## ROS Robotics Projects Second Edition

### 第三章 建立移动机械臂

#### 目标

建立移动小车模型和机械臂模型

将两个模型连接到一起

#### 1.初始化工作空间

$ initros1 启动roscore

$ mkdir -p ~/chapter3\_ws/src

$ catkin\_init\_workspace 初始化工作空间

$ cd ~/chapter3\_ws/src

$ catkin\_create\_pkg robot\_description catkin 生成package

$ cd ~/chapter3\_ws/

$ catkin\_make

$ cd ~/chapter3\_ws/src/robot\_description/

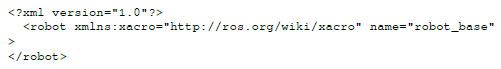
生成新的文件夹

$ mkdir config launch meshes urdf

将下载的几何文件mesh放入meshes文件夹中，是机械臂的外观文件

进入urdf文件夹，定义urdf

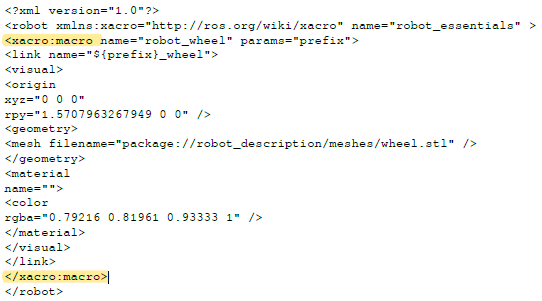
本书中均采用.urdf.xacro的形式，图片为必须有的XML版本和<robot>标签



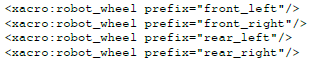
#### 2.定义连杆

在XML,<robot> 标签中定义连杆——原点，几何文件，材料，颜色等

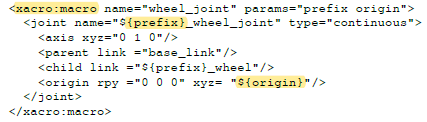
定义轮子的xacro



在其他urdf.xacro中调用机器人轮子的xacro



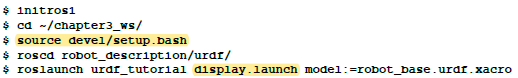
#### 3.定义关节的xacro，可用”${prefix}”等用来在调用时进行参数或者名称的直接定义



调用，可在调用时对待定义的参数或名称进行赋值



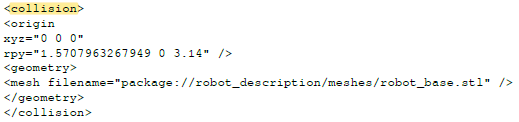
Source非常重要，roslaunch实现在rviz中的机器人可视化



以上已经有了ROS可理解的机器人模型，需要加入一些标签，使其在Gazebo中可见

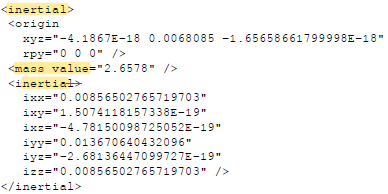
#### 4.定义碰撞collision

<collision>标签，在<link>标签中跟随<visual>标签进行定义



#### 5.定义物体特性-<inertial>标签

定义质量，惯性特征，在<link>标签中

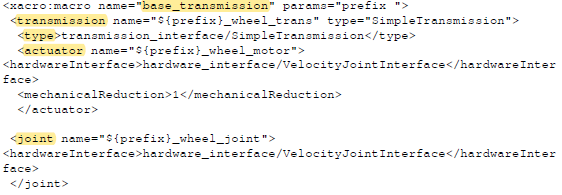


#### 6.定义驱动信息

采用xacro的形式定义

<transmission>标签——类型，驱动器

<joint>——采用的关节

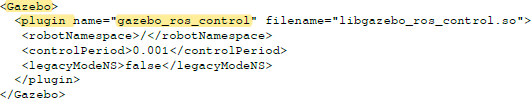


调用

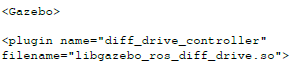


#### 7.定义ROS的控制器

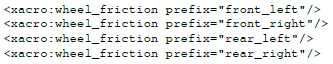
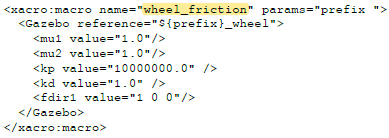
建立Gazebo和ROS直接的通讯，新建gazebo\_essentials\_base.xacro文件，其中包含<Gazebo>标签，加入gazebo\_ros\_control外挂程式



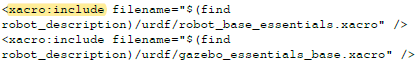
Differential drive外挂，微分运动

未截完全

用xacro方程定义轮子的摩擦特性，并调用

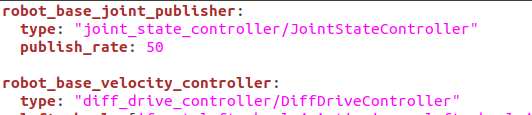


将定义的macros和Gazebo plugins加入机器人文件，输入以下代码在<robot>标签中



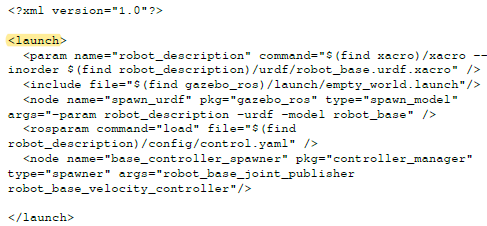
配置控制器，创建config文件，control.yaml，将代码复制进去



控制方法及细节？

创建一个launch文件





Gazebo中的可视化



另一个端，看目前的话题列表



采用rqt控制机器人转向



#### 8.开始机械臂的创建

连杆+驱动器+控制



左侧话题，用来命令和控制机器人，输出关节轨迹

joint\_trajectory\_controller：用来执行关节空间的轨迹

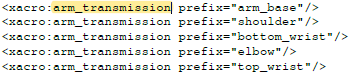
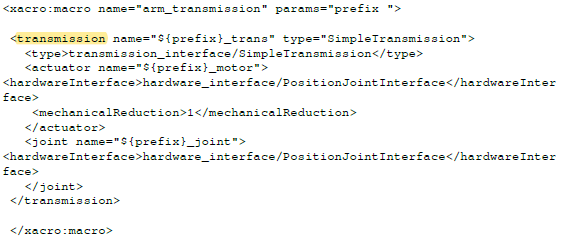
轨迹通过动作界面传输到控制器。control\_msgs::FollowJointTrajectoryAction，在follow\_joint\_trajectory的命名空间中

初始化



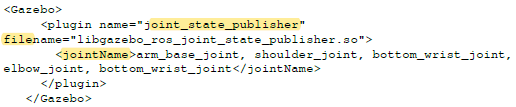
连杆、关节、碰撞、惯性

定义驱动器及调用



定义ROS\_controllers控制器

将joint\_state\_publisher（发布手臂连杆的状态信息）加入到gazebo\_essentials\_arm.sacro中



将定义好的宏和外挂程式加入<robot>



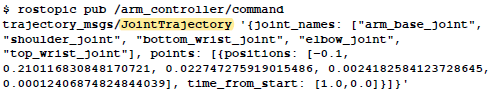
新的.yaml配置文件



测试机械臂，新的launch文件

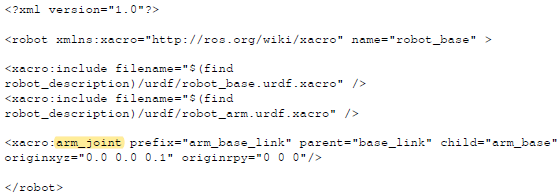


Gazebo中可视化（有问题，组件飞起来）

用命令控制机械臂运动

将机械臂和基座小车连接

新的文件mobile\_manipulator.urdf.xacro，调用小车和机械臂定义xmacro，定义一个连接二者的关节



在rviz中可视化



编辑新的.launch文件，用于gazebo中的control

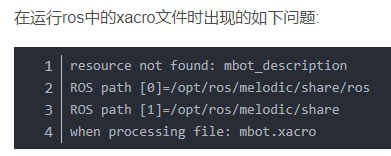


在Gazebo中显示，可控制？



#### 问题

**A. Resource not found**



解决代码：

sudo cp -r ~/catkin\_ws/src/mbot\_description /opt/ros/melodic/share

**B. No transform from[xxx] to [base\_link]**

需要安装Unicode后重启roscore

sudo apt-get install unicode

**C. 运行前，先在ws中输入**

source devel/setup.bash

随后，.launch文件和.urdf文件分别进入对应的文件夹开启终端再运行

**D. 新生成的Workspace，要进行catkin\_make，环境配置**

### 第四章 运用状态机处理复杂机器人任务

#### 目标

ROS中的action

服务机器人的比喻

状态机的介绍

SMACH的介绍

SMACH的案例

1.ROS中action介绍

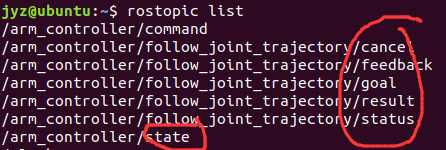
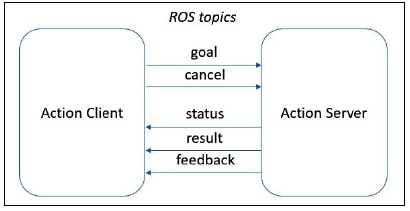
#### 1. ROS actions的介绍

对长时间或目标行为的实现方式，灵活

Client-server客户端和服务器：

客户端：发送控制信号

服务器：接受控制信号，提供必要反馈



为action implementation

采用Joint\_trajectory\_controller控制关节空间轨迹：

通过control\_msgs::FollowJointTrajectoryAction发送轨迹到controller，在其follow\_joint\_trajectory的动作界面中。

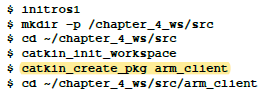
FollowJointTracjectoryAction（为动作服务器）是position\_controllers/JointTrajectoryController的结果，称为arm\_controller插件

