第5章

ROS2 工具之坐标变换

本章导论


ROS2 工具之坐标变换_引言

在机器人系统中，会经常性地使用“相对位置关系”这一概念，例如机器人自身不同部件的相对位置关系，机器人与出发点的相对位置关系，传感器与障碍物的相对位置关系，机器人组队中不同机器人之间的相对位置关系等。毋庸置疑，相对位置关系在机器人系统中有着重要的意义，那么在 ROS2 中如何表述、使用相对位置关系呢？本章将会给出答案。

5.1 坐标变换简介

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（以坐标的方式表示物体与传感器的横向距离、纵向距离、垂直高度等信息）的，以协助机器人定位障碍物，我们可以直接将物体相对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。

1. 坐标变换的适用场景


坐标变换_简介 01_场景

场景 1：现有一移动式机器人底盘，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？在此大背景下，便诞生了 ROS。ROS 是一套机器人通用软件框架，可以提升功能模块的复用性，并且随着 ROS2 的推出，ROS 日臻完善，是机器人软件开发的不二之选，如图 5-1 所示为 tf 坐标。

场景 2：现有一带机械臂的机器人（比如 PR2）需要夹取目标物，当前机器人头部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的夹具，当前我们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？如图 5-2 的 PR2 机械臂所示。


坐标变换_简介 02_概念与作用

当然，根据我们学习的知识，在明确了不同坐标系之间的相对关系时，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：坐标变换（tf）。

2. 坐标变换的概念

TF（TransForm Frame）是指坐标变换，它允许用户随时间跟踪多个坐标系。

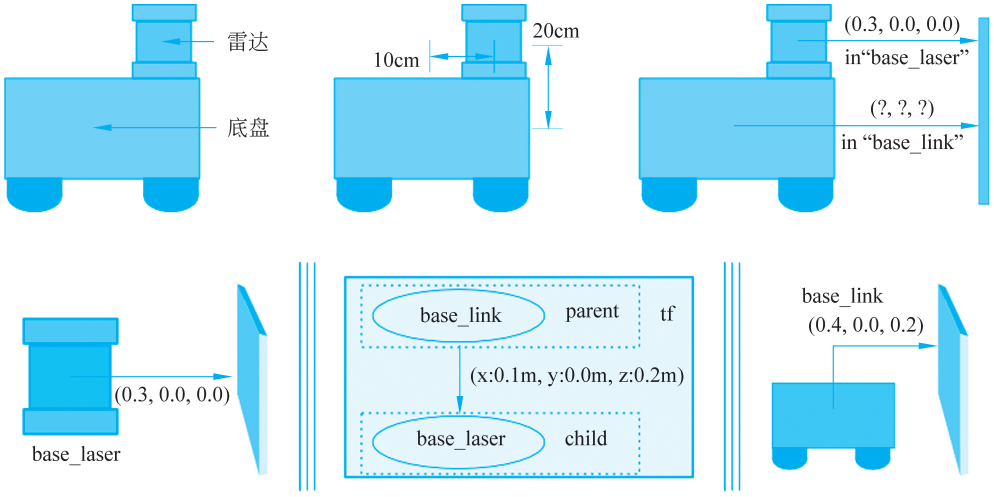

图 5-1　tf 坐标

它在时间缓冲的树结构中维护坐标帧之间的关系，并让用户在任何所需的时间点在任意两个坐标帧之间变换点、向量等。在 ROS 中已经提供了同名的库实现，并且随着 ROS 的迭代，该库升级为了 tf2，也即第二代坐标变换库。本阶段课程主要内容也是以 tf2 为主。

完整的坐标变换实现由坐标变换广播方和坐标变换监听方两部分组成。每个坐标变换广播方一般会发布一组坐标系相对关系，而坐标变换监听方则会将多组坐标系相对关系融合为一棵坐标树（该坐标树有且仅有一个根坐标系），并可以实现任意坐标系之间或坐标点与坐标系的变换。

