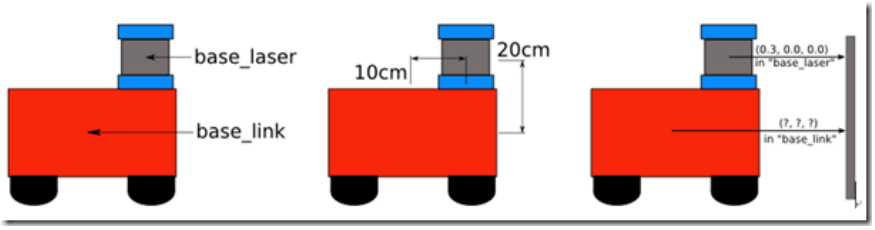
第十四课 机器人 tf 转换和 tf 树

1. 什么是 tf

tf 是一个让用户随时间跟踪多个参考系的功能包，它使用一种树型数据结构，根据时间缓冲并维护多个参考系之间的坐标变换关系，可以帮助用户在任意时间，将点、向量等数据的坐标，在两个参考系中完成坐标变换。一个机器人系统通常有很多三维的参考系，而且会随着时间的推移发生变化，例如全局参考系（odom），机器人中心参考系（base_footprint），姿态传感器参考系（gyro_link），地图参考系（map），激光雷达参考系（laser）等等。tf 可以以时间为轴，跟踪这些参考系。

tf 可以在分布式系统中进行操作，也就是说一个机器人系统中所有的参考系变换关系，对于所有节点组件，都是可用的，所有订阅 tf 消息的节点都会缓冲一份所有参考系的变换关系数据，所以这种结构不需要中心服务器来存储任何数据。

2. tf 的使用流程



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，这些数据当然是以激光雷达为原点的测量值，换句话说，也就是 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。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上，在极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。这个时候，如果我们采用一种坐标变换，将及激光数据从 laser 参考系变换到 base_footprint 参考下。

参考系之间的坐标变换并不复杂，但是在复杂的系统中，存在的参考系可能有很多个，如果我们都通过自己计算的方式进行变换，那么在代码中很不好维护，而且代码量也会随着参考系增加。ROS 提供的 tf 变换就是为解决这个问题而生的，tf 功能包提供了不同参考系之间变换的功能，我们只需要告诉 tf 树这些参考系之间的变换公式 tf 会自动管理我们所需要的参考系变换。

同时我们知道激光雷达和姿态传感器都是固定在机器人本体上的，他们和机器人中心之间的位置是永远固定的，不会随着机器人运动而变化。所以 tf 提供了一种静态发布固定 tf 转换的功能。下面的的代码片段是 HiBot 启动文件中的静态 tf 转换。

```
<node pkg="tf" type="static_transform_publisher" name="base_to_link" args="0 0 0 0 0 0 base_footprint base_link 100" />
<node pkg="tf" type="static_transform_publisher" name="base_to_gyro" args="0 0 0 0 0 0 base_footprint gyro_link 100" />
<node pkg="tf" type="static_transform_publisher" name="base_to_laser" args="0.05 0 0.1 0 0 1.0 6.12323399574e-17 base_footprint laser 100" />
```

3. tf 广播（C++）

下面我们将创建一个 tf 广播，时刻发布坐标系之间的 tf 转换。首先需要为此项目创建一个新的 ros 包。，创建一个名为 learning_tf 的包，创建的新软件包依赖于 tf、roscpp、rospy 和 turtlesim。

```
huanyu@ubuntu:~$ cd robot_ws/src/
huanyu@ubuntu:~/robot_ws/src$ catkin_create_pkg learning_tf tf roscpp rospy turt
lesim
Created file learning_tf/package.xml
Created file learning_tf/CMakeLists.txt
Created folder learning_tf/include/learning_tf
Created folder learning_tf/src
Successfully created files in /home/huanyu/robot_ws/src/learning_tf. Please adju
st the values in package.xml.
```

接下来我们创建 cpp 文件，用代码实现 tf 广播功能。

```
huanyu@ubuntu:~$ rospack profile
```

```
huanyu@ubuntu:~$ roscd learning_tf/src/
huanyu@ubuntu:~/robot_ws/src/learning_tf/src$ touch turtle_tf_broadcaster.cpp
huanyu@ubuntu:~/robot_ws/src/learning_tf/src$ ls
turtle_tf_broadcaster.cpp
huanyu@ubuntu:~/robot_ws/src/learning_tf/src$
```

开始编辑 turtle_tf_broadcaster.cpp 文件，添加代码段。

```
#include "ros/ros.h"
#include "std_msgs/Float32.h"
#include <tf/transform_broadcaster.h>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "tf_transform_broadcaster");
    ros::NodeHandle n;

    std_msgs::Float32 x, y;
    tf::TransformBroadcaster odom_broadcaster;
    geometry_msgs::Quaternion odom_quat;
    geometry_msgs::TransformStamped odom_trans;

    ros::Rate loop_rate(2);
    while (ros::ok())
    {
        odom_quat = tf::createQuaternionMsgFromYaw(0.0);

        odom_trans.header.stamp = ros::Time::now();;
        odom_trans.header.frame_id = "odom";
        odom_trans.child_frame_id = "base_link";

        odom_trans.transform.translation.x = x.data;
        odom_trans.transform.translation.y = y.data;
        odom_trans.transform.translation.z = 0.0;
        odom_trans.transform.rotation = odom_quat;
        //send the transform
        odom_broadcaster.sendTransform(odom_trans);
        ROS_INFO("tf Transform Broadcaster ok!");


        x.data += 0.1;
        y.data += 0.05;

        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

请将以上代码输入到文件中（我们也提供了相关的源码包），让我们先编译它。 打开 CMakeLists.txt 文件，并在底部添加编译和链接语句，确保 cmake 正常解析。