#### 2. Actionlib的例子

**A.机械臂控制**

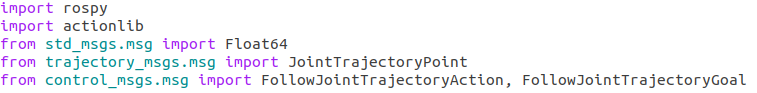
写一个client，发送goal到server

建立arm\_client的包



Arm\_action\_client.py文件，三部分

第一部分：必要的声明（使用ROS函数，action library和ROS messages）

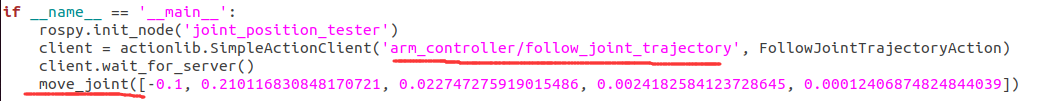


第二部分：

主程序中，初始化节点和客户端，发送目标，例子中使用SimpleActionClient

呼叫服务端名，arm\_controller/follow\_joint\_trajectory，在ros\_controller插件中定义；

等待服务端的回复，一旦受到了反馈，执行move\_joint函数

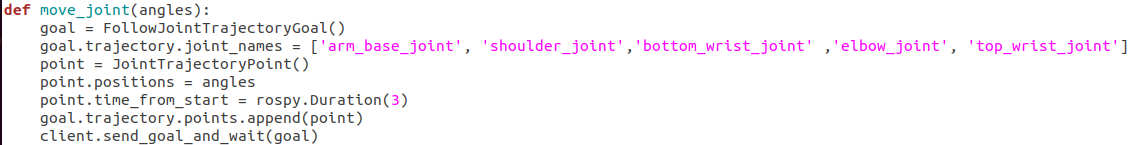


第三部分：

Move\_joint函数收到了各关节的角度，将其作为轨迹发送到机器人

通过FollowJointTrajectoryGoal()将动作目标传递

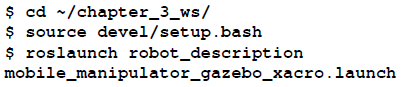
定义动作的参数：关节名字、point（角度）、执行时间



一直发送直到目标完成

此句定义结束的时间限制

使用步骤：

启动gazebo

开启动作客户端文件（需要在ws中catkin\_make）

**B.电池仿真服务端-客户端**

通过scratchs使用ROS action

流程：

创建package,创建action文件夹-创建action文件，包含goal,result,feedback—修改，编译—定义服务端—定义客户端

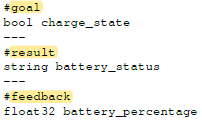
创建package



创建action文件夹



创建action文件，名称为battery\_sim.action，并定义goal,result,feedback

修改、编译

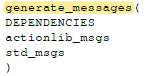
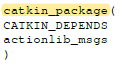
Package.xml文件中加入以下行



CmakeLists.txt加入action文件



在两个函数内声明依赖关系

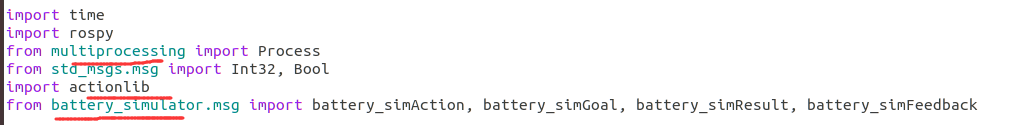
 

编译工作环境

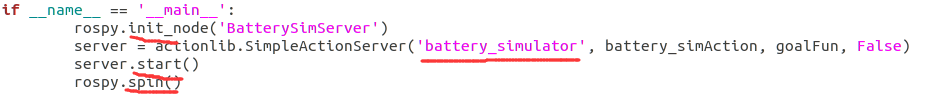


定义服务端server，文件名battery\_sim\_server.py

必要声明：multiprocessing多进程；从battery\_simulator包中发送actionlib信息



主函数：初始化节点和服务端，开启服务端，.spin为保持服务端运行



定义两个并行的函数，goalFun()和batterySim()

GoalFun()函数根据charge\_state设置为参数1和0，表示是否在充电中

BatterySim()函数检查参数1和0，根据charge\_state参数决定运行哪一个代码

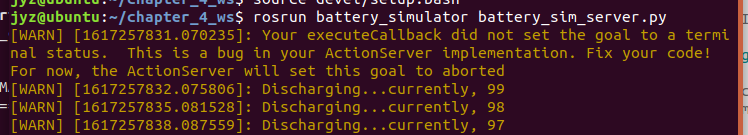
定义客户端client——battery\_sim\_client.py文件

类似前一个案例：包括必要声明、主程序、定义

运行

开启服务端

开启客户端，改为rosrun



#### 3. 餐厅服务生机器人简介

到桌点单，确认点单，送餐——伪代码

还需要监控电量，不够时要去充电，换另一个机器人替代

#### 4. 状态机state machine简介

将问题进行图形表达，任务分解成小的操作

框：代表一个特定状态

Edge：连接两个状态，表示两个状态间的转换

Edge上的值或描述：表示此状态的outcome（完成情况），成功失败或正在进行

#### 5. SMACH简介

SMACH是一个python库，处理复杂机器人行为，构建分层或同时的状态机

Smach\_roc package包，用来支持ROS的使用

概念：状态state，ROS中用execute()函数定义行为或action

Container：状态的集。表示执行的策略——分层的hierarchical，同时的concurrent，或分段的iterative

\*\*SMACH中的概念

Outcome：状态的可能结果-执行完动作后的状态反馈，用于状态转换。在execute()框后

User data：状态转换所需要的条件，input（执行需要的） and output（状态反馈的） keys

Preemption：抢占？是一种中断状态

Introspection：内省。用SMACH vierwer对定义的状态机进行可视化。必须定义一个introspection server才能可视化

#### 6. SMACH案例

官方案例 http://wiki.ros.org/cn/smach/Tutorials

安装

sudo apt-get install ros-melodic-smach ros-melodic-smach-ros ros-melodic-executive-smach ros-melodic-smach-viewer

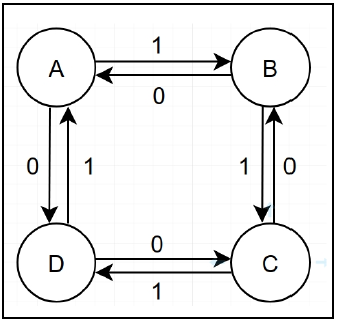
开启smach\_viewer窗口的命令

rosrun smach\_viewer smach\_viewer.py

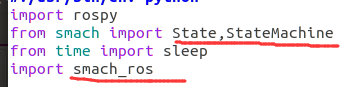
开启.py文件-在ws中

rosrun smach\_example（文件夹） simple\_fsm.py（文件）

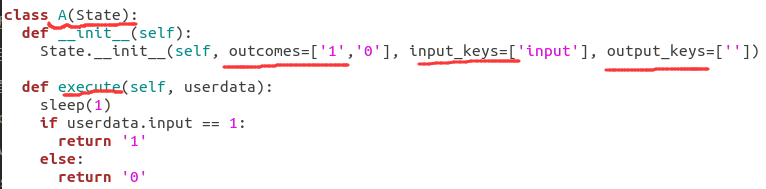
A.四个状态在受到输出output时的转换



文件名simple\_fsm.py



单独定义每一个状态，可以收到从user发出的input



上图，定义了状态A，结果outcome分别是0和1。定义状态需要的输入input通过input\_keys。Output\_keys空，不输入用户信息。Execute()为执行状态动作——user输入1，返回1；输入其他，则返回0.

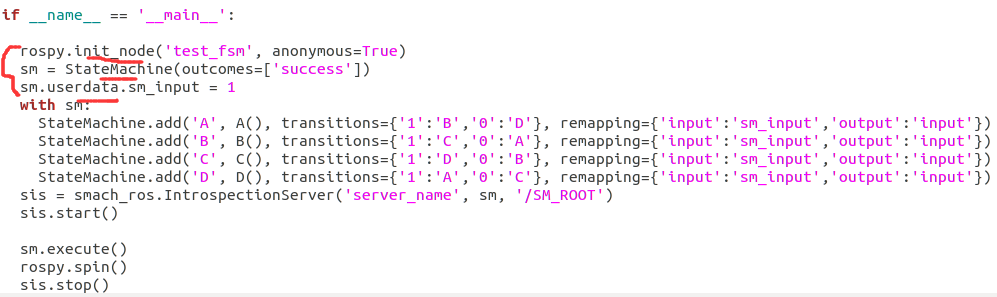
主函数：

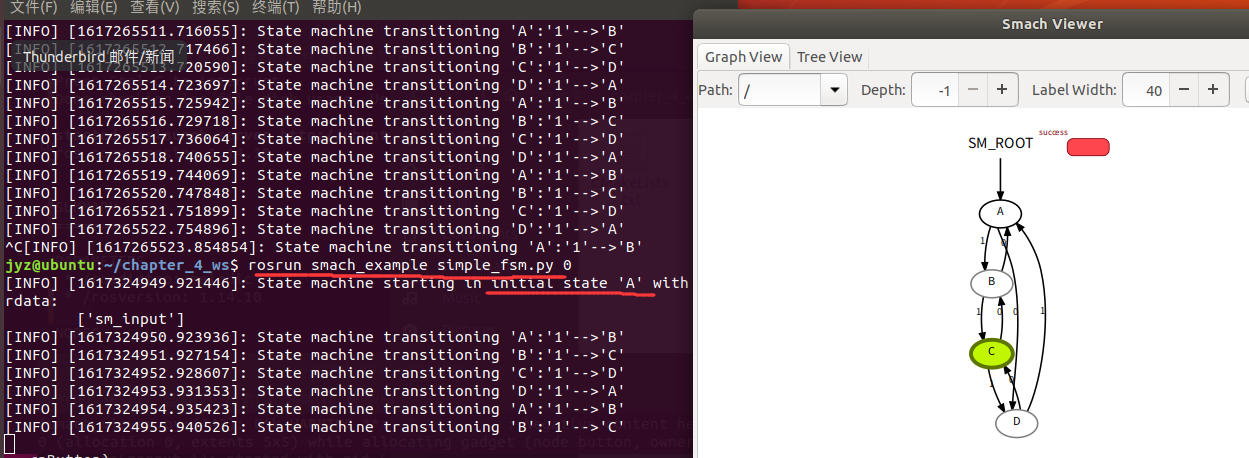
初始化node,状态机，分配user data给状态机

With()函数，加入先前状态，定义转换

要通过smach\_viewer包可视化状态机，需要call introspectionserver()通过servername,statemachine和root name根名字

