第5章

ROS 常用组件

在 ROS 中内置一些比较实用的工具，通过这些工具可以方便快捷地实现某个功能或调试程序，从而提高开发效率。本章主要介绍 ROS 中内置的如下组件：

- TF 坐标变换，实现不同类型的坐标系之间的转换。
- rosbag 用于录制 ROS 节点的执行过程并可以重放该过程。
- rqt 工具箱，集成了多款图形化的调试工具。

本章预期达成的学习目标如下：

- 了解 TF 坐标变换的概念以及应用场景。
- 能够独立完成 TF 案例：小乌龟跟随。
- 可以使用 rosbag 命令或编码的形式实现录制与回放。
- 能够熟练使用 rqt 中的图形化工具。

案例演示：

小乌龟跟随是 ROS 的内置案例。在终端上输入启动命令：

```
roslaunch turtle_tf2 turtle_tf2_demo_cpp.launch
```

或

```
roslaunch turtle_tf2 turtle_tf2_demo.launch
```

利用键盘可以控制一只乌龟运动，另一只乌龟跟随运动，如图 5-1 所示。

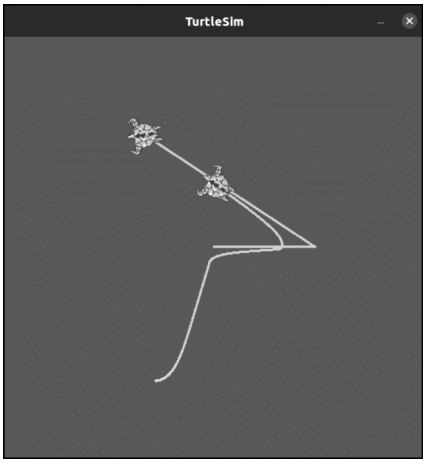

图 5-1　小乌龟跟随案例

5.1 TF 坐标变换

在机器人系统中有多个传感器，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，即横向、纵向和高度的距离信息）的，以协助机器人定位障碍物。可以直接将物体相对于该传感器的方位信息等价于物体相对于机器人系统或机器人其他组件的方位信息吗？显然是不行的，这中间需要一个转换过程。更具体的描述如下。

场景 1：雷达与小车

现有一个移动式机器人底盘，在底盘上安装了一个雷达，雷达相对于底盘的偏移量已知。现雷达检测到一个障碍物的信息，获取的坐标为（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？雷达与小车的坐标关系及坐标变换分别如图 5-2 和图 5-3 所示。

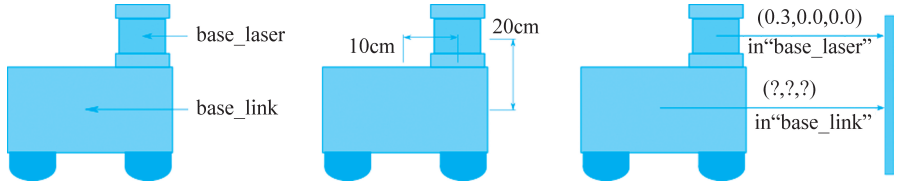

图 5-2 雷达与小车的坐标关系

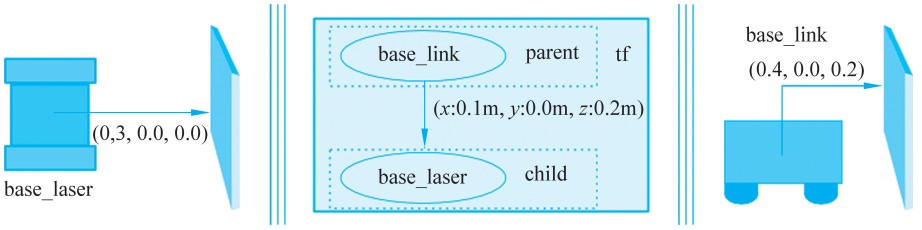

图 5-3 雷达与小车的坐标变换

场景 2：PR2

现有一个带机械臂的机器人 PR2（图 5-4）需要夹取目标物。当前机器人头部的摄像头可以探测到目标物的坐标（x，y，z），不过该坐标是以摄像头为参考系的，而实际操作目标物的是机械臂夹具。当前需要将该坐标转换成相对于机械臂夹具的坐标，这个过程如何实现？

