ROS机器人创新实践

第三讲 机器人仿真

高斌

目录/Contents

01 机器人定义与组成

1.机器人的定义与组成

- 机器人问世已有几十年，机器人的定义仍然仁者见仁，智者见智，没有一个统一的意见。原因之一是机器人还在发展，新的机型，新的功能不断涌现。同时由于机器人涉及到了人的概念，成为一个难以回答的哲学问题。就像机器人一词最早诞生于科幻小说之中一样，人们对机器人充满了幻想。也许正是由于机器人定义的模糊，才给了人们充分的想象和创造空间。

- 随着机器人技术的飞速发展和信息时代的到来,机器人所涵盖的内容越来越丰富，机器人的定义也不断充实和创新。下面给出一些有代表性的定义。

1.机器人的定义与组成

- 按用途分：工业、军事、 探索、服务、 娱乐
- 按完成的主要功能分：操作机器人，移动机器人，信息机器人，人机机器人
- 按技术级别分：示教再现机器人带感觉的机器人、智能机器人
- 按执行机构驱动方式分：液压，气动，电气驱动
- 按受控方式分：非伺服，伺服（点位伺服，连续路径）
- 按信息输入方式分：手控，定序，变序，程控，智能
- 按手臂运动坐标形式：直角，园柱坐标，极坐标，多关节

1.机器人的定义与组成

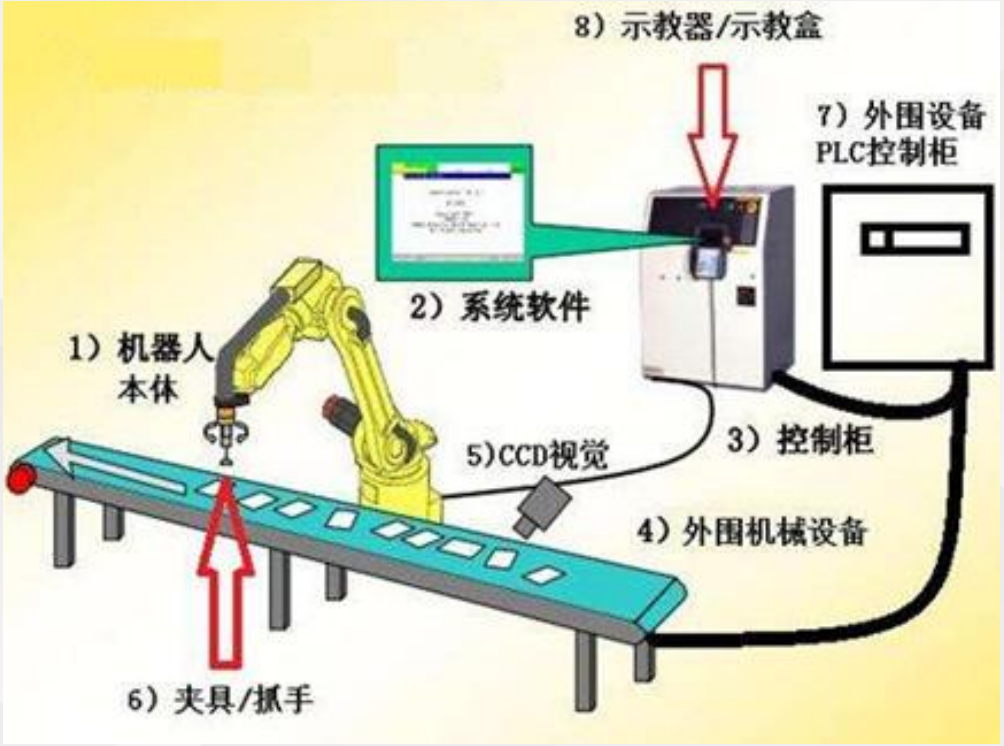
按照从低级→高级的发展程度可分为三类机器人

- **第一代机器人**（First Generation Robots）：即可编程、示教再现工业机器人机器人，已进入商品化、实用化。
- **第二代机器人**（Second Generation Robots）：装备有一定的传感装置，能获取作业环境、操作对象的简单信息，通过计算机处理、分析，能作出简单的推理，对动作进行反馈的机器人，通常称为低级智能机器人，由于信息处理系统的庞大与昂贵，第二代机器人目前只有少数可投入应用。
- **第三代机器人**（Third Generation Robots）：具有高度适应性的自治机器人。它具有多种感知功能，可进行复杂的逻辑思维、判断决策，在作业环境中独立行动。第三代机器人又称作高级智能机器人，它与第五代计算机关系密切，目前还处于研究阶段。

1.机器人的定义与组成

作业系统组成

- 机器人本体
- 控制柜
- 示教盒
- 系统软件
- 夹具/抓手
- 外传感器（如视觉、力觉）
- 外围设备



1.机器人的定义与组成

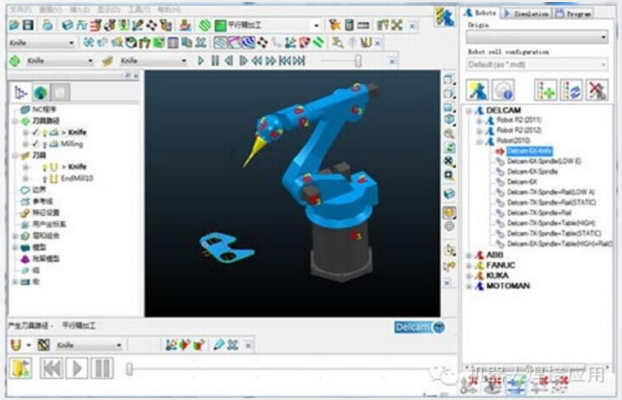
通用工业机器人系统

- 机器人本体
- 控制柜
- 示教盒
- 系统软件

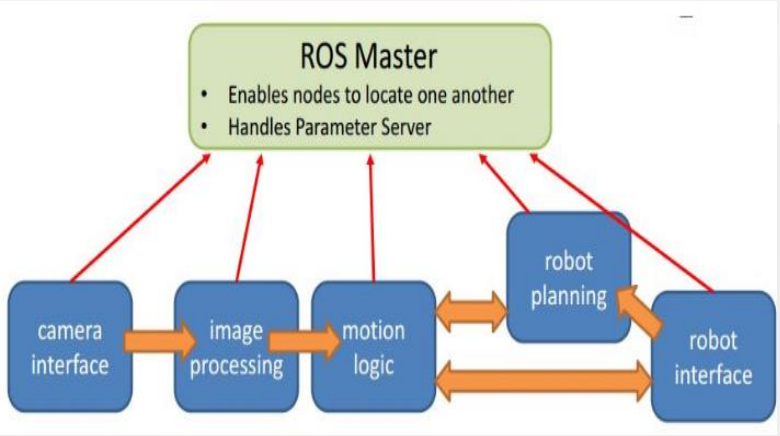
机器人本体

控制柜

示教盒



机器人系统软件



机器人操作系统ROS (Robot Operation System)

