机器人操作系统

机电工程学院

2025 · 秋

课程安排

01 第一章 ROS概述与环境搭建

02 第二章 ROS通信机制

03 第三章 ROS架构与运行管理

04 第四章 ROS常用组件

05 第五章 机器人建模与仿真

06 第六章 ROS进阶功能

第 四 章

ROS常用组件

ROSbag

ROSbag通过数据采集、回放、分析和共享，显著提升了机器人开发的效率和可靠性。在实际应用中，从调试到验证、从训练到协作，ROSbag都发挥了关键作用，是开发机器人系统不可或缺的工具。

ROSbag录制和回放数据

- 数据采集与调试
- 算法验证
- 环境重现
- 开发与协作
- 训练与测试
- 应对安全和风险场景

数据采集与调试

- 在机器人开发中，环境和机器人行为的实时性和复杂性增加了问题定位和解决的难度。ROSbag可以帮助录制实时数据，用于后续分析和调试。
- 示例：在开发一个导航机器人时，机器人可能在某些地点反复出现定位偏差问题。通过ROSbag录制传感器数据（如激光雷达、IMU、里程计）和导航算法的输出，可以回放数据多次调试算法，而无需重新执行机器人运行。

算法验证

- 在开发新算法时，可能需要验证其性能。使用ROSbag录制的真实场景数据，可以在离线环境中反复测试和优化算法，确保其稳定性和鲁棒性。
- 示例：在开发一个SLAM（同步定位与建图）算法时，研究者可以录制传感器数据（激光雷达、相机等）并在实验室内回放这些数据，验证SLAM算法的性能，而不需要一直跑实地实验。

环境重现

- 某些场景在实际操作中可能很难重现，但通过ROSbag录制可以保存关键数据，用于后续分析和再现。
- 示例：在一个仓储机器人系统中，如果某一次抓取任务失败，通过回放录制的机械臂关节数据和视觉数据，可以重现失败场景，分析失败原因。

开发与协作

- ROSbag录制的数据可以共享，便于团队中的不同开发者基于相同的真实数据开展工作。
- 示例：一个无人车团队中，感知团队录制了摄像头和雷达的原始数据，分享给定位和决策团队。各团队成员可以并行开展工作，而不需要实际控制车辆运行。

训练与测试

- 机器学习模型的训练和测试需要大量的真实数据，ROSbag可以作为重要的数据来源。
- 示例：在开发一个基于深度学习的目标检测模型时，通过ROSbag录制摄像头的图像数据，生成训练集和测试集，用于训练模型或验证模型效果。

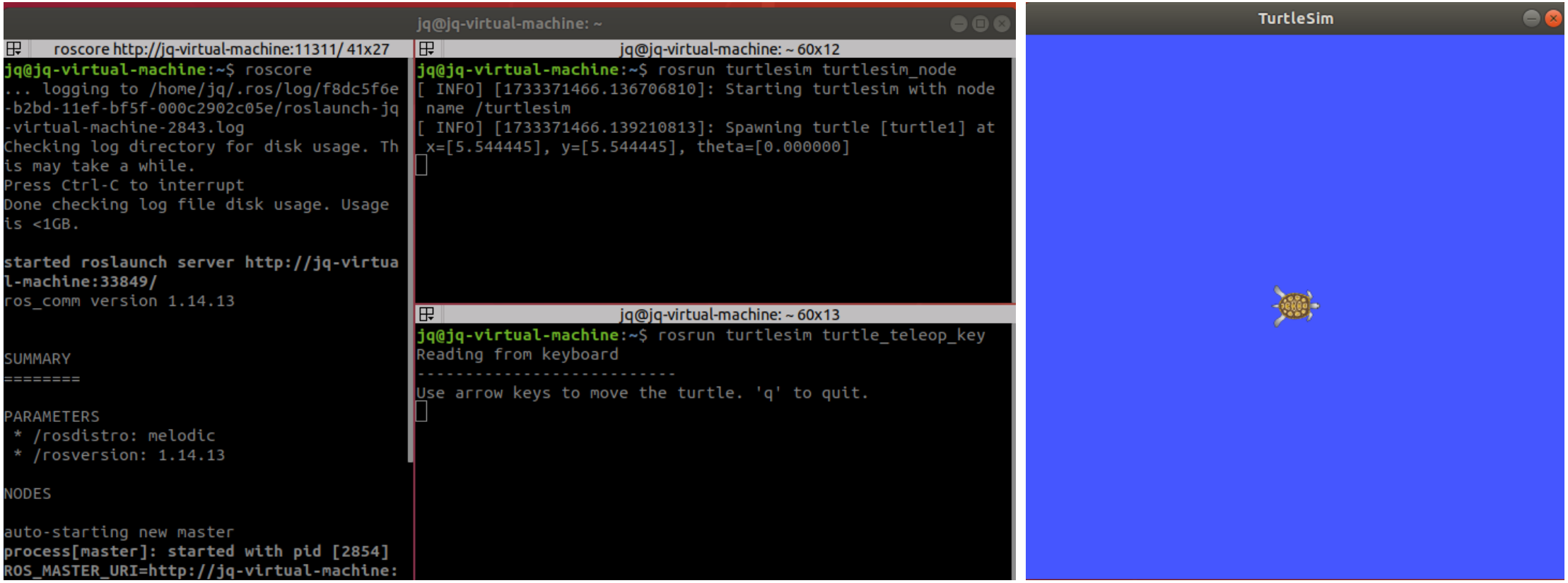
应对安全和风险场景

- 一些场景可能具有高风险或高成本，无法多次尝试。ROSbag录制的现场数据，可以用于虚拟环境中的模拟。
- 示例：在开发一个无人机的障碍物规避算法时，录制飞行数据和环境感知数据后，可在仿真环境中多次测试算法，避免实际飞行中发生意外。