```
add_executable(turtle_tf_broadcaster src/turtle_tf_broadcaster.cpp)
target_link_libraries(turtle_tf_broadcaster ${catkin_LIBRARIES})
```

做好以上步骤后，就可以开始编译 ROS 节点。

```
[100%] Linking CXX executable /home/huanyu/robot_ws/devel/lib/learning_tf/turtle
_tf_broadcaster
[100%] Built target turtle_tf_broadcaster
huanyu@ubuntu:~/robot_ws$ catkin_make
```

测试 tf 广播，首先打开新的终端运行 roscore

```
huanyu@ubuntu:~$ roscore
```

然后运行 tf 广播代码，会在终端看到 turtle_tf_broadcaster 节点会以 2HZ 的速度发布 odom->base_link 之间的 tf 广播。同样我们可以使用 rviz 或者 rqt_tf_tree 查看两个坐标系的变化。

```
huanyu@ubuntu:~/robot_ws$ rosrun learning_tf turtle_tf_broadcaster
[ INFO] [1558419875.758211495]: tf Transform Broadcaster ok!
[ INFO] [1558419876.259039960]: tf Transform Broadcaster ok!
[ INFO] [1558419876.758686846]: tf Transform Broadcaster ok!
[ INFO] [1558419877.258728434]: tf Transform Broadcaster ok!
```

4.tf 监听器（C++）

上述过程已经成功编写和编译了 tf 广播的节点，下一步我们开始编译监听器节点。

一个功能包中可以同时存在多个节点，所以我们只需要在创建好的 learning_tf 包中新建监听节点。

```
huanyu@ubuntu:~/robot_ws$ roscd learning_tf/src/
huanyu@ubuntu:~/robot_ws/src/learning_tf/src$ touch turtle_tf_listener.cpp
huanyu@ubuntu:~/robot_ws/src/learning_tf/src$ ls
turtle_tf_broadcaster.cpp  turtle_tf_listener.cpp
huanyu@ubuntu:~/robot_ws/src/learning_tf/src$
```

打开新建的文件，添加以下代码。

```
#include "ros/ros.h"
#include "std_msgs/Float32.h"
#include <tf/transform_listener.h>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "tf_transform_listener");
    ros::NodeHandle n;

    tf::TransformListener listener;
    tf::StampedTransform transform;

    ros::Rate loop_rate(2);
    while (ros::ok())
    {
        try{
            listener.lookupTransform("/odom", "/base_link", ros::Time(0), transform);
        }
        catch (tf::TransformException ex)
        {
            ROS_ERROR("%s",ex.what());
            ros::Duration(1.0).sleep();
        }
        ROS_INFO("Llstener <odom -base_link> = [%f, %f]",transform.getOrigin().x(),transform.getOrigin().y());

        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

打开 CMakeLists.txt 文件，并在底部添加编译和链接语句，确保 cmake 正常解析。

```
add_executable(turtle_tf_listener src/turtle_tf_listener.cpp)
target_link_libraries(turtle_tf_listener ${catkin_LIBRARIES})
```

使用 catkin_make 编译刚创建的监听节点。

```
huanyu@ubuntu:~/robot_ws$ catkin_make
```

4. 测试 tf 广播和监听节点

1> 打开新的终端运行 roscore

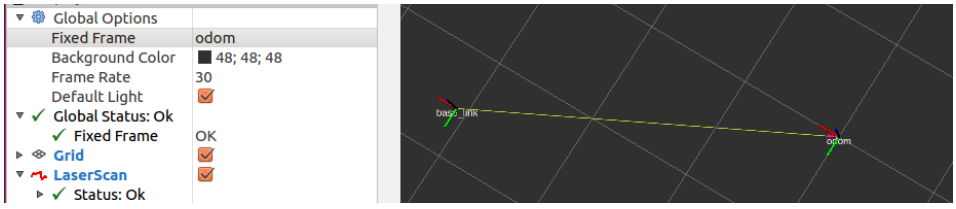
2> 打开新的终端运行 rosrun learning_tf turtle_tf_broadcaster

3> 打开新的终端运行 rosrun learning_tf turtle_tf_listener

然后再运行监听节点的终端中可以看到如下 tf 坐标系推移过程。

```
[ INFO] [1558421576.011374775]: Llstener <odom -base_link> = [0.500000, 0.250000]
[ INFO] [1558421576.511572341]: Llstener <odom -base_link> = [0.600000, 0.300000]
[ INFO] [1558421577.012490634]: Llstener <odom -base_link> = [0.700000, 0.350000]
[ INFO] [1558421577.512413788]: Llstener <odom -base_link> = [0.800000, 0.400000]
[ INFO] [1558421578.011656991]: Llstener <odom -base_link> = [0.900000, 0.450000]
[ INFO] [1558421578.512822907]: Llstener <odom -base_link> = [1.000000, 0.500000]
```

打开 rviz，基础参考系选择 Odom，然后就可以看到 base_link 坐标系随着时间运动。



5. 总结

本节课程学了了 ROS tf 库的使用，主要是利用 tf 功能包进行坐标系之间的 tf 广播和 tf 参考系的变化监听。深度掌握 tf 是机器人导航建图的前提。