# **用ros\_control控制UR5机械臂**

在上一篇文章中，我们在Gazebo中仿真了UR5机械臂，但机械臂软塌塌地趴在地面上。这是因为机械臂受到重力作用，其各个关节又没有摩擦和阻尼。

要想它不趴在地上，就需要为UR5机械臂的每个关节添加一个控制器（controller）来控制它，本篇文章我们将讨论这个问题。

## **一、实验内容**

本实验中，我们用Gazebo可视化并仿真添加了关节控制器的UR5机械臂。

然后，我们通过研究下载的.urdf文件，来详细了解为UR5机械臂添加[关节控制器](https://zhida.zhihu.com/search?content_id=236374960&content_type=Article&match_order=2&q=%E5%85%B3%E8%8A%82%E6%8E%A7%E5%88%B6%E5%99%A8&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)的步骤。

## **二 、实验环境**

* Ubuntu 18.04.5 LTS (bionic)
* ROS [melodic](https://zhida.zhihu.com/search?content_id=236374960&content_type=Article&match_order=1&q=melodic&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)

## **三、实验步骤**

**1、检查Gazebo是否安装？**

如果你顺利完成了上一篇文章中的实验，那你应该已经安装好了Gazebo 。否则，请运行如下命令以重新安装Gazebo：

$ sudo apt-get install ros-${ROS\_DISTRO}-gazebo-ros

**2、安装universal\_robots功能包**

$ sudo apt-get install ros-$ROS\_DISTRO-universal-robots

**3、安装ros\_control功能包**

$ sudo apt-get install ros-$ROS\_DISTRO-ros-control

$ sudo apt-get install ros-$ROS\_DISTRO-ros-controllers

**4、创建~/ur5\_tutorial工作区**

$ mkdir -p ~/ur5\_tutorial/src

$ cd ~/ur5\_tutorial/src

$ catkin\_init\_workspace

$ cd ~/ur5\_tutorail

$ catkin\_make

$ echo "source ~/ur5\_tutorial/devel/setup.bash" >> ~/.bashrc

$ source ~/ur5\_tutorial/devel/setup.bash

**5、下载ur5-joint-position-control功能包，放在~/ur5\_tutorial/src目录下**

$ cd ~/ur5\_tutorial/src

$ git clone https://github.com/dairal/ur5-joint-position-control.git

$ cd ~/ur5\_tutorial

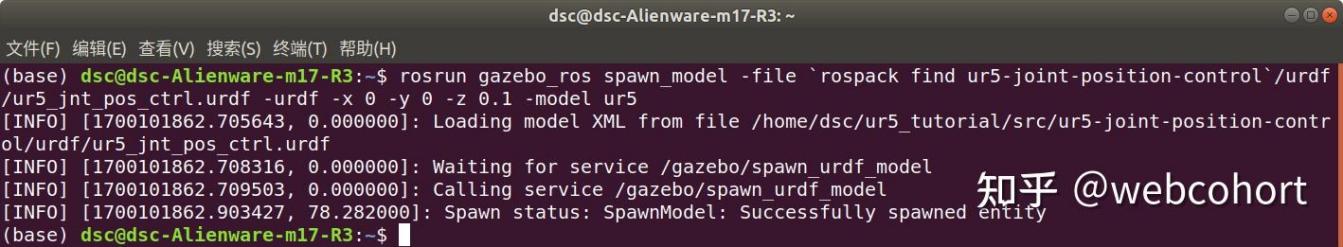
$ catkin\_make

**6a、运行Gazebo，加载一个空的环境（环境什么也没有）**

$ roslaunch gazebo\_ros empty\_world.launch

**7a、将UR5机械臂模型ur5\_jnt\_pos\_ctrl.urdf加载到Gazebo中**

$ rosrun gazebo\_ros spawn\_model -file `rospack find ur5-joint-position-control`/urdf/ur5\_jnt\_pos\_ctrl.urdf -urdf -x 0 -y 0 -z 0.1 -model ur5



**8a、运行UR5机械臂的关节控制器节点**

$ roslaunch ur5-joint-position-control ur5\_joint\_position\_control.launch

哎呀！UR5机械臂咋还趴地上呢？不是说添加了关节控制器，它就不会趴在地上了吗？

原因：虽然我们已经在ur5\_jnt\_pos\_ctrl.urdf文件中给机械臂的每个关节添加了相应的控制器，但每个关节都是处于初始状态，所以它还是趴在地上。

退出Gazebo，重新执行下面的6b、7b、8b三步，就可以看到机械臂不再趴在地上了：

**6b、重新运行Gazebo，加载一个空的环境**

$ roslaunch gazebo\_ros empty\_world.launch

**7b、重新将UR5机械臂模型ur5\_jnt\_pos\_ctrl.urdf加载到Gazebo中，并给机械臂指定一个初始姿态**

$ rosrun gazebo\_ros spawn\_model -file `rospack find ur5-joint-position-control`/urdf/ur5\_jnt\_pos\_ctrl.urdf -urdf -x 0 -y 0 -z 0.1 -model ur5 -J shoulder\_lift\_joint -1.5 -J elbow\_joint 1.0

-J参数用于指定某个关节的初始角度，-J shoulder\_lift\_joint -1.5指定shoulder\_lift\_joint关节的初始角度为-1.5弧度，-J elbow\_joint 1.0指定elbow\_joint关节的初始角度为1.0弧度。

