[一、操作系统探索总结（一）ROS简介](#操作系统探索总结（一）ROS简介)

[二、操作系统探索总结（二）ROS总体框架](#操作系统探索总结（二）ROS总体框架)

三、[操作系统探索总结（三）ROS新手教程](#操作系统探索总结（三）ROS新手教程)

四、[操作系统探索总结（四）简单的机器人仿真](#操作系统探索总结（四）简单的机器人仿真)

五、[操作系统探索总结（五）创建简单的机器人模型smartcar](#操作系统探索总结（五）创建简单的机器人模型smartcar)

六、[操作系统探索总结（六）使用smartcar进行仿真](#操作系统探索总结（六）使用smartcar进行仿真)

七、[操作系统探索总结（七）smartcar源码上传](#操作系统探索总结（七）smartcar源码上传)

八、[操作系统探索总结（八）键盘控制](#操作系统探索总结（八）键盘控制)

九、[操作系统探索总结（九）操作杆控制](#操作系统探索总结（九）操作杆控制)

十、[操作系统探索总结（十）语音控制](#操作系统探索总结（十）语音控制)

十一、[操作系统探索总结（十一）机器视觉](#操作系统探索总结（十一）机器视觉)

十二、[操作系统探索总结（十二）坐标系统](#操作系统探索总结（十二）坐标系统)

十三、[操作系统探索总结（十三）导航与定位框架](#操作系统探索总结（十三）导航与定位框架)

十四、[操作系统探索总结（十四）move\_base（路径规划）](#操作系统探索总结（十四）move_base（路径规划）)

十五、[操作系统探索总结（十五）amcl（导航与定位）](#操作系统探索总结（十五）amcl（导航与定位）)

**操作系统探索总结（一）ROS简介**

**一、历史**

　　随着机器人领域的快速发展和复杂化，代码的复用性和模块化的需求原来越强烈，而已有的开源机器人系统又不能很好的适应需求。2010年Willow Garage公司发布了开源机器人操作系统ROS(robot operating system)，很快在机器人研究领域展开了学习和使用ROS的热潮。



　　ROS系统是起源于2007年斯坦福大学人工智能实验室的项目与机器人技术公司Willow Garage的个人机器人项目(Personal Robots Program)之间的合作，2008年之后就由Willow Garage来进行推动。已经有四年多的时间了 (视频)。随着PR2那些不可思议的表现，譬如叠衣服，插插座，做早饭，ROS也得到越来越多的关注。Willow Garage公司也表示希望借助开源的力量使PR2变成“全能”机器人。

　　PR2价格高昂，2011年零售价高达40万美元。PR2现主要用于研究。PR2有两条手臂，每条手臂七个关节，手臂末端是一个可以张合的钳子。PR2依靠底部的四个轮子移动。在PR2的头部，胸部，肘部，钳子上安装有高分辨率摄像头，激光测距仪，惯性测量单元，触觉传感器等丰富的传感设备。在PR2的底部有两台8核的电脑作为机器人各硬件的控制和通讯中枢。两台电脑安装有[Ubuntu](http://linux.chinaitlab.com/Special/Ubuntu/Index.html)和ROS。

**二、设计目标**

　　ROS是开源的，是用于机器人的一种后操作系统，或者说次级操作系统。它提供类似操作系统所提供的功能，包含硬件抽象描述、底层驱动程序管理、共用功能的执行、程序间的消息传递、程序发行包管理，它也提供一些工具程序和库用于获取、建立、编写和运行多机整合的程序。



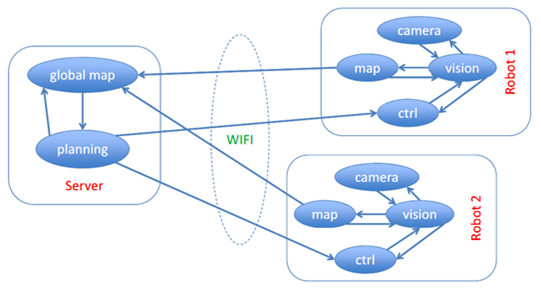
ROS的首要设计目标是在机器人研发领域提高代码复用率。ROS是一种分布式处理框架(又名Nodes)。这使可执行文件能被单独设计，并且在运行时松散耦合。这些过程可以封装到数据包(Packages)和堆栈(Stacks)中，以便于共享和分发。ROS还支持代码库的联合系统。使得协作亦能被分发。这种从文件系统级别到社区一级的设计让独立地决定发展和实施工作成为可能。上述所有功能都能由ROS的基础工具实现。

**三、主要特点**

　　ROS的运行架构是一种使用ROS通信模块实现模块间P2P的松耦合的网络连接的处理架构，它执行若干种类型的通讯，包括基于服务的同步RPC(远程过程调用)通讯、基于Topic的异步数据流通讯，还有参数[服务器](http://server.chinaitlab.com/)上的数据[存储](http://www.storworld.com/)。但是ROS本身并没有实时性。

　　ROS的主要特点可以归纳为以下几条：

　　(1)点对点设计



　　一个使用ROS的系统包括一系列进程，这些进程存在于多个不同的主机并且在运行过程中通过端对端的拓扑结构进行联系。虽然基于中心[服务器](http://server.chinaitlab.com/)的那些软件框架也可以实现多进程和多主机的优势，但是在这些框架中，当各电脑通过不同的网络进行连接时，中心数据服务器就会发生问题。

　　ROS的点对点设计以及服务和节点管理器等机制可以分散由计算机视觉和语音识别等功能带来的实时计算压力，能够适应多机器人遇到的挑战。

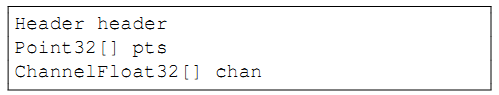
　　(2)多语言支持

　　在写代码的时候，许多编程者会比较偏向某一些编程语言。这些偏好是个人在每种语言的编程时间、调试效果、语法、执行效率以及各种技术和文化的原因导致的结果。为了解决这些问题，我们将ROS设计成了语言中立性的框架结构。ROS现在支持许多种不同的语言，例如[C++](http://c.chinaitlab.com/)、Python、Octave和LISP，也包含其他语言的多种接口实现。



　　ROS的特殊性主要体现在消息通讯层，而不是更深的层次。端对端的连接和配置利用XML-RPC机制进行实现，XML-RPC也包含了大多数主要语言的合理实现描述。我们希望ROS能够利用各种语言实现的更加自然，更符合各种语言的语法约定，而不是基于C语言给各种其他语言提供实现接口。然而，在某些情况下利用已经存在的库封装后支持更多新的语言是很方便的，比如Octave的客户端就是通过[C++](http://c.chinaitlab.com/)的封装库进行实现的。

　　为了支持交叉语言，ROS利用了简单的、语言无关的接口定义语言去描述模块之间的消息传送。接口定义语言使用了简短的文本去描述每条消息的结构，也允许消息的合成，例如下图就是利用接口定义语言描述的一个点的消息：



　　每种语言的代码产生器就会产生类似本种语言目标文件，在消息传递和接收的过程中通过ROS自动连续并行的实现。这就节省了重要的编程时间，也避免了错误：之前3行的接口定义文件自动的扩展成137行的C++代码，96行的Python代码，81行的Lisp代码和99行的Octave代码。因为消息是从各种简单的文本文件中自动生成的，所以很容易列举出新的消息类型。在编写的时候，已知的基于ROS的代码库包含超过四百种消息类型，这些消息从传感器传送数据，使得物体检测到了周围的环境。

　　最后的结果就是一种语言无关的消息处理，让多种语言可以自由的混合和匹配使用。

　　(3)精简与集成

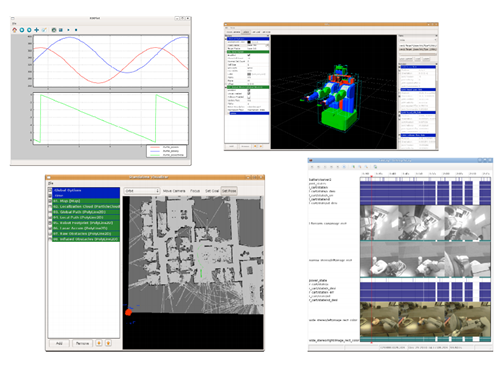
　　大多数已经存在的机器人软件工程都包含了可以在工程外重复使用的驱动和算法，不幸的是，由于多方面的原因，大部分代码的中间层都过于混乱，以至于很困难提取出它的功能，也很难把它们从原型中提取出来应用到其他方面。

　　为了应对这种趋势，我们鼓励将所有的驱动和算法逐渐发展成为和ROS没有依赖性单独的库。ROS建立的系统具有模块化的特点，各模块中的代码可以单独编译，而且编译使用的CMake工具使它很容易的就实现精简的理念。ROS基本将复杂的代码封装在库里，只是创建了一些小的应用程序为ROS显示库的功能，就允许了对简单的代码超越原型进行移植和重新使用。作为一种新加入的有优势，单元[测试](http://softtest.chinaitlab.com/)当代码在库中分散后也变得非常的容易，一个单独的[测试](http://softtest.chinaitlab.com/)程序可以测试库中很多的特点。

　　ROS利用了很多现在已经存在的开源项目的代码，比如说从Player项目中借鉴了驱动、运动控制和仿真方面的代码，从OpenCV中借鉴了视觉算法方面的代码，从OpenRAVE借鉴了规划算法的内容，还有很多其他的项目。在每一个实例中，ROS都用来显示多种多样的配置选项以及和各软件之间进行数据通信，也同时对它们进行微小的包装和改动。ROS可以不断的从社区维护中进行升级，包括从其他的软件库、应用补丁中升级ROS的源代码。

　　(4)工具包丰富

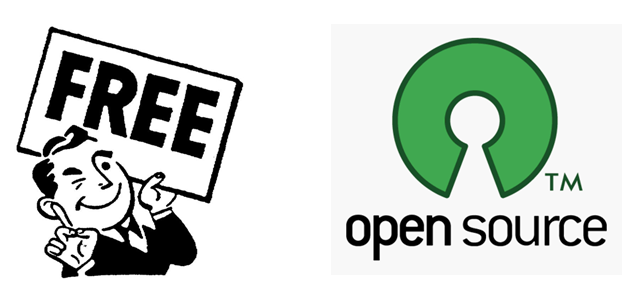
　　为了管理复杂的ROS软件框架，我们利用了大量的小工具去编译和运行多种多样的ROS组建，从而设计成了内核，而不是构建一个庞大的开发和运行环境。



　　这些工具担任了各种各样的任务，例如，组织源代码的结构，获取和设置配置参数，形象化端对端的拓扑连接，测量频带使用宽度，生动的描绘信息数据，自动生成文档等等。尽管我们已经测试通过像全局时钟和控制器模块的记录器的核心服务，但是我们还是希望能把所有的代码模块化。我们相信在效率上的损失远远是稳定性和管理的复杂性上无法弥补的。

　　(5)免费并且开源

　　ROS所有的源代码都是公开发布的。我们相信这将必定促进ROS软件各层次的调试，不断的改正错误。虽然像Microsoft Robotics Studio和Webots这样的非开源软件也有很多值得赞美的属性，但是我们认为一个开源的平台也是无可为替代的。当硬件和各层次的软件同时设计和调试的时候这一点是尤其真实的。



　　ROS以分布式的关系遵循这BSD许可，也就是说允许各种商业和非商业的工程进行开发。ROS通过内部处理的通讯系统进行数据的传递，不要求各模块在同样的可执行功能上连接在一起。如此，利用ROS构建的系统可以很好的使用他们丰富的组件：个别的模块可以包含被各种[协议](http://cisco.chinaitlab.com/List_11.html)保护的软件，这些[协议](http://cisco.chinaitlab.com/List_11.html)从GPL到BSD，但是许可的一些“污染物”将在模块的分解上就完全消灭掉

**操作系统探索总结（二）ROS总体框架**

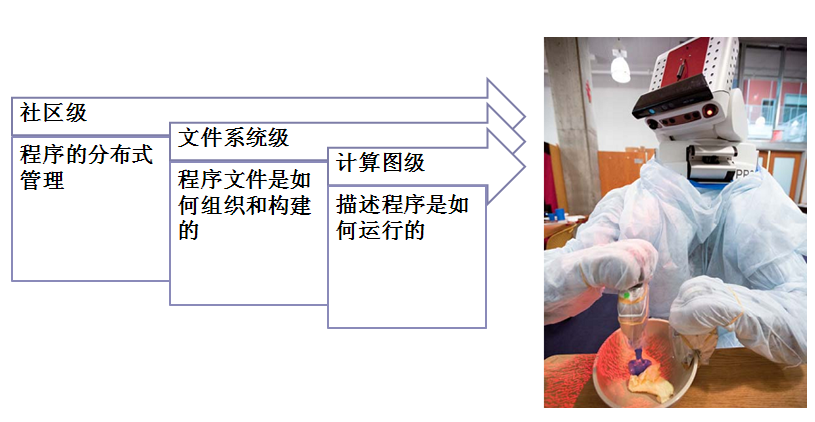
**一、 总体结构**

　　根据ROS系统代码的维护者和分布来标示，主要有两大部分：

　　(1)main：核心部分，主要由Willow Garage公司和一些开发者设计、提供以及维护。它提供了一些分布式计算的基本工具，以及整个ROS的核心部分的程序编写。

　　(2)universe：全球范围的代码，有不同国家的ROS社区组织开发和维护。一种是库的代码，如OpenCV、PCL等;库的上一层是从功能角度提供的代码，如人脸识别，他们调用下层的库;最上层的代码是应用级的代码，让机器人完成某一确定的功能。

　　一般是从另一个角度对ROS分级的，主要分为三个级别：计算图级、文件系统级、社区级。



**二、 计算图级**

　　计算图是ROS处理数据的一种点对点的网络形式。程序运行时，所有进程以及他们所进行的数据处理，将会通过一种点对点的网络形式表现出来。这一级主要包括几个重要概念：节点(node)、消息(message)、主题(topic)、服务(service)。



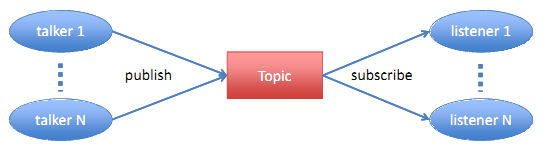
　　(1) 节点

　　节点就是一些直行运算任务的进程。ROS利用规模可增长的方式是代码模块化：一个系统就是典型的由很多节点组成的。在这里，节点也可以被称之为“软件模块”。我们使用“节点”使得基于ROS的系统在运行的时候更加形象化：当许多节点同时运行时，可以很方便的将端对端的通讯绘制成一个图表，在这个图表中，进程就是图中的节点，而端对端的连接关系就是其中弧线连接。

　　(2) 消息

　　节点之间是通过传送消息进行通讯的。每一个消息都是一个严格的数据结构。原来标准的数据类型(整型，浮点型，布尔型等等)都是支持的，同时也支持原始数组类型。消息可以包含任意的嵌套结构和数组(很类似于C语言的结构structs)。

　　(3) 主题



　　消息以一种发布/订阅的方式传递。一个节点可以在一个给定的主题中发布消息。一个节点针对某个主题关注与订阅特定类型的数据。可能同时有多个节点发布或者订阅同一个主题的消息。总体上，发布者和订阅者不了解彼此的存在。

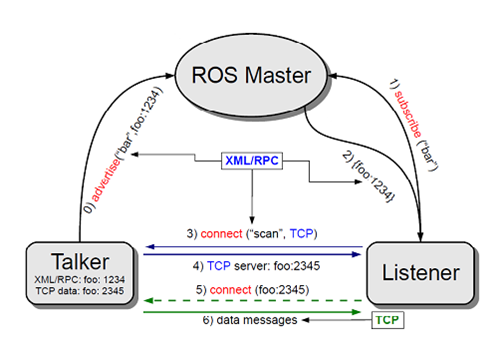
(4) 服务

　　虽然基于话题的发布/订阅模型是很灵活的通讯模式，但是它广播式的路径规划对于可以简化节点设计的同步传输模式并不适合。在ROS中，我们称之为一个服务，用一个字符串和一对严格规范的消息定义：一个用于请求，一个用于回应。这类似于web[服务器](http://server.chinaitlab.com/)，web[服务器](http://server.chinaitlab.com/)是由URIs定义的，同时带有完整定义类型的请求和回复文档。需要注意的是，不像话题，只有一个节点可以以任意独有的名字广播一个服务：只有一个服务可以称之为“分类象征”，比如说，任意一个给出的URI地址只能有一个web服务器。

　　在上面概念的基础上，需要有一个控制器可以使所有节点有条不紊的执行，这就是一个ROS的控制器(ROS Master)。

　　ROS Master 通过RPC(Remote Procedure Call Protocol，远程过程调用)提供了登记列表和对其他计算图表的查找。没有控制器，节点将无法找到其他节点，[交换](http://cisco.chinaitlab.com/List_7.html)消息或调用服务。

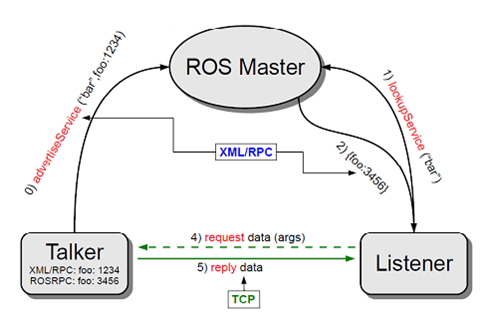
　　比如控制节点订阅和发布消息的模型如下：



　　ROS的控制器给ROS的节点[存储](http://www.storworld.com/)了主题和服务的注册信息。节点与控制器通信从而报告它们的注册信息。当这些节点与控制器通信的时候，它们可以接收关于其他以注册及节点的信息并且建立与其它以注册节点之间的联系。当这些注册信息改变时控制器也会回馈这些节点，同时允许节点动态创建与新节点之间的连接。

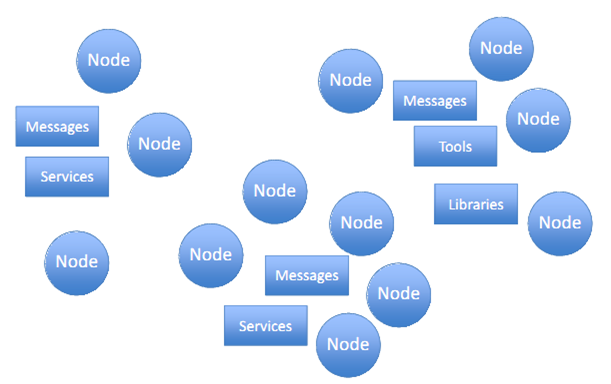
　　节点与节点之间的连接是直接的，控制器仅仅提供了查询信息，就像一个DNS服务器。节点订阅一个主题将会要求建立一个与出版该主题的节点的连接，并且将会在同意连接[协议](http://cisco.chinaitlab.com/List_11.html)的基础上建立该连接。