ROSbag Demo

cn/ROS/Tutorials/Recording and playing back data - ROS Wiki

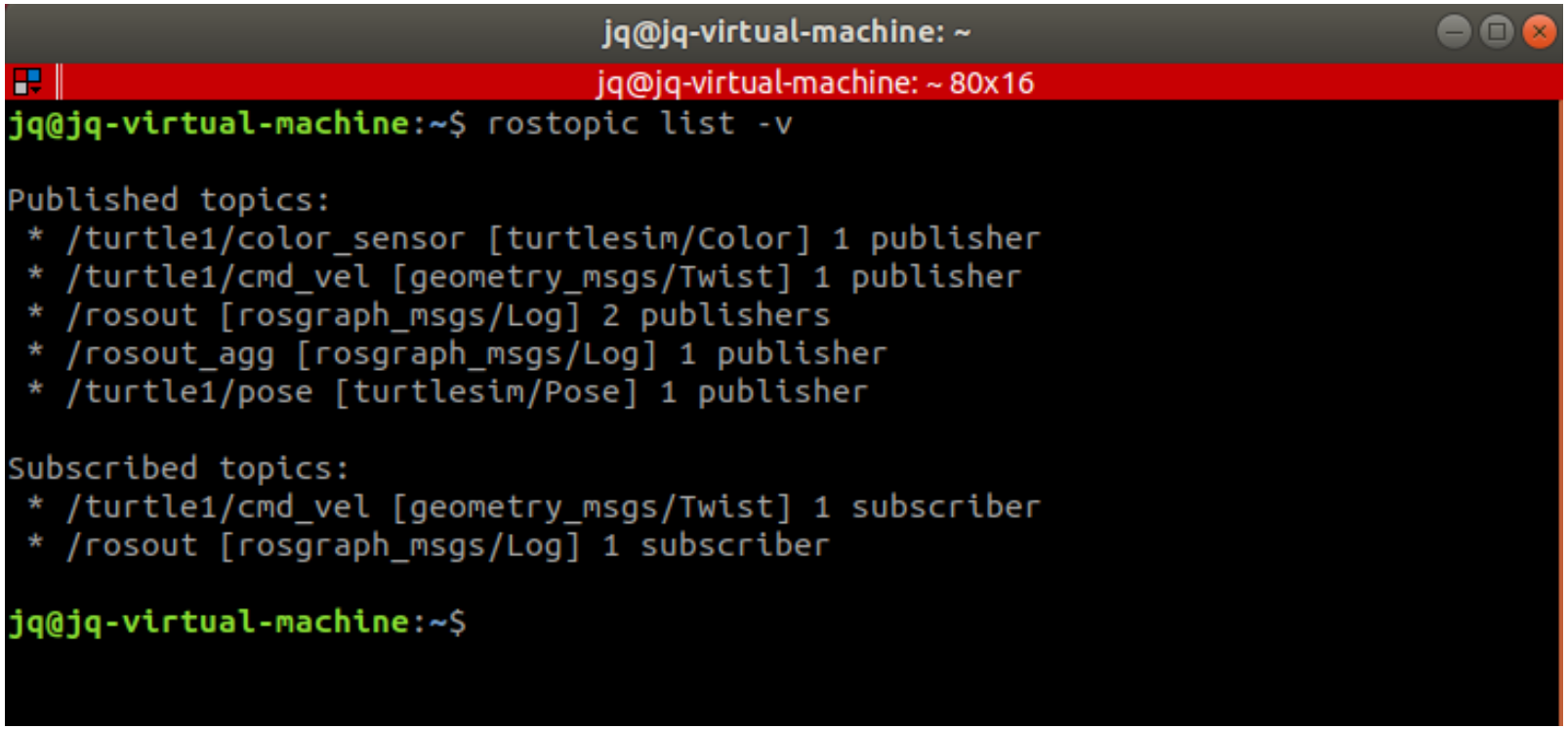
1.录制数据（创建bag文件）



录制所有发布的话题

- `rostopic`：ROS 的命令行工具之一，用于与 ROS 主题进行交互。
- `list`：列出所有正在运行的 ROS 主题。
- `-v` **(verbose)**：以详细模式显示主题的附加信息，例如发布者和订阅者。