另外需要说明的是，ROS 中的坐标变换是基于右手坐标系的。右手坐标系的具体规则如图 5-3 所示：将右手处于坐标系原点，大拇指、食指与中指互成直角，食指指向的是 x 轴正方向，中指指向的是 y 轴正方向，大拇指指向的是 z 轴正方向。

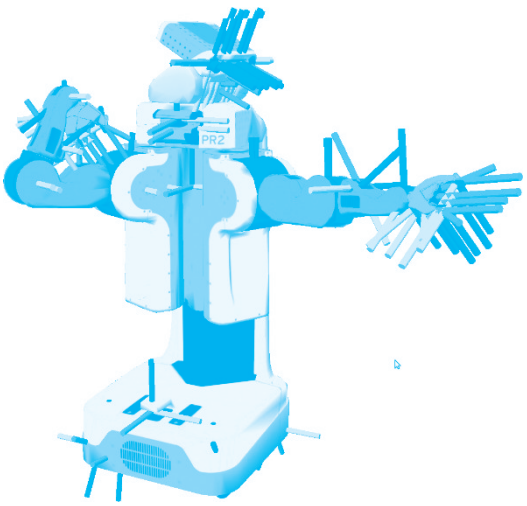

图 5-2　PR2 机械臂

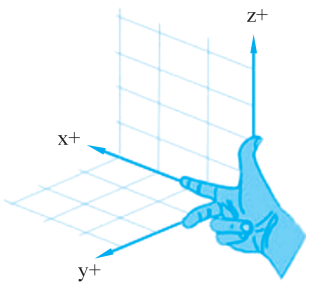

图 5-3　右手坐标系

3. 坐标变换的作用

在 ROS 中用于实现不同坐标系之间的点或向量的转换。

关于坐标变换的实现有一个经典的“乌龟跟随”案例，在学习坐标变换的具体知识点之前，建议读者先安装并运行此案例。

(1) “乌龟跟随”案例的安装

首先安装“乌龟跟随”案例的功能包以及依赖项。

- 安装方式 1(二进制方式安装)：


坐标变换_案例安装以及运行

```
sudo apt-get install ros-humble-turtle-tf2-py ros-humble-tf2-tools ros-humble-tf-transformations
```

- 安装方式 2(克隆源码并构建)：

```
git clone https://github.com/ros/geometry_tutorials.git -b ros2
```

此外，还需要安装一个名为 transforms3d 的 Python 包，它为 tf_transformations 包提供四元数和欧拉角变换功能，安装命令如下：

```
sudo apt intall python3-pip
pip3 install transforms3d
```

(2) “乌龟跟随”案例的执行

启动两个终端，终端 1 输入如下命令：

```
ros2 launch turtle_tf2_py turtle_tf2_demo.launch.py
```

该命令会启动 turtlesim_node 节点，turtlesim_node 节点中自带一只小乌龟 turtle1，除此之外还会新生成一只乌龟 turtle2，turtle2 会运行至 turtle1 的位置。

终端 2 输入如下命令：

```
ros2 run turtlesim turtle_teleop_key
```

该终端下可以通过键盘控制 turtle1 运动，并且 turtle2 会跟随 turtle1 运动。

5.2　坐标相关消息


坐标变换相关消息

坐标变换的实现其本质是基于话题通信的发布订阅模型，发布方可以发布坐标系之间的相对关系，订阅方则可以监听这些消息，并实现不同坐标系之间的变换。根据之前的介绍，在话题通信中，接口消息作为数据载体在整个通信模型中是比较重要的一部分，本节将会介绍坐标变换中常用的两种接口消息：geometry_msgs/msg/TransformStamped 和 geometry_msgs/msg/PointStamped。