　　另：ROS控制器控制服务：



**三、 文件系统级**

　　ROS文件系统级指的是在硬盘上面查看的ROS源代码的组织形式。



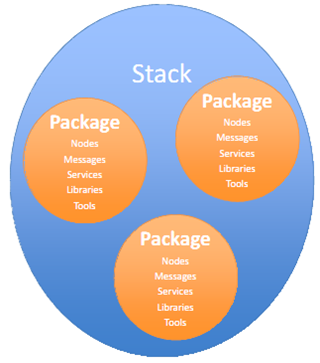
　　ROS中有无数的节点、消息、服务、工具和库文件，需要有效的结构去管理这些代码。在ROS的文件系统级，有以下几个重要概念：包(package)、堆(stack)、

　　(1) 包



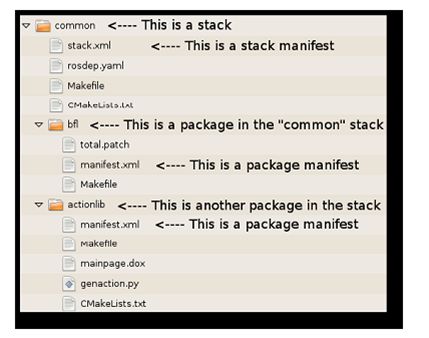
　　ROS的软件以包的方式组织起来。包包含节点、ROS依赖库、数据套、配置文件、第三方软件、或者任何其他逻辑构成。包的目标是提供一种易于使用的结构以便于软件的重复使用。总得来说，ROS的包短小精干。

　　(2) 堆



　　堆是包的集合，它提供一个完整的功能，像“navigation stack”。Stack与版本号关联，同时也是如何发行ROS软件方式的关键

　ROS是一种分布式处理框架。这使可执行文件能被单独设计，并且在运行时松散耦合。这些过程可以封装到包(Packages)和堆(Stacks)中，以便于共享和分发。下图是在包和堆在文件中的具体结构：

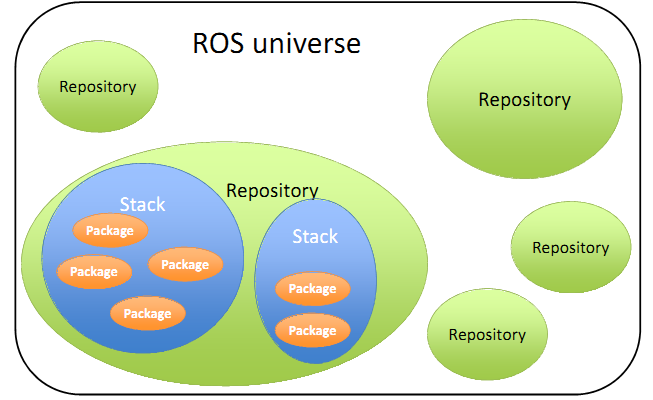


　　Manifests (manifest.xml)：提供关于Package元数据，包括它的许可信息和Package之间依赖关系，以及语言特性信息像编译旗帜(编译优化参数)。

　　Stack manifests (stack.xml)：提供关于Stack元数据，包括它的许可信息和Stack之间依赖关系。

**四、 社区级**

　　ROS的社区级概念是ROS网络上进行代码发布的一种表现形式。结构如下图所示：



　　代码库的联合系统。使得协作亦能被分发。这种从文件系统级别到社区一级的设计让独立地发展和实施工作成为可能。正是因为这种分布式的结构，似的ROS迅速发展，软件仓库中包的数量指数级增加。

**操作系统探索总结（三）ROS新手教程**

　　前面我们介绍了ROS的特点和结构，接下来就要开始准备动手感受一下ROS的强大了。ROS官网的wiki上针对新手的教程很详细，最好把所有的新手教程都搞清楚，这是后面开发最基础的东西。尽管如此，ROS对于新手来说还是很难上手，这里，我就来总结一下我当时学习的历程，也为其他新手作为一个参考。

**一、ROS的安装**



　　ROS的安装当然是我们开始动手的第一步了，这里我们使用的操作系统是ubuntu，因为ROS在ubuntu上的支持是最好的。

　　如果是新手，我建议使用”apt-get“的方法进行安装，不走很简单，按照wiki上说的，大概半个小时就可以安装完毕完全版的ROS：(现在最新版的ROS是groovy，但是我还是习惯使用fuerte)

　　http://ros.org/wiki/fuerte/Installation/[Ubuntu](http://linux.chinaitlab.com/Special/Ubuntu/Index.html)

　　如果想挑战源码编译，当然也没有问题：

　　http://ros.org/wiki/fuerte/Installation/[Ubuntu](http://linux.chinaitlab.com/Special/Ubuntu/Index.html)/Source

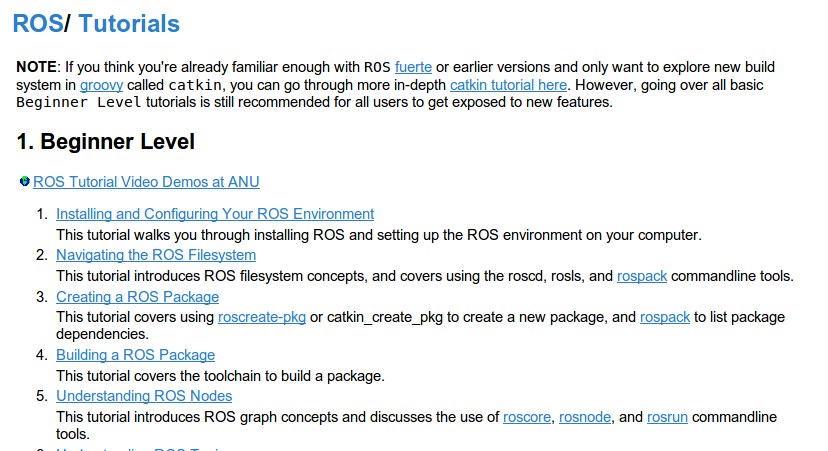
　　安装完毕之后运行一下“roscore”，如果没有问题，安装就成功了!

　　注：groovy版本的安装：

　　“apt-get”：http://www.ros.org/wiki/groovy/Installation/[Ubuntu](http://linux.chinaitlab.com/Special/Ubuntu/Index.html)

　　源码编译：http://www.ros.org/wiki/groovy/Installation/Source

**二、ROS的新手教程**



　　wiki上的新手教程还是很详细的，对代码都有解释，新手一定要把这些例子和代码搞明白：

　　http://www.ros.org/wiki/ROS/Tutorials

　　上面的教程都是英文的，如果感觉略有压力(本人就是)，可以参考下面这两个博客中的部分翻译：

　　http://blog.sina.com.cn/s/articlelist\_1712413141\_0\_1.html

　　不过往后面的学习都是英文的资料了，还是要努力适应看英文的文档。

　　ROS使用的编程语言主要是[C++](http://c.chinaitlab.com/)和python，所以也有针对这两种语言的功能包roscpp和rospy，这两个包的教程与上面的教程基本相似，看完上面的教程也可以看看这两个包的教程：

　　http://www.ros.org/wiki/rospy\_tutorials

　　http://www.ros.org/wiki/roscpp/Tutorials

　　努力学习完上面的这些教程，你至少应该明白ROS里面的节点和消息是干什么用的了吧，如果还没理解，那就再多看几遍吧!

**三、ROS中的常用功能**

　　ROS中提供了很多强大的功能，我们学习完上面的基本知识之后要继续进行深入。

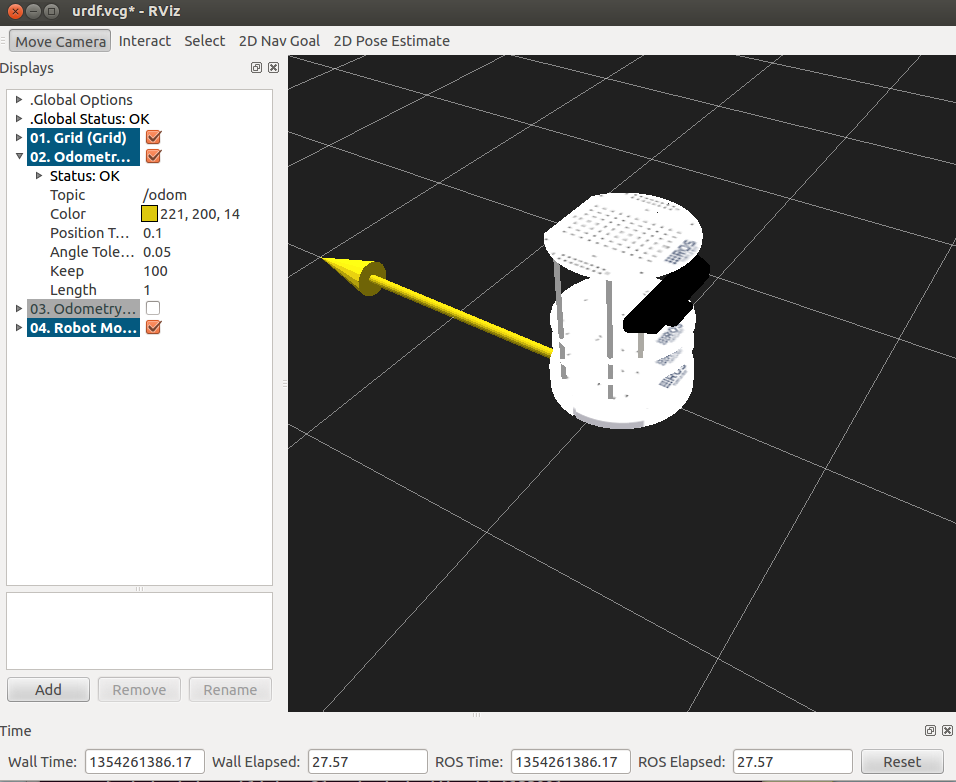
　　1、rviz

　　rviz是ROS中一款强大的3D可视化工具，这个玩意在后面可是要频繁用到的，是必须要弄明白的，详细的教程可以参考wiki：

　　http://www.ros.org/wiki/rviz

　　我们可以在里面创建自己的机器人，并且让机器人动起来。还可以创建地图，显示3D点云等等，总之，想在ROS中显示的东东都可以在这里显示出来。当然这些显示都是通过消息的订阅来完成的，机器人通过ROS发布数据，rviz订阅消息接收数据，然后显示，这些数据也是有一定的数据格式的，可以参考下面的链接：

　　http://www.ros.org/wiki/rviz/DisplayTypes

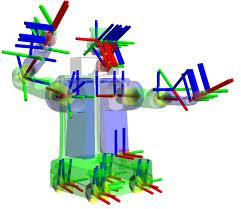


　　看到上面的机器人了吧，是不是很酷，在rviz中，这样的机器人模型是通过urdf文件描述的，具体urdf文件怎么写，参考wiki：

　　http://www.ros.org/wiki/urdf

　　2、tf

　　tf是ROS中的坐标变换系统，在机器人的建模仿真中经常用到。



　　ROS中主要有两种坐标系：

　　(1)固定坐标系：用于表示世界的参考坐标系;

　　(2)目标坐标系：相对于摄像机视角的参考坐标系。

　　教程见：http://www.ros.org/wiki/tf

　　3、gazebo

　　这个工具是ROS中的物理仿真环境，gazebo本身就是一款机器人的仿真软件，基于ODE的物理引擎，可以模拟机器人以及环境中的很多物理特性，这个软件可以稍作了解，并不是后面开发所必须要的。

　　教程见：http://www.ros.org/wiki/simulator\_gazebo/Tutorials

**四、ROS常用机器人**

　　1、PR2



　　看ROS的应用时，最常见到的机器人就是PR2。这个机器人是ROS的主要维护者(Willow Garage)针对ROS量身定做的机器人，有两个运行ubuntu和ROS的电脑，和两个机器臂以及很多牛逼的传感器，功能是非常强大的，但是售价那是相当的昂贵，国内很少见到，基本都是在国外的研究所里。这款机器人的ROS包比较多，从仿真到导航，所以代码具有比较高的参考价值，当然新手还是先看看其他机器人的代码再来挑战PR2吧，代码比较庞杂。

　　wiki：http://www.ros.org/wiki/Robots/PR2

　　2、TurtleBot



　　这个机器人应该算是应用ROS小型移动机器人的典型代表了，资料、文档和代码比较多，主要在建立模型和导航定位方面，代码比较容易理解，可以作为新手参考的最佳机器人了，上面rviz中显示的那个机器人就是它了。

　　wiki：http://www.ros.org/wiki/Robots/TurtleBot

　　3、Husky、Erratic



　　这两款机器人和TurtleBot机器人差不多，都是小型的轮式移动机器人，同样可以作为新手学习的参考：

　　http://www.ros.org/wiki/Robots/Husky

　　http://www.ros.org/wiki/Robots/Erratic

　　这些教程和机器人基本都是我当时学习的时候研究和了解过的，总结在此也让那些比较迷茫的初学者作为参考，可以尽快了解和掌握ROS。再次强调一下，wiki上的新手教程是一定要先熟悉的，后面的内容在后续学习过程中可以一边做一边学习。

**操作系统探索总结（四）简单的机器人仿真**

　　前边我们已经介绍了ROS的基本情况，以及新手入门ROS的初级教程，现在就要真正的使用ROS进入机器人世界了。接下来我们涉及到的很多例程都是《ROS by Example》这本书的内容，我是和群里的几个人一起从国外的亚马逊上买到的，还是很有参考价值的，不过前提是你已经熟悉之前的新手教程了。

　　一、ROS by Example



　　这本书是关于国外关于ROS出版的第一本书，主要针对Electric和Fuerte版本，使用机器人主要是TurtleBot。书中详细讲解了关于机器人的基本仿真、导航、路径规划、图像处理、语音识别等等，而且在google的svn上发布了所有代码，可以通过以下命令[下载](http://download.chinaitlab.com/)、编译：

　　svn checkout http://ros-by-example.googlecode.com/svn/trunk/rbx\_vol\_1

　　rosmake rbx\_vol\_1

　　rospack profile //加入ROS package路径

　　svn checkout http://ros-by-example.googlecode.com/svn/trunk/rbx\_vol\_1

　　rosmake rbx\_vol\_1

　　rospack profile //加入ROS package路径

　　二、rviz简单机器人模拟

　　1、安装机器人模拟器

　　rviz是一个显示机器人实体的工具，本身不具有模拟的功能，需要安装一个模拟器arbotix。

　　svn checkout http://vanadium-ros-pkg.googlecode.com/svn/trunk/arbotix

　　rosmake arbotix

　　svn checkout http://vanadium-ros-pkg.googlecode.com/svn/trunk/arbotix

　　rosmake arbotix

　　2、TurtleBot机器人的模拟

　　在书中的rbx\_vol\_1包里已经为我们写好了模拟的代码，我们先进行实验，完成后再仔细研究代码。

　　机器人模拟运行：

　　roscore

　　roslaunch rbx1\_bringup fake\_pi\_robot.launch

　　roscore

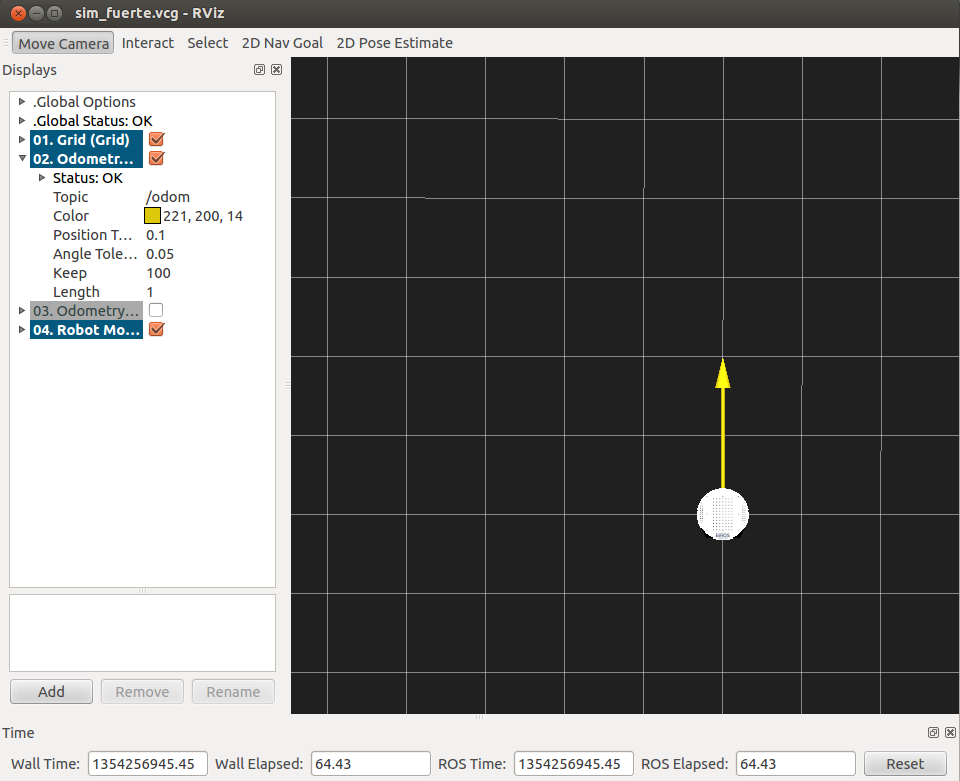
　　roslaunch rbx1\_bringup fake\_pi\_robot.launch