**8b、重新运行UR5机械臂的关节控制器节点**

$ roslaunch ur5-joint-position-control ur5\_joint\_position\_control.launch

由上图可见，UR5机械臂站起来了，不再趴在地上了，说明机械臂的关节控制器起作用了。

**9、查看UR5机械臂的各个关节的控制器**

$ rosrun controller\_manager controller\_manager list

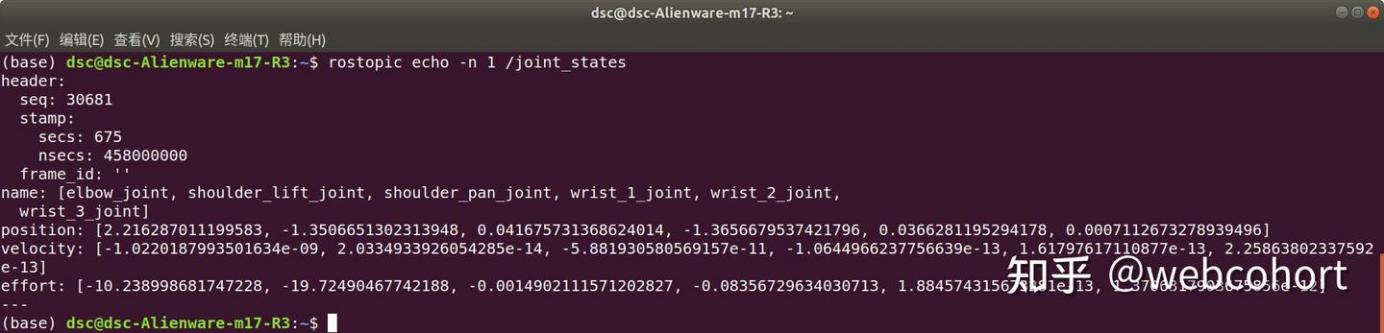
UR5机械臂的每个关节都对应了一个控制器，如下表所示：

|  |  |  |
| --- | --- | --- |
| 关节joint | 控制器controller | 硬件接口hardware\_interface |
| shoulder\_pan\_joint | shoulder\_pan\_joint\_position\_controller | EffortJointInterface |
| shoulder\_lift\_joint | shoulder\_lift\_joint\_position\_controller | EffortJointInterface |
| elbow\_joint | elbow\_joint\_position\_controller | EffortJointInterface |
| wrist\_1\_joint | wrist\_1\_joint\_position\_controller | EffortJointInterface |
| wrist\_2\_joint | wrist\_2\_joint\_position\_controller | EffortJointInterface |
| wrist\_3\_joint | wrist\_3\_joint\_position\_controller | EffortJointInterface |
|  | joint\_state\_controller | JointStateInterface |

joint\_state\_controller控制器比较特殊，不是用来给某个关节发指令的，而是用来读取所有关节的当前状态，并且发布在 "/joint\_states"话题上。

**10、通过joint\_state\_controller控制器发布的/joint\_states话题，读取机械臂所有关节的当前状态**

$ rostopic echo -n 1 /joint\_states



从上图可见，/joint\_states话题返回的是机械臂所有关节的状态，包括每个关节的位置（position）、速度（velocity）、关节驱动器接收到的力（effort）指令，如：wrist\_1\_joint关节的位置=-1.3656679537421796，速度=-1.0644966237756639e-13，关节驱动器接收到的力（effort）指令=-0.08356729634030713。

**11、通过某个关节的控制器发布的state话题，读取此关节的当前状态**

joint\_state\_controller控制器发布的是机械臂所有关节的当前状态，而每个关节对应的控制器通过 ***/关节控制器/state***话题来发布此关节的当前状态，如/wrist\_1\_joint关节发布的状态话题为/wrist\_1\_joint\_position\_controller/state。我们可以通过订阅此话题，来了解/wrist\_1\_joint关节的当前状态。

$ rostopic echo -n 1 /wrist\_1\_joint\_position\_controller/state



/wrist\_1\_joint\_position\_controller/state话题返回的信息中，最重要的是：

* process\_value: -1.36565933365，关节的当前位置（角度）为 -1.37弧度。
* command: -0.0835658906879，关节驱动器接收到的力（effort）指令为-0.08N。

这些信息与我们通过joint\_state\_controller控制器发布的/joint\_states话题所读取的信息，是一致的。

**12、通过某个关节的控制器订阅的command话题，控制关节旋转到某个角度**

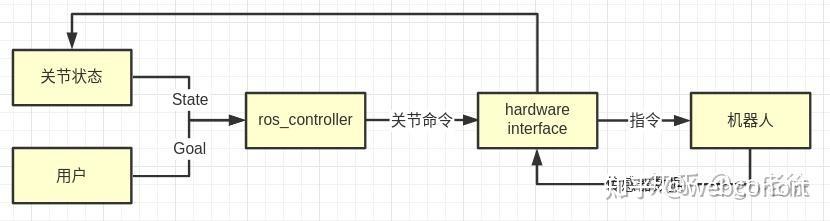
每个关节对应的控制器通过***/关节控制器/command***话题来订阅此关节要接收的指令，如/wrist\_1\_joint关节订阅的指令话题为/wrist\_1\_joint\_position\_controller/command。我们可以通过向此话题发布指令，来控制/wrist\_1\_joint关节旋转到一个新的角度。