随后执行状态机





#### 7. 状态机应用：餐厅机器人

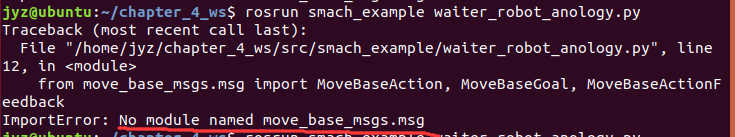
STATES\_MACHINES：各个状态的结合

定义各个状态：通过class XX(state)类函数，包括初始化和执行

定义机器人的运动状态：move\_base\_state（move\_base服务器接收信息，定义一个客户端发出状态）

给出goals目标

运行有问题



### 第五章 机器人的工业应用实例

机器人家中应用

使移动车和机械臂智能化，并仿真，改进

## ROS学习计划

1. ROSwiki中文版（阅读）
2. Moveit!创客智造+官方教程
3. Gazebo
4. 引入强化学习

## A(1).ROSwiki中文版初级教程

先创立工作空间：

在/src中创建软件包package：“catkin\_create\_pkg beginner\_tutorials std\_msgs rospy roscpp”，命令+名称+依赖关系。

# catkin\_create\_pkg <package\_name> [depend1] [depend2] [depend3]

依赖关系查询：

$ rospack depends beginner\_tutorials

编译软件包：

$ catkin\_make

$ source /opt/ros/<distro>/setup.bash 环境依赖项

文件package.xml：包含维护者、许可、编译和运行时的依赖项（build\_depend, run\_depend）

运行所有ROS程序前都要运行$ roscore

节点node之间通过topic话题进行通信（发布publish+订阅subscribe），可使用rqt\_graph图进行查看。

$ rostopic 获取ROS话题的信息，举例：

$ rostopic echo [topic] 显示某话题上发布的数据

$ rostopic echo /turtle1/cmd\_vel 可在公屏显示速度指令

$ rostopic list 列出所有被订阅和发布的话题

订阅和发布需要采用相同类型的信息

$ rostopic type [topic] 查看所发布话题的消息类型

$ rosmsg show [以上消息类型] 消息的详细信息

$ rostopic pub将数据发布到正在广播的话题上

rostopic pub [topic] [msg\_type] [args]

$ rostopic pub -1 /turtle1/cmd\_vel geometry\_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

解释：-1，只发布一条消息，若要一直发布，则为-r；/turtle1/cmd\_vel，话题topic的名称；--，说明后面的参数不是选项。根据数据类型写后面的数值。

$ rostopic hz [topic] 报告数据发布速率

$ rosrun rqt\_plot rqt\_plot 滚动时间图显示某话题的数据

$ rosservice 服务为节点通讯的另一个方式，节点发送请求request，获得响应response。

$ rosservice list 输出活跃服务的信息

$ rosservice type [service] 查看服务的类型

$ rosservice call [service] [args] 调用服务及其数值

$ rosparam 在参数服务器存储和操作数据，YAML语法

$ rosparam list

$ rosparam set [param\_name] 设置参数值，随后$ rosservice call /clear使其生效

$ rosparam get [param\_name] 查看参数值

$ rosparam get / 查看所有服务器上的参数值

$ rosparam dump [file\_name] [namespace] 将参数写入文件，如params.yaml

$ rosparam load [file\_name] [namespace] 将yaml文件载入新的空间

$ roslaunch [package] [filename.launch] 采用.launch文件同时启动多个节点

.launch文件中，包含分组<group>和使用的节点<node>

Msg消息文件和srv服务文件（有请求和相应）

在package.xml中，以下两行不能被注释：

<build\_depend>message\_generation</build\_depend>

<exec\_depend>message\_runtime</exec\_depend>

在CmakeLists.txt中，在find\_package字段中添加message\_generation依赖项

取消以下几行的注释，并替换为文件名.msg。

# add\_message\_files(

# FILES

# Message1.msg

# Message2.msg

# )

$ rosmsg show [message type] 查看ROS能否识别该msg。

Srv则替换以下几行

# add\_service\_files(

# FILES

# Service1.srv

# Service2.srv

# )

$ rossrv show <service type>

对于命令的帮助，可以键入-h

如$ rosmsg -h

### 对于发布者和订阅者的编写（python）

发布者节点lker的Python脚本scripts。

[1](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_1) #!/usr/bin/env python 必要声明，确保为Python脚本

[2](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_2) # license removed for brevity

[3](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_3) import rospy # ROS节点需要导入rospy

[4](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_4) from std\_msgs.msg import String # 可使用std\_msgs/String消息类型

[5](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_5)

[6](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_6) def talker():

[7](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_7) pub = rospy.Publisher('chatter', String, queue\_size=10)

# 定义talker与ROS部分的接口，节点使用String类型发布到chatter话题

[8](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_8) rospy.init\_node('talker', anonymous=True)

# 将节点名称告诉rospy，从而建立与主节点的通信

# 后面的True会自动添加随机数，使节点名称唯一

[9](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_9) rate = rospy.Rate(10) # 10hz

# 速度循环

[10](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_10) while not rospy.is\_shutdown():

[11](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_11) hello\_str = "hello world %s" % rospy.get\_time()

[12](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_12) rospy.loginfo(hello\_str)

[13](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_13) pub.publish(hello\_str)

[14](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_14) rate.sleep()

# 标准的rospy结构。检查是否is\_shutdown，进而执行代码逻辑

# pub.publish 将字符串发布到chatter话题

# rospy.loginfo 打印消息到屏幕；写入节点日志；写入rosout

[15](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_15)

[16](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_16) if \_\_name\_\_ == '\_\_main\_\_':

[17](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_17) try:

[18](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_18) talker()

[19](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_19) except rospy.ROSInterruptException:

[20](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-c82832e0d612370fe9886563f0b7f5433f6caee1_20) pass

# 标准的python \_\_main\_\_检查

将以下内容添加到Cmakelists.txt中编辑调用

catkin\_install\_python(PROGRAMS scripts/talker.py

DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}

)

订阅者节点stener的编写脚本

[1](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_1) #!/usr/bin/env python

[2](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_2) import rospy

[3](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_3) from std\_msgs.msg import String

[4](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_4)

[5](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_5) def callback(data):

[6](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_6) rospy.loginfo(rospy.get\_caller\_id() + "I heard %s", data.data)

[7](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_7)

[8](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_8) def listener():

[9](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_9)

[10](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_10) # In ROS, nodes are uniquely named. If two nodes with the same

[11](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_11) # name are launched, the previous one is kicked off. The

[12](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_12) # anonymous=True flag means that rospy will choose a unique

[13](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_13) # name for our 'listener' node so that multiple listeners can

[14](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_14) # run simultaneously.

[15](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_15) rospy.init\_node('listener', anonymous=True)

[16](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_16) # rospy给listener生成唯一节点名称，使得可以多个listener.py一起运行

[17](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_17) rospy.Subscriber("chatter", String, callback)

[18](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_18) # 声明订阅了chatter话题，接受新消息，则callback函数被调用

[19](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_19) # spin() simply keeps python from exiting until this node is stopped

[20](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_20) rospy.spin()

# 直到节点被明确关闭，才退出

[21](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_21)

[22](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_22) if \_\_name\_\_ == '\_\_main\_\_':

[23](http://wiki.ros.org/cn/ROS/Tutorials/WritingPublisherSubscriber%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingPublisherSubscriber.CA-56fb82d4681d7880a3d3a97f7af70cfe17618f86_23) listener()

随后将以下内容添加到Cmakelists.txt中编辑调用

catkin\_install\_python(PROGRAMS scripts/talker.py scripts/listener.py

DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}

)

测试

$ rosrun beginner\_tutorials talker.py # (Python)

$ rosrun beginner\_tutorials listener.py # (Python)

### 编写服务和客户端（python）

服务节点add\_two\_inis\_server，接受两个整数，返回和。

[1](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_1) #!/usr/bin/env python

[2](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_2)

[3](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_3) from \_\_future\_\_ import print\_function

[4](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_4)

[5](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_5) from beginner\_tutorials.srv import AddTwoInts,AddTwoIntsResponse

[6](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_6) import rospy

[7](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_7)

[8](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_8) def handle\_add\_two\_ints(req):

[9](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_9) print("Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b)))

[10](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_10) return AddTwoIntsResponse(req.a + req.b)

[11](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_11)

[12](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_12) def add\_two\_ints\_server():

[13](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_13) rospy.init\_node('add\_two\_ints\_server')

# 声明节点

[14](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_14) s = rospy.Service('add\_two\_ints', AddTwoInts, handle\_add\_two\_ints)

# 声明服务（’服务名’,服务类型,函数）

[15](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_15) print("Ready to add two ints.")