　　然后在终端中可以看到，机器人已经开始运行了，打开rviz界面，才能看到机器人实体。

　　rosrun rviz rviz -d `rospack find rbx1\_nav`/sim\_fuerte.vcg

　　rosrun rviz rviz -d `rospack find rbx1\_nav`/sim\_fuerte.vcg

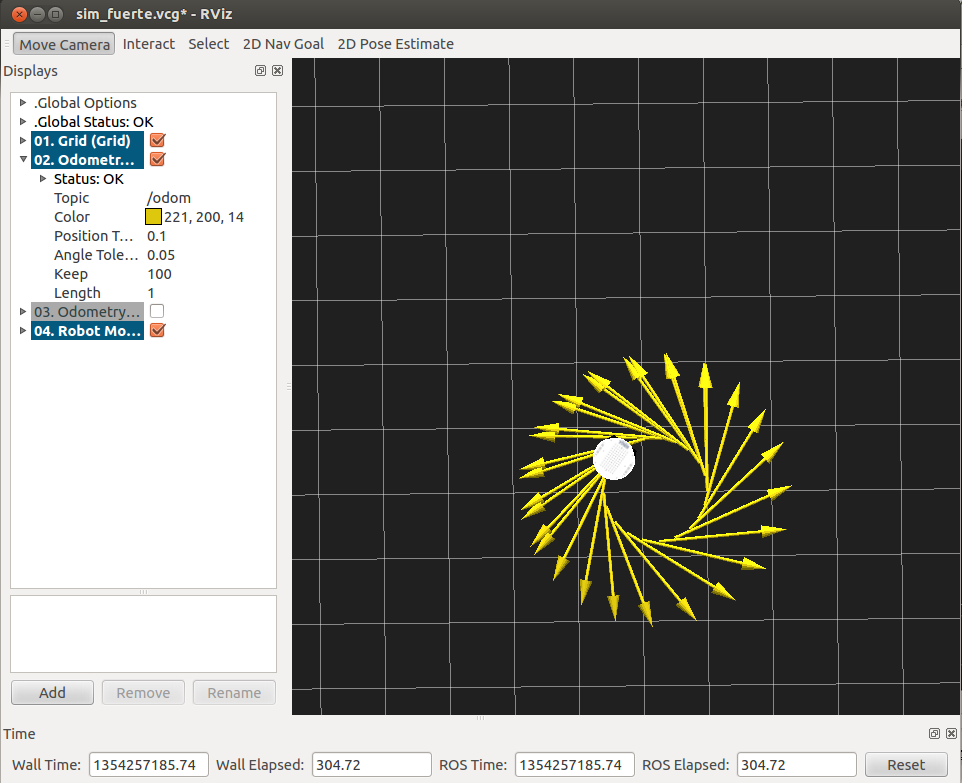
　　后面的参数是加载了rviz的配置文件sim\_fuerte.vcg。效果如下：



　　此时的机器人是静止的，需要发布一个消息才能让它动起来。

　　rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.2, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0.5}}'

　　rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.2, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0.5}}'



　　如果要让机器人停下来，需要在中断中按下“Ctrl+c”，然后输入：

　　rostopic pub -1 /cmd\_vel geometry\_msgs/Twist '{}'

　　rostopic pub -1 /cmd\_vel geometry\_msgs/Twist '{}'

　　也可以改变发送的topic信息，使机器人走出不同的轨迹。

　　三、实现分析

　　按照上面的仿真过程，我们详细分析每一步的代码实现。

　　1、TurtleBot机器人运行

　　机器人运行使用的是launch文件，首先打开fake\_turtlebot.launch文件。

　　文件可以大概分为四个部分：

　　(1) 从指定的包中加载urdf文件

　　(2) 启动arbotix模拟器

　　(3) 启动状态发布节点

　　(4) tf坐标系配置

　　2、rviz配置文件

　　在打开rviz的时候需要加载一个.vcg的配置文件，主要对rviz中的插件选项进行默认的配置。这里打开的是sim\_fuerte.vcg文件，由于文件比较长，这里只列举重点的部分。

　　Background\ ColorB=0.12549

　　Background\ ColorG=0.12549

　　Background\ ColorR=0.12549

　　Camera\ Config=158.108 0.814789 0.619682 -1.57034

　　Camera\ Type=rviz::FixedOrientationOrthoViewController

　　Fixed\ Frame=/odom

　　Grid.Alpha=0.5

　　Grid.Cell\ Size=0.5

　　Grid.ColorB=0.941176

　　Grid.ColorG=0.941176

　　Grid.ColorR=0.941176

　　Grid.Enabled=1

　　Grid.Line\ Style=0

　　Grid.Line\ Width=0.03

　　Grid.Normal\ Cell\ Count=0

　　Grid.OffsetX=0

　　Grid.OffsetY=0

　　Grid.OffsetZ=0

　　Grid.Plane=0

　　Background\ ColorB=0.12549

　　Background\ ColorG=0.12549

　　Background\ ColorR=0.12549

　　Camera\ Config=158.108 0.814789 0.619682 -1.57034

　　Camera\ Type=rviz::FixedOrientationOrthoViewController

　　Fixed\ Frame=/odom

　　Grid.Alpha=0.5

　　Grid.Cell\ Size=0.5

　　Grid.ColorB=0.941176

　　Grid.ColorG=0.941176

　　Grid.ColorR=0.941176

　　Grid.Enabled=1

　　Grid.Line\ Style=0

　　Grid.Line\ Width=0.03

　　Grid.Normal\ Cell\ Count=0

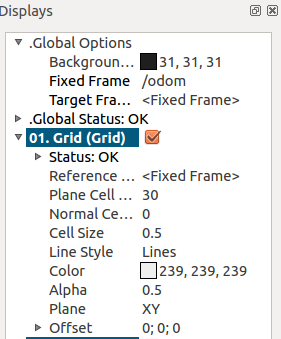
　　Grid.OffsetX=0

　　Grid.OffsetY=0

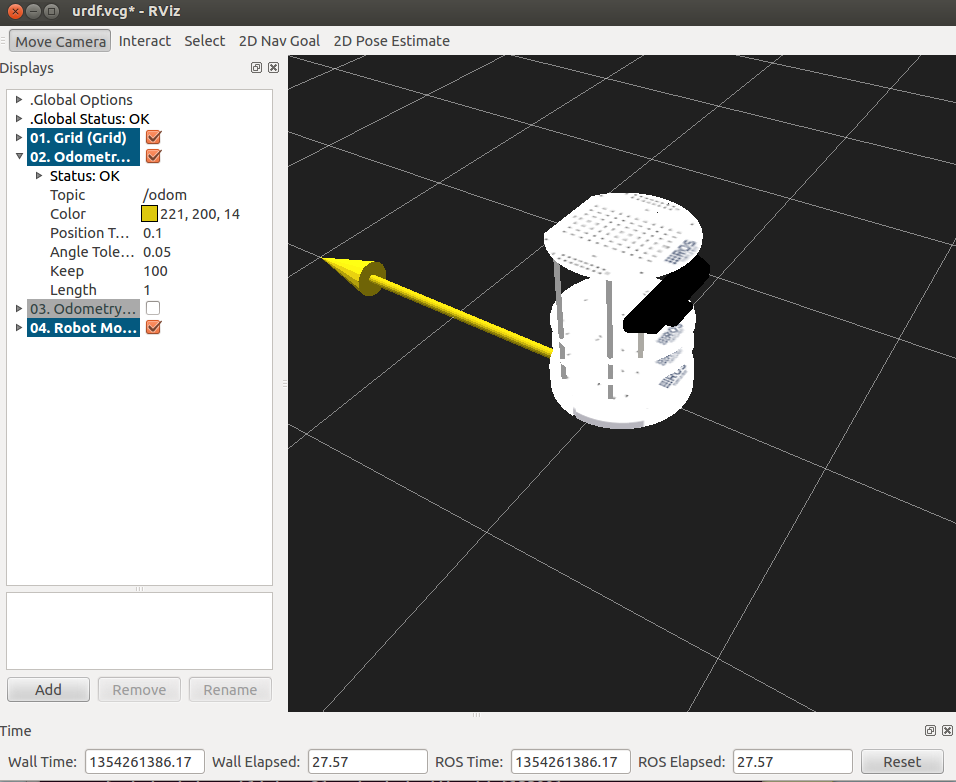
　　Grid.OffsetZ=0

　　Grid.Plane=0

　　上面的代码是配置背景颜色和网格属性的，对应rviz中的选项如下图所示。



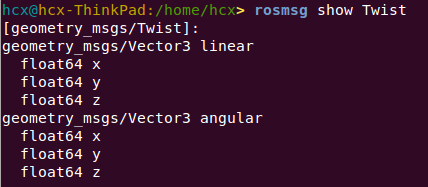
　其中比较重要的一个选项是Camera的type，这个选项是控制开发者的观察角度的，书中用的是FixedOrientationOrthoViewController的方式，就是上面图中的俯视角度，无法看到机器人的三维全景，所以可以改为OrbitViewController方式，如下图所示：



　　3、发布topic

　　要让机器人动起来，还需要给他一些运动需要的信息，这些信息都是通过topic的方式发布的。

　　这里的topic就是速度命令，针对这个topic，我们需要发布速度的信息，在ROS中已经为我们写好了一些可用的数据结构，这里用的是Twist信息的数据结构。在终端中可以看到Twist的结构如下：



　　用下面的命令进行消息的发布，其中主要包括力的大小和方向。

　　Background\ ColorB=0.12549

　　Background\ ColorG=0.12549

　　Background\ ColorR=0.12549

　　Camera\ Config=158.108 0.814789 0.619682 -1.57034

　　Camera\ Type=rviz::FixedOrientationOrthoViewController

　　Fixed\ Frame=/odom

　　Grid.Alpha=0.5

　　Grid.Cell\ Size=0.5

　　Grid.ColorB=0.941176

　　Grid.ColorG=0.941176

　　Grid.ColorR=0.941176

　　Grid.Enabled=1

　　Grid.Line\ Style=0

　　Grid.Line\ Width=0.03

　　Grid.Normal\ Cell\ Count=0

　　Grid.OffsetX=0

　　Grid.OffsetY=0

　　Grid.OffsetZ=0

　　Grid.Plane=0

　　Background\ ColorB=0.12549

　　Background\ ColorG=0.12549

　　Background\ ColorR=0.12549

　　Camera\ Config=158.108 0.814789 0.619682 -1.57034

　　Camera\ Type=rviz::FixedOrientationOrthoViewController

　　Fixed\ Frame=/odom

　　Grid.Alpha=0.5

　　Grid.Cell\ Size=0.5

　　Grid.ColorB=0.941176

　　Grid.ColorG=0.941176

　　Grid.ColorR=0.941176

　　Grid.Enabled=1

　　Grid.Line\ Style=0

　　Grid.Line\ Width=0.03

　　Grid.Normal\ Cell\ Count=0

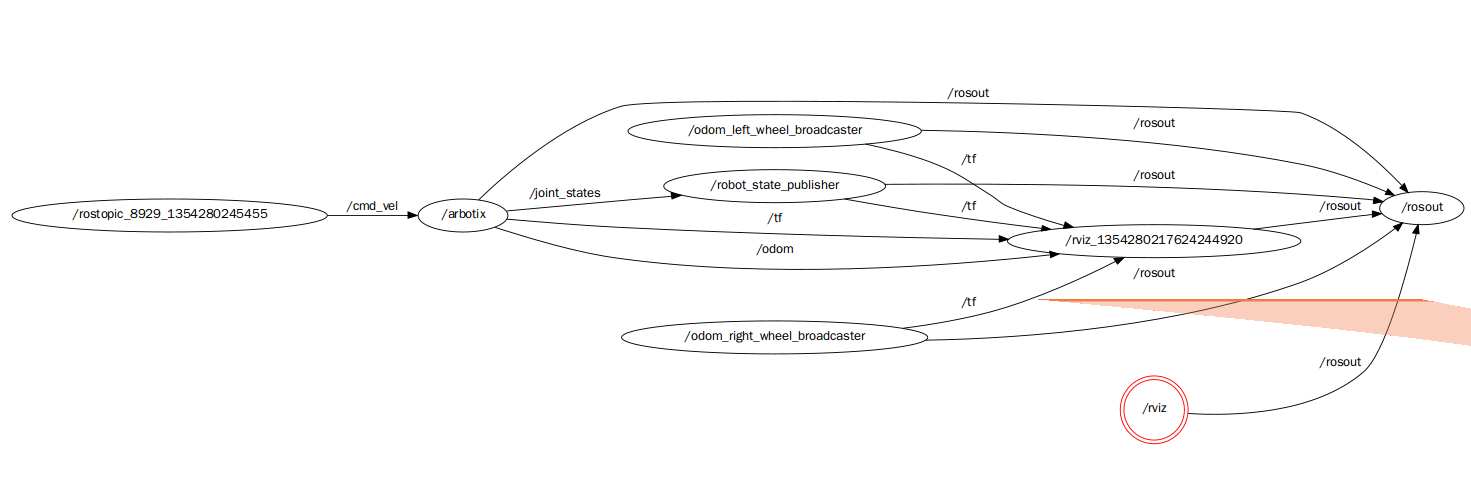
　　Grid.OffsetX=0

　　Grid.OffsetY=0

　　Grid.OffsetZ=0

　　Grid.Plane=0

　　4、节点关系图

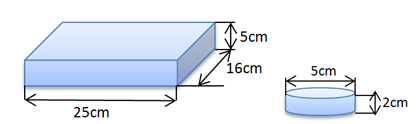


**操作系统探索总结（五**）**创建简单的机器人模型smartcar**

    前面我们使用的是已有的机器人模型进行仿真，这一节我们将建立一个简单的智能车机器人smartcar,为后面建立复杂机器人打下基础。

**一、创建硬件描述包**

    01.roscreat-pkg  smartcar\_description  urdf  
  
**二、智能车尺寸数据**  
    因为建立的是一个非常简单的机器人，所以我们尽量使用简单的元素：使用长方体代替车模，使用圆柱代替车轮，具体尺寸如下：



**三、建立urdf文件**  
    在smartcar\_description文件夹下建立urdf文件夹，创建智能车的描述文件smartcar.urdf,描述代码如下：

    01.<?xml version="1.0"?>  
    02.<robot name="smartcar">  
    03.  <link name="base\_link">  
    04.    <visual>  
    05.      <geometry>  
    06.        <box size="0.25 .16 .05"/>  
    07.    </geometry>  
    08.    <origin rpy="0 0 1.57075" xyz="0 0 0"/>  
    09.    <material name="blue">  
    10.        <color rgba="0 0 .8 1"/>  
    11.    </material>  
    12.    </visual>  
    13. </link>  
    14.  
    15. <link name="right\_front\_wheel">  
    16.    <visual>  
    17.      <geometry>  
    18.        <cylinder length=".02" radius="0.025"/>  
    19.      </geometry>  
    20.      <material name="black">  
    21.        <color rgba="0 0 0 1"/>  
    22.      </material>  
    23.    </visual>  
    24.  </link>  
    25.  
    26.  <joint name="right\_front\_wheel\_joint" type="continuous">  
    27.    <axis xyz="0 0 1"/>  
    28.    <parent link="base\_link"/>  
    29.    <child link="right\_front\_wheel"/>  
    30.    <origin rpy="0 1.57075 0" xyz="0.08 0.1 -0.03"/>  
    31.    <limit effort="100" velocity="100"/>  
    32.    <joint\_properties damping="0.0" friction="0.0"/>  
    33.  </joint>  
    34.  
    35.  <link name="right\_back\_wheel">  
    36.    <visual>  
    37.      <geometry>  
    38.        <cylinder length=".02" radius="0.025"/>  
    39.      </geometry>  
    40.      <material name="black">  
    41.        <color rgba="0 0 0 1"/>  
    42.      </material>  
    43.    </visual>  
    44.  </link>  
    45.  
    46.  <joint name="right\_back\_wheel\_joint" type="continuous">  
    47.    <axis xyz="0 0 1"/>  
    48.    <parent link="base\_link"/>  
    49.    <child link="right\_back\_wheel"/>  
    50.    <origin rpy="0 1.57075 0" xyz="0.08 -0.1 -0.03"/>  
    51.    <limit effort="100" velocity="100"/>  
    52.    <joint\_properties damping="0.0" friction="0.0"/>  
    53. </joint>  
    54.  
    55. <link name="left\_front\_wheel">  
    56.    <visual>  
    57.      <geometry>  
    58.        <cylinder length=".02" radius="0.025"/>  
    59.      </geometry>  
    60.      <material name="black">  
    61.        <color rgba="0 0 0 1"/>  
    62.      </material>  
    63.    </visual>  
    64.  </link>  
    65.  
    66.  <joint name="left\_front\_wheel\_joint" type="continuous">  
    67.    <axis xyz="0 0 1"/>  
    68.    <parent link="base\_link"/>  
    69.    <child link="left\_front\_wheel"/>  
    70.    <origin rpy="0 1.57075 0" xyz="-0.08 0.1 -0.03"/>  
    71.    <limit effort="100" velocity="100"/>  
    72.    <joint\_properties damping="0.0" friction="0.0"/>  
    73.  </joint>  
    74.  
    75.  <link name="left\_back\_wheel">  
    76.    <visual>  
    77.      <geometry>  
    78.        <cylinder length=".02" radius="0.025"/>  
    79.      </geometry>  
    80.      <material name="black">  
    81.        <color rgba="0 0 0 1"/>  
    82.      </material>  
    83.    </visual>  
    84.  </link>  
    85.  
    86.  <joint name="left\_back\_wheel\_joint" type="continuous">  
    87.    <axis xyz="0 0 1"/>  
    88.    <parent link="base\_link"/>  
    89.    <child link="left\_back\_wheel"/>  
    90.    <origin rpy="0 1.57075 0" xyz="-0.08 -0.1 -0.03"/>  
    91.    <limit effort="100" velocity="100"/>  
    92.    <joint\_properties damping="0.0" friction="0.0"/>  
    93.  </joint>  
    94.  
    95.  <link name="head">  
    96.    <visual>  
    97.      <geometry>  
    98.        <box size=".02 .03 .03"/>  
    99.      </geometry>  
    100.      <material name="white">  
    101.          <color rgba="1 1 1 1"/>  
    102.      </material>  
    103.    </visual>  
    104.  </link>  
    105.  
    106.  <joint name="tobox" type="fixed">  
    107.    <parent link="base\_link"/>  
    108.    <child link="head"/>  
    109.    <origin xyz="0 0.08 0.025"/>  
    110.  </joint>  
    111.</robot>