`rostopic list -v` 是用于显示 ROS 系统中当前可用的所有主题及其详细信息的命令。

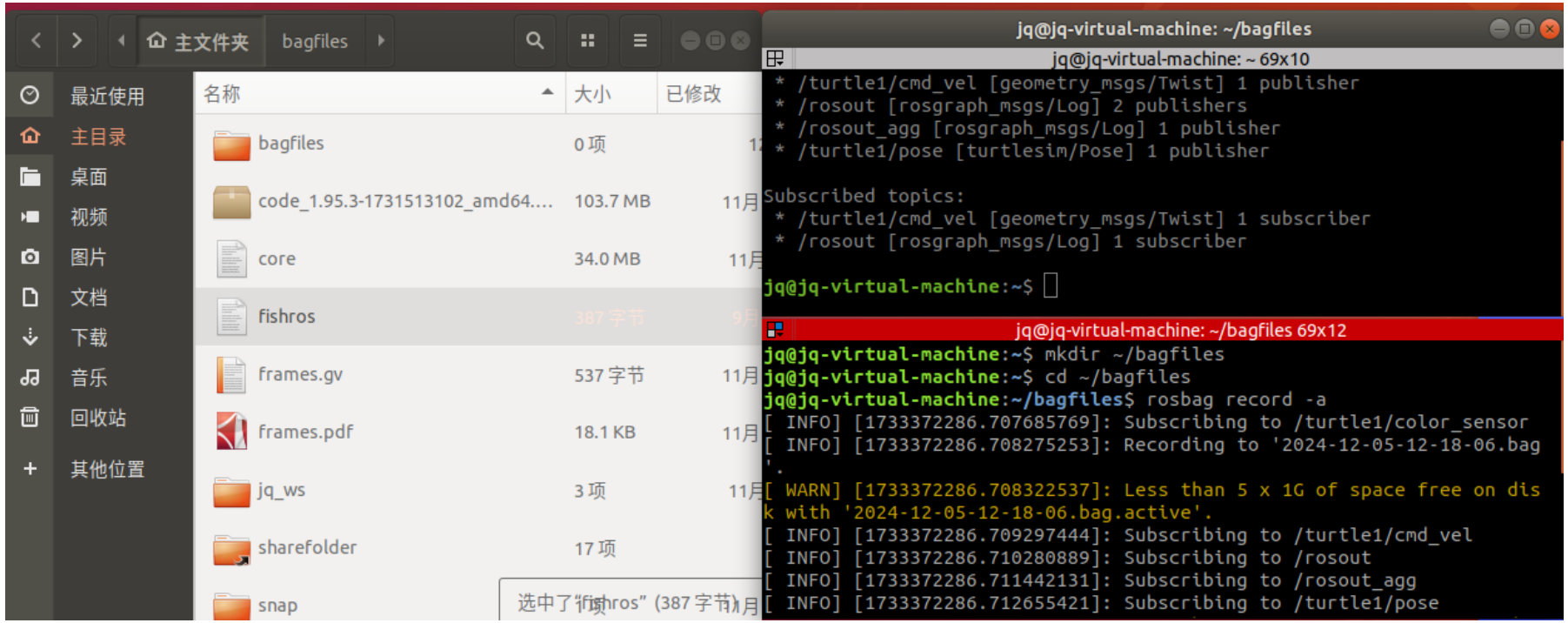


✓ 已发布主题的列表是唯一可能被记录在数据日志文件中的消息类型，因为只有发布的消息才能被录制。
✓ 由teleop_turtle发布的/turtle1/cmd_vel话题是指令消息，作为turtlesim进程的输入。
✓ 消息/turtle1/color_sensor和/turtle1/pose是turtlesim发布的输出消息。