[16](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_16) rospy.spin()

[17](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_17)

[18](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_18) if \_\_name\_\_ == "\_\_main\_\_":

[19](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-1b44a8b12515c870975d3d35cd5cd98f6145a087_19) add\_two\_ints\_server()

给节点执行权限

chmod +x scripts/add\_two\_ints\_server.py

添加至CmakeLists.txt

catkin\_install\_python(PROGRAMS scripts/add\_two\_ints\_server.py

DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}

)

客户端client节点

[1](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_1) #!/usr/bin/env python

[2](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_2)

[3](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_3) from \_\_future\_\_ import print\_function

[4](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_4)

[5](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_5) import sys

[6](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_6) import rospy

[7](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_7) from beginner\_tutorials.srv import \*

[8](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_8)

[9](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_9) def add\_two\_ints\_client(x, y):

[10](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_10) rospy.wait\_for\_service('add\_two\_ints')

# 调用服务

[11](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_11) try:

[12](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_12) add\_two\_ints = rospy.ServiceProxy('add\_two\_ints', AddTwoInts)

# 为服务的调用创建句柄handle

[13](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_13) resp1 = add\_two\_ints(x, y)

[14](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_14) return resp1.sum

# 使用句柄

[15](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_15) except rospy.ServiceException as e:

[16](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_16) print("Service call failed: %s"%e)

[17](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_17)

[18](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_18) def usage():

[19](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_19) return "%s [x y]"%sys.argv[0]

[20](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_20)

[21](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_21) if \_\_name\_\_ == "\_\_main\_\_":

[22](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_22) if len(sys.argv) == 3:

[23](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_23) x = int(sys.argv[1])

[24](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_24) y = int(sys.argv[2])

[25](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_25) else:

[26](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_26) print(usage())

[27](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_27) sys.exit(1)

[28](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_28) print("Requesting %s+%s"%(x, y))

[29](http://wiki.ros.org/cn/ROS/Tutorials/WritingServiceClient%28python%29#cn.2Frospy_tutorials.2FTutorials.2FWritingServiceClient.CA-ac633897e45f717b1a6ad7685dbe0a93c834abbd_29) print("%s + %s = %s"%(x, y, add\_two\_ints\_client(x, y)))

节点执行权限

$ chmod +x scripts/add\_two\_ints\_client.py

添加至CmakeLists.txt

catkin\_install\_python(PROGRAMS scripts/add\_two\_ints\_server.py scripts/add\_two\_ints\_client.py

DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}

)

检验

$ rosrun beginner\_tutorials add\_two\_ints\_server.py # (Python)

返回：

Ready to add two ints.

$ rosrun beginner\_tutorials add\_two\_ints\_client.py 1 3 # (Python)

返回：

Requesting 1+3

1 + 3 = 4

### 数据的录制和回放

检查当前所有发布中的话题

rostopic list -v

只有在此列表中的消息才可以被记录在bag中

生成记录的bag文件

mkdir ~/bagfiles

cd ~/bagfiles

rosbag record -a

检查bag文件

rosbag info <your bagfile> # 基本信息：包含的话题名称，类型和消息数量

rosbag play <your bagfile> # 回放数据，再现系统运行过程

rosbag record -O subset /turtle1/cmd\_vel /turtle1/pose # 录制特定的数据在subset.bag中

局限性：受到系统定时精度的影响，形状并不完全一样，不能完美地模仿系统行为

从bag中读取消息

A.一种：

rostopic echo /obs1/gps/fix | tee topic1.yaml

订阅 /obs1/gps/fix话题，复读所有内容，用tee命令储存到一个yaml格式文件中。

在另一个终端上回放bag：

time rosbag play --immediate demo.bag --topics /topic1 /topic2 /topic3 /topicN

B.另一种

使用ros\_readbagfile脚本，更快，可同时读取任意多的话题

ros\_readbagfile <mybagfile.bag> [topic1] [topic2] [topic3] [...]

### roswtf入门

在roscore没有运行时，检查当前系统是否存在问题。

安装检查

$ roscd rosmaster

$ roswtf

在线检查（启动roscore）

$ roscd

$ roswtf

## A(2).ROSwiki中文版中级教程

### 手动创建package

在工作空间的src中Catkin\_create\_pkg创建包后，要配置package.xml（配置依赖项）和CmakeList.txt（使得catkin\_make可以利用Cmake的跨平台特性来编译创建的包）。

Package.xml

<package>

<name>foobar</name>

<version>1.2.4</version>

<description>

This package provides foo capability.

</description>

<maintainer email="foobar@foo.bar.willowgarage.com">PR-foobar</maintainer>

<license>BSD</license>

<buildtool\_depend>catkin</buildtool\_depend>

<build\_depend>roscpp</build\_depend>

<build\_depend>std\_msgs</build\_depend>

<run\_depend>roscpp</run\_depend>

<run\_depend>std\_msgs</run\_depend>

</package>

CmakeList.txt

cmake\_minimum\_required(VERSION 2.8.3)

project(foobar)

find\_package(catkin REQUIRED roscpp std\_msgs)

catkin\_package()

### 管理系统依赖项

软件包所需的外部函数库，称为“系统依赖项”

$ rosdep install [package] 下载斌安装所需的系统依赖项小工具

### Roslaunch在大型项目中的使用技巧

以二维的PR2导航为例（2dnav\_pr2）。2dnav\_pr2.launch

希望构建launch文件，使其最大化的重用。

高层次结构尽量简短，利用include指令将系统组成部分和ROS parameter引用即可。

<launch>

<group name="wg">

<include file="$(find pr2\_alpha)/$(env ROBOT).machine" />

# (env ROBOT)置换符，使用ROBOT变量的值-该符号也可以使用其他的依赖环境变量的值

<include file="$(find 2dnav\_pr2)/config/new\_amcl\_node.xml" />

<include file="$(find 2dnav\_pr2)/config/base\_odom\_teleop.xml" />

<include file="$(find 2dnav\_pr2)/config/lasers\_and\_filters.xml" />

<include file="$(find 2dnav\_pr2)/config/map\_server.xml" />

<include file="$(find 2dnav\_pr2)/config/ground\_plane.xml" />

<!-- The navigation stack and associated parameters -->

<include file="$(find 2dnav\_pr2)/move\_base/move\_base.xml" />

</group>

</launch>

Pre.machine文件

<launch>

<machine name="c1" address="pre1" ros-root="$(env ROS\_ROOT)" ros-package-path="$(env ROS\_PACKAGE\_PATH)" default="true" />

<machine name="c2" address="pre2" ros-root="$(env ROS\_ROOT)" ros-package-path="$(env ROS\_PACKAGE\_PATH)" />

</launch>

对机器人名进行映射，将c1对应于机器名pre1。

如其他文件中的语句，控制amcl节点在名为c1的机器上允许：

<node pkg="amcl" type="amcl" name="amcl" machine="c1">

Move\_base.xml的一部分

<node pkg="move\_base" type="move\_base" name="move\_base" machine="c2">

<remap from="odom" to="pr2\_base\_odometry/odom" />

# remapping,从odom接收里程计信息，该信息发布在pr2\_base\_odometry话题上

<param name="controller\_frequency" value="10.0" />

<param name="footprint\_padding" value="0.015" />

<param name="controller\_patience" value="15.0" />

<param name="clearing\_radius" value="0.59" />

# <param> 节点的内部参数，私有

<rosparam file="$(find 2dnav\_pr2)/config/costmap\_common\_params.yaml" command="load" ns="global\_costmap" />

<rosparam file="$(find 2dnav\_pr2)/config/costmap\_common\_params.yaml" command="load" ns="local\_costmap" />

<rosparam file="$(find 2dnav\_pr2)/move\_base/local\_costmap\_params.yaml" command="load" />

<rosparam file="$(find 2dnav\_pr2)/move\_base/global\_costmap\_params.yaml" command="load" />

<rosparam file="$(find 2dnav\_pr2)/move\_base/navfn\_params.yaml" command="load" />

<rosparam file="$(find 2dnav\_pr2)/move\_base/base\_local\_planner\_params.yaml" command="load" />

# <rosparam> 从yaml文件中读取参数

</node>

### ROS在多机器人上的使用

只需要一个master，所有节点通过配置ROS\_MASTER\_URI连接到同一个master。

### 自定义消息

.msg文件放置package的msg文件夹中。

消息类型：

C++

[1](http://wiki.ros.org/cn/ROS/Tutorials/DefiningCustomMessages#CA-8c8acf0cea72e86f6fa563c6cb301f649db09d79_1) #include <std\_msgs/String.h>

[2](http://wiki.ros.org/cn/ROS/Tutorials/DefiningCustomMessages#CA-8c8acf0cea72e86f6fa563c6cb301f649db09d79_2)

[3](http://wiki.ros.org/cn/ROS/Tutorials/DefiningCustomMessages#CA-8c8acf0cea72e86f6fa563c6cb301f649db09d79_3) std\_msgs::String msg;

Python

[1](http://wiki.ros.org/cn/ROS/Tutorials/DefiningCustomMessages#CA-3370392f843a00ef86c85a9b0065a438aeca9ac2_1) from std\_msgs.msg import String

[2](http://wiki.ros.org/cn/ROS/Tutorials/DefiningCustomMessages#CA-3370392f843a00ef86c85a9b0065a438aeca9ac2_2)

[3](http://wiki.ros.org/cn/ROS/Tutorials/DefiningCustomMessages#CA-3370392f843a00ef86c85a9b0065a438aeca9ac2_3) msg = String()