前者用于描述某一时刻两个坐标系之间相对关系的接口，后者用于描述某一时刻坐标系内某个坐标点的位置的接口。在坐标变换中，会经常性地使用到坐标系相对关系以及坐标点信息。

1. 接口消息 geometry_msgs/msg/TransformStamped

通过如下命令查看接口定义：

```
ros2 interface show geometry_msgs/msg/TransformStamped
```

接口定义解释如下：

```
std_msgs/Header header
    builtin_interfaces/Time stamp          #时间戳
        int32 sec
        uint32 nanosec
    string frame_id                        #父级坐标系

string child_frame_id                      #子级坐标系

Transform transform                        #子级坐标系相对于父级坐标系的位姿
    Vector3 translation                    #三维偏移量
        float64 x
        float64 y
        float64 z
    Quaternion rotation                    #四元数
        float64 x 0
        float64 y 0
        float64 z 0
        float64 w 1
```

四元数类似于欧拉角用于表示坐标系的相对姿态。

2. 接口消息 geometry_msgs/msg/PointStamped

通过如下命令查看接口定义：

```
ros2 interface show geometry_msgs/msg/PointStamped
```

接口定义解释如下：

```
std_msgs/Header header
    builtin_interfaces/Time stamp          #时间戳
        int32 sec
        uint32 nanosec
    string frame_id                        #参考系
Point point                                #三维坐标
    float64 x
    float64 y
    float64 z
```

坐标变换广播_引言

5.3 坐标变换广播

坐标系相对关系主要有两种：静态坐标系相对关系与动态坐标系相对关系。

所谓静态坐标系相对关系是指两个坐标系之间的相对位置是固定不变的，例如，车辆上的雷达、摄像头等组件一般是固定式的，那么雷达坐标系相对于车辆底盘坐标系或摄像头坐

标系相对于车辆底盘坐标系就是一种静态关系。

所谓动态坐标系相对关系是指两个坐标系之间的相对位置关系是动态改变的，例如，车辆上机械臂的关节或夹爪、多车编队中不同车辆等都是可以运动的，那么机械臂的关节或夹爪坐标系相对车辆底盘坐标系或不同车辆坐标系的相对关系就是一种动态关系。

本节主要介绍如何实现静态坐标变换广播与动态坐标变换广播。另外，本节还将演示如何发布坐标点消息。

5.3.1　坐标系广播案例及分析

坐标变换广播案例以及分析

1. 坐标系广播案例需求

案例 1：现有一无人车，在无人车底盘上装有固定式的雷达与摄像头，已知车辆底盘、雷达与摄像头各对应一坐标系，如图 5-4 所示，各坐标系的原点取其几何中心。现又已知雷达坐标系相对于底盘坐标系的三维平移量分别为：x 方向 0.4 米，y 方向 0 米，z 方向 0.2 米，无旋转。摄像头坐标系相对于底盘坐标系的三维平移量分别为：x 方向 −0.5 米，y 方向 0 米，z 方向 0.4 米，无旋转。请广播雷达与底盘的坐标系相对关系，摄像头与底盘的坐标系相对关系，并在 rviz2 中查看广播的结果。