$ rostopic pub /wrist\_1\_joint\_position\_controller/command std\_msgs/Float64 3.14

## **四、进一步说明**

### **1、ROS\_Control**

ROS\_Control是一个软件包，包括controller interface, controller manager, [transmissions](https://zhida.zhihu.com/search?content_id=236374960&content_type=Article&match_order=1&q=transmissions&zhida_source=entity" \t "/home/smart/Documents\\x/_blank), hardware interfaces and control toolbox，可以完成对机器人关节的状态读取和控制。



ros\_controller负责根据关节目标和关节状态计算出关节命令。

hardware\_interfaces负责与真正的机器人硬件打交道，让ros\_control不必拘泥于具体的硬件。1. 它将ros\_control 传过来的指令转换为机器人能接受的指令;2.它读取机器人的当前状态，并转化成ros\_control能听懂的格式发送给ros\_control。

### **2、控制器controller**

**（1）joint\_state\_controller**

用来读取机器人所有关节的状态，并且发布在 "/joint\_states"话题上。

**（2）effort\_controllers：驱动器接受力矩指令**

* JointPositionController : "effort\_controllers/JointPositionController"。 controller接受位置指令，Error=（期望位置-当前位置）。根据误差输出力指令（PID闭环)。
* JointVelocityController : "effort\_controllers/JointVelocityController"。controller接受速度指令，Error=（期望速度-当前速度）。根据误差输出力指令（PID闭环）。
* JointEffortController: "effort\_controllers/JointEffortController"。controller接受力矩指令，输出力指令（PID闭环在这里无效）。

**（3）velocity\_controllers: 驱动器接受速度指令**

* JointPositionController : "velocity\_controllers/JointPositionController"。controller接受位置指令，Error=（期望位置-当前位置）。根据误差输出速度指令（PID闭环）。
* JointVelocityController : "velocity\_controllers/JointVelocityController"。controller接受速度指令，输出速度指令（PID闭环在这里无效）。

**（4）position\_controllers: 驱动器接受位置指令**

* JointPositionController : "position\_controllers/JointPositionController"。controller接受位置指令，输出位置指令（PID闭环在这里无效）。

ros\_control包提供了很多控制器。你可以根据自己的驱动器接受指令的类型来选择，也可以根据需求重写自己的控制器。  
controller发送的指令会经过hardware\_interface，而不是直接传到机器人硬件上。下面，我们来了解一下有哪些hardware\_interface。

### **3、硬件接口hardware\_interface**

hardware\_interface完成了机器人的硬件抽象，是ros\_control和实体机器人沟通的桥梁。hardware\_interface有两项功能，一是向实体机器人发送指令，二是从机器人的传感器读取机器人状态数据。

**（1）Interfaces for Joint Actuators**

* EffortJointInterface：用于接受力矩指令的驱动器，对应使用effort\_controllers。
* VelocityJointInterface：用于接受速度指令的驱动器，对应使用velocity\_controllers
* PositionJointInterface：用于接受位置指令的驱动器，对应使用position\_controllers。

**（2）Interfaces for Joint Sensors**

* JointStateInterface：当关节有位置/速度/力传感器时使用，对应的controller为 joint\_state\_controller。
* ImuSensorInterface：当有IMU传感器时使用，对应的controller为imu\_sensor\_controller。

根据驱动器和传感器的类型，可以选择合适的hardware\_interface。

### **4、如何用ros\_control控制UR5机械臂？**

在上一篇文章中，我们谈到了“**从RViz可视化到Gazebo仿真，需要给机器人模型再添加什么物理属性？”**：

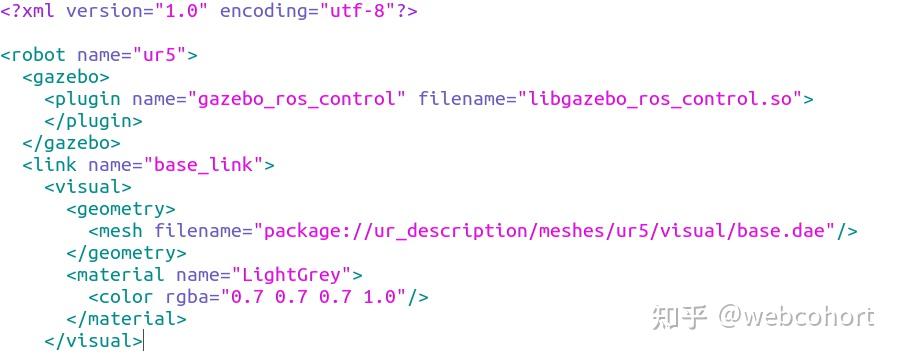
* 为每一根连杆添加惯性参数<inertial>
* 为每一根连杆添加碰撞参数<collision>
* 为每一个关节添加[关节约束](https://zhida.zhihu.com/search?content_id=236374960&content_type=Article&match_order=1&q=%E5%85%B3%E8%8A%82%E7%BA%A6%E6%9D%9F&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)<limit>
* 为每一个关节添加动力学参数<dynamics>
* 将机器人固定到地板上