1.机器人的定义与组成

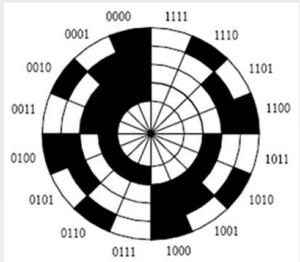
(1)机器人本体

- 伺服电机
- 减速器
- 连接件
- 传感器（内传感器，如编码器）

机器人本体

伺服电机

减速器



编码器

02 ROS坐标变换

2.ROS坐标变换

机器人没法做到像人类一样自如的进行各种行为操作．那么在这个过程中，TF又扮演着什么样的角色呢？还拿该图来说，当机器人的＂眼睛＂获取一组数据，关于物体的坐标方位，但是相对于机器人手臂来说，这个坐标只是相对于机器人头部的传感器，并不直接适用于机器人手臂执行，那么物体相对于头部和手臂之间的坐标转换，就是TF.

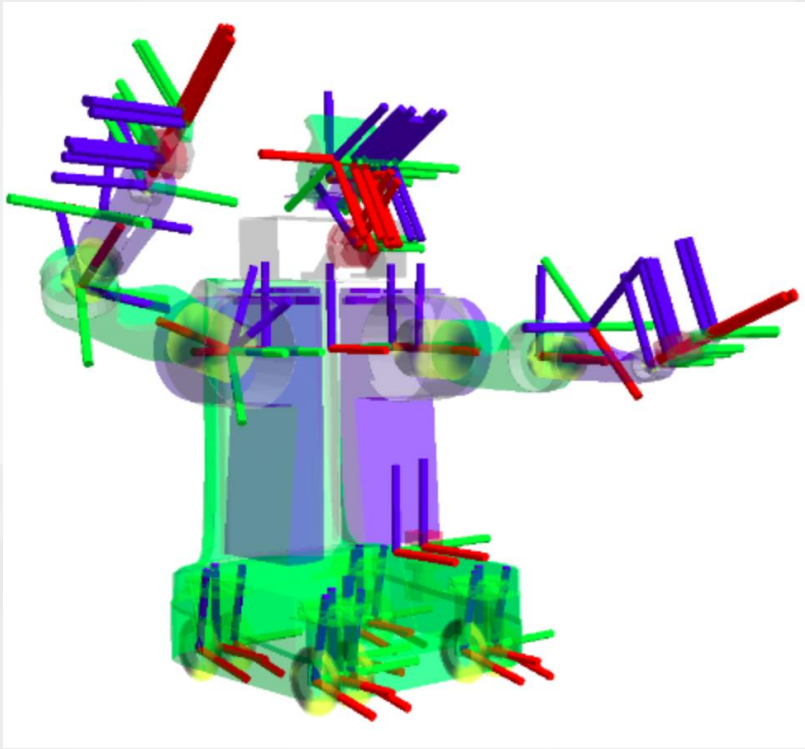
2.ROS坐标变换

- 机器人系统一般都包含很多坐标系，如世界坐标系，机体坐标系，机械手坐标系，头坐标系等。TF是一个让使用者可以跟踪所有这些坐标系，它采用树型数据结构，根据时间缓冲并维护各个参考系的坐标变换关系，可以在任意时间完成2个参考系的坐标变换。常用操作:

  - 监听坐标变换
  - 广播坐标变换

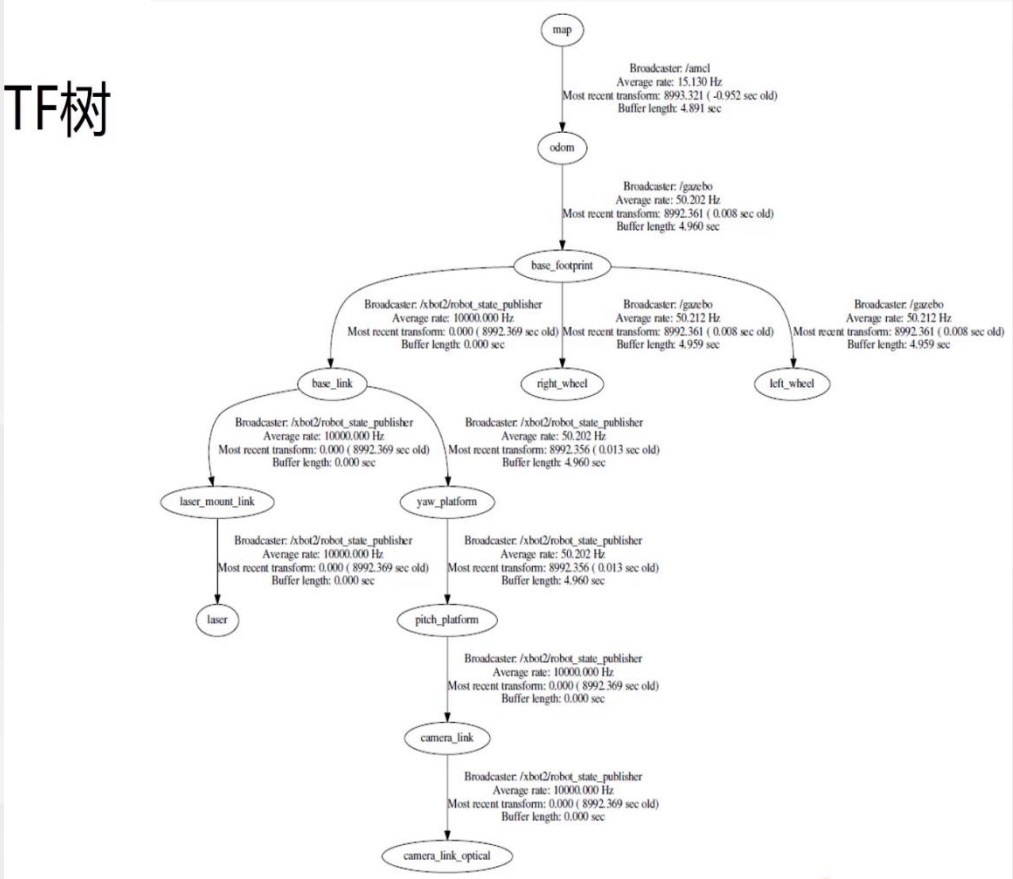
2.ROS坐标变换

- TF的定义不是那么的死板，它可以被当做是一种标准规范，这套标准定义了坐标转换的数据格式和数据结构.
- TF本质是**树状的数据结构**，所以我们通常称之为**"tf tree"**,TF也可以看成是一个topic:/tf，话题中的message保存的就是tf tree的数据结构格式．维护了整个机器人的甚至是地图的坐标转换关系.
- TF还可以看成是一个package,它当中包含了很多的工具．比如可视化，查看关节间的tf,debug tf等等.