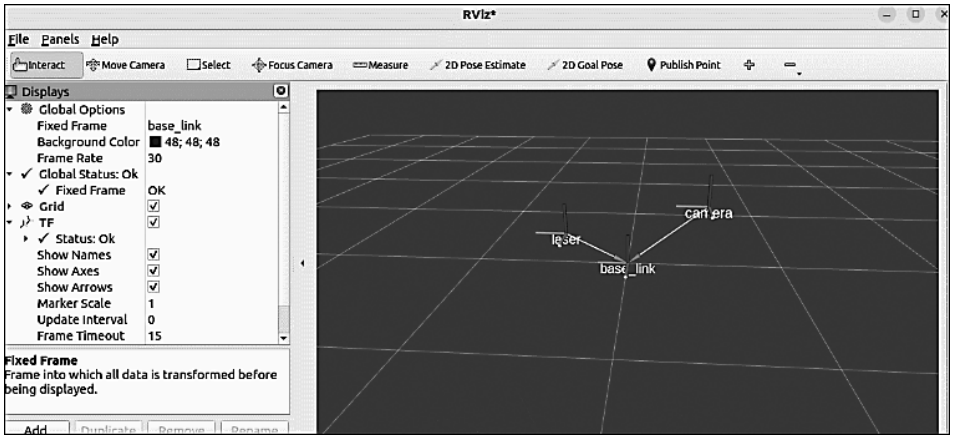

图 5-4　无人车底盘、雷达和摄像头坐标系

案例 2：启动 turtlesim_node，设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，乌龟可以通过键盘控制运动，请动态发布乌龟坐标系与世界坐标系的相对关系，如图 5-5 所示。

2. 坐标系广播案例分析

在上述案例中，案例 1 需要使用静态坐标变换，案例 2 则需要使用动态坐标变换，无论何种实现关注的要素都有以下两个：

（1）如何广播坐标系相对关系。

（2）如何使用 rviz2 显示坐标系相对关系。

3. 坐标系广播流程简介

与编码实现静态或动态坐标变换的流程类似，主要步骤如下：

（1）编写广播实现。

（2）编辑配置文件。

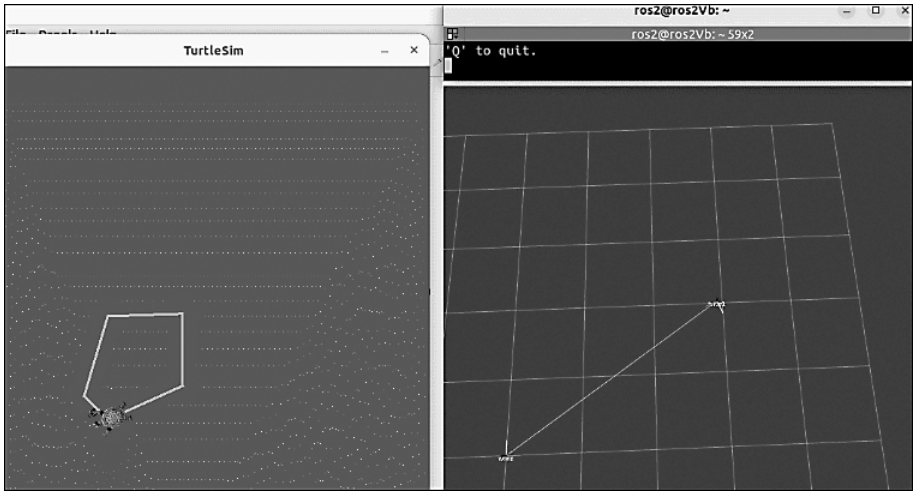

图 5-5　乌龟坐标系与世界坐标系的相对关系

（3）编译。

（4）执行。

（5）在 rviz2 中查看坐标系关系。

案例我们会采用 C++ 和 Python 分别实现，二者都遵循上述实现流程。

另外需要说明的是，静态广播器除了可以以编码的方式实现外，在 tf2 中还内置了相关工具，可以无须编码，直接执行节点并传入表示坐标系相对关系的参数，即可实现静态坐标系关系的发布。而动态广播器没有提供类似的工具。

4. 坐标系广播的准备工作

终端下进入工作空间的 src 目录，调用如下两条命令分别创建 C++ 功能包和 Python 功能包。

```
ros2 pkg create cpp03_tf_broadcaster --build-type ament_cmake --dependencies
rclcpp tf2 tf2_ros geometry_msgs turtlesim
ros2 pkg create py03_tf_broadcaster --build-type ament_python --dependencies
rclpy tf_transformations tf2_ros geometry_msgs turtlesim
```