为了用ros\_control控制UR5机械臂，我们必须对上一篇文章中的UR5机械臂模型文件ur5\_gazebo.urdf进一步做如下修改（修改后的模型文件为ur5\_jnt\_pos\_ctrl.urdf，位于~/ur5\_tutorial/src/ur5-joint-position-control/urdf目录下）：

**（1）添加ros\_control插件**

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

首先要在模型文件中，用如上的方式添加ros\_control插件，插件的文件名为libgazebo\_ros\_control.so。添加了插件后的模型文件ur5\_jnt\_pos\_ctrl.urdf部分内容如下：



**（2）为每个关节添加传动（transmission），指定关节的硬件接口（hardware\_interface）**

传动（transmission）要位于所有<link> 和 <joint> 标签的后面，用来指定每个关节具体要用到的硬件接口（hardware\_interface），如下所示：

<transmission name="shoulder\_lift\_trans">

<type>[transmission\_interface](https://zhida.zhihu.com/search?content_id=236374960&content_type=Article&match_order=1&q=transmission_interface&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)/SimpleTransmission</type>

<joint name="shoulder\_lift\_joint">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="shoulder\_lift\_motor">

<mechanicalReduction>1</mechanicalReduction>

</actuator></transmission>

* <type>: 传动类型，transmission\_interface/SimpleTransmission代表的是简单的齿轮减速器传动。
* <joint name="">: 对应URDF中定义的关节名称。
* <hardwareInterface>: 在<actuator>和<joint>标签中，告诉gazebo\_ros\_control插件要加载的硬件接口（位置,速度或力矩接口）。
* <mechanicalReduction>: 减速比。

对UR5机械臂，我们在*ur5\_jnt\_pos\_ctrl.urdf文件中*添加了如下传动：

<transmission name="shoulder\_pan\_trans">

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

<joint name="shoulder\_pan\_joint">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="shoulder\_pan\_motor">

<mechanicalReduction>1</mechanicalReduction>

</actuator>

</transmission>

<transmission name="shoulder\_lift\_trans">

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

<joint name="shoulder\_lift\_joint">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="shoulder\_lift\_motor">

<mechanicalReduction>1</mechanicalReduction>

</actuator>

</transmission>

<transmission name="elbow\_trans">

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

<joint name="elbow\_joint">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="elbow\_motor">

<mechanicalReduction>1</mechanicalReduction>

</actuator>

</transmission>

<transmission name="wrist\_1\_trans">

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

<joint name="wrist\_1\_joint">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="wrist\_1\_motor">

<mechanicalReduction>1</mechanicalReduction>

</actuator>

</transmission>

<transmission name="wrist\_2\_trans">

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

<joint name="wrist\_2\_joint">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="wrist\_2\_motor">

<mechanicalReduction>1</mechanicalReduction>

</actuator>

</transmission>

<transmission name="wrist\_3\_trans">

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

<joint name="wrist\_3\_joint">

<hardwareInterface>EffortJointInterface</hardwareInterface>

</joint>

<actuator name="wrist\_3\_motor">

<mechanicalReduction>1</mechanicalReduction>

</actuator>

</transmission>

由上可知，我们为每一个关节，添加的hardware\_interface都是EffortJointInterface，也即关节的驱动器接受力矩指令，对应使用effort\_controllers。

**（3）为每个关节添加控制器（controller）**

除了在URDF文件中为每个关节添加传动以外，我们还需要创建一个单独的[配置文件](https://zhida.zhihu.com/search?content_id=236374960&content_type=Article&match_order=1&q=%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)，在配置文件中，我们添加每个关节要用到的控制器。这个配置文件，要加载到参数服务器（parameter server）中。

我们为UR5机械臂创建的配置文件为ur5\_jnt\_pos\_ctrl.yaml，位于~/ur5\_tutorial/src/ur5-joint-position-control/config目录下，下面是文件内容：

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

joint\_state\_controller:

type: joint\_state\_controller/JointStateController

publish\_rate: 50

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

shoulder\_pan\_joint\_position\_controller:

type: effort\_controllers/JointPositionController

joint: shoulder\_pan\_joint

pid: {p: 500.0, i: 0.01, d: 50.0, i\_clamp\_min: -100.0, i\_clamp\_max: 100.0}

shoulder\_lift\_joint\_position\_controller:

type: effort\_controllers/JointPositionController

joint: shoulder\_lift\_joint

pid: {p: 500.0, i: 100.0, d: 30.0, i\_clamp\_min: -400.0, i\_clamp\_max: 400.0}

elbow\_joint\_position\_controller:

type: effort\_controllers/JointPositionController

joint: elbow\_joint

pid: {p: 10000.0, i: 0.01, d: 50.0, i\_clamp\_min: -100.0, i\_clamp\_max: 100.0}

wrist\_1\_joint\_position\_controller:

type: effort\_controllers/JointPositionController

joint: wrist\_1\_joint

pid: {p: 200.0, i: 10.0, d: 20.0, i\_clamp\_min: -400.0, i\_clamp\_max: 400.0}

wrist\_2\_joint\_position\_controller:

type: effort\_controllers/JointPositionController

joint: wrist\_2\_joint

pid: {p: 100.0, i: 0.1, d: 10.0, i\_clamp\_min: -100.0, i\_clamp\_max: 100.0}

wrist\_3\_joint\_position\_controller:

type: effort\_controllers/JointPositionController

joint: wrist\_3\_joint

pid: {p: 100.0, i: 0.1, d: 10.0, i\_clamp\_min: -100.0, i\_clamp\_max: 100.0}

由上可知，我们为UR5的每个关节都添加了一个effort\_controllers/JointPositionController控制器，controller接受位置指令，error=（期望位置-当前位置），根据误差输出速度指令（PID闭环）。配置文件中还指定了PID控制的参数。