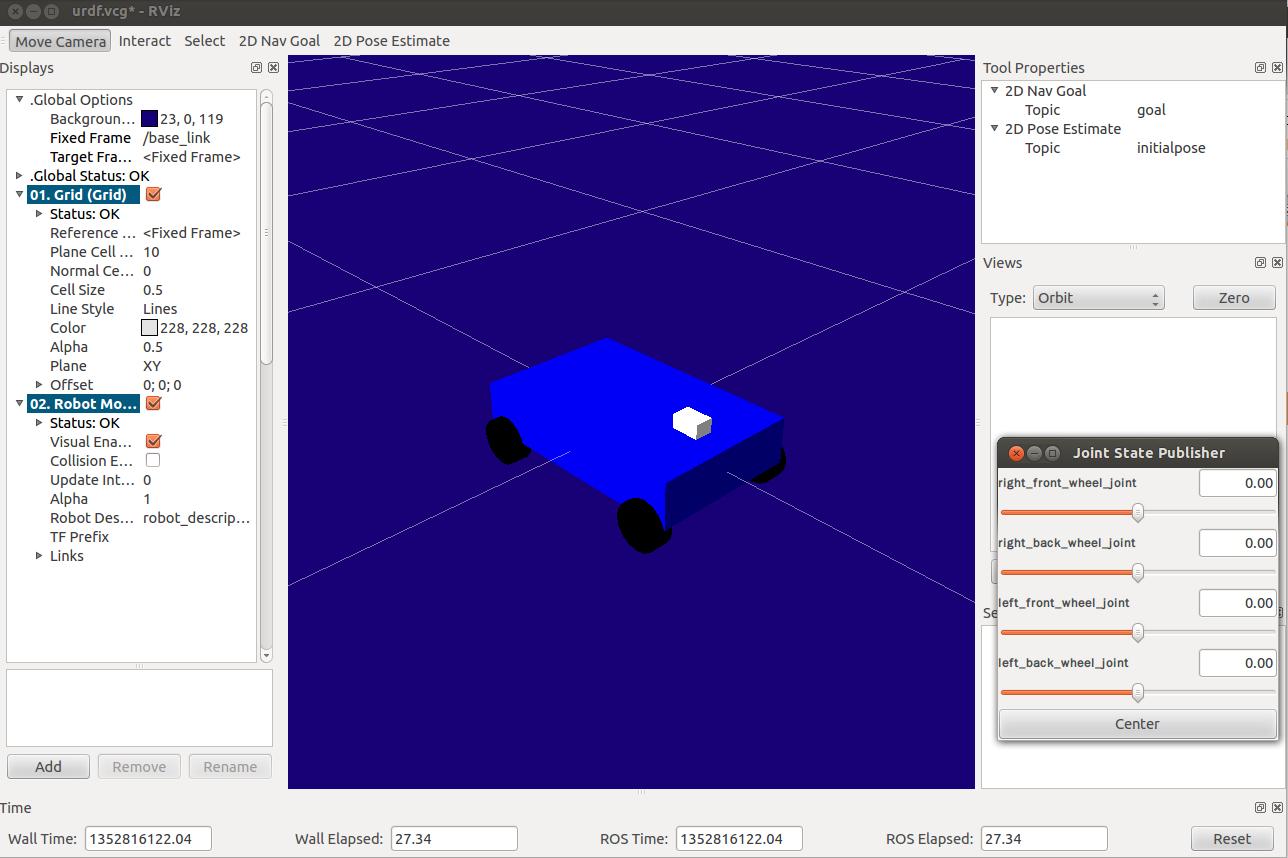
**四、建立launch命令文件**  
    在smartcar\_description文件夹下建立launch文件夹，创建智能车的描述文件 base.urdf.rviz.launch,描述代码如下：

    01.<launch>  
    02.    <arg name="model" />  
    03.    <arg name="gui" default="False" />  
    04.    <param name="robot\_description" textfile="$（find smartcar\_description）/urdf/smartcar.urdf" />  
    05.    <param name="use\_gui" value="$（arg gui）"/>  
    06.    <node name="joint\_state\_publisher" pkg="joint\_state\_publisher" type="joint\_state\_publisher" ></node>  
    07.    <node name="robot\_state\_publisher" pkg="robot\_state\_publisher" type="state\_publisher" />  
    08.    <node name="rviz" pkg="rviz" type="rviz" args="-d $（find urdf\_tutorial）/urdf.vcg" />  
    09.</launch>

    五、效果演示  
    在终端中输入显示命令：

    01.roslaunch  smartcar\_description  base.urdf.rviz.launch  gui:=true

    显示效果如下图所示，使用gui中的控制bar可以控制四个轮子单独旋转。



**操作系统探索总结（六）使用smartcar进行仿真**

　　之前的博客中，我们使用rviz进行了TurtleBot的仿真，而且使用urdf文件建立了自己的机器人smartcar，本篇博客是将两者进行结合，使用smartcar机器人在rviz中进行仿真。

**一、模型完善**

　　之前我们使用的都是urdf文件格式的模型，在很多情况下，ROS对urdf文件的支持并不是很好，使用宏定义的.xacro文件兼容性更好，扩展性也更好。所以我们把之前的urdf文件重新整理编写成.xacro文件。

**.xacro文件主要分为三部分：**

　　1、机器人主体

    01.<?xml version="1.0"?>  
    02.<robot name="smartcar" xmlns:xacro="<http://ros.org/wiki/xacro>">  
    03.  <property name="M\_PI" value="3.14159"/>  
    04.  
    05.  <!-- Macro for SmartCar body. Including Gazebo extensions, but does not include Kinect -->  
    06.  <include filename="$（find smartcar\_description）/urdf/gazebo.urdf.xacro"/>  
    07.  
    08.  <property name="base\_x" value="0.33" />  
    09.  <property name="base\_y" value="0.33" />  
    10.  
    11.  <xacro:macro name="smartcar\_body">  
    12.  
    13.  
    14.    <link name="base\_link">  
    15.    <inertial>  
    16.      <origin xyz="0 0 0.055"/>  
    17.      <mass value="1.0" />  
    18.      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>  
    19.    </inertial>  
    20.    <visual>  
    21.      <geometry>  
    22.        <box size="0.25 .16 .05"/>  
    23.      </geometry>  
    24.      <origin rpy="0 0 0" xyz="0 0 0.055"/>  
    25.      <material name="blue">  
    26.      <color rgba="0 0 .8 1"/>  
    27.      </material>  
    28.   </visual>  
    29.   <collision>  
    30.      <origin rpy="0 0 0" xyz="0 0 0.055"/>  
    31.      <geometry>  
    32.        <box size="0.25 .16 .05" />  
    33.      </geometry>  
    34.    </collision>  
    35.  </link>  
    36.  
    37.  
    38. <link name="left\_front\_wheel">  
    39.    <inertial>  
    40.      <origin  xyz="0.08 0.08 0.025"/>  
    41.      <mass value="0.1" />  
    42.       <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>  
    43.    </inertial>  
    44.    <visual>  
    45.      <geometry>  
    46.        <cylinder length=".02" radius="0.025"/>  
    47.      </geometry>  
    48.      <material name="black">  
    49.        <color rgba="0 0 0 1"/>  
    50.      </material>  
    51.    </visual>  
    52.    <collision>  
    53.      <origin rpy="0 1.57075 1.57075" xyz="0.08 0.08 0.025"/>  
    54.      <geometry>  
    55.         <cylinder length=".02" radius="0.025"/>  
    56.      </geometry>  
    57.    </collision>  
    58.  </link>  
    59.  
    60.  <joint name="left\_front\_wheel\_joint" type="continuous">  
    61.    <axis xyz="0 0 1"/>  
    62.    <parent link="base\_link"/>  
    63.    <child link="left\_front\_wheel"/>  
    64.    <origin rpy="0 1.57075 1.57075" xyz="0.08 0.08 0.025"/>  
    65.    <limit effort="100" velocity="100"/>  
    66.    <joint\_properties damping="0.0" friction="0.0"/>  
    67.  </joint>  
    68.  
    69.  <link name="right\_front\_wheel">  
    70.    <inertial>  
    71.      <origin xyz="0.08 -0.08 0.025"/>  
    72.      <mass value="0.1" />  
    73.       <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>  
    74.    </inertial>  
    75.    <visual>  
    76.      <geometry>  
    77.        <cylinder length=".02" radius="0.025"/>  
    78.      </geometry>  
    79.      <material name="black">  
    80.        <color rgba="0 0 0 1"/>  
    81.      </material>  
    82.    </visual>  
    83.    <collision>  
    84.      <origin rpy="0 1.57075 1.57075" xyz="0.08 -0.08 0.025"/>  
    85.      <geometry>  
    86.         <cylinder length=".02" radius="0.025"/>  
    87.      </geometry>  
    88.    </collision>  
    89.  </link>  
    90.  
    91.  <joint name="right\_front\_wheel\_joint" type="continuous">  
    92.    <axis xyz="0 0 1"/>  
    93.    <parent link="base\_link"/>  
    94.    <child link="right\_front\_wheel"/>  
    95.    <origin rpy="0 1.57075 1.57075" xyz="0.08 -0.08 0.025"/>  
    96.    <limit effort="100" velocity="100"/>  
    97.    <joint\_properties damping="0.0" friction="0.0"/>  
    98. </joint>  
    99.  
    100. <link name="left\_back\_wheel">  
    101.    <inertial>  
    102.      <origin xyz="-0.08 0.08 0.025"/>  
    103.      <mass value="0.1" />  
    104.       <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>  
    105.    </inertial>  
    106.    <visual>  
    107.      <geometry>  
    108.        <cylinder length=".02" radius="0.025"/>  
    109.      </geometry>  
    110.      <material name="black">  
    111.        <color rgba="0 0 0 1"/>  
    112.      </material>  
    113.   </visual>  
    114.   <collision>  
    115.       <origin rpy="0 1.57075 1.57075" xyz="-0.08 0.08 0.025"/>  
    116.      <geometry>  
    117.         <cylinder length=".02" radius="0.025"/>  
    118.      </geometry>  
    119.    </collision>  
    120.  </link>  
    121.  
    122.  <joint name="left\_back\_wheel\_joint" type="continuous">  
    123.    <axis xyz="0 0 1"/>  
    124.    <parent link="base\_link"/>  
    125.    <child link="left\_back\_wheel"/>  
    126.    <origin rpy="0 1.57075 1.57075" xyz="-0.08 0.08 0.025"/>  
    127.    <limit effort="100" velocity="100"/>  
    128.    <joint\_properties damping="0.0" friction="0.0"/>  
    129.  </joint>  
    130.  
    131.  <link name="right\_back\_wheel">  
    132.    <inertial>  
    133.       <origin xyz="-0.08 -0.08 0.025"/>  
    134.       <mass value="0.1" />  
    135.       <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>  
    136.    </inertial>  
    137.    <visual>  
    138.      <geometry>  
    139.        <cylinder length=".02" radius="0.025"/>  
    140.      </geometry>  
    141.      <material name="black">  
    142.        <color rgba="0 0 0 1"/>  
    143.      </material>  
    144.   </visual>  
    145.   <collision>  
    146.      <origin rpy="0 1.57075 1.57075" xyz="-0.08 -0.08 0.025"/>  
    147.      <geometry>  
    148.         <cylinder length=".02" radius="0.025"/>  
    149.      </geometry>  
    150.    </collision>  
    151.  </link>  
    152.  
    153.  
    154.  <joint name="right\_back\_wheel\_joint" type="continuous">  
    155.    <axis xyz="0 0 1"/>  
    156.    <parent link="base\_link"/>  
    157.    <child link="right\_back\_wheel"/>  
    158.    <origin rpy="0 1.57075 1.57075" xyz="-0.08 -0.08 0.025"/>  
    159.    <limit effort="100" velocity="100"/>  
    160.    <joint\_properties damping="0.0" friction="0.0"/>  
    161.  </joint>  
    162.  
    163.  <link name="head">  
    164.    <inertial>  
    165.      <origin xyz="0.08 0 0.08"/>  
    166.      <mass value="0.1" />  
    167.      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>  
    168.    </inertial>  
    169.    <visual>  
    170.      <geometry>  
    171.        <box size=".02 .03 .03"/>  
    172.      </geometry>  
    173.      <material name="white">  
    174.        <color rgba="1 1 1 1"/>  
    175.      </material>  
    176.     </visual>  
    177.     <collision>  
    178.      <origin xyz="0.08 0 0.08"/>  
    179.      <geometry>  
    180.         <cylinder length=".02" radius="0.025"/>  
    181.      </geometry>  
    182.    </collision>  
    183.  </link>  
    184.  
    185.  <joint name="tobox" type="fixed">  
    186.    <parent link="base\_link"/>  
    187.    <child link="head"/>  
    188.    <origin xyz="0.08 0 0.08"/>  
    189.  </joint>  
    190.  </xacro:macro>  
    191.  
    192.</robot>

　2、gazebo属性部分

    01.<?xml version="1.0"?>  
    02.  
    03.<robot xmlns:controller="<http://playerstage.sourceforge.net/gazebo/xmlschema/#controller>"  
    04.    xmlns:interface="<http://playerstage.sourceforge.net/gazebo/xmlschema/#interface>"  
    05.    xmlns:sensor="<http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor>"  
    06.    xmlns:xacro="<http://ros.org/wiki/xacro>"  
    07.    name="smartcar\_gazebo">  
    08.  
    09.<!-- ASUS Xtion PRO camera for simulation -->  
    10.<!-- gazebo\_ros\_wge100 plugin is in kt2\_gazebo\_plugins package -->  
    11.<xacro:macro name="smartcar\_sim">  
    12.    <gazebo reference="base\_link">  
    13.        <material>Gazebo/Blue</material>  
    14.    </gazebo>  
    15.  
    16.    <gazebo reference="right\_front\_wheel">  
    17.        <material>Gazebo/FlatBlack</material>  
    18.    </gazebo>  
    19.  
    20.    <gazebo reference="right\_back\_wheel">  
    21.        <material>Gazebo/FlatBlack</material>  
    22.    </gazebo>  
    23.  
    24.    <gazebo reference="left\_front\_wheel">  
    25.        <material>Gazebo/FlatBlack</material>  
    26.    </gazebo>  
    27.  
    28.    <gazebo reference="left\_back\_wheel">  
    29.        <material>Gazebo/FlatBlack</material>  
    30.    </gazebo>  
    31.  
    32.    <gazebo reference="head">  
    33.        <material>Gazebo/White</material>  
    34.    </gazebo>  
    35.  
    36.</xacro:macro>  
    37.  
    38.</robot>

　　3、主文件

    01.<SPAN style="FONT-SIZE: 14px"><?xml version="1.0"?>  
    02.  
    03.<robot name="smartcar"  
    04.    xmlns:xi="<http://www.w3.org/2001/XInclude>"  
    05.    xmlns:gazebo="<http://playerstage.sourceforge.net/gazebo/xmlschema/#gz>"  
    06.    xmlns:model="<http://playerstage.sourceforge.net/gazebo/xmlschema/#model>"  
    07.    xmlns:sensor="<http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor>"  
    08.    xmlns:body="<http://playerstage.sourceforge.net/gazebo/xmlschema/#body>"  
    09.    xmlns:geom="<http://playerstage.sourceforge.net/gazebo/xmlschema/#geom>"  
    10.    xmlns:joint="<http://playerstage.sourceforge.net/gazebo/xmlschema/#joint>"  
    11.    xmlns:controller="<http://playerstage.sourceforge.net/gazebo/xmlschema/#controller>"  
    12.    xmlns:interface="<http://playerstage.sourceforge.net/gazebo/xmlschema/#interface>"  
    13.    xmlns:rendering="<http://playerstage.sourceforge.net/gazebo/xmlschema/#rendering>"  
    14.    xmlns:renderable="<http://playerstage.sourceforge.net/gazebo/xmlschema/#renderable>"  
    15.    xmlns:physics="<http://playerstage.sourceforge.net/gazebo/xmlschema/#physics>"  
    16.    xmlns:xacro="<http://ros.org/wiki/xacro>">  
    17.  
    18.  <include filename="$（find smartcar\_description）/urdf/smartcar\_body.urdf.xacro" />  
    19.  
    20.  <!-- Body of SmartCar, with plates, standoffs and Create （including sim sensors） -->  
    21.  <smartcar\_body/>  
    22.  
    23.  <smartcar\_sim/>  
    24.  
    25.</robot></SPAN>

**二、lanuch文件**

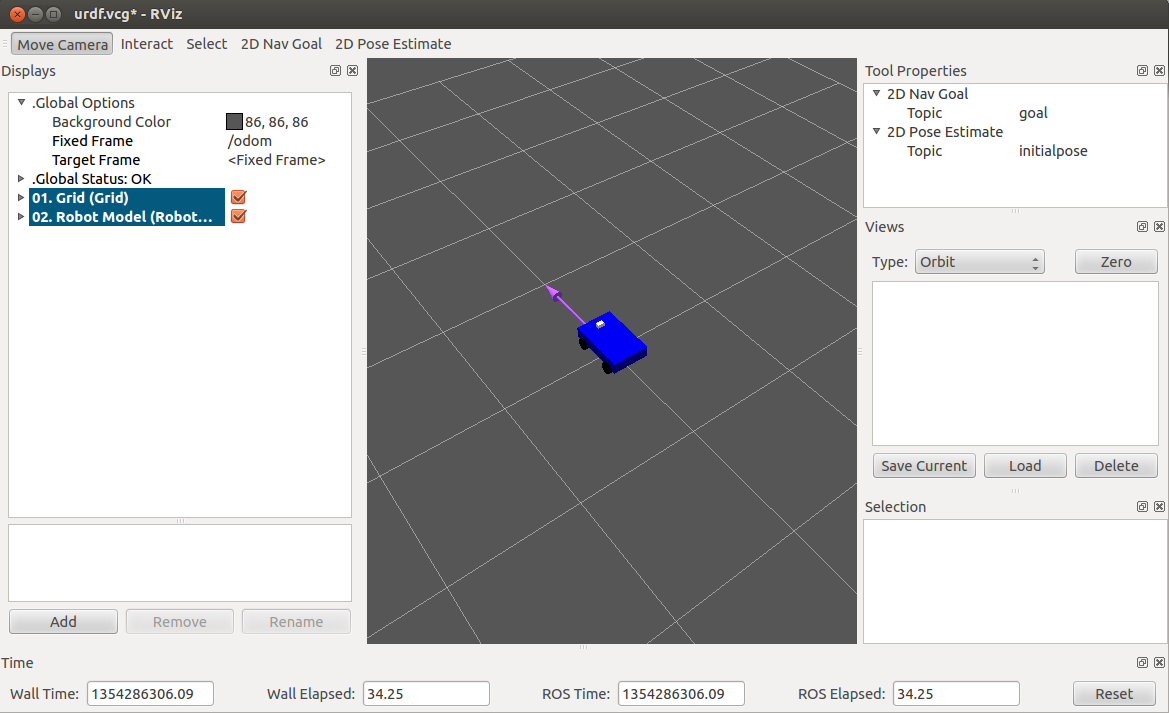
　　在launch文件中要启动节点和模拟器。

    01.<launch>  
    02.    <param name="/use\_sim\_time" value="false" />  
    03.  
    04.    <!-- Load the URDF/Xacro model of our robot -->  
    05.    <arg name="urdf\_file" default="$（find xacro）/xacro.py '$（find smartcar\_description）/urdf/smartcar.urdf.xacro'" />  
    06.    <arg name="gui" default="false" />  
    07.  
    08.    <param name="robot\_description" command="$（arg urdf\_file）" />  
    09.    <param name="use\_gui" value="$（arg gui）"/>  
    10.  
    11.    <node name="arbotix" pkg="arbotix\_python" type="driver.py" output="screen">  
    12.        <rosparam file="$（find smartcar\_description）/config/smartcar\_arbotix.yaml" command="load" />  
    13.        <param name="sim" value="true"/>  
    14.    </node>  
    15.  
    16.    <node name="joint\_state\_publisher" pkg="joint\_state\_publisher" type="joint\_state\_publisher" >  
    17.    </node>  
    18.  
    19.    <node name="robot\_state\_publisher" pkg="robot\_state\_publisher" type="state\_publisher">  
    20.        <param name="publish\_frequency" type="double" value="20.0" />  
    21.    </node>  
    22.  
    23.     <!-- We need a static transforms for the wheels -->  
    24.    <node pkg="tf" type="static\_transform\_publisher" name="odom\_left\_wheel\_broadcaster" args="0 0 0 0 0 0 /base\_link /left\_front\_link 100" />  
    25.    <node pkg="tf" type="static\_transform\_publisher" name="odom\_right\_wheel\_broadcaster" args="0 0 0 0 0 0 /base\_link /right\_front\_link 100" />  
    26.  
    27.    <node name="rviz" pkg="rviz" type="rviz" args="-d $（find smartcar\_description）/urdf.vcg" />  
    28.</launch>