依赖项要添加到package.xml中

<build\_depend>name\_of\_package\_containing\_custom\_msg</build\_depend>

<run\_depend>name\_of\_package\_containing\_custom\_msg</run\_depend>

## A(3). Ros\_control

官方ros\_control

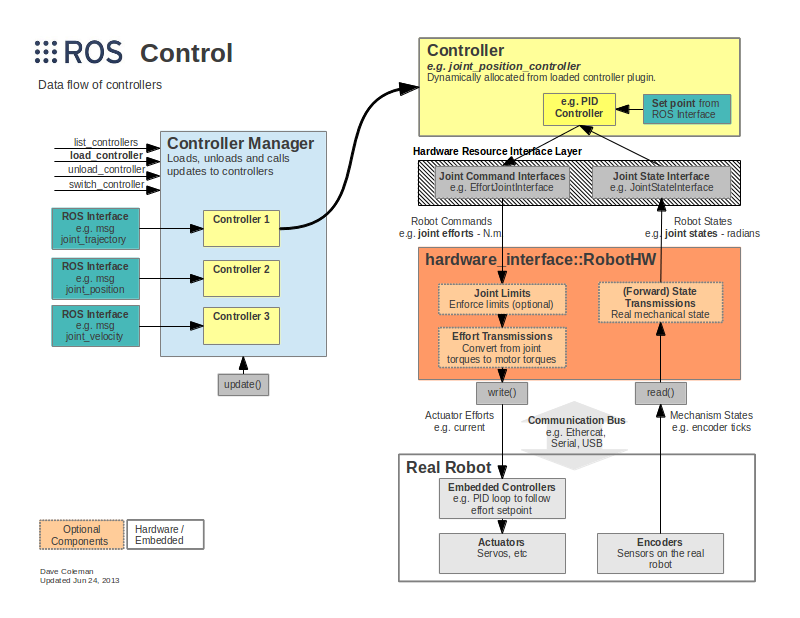
<http://wiki.ros.org/ros_control>

古月居ros\_control

<https://www.baidu.com/link?url=WgVbSpfbmY8kBb1AzAtYxlxxevkeD-pA7F_BWZ0eb_M88cP0obYI5g-VdHPxSG5o&wd=&eqid=f90a543b00050cd20000000361d6d65d>

控制软件包，包含控制器接口（controller interfaces），控制器管理（controller managers），传动装置接口（transmissions）和硬件接口（hardware\_interfaces）等。

### 1.总体框架



不同层次的功能：

Controller Manager：每个机器人可能有多个controller，所以这里有一个控制器管理器的概念，提供一种通用的接口来管理不同的controller。controller manager的输入就是ROS上层应用的输出。

Controller：controller可以完成每个joint的控制，请求下层的硬件资源，并且提供了PID控制器，读取硬件资源接口中的状态，在发布控制命令。

Hardware Rescource：为上下两层提供硬件资源的接口。

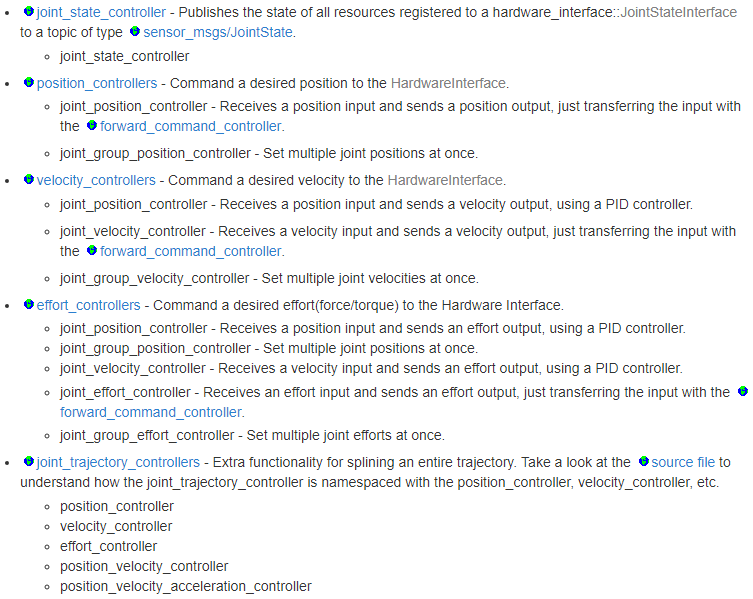
RobotHW：硬件抽象层和硬件直接打交道，通过write和read方法来完成硬件的操作，这一层也包含关节限位、力矩转换、状态转换等功能。

Real Robot：实际的机器人上也需要有自己的嵌入式控制器，接收到命令后需要反映到执行器上，比如接收到位置1的命令后，那就需要让执行器快速、稳定的到达位置1。

### 2. controllers

Ros\_controllers功能包提供已有的一些controllers：

State, position, velocity, effort, trajectory

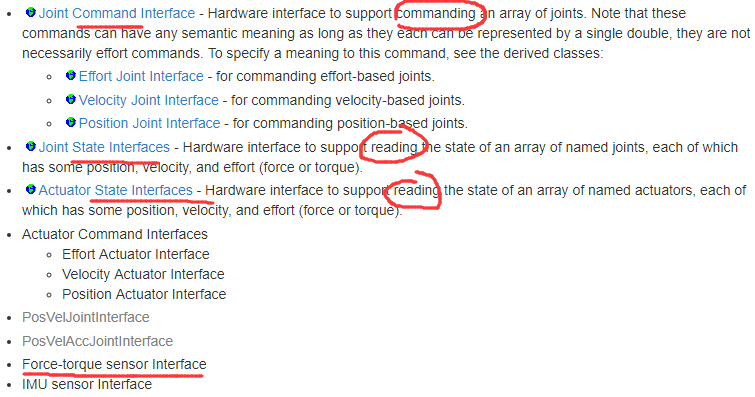


也可以根据需求，自己创建controller。

### 3. 硬件接口hardware interfaces

沟通controller和RobotHW的接口，和controllers的种类是对应的。也可以自己创建。

分为命令接口command interface和状态读取接口state interface。



### 4.传动transmissions

配置机器人关节的传动系统，一般在URDF文件中直接添加。

能量守恒P\_in=P\_out

F\_IN\*V\_IN=F\_OUT\*V\_OUT

减速比的计算

effort map: F\_joint = F\_actuator \* n

flow map: V\_joint = V\_actuator / n

URDF示例代码：

[1](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_1) <transmission name="simple\_trans">

[2](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_2) <type>transmission\_interface/SimpleTransmission</type>

# 明确传动类型

[3](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_3) <joint name="foo\_joint">

# 关节名称

[4](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_4) <hardwareInterface>EffortJointInterface</hardwareInterface>

# 硬件接口

[5](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_5) </joint>

[6](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_6) <actuator name="foo\_motor">

# 连接的驱动器名称

[7](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_7) <mechanicalReduction>50</mechanicalReduction>

# 减速比

[8](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_8) <hardwareInterface>EffortJointInterface</hardwareInterface>

# 硬件接口

[9](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_9) </actuator>

[10](https://wiki.ros.org/urdf/XML/Transmission#CA-e947bf6860ec8d8d47a567655431163a1604b519_10) </transmission>

传动类型<type>包含：

* Simple Reduction Transmission
* Differential Transmission
* Four Bar Linkage Transmission

为了让Gazebo识别<transmission>标签，要在URDF文件.xacro中，加入：

<gazebo>

<plugin name="gazebo\_ros\_control" filename="libgazebo\_ros\_control.so">

<robotNamespace>/rrbot</robotNamespace>

</plugin>

</gazebo>

### 5.关节限制joint limits

Joint limits是硬件抽象层RobotHW的一部分。限位数据可以从机器人的URDF文件中加载，也可以ROS的参数服务器上加载（先用YAML配置文件导入ROS parameter server），这些限位数据不仅包含关节速度、位置、加速度、加加速度、力矩等方面的限位，还包含安全作用的位置软限位、速度边界（k\_v）和位置边界（k\_p）等等。

以URDF中的参数设置为例：

<joint name="$foo\_joint" type="revolute">

<!-- other joint description elements -->

<!-- Joint limits -->

<limit lower="0.0"

upper="1.0"

effort="10.0"

velocity="5.0" />

<!-- Soft limits -->

<safety\_controller k\_position="100"

k\_velocity="10"

soft\_lower\_limit="0.1"

soft\_upper\_limit="0.9" />

</joint>

也可以在代码中采用joint\_limits\_interface来加载和设置限位参数。

两种加载模式的区别：

* Loading from URDF： 用于从URDF文件加载joint limits information(position, velocity and effort)和soft joint limits信息
* Loading from ROS params(YAML) 用于从ROS参数服务器加载joint limits(position, velocity, acceleration, jerk and effort). 参数规范同 MoveIt定义

### 6. controller\_manager

多个controller的控制机制，加载、开始和停止运行、卸载controller。

#### （1）命令行

$ rosrun controller\_manager controller\_manager <command> <name1>

支持的命令command：

* load: load a controller (construct and initialize)
* unload: unload a controller (destruct)
* start: start a controller
* stop: stop a controller
* spawn: load and start a controller
* kill: stop and unload a controller

若想获得所有controllers的状态：

$ rosrun controller\_manager controller\_manager <command>

命令包括

* List: 依照执行顺序列出所有的controllers，并给出状态
* List-type
* Reload-libraries：重新加载所有的可行插件的控制器库
* Reload-libraries -restore:重新加载所有的可行插件的控制器库，并恢复初始值

如果控制器过多，如六轴机器人有六个controllers。采用“spawner”命令同时控制

$ rosrun controller\_manager spawner [--stopped] <name1> <name2> ...