配置文件的最顶部还添加了一个joint\_state\_controller控制器，与我们在实验步骤中看到的结果一致。

**（4）创建一个launch文件来加载关节的controller**

我们为UR5机械臂创建的配置文件为ur5\_joint\_position\_control.launch，位于~/ur5\_tutorial/src/ur5-joint-position-control/launch目录下，下面是文件内容：

<launch>

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

<rosparam file="$(find ur5-joint-position-control)/config/ur5\_jnt\_pos\_ctrl.yaml" command="load"/>

<param name="robot\_description" textfile="$(find ur5-joint-position-control)/urdf/ur5\_jnt\_pos\_ctrl.urdf"/>

*<!-- load the controllers -->*

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

output="screen" args="shoulder\_pan\_joint\_position\_controller shoulder\_lift\_joint\_position\_controller elbow\_joint\_position\_controller wrist\_1\_joint\_position\_controller wrist\_2\_joint\_position\_controller wrist\_3\_joint\_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="/ur5/joint\_states" />

</node>

</launch>

此launch文件，主要实现了如下功能：

* 把配置文件ur5\_jnt\_pos\_ctrl.yaml，加载到参数服务器：

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

<rosparam file="$(find ur5-joint-position-control)/config/ur5\_jnt\_pos\_ctrl.yaml" command="load"/>

* 把机械臂模型文件ur5\_jnt\_pos\_ctrl.urdf，加载到参数服务器：

<param name="robot\_description" textfile="$(find ur5-joint-position-control)/urdf/ur5\_jnt\_pos\_ctrl.urdf"/>

* 加载机械臂膀关节控制器和joint\_state\_controller控制器：

*<!-- load the controllers -->*

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

output="screen" args="shoulder\_pan\_joint\_position\_controller shoulder\_lift\_joint\_position\_controller elbow\_joint\_position\_controller wrist\_1\_joint\_position\_controller wrist\_2\_joint\_position\_controller wrist\_3\_joint\_position\_controller joint\_state\_controller"/>

* 加载robot\_state\_publisher节点，订阅/ur5/joint\_states话题，发布/tf话题：

*<!-- 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="/ur5/joint\_states" />

</node>

### **5、如何通过ros\_control编程控制UR5机械臂？**

**（1）正运动学控制**

编程订阅每个关节的/(关节控制器名字)/command话题，向其发送消息，就可以控制每个关节的位置。

**（2）[逆运动学](https://zhida.zhihu.com/search?content_id=236374960&content_type=Article&match_order=1&q=%E9%80%86%E8%BF%90%E5%8A%A8%E5%AD%A6&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)控制**

如果要控制UR5机械臂的末端tool center point (tcp) 的位姿，则需要在程序中调用第三方的运动学库（如KDL）来进行逆运动学求解，可以参见如下文章：

[Tutorial: Control the TCP position of a UR5 robot in C++ with KDL – Inverse Kinematics explained​roboticscasual.com/tutorial-controlling-tcp-position-of-the-ur5-robot-with-kdl-in-c-inverse-kinematics/](https://link.zhihu.com/?target=https://roboticscasual.com/tutorial-controlling-tcp-position-of-the-ur5-robot-with-kdl-in-c-inverse-kinematics/" \t "/home/smart/Documents\\x/_blank)

这么做比较麻烦，更好的做法是通过MoveIt!来控制机械臂，下一篇文章将讲述这个问题。

## **五、参考文献**

【1】ROS Tutorial: Control the UR5 robot with ros\_control – How to tune a PID Controller

[https://roboticscasual.com/ros-tutorial-control-the-ur5-robot-with-ros\_control-tuning-a-pid-controller/​roboticscasual.com/ros-tutorial-control-the-ur5-robot-with-ros\_control-tuning-a-pid-controller/](https://link.zhihu.com/?target=https://roboticscasual.com/ros-tutorial-control-the-ur5-robot-with-ros_control-tuning-a-pid-controller/" \t "/home/smart/Documents\\x/_blank)

【2】ROScon 2014 Talk slides of ***ros-control: An overview***

[https://roscon.ros.org/2014/wp-content/uploads/2014/07/ros\_control\_an\_overview.pdf​roscon.ros.org/2014/wp-content/uploads/2014/07/ros\_control\_an\_overview.pdf](https://link.zhihu.com/?target=https://roscon.ros.org/2014/wp-content/uploads/2014/07/ros_control_an_overview.pdf" \t "/home/smart/Documents\\x/_blank)

【3】***ROS Control***

[ROS Control | ROS Robotics​www.rosroboticslearning.com/ros-control](https://link.zhihu.com/?target=https://www.rosroboticslearning.com/ros-control" \t "/home/smart/Documents\\x/_blank)

