点与订阅节点，如果程序无异常，将输出换算后的结果。

# 5 坐标系关系查看

在机器人系统中，涉及的坐标系有多个，为了方便查看，ros 提供了专门的工具，可以用于生成显示坐标系关系的 pdf 文件，该文件包含树形结构的坐标系图谱。

### 1准备

首先调用rospack find tf2\_tools查看是否包含该功能包，如果没有，请使用如下命令安装:

sudo apt install ros-noetic-tf2-tools

### 2使用

### 2.1生成 pdf 文件

启动坐标系广播程序之后，运行如下命令:

rosrun tf2\_tools view\_frames.py

会产生类似于下面的日志信息:

[INFO] [1592920556.827549]: Listening to tf data during 5 seconds...

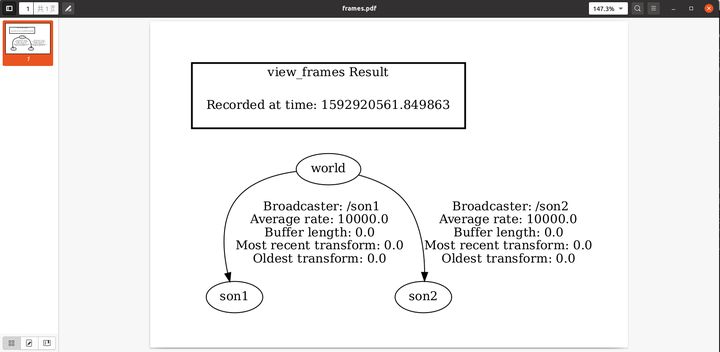
[INFO] [1592920561.841536]: Generating graph in frames.pdf file...

查看当前目录会生成一个 frames.pdf 文件

### 2.2查看文件

可以直接进入目录打开文件，或者调用命令查看文件:evince frames.pdf

内如如图所示:

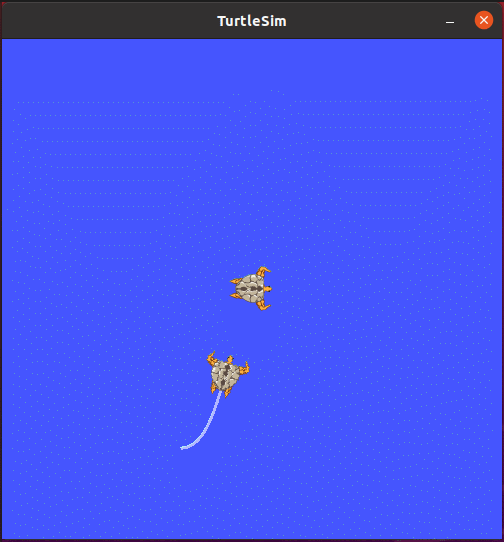


# 6 TF坐标变换实操

**需求描述:**

程序启动之初: 产生两只乌龟，中间的乌龟(A) 和 左下乌龟(B), B 会自动运行至A的位置，并且键盘控制时，只是控制 A 的运动，但是 B 可以跟随 A 运行

**结果演示:**



**实现分析:**

乌龟跟随实现的核心，是乌龟A和B都要发布相对世界坐标系的坐标信息，然后，订阅到该信息需要转换获取A相对于B坐标系的信息，最后，再生成速度信息，并控制B运动。

1. 启动乌龟显示节点
2. 在乌龟显示窗体中生成一只新的乌龟(需要使用服务)
3. 编写两只乌龟发布坐标信息的节点
4. 编写订阅节点订阅坐标信息并生成新的相对关系生成速度信息



**实现流程:**C++ 与 Python 实现流程一致

1. 新建功能包，添加依赖
2. 编写服务客户端，用于生成一只新的乌龟
3. 编写发布方，发布两只乌龟的坐标信息
4. 编写订阅方，订阅两只乌龟信息，生成速度信息并发布
5. 运行