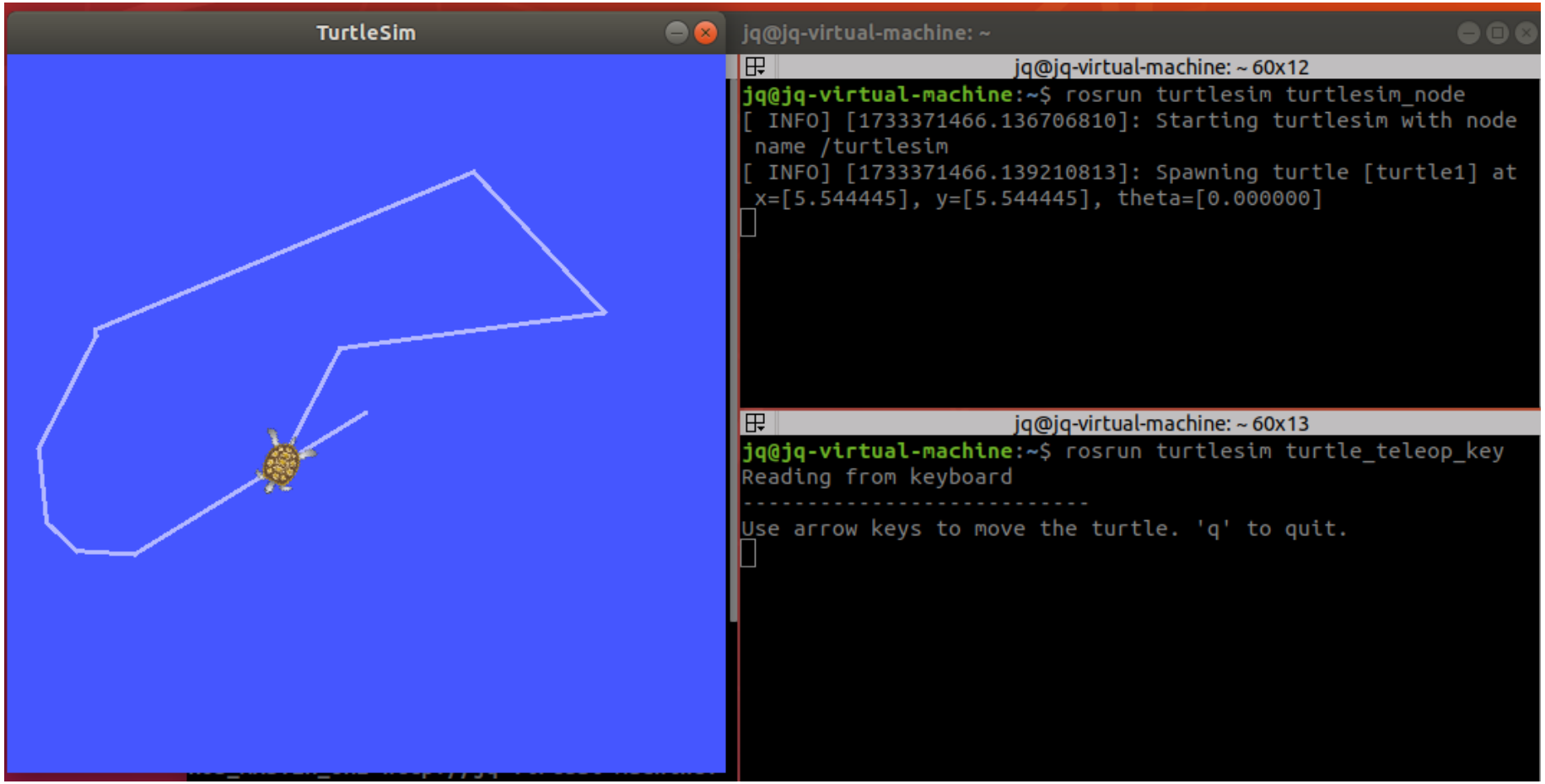
设置一个目录以存储ROS bag文件并开始录制当前系统中的所有话题数据

```
mkdir ~/bagfiles
cd ~/bagfiles
rosbag record -a
```

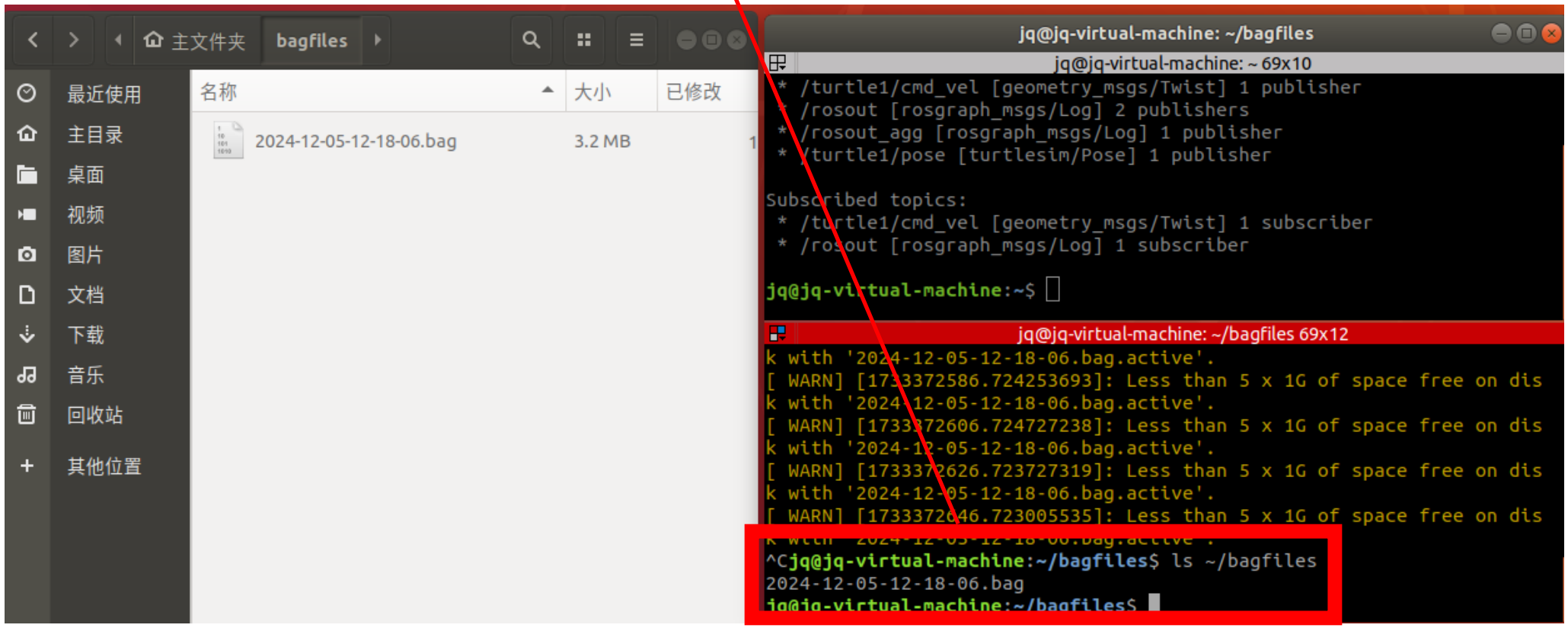
- 在当前用户的主目录下创建一个名为 `bagfiles` 的目录，用于存放ROS bag文件。
- 切换到刚刚创建的 `bagfiles` 目录，以便录制的ROS bag文件可以直接保存到该目录中。
- 启动 `rosbag` 命令，使用 `-a` 参数录制系统中**所有的主题数据**。
- 录制的内容包括系统中正在发布的所有消息，适用于需要捕获完整系统运行状态的情况。