2.ROS坐标变换

- TF tree结构中，每一个圆圈代表一个frame,对应着机器人上的一个link，任意的两个frame之间都必须是联通的，如果出现某一环节的断裂，就会引发error系统报错。所以完整的tf tree不能有任何断层的地方，这样我们才能查清楚任意两个frame之间的关系.



2.ROS坐标变换

TF树标准数据格式

观察标准的格式规范，首先header定义了序号，时间以及frame的名称．接着还写了child_frame，这两个frame之间要做那种变换就是由geometry_msgs/Transform来定义．
Vector3三维向量表示平移，Quaternion四元数表示旋转．

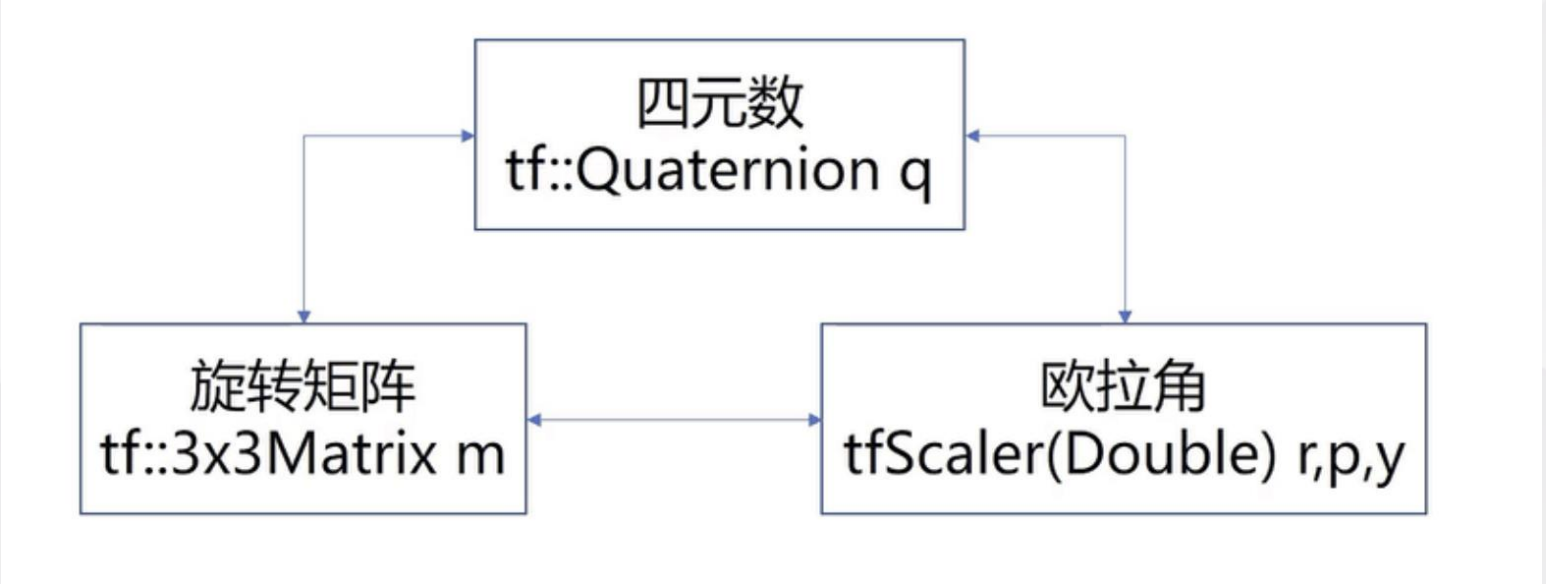
```
std_mags/Header header
        uint32 seq
        time stamp
        string frame_id
string child_frame_id
geometry_msgs/Transform transform
        geometry_msgs/Vector3 translation
                float64 x
                float64 y
                float64 z
        geometry_msgs/Quaternion rotation
                float64 x
                float64 y
                flaot64 z
                float64 w
```

TF中C++数据类型

名称	数据类型
向量	tf::Vector3
点	tf::Point
四元数	tf::Quaternion
3*3矩阵（旋转矩阵）	tf::Matrix3x3
位姿	tf::pose
变换	tf::Transform
带时间戳的以上类型	tf::Stamped
带时间戳的变换	tf::StampedTransform

2.ROS坐标变换

TF中C++数据类型



2.ROS坐标变换——写一个TF广播CPP

- 在src活页夹建立如下cpp文件

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <turtlesim/Pose.h>
std::string turtle_name;
void poseCallback(const turtlesim::PoseConstPtr& msg)
{                       static tf::TransformBroadcaster br;
                        tf::Transform transform;
                        transform.setOrigin( tf::Vector3(msg->x, msg->y, 0.0) );
                        tf::Quaternion q;  q.setRPY(0, 0, msg->theta);
                        transform.setRotation(q);
                        br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world", turtle_name));}
int main(int argc, char** argv)
{                       ros::init(argc, argv, "my_tf_broadcaster");
                        if (argc != 2)
                        {
                                        ROS_ERROR("need turtle name as argument");
                                        return -1;
                        }
                         turtle_name = argv[1];
                         ros::NodeHandle node;
                        ros::Subscriber sub = node.subscribe(turtle_name+"/pose", 10, &poseCallback);
                        ros::spin();
                        return 0;
}
```

2.ROS坐标变换——写一个TF广播CPP

- 程序订阅了turtle_name/pose话题， turtle_name是程序运行的输入参数，有了前面的基础这块就不细说了，主要介绍回调函数的代码。