**三、仿真**[**测试**](http://softtest.chinaitlab.com/)

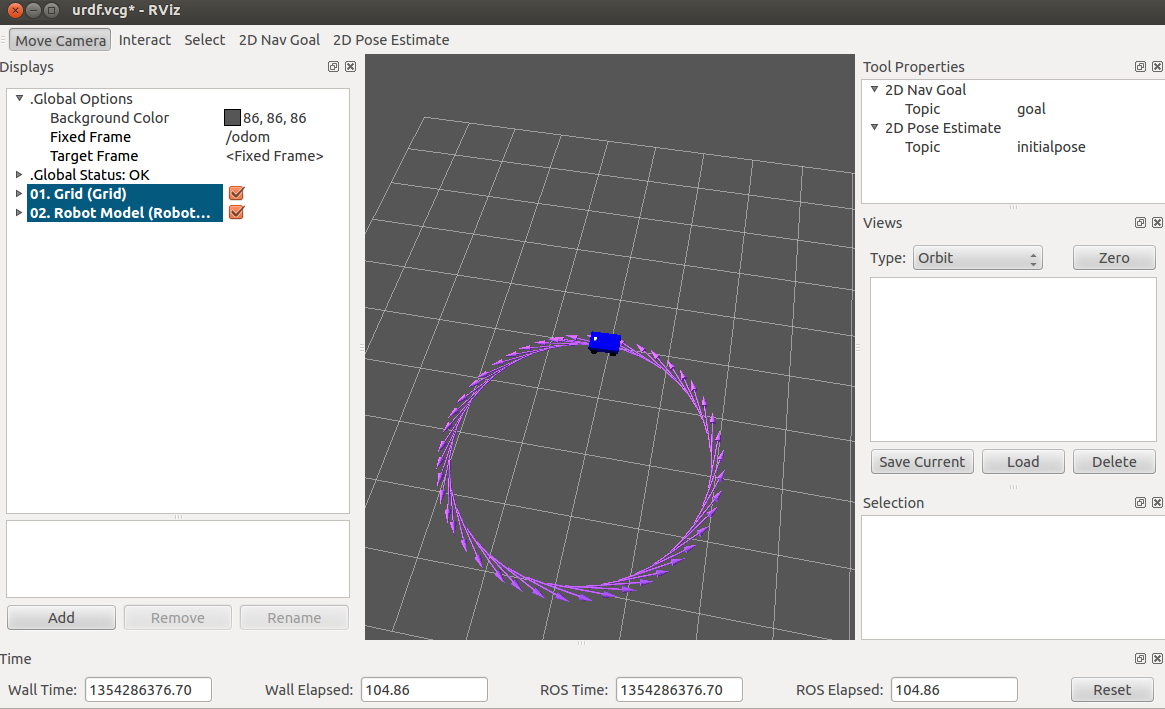
　　首先运行lanuch，既可以看到rviz中的机器人：

    01.roslaunch smartcar\_description smartcar\_display.rviz.launch

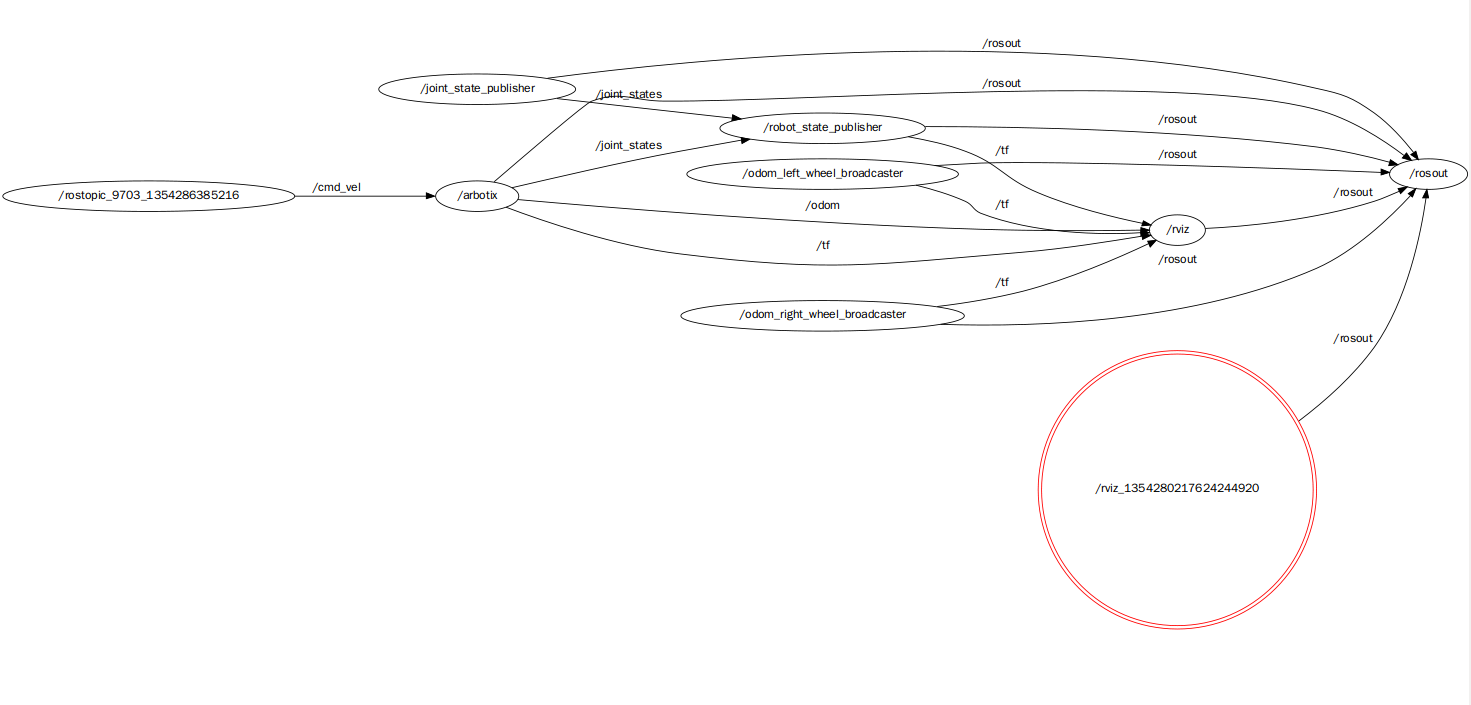


　　发布一条动作的消息。

    01.rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.5, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0.5}}'



**四、节点关系**



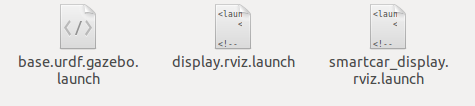
**操作系统探索总结（七）smartcar源码上传**

　　看到前面写的文章还是帮助了很多ROS的学习者，我感到非常荣幸。其实我也是一名ROS的新手，ROS的相关资料少，上手难度大，我现在也在摸索着学习，还希望大家都将自己的学习成果在网上或者ROS群里分享。

　　代码的内容主要是前边用到的机器人smartcar的urdf文件，已经运行使用的launch文件，大家[下载](http://download.chinaitlab.com/)后可以直接编译使用，由于代码在我的计算机上编译时带有路径信息，所以大家编译的时候请使用下面的命令先清除再编译：

　　01 rosmake smartcar\_description --pre-clean

　　代码中的urdf文件里室机器人的urdf描述，和上一篇博客中讲的是一样的，launch文件夹中主要有以下文件：

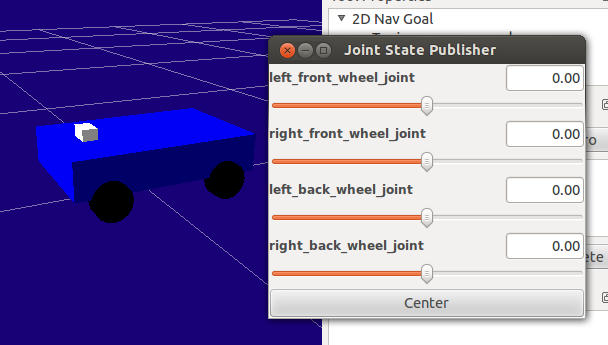


　　第一个实在gazebo仿真中用到的，需要先运行gazebo，然后再运行。但是貌似有问题，我每次加载到gazebo中的时候似乎物理参数不对，机器人在gazebo里飞来飞去的，至今还没有解决。如果哪位仁兄发现问题了，还请多多指教。

　　第二个文件是我在绘制模型的时候用到的，主要功能是在rviz中显示机器人模型，文件中的以下代码：

　　01.<arg name="gui" default="false" />

　　如果把false改为true，则会打开一个gui的调试界面，可以通过滑动条旋转车轮，如下图所示。默认是false。



　　第三个文件是在方针的时候用到的，就是前面博客中使用的，会打开arbotix仿真器。

　　根目录中还有rviz的配置文件urdf.vcg，只要大家在rviz中修改界面或者参数，推出的时候保存，就会自动修改这个文件了。

config文件夹下的是仿真用到的配置文件。

**操作系统探索总结（八）键盘控制**

　　如果尝试过前面的例子，有没有感觉每次让机器人移动还要在终端里输入指令，这也太麻烦了，有没有办法通过键盘来控制机器人的移动呢?答案室当然的了。我研究了其他几个机器人键盘控制的代码，还是有所收获的，最后移植到了smartcar上，实验成功。

**一、创建控制包**

　　首先，我们为键盘控制单独建立一个包：

    01.roscreate-pkg smartcar\_teleop rospy geometry\_msgs std\_msgs roscpp  
    02.rosmake

**二、简单的消息发布**

　　在机器人仿真中，主要控制机器人移动的就是 在机器人仿真中，主要控制机器人移动的就是Twist()结构，如果我们编程将这个结构通过程序发布成topic，自然就可以控制机器人了。我们先用简单的python来尝试一下。

　　之前的模拟中，我们使用的都是在命令行下进行的消息发布，现在我们需要把这些命令转换成python代码，封装到一个单独的节点中去。针对之前的命令行，我们可以很简单的在smartcar\_teleop /scripts文件夹下编写如下的控制代码：

    01.#!/usr/bin/env python  
    02.import roslib; roslib.load\_manifest（'smartcar\_teleop'）  
    03.import rospy  
    04.from geometry\_msgs.msg import Twist  
    05.from std\_msgs.msg import String  
    06.  
    07.class Teleop:  
    08.    def \_\_init\_\_（self）：  
    09.        pub = rospy.Publisher（'cmd\_vel', Twist）  
    10.        rospy.init\_node（'smartcar\_teleop'）  
    11.        rate = rospy.Rate（rospy.get\_param（'~hz', 1））  
    12.        self.cmd = None  
    13.  
    14.        cmd = Twist（）  
    15.        cmd.linear.x = 0.2  
    16.        cmd.linear.y = 0  
    17.        cmd.linear.z = 0  
    18.        cmd.angular.z = 0  
    19.        cmd.angular.z = 0  
    20.        cmd.angular.z = 0.5  
    21.  
    22.        self.cmd = cmd  
    23.        while not rospy.is\_shutdown（）：  
    24.            str = "hello world %s" % rospy.get\_time（）  
    25.            rospy.loginfo（str）  
    26.            pub.publish（self.cmd）  
    27.            rate.sleep（）  
    28.  
    29.if \_\_name\_\_ == '\_\_main\_\_':Teleop（）

　　python代码在ROS重视不需要编译的。先运行之前教程中用到的smartcar机器人，在rviz中进行显示，然后新建终端，输入如下命令：

    01.rosrun smartcar\_teleop teleop.py

　　也可以建立一个launch文件运行：

    01.<launch>  
    02.  <arg name="cmd\_topic" default="cmd\_vel" />  
    03.  <node pkg="smartcar\_teleop" type="teleop.py" name="smartcar\_teleop">  
    04.    <remap from="cmd\_vel" to="$（arg cmd\_topic）" />  
    05.  </node>  
    06.</launch>

　　在rviz中看一下机器人是不是动起来了!

**三、加入键盘控制**

　　当然前边的程序不是我们要的，我们需要的键盘控制。

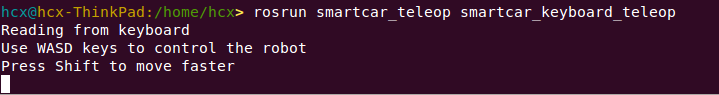
　　1、移植

　　因为ROS的代码具有很强的可移植性，所以用键盘控制的代码其实可以直接从其他机器人包中移植过来，在这里我主要参考的是erratic\_robot，在这个机器人的代码中有一个erratic\_teleop包，可以直接移植过来使用。

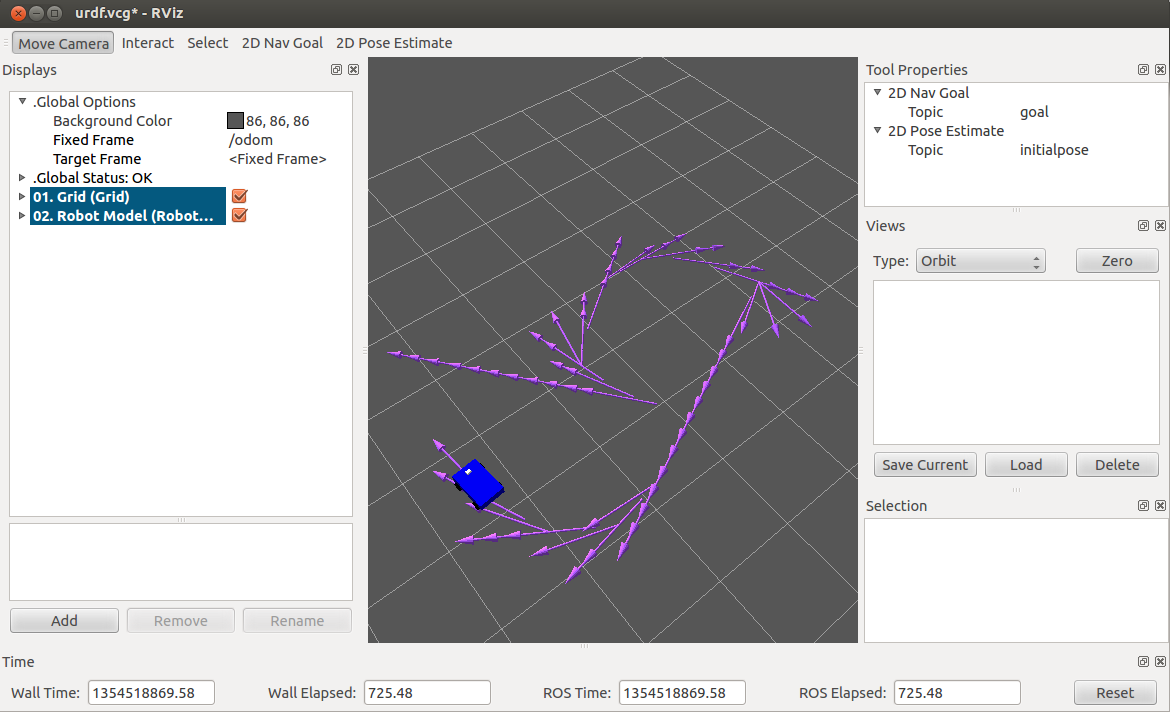
　　首先，我们将其中src文件夹下的keyboard.cpp代码文件直接拷贝到我们smartcar\_teleop包的src文件夹下，然后修改CMakeLists.txt文件，将下列代码加入文件底部：

    01.rosbuild\_add\_boost\_directories（）  
    02.rosbuild\_add\_executable（smartcar\_keyboard\_teleop src/keyboard.cpp）  
    03.target\_link\_libraries（smartcar\_keyboard\_teleop boost\_thread）