- 在另一个终端窗口中运行 `turtle_teleop_key` 节点
- 使用键盘控制Turtlesim中的乌龟随意移动10秒钟左右，这会发布相关的主题消息



- 返回到运行 `rosbag record` 的终端窗口，按 `Ctrl+C` 停止录制。
- 停止后，rosbag会将录制的所有数据写入 `.bag` 文件。
- 进入 `~/bagfiles` 目录，可以通过以下命令查看：
- 输出中会显示一个以时间戳命名的 `.bag` 文件



2.检查并回放bag文件

这些信息告诉你bag文件中所包含话题的名称、类型和消息数量。

```
rosbag info <your bagfile>
```

- `<your bagfile>` 是你的 `.bag` 文件的完整路径或文件名。

```
jq@jq-virtual-machine:~/bagfiles$ rosbag info 2024-12-05-12-18-06.bag
path:        2024-12-05-12-18-06.bag
version:     2.0
duration:    6:04s (364s)
start:       Dec 05 2024 12:18:06.71 (1733372286.71)
end:         Dec 05 2024 12:24:10.87 (1733372650.87)
size:        3.1 MB
messages:    45547
compression: none [4/4 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             rosgraph_msgs/Log   [acffd30cd6b6de30f120938c17c593fb]
             turtlesim/Color     [353891e354491c51aabe32df673fb446]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /rosout                 22 msgs    : rosgraph_msgs/Log
 (2 connections)
             /rosout_agg             18 msgs    : rosgraph_msgs/Log
             /turtle1/cmd_vel       283 msgs    : geometry_msgs/Twist
             /turtle1/color_sensor 22612 msgs    : turtlesim/Color
             /turtle1/pose         22612 msgs    : turtlesim/Pose
jq@jq-virtual-machine:~/bagfiles$
```

- 表示 `.bag` 文件录制的时间长度。
- 显示 `.bag` 文件录制的开始和结束时间，包括日期和具体时间戳。
- 录制的主题消息的总数量。
- 文件是否使用了压缩（如 `none` 表示无压缩）。

表示控制乌龟的速度命令。

记录颜色传感器的数据。

记录乌龟的位置和方向。

`rosbag play <your bagfile>` 是用于回放 `.bag` 文件的命令，可以将录制的 ROS 话题数据重新发布到系统中，模拟运行时的真实环境。

回放录制的数据：

- `.bag` 文件中包含的所有话题数据会按照录制时的时间戳重新发布到 ROS 系统中。
- 订阅这些话题的节点会收到回放的数据，并表现出录制时的行为。
- 使用 `-r` 参数指定回放的速度倍数（默认为 1.0）。
- 使用 `--duration` 参数只回放指定时间段内的数据。

注意要按 Ctrl+C 终止相应终端，并重启 turtlesim node

3.录制数据子集

`-O subset`：

- 指定输出的 `.bag` 文件名为 `subset.bag`。

命令 `rosbag record -O subset /turtle1/cmd_vel /turtle1/pose` 的作用是录制指定的主题（`/turtle1/cmd_vel` 和 `/turtle1/pose`）并将其保存到一个自定义命名的 `.bag` 文件中。

- 按下 `Ctrl+C` 停止录制，数据会保存到当前目录下的 `subset.bag` 文件中。

  运行命令 `rosbag info subset.bag` 后，你会获得关于 `subset.bag` 文件的详细信息。

```
jq@jq-virtual-machine:~/bagfiles$ rosbag info subset.bag
path:        subset.bag
version:     2.0
duration:    17.3s
start:       Dec 05 2024 15:13:36.34 (1733382816.34)
end:         Dec 05 2024 15:13:53.67 (1733382833.67)
size:        89.7 KB
messages:    1090
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /turtle1/cmd_vel      6 msgs    : geometry_msgs/Twist
             /turtle1/pose      1084 msgs    : turtlesim/Pose
jq@jq-virtual-machine:~/bagfiles$
```

机器人建模和仿真

cn/urdf/Tutorials - ROS Wiki

在ROS系统中，机器人的三维模型是通过URDF文件进行描述的。URDF全称为Unified Robot Description Format，是一种基于XML规范扩展出来的文本格式。

从机构学角度讲，机器人通常被分解为由连杆和关节组成的结构。

连杆是带有质量属性的刚体，而关节是连接和限制两个刚体相对运动的结构，也被称为“运动副”。通过关节将连杆依次连接起来，就构成了一个个运动链，也就是机器人的机构模型。