发布于 2023-11-16 16:10・IP 属地北京

# **在Gazebo中仿真UR5机械臂**

Gazebo是一款3D[仿真软件](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=%E4%BB%BF%E7%9C%9F%E8%BD%AF%E4%BB%B6&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)，它不但能像RViz那样可视化机器人，而且集成了物理引擎，允许用户在仿真的环境下测试自己的机器人。

## **一、实验内容**

本实验中，我们从网络上下载UR5机械臂的.urdf文件，用Gazebo可视化并仿真UR5机械臂。

然后，我们通过研究下载的.urdf文件，来了解这种URDF文件格式的细节。

## **二 、实验环境**

* Ubuntu 18.04.5 LTS (bionic)
* ROS [melodic](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=melodic&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)

## **三、实验步骤**

### **1、检查Gazebo是否安装？**

如果你已经安装好了ROS，一般来说，Gazebo 也应该随之安装好了。可以用如下的命令检查Gazebo能否正常运行：

$ gazebo

如果出现下图，说明Gazebo已经正确安装并能顺利运行。否则，请运行如下命令以重新安装Gazebo：

$ sudo apt-get install ros-${ROS\_DISTRO}-gazebo-ros

### **2、单独下载UR5机械臂3D模型的.urdf文件（Gazebo版）**

从如下网址下载ur5\_gazebo.txt文件，将它放在~/ur5\_description/文件夹下，并将它改名为ur5\_gazebo.urdf，这就是UR5机械臂3D模型的.urdf文件（Gazebo版）。

[UR5机械臂3D模型的URDF文件（Gazebo版）​roboticscasual.com/files/ur5\_gazebo.txt](https://link.zhihu.com/?target=http://roboticscasual.com/files/ur5_gazebo.txt" \t "/home/smart/Documents\\x/_blank)

[https://link.zhihu.com/?target=http://roboticscasual.com/files/ur5\_gazebo.txt](https://link.zhihu.com/?target=http:/roboticscasual.com/files/ur5_gazebo.txt)

### **3、在Gazebo中加载并显示UR5的.urdf文件（Gazebo版）**

（1）运行Gazebo，加载一个空的世界文件

$ roslaunch gazebo\_ros empty\_world.launch

Gazebo如下图所示正常运行，但因为加载的是一个空世界（empty\_world）文件。Gazebo的主窗口中空空如也，既见不到机器人，也见不到周边有任何环境。

（2）加载并显示UR5的.urdf文件（Gazebo版）

$ rosrun gazebo\_ros spawn\_model -file ~/ur5\_description/ur5\_gazebo.urdf -urdf -x 0 -y 0 -z 0.1 -model ur5

一切顺利的话，你会看到如下的界面，说明Gazebo正确加载并显示了UR5机械臂的3D模型。

在实验步骤2中，我们单独下载了UR5机械臂3D模型的.urdf文件（Gazebo版）；在实验步骤3中，我们在命令行中运行Gazebo，并加载并显示UR5的.urdf文件（Gazebo版）。

步骤2、步骤3这两步也可以被如下的步骤4和步骤5代替，实验结果相同：

### **4、下载universal\_robots包**

sudo apt-get install ros-$ROS\_DISTRO-universal-robots

步骤2下载的是单独的UR5机械臂3D模型的.urdf文件（Gazebo版），而上述指令是下载整个universal\_robots包，其中包括了ur\_description包（存放着UR机械臂的模型文件，前两篇关于RViz的文章中我们已经见到过）、ur\_gazebo包（在Gazebo中仿真机械臂要用到的包）。

### **5、用launch文件运行Gazebo，并加载并显示UR5的.urdf文件（Gazebo版）**

$ roslaunch ur\_gazebo ur5\_bringup.launch

在步骤3中，Gazebo的运行和UR5模型的加载是分成两步进行的。而在步骤5中，我们把这两步集成到了一个启动脚本（launch）文件ur5\_bringup.lauch中。

## **四、进一步说明**

### **1、Gazebo**

Gazebo是一款3D[仿真器](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=%E4%BB%BF%E7%9C%9F%E5%99%A8&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)，支持机器人开发所需的机器人、传感器和环境模型，并且通过搭载的物理引擎可以得到逼真的仿真结果。

Gazebo是近年来最受欢迎的三维仿真器之一，被选为美国DARPA[机器人挑战赛](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=%E6%9C%BA%E5%99%A8%E4%BA%BA%E6%8C%91%E6%88%98%E8%B5%9B&zhida_source=entity" \t "/home/smart/Documents\\x/_blank) 7 的官方仿真器。它是开源仿真器，却具有高水准的仿真性能，因此在机器人工程领域中非常流行。不仅如此，负责开发和普及ROS的Open Robotics负责开发ROS和Gazebo，因此ROS和Gazebo非常兼容。