**准备工作:**

1.了解如何创建第二只乌龟，且不受键盘控制

rosservice list

rosservice type /spawn

rossrv info turtlesim/Spawn

创建第二只乌龟需要使用rosservice,话题使用的是 spawn

rosservice call /spawn "x: 1.0

y:1.0

theta:1.0

name:'turtle\_flow'"

输出结果是：

name: "turtle\_flow"

键盘是无法控制第二只乌龟运动的，因为使用的话题: /第二只乌龟名称/cmd\_vel,对应的要控制乌龟运动必须发布对应的话题消息

2.了解如何获取两只乌龟的坐标

是通过话题 /乌龟名称/pose 来获取的

x: 1.0 //x坐标

y: 1.0 //y坐标

theta: -1.21437060833 //角度

linear\_velocity: 0.0 //线速度

angular\_velocity: 1.0 //角速度

具体实现：

### 1.创建功能包

创建项目功能包依赖于 tf2、tf2\_ros、tf2\_geometry\_msgs、roscpp rospy std\_msgs geometry\_msgs、turtlesim

### 2.服务的客户端(请求turtlesim节点生成乌龟)

*/\**

*创建第二只小乌龟*

*\*/*

#include "ros/ros.h"

#include "turtlesim/Spawn.h"

**int** **main**(**int** argc, **char** **\***argv[])

{

setlocale(LC\_ALL,"");

*//执行初始化*

ros**::**init(argc,argv,"create\_turtle");

*//创建节点*

ros**::**NodeHandle nh;

*//创建服务客户端*

ros**::**ServiceClient client **=**

nh.serviceClient**<**turtlesim**::**Spawn**>**("/spawn");

ros**::**service**::**waitForService("/spawn");

//填写请求

turtlesim**::**Spawn spawn;

spawn.request.name **=** "turtle2";

spawn.request.x **=** 1.0;

spawn.request.y **=** 2.0;

spawn.request.theta **=** 3.12415926;

**bool** flag **=** client.call(spawn);

**if** (flag)

{

ROS\_INFO("乌龟%s创建成功!",spawn.response.name.c\_str());

}

**else**

{

ROS\_INFO("乌龟2创建失败!");

}

ros**::**spin();

**return** 0;

}

### 3.发布方(发布两只乌龟的坐标信息)

可以订阅乌龟的位姿信息，然后再转换成坐标信息，两只乌龟的实现逻辑相同，只是订阅的话题名称，生成的坐标信息等稍有差异，可以将差异部分通过参数传入:

* 该节点需要启动两次
* 每次启动时都需要传入乌龟节点名称(第一次是 turtle1 第二次是 turtle2)

*/\**

*该文件实现:需要订阅 turtle1 和 turtle2 的 pose，然后广播相对 world 的坐标系信息*

*注意: 订阅的两只 turtle,除了命名空间(turtle1 和 turtle2)不同外,*

*其他的话题名称和实现逻辑都是一样的，*

*所以我们可以将所需的命名空间通过 args 动态传入*

*实现流程:*

*1.包含头文件*

*2.初始化 ros 节点*

*3.解析传入的命名空间*

*4.创建 ros 句柄*

*5.创建订阅对象*

*6.回调函数处理订阅的 pose 信息*

*6-1.创建 TF 广播器*

*6-2.将 pose 信息转换成 TransFormStamped*

*6-3.发布*

*7.spin*

*\*/*

*//1.包含头文件*

#include "ros/ros.h"

#include "turtlesim/Pose.h"

#include "tf2\_ros/transform\_broadcaster.h"

#include "tf2/LinearMath/Quaternion.h"

#include "geometry\_msgs/TransformStamped.h"

*//保存乌龟名称*

std**::**string turtle\_name;