没有[--stopped]，则自动加载、启动controller；加上[--stopped]，只会加载，不运行。

Melodic中，新增controller\_groups，允许在运行中（at run time）从多个控制组中进行切换。

$ rosrun controller\_manager controller\_group <command> <args>

Command lists：

* List: 列出所有controller\_goups中的参数
* Spawn <group>：加载并开启所有<group>组中的controllers
* Switch <group>：切换到<group>组中的controllers

#### （2）launch工具

在launch文件中，加载和启动controllers。

<launch>

<node pkg="controller\_manager"

type="spawner"

args="controller\_name1 controller\_name2" />

</launch>

### 实例（古月居）

#### Transmission

URDF文件.xacro，<transmission>定义：

<transmission name="tran1">

<type>transmission\_interface/SimpleTransmission</type>

<joint name="joint1">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="motor1">

<hardwareInterface>EffortJointInterface</hardwareInterface>

<mechanicalReduction>1</mechanicalReduction>

</actuator>

</transmission>

<transmission name="tran2">

<type>transmission\_interface/SimpleTransmission</type>

<joint name="joint2">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="motor2">

<hardwareInterface>EffortJointInterface</hardwareInterface>

<mechanicalReduction>1</mechanicalReduction>

</actuator>

</transmission>

添加Gazebo识别插件

<gazebo>

<plugin name="gazebo\_ros\_control" filename="libgazebo\_ros\_control.so">

<robotNamespace>/rrbot</robotNamespace>

</plugin>

</gazebo>

#### Controller和controller\_manager

采用.yaml文件声明所需的controllers和对应参数

rrbot:

# Publish all joint states -----------------------------------

joint\_state\_controller:

type: joint\_state\_controller/JointStateController

publish\_rate: 50

# Position Controllers ---------------------------------------

joint1\_position\_controller:

type: effort\_controllers/JointPositionController

joint: joint1

pid: {p: 100.0, i: 0.01, d: 10.0}

joint2\_position\_controller:

type: effort\_controllers/JointPositionController

joint: joint2

pid: {p: 100.0, i: 0.01, d: 10.0}

使用launch文件运行controller\_manager中的spawner，加载并运行以上controllers。

<launch>

<!-- Load joint controller configurations from YAML file to parameter server -->

<rosparam file="$(find rrbot\_control)/config/rrbot\_control.yaml" command="load"/>

<!-- load the controllers -->

<node name="controller\_spawner" pkg="controller\_manager" type="spawner" respawn="false"

output="screen" ns="/rrbot" args="joint1\_position\_controller joint2\_position\_controller joint\_state\_controller"/>

<!-- convert joint states to TF transforms for rviz, etc -->

<node name="robot\_state\_publisher" pkg="robot\_state\_publisher" type="robot\_state\_publisher"

respawn="false" output="screen">

<remap from="/joint\_states" to="/rrbot/joint\_states" />

</node>

</launch>

在gazebo中启动机器人rrbot并开始控制：

$ roslaunch rrbot\_gazebo rrbot\_world.launch

$ roslaunch rrbot\_control rrbot\_control.launch

运动命令：

$ rostopic pub -1 /rrbot/joint1\_position\_controller/command std\_msgs/Float64 "data: 1.5"

$ rostopic pub -1 /rrbot/joint2\_position\_controller/command std\_msgs/Float64 "data: 1.0"

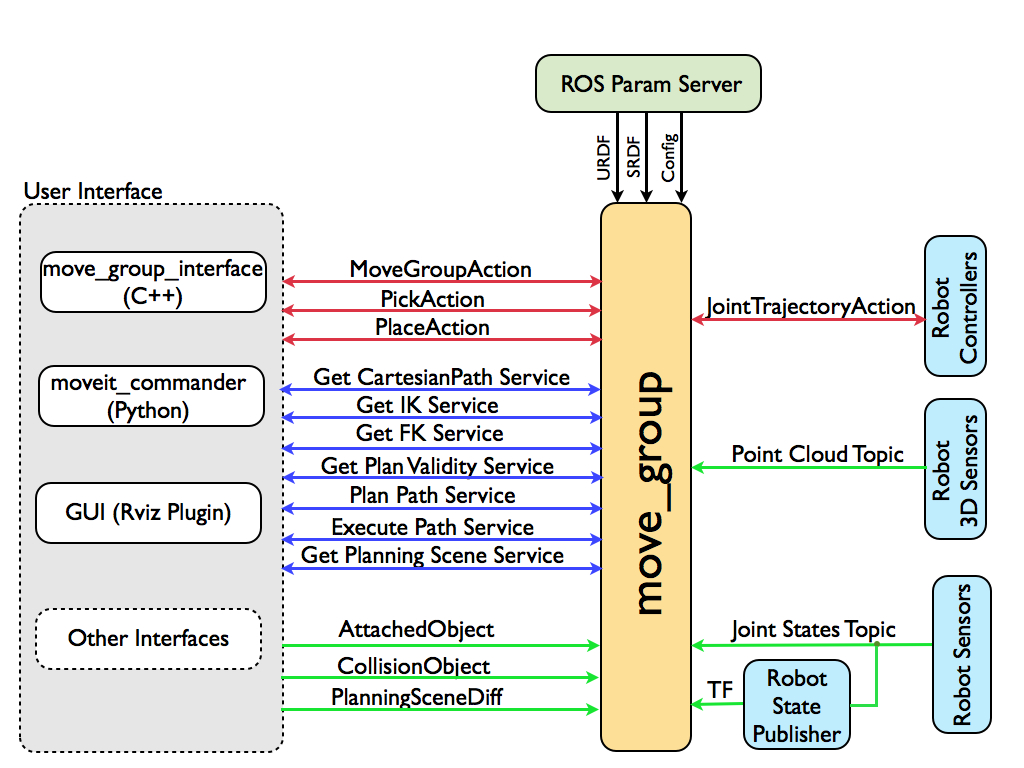
## B(1). MoveIt!创客智造

### 补充：Ubuntu18.04+Melodic安装带有PR2的moveit!

https://guyuehome.com/34562

### 1.简介

运动规划，操作，控制，三维感知，运动学。



#### 系统结构

Move\_group节点：整合器，提供actionhe service.

User Interface：用户接口，有C++（move\_group\_interface），Python（moveit\_commander）和GUI。

配置Move\_group，param server：参数，获取URDF，SRDF，MoveIt! Configuration，由MoveIt! Setup Assistant生成，存放在MoveIt! config目录中。

Joint State Information（节点状态信息）

Transform Information（变换信息）：Move\_group通过ROS TF库来监视变换信息，获取全局的姿态信息。

Controller Interface （控制器接口）：通过ROS的action接口，FollowJointTrajectoryAction接口来使用控制器。

Planning Scene（规划场景）：场景是世界的和机器人的状态的表现。

Extensible Capabilities（可扩展能力）：插件可经由一系列的ROS yaml parameters 和ROS pluginlib库配置。

#### 运动规划

The Motion Planning Plugin（运动规划插件）：针对 move\_group的默认规划器通过MoveIt! Setup Assistant来配置使用OMPL或OMPL的MoveIt!接口。

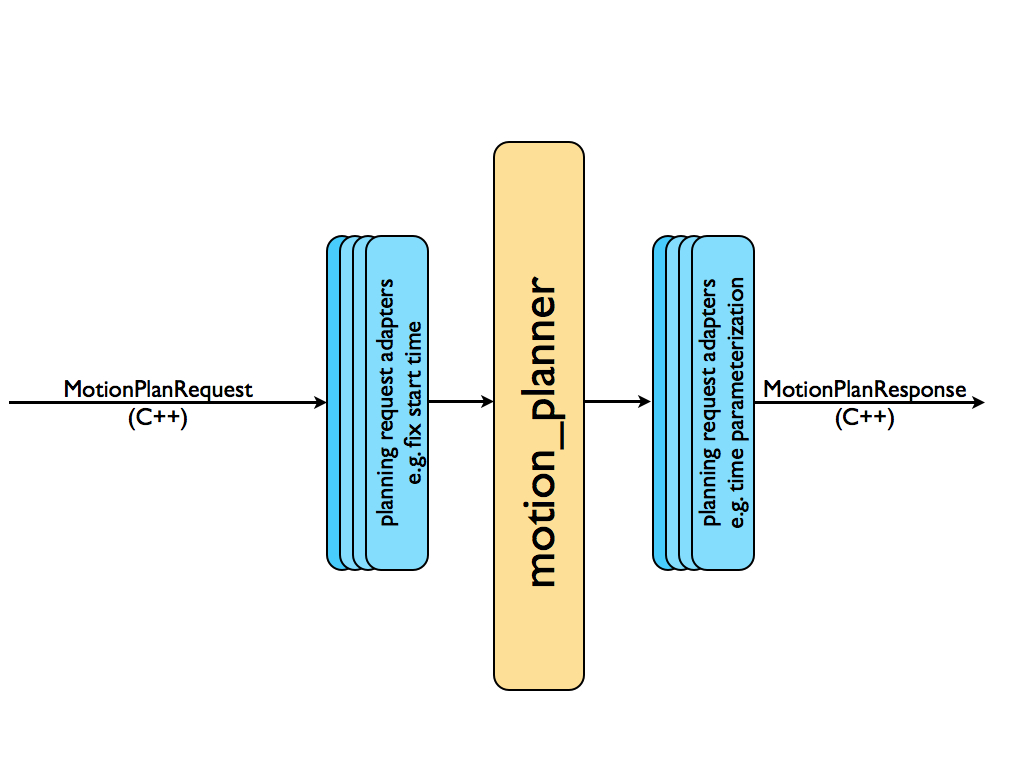
The Motion Plan Request（运动规划请求）：默认碰撞检测，可以将对象附加到末端执行器上，运动规划器也要考虑。需要检查的约束由kinematic constraints提供：

* Position constraints（位置约束） - 限制连接的位置在某个空间区域
* Orientation constraints（方向约束）-限制连接的方向在指定的roll, pitch和yaw范围
* Visibility constraints（可视化约束）-限制连接的点在特定传感器的可视化的锥形范围
* Joint constraints（节点约束） - 限制节点位于两个值之间
* User-specified constraints（自定义约束） - 利用自定义回调函数来指定自定义的约束。

The Motion Plan Result（运动规划结果）：move\_group节点会根据的运动规划请求，产生一个期望的轨迹。轨迹不仅仅是路径 - move\_group会以希望最大的速度和加速度（需指定）来生成用于轨迹。此轨迹需要遵循速度和加速度的限制。