```
static tf::TransformBroadcaster br;
```

- tf是一个命名空间，详细的文档介绍在如下网址：
- http://docs.ros.org/neotic/api/tf/html/c++/namespacetf.html
- TransformBroadcaster是tf命名空间下的一个类，这个类提供一个简单的方法去发布坐标转换信息，处理说有的发布和填充消息。
- 常用成员函数：

```
void sendTransform (const StampedTransform &transform)
```

2.ROS坐标变换——写一个TF广播CPP

- StampedTransform类：tf的采样转换的数据类型。
- 常用成员函数：

```
StampedTransform (const tf::Transform &input, const ros::Time &timestamp, const std::string &frame_id, const std::string &child_frame_id)
```

- Quaternion类，常规的四元数类，表示刚体的旋转。

2.ROS坐标变换——写一个TF广播CPP

- Transform类，支持刚体的平移和旋转变换。

```
tf::Transform transform;
transform.setOrigin( tf::Vector3(msg->x, msg->y, 0.0) );
tf::Quaternion q;
q.setRPY(0, 0, msg->theta);
transform.setRotation(q);
```

- 建立了一个tf::Transform实例，然后把turtle2D的pose信息转换成3D的信息。

```
br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world", turtle_name));
```

- 第一个参数是transform实例，第二个是时间戳，第三个是父坐标系名，第四个是子坐标系名。

2.ROS坐标变换——写一个TF广播CPP

- 打开该package的CMakeLists.txt文件，添加下面命令代码：

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

- 然后编译工作区 →

```
$ catkin_make
```

- 编译成功后，我们在编写如下的launch文件：

```
<launch>
<!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
  <!-- Axes -->    <param name="scale_linear" value="2" type="double"/>
  <param name="scale_angular" value="2" type="double"/>
  <node pkg="learning_tf" type="turtle_tf_broadcaster" args="/turtle1" name="turtle1_tf_broadcaster" />
  <node pkg="learning_tf" type="turtle_tf_broadcaster" args="/turtle2" name="turtle2_tf_broadcaster" />
</launch>
```

- 然后运行该launch文件→

```
$ roslaunch learning_tf start_demo.launch
```

- 检验下结果→

```
$ rosrun tf tf_echo /world /turtle1
```

2.ROS坐标变换——写一个TF订阅者CPP

- 切换到learning_tf目录下

```
$ roscd learning_tf
```

- 在src活页夹下建立src/turtle_tf_listener.cpp文件

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <geometry_msgs/Twist.h>
#include <turtlesim/Spawn.h>
int main(int argc, char** argv){
        ros::init(argc, argv, "my_tf_listener");
        ros::NodeHandle node;
        ros::service::waitforService("spawn");
        ros::ServiceClient add_turtle = node.serviceClient<turtlesim::Spawn>("spawn");
        turtlesim::Spawn srv;  add_turtle.call(srv);
        ros::Publisher turtle_vel = node.advertise<geometry_msgs::Twist>("turtle2/cmd_vel", 10);
        tf::TransformListener listener;
        ros::Rate rate(10.0);
```

2.ROS坐标变换——写一个TF订阅者CPP

```
        while (node.ok()){
          tf::StampedTransform transform;
          try {
            listener.lookupTransform("/turtle2", "/turtle1",
            ros::Time(0), transform);
          }
          catch (tf::TransformException &ex){
            ROS_ERROR("%s",ex.what());
            ros::Duration(1.0).sleep();
            continue;
          }
          geometry_msgs::Twist vel_msg;
          vel_msg.angular.z = 4.0 * atan2(transform.getOrigin().y(), transform.getOrigin().x());
          vel_msg.linear.x = 0.5 * sqrt(pow(transform.getOrigin().x(), 2) +  pow(transform.getOrigin().y(), 2));
          turtle_vel.publish(vel_msg);
          rate.sleep();
        }
    return 0;
}
```

2.ROS坐标变换——写一个TF订阅者CPP

- tf::Transformer类，提供一个系统内的任意两个坐标系的转换。
- tf::TransformListener类，继承tf::Transformer类，自动订阅ROS的transform消息。
- 常用成员函数：

```
void Transformer::lookupTransform(const std::string & target_frame,const std::string & source_frame,
const ros::Time & time,StampedTransform & transform ) const
```

- 通过 frame ID 得到两个坐标系的转换关系。
- target_frame 目标坐标系：需要将数据变换到的坐标系
- source_frame 起始坐标系：数据的起始坐标系
- time 时间：变换所需的时间（0表示时间待定）
- transform 变换：添加变换参考

2.ROS坐标变换——写一个TF订阅者CPP

```
tf::TransformListener listener;
try{ listener.lookupTransform("/turtle2", "/turtle1", ros::Time(0), transform); }
```

- 建立一个TransformListener 实例，一旦实例创建，它就自动接收tf的转换消息，并缓存10秒。然后查询一个指定的转换，就是查询从/turtle2到/turtle1的转换，ros::Time(0) 表示我们想要的是最新的一次变换，然后把变换存到transform变量。

```
geometry_msgs::Twist vel_msg;
 vel_msg.angular.z = 4.0 *atan2(transform.getOrigin().y(),transform.getOrigin().x());
 vel_msg.linear.x = 0.5 * sqrt(pow(transform.getOrigin().x(), 2) +pow(transform.getOrigin().y(), 2));
   turtle_vel.publish(vel_msg);
```