当然，根据高中数学知识，在明确了不同坐标系之间的相对关系，就可以实现任何坐标点在不同坐标系之间的转换，但是该计算的实现是较为常用的，且算法也有点复杂，因此在 ROS 中直接封装了相关的模块——tf（坐标变换）。

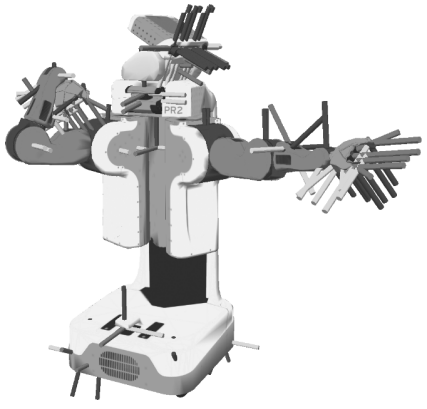

图 5-4 PR2 机器人

概念：

在 ROS 中是通过坐标系标定物体的，确切地说是通过右手坐标系标定的，如图 5-5 所示。

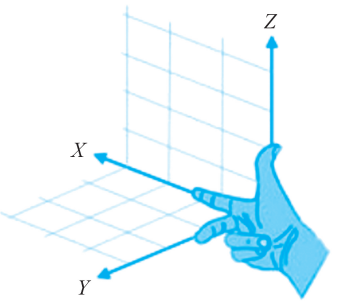

图 5-5　右手坐标系

tf 的作用是在 ROS 中实现不同坐标系之间的点或向量的转换。

案例：

小乌龟跟随案例

说明：

在 ROS 中，坐标变换最初对应的是 tf。不过从 hydro 版本开始，tf 被弃用，迁移到 tf2，后者更为简洁高效。tf2 的常用功能包如下：

- **tf2_geometry_msgs**：可以将 ROS 消息转换成 tf2 消息。
- **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。

5.1.1　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的实现中常用的 msg：geometry_msgs/TransformStamped 和 geometry_msgs/PointStamped，前者用于传输坐标系相关位置信息，后者用于传输某个坐标系内坐标点的信息。在坐标变换中，需要频繁地用到坐标系的相对关系以及坐标点信息。

1. geometry_msgs/TransformStamped

在命令行输入

```
rosmsg info geometry_msgs/TransformStamped
```

输出如下：

```
std_msgs/Header header                          #头信息
  uint32 seq                                    #序列号
  time stamp                                    #时间戳
  string frame_id                               #坐标系 ID
string child_frame_id                           #子坐标系 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 方向的偏移量
  geometry_msgs/Quaternion rotation             #四元数
    float64 x
    float64 y
    float64 z
    float64 w
```

四元数用于表示坐标的相对姿态。

2. geometry_msgs/PointStamped

在命令行输入

```
rosmsg info geometry_msgs/PointStamped
```

输出如下：

```
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
  float64 y
  float64 z
```

5.1.2 静态坐标变换

静态坐标变换中的两个坐标系之间的相对位置是固定的。

需求描述

现有一个机器人模型，其核心构成包含主体与雷达，各对应一个坐标系，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的位移关系如下：X 方向为 0.2，Y 方向为 0.0，Z 方向为 0.5。当前，雷达检测到一个障碍物，在雷达坐标系中障碍物的坐标为（2.0，3.0，5.0）。求出该障碍物相对于主体的坐标。