#### The Motion Planning Pipeline（运动规划管道）

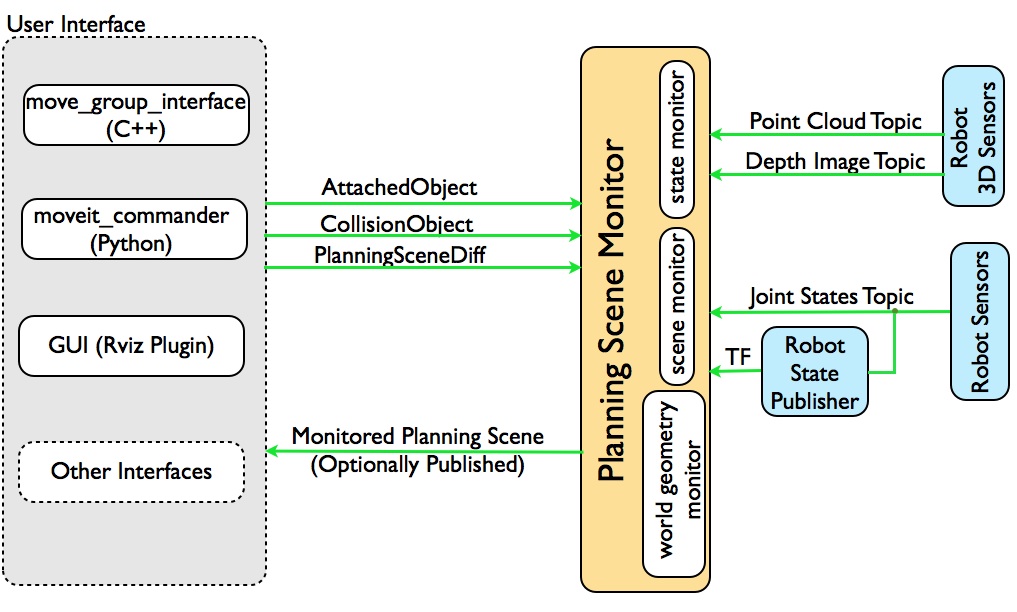
运动规划器motion\_planner和规划请求适配器plan request adapters。



规划请求适配器中，包含规划请求预处理和规划反馈后处理：预处理在某些情况有用，例如：机器人的起始状态稍微超出关节限制之外的情况。后处理需要处理几个操作，例如：转换生成的路径为带时间参数的轨迹。

#### OMPL

OMPL为开源运动规划库，moveit直接整合OMPL。



规划场景：状态信息，传感器信息，世界几何信息。

3D感知：由occupancy map monitor处理，它使用插件结构处理不同的传感器输入。MoveIt有两个内置支持可以处理两种输入，点云和深度图像。

运动学：可自己编写逆运动学算法。IKFast插件很不错。免检冲突矩阵（allowed collision matrix，ACM）。

Trajectory Processing（轨迹处理）：Time parameterization（时间参数化），运动规划器一般只会生成路径，这个路径不带时间信息。MoveIt包含轨迹处理程序。它对结合路径和时间参数化的关节限制的速度和加速度来生成轨迹。这些限制是在joint\_limits.yaml中为每个机器人指定的。

### 2.配置助手（Setup assistant）

roslaunch moveit\_setup\_assistant setup\_assistant.launch

选择已创建URDF的.xacro文件进行配置。

配置ACM：自动生成。

虚拟关节virtual joint的配置：用于将机器人连接到世界上，如基座运动。

规划组群planning groups：语义上描述机器人的不同部分，如机械臂，末端执行器等。

其中，机械臂为add\_joints增加关节，末端执行器则为add\_links增加连接。

增加机器人的姿态：add pose，如初始home和前倾等等。调整关节角。

末端执行器的配置：允许末端执行器的特别操作。

被动关节：跳过规划。

生成配置文件：一般命名为[robot name]\_moveit\_config

Rviz插件：拖动，初始/目标状态，执行。

### 3.Move Group Python接口

通过Movegroup类实现

可以设置关节或目标姿态，创建行为规划，移动机器人，在环境中增加对象或给机器人增加或减少对象。

以下为move\_group\_python\_interface\_tutorial.py文件，scripts。

#!/usr/bin/env python

# Software License Agreement (BSD License)

#

# Copyright (c) 2013, SRI International

# All rights reserved.

#

# Redistribution and use in source and binary forms, with or without

# modification, are permitted provided that the following conditions

# are met:

#

# \* Redistributions of source code must retain the above copyright

# notice, this list of conditions and the following disclaimer.

# \* Redistributions in binary form must reproduce the above

# copyright notice, this list of conditions and the following

# disclaimer in the documentation and/or other materials provided

# with the distribution.

# \* Neither the name of SRI International nor the names of its

# contributors may be used to endorse or promote products derived

# from this software without specific prior written permission.

#

# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS

# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE

# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,

# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,

# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER

# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT

# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN

# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE

# POSSIBILITY OF SUCH DAMAGE.

#

# Author: Acorn Pooley

## BEGIN\_SUB\_TUTORIAL imports

##

## To use the python interface to move\_group, import the moveit\_commander

## module. We also import rospy and some messages that we will use.

import sys # 导入相关的Python包

import copy

import rospy

import moveit\_commander

import moveit\_msgs.msg

import geometry\_msgs.msg

## END\_SUB\_TUTORIAL

from std\_msgs.msg import String

def move\_group\_python\_interface\_tutorial():

## BEGIN\_TUTORIAL

##

## Setup

## ^^^^^

## CALL\_SUB\_TUTORIAL imports

##

## First initialize moveit\_commander and rospy.初始化moveit\_commander和rospy

print "============ Starting tutorial setup"

moveit\_commander.roscpp\_initialize(sys.argv)

rospy.init\_node('move\_group\_python\_interface\_tutorial',

anonymous=True)

## Instantiate a RobotCommander object. This object is an interface to

## the robot as a whole. 实例化RobotCommander的对象，本接口为机器人总入口

robot = moveit\_commander.RobotCommander()

## Instantiate a PlanningSceneInterface object. This object is an interface

## to the world surrounding the robot.

## 实例化PlanningSceneInterface的对象，本接口为围绕机器人的世界

scene = moveit\_commander.PlanningSceneInterface()

## Instantiate a MoveGroupCommander object. This object is an interface

## to one group of joints. In this case the group is the joints in the left

## arm. This interface can be used to plan and execute motions on the left

## arm.

## 实例化MoveGroupCommander的对象，本接口为所规划的关节组

group = moveit\_commander.MoveGroupCommander("left\_arm")

## We create this DisplayTrajectory publisher which is used below to publish

## trajectories for RVIZ to visualize.

## 创建DisplayTrajectory发布器，轨迹在Rviz中可视化

display\_trajectory\_publisher = rospy.Publisher(

'/move\_group/display\_planned\_path',

moveit\_msgs.msg.DisplayTrajectory)

## Wait for RVIZ to initialize. This sleep is ONLY to allow Rviz to come up.

## 等待Rviz初始化

print "============ Waiting for RVIZ..."

rospy.sleep(10)

print "============ Starting tutorial "

## Getting Basic Information

## ^^^^^^^^^^^^^^^^^^^^^^^^^

##

## We can get the name of the reference frame for this robot参考系名称

print "============ Reference frame: %s" % group.get\_planning\_frame()

## We can also print the name of the end-effector link for this group末端执行器名称

print "============ Reference frame: %s" % group.get\_end\_effector\_link()

## We can get a list of all the groups in the robot获得机器人所有组

print "============ Robot Groups:"

print robot.get\_group\_names()

## Sometimes for debugging it is useful to print the entire state of the

## robot. 机器人的所有状态

print "============ Printing robot state"

print robot.get\_current\_state()

print "============"

## Planning to a Pose goal

## ^^^^^^^^^^^^^^^^^^^^^^^

## We can plan a motion for this group to a desired pose for the

## end-effector 为该组规划动作，为目标姿态

print "============ Generating plan 1"

pose\_target = geometry\_msgs.msg.Pose()

pose\_target.orientation.w = 1.0

pose\_target.position.x = 0.7

pose\_target.position.y = -0.05

pose\_target.position.z = 1.1

group.set\_pose\_target(pose\_target)

## Now, we call the planner to compute the plan

## and visualize it if successful

## Note that we are just planning, not asking move\_group

## to actually move the robot规划，并在Rviz中显示

plan1 = group.plan()

print "============ Waiting while RVIZ displays plan1..."

rospy.sleep(5)

## You can ask RVIZ to visualize a plan (aka trajectory) for you. But the

## group.plan() method does this automatically so this is not that useful

## here (it just displays the same trajectory again).Rviz显示轨迹

print "============ Visualizing plan1"

display\_trajectory = moveit\_msgs.msg.DisplayTrajectory()

display\_trajectory.trajectory\_start = robot.get\_current\_state()

display\_trajectory.trajectory.append(plan1)

display\_trajectory\_publisher.publish(display\_trajectory);

print "============ Waiting while plan1 is visualized (again)..."

rospy.sleep(5)

## Moving to a pose goal

## ^^^^^^^^^^^^^^^^^^^^^

##运动到目标姿态

## Moving to a pose goal is similar to the step above

## except we now use the go() function. Note that

## the pose goal we had set earlier is still active

## and so the robot will try to move to that goal. We will

## not use that function in this tutorial since it is

## a blocking function and requires a controller to be active

## and report success on execution of a trajectory.

# Uncomment below line when working with a real robot

# group.go(wait=True)

## Planning to a joint-space goal

## ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

## 规划关节空间目标

## Let's set a joint space goal and move towards it.

## First, we will clear the pose target we had just set.

group.clear\_pose\_targets() #先清除姿态目标

## Then, we will get the current set of joint values for the group返回控制组当前的关节值

group\_variable\_values = group.get\_current\_joint\_values()

print "============ Joint values: ", group\_variable\_values

## Now, let's modify one of the joints, plan to the new joint

## space goal and visualize the plan 修改一个关节规划的目标，并可视化

group\_variable\_values[0] = 1.0

group.set\_joint\_value\_target(group\_variable\_values)

plan2 = group.plan()

print "============ Waiting while RVIZ displays plan2..."

rospy.sleep(5)

## Cartesian Paths

## ^^^^^^^^^^^^^^^ 笛卡尔空间轨迹规划

## You can plan a cartesian path directly by specifying a list of waypoints

## for the end-effector to go through.

waypoints = [] ## 路点

# start with the current pose 由当前位姿为起点

waypoints.append(group.get\_current\_pose().pose)