**Gazebo的特征**

* 动力学仿真：支持ODE（开放式动力引擎）、Bullet、Simbody和DART等各种物理引擎。
* 3D图形：Gazebo采用经常在游戏中使用的OGRE（开源图形渲染引擎），不仅可以实现机器人模型，还可以逼真地表达光、阴影和材质。
* 支持传感器和噪声：支持虚拟的[激光测距仪](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=%E6%BF%80%E5%85%89%E6%B5%8B%E8%B7%9D%E4%BB%AA&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)（LRF）、2/3D相机、深度相机、触摸传感器、力矩传感器，并且在检测到的数据中包含与真实世界相似的噪声。
* 可添加插件：提供API，以便用户可以以插件的形式亲手创建机器人、传感器和环境控制等。
* 机器人模型：PR2、Pioneer2 DX、iRobot Create和TurtleBot已经以SDF格式存在于Gazebo中。SDF格式是一个Gazebo[模型文件](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=2&q=%E6%A8%A1%E5%9E%8B%E6%96%87%E4%BB%B6&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)格式。此外，用户还可以添加自己创建的SDF格式的机器人。
* TCP/IP数据传输：仿真也可以在远程服务器上执行，这是使用Google的protobufs（基于socket的消息传递）实现的。
* 云仿真：提供CloudSim云仿真环境，因此可以在Amazon、Softlayer和OpenStack等云环境中使用Gazebo。
* [命令行工具](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=%E5%91%BD%E4%BB%A4%E8%A1%8C%E5%B7%A5%E5%85%B7&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)：不仅可以使用GUI界面，还可以使用CUI风格的命令行工具来查看和控制仿真过程。

[Gazebo​gazebosim.org/home](https://link.zhihu.com/?target=https://gazebosim.org/home" \t "/home/smart/Documents\\x/_blank)

[Installing Gazebo with ROS​gazebosim.org/docs/garden/ros\_installation](https://link.zhihu.com/?target=https://gazebosim.org/docs/garden/ros_installation" \t "/home/smart/Documents\\x/_blank)

### **2、从RViz可视化到Gazebo仿真，需要给机器人模型再添加什么物理属性？**

在ur5\_rviz.urdf 中，对于每根连杆，我们只定义了其<visual>标签，因为可视化只是把机器人的模型显示在屏幕上，故而在URDF文件中，只需要定义好机器人的外观（visual）即可。下图为base\_link 连杆在ur5\_rviz.urdf 中的定义：

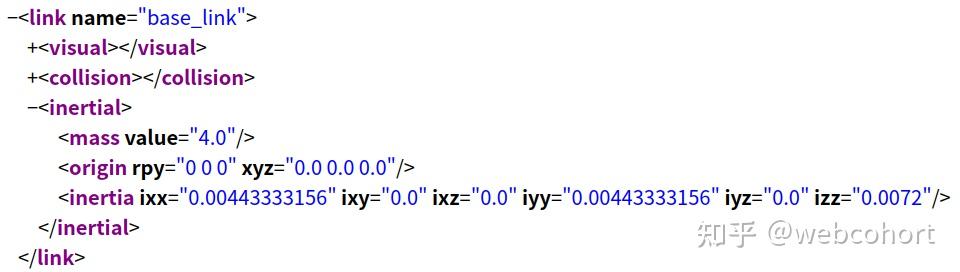


而在Gazebo中仿真机器人模型时，除了机器人的外观（visual）属性之外，Gazebo还需要知道机器人如何与仿真环境交互，机器人在仿真环境中如何运动。因此，只知道机器人的外观（visual）属性是不够的，我们还必须在URDF文件中为机器人模型添加一些新的物理属性。这些新添加的物理属性，也是用于Gazebo仿真的URDF模型与只用于RViz可视化的URDF模型所不同的地方。

**（1）为每一根连杆添加惯性参数<inertial>**

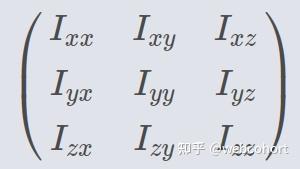
要进行精确仿真，Gazebo必须知道机器人部件（如连杆）的物理惯性参数：质量、质心位置以及惯性矩矩阵（惯性矩矩阵表明了机器人部件的质量是如何分布的）。

惯性参数标签为<inertial>，位于<link>……</link>之间，在<visual>和<collision>标签之后。下图为ur5\_gazebo.urdf 文件中base\_link连杆的<inertial>标签：



* **质量（Mass）**：由上图可知，连杆base\_link的质量=4.0kg，子标签为<mass>。
* **质心（Center of Mass）**：均质物体的质心等于其几何质心，连杆base\_link的[质心坐标](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=%E8%B4%A8%E5%BF%83%E5%9D%90%E6%A0%87&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)（x, y, z）=（0.0，0.0，0.0），子标签为<origin>。
* **惯性矩矩阵（Moment of Inertia Matrix）**：惯性矩代表一个刚体中质量的空间分布。惯性矩取决于物体的质量、尺寸和形状，其单位为[质量\*长度^2]，子标签为<inertia>。

机器人连杆的惯性矩矩阵可以表示为一个对称正定的3×3矩阵（如下图所示），有3个对角元素和3个唯一的非对角元素。每个[惯性矩阵](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=%E6%83%AF%E6%80%A7%E7%9F%A9%E9%98%B5&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)都是相对于一个坐标系或一组轴来定义的。



惯性矩矩阵是关于主对角线对称的，故有 *Ixy=Iyx*，*Ixz=Izx*，*Iyz=Izy*。因而，在<inertia>子标签中，实际上只有6个值。

惯性矩矩阵的具体数值如何获得？这个问题比较复杂，可以参考下面的文章：

[Inertial parameters of triangle meshes​classic.gazebosim.org/tutorials?tut=inertia](https://link.zhihu.com/?target=https://classic.gazebosim.org/tutorials?tut=inertia" \t "/home/smart/Documents\\x/_blank)