结果演示

结果如图 5-6 所示。

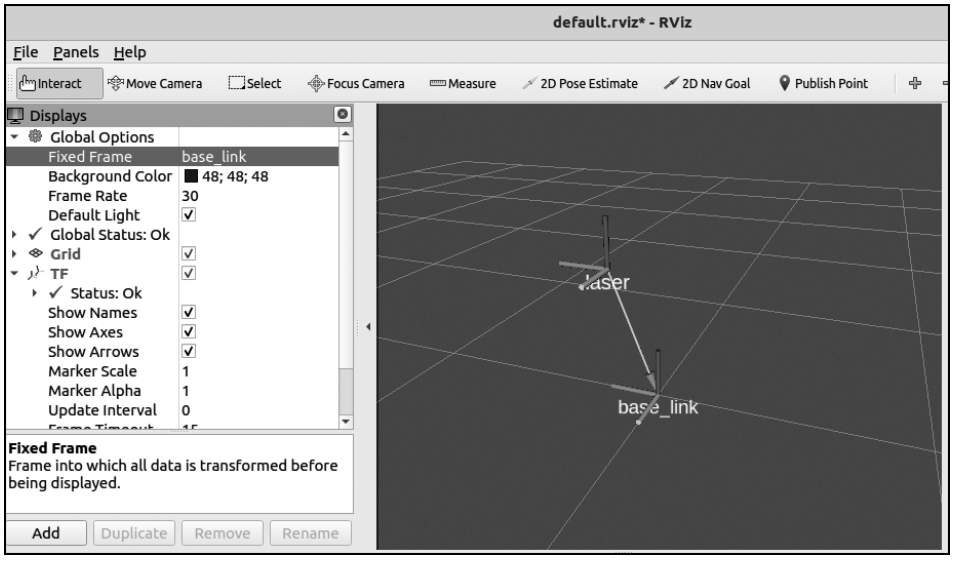

图 5-6 主体与雷达的坐标系关系

实现分析

坐标系相对关系可以通过发布方发布。

订阅方订阅到发布的坐标系相对关系，再传入坐标点信息（可以写死），然后借助于 tf 实现坐标变换，并将结果输出。

实现流程

C++ 与 Python 的实现流程一致：

(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。

方案 A(C++ 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。

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。

实现流程如下：

(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) 回旋。

```
//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 = "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);
    //6.回旋
    ros::spin();
    return 0;
}
```

配置文件此处略。

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) 回旋。

订阅方实现代码如下：

```
//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:",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;
}
```

配置文件此处略。

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。

方案 B(Python 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。

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) 导入功能包。

(2) 初始化 ROS 节点。

(3) 创建静态坐标广播器。

(4) 创建并组织被广播的消息。

(5) 广播器发送消息。

(6) 回旋。

发布方实现代码如下:

```
#! /usr/bin/env python
# 1.导入功能包
import rospy
import tf2_ros
import tf
```

```
from geometry_msgs.msg import TransformStamped
if __name__ == "__main__":
    # 2.初始化 ROS 节点
    rospy.init_node("static_tf_pub_p")
    # 3.创建静态坐标广播器
    broadcaster = tf2_ros.StaticTransformBroadcaster()
    # 4.创建并组织被广播的消息
    tfs = TransformStamped()
    # 头信息
    tfs.header.frame_id = "world"
    tfs.header.stamp = rospy.Time.now()
    tfs.header.seq = 101
    # 子坐标系
    tfs.child_frame_id = "radar"
    # 坐标系相对信息
    # 偏移量
    tfs.transform.translation.x = 0.2
    tfs.transform.translation.y = 0.0
    tfs.transform.translation.z = 0.5
    # 四元数
    qtn = tf.transformations.quaternion_from_euler(0,0,0)
    tfs.transform.rotation.x = qtn[0]
    tfs.transform.rotation.y = qtn[1]
    tfs.transform.rotation.z = qtn[2]
    tfs.transform.rotation.w = qtn[3]
    # 5.广播器发送消息
    broadcaster.sendTransform(tfs)
    # 6.回旋
    rospy.spin()
```