**void** **doPose**(**const** turtlesim**::**Pose**::**ConstPtr**&** pose){

*// 6-1.创建 TF 广播器 ---------------------------------------- 注意 static*

**static** tf2\_ros**::TransformBroadcaster** broadcaster;

*// 6-2.将 pose 信息转换成 TransFormStamped*

geometry\_msgs**::**TransformStamped tfs;

tfs.header.frame\_id **=** "world";

tfs.header.stamp **=** ros**::**Time**::**now();

tfs.child\_frame\_id **=** turtle\_name;

tfs.transform.translation.x **=** pose**->**x;

tfs.transform.translation.y **=** pose**->**y;

tfs.transform.translation.z **=** 0.0;

tf2**::**Quaternion qtn;

qtn.setRPY(0,0,pose**->**theta);

tfs.transform.rotation.x **=** qtn.getX();

tfs.transform.rotation.y **=** qtn.getY();

tfs.transform.rotation.z **=** qtn.getZ();

tfs.transform.rotation.w **=** qtn.getW();

*// 6-3.发布*

broadcaster.sendTransform(tfs);

}

**int** **main**(**int** argc, **char** **\***argv[])

{

setlocale(LC\_ALL,"");

*// 2.初始化 ros 节点*

ros**::**init(argc,argv,"pub\_tf");

*// 3.解析传入的命名空间*

**if** (argc **!=** 2)

{

ROS\_ERROR("请传入正确的参数");

} **else** {

turtle\_name **=** argv[1];

ROS\_INFO("乌龟 %s 坐标发送启动",turtle\_name.c\_str());

}

*// 4.创建 ros 句柄*

ros**::**NodeHandle nh;

*// 5.创建订阅对象*

ros**::**Subscriber sub **=** nh.subscribe**<**turtlesim**::**Pose**>**(turtle\_name **+** "/pose",1000,doPose);

*// 6.回调函数处理订阅的 pose 信息*

*// 6-1.创建 TF 广播器*

*// 6-2.将 pose 信息转换成 TransFormStamped*

*// 6-3.发布*

*// 7.spin*

ros**::**spin();

**return** 0;

}

### 4.订阅方(解析坐标信息并生成速度信息)

*/\**

*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，在计算线速度和角速度并发布*

*实现流程:*

*1.包含头文件*

*2.初始化 ros 节点*

*3.创建 ros 句柄*

*4.创建 TF 订阅对象*

*5.处理订阅到的 TF*

*6.spin*

*\*/*

*//1.包含头文件*

#include "ros/ros.h"

#include "tf2\_ros/transform\_listener.h"

#include "geometry\_msgs/TransformStamped.h"

#include "geometry\_msgs/Twist.h"

**int** **main**(**int** argc, **char** **\***argv[])

{

setlocale(LC\_ALL,"");

*// 2.初始化 ros 节点*

ros**::**init(argc,argv,"sub\_TF");

*// 3.创建 ros 句柄*

ros**::**NodeHandle nh;

*// 4.创建 TF 订阅对象*

tf2\_ros**::**Buffer buffer;

tf2\_ros**::**TransformListener listener(buffer);

*// 5.处理订阅到的 TF*

*// 需要创建发布 /turtle2/cmd\_vel 的 publisher 对象*

ros**::**Publisher pub **=**

nh.advertise**<**geometry\_msgs**::**Twist**>**("/turtle2/cmd\_vel",1000);

ros**::**Rate rate(10);

**while** (ros**::**ok())

{

**try**

{

*//5-1.先获取 turtle1 相对 turtle2 的坐标信息*

geometry\_msgs**::**TransformStamped tfs **=**

buffer.lookupTransform("turtle2","turtle1",ros**::**Time(0));

*//5-2.根据坐标信息生成速度信息 -- geometry\_msgs/Twist.h*

geometry\_msgs**::**Twist twist;

//turtle1的坐标原点在turtle2坐标系的位置

twist.linear.x **=** sqrt(pow(tfs.transform.translation.x,2) **+** pow(tfs.transform.translation.y,2)) / 2.0;

 twist.angular.z **=** atan2(tfs.transform.translation.y,tfs.transform.translation.x)/0.25;

*//5-3.发布速度信息 -- 需要提前创建 publish 对象*

pub.publish(twist);