- 根据上面检查的变换矩阵，把位置变成线速度，把角度变成角度度，然后发布turtle2/cmd_vel话题，驱动turtle2走向turtle1。

2.ROS坐标变换——写一个TF订阅者CPP

- 然后在CMakeLists.txt中加入如下代码：

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

- 然后进行编译

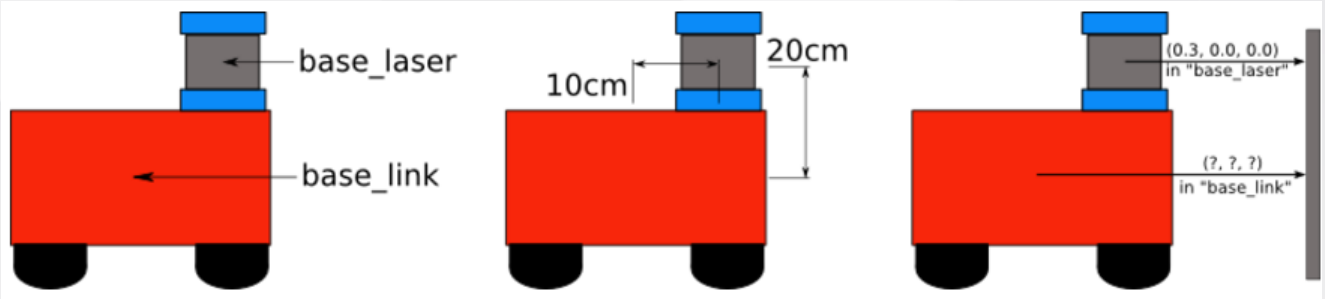
```
$ catkin_make
```

- 然后编写launch文件，代码如下：

```
<launch>
<!-- Turtlesim Node-->
<node pkg="turtlesim" type="turtlesim_node" name="sim"/>
<node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
<!-- Axes -->
<param name="scale_linear" value="2" type="double"/>
<param name="scale_angular" value="2" type="double"/>
<node pkg="learning_tf" type="turtle_tf_broadcaster" args="/turtle1" name="turtle1_tf_broadcaster" />
<node pkg="learning_tf" type="turtle_tf_broadcaster" args="/turtle2" name="turtle2_tf_broadcaster" />
<node pkg="learning_tf" type="turtle_tf_listener" name="listener" />
</launch>
```

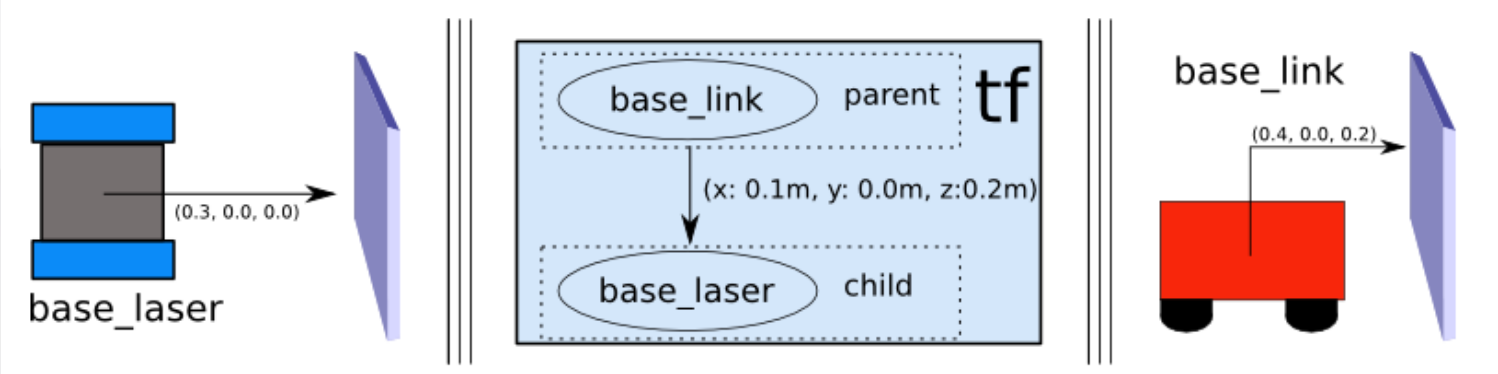
2.ROS坐标变换——用TF配置机器人

- 很多ros包都要求用tf软件库发布的机器人的变换树。抽象层面，一个变换树定义了在不同坐标系间的平移和旋转。举例来说，一个机器人有一个移动底盘和一个安装在它上面的激光雷达。然后我们可以定义两个坐标系，一个在机器人底部的中心（base_link)，一个在激光雷达的中心(base_laser）。对于导航包来说，base_link就是机器人的旋转中心。



2.ROS坐标变换——用TF配置机器人

- 现在我们有一些来自laser的距离数据，是在base_laser坐标系下的，现在我们要把这个数据转换到base_link下，帮助机器人避障。
- 假设我们知道激光在机器人底部中心的前方10cm上方20cm，没有旋转，如果我们在base_laser下测得的距离是(0.3,0.0,0.0),那么在base_link下距离为(0.4,0.0,0.2).



2.ROS坐标变换——用TF配置机器人

- 在catkin_ws/src目录输入如下命令

```
$ catkin_create_pkg robot_setup_tf roscpp tf geometry_msgs
```

- 接下来我们将在ros下广播一个base_laser->base_link的变换，在robot_setup_tf包下加上以下代码。

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
int main(int argc, char** argv){
   ros::init(argc, argv, "robot_tf_publisher");
   ros::NodeHandle n;
   ros::Rate r(100);
   tf::TransformBroadcaster broadcaster;
   while(n.ok()){
                broadcaster.sendTransform( tf::StampedTransform( tf::Transform(tf::Quaternion(0, 0, 0, 1), tf::Vector3(0.1,
0.0, 0.2)), ros::Time::now(),"base_link", "base_laser"));
 r.sleep(); } }
```

2.ROS坐标变换——用TF配置机器人

```
#include <ros/ros.h>
#include <geometry_msgs/PointStamped.h>
#include <tf/transform_listener.h>
 void transformPoint(const tf::TransformListener& listener)
{ //we'll create a point in the base_laser frame that we'd like to transform to the base_link frame
                geometry_msgs::PointStamped laser_point;
                laser_point.header.frame_id = "base_laser";
//we'll just use the most recent transform available for our simple example
laser_point.header.stamp = ros::Time();
//just an arbitrary point in space
laser_point.point.x = 1.0;
laser_point.point.y = 0.2;
laser_point.point.z = 0.0;
```

2.ROS坐标变换——用TF配置机器人

```
try
{
                geometry_msgs::PointStamped base_point;
                listener.transformPoint("base_link", laser_point, base_point);
                 ROS_INFO("base_laser: (%.2f, %.2f. %.2f) -----> base_link: (%.2f, %.2f, %.2f) at time %.2f",
laser_point.point.x, laser_point.point.y, laser_point.point.z, base_point.point.x, base_point.point.y, base_point.point.z,
base_point.header.stamp.toSec());
 }
 catch(tf::TransformException& ex){
               ROS_ERROR("Received an exception trying to transform a point from \"base_laser\" to \"base_link\": %s",
ex.what());
                }
 }
```

2.ROS坐标变换——用TF配置机器人