# first orient gripper and move forward (+x) 第一个点

wpose = geometry\_msgs.msg.Pose()

wpose.orientation.w = 1.0

wpose.position.x = waypoints[0].position.x + 0.1

wpose.position.y = waypoints[0].position.y

wpose.position.z = waypoints[0].position.z

waypoints.append(copy.deepcopy(wpose))

# second move down 第二个点

wpose.position.z -= 0.10

waypoints.append(copy.deepcopy(wpose))

# third move to the side 第三个点

wpose.position.y += 0.05

waypoints.append(copy.deepcopy(wpose))

## We want the cartesian path to be interpolated at a resolution of 1 cm

## which is why we will specify 0.01 as the eef\_step in cartesian

## translation. We will specify the jump threshold as 0.0, effectively

## disabling it. 插值分辨率为1cm，

(plan3, fraction) = group.compute\_cartesian\_path(

waypoints, # waypoints to follow

0.01, # eef\_step

0.0) # jump\_threshold

print "============ Waiting while RVIZ displays plan3..."

rospy.sleep(5)

## Adding/Removing Objects and Attaching/Detaching Objects

## ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

## First, we will define the collision object message定义碰撞物体的消息

collision\_object = moveit\_msgs.msg.CollisionObject()

## When finished shut down moveit\_commander. 完成后关闭

moveit\_commander.roscpp\_shutdown()

## END\_TUTORIAL

print "============ STOPPING"

if \_\_name\_\_=='\_\_main\_\_':

try:

move\_group\_python\_interface\_tutorial()

except rospy.ROSInterruptException:

pass

随后，编辑.launch文件

<launch>

<include file="$(find pr2\_moveit\_config)/launch/demo.launch"/>

<node name="move\_group\_python\_interface\_tutorial" pkg="moveit\_tutorials" type="move\_group\_python\_interface\_tutorial.py" respawn="false" output="screen">

</node>

</launch>

### 4.运动学模型kinematic model

通过C++ API应用运动模型

核心类为RobotModel和RobotState。

后续章节中的文件，都是C++语言，实际是.script文件，可以参考古月居的python文件讲解。对应关系如下：

* MoveGroup Python接口——moveit\_kt(it)\_demo.py。重要的类MoveGroupCommander。
* 规划场景（planning scene）——是moveit\_setup\_assistant中的设置内容，如碰撞检测和约束检测
* ROS API规划场景——API需要初始化，封装在python代码中，如page327。
* 运动规划（motion planners）——moveit\_it\_demo.py中的函数plan()。另一个示例为moveit\_cartesian\_demo.py笛卡尔路径规划API——compute\_cartesian\_path()，返回plan规划出的运动轨迹，和fraction轨迹在路点中的覆盖率。若fraction<1则未完整规划。
* 控制器管理——fake\_controller.yaml。虚拟控制器的参数，关节。
* 控制器配置——~/config/controllers.yaml。每个规划组所需配置的控制器插件。需要.launch中加载。
* 3D感知/配置——书上不在Moveit!中介绍。可用插件为点云数据和深度图数据。两个.yaml文件
* IKFast插件和Trac-Ik插件——需要自行安装的运动学求解插件，其中Trac-IK安装成功，IKFast安装失败。
* 运动学配置——kinematics.yaml，由setup\_assistant生成，包含运动学求解器的详细参数。
* OMPL接口——应有ompl\_planning.yaml，选择规划器及优化目标。
* 在rviz中增加场景物体——moveit\_obstacles\_demo.py，增加规则形状物体。

### 5.圆弧轨迹规划

ROS机械臂开发：从入门到实战

——第7讲内容 MoveIt!潜规则

<https://www.ncnynl.com/archives/201905/3104.html>

<https://blog.csdn.net/huangjunsheng123/article/details/114760366>

C++代码部分：

waypoints.push\_back(target\_pose);

double centerA = target\_pose.position.x; ##圆心和半径的定义

double centerB = target\_pose.position.z;

double radius = 0.13;

for(double th=0.0; th<6.28; th=th+0.01) ##圆形轨迹的路点定义

{

target\_pose.position.y = centerA + radius \* cos(th);

target\_pose.position.z = centerB + radius \* sin(th);

waypoints.push\_back(target\_pose);

}

// 笛卡尔空间下的路径规划

moveit\_msgs::RobotTrajectory trajectory;

const double jump\_threshold = 0.0;

const double eef\_step = 0.01;

double fraction = 0.0;

int maxtries = 100; //最大尝试规划次数

int attempts = 0; //已经尝试规划次数

#### Python编程尝试

启用roslaunch marm\_planning arm\_planning.launch

随后启用moveit\_cartesian\_circle.py

从计算库中调用sin和cos函数

from math import sin

from math import cos

# 设置circle路点数据，并加入路点列表

center\_x = wpose.position.x

center\_y = wpose.position.y

radius = 0.1;

for i in range(0,628,1):

wpose.position.x = center\_x+ radius\*cos(i/100);

wpose.position.y = center\_y+ radius\*sin(i/100);

if cartesian:

waypoints.append(deepcopy(wpose))

else:

arm.set\_pose\_target(wpose)

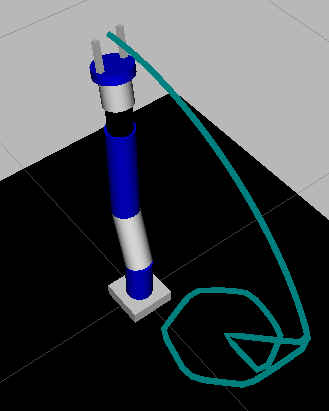
arm.go()

rospy.sleep(1)

Rviz中的效果图，不是很连贯。

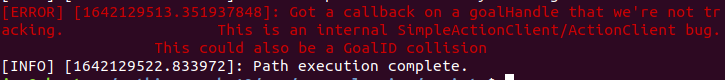
文件名：moveit\_cartesian\_circle.py





运行中存在报错：

Got a callback on a goalHandle that we're not tracking. This is an internal SimpleActionClient/ActionClient bug. This could also be a GoalID collision



#### 末端位姿的输出，建立监测器

以关节状态为例：

rostopic echo /joint\_states # 订阅关节状态的监控

rosbag record -O subset2 /joint\_states # 选定话题joint\_states，记录为subset2文件。注意，要在运动执行结束后，采用ctrl+c的方式停止，才能由.bag.active文件生成.bag文件。

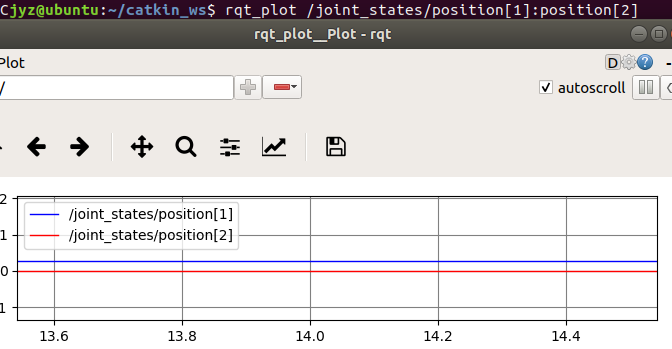
rostopic echo -b subset2.bag -p /joint\_states > joint\_trajectory.txt

# 将订阅joint\_states话题生成的.bag文件subset2.bag转换为.txt文件。

以上bag文件的操作均需在.bag文件的路径终端中进行。

或者，可以采用rqt\_plot进行执行过程中的可视化。

$ rqt\_plot /joint\_states/position[1]:position[2]:…:position[n]



rostopic echo /execute\_trajectory/goal

为执行时的路点waypoint，但并不是实时的反馈。在后续的.bag文件和.txt文件输出后，为2行的多列文件，说明信息相当于两次写入，无法有效的读取。

若要获得实时反馈，需要去查看follow\_joint\_trajectory/goal，根据关节角度编写脚本计算末端路点位姿。将路点写成文件，并输出。（未完成）

一个思路，可以订阅geometry\_msgs/Point.msg

<http://docs.ros.org/en/api/geometry_msgs/html/msg/Point.html>

一个未试用的代码

<http://www.bubuko.com/infodetail-2913626.html>

用python的节点订阅和发布

<https://www.cnblogs.com/sea-stream/p/10246046.html>

补充：如何查询机械臂末端的当前位姿

启动rviz，

$ rosrun moveit\_commander moveit\_commander\_cmdline.py

输入规划所用的运动群组group：（marm为’arm’）

$ use <group name>

$ current # 输出当前末端的pose

也可在rviz中直接查看，如：【MotionPlanning】->【Scene Robot】->【Links】

参考：

<https://blog.csdn.net/zzu_seu/article/details/90731201?spm=1001.2101.3001.6650.5&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5.pc_relevant_default&utm_relevant_index=9>

## D.UR5使用

### 1.软件包安装

有官方软件包可以选装

### 2.Gazebo和Rviz对UR5进行控制

Gazebo中显示UR5

$ roslaunch ur\_gazebo ur5.launch

新开终端，运行路径规划节点moveit!

$ roslaunch ur5\_moveit\_config ur5\_moveit\_planning\_execution.launch sim:=true

新开终端，Rviz中显示UR5

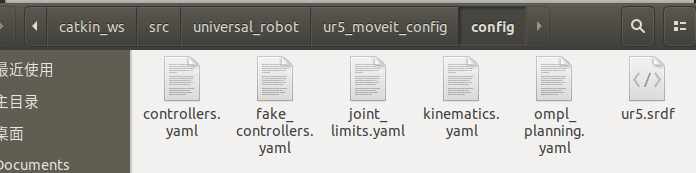
$ roslaunch ur5\_moveit\_config moveit\_rviz.launch config:=true

### 3.进行轨迹规划

在rviz中显示UR5，并进行拖动规划

roslaunch ur5\_moveit\_config demo.launch

一系列的.yaml文件路径如下：



### 4. C++的编程规划示例

<https://blog.csdn.net/zzu_seu/article/details/90673179>