**（2）为每一根连杆添加碰撞参数<collision>**

Gazebo还必须知道连杆的碰撞参数，以便在仿真时进行碰撞检测。碰撞参数标签为<collision>，位于<link>……</link>之间，在<visual>标签之后，<inertial>标签之前，如下图所示：

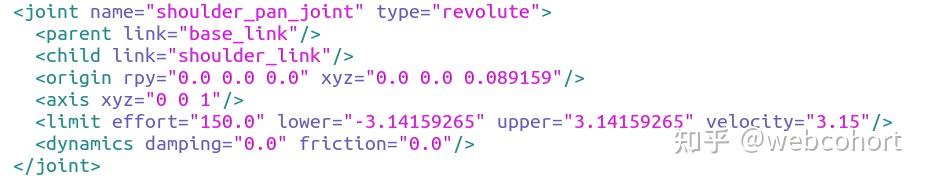


为了简单起见，[base\_link](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=5&q=base_link&zhida_source=entity" \t "/home/smart/Documents\\x/_blank) 连杆的碰撞参数的<geometry>子标签，直接使用了UR5机械臂CAD模型的STL文件，没有定义额外的颜色和材质。

（1）、（2）两步，都是**给连杆添加额外的物理属性。**完成这两步以后，在ur5\_gazebo.urdf 中，base\_link 的完整定义如下图：

**（3）为每一个关节添加关节约束<limit>**

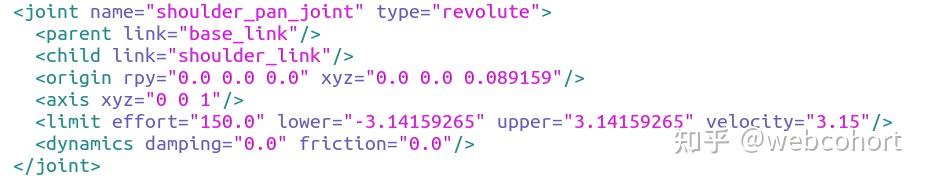
关节的约束标签为<limit>，位于<joint>……</joint>之间。下图为ur5\_gazebo.urdf 文件中shoulder\_pan\_joint关节的<limit>标签：



* effort：所能施加的最大的力/力矩
* lower, upper：关节所能旋转的最小角、最大角，单位为弧度
* velocity：关节的最大速度

**（4）为每一个关节添加动力学参数<dynamics>**

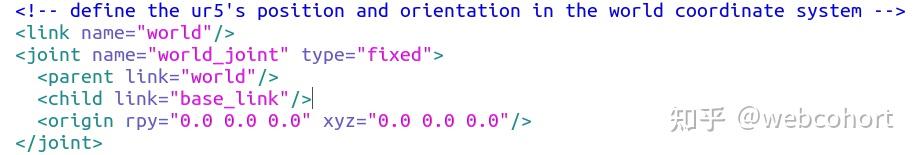
关节的动力学标签为<dynamics>，位于<joint>……</joint>之间。下图为ur5\_gazebo.urdf 文件中shoulder\_pan\_joint关节的<dynamics>标签：



* damping：关节的阻尼，现在为理想情况，=0；
* friction：关节的摩擦，现在为理想情况，=0。

**（5）将机器人固定到地板上**

最后一步，是指定机器人与仿真环境[世界坐标系](https://zhida.zhihu.com/search?content_id=231138615&content_type=Article&match_order=1&q=%E4%B8%96%E7%95%8C%E5%9D%90%E6%A0%87%E7%B3%BB&zhida_source=entity" \t "/home/smart/Documents\\x/_blank)之间的关系，在下图中，定义了一个额外的固定（fixed）关节world\_joint，其父连杆为world，子连杆为base\_link：



### **3、机械臂为什么软塌塌地趴在地上？**

我们可以看到，加载后的UR5模型软塌塌地趴在地面上。这是因为机械臂受到重力作用，其各个关节又没有摩擦和阻尼。

要想它不趴在地上，就需要为UR5机械臂添加一个控制器（controller）来控制它，下一篇文章我们将讨论这个问题。

问题及解决

<https://blog.51cto.com/u_12369060/10370712>

## **五、参考文献**

[ROS Tutorial: Simulate the UR5 robot in Gazebo – URDF explained​roboticscasual.com/ros-tutorial-simulate-ur5-robot-in-gazebo-urdf-explained/](https://link.zhihu.com/?target=https://roboticscasual.com/ros-tutorial-simulate-ur5-robot-in-gazebo-urdf-explained/" \t "/home/smart/Documents\\x/_blank)

<https://roboticscasual.com/ros-tutorial-simulate-ur5-robot-in-gazebo-urdf-explained/>

<https://zhuanlan.zhihu.com/p/643536135>

<https://roboticscasual.com/ros-tutorial-simulate-ur5-robot-in-gazebo-urdf-explained/>

<https://zhuanlan.zhihu.com/p/667103750>

<https://blog.csdn.net/inghoG/article/details/116356748>

<https://roboticscasual.com/ros-tutorial-control-the-ur5-robot-with-ros_control-tuning-a-pid-controller/>

<https://github.com/dairal/ur5-joint-position-control>

<https://github.com/ros-industrial/universal_robot/tree/noetic-devel>