```
//we'll create a point in the base_laser frame that we'd like to transform to the base_link frame
geometry_msgs::PointStamped laser_point;
laser_point.header.frame_id = "base_laser";
 //we'll just use the most recent transform available for our simple example
laser_point.header.stamp = ros::Time();
//just an arbitrary point in space
laser_point.point.x = 1.0;
laser_point.point.y = 0.2;
laser_point.point.z = 0.0;
```

- 我们手动生成了一个点的激光数据，(1.0,0.2,0.0)

2.ROS坐标变换——用TF配置机器人

- 在 CMakeLists.txt文件中输入

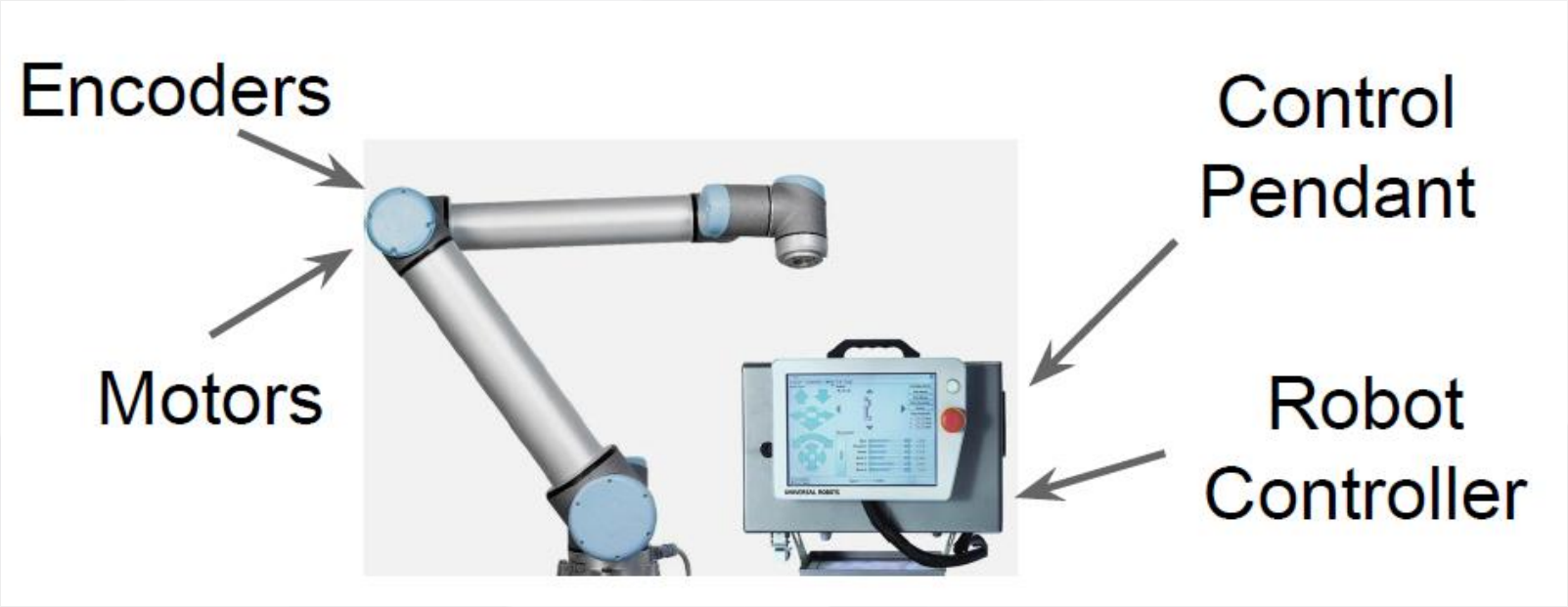
```
add_executable(tf_broadcaster src/tf_broadcaster.cpp)
add_executable(tf_listener src/tf_listener.cpp)
target_link_libraries(tf_broadcaster ${catkin_LIBRARIES})
target_link_libraries(tf_listener ${catkin_LIBRARIES})
```

- 编译

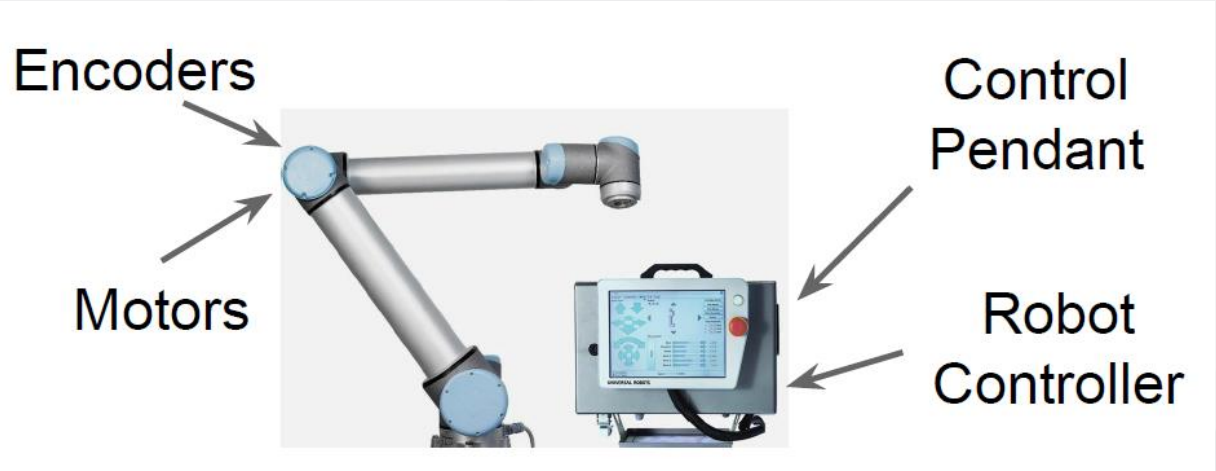
```
$ catkin_make
```

03 机器人系统构建

3.机器人系统构建——机械臂

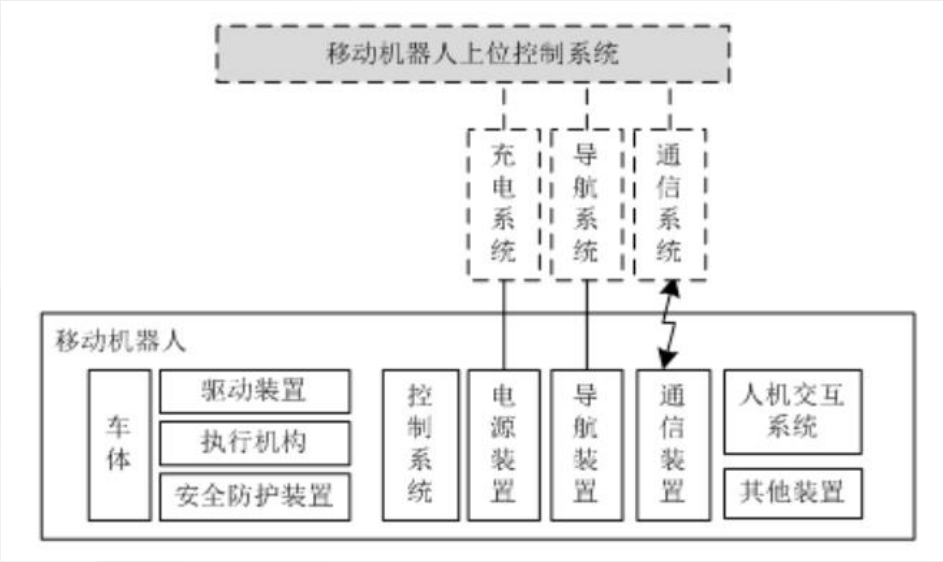


3.机器人系统构建——机械臂



Motor controllers	Inverse Kinematics	GUI	Collision Detection