权限设置以及配置文件此处略。

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) 导入功能包。

(2) 初始化 ROS 节点。

(3) 创建 TF 订阅对象。

(4) 创建 radar 坐标系中的一个点。

(5) 调用订阅对象的 API，将(4)中的点的坐标转换成 world 坐标系的坐标。

(6) 回旋。

订阅方实现代码如下：

```
#! /usr/bin/env python
# 1.导入功能包
import rospy
import tf2_ros
# 不要使用 geometry_msgs,需要使用 tf2 内置的消息类型
from tf2_geometry_msgs import PointStamped
```

```
# from geometry_msgs.msg import PointStamped
if __name__ == "__main__":
    # 2.初始化 ROS 节点
    rospy.init_node("static_sub_tf_p")
    # 3.创建 TF 订阅对象
    buffer = tf2_ros.Buffer()
    listener = tf2_ros.TransformListener(buffer)
    rate = rospy.Rate(1)
    while not rospy.is_shutdown():
    # 4.创建 radar 坐标系中的一个点
        point_source = PointStamped()
        point_source.header.frame_id = "radar"
        point_source.header.stamp = rospy.Time.now()
        point_source.point.x = 10
        point_source.point.y = 2
        point_source.point.z = 3
        try:
        #5.调用订阅对象的 API,将(4)中的点的坐标转换成 world 坐标系的坐标
            point_target = buffer.transform(point_source,"world")
            rospy.loginfo("转换结果:x = %.2f, y = %.2f, z = %.2f",
                          point_target.point.x,
                          point_target.point.y,
                          point_target.point.z)
        except Exception as e:
            rospy.logerr("异常:%s",e)
        #6.回旋
        rate.sleep()
```

权限设置以及配置文件此处略。

提示：在 tf2 的 Python 实现中，tf2 已经封装了一些消息类型，不可以使用 geometry_msgs.msg 中的类型。

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。

补充 1

当坐标系之间的相对位置固定时，所需参数也是固定的，包括父系坐标名称、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。而当实现逻辑相同但参数不同时，可以使用 ROS 系统已经封装好的专门节点，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。

补充 2

可以借助于 RViz 显示坐标系之间的关系，具体操作如下：

(1) 新建窗口，输入命令 rviz。

(2) 在启动的 RViz 中设置 Fixed Frame 为 base_link。

(3) 单击左下角的 add 按钮,在弹出的窗口中选择 TF 组件,即可显示坐标关系。

5.1.3 动态坐标变换

动态坐标变换中的两个坐标系之间的相对位置是变化的。

需求描述

启动 turtlesim_node 节点,该节点中的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,动态发布两个坐标系的相对位置。

结果演示

结果如图 5-7 所示。

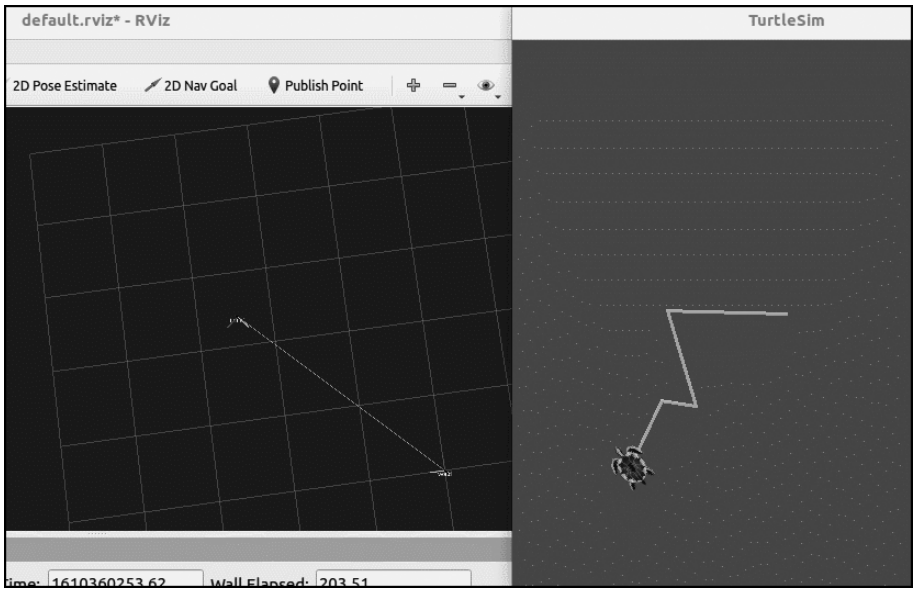

图 5-7 世界坐标系和乌龟坐标系的动态坐标变换

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中的 X 坐标、Y 坐标、偏移量、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。

实现流程

C++ 与 Python 的实现流程一致:

(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。

方案 A(C++ 实现):