　　编译完成后，运行smartcar模型。重新打开一个终端，打开键盘控制节点：



　　在终端中按下键盘里的“W”、“S”、“D”、“A”以及“Shift”键进行机器人的控制。效果如下图：



　　2、复用

　　因为代码是我们直接复制过来的，其中有很多与之前erratic机器人相关的变量，我们把代码稍作修改，变成自己机器人可读性较强的代码。

    01.#include <termios.h>  
    02.#include <signal.h>  
    03.#include <math.h>  
    04.#include <stdio.h>  
    05.#include <stdlib.h>  
    06.#include <sys/poll.h>  
    07.  
    08.#include <boost/thread/thread.hpp>  
    09.#include <ros/ros.h>  
    10.#include <geometry\_msgs/Twist.h>  
    11.  
    12.#define KEYCODE\_W 0x77  
    13.#define KEYCODE\_A 0x61  
    14.#define KEYCODE\_S 0x73  
    15.#define KEYCODE\_D 0x64  
    16.  
    17.#define KEYCODE\_A\_CAP 0x41  
    18.#define KEYCODE\_D\_CAP 0x44  
    19.#define KEYCODE\_S\_CAP 0x53  
    20.#define KEYCODE\_W\_CAP 0x57  
    21.  
    22.class SmartCarKeyboardTeleopNode  
    23.{  
    24.    private:  
    25.        double walk\_vel\_;  
    26.        double run\_vel\_;  
    27.        double yaw\_rate\_;  
    28.        double yaw\_rate\_run\_;  
    29.  
    30.        geometry\_msgs::Twist cmdvel\_;  
    31.        ros::NodeHandle n\_;  
    32.        ros::Publisher pub\_;  
    33.  
    34.    public:  
    35.        SmartCarKeyboardTeleopNode（）  
    36.        {  
    37.            pub\_ = n\_.advertise<geometry\_msgs::Twist>（"cmd\_vel", 1）；  
    38.  
    39.            ros::NodeHandle n\_private（"~"）；  
    40.            n\_private.param（"walk\_vel", walk\_vel\_, 0.5）；  
    41.            n\_private.param（"run\_vel", run\_vel\_, 1.0）；  
    42.            n\_private.param（"yaw\_rate", yaw\_rate\_, 1.0）；  
    43.            n\_private.param（"yaw\_rate\_run", yaw\_rate\_run\_, 1.5）；  
    44.        }  
    45.  
    46.        ~SmartCarKeyboardTeleopNode（） { }  
    47.        void keyboardLoop（）；  
    48.  
    49.        void stopRobot（）  
    50.        {  
    51.            cmdvel\_.linear.x = 0.0;  
    52.            cmdvel\_.angular.z = 0.0;  
    53.            pub\_.publish（cmdvel\_）；  
    54.        }  
    55.};  
    56.  
    57.SmartCarKeyboardTeleopNode\* tbk;  
    58.int kfd = 0;  
    59.struct termios cooked, raw;  
    60.bool done;  
    61.  
    62.int main（int argc, char\*\* argv）  
    63.{  
    64.    ros::init（argc,argv,"tbk", ros::init\_options::AnonymousName | ros::init\_options::NoSigintHandler）；  
    65.    SmartCarKeyboardTeleopNode tbk;  
    66.  
    67.    boost::thread t = boost::thread（boost::bind（&SmartCarKeyboardTeleopNode::keyboardLoop, &tbk））；  
    68.  
    69.    ros::spin（）；  
    70.  
    71.    t.interrupt（）；  
    72.    t.join（）；  
    73.    tbk.stopRobot（）；  
    74.    tcsetattr（kfd, TCSANOW, &cooked）；  
    75.  
    76.    return（0）；  
    77.}  
    78.  
    79.void SmartCarKeyboardTeleopNode::keyboardLoop（）  
    80.{  
    81.    char c;  
    82.    double max\_tv = walk\_vel\_;  
    83.    double max\_rv = yaw\_rate\_;  
    84.    bool dirty = false;  
    85.    int speed = 0;  
    86.    int turn = 0;  
    87.  
    88.    // get the console in raw mode  
    89.    tcgetattr（kfd, &cooked）；  
    90.    memcpy（&raw, &cooked, sizeof（struct termios））；  
    91.    raw.c\_lflag &=~ （ICANON | ECHO）；  
    92.    raw.c\_cc[VEOL] = 1;  
    93.    raw.c\_cc[VEOF] = 2;  
    94.    tcsetattr（kfd, TCSANOW, &raw）；  
    95.  
    96.    puts（"Reading from keyboard"）；  
    97.    puts（"Use WASD keys to control the robot"）；  
    98.    puts（"Press Shift to move faster"）；  
    99.  
    100.    struct pollfd ufd;  
    101.    ufd.fd = kfd;  
    102.    ufd.events = POLLIN;  
    103.  
    104.    for（；；）  
    105.    {  
    106.        boost::this\_thread::interruption\_point（）；  
    107.  
    108.        // get the next event from the keyboard  
    109.        int num;  
    110.  
    111.        if （（num = poll（&ufd, 1, 250）） < 0）  
    112.        {  
    113.            perror（"poll（）："）；  
    114.            return;  
    115.        }  
    116.        else if（num > 0）  
    117.        {  
    118.            if（read（kfd, &c, 1） < 0）  
    119.            {  
    120.                perror（"read（）："）；  
    121.                return;  
    122.            }  
    123.        }  
    124.        else  
    125.        {  
    126.            if （dirty == true）  
    127.            {  
    128.                stopRobot（）；  
    129.                dirty = false;  
    130.            }  
    131.  
    132.            continue;  
    133.        }  
    134.  
    135.        switch（c）  
    136.        {  
    137.            case KEYCODE\_W:  
    138.                max\_tv = walk\_vel\_;  
    139.                speed = 1;  
    140.                turn = 0;  
    141.                dirty = true;  
    142.                break;  
    143.            case KEYCODE\_S:  
    144.                max\_tv = walk\_vel\_;  
    145.                speed = -1;  
    146.                turn = 0;  
    147.                dirty = true;  
    148.                break;  
    149.            case KEYCODE\_A:  
    150.                max\_rv = yaw\_rate\_;  
    151.                speed = 0;  
    152.                turn = 1;  
    153.                dirty = true;  
    154.                break;  
    155.            case KEYCODE\_D:  
    156.                max\_rv = yaw\_rate\_;  
    157.                speed = 0;  
    158.                turn = -1;  
    159.                dirty = true;  
    160.                break;  
    161.  
    162.            case KEYCODE\_W\_CAP:  
    163.                max\_tv = run\_vel\_;  
    164.                speed = 1;  
    165.                turn = 0;  
    166.                dirty = true;  
    167.                break;  
    168.            case KEYCODE\_S\_CAP:  
    169.                max\_tv = run\_vel\_;  
    170.                speed = -1;  
    171.                turn = 0;  
    172.                dirty = true;  
    173.                break;  
    174.            case KEYCODE\_A\_CAP:  
    175.                max\_rv = yaw\_rate\_run\_;  
    176.                speed = 0;  
    177.                turn = 1;  
    178.                dirty = true;  
    179.                break;  
    180.            case KEYCODE\_D\_CAP:  
    181.                max\_rv = yaw\_rate\_run\_;  
    182.                speed = 0;  
    183.                turn = -1;  
    184.                dirty = true;  
    185.                break;  
    186.            default:  
    187.                max\_tv = walk\_vel\_;  
    188.                max\_rv = yaw\_rate\_;  
    189.                speed = 0;  
    190.                turn = 0;  
    191.                dirty = false;  
    192.        }  
    193.        cmdvel\_.linear.x = speed \* max\_tv;  
    194.        cmdvel\_.angular.z = turn \* max\_rv;  
    195.        pub\_.publish（cmdvel\_）；  
    196.    }  
    197.}

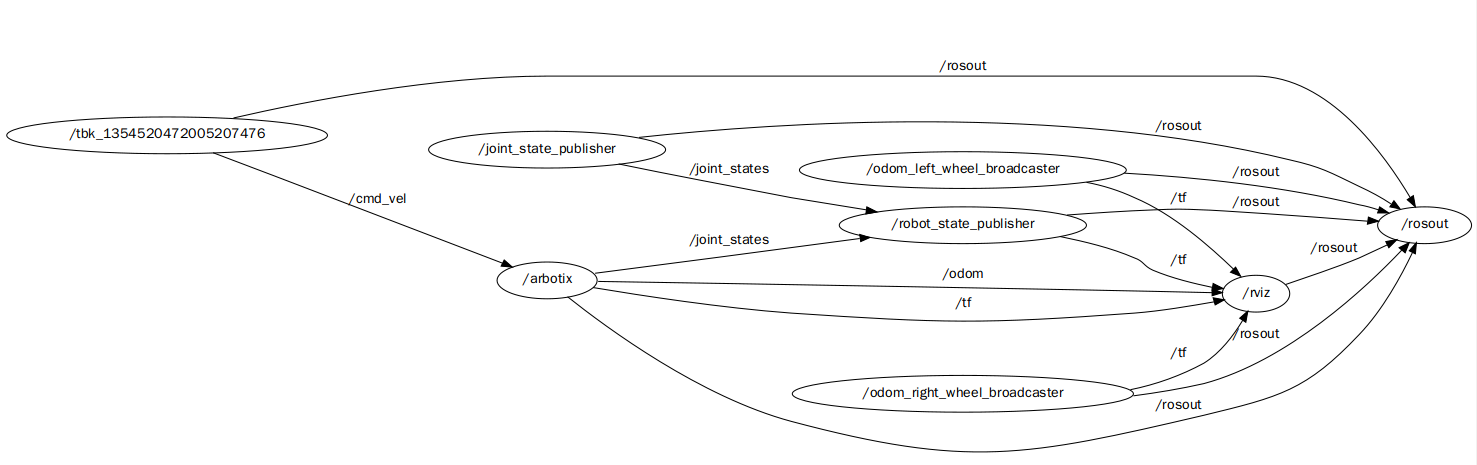
　　3、创新

　　虽然很多机器人的键盘控制使用的都是[C++](http://c.chinaitlab.com/)编写的代码，但是考虑到python的强大，我们还是需要尝试使用python来编写程序。

　　首先需要理解上面[C++](http://c.chinaitlab.com/)程序的流程。在上面的程序中，我们单独创建了一个线程来读取中断中的输入，然后根据输入发布不同的速度和角度消息。介于线程的概念还比较薄弱，在python中使用循环替代线程。然后需要考虑的只是如何使用python来处理中断中的输入字符，通过上网查找资料，发现使用的API和C++的基本是一致的。最终的程序如下：

    01.#!/usr/bin/env python  
    02.# -\*- coding: utf-8 -\*  
    03.  
    04.import  os  
    05.import  sys  
    06.import  tty, termios  
    07.import roslib; roslib.load\_manifest（'smartcar\_teleop'）  
    08.import rospy  
    09.from geometry\_msgs.msg import Twist  
    10.from std\_msgs.msg import String  
    11.  
    12.# 全局变量  
    13.cmd = Twist（）  
    14.pub = rospy.Publisher（'cmd\_vel', Twist）  
    15.  
    16.def keyboardLoop（）：  
    17.    #初始化  
    18.    rospy.init\_node（'smartcar\_teleop'）  
    19.    rate = rospy.Rate（rospy.get\_param（'~hz', 1））  
    20.  
    21.    #速度变量  
    22.    walk\_vel\_ = rospy.get\_param（'walk\_vel', 0.5）  
    23.    run\_vel\_ = rospy.get\_param（'run\_vel', 1.0）  
    24.    yaw\_rate\_ = rospy.get\_param（'yaw\_rate', 1.0）  
    25.    yaw\_rate\_run\_ = rospy.get\_param（'yaw\_rate\_run', 1.5）  
    26.  
    27.    max\_tv = walk\_vel\_  
    28.    max\_rv = yaw\_rate\_  
    29.  
    30.    #显示提示信息  
    31.    print "Reading from keyboard"  
    32.    print "Use WASD keys to control the robot"  
    33.    print "Press Caps to move faster"  
    34.    print "Press q to quit"  
    35.  
    36.    #读取按键循环  
    37.    while not rospy.is\_shutdown（）：  
    38.        fd = sys.stdin.fileno（）  
    39.        old\_settings = termios.tcgetattr（fd）  
    40.        #不产生回显效果  
    41.        old\_settings[3] = old\_settings[3] & ~termios.ICANON & ~termios.ECHO  
    42.        try :  
    43.            tty.setraw（ fd ）  
    44.            ch = sys.stdin.read（ 1 ）  
    45.        finally :  
    46.            termios.tcsetattr（fd, termios.TCSADRAIN, old\_settings）  
    47.  
    48.        if ch == 'w':  
    49.            max\_tv = walk\_vel\_  
    50.            speed = 1  
    51.            turn = 0  
    52.        elif ch == 's':  
    53.            max\_tv = walk\_vel\_  
    54.            speed = -1  
    55.            turn = 0  
    56.        elif ch == 'a':  
    57.            max\_rv = yaw\_rate\_  
    58.            speed = 0  
    59.            turn = 1  
    60.        elif ch == 'd':  
    61.            max\_rv = yaw\_rate\_  
    62.            speed = 0  
    63.            turn = -1  
    64.        elif ch == 'W':  
    65.            max\_tv = run\_vel\_  
    66.            speed = 1  
    67.            turn = 0  
    68.        elif ch == 'S':  
    69.            max\_tv = run\_vel\_  
    70.            speed = -1  
    71.            turn = 0  
    72.        elif ch == 'A':  
    73.            max\_rv = yaw\_rate\_run\_  
    74.            speed = 0  
    75.            turn = 1  
    76.        elif ch == 'D':  
    77.            max\_rv = yaw\_rate\_run\_  
    78.            speed = 0  
    79.            turn = -1  
    80.        elif ch == 'q':  
    81.            exit（）  
    82.        else:  
    83.            max\_tv = walk\_vel\_  
    84.            max\_rv = yaw\_rate\_  
    85.            speed = 0  
    86.            turn = 0  
    87.  
    88.        #发送消息  
    89.        cmd.linear.x = speed \* max\_tv;  
    90.        cmd.angular.z = turn \* max\_rv;  
    91.        pub.publish（cmd）  
    92.        rate.sleep（）  
    93.        #停止机器人  
    94.        stop\_robot（）；  
    95.  
    96.def stop\_robot（）：  
    97.    cmd.linear.x = 0.0  
    98.    cmd.angular.z = 0.0  
    99.    pub.publish（cmd）  
    100.  
    101.if \_\_name\_\_ == '\_\_main\_\_':  
    102.    try:  
    103.        keyboardLoop（）  
    104.    except rospy.ROSInterruptException:  
    105.        pass

**四、节点关系图**



**操作系统探索总结（九）操作杆控制**

　　对于移动机器人，键盘的控制往往满足不了我们的需求，以前看好多电影里边都是用一个摇杆来控制机器人的，简直帅爆了，正好我这里有一个操作杆，那就来尝试感受一下。

　　操作杆(joystick)控制会更加有操作感，ROS中的很多机器人也带有操作杆的相关代码，只需要简单的移植即可。我们使用的是赛钛客(saitek)的一款操作杆，如下图所示：



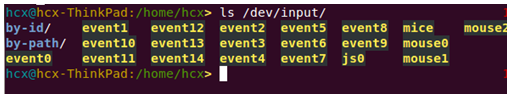
　　使用的移植代码是clearpath\_husky机器人中的python代码。

**一、**[**测试**](http://softtest.chinaitlab.com/)**操作杆驱动**

　　首先将操作杆的接口插入电脑，然后在终端中输入：

    01.ls /dev/input/

　　显示如下：



　　其中的js0就代表我们的操作杆。然后[测试](http://softtest.chinaitlab.com/)操作杆的操作是否有效，输入:

    01.sudo jstest /dev/input/js0

　　然后会在终端中显示操作杆的各个控制值的即时值，操作操作杆，如果每个按键和操作都有效，说明操作杆是正常的。最后在ROS中的节点里进行测试。打开joy节点：

    01.rosrun joy joy\_node

　　再打开一个窗口，输入下面命令，查看数据:

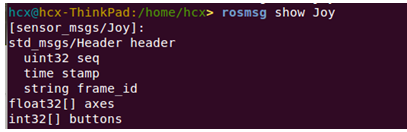
    01.rostopic echo joy

　　操作操作杆，窗口下面的数据就开始刷新。

**二、控制代码**

　　从上面的测试中，我们发现在前后左右摇动操作杆时，相应改变的数据是Axes中0号和1号位的数据，也是我们最常用的数据，其他按键对应的位置也可以找到，编程的时候就是利用的这些数据位置，所以一定要找到每个按键的对应编号。

　　ROS中已经为我们建立了操作杆的数据结构：



　　我们主要用到的就是axes和buttons数据。最终的代码如下：

    01.import roslib; roslib.load\_manifest（'smartcar\_teleop'）  
    02.import rospy  
    03.  
    04.from sensor\_msgs.msg import Joy  
    05.from geometry\_msgs.msg import Twist  
    06.from std\_msgs.msg import String  
    07.  
    08.class Teleop:  
    09.    def \_\_init\_\_（self）：  
    10.        rospy.init\_node（'smartcar\_teleop\_joy'）  
    11.  
    12.        self.turn\_scale = rospy.get\_param（'~turn\_scale'）  
    13.        self.drive\_scale = rospy.get\_param（'~drive\_scale'）  
    14.        self.deadman\_button = rospy.get\_param（'~deadman\_button', 0）  
    15.  
    16.        self.cmd = None  
    17.        cmd\_pub = rospy.Publisher（'cmd\_vel', Twist）  
    18.  
    19.        announce\_pub = rospy.Publisher（'/smartcar/announce/teleops',  
    20.                                       String, latch=True）  
    21.        announce\_pub.publish（rospy.get\_namespace（））；  
    22.  
    23.        rospy.Subscriber（"joy", Joy, self.callback）  
    24.        rate = rospy.Rate（rospy.get\_param（'~hz', 20））  
    25.  
    26.        while not rospy.is\_shutdown（）：  
    27.            rate.sleep（）  
    28.            if self.cmd:  
    29.                cmd\_pub.publish（self.cmd）  
    30.  
    31.    def callback（self, data）：  
    32.        """ Receive joystick data, formulate Twist message. """  
    33.        cmd = Twist（）  
    34.        cmd.linear.x = data.axes[1] \* self.drive\_scale  
    35.        cmd.angular.z = data.axes[0] \* self.turn\_scale  
    36.  
    37.        if data.buttons[self.deadman\_button] == 1:  
    38.            self.cmd = cmd  
    39.        else:  
    40.            self.cmd = None  
    41.  
    42.if \_\_name\_\_ == "\_\_main\_\_": Teleop（）

**三、机器人控制**

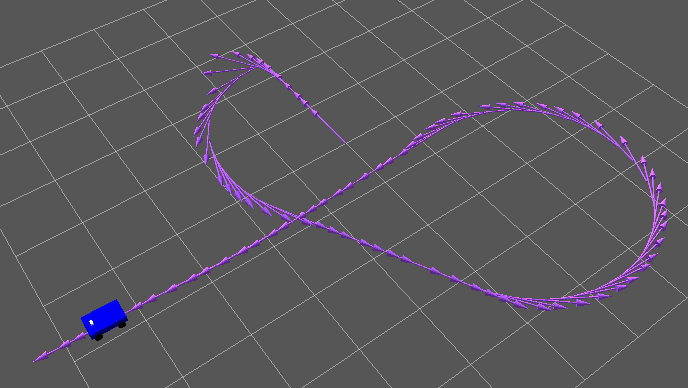
　　首先来创建一个launch文件(teleop\_joy.launch):

    01.<launch>  
    02.  <arg name="drive\_speed" default="1.0" />  
    03.  <arg name="turn\_speed" default="1.0" />  
    04.  <arg name="joy\_dev" default="/dev/input/js0" />  
    05.  <arg name="cmd\_topic" default="cmd\_vel" />  
    06.  
    07.  <node pkg="joy" type="joy\_node" name="joy\_node">  
    08.    <param name="dev" value="$（arg joy\_dev）" />  
    09.    <param name="deadzone" value="0.3" />  
    10.  </node>  
    11.  
    12.  <node pkg="smartcar\_teleop" type="teleop\_joy.py" name="smartcar\_teleop">  
    13.    <param name="turn\_scale" value="$（arg turn\_speed）" />  
    14.    <param name="drive\_scale" value="$（arg drive\_speed）" />  
    15.    <remap from="cmd\_vel" to="$（arg cmd\_topic）" />  
    16.  </node>  
    17.</launch>