Trajectory Planner	PID Controller	Error Monitoring	Teaching Mode

3.机器人系统构建——移动机器人



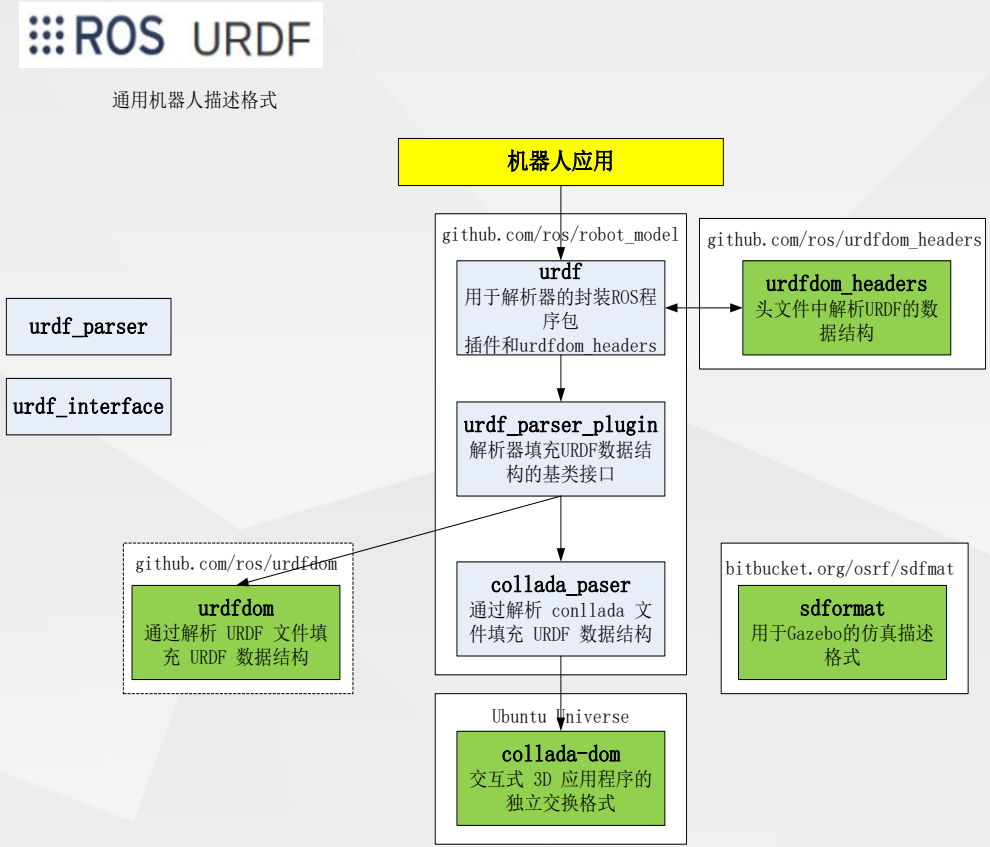
04 机器人建模与仿真

4.机器人建模与仿真

- 机器人建模系统可建立机器人所处环境模型和机器人自身模型。
- 环境模型包括对机器人所处现实物理环境和各种物体的模拟，通过建立地面、障碍物、目标物等模型支持机器人与虚拟环境的交互。
- 机器人自身模型包括机器人几何外型模型、连杆结构模型、物理结构模型、驱动器模型等。
- 通过此类建模可以实现在虚拟现实环境中构建模型并进行相应测试和修改，验证其真实机械结构设计的精确度，从而减少人工计算量和实物实验成本。

4.机器人建模与仿真——URDF

URDF（Unified Robot Description Format，统一机器人描述格式）由一些不同的功能包和组件构成，是一种XML格式，专门用来对机器人硬件进行抽象的模型描述。



4.机器人建模与仿真——URDF

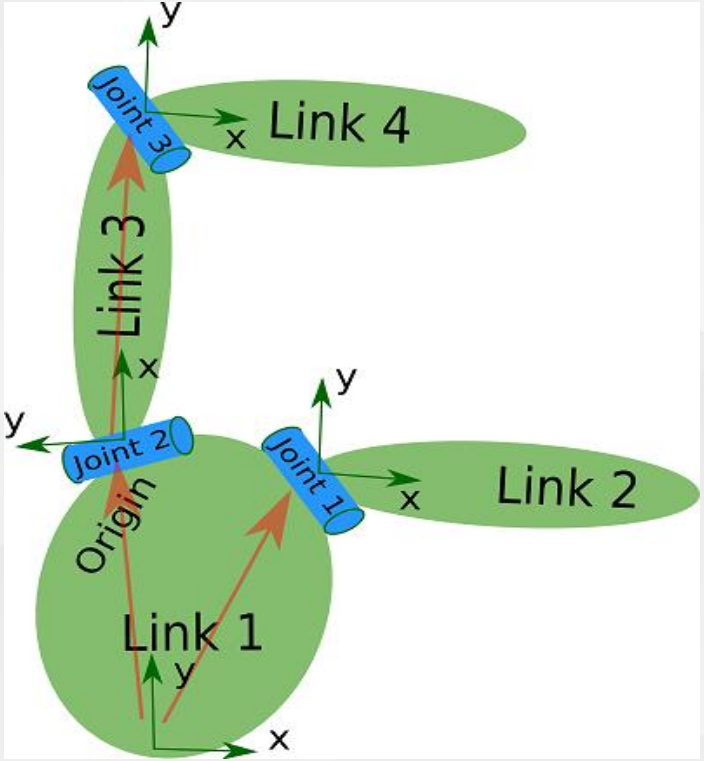
URDF创建的机器人模型包含的内容有：

- 连杆 link
- 关节 joint
- 运动学参数 axis
- 动力学参数 dynamics
- 可视化模型 visual
- 碰撞检测模型 collision

关节类型

- Revolute：转动关节，但是有转动范围，必须要求之后的limit参数中必须含有上下界才合法。
- Continuous：连续转动关节，没有转动范围。
- Prismatic：平动关节，也需要上下界的定义
- Fixed：完全固定，连接的两个link并没有可动关系，可以理解为纯粹为了标识固定传感器相对位置与姿态而设立的。
- Floating：允许link与link之间有完全6个自由度的转换，跟fixed一样，一般是标识一些环境物体或者无人机在全局位置姿态的。
- Planar：与floating类似，用于标识地面机器人在地面上的位置姿态。

4.机器人建模与仿真——URDF



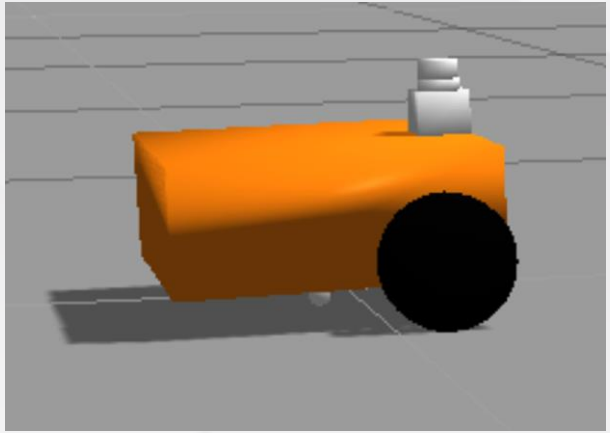