URDF就是就是用来描述这一系列关节与连杆的相对关系的工具。除此之外，还包括惯性属性、几何特点和碰撞模型等一系列附加参数。

URDF 文件是一个标准的 XML 文件，在 ROS 中预定义了一系列的标签用于描述机器人模型，机器人模型可能较为复杂，但是 ROS 的 URDF 中机器人的组成却是较为简单，可以主要简化为两部分:连杆(link标签)与关节(joint标签)，URDF标签类型分为:

- robot 根标签，类似于 launch文件中的launch标签
- link 连杆标签
- joint 关节标签
- gazebo 集成gazebo需要使用的标签

在URDF文件中，通常存在一个<robot>根节点，在这个根节点之下是一连串的<joint>和<link>子节点。

<joint>对应的就是关节，<link>对应的就是连杆。这些<joint>和<link>组合在一起，就形成了机器人的完整模型。

<joint>仅起到连接作用，内部参数相对固定。而<link>通常会对应机器人的某个零部件，所以参数内容比较丰富，比如惯性属性、几何特点和碰撞模型这些参数一般都放置在<link>中进行描述。

Robot根标签：

urdf 中为了保证 xml 语法的完整性，使用了robot标签作为根标签，所有的 link 和 joint 以及其他标签都必须包含在 robot 标签内,在该标签内可以通过 name 属性设置机器人模型的名称。

1.属性

name: 指定机器人模型的名称

2.子标签

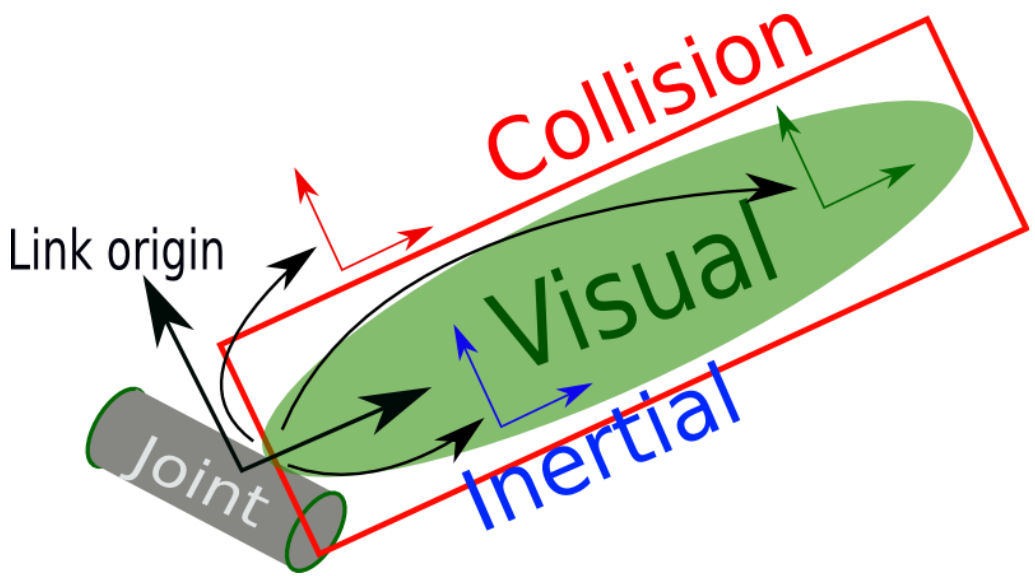
其他标签都是子级标签

3.代码示例

```
<robot name="mycar">
    <!--……-->
</robot>
```

Link标签:

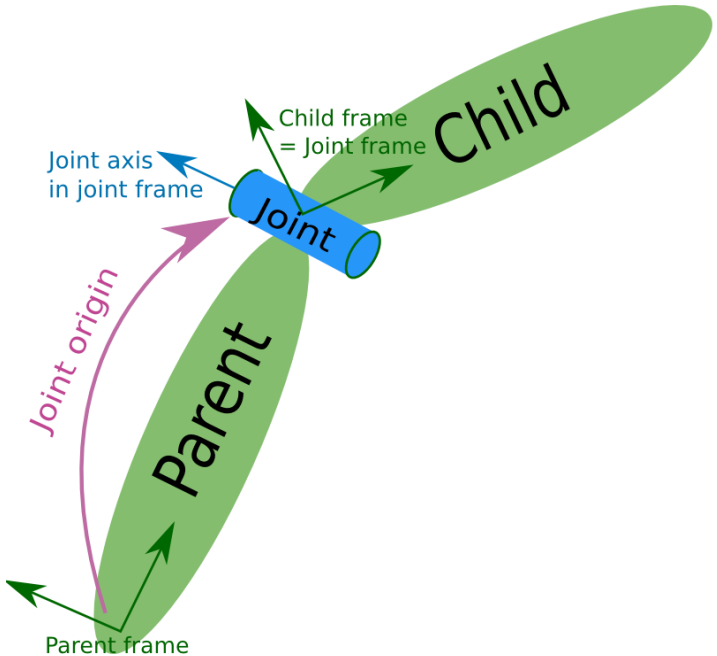
urdf 中的 link 标签用于描述机器人某个部件(也即刚体部分)的外观和物理属性，比如: 机器人底座、轮子、激光雷达、摄像头...每一个部件都对应一个 link, 在 link 标签内，可以设计该部件的形状、尺寸、颜色、惯性矩阵、碰撞参数等一系列属性。