}

**catch**(**const** std**::**exception**&** e)

{

*// std::cerr << e.what() << '\n';*

ROS\_INFO("错误提示:%s",e.what());

}

rate.sleep();

*// 6.spin*

ros**::**spinOnce();

}

**return** 0;

}

### 5.运行

使用 launch 文件组织需要运行的节点，内容示例如下:

*<!--*

*tf2 实现小乌龟跟随案例*

*-->*

<launch>

*<!-- 启动乌龟节点与键盘控制节点 -->*

<node pkg="turtlesim" type="turtlesim\_node" name="turtle1" output="screen" />

<node pkg="turtlesim" type="turtle\_teleop\_key" name="key\_control" output="screen"/>

*<!-- 启动创建第二只乌龟的节点 -->*

<node pkg="demo\_tf\_ext" type="new\_turtle" name="turtle2" output="screen" />

*<!-- 启动两个坐标发布节点 -->*

<node pkg="demo\_tf\_ext" type="demo\_tf\_ext\_pub" name="turtle1\_pub" output="screen" args="turtle1" />

<node pkg="demo\_tf\_ext" type="demo\_tf\_ext\_pub" name="turtle2\_pub" output="screen" args="turtle2" />

*<!-- 启动坐标转换节点 -->*

<node pkg="demo\_tf\_ext" type="demo\_tf\_ext\_sub" name="listener" output="screen" />

</launch>

# 7 TF2与TF

### 1.TF2与TF比较\_简介

* TF2已经替换了TF，TF2是TF的超集，建议学习 TF2 而非 TF
* TF2 功能包的增强了内聚性，TF 与 TF2 所依赖的功能包是不同的，TF 对应的是tf包，TF2 对应的是tf2和tf2\_ros包，在 TF2 中不同类型的 API 实现做了分包处理。
* TF2 实现效率更高，比如在:TF2 的静态坐标实现、TF2 坐标变换监听器中的 Buffer 实现等

### 2.TF2与TF比较\_静态坐标变换演示

接下来，我们通过静态坐标变换来演示TF2的实现效率。

### 2.1启动 TF2 与 TF 两个版本的静态坐标变换

TF2 版静态坐标变换:rosrun tf2\_ros static\_transform\_publisher 0 0 0 0 0 0 /base\_link /laser

TF 版静态坐标变换:rosrun tf static\_transform\_publisher 0 0 0 0 0 0 /base\_link /laser 100

会发现，TF 版本的启动中最后多一个参数，该参数是指定发布频率

### 2.2运行结果比对

使用rostopic查看话题，包含/tf与/tf\_static, 前者是 TF 发布的话题，后者是 TF2 发布的话题，分别调用命令打印二者的话题消息

rostopic echo /tf: 当前会循环输出坐标系信息

rostopic echo /tf\_static: 坐标系信息只有一次

### 2.3结论

如果是静态坐标转换，那么不同坐标系之间的相对状态是固定的，既然是固定的，那么没有必要重复发布坐标系的转换消息，很显然的，tf2 实现较之于 tf 更为高效

# 8 小结

坐标变换在机器人系统中是一个极其重要的组成模块，在 ROS 中 TF2 组件是专门用于实现坐标变换的，TF2 实现具体内容又主要介绍了如下几部分:

1.静态坐标变换广播器，可以编码方式或调用内置功能包来实现(建议后者)，适用于相对固定的坐标系关系

2.动态坐标变换广播器，以编码的方式广播坐标系之间的相对关系，适用于易变的坐标系关系

3.坐标变换监听器，用于监听广播器广播的坐标系消息，可以实现不同坐标系之间或同一点在不同坐标系之间的变换

4.机器人系统中的坐标系关系是较为复杂的，还可以通过 tf2\_tools 工具包来生成 ros 中的坐标系关系图

5.当前 TF2 已经替换了 TF，官网建议直接学习 TF2，并且 TF 与 TF2 的使用流程与实现 API 比较类似，只要有任意一方的使用经验，另一方也可以做到触类旁通。