　　在rviz中打开我们的机器人模型，然后打开操作杆的控制节点：

    01.roslaunch smartcar\_display.rviz.launch  
    02.roslaunch smartcar\_teleop teleop\_joy.launch

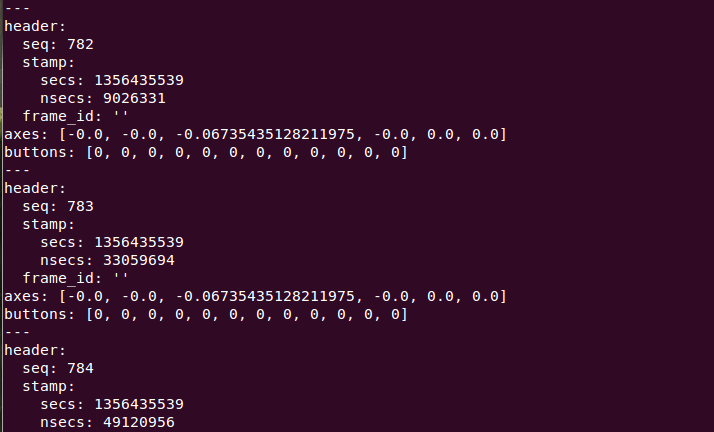
　　然后按住刹车键进行操作，机器人就可以开始移动了：



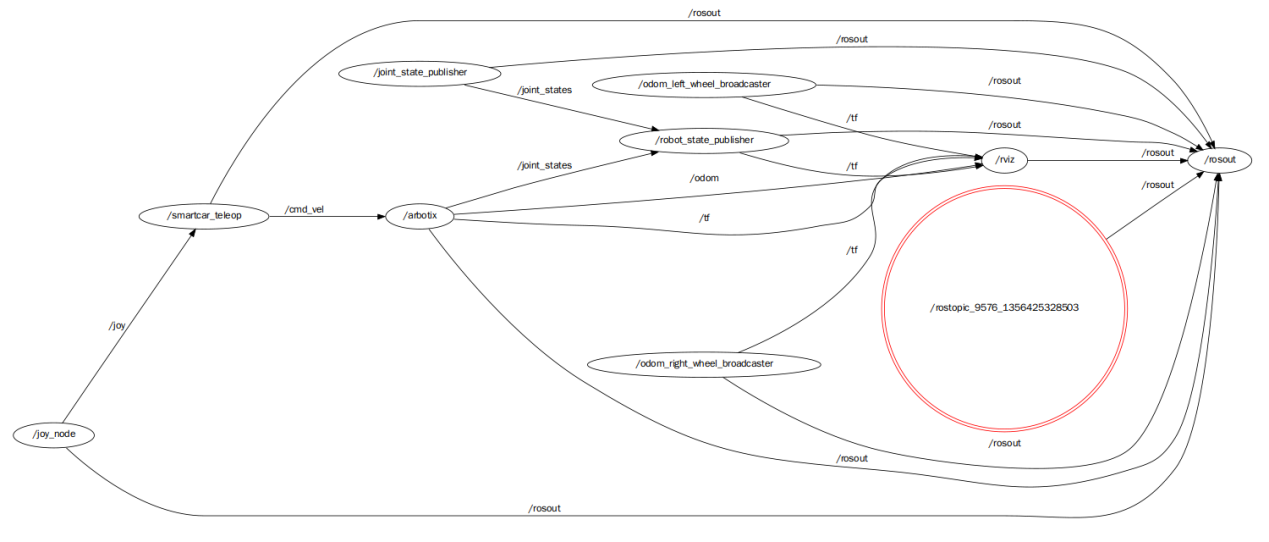
　在新终端中输入：

    01.rostopic echo joy

　　可以查看到实时的操作杆控制数据：



　　四、节点关系图



**操作系统探索总结（十）语音控制**

　　如今语音识别在PC机和智能手机上炒的火热，ROS走在技术的最前沿当然也不会错过这么帅的技术。ROS中使用了CMU Sphinx和Festival开源项目中的代码，发布了独立的语音识别包，而且可以将识别出来的语音转换成文字，然后让机器人智能处理后说话。

**一、语音识别包**

　　1、安装

　　安装很简单，直接使用ubuntu命令即可，首先安装依赖库：

    01.$ sudo apt-get install gstreamer0.10-pocketsphinx  
    02.$ sudo apt-get install ros-fuerte-audio-common  
    03.$ sudo apt-get install libasound2

　　然后来安装ROS包：

    01.$ svn checkout <http://albany-ros-pkg.googlecode.com/svn/trunk/rharmony>  
    02.$ rosmake pocketsphinx

　　其中的核心文件就是nodes文件夹下的recognizer.py文件了。这个文件通过麦克风收集语音信息，然后调用语音识别库进行识别生成文本信息，通过/recognizer/output消息发布，其他节点就可以订阅该消息然后进行相应的处理了。

　　2、[测试](http://softtest.chinaitlab.com/)

　　安装完成后我们就可以运行[测试](http://softtest.chinaitlab.com/)了。

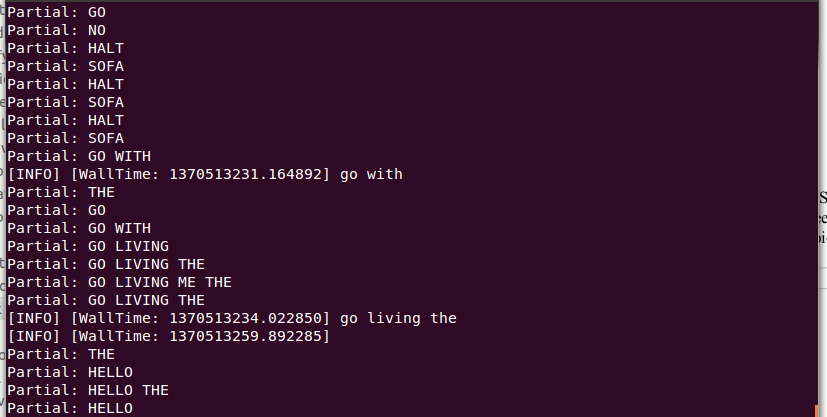
　　首先，插入你的麦克风设备，然后在系统设置里测试麦克风是否有语音输入。

　　然后，运行包中的测试程序：

    01.$ roslaunch pocketsphinx robocup.launch

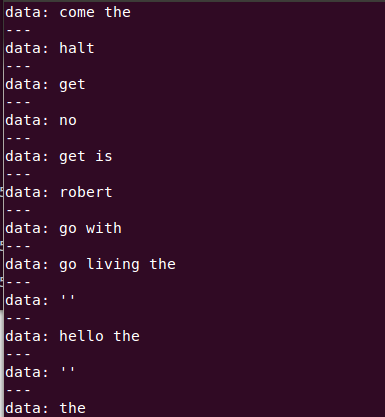
　　此时，在终端中会看到一大段的信息。尝试说一些简单的语句，当然，必须是英语，例如：bring me the glass，come with me，看看能不能识别出来。

　　《ros by example》这本书中写得测试还是很准确的，但是我在测试中感觉识别相当不准确，可能是我的英语太差了吧。



　　我们也可以直接看ROS最后发布的结果消息：

    01.$ rostopic echo /recognizer/output



**二、语音库**

　　1、查看语音库

　　这个语音识别时一种离线识别的方法，将一些常用的词汇放到一个文件中，作为识别的文本库，然后分段识别语音信号，最后在库中搜索对应的文本信息。如果想看语音识别库中有哪些文本信息，可以通过下面的指令进行查询：

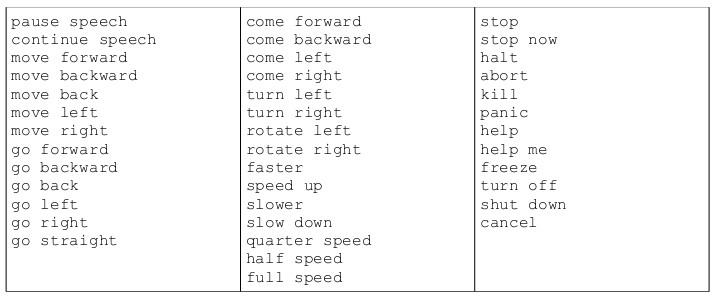
    01.$ roscd pocketsphinx/demo  
    02.$ more robocup.corpus

2、添加语音库

　　我们可以自己向语音库中添加其他的文本识别信息，《ros by example》自带的例程中是带有语音识别的例程的，而且有添加语音库的例子。

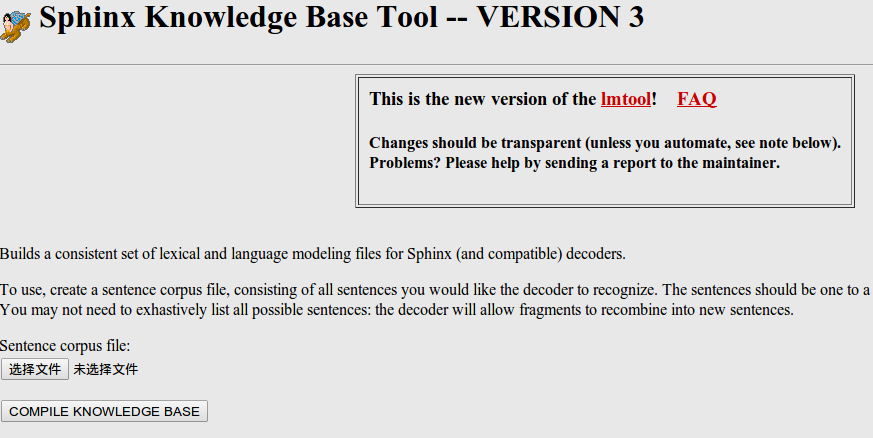
　　首先看看例子中要添加的文本信息：

    01.$ roscd rbx1\_speech/config  
    02.$ more nav\_commands.txt



　　这就是需要添加的文本，我们也可以修改其中的某些文本，改成自己需要的。

　　然后我们要把这个文件在线生成语音信息和库文件，这一步需要登陆网站http://www.speech.cs.cmu.edu/tools/lmtool-new.html，根据网站的提示上传文件，然后在线编译生成库文件。



　　把[下载](http://download.chinaitlab.com/)的文件都解压放在rbx1\_speech包的config文件夹下。我们可以给这些文件改个名字：

    01.$ roscd rbx1\_speech/config  
    02.$ rename -f 's/3026/nav\_commands/' \*

　　在rbx1\_speech/launch文件夹下看看voice\_nav\_commands.launch这个文件：

    01.<launch>  
    02.<node name="recognizer" pkg="pocketsphinx" type="recognizer.py"  
    03.output="screen">  
    04.<param name="lm" value="$（find rbx1\_speech）/config/nav\_commands.lm"/>  
    05.<param name="dict" value="$（find rbx1\_speech）/config/nav\_commands.dic"/>  
    06.</node>  
    07.</launch>

　　可以看到，这个launch文件在运行recognizer.py节点的时候使用了我们生成的语音识别库和文件参数，这样就可以实用我们自己的语音库来进行语音识别了。

　　通过之前的命令来测试一下效果如何吧：

    01.$ roslaunch rbx1\_speech voice\_nav\_commands.launch  
    02.$ rostopic echo /recognizer/output

**三、语音控制**

　　有了语音识别，我们就可以来做很多犀利的应用了，首先先来尝试一下用语音来控制机器人动作。

　　1、机器人控制节点

　　前面说到的recognizer.py会将最后识别的文本信息通过消息发布，那么我们来编写一个机器人控制节点接收这个消息，进行相应的控制即可。

　　在pocketsphinx包中本身有一个语音控制发布Twist消息的例程voice\_cmd\_vel.py，rbx1\_speech包对其进行了一些简化修改，在nodes文件夹里可以查看voice\_nav.py文件：

    01.#!/usr/bin/env python  
    02.  
    03."""  
    04.  voice\_nav.py  
    05.  
    06.  Allows controlling a mobile base using simple speech commands.  
    07.  
    08.  Based on the voice\_cmd\_vel.py script by Michael Ferguson in  
    09.  the pocketsphinx ROS package.  
    10.  
    11.  See <http://www.ros.org/wiki/pocketsphinx>  
    12."""  
    13.  
    14.import roslib; roslib.load\_manifest（'rbx1\_speech'）  
    15.import rospy  
    16.from geometry\_msgs.msg import Twist  
    17.from std\_msgs.msg import String  
    18.from math import copysign  
    19.  
    20.class VoiceNav:  
    21.    def \_\_init\_\_（self）：  
    22.        rospy.init\_node（'voice\_nav'）  
    23.  
    24.        rospy.on\_shutdown（self.cleanup）  
    25.  
    26.        # Set a number of parameters affecting the robot's speed  
    27.        self.max\_speed = rospy.get\_param（"~max\_speed", 0.4）  
    28.        self.max\_angular\_speed = rospy.get\_param（"~max\_angular\_speed", 1.5）  
    29.        self.speed = rospy.get\_param（"~start\_speed", 0.1）  
    30.        self.angular\_speed = rospy.get\_param（"~start\_angular\_speed", 0.5）  
    31.        self.linear\_increment = rospy.get\_param（"~linear\_increment", 0.05）  
    32.        self.angular\_increment = rospy.get\_param（"~angular\_increment", 0.4）  
    33.  
    34.        # We don't have to run the script very fast  
    35.        self.rate = rospy.get\_param（"~rate", 5）  
    36.        r = rospy.Rate（self.rate）  
    37.  
    38.        # A flag to determine whether or not voice control is paused  
    39.        self.paused = False  
    40.  
    41.        # Initialize the Twist message we will publish.  
    42.        self.cmd\_vel = Twist（）  
    43.  
    44.        # Publish the Twist message to the cmd\_vel topic  
    45.        self.cmd\_vel\_pub = rospy.Publisher（'cmd\_vel', Twist）  
    46.  
    47.        # Subscribe to the /recognizer/output topic to receive voice commands.  
    48.        rospy.Subscriber（'/recognizer/output', String, self.speech\_callback）  
    49.  
    50.        # A mapping from keywords or phrases to commands  
    51.        self.keywords\_to\_command = {'stop': ['stop', 'halt', 'abort', 'kill', 'panic', 'off', 'freeze', 'shut down', 'turn off', 'help', 'help me'],  
    52.                                    'slower': ['slow down', 'slower'],  
    53.                                    'faster': ['speed up', 'faster'],  
    54.                                    'forward': ['forward', 'ahead', 'straight'],  
    55.                                    'backward': ['back', 'backward', 'back up'],  
    56.                                    'rotate left': ['rotate left'],  
    57.                                    'rotate right': ['rotate right'],  
    58.                                    'turn left': ['turn left'],  
    59.                                    'turn right': ['turn right'],  
    60.                                    'quarter': ['quarter speed'],  
    61.                                    'half': ['half speed'],  
    62.                                    'full': ['full speed'],  
    63.                                    'pause': ['pause speech'],  
    64.                                    'continue': ['continue speech']}  
    65.  
    66.        rospy.loginfo（"Ready to receive voice commands"）  
    67.  
    68.        # We have to keep publishing the cmd\_vel message if we want the robot to keep moving.  
    69.        while not rospy.is\_shutdown（）：  
    70.            self.cmd\_vel\_pub.publish（self.cmd\_vel）  
    71.            r.sleep（）  
    72.  
    73.    def get\_command（self, data）：  
    74.        # Attempt to match the recognized word or phrase to the  
    75.        # keywords\_to\_command dictionary and return the appropriate  
    76.        # command  
    77.        for （command, keywords） in self.keywords\_to\_command.iteritems（）：  
    78.            for word in keywords:  
    79.                if data.find（word） > -1:  
    80.                    return command  
    81.  
    82.    def speech\_callback（self, msg）：  
    83.        # Get the motion command from the recognized phrase  
    84.        command = self.get\_command（msg.data）  
    85.  
    86.        # Log the command to the screen  
    87.        rospy.loginfo（"Command: " + str（command））  
    88.  
    89.        # If the user has asked to pause/continue voice control,  
    90.        # set the flag accordingly  
    91.        if command == 'pause':  
    92.            self.paused = True  
    93.        elif command == 'continue':  
    94.            self.paused = False  
    95.  
    96.        # If voice control is paused, simply return without  
    97.        # performing any action  
    98.        if self.paused:  
    99.            return  
    100.  
    101.        # The list of if-then statements should be fairly  
    102.        # self-explanatory  
    103.        if command == 'forward':  
    104.            self.cmd\_vel.linear.x = self.speed  
    105.            self.cmd\_vel.angular.z = 0  
    106.  
    107.        elif command == 'rotate left':  
    108.            self.cmd\_vel.linear.x = 0  
    109.            self.cmd\_vel.angular.z = self.angular\_speed  
    110.  
    111.        elif command == 'rotate right':  
    112.            self.cmd\_vel.linear.x = 0  
    113.            self.cmd\_vel.angular.z = -self.angular\_speed  
    114.  
    115.        elif command == 'turn left':  
    116.            if self.cmd\_vel.linear.x != 0:  
    117.                self.cmd\_vel.angular.z += self.angular\_increment  
    118.            else:  
    119.                self.cmd\_vel.angular.z = self.angular\_speed  
    120.  
    121.        elif command == 'turn right':  
    122.            if self.cmd\_vel.linear.x != 0:  
    123.                self.cmd\_vel.angular.z -= self.angular\_increment  
    124.            else:  
    125.                self.cmd\_vel.angular.z = -self.angular\_speed  
    126.  
    127.        elif command == 'backward':  
    128.            self.cmd\_vel.linear.x = -self.speed  
    129.            self.cmd\_vel.angular.z = 0  
    130.  
    131.        elif command == 'stop':  
    132.            # Stop the robot!  Publish a Twist message consisting of all zeros.  
    133.            self.cmd\_vel = Twist（）  
    134.  
    135.        elif command == 'faster':  
    136.            self.speed += self.linear\_increment  
    137.            self.angular\_speed += self.angular\_increment  
    138.            if self.cmd\_vel.linear.x != 0:  
    139.                self.cmd\_vel.linear.x += copysign（self.linear\_increment, self.cmd\_vel.linear.x）  
    140.            if self.cmd\_vel.angular.z != 0:  
    141.                self.cmd\_vel.angular.z += copysign（self.angular\_increment, self.cmd\_vel.angular.z）  
    142.  
    143.        elif command == 'slower':  
    144.            self.speed -= self.linear\_increment  
    145.            self.angular\_speed -= self.angular\_increment  
    146.            if self.cmd\_vel.linear.x != 0:  
    147.                self.cmd\_vel.linear.x -= copysign（self.linear\_increment, self.cmd\_vel.linear.x）  
    148.            if self.cmd\_vel.angular.z != 0:  
    149.                self.cmd\_vel.angular.z -= copysign（self.angular\_increment, self.cmd\_vel.angular.z）  
    150.  
    151.        elif command in ['quarter', 'half', 'full']:  
    152.            if command == 'quarter':  
    153.                self.speed = copysign（self.max\_speed / 4, self.speed）  
    154.  
    155.            elif command == 'half':  
    156.                self.speed = copysign（self.max\_speed / 2, self.speed）  
    157.  
    158.            elif command == 'full':  
    159.                self.speed = copysign（self.max\_speed, self.speed）  
    160.  
    161.            if self.cmd\_vel.linear.x != 0:  
    162.                self.cmd\_vel.linear.x = copysign（self.speed, self.cmd\_vel.linear.x）  
    163.  
    164.            if self.cmd\_vel.angular.z != 0:  
    165.                self.cmd\_vel.angular.z = copysign（self.angular\_speed, self.cmd\_vel.angular.z）  
    166.  
    167.        else:  
    168.            return  
    169.  
    170.        self.cmd\_vel.linear.x = min（self.max\_speed, max（-self.max\_speed, self.cmd\_vel.linear.x））  
    171.        self.cmd\_vel.angular.z = min（self.max\_angular\_speed, max（-self.max\_angular\_speed, self.cmd\_vel.angular.z））  
    172.  
    173.    def cleanup（self）：  
    174.        # When shutting down be sure to stop the robot!  
    175.        twist = Twist（）  
    176.        self.cmd\_vel\_pub.publish（twist）  
    177.        rospy.sleep（1）  
    178.  
    179.if \_\_name\_\_=="\_\_main\_\_":  
    180.    try:  
    181.        VoiceNav（）  
    182.        rospy.spin（）  
    183.    except rospy.ROSInterruptException:  
    184.        rospy.loginfo（"Voice navigation terminated."）