- visual ---> 描述外观(对应的数据是可视的)
  - geometry 设置连杆的形状
    - 标签1: box(盒状)
      - 属性:size=长(x) 宽(y) 高(z)
    - 标签2: cylinder(圆柱)
      - 属性:radius=半径 length=高度
    - 标签3: sphere(球体)
      - 属性:radius=半径
    - 标签4: mesh(为连杆添加皮肤)
      - 属性: filename=资源路径(格式:**package://<packagename>/<path>/文件**)
  - origin 设置偏移量与倾斜弧度
    - 属性1: xyz=x偏移 y便宜 z偏移
    - 属性2: rpy=x翻滚 y俯仰 z偏航 (单位是弧度)
  - metrial 设置材料属性(颜色)
    - 属性: name
    - 标签: color
      - 属性: rgba=红绿蓝权重值与透明度 (每个权重值以及透明度取值[0,1])
- collision ---> 连杆的碰撞属性
- Inertial ---> 连杆的惯性矩阵

Link标签:

urdf 中的 joint 标签用于描述机器人关节的运动学和动力学属性，还可以指定关节运动的安全极限，机器人的两个部件(分别称之为 parent link 与 child link)以"关节"的形式相连接，不同的关节有不同的运动形式: 旋转、滑动、固定、旋转速度、旋转角度限制....,比如:安装在底座上的轮子可以360度旋转，而摄像头则可能是完全固定在底座上。



1.属性

- name ---> 为关节命名
- type ---> 关节运动形式
  - continuous: 旋转关节，可以绕单轴无限旋转
  - revolute: 旋转关节，类似于 continues,但是有旋转角度限制
  - prismatic: 滑动关节，沿某一轴线移动的关节，有位置极限
  - planer: 平面关节，允许在平面正交方向上平移或旋转
  - floating: 浮动关节，允许进行平移、旋转运动
  - fixed: 固定关节，不允许运动的特殊关节

2.子标签

- parent(必需的)

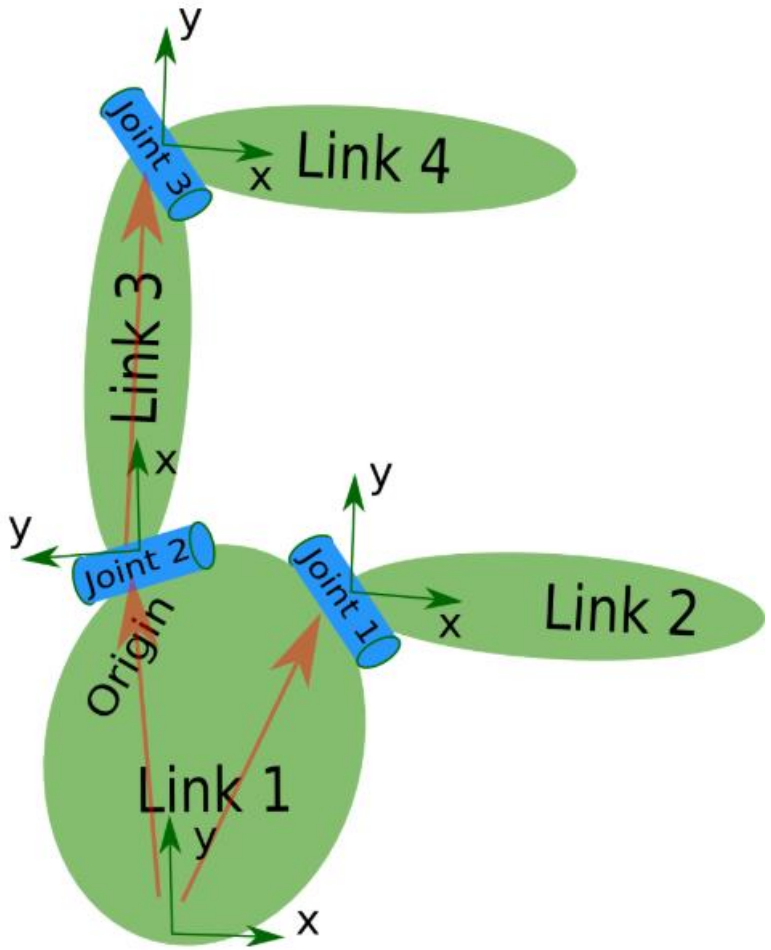
  parent link的名字是一个强制的属性:
  - link:父级连杆的名字，是这个link在机器人结构树中的名字。
- child(必需的)