静态广播器_命令实现（上）

5.3.2　静态广播器（命令）

1. 静态广播器工具

在 tf2_ros 功能包中提供了一个名为 static_transform_publisher 的可执行文件，通过该文件可以直接广播静态坐标系关系，其使用语法如下。

格式 1：

使用以米为单位的 x/y/z 偏移量和以弧度为单位的 roll/pitch/yaw（可直译为滚动/俯仰/偏航，分别指的是围绕 x/y/z 轴的旋转）向 tf2 发布静态坐标变换：

```
ros2 run tf2_ros static_transform_publisher --x x --y y --z z --yaw yaw --pitch
pitch --roll roll --frame-id frame_id --child-frame-id child_frame_id
```

格式 2：

使用以米为单位的 x/y/z 偏移量和 qx/qy/qz/qw 四元数向 tf2 发布静态坐标变换：

```
ros2 run tf2_ros static_transform_publisher --x x --y y --z z --qx qx --qy qy --
qz qz --qw qw --frame-id frame_id --child-frame-id child_frame_id
```

注意：在上述两种格式中除了用于表示父级坐标系的--frame-id 和用于表示子级坐标系的--child-frame-id 之外，其他参数都是可选的，如果未指定特定选项，将直接使用默认值。

2. 静态广播器工具的使用


静态广播器_命令实现（下）

打开两个终端，终端 1 输入如下命令发布雷达（laser）相对于底盘（base_link）的静态坐标变换：

```
ros2 run tf2_ros static_transform_publisher --x 0.4 --y 0 --z 0.2 --yaw 0 --roll
0 --pitch 0 --frame-id base_link --child-frame-id laser
```

终端 2 输入如下命令发布摄像头（camera）相对于底盘（base_link）的静态坐标变换：

```
ros2 run tf2_ros static_transform_publisher --x -0.5 --y 0 --z 0.4 --yaw 0 --
roll 0 --pitch 0 --frame-id base_link --child-frame-id camera
```

3. rviz2 查看坐标系关系

新建终端，通过命令 rviz2 打开 rviz2 并配置相关插件查看坐标变换消息：

（1）将 Global Options 中的 Fixed Frame 设置为 base_link；

（2）单击 Add 按钮添加 TF 插件。

右侧 Grid 中将以图形化的方式显示坐标变换关系，如图 5-6 所示。

图 5-6　以图形化的方式显示坐标变换关系

5.3.3　静态广播器（C++）


静态广播器_C++实现 01_流程简介

1. 静态广播器（C++）的广播实现

在功能包 cpp03_tf_broadcaster 的 src 目录下，新建 C++ 文件 demo01_static_tf_broadcaster.

cpp,并编辑文件,输入如下内容:


静态广播器_C++ 实现 02_框架搭建

```
/*
  需求:编写静态坐标变换程序,执行时传入两个坐标系的相对位姿关系以及父子级坐标系 id,
       程序运行发布静态坐标变换。
  步骤:
    1.包含头文件
    2.判断终端传入的参数是否合法
    3.初始化 ROS2 客户端
    4.定义节点类
      4-1.创建静态坐标变换发布方
      4-2.组织并发布消息
    5.调用 spin 函数,并传入对象指针
    6.释放资源

*/

//1.包含头文件
#include <geometry_msgs/msg/transform_stamped.hpp>

#include <rclcpp/rclcpp.hpp>
#include <tf2/LinearMath/Quaternion.h>
#include <tf2_ros/static_transform_broadcaster.h>

using std::placeholders::_1;

//4.定义节点类
class MinimalStaticFrameBroadcaster : public rclcpp::Node
{
public:
  explicit MinimalStaticFrameBroadcaster(char * transformation[]): Node("
minimal_static_frame_broadcaster")
  {
    //4-1.创建静态坐标变换发布方
    tf_publisher_ = std::make_shared<tf2_ros::StaticTransformBroadcaster>(this);

    this->make_transforms(transformation);
  }

private:
  //4-2.组织并发布消息
  void make_transforms(char * transformation[])
  {
    //组织消息
    geometry_msgs::msg::TransformStamped t;

    rclcpp::Time now = this->get_clock()->now();
    t.header.stamp = now;
    t.header.frame_id = transformation[7];
    t.child_frame_id = transformation[8];
```