```
<robot name="pr2">
     <link> ... </link>
     <link> ... </link>
     <link> ... </link>

     <joint>  ....  </joint>
     <joint>  ....  </joint>
     <joint>  ....  </joint>
   </robot>
```

4.机器人建模与仿真——XACRO

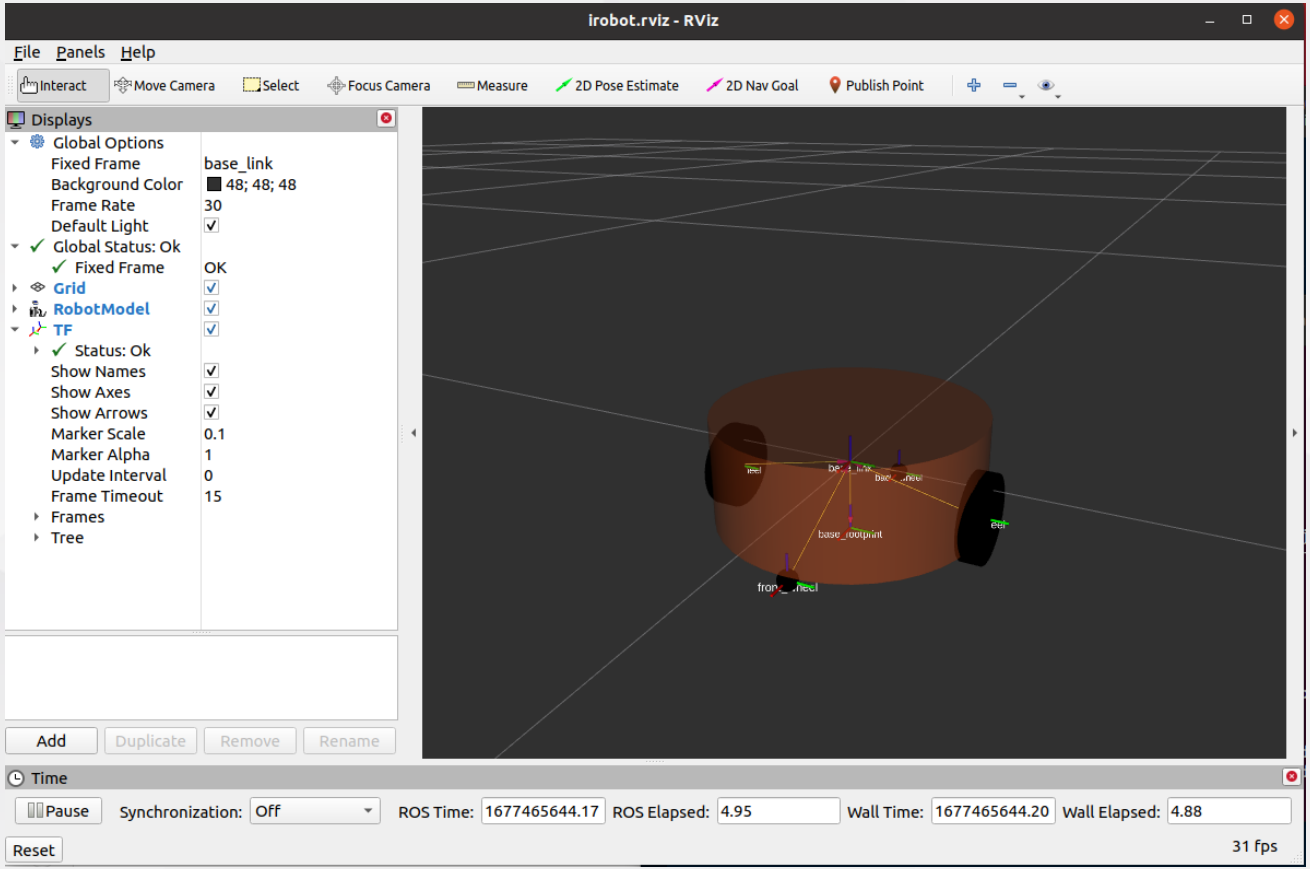
- XML macros是ROS中简化大型xml文件的写法、增加结构性与扩展性的XML宏语言。最常用在机器人描述檔urdf的构建中。
- 复杂的机器人常常驱动多达十几个joint，而标注传感器位置常常也需要使用单独的joint。使得整个urdf显得非常复杂，使用宏能极大增加urdf文件的层次感与可读性，在各个机器人公开的ROS urdf package中，基本都是用 XACRO工具进行组织。

4.机器人建模与仿真——XACRO

1. 实体与实体块（property and property blocks）
2. 数学表达式（math expression）
3. 条件判断块（conditional blocks）
4. 宏（macros）
5. 引用其他xacro文件



4.机器人建模与仿真——模型加载



THANKS