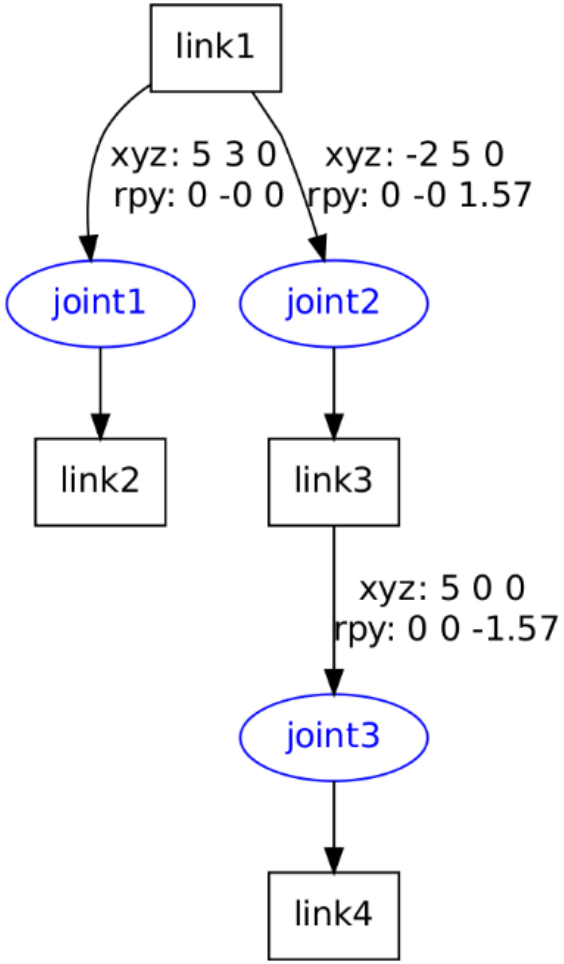
  child link的名字是一个强制的属性:
  - link:子级连杆的名字，是这个link在机器人结构树中的名字。
- origin
  - 属性: xyz=各轴线上的偏移量 rpy=各轴线上的偏移弧度。
- axis
  - 属性: xyz用于设置围绕哪个关节轴运动。

joint标签对应的数据在模型中是不可见的

Create your own urdf file

cn/urdf/Tutorials - ROS Wiki





Parse a urdf file

```
#include <urdf/model.h>
#include "ros/ros.h"
```

`#include <urdf/model.h>`：包含 `urdf` 模型头文件，该文件提供了对 URDF（统一机器人描述格式）模型的支持，使得程序能够加载和解析 URDF 文件。

`#include "ros/ros.h"`：包含 ROS 的基本头文件，提供了与 ROS 系统交互的功能，例如节点初始化、日志记录、订阅/发布等功能。

```
int main(int argc, char** argv){
  ros::init(argc, argv, "my_parser");
```

`int main(int argc, char** argv){`：主函数入口，`argc` 是命令行参数的数量，`argv` 是一个字符指针数组，用于存储命令行参数。该函数是程序的起点。

`ros::init(argc, argv, "my_parser");`：初始化 ROS 节点，节点名为 `"my_parser"`。`argc` 和 `argv` 是从命令行传入的参数，用于 ROS 内部进行初始化。

Parse a urdf file

```
if (argc != 2){
  ROS_ERROR("Need a urdf file as argument");
  return -1;
}
std::string urdf_file = argv[1];
```

`if (argc != 2){`：检查命令行参数的数量。期望传入一个 URDF 文件作为参数，因此参数数量应该为 2（程序名和 URDF 文件路径）。

`ROS_ERROR("Need a urdf file as argument");`：如果命令行参数数量不对（即不等于 2），使用 `ROS_ERROR` 打印错误信息，提示用户需要提供一个 URDF 文件路径。

`return -1;`：如果参数不正确，则退出程序，返回 -1 表示发生错误。

`std::string urdf_file = argv[1];`：获取命令行参数中 URDF 文件的路径，并将其存储在 `urdf_file` 字符串中。`argv[1]` 是第一个命令行参数（即 URDF 文件的路径）。

Parse a urdf file

```
urdf::Model model;
if (!model.initFile(urdf_file)){
 ROS_ERROR("Failed to parse urdf file");
 return -1;
}
```

`urdf::Model model;`：创建一个 `urdf::Model` 对象 `model`，该对象将用于加载和存储 URDF 文件的模型数据。

`if (!model.initFile(urdf_file)){`：调用 `model.initFile()` 函数加载 URDF 文件，并检查是否成功。`urdf_file` 是从命令行参数获取的 URDF 文件路径。如果加载失败，返回 `false`。

`ROS_ERROR("Failed to parse urdf file");`：如果解析 URDF 文件失败，使用 `ROS_ERROR` 打印错误信息，提示解析失败。

`return -1;`：如果解析 URDF 文件失败，则退出程序，返回 -1 表示发生错误。

Parse a urdf file

```
    ROS_INFO("Successfully parsed urdf file");
    return 0;
}
```

`ROS_INFO("Successfully parsed urdf file");`：如果 URDF 文件成功解析，使用 `ROS_INFO` 打印信息，表示成功解析文件。

`return 0;`：程序成功完成，返回 0，表示没有错误发生。

这段代码是一个简单的 ROS 节点，它的功能是从命令行参数获取一个 URDF 文件路径，并尝试解析该 URDF 文件。如果解析成功，它将输出一条成功的信息；如果解析失败，它将输出错误信息并退出程序。

Now let's try to run this code. First add the following lines to your CMakeList.txt file:

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

build your package, and run it.

```
$ cd ~/catkin_ws
$ catkin_make
$ .<path>/parser <path>my_robot.urdf
# Example: ./devel/lib/testbot_description/parser ./src/testbot_description/urdf/my_robot.urdf
```

The output should look something like this:

```
  [ INFO] 1254520129.560927000: Successfully parsed urdf file
```

Using urdf with robot_state_publisher