静态广播器_C++ 实现 03_广播实现

```
    t.transform.translation.x = atof(transformation[1]);
    t.transform.translation.y = atof(transformation[2]);
    t.transform.translation.z = atof(transformation[3]);
    tf2::Quaternion q;
    q.setRPY(
      atof(transformation[4]),
      atof(transformation[5]),
      atof(transformation[6]));
    t.transform.rotation.x = q.x();
    t.transform.rotation.y = q.y();
    t.transform.rotation.z = q.z();
    t.transform.rotation.w = q.w();

    //发布消息
    tf_publisher_->sendTransform(t);
  }
  std::shared_ptr<tf2_ros::StaticTransformBroadcaster> tf_publisher_;
};

int main(int argc, char * argv[])
{
  //2.判断终端传入的参数是否合法
  auto logger = rclcpp::get_logger("logger");

  if (argc != 9) {
    RCLCPP_INFO(
      logger, "运行程序时请按照:x y z roll pitch yaw frame_id child_frame_id 的格式
传入参数");
    return 1;
  }

  //3.初始化 ROS2 客户端
  rclcpp::init(argc, argv);
  //5.调用 spin 函数,并传入对象指针
  rclcpp::spin(std::make_shared<MinimalStaticFrameBroadcaster>(argv));
  //6.释放资源
  rclcpp::shutdown();
  return 0;
}
```

2. 编辑静态广播器(C++)的配置文件

(1) 编辑 package.xml

在创建功能包时,所依赖的功能包已经自动配置了,配置内容如下:

静态广播器_C++ 实现 04_补充

```
<depend>rclcpp</depend>
<depend>tf2</depend>
<depend>tf2_ros</depend>
<depend>geometry_msgs</depend>
<depend>turtlesim</depend>
```

(2) 编辑 CMakeLists.txt

CMakeLists.txt 中发布和订阅程序核心配置如下：

```
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(tf2 REQUIRED)
find_package(tf2_ros REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(turtlesim REQUIRED)

add_executable(demo01_static_tf_broadcaster src/demo01_static_tf_broadcaster.cpp)
ament_target_dependencies(
  demo01_static_tf_broadcaster
  "rclcpp"
  "tf2"
  "tf2_ros"
  "geometry_msgs"
  "turtlesim"
)

install(TARGETS demo01_static_tf_broadcaster
  DESTINATION lib/${PROJECT_NAME})
```

3. 编译

终端下进入当前工作空间，编译功能包：

```
colcon build --packages-select cpp03_tf_broadcaster
```

4. 执行

当前工作空间下，启动两个终端，终端 1 输入如下命令发布雷达(laser)相对于底盘(base_link)的静态坐标变换：

```
. install/setup.bash
ros2 run cpp03_tf_broadcaster demo01_static_tf_broadcaster 0.4 0 0.2 0 0 0 base_
link laser
```

终端 2 输入如下命令发布摄像头(camera)相对于底盘(base_link)的静态坐标变换：

```
. install/setup.bash
ros2 run cpp03_tf_broadcaster demo01_static_tf_broadcaster -0.5 0 0.4 0 0 0 base_
link camera
```

5. rviz2 查看坐标系关系

参考 5.3.2 节静态广播器(命令)内容启动并配置 rviz2，最终执行结果与案例 1 类似。

静态广播器_Python 实现 01_框架搭建

5.3.4 静态广播器(Python)

1. 静态广播器(Python)的广播实现

在功能包 py03_tf_broadcaster 的 py03_tf_broadcaster 目录下，新建 Python 文件