　　可以看到，代码中定义了接收到各种命令时的控制策略。

2、仿真测试

　　和前面一样，我们在rviz中进行仿真测试。

　　首先是运行一个机器人模型：

    01.$ roslaunch rbx1\_bringup fake\_turtlebot.launch

　　然后打开rviz：

    01.$ rosrun rviz rviz -d `rospack find rbx1\_nav`/sim\_fuerte.vcg

　　如果不喜欢在终端里看输出，可以打开gui界面：

    01.$ rxconsole

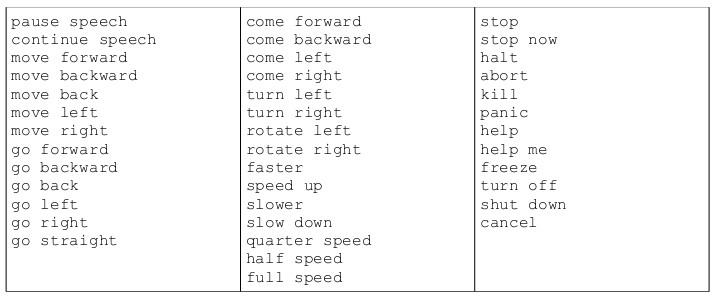
　　再打开语音识别的节点：

    01.$ roslaunch rbx1\_speech voice\_nav\_commands.launch

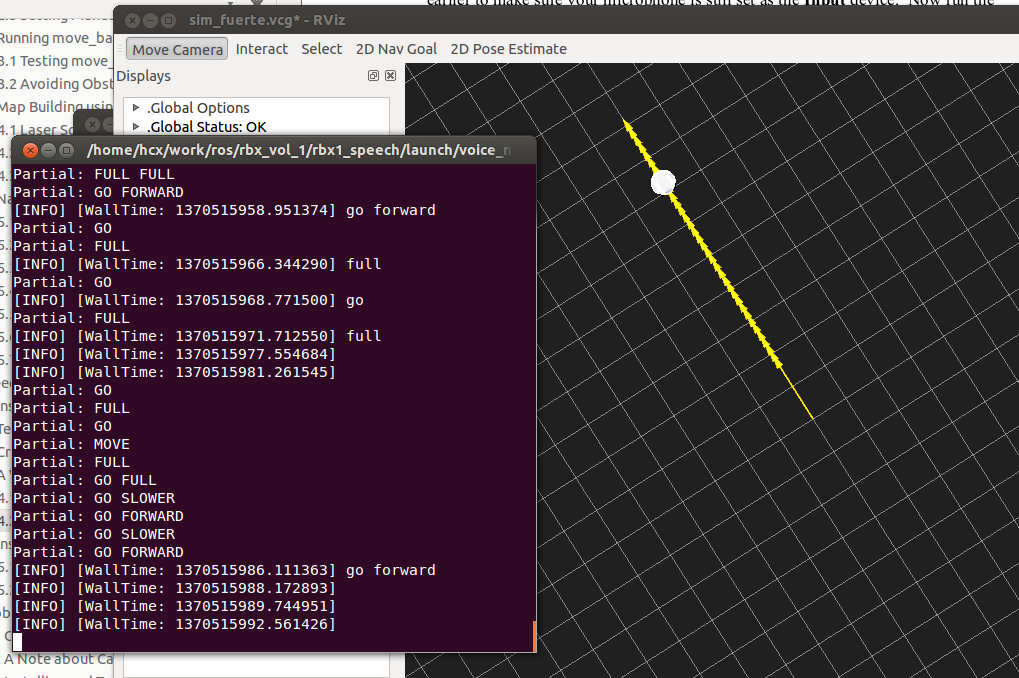
　　最后就是机器人的控制节点了：

    01.$ roslaunch rbx1\_speech turtlebot\_voice\_nav.launch

　　OK，打开上面这一堆的节点之后，就可以开始了。可以使用的命令如下：



　　下图是我的测试结果，不过感觉准确度还是欠佳：



**四、播放语音**

　　现在机器人已经可以按照我们说的话行动了，要是机器人可以和我们对话就更好了。ROS中已经集成了这样的包，下面就来尝试一下。

　　运行下面的命令：

    01.$ rosrun sound\_play soundplay\_node.py  
    02.$ rosrun sound\_play say.py "Greetings Humans. Take me to your leader."

　　有没有听见声音!ROS通过识别我们输入的文本，让机器人读了出来。发出这个声音的人叫做kal\_diphone，如果不喜欢，我们也可以换一个人来读：

    01.$ sudo apt-get install festvox-don  
    02.$ rosrun sound\_play say.py "Welcome to the future" voice\_don\_diphone

　　哈哈，这回换了一个人吧，好吧，这不也是我们的重点。

　　在rbx1\_speech/nodes文件夹中有一个让机器人说话的节点talkback.py:

    01.#!/usr/bin/env python  
    02.  
    03."""  
    04.    talkback.py - Version 0.1 2012-01-10  
    05.  
    06.    Use the sound\_play client to say back what is heard by the pocketsphinx recognizer.  
    07.  
    08.    Created for the Pi Robot Project: [http://www.pirobot.org](http://www.pirobot.org/)  
    09.    Copyright （c） 2012 Patrick Goebel.  All rights reserved.  
    10.  
    11.    This program is free software; you can redistribute it and/or modify  
    12.    it under the terms of the GNU General Public License as published by  
    13.    the Free Software Foundation; either version 2 of the License, or  
    14.    （at your option） any later version.5  
    15.  
    16.    This program is distributed in the hope that it will be useful,  
    17.    but WITHOUT ANY WARRANTY; without even the implied warranty of  
    18.    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the  
    19.    GNU General Public License for more details at:  
    20.  
    21.    <http://www.gnu.org/licenses/gpl.htmlPoint>  
    22."""  
    23.  
    24.import roslib; roslib.load\_manifest（'rbx1\_speech'）  
    25.import rospy  
    26.from std\_msgs.msg import String  
    27.from sound\_play.libsoundplay import SoundClient  
    28.import sys  
    29.  
    30.class TalkBack:  
    31.    def \_\_init\_\_（self, script\_path）：  
    32.        rospy.init\_node（'talkback'）  
    33.  
    34.        rospy.on\_shutdown（self.cleanup）  
    35.  
    36.        # Set the default TTS voice to use  
    37.        self.voice = rospy.get\_param（"~voice", "voice\_don\_diphone"）  
    38.  
    39.        # Set the wave file path if used  
    40.        self.wavepath = rospy.get\_param（"~wavepath", script\_path + "//sounds"）  
    41.  
    42.        # Create the sound client object  
    43.        self.soundhandle = SoundClient（）  
    44.  
    45.        # Wait a moment to let the client connect to the  
    46.        # sound\_play server  
    47.        rospy.sleep（1）  
    48.  
    49.        # Make sure any lingering sound\_play processes are stopped.  
    50.        self.soundhandle.stopAll（）  
    51.  
    52.        # Announce that we are ready for input  
    53.        self.soundhandle.playWave（self.wavepath + "/R2D2a.wav"）  
    54.        rospy.sleep（1）  
    55.        self.soundhandle.say（"Ready", self.voice）  
    56.  
    57.        rospy.loginfo（"Say one of the navigation commands…"）  
    58.  
    59.        # Subscribe to the recognizer output and set the callback function  
    60.        rospy.Subscriber（'/recognizer/output', String, self.talkback）  
    61.  
    62.    def talkback（self, msg）：  
    63.        # Print the recognized words on the screen  
    64.        rospy.loginfo（msg.data）  
    65.  
    66.        # Speak the recognized words in the selected voice  
    67.        self.soundhandle.say（msg.data, self.voice）  
    68.  
    69.        # Uncomment to play one of the built-in sounds  
    70.        #rospy.sleep（2）  
    71.        #self.soundhandle.play（5）  
    72.  
    73.        # Uncomment to play a wave file  
    74.        #rospy.sleep（2）  
    75.        #self.soundhandle.playWave（self.wavepath + "/R2D2a.wav"）  
    76.  
    77.    def cleanup（self）：  
    78.        self.soundhandle.stopAll（）  
    79.        rospy.loginfo（"Shutting down talkback node…"）  
    80.  
    81.if \_\_name\_\_=="\_\_main\_\_":  
    82.    try:  
    83.        TalkBack（sys.path[0]）  
    84.        rospy.spin（）  
    85.    except rospy.ROSInterruptException:  
    86.        rospy.loginfo（"Talkback node terminated."）

　　我们来运行看一下效果：

    01.$ roslaunch rbx1\_speech talkback.launch

　　然后再说话，ROS不仅将文本信息识别出来了，而且还读了出来，厉害吧。当然了，现在还没有加入什么人工智能的算法，不能让机器人和我们聪明的说话，不过这算是基础了，以后有时间再研究一下人工智能就更犀利了。

**操作系统探索总结（十一）机器视觉**

　　机器视觉在计算机时代已经越来越流行，摄像头价格越来越低廉，部分集成深度传感器的混合型传感器也逐渐在研究领域普及，例如微软推出的Kinect，而且与之配套的软件功能十分强大，为开发带来了极大的便利。ROS集成了Kinect的的驱动包OpenNI，而且使用OpenCV库可以进行多种多样的图像处理。

　　注：本章内容及代码均参考《ROS by Example》书中的第十章。

**一、图像显示**

　　我们从最基础的开始，想办法显示Kinect的图像数据。

　　1、安装驱动包

　　安装步骤很简单：

    01.$sudo apt-get install ros-fuerte-openni-kinect

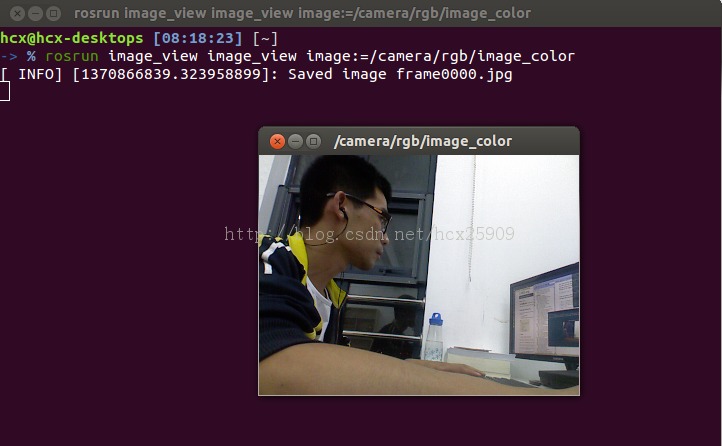
　　2、[测试](http://softtest.chinaitlab.com/)

　　首先运行kinect节点：

    01.$roslaunch rbx1\_vision openni\_node\_fuerte.launch

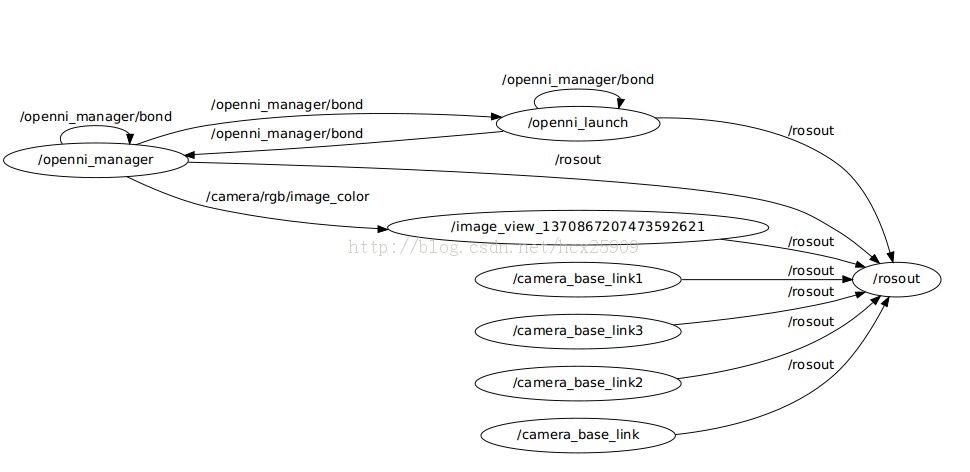
　　然后我们调用ROS的image\_view包来直接显示摄像头的数据库。  
  
    01.$rosrun image\_view image\_view image:=/camera/rgb/image\_color

　　我们可以看到弹出了一个独立的图像显示框：



　　3、分析数据

　　下图是我们上面[测试](http://softtest.chinaitlab.com/)中的节点图。



　　我们可以使用如下的命令来查看节点之间发送的图像消息是什么样的：

    01.rostopic echo /camera/rgb/image\_color

　　然后就会看到数据刷刷的在显示，有没有感觉看不清楚，我们使用终端的重定向功能将数据全部[存储](http://www.storworld.com/)到文件中：

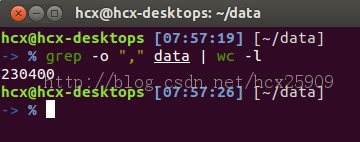
    01.rostopic echo /camera/rgb/image\_color > test

　　好了，现在看看文件中是不是已经有数据了，我们来看一下其中的一帧数据：

    01.header:  
    02.  seq: 19285  
    03.  stamp:  
    04.    secs: 1370867560  
    05.    nsecs: 538447820  
    06.  frame\_id: camera\_rgb\_optical\_frame  
    07.height: 240  
    08.width: 320  
    09.encoding: rgb8  
    10.is\_bigendian: 0  
    11.step: 960  
    12.data: [223, 225, 225, 220, 225, 225……

　　从数据中我们可以的出来几个很重要的参数，首先是图像的分辨率：240\*320，编码的格式是rgb8，也就是说图像应该有240\*320=76800个像素，而每个像素由八位的R、G、B三个数据组成，因此我们可以预计下面的data中应该有76800\*3=230400个八位的数据了。

我们可以验证一下data中到底有多少个数据，这个很简单了，数一下就行了，不过好像有点多，我使用的是[linux](http://linux.chinaitlab.com/)的“wc”命令。首先我一帧数据复制到单独的文件中，每个数据都是用“，”号隔开的，只要计算“，”的数量就知道数据的数量了。



　　结果和我们预想的是一样的。知道这个数据格式以后，我们以后就可以直接把其他摄像头的数据装换成这种格式的数据就可以直接在ROS中使用了。

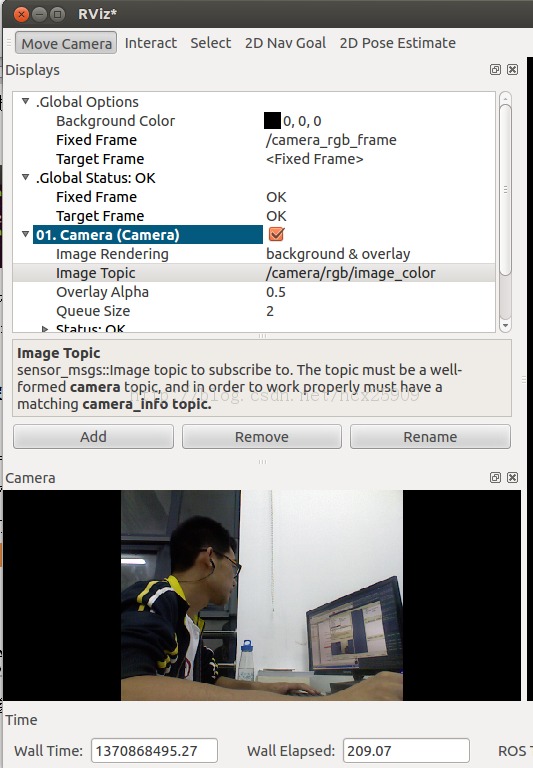
　　4、rviz显示图像

　　我们再来点高级的。rviz是我们经常使用的工具，把图像显示在rviz中才更有应用价值。rviz已经为我们提供好了显示图像的接口，使用非常简单。

　　首先按照上一节的方法运行kinect节点，然后打开rviz：

    01.rosrun rviz rviz

　　然后修改“Fixed Frame”为/camera\_rgb\_color，修改“Target Frame”为，接着点击add，选择camera类型。添加成功后选择camera菜单下的Iamge Topic选项，选择/camera/rgb/image\_color，确定后下面的显示框内就显示图像了。



**二、深度显示**

　　使用kinect的一大特色就是可以获得传感器的深度数据，也就是物体距离传感器的距离，传说kinect的可识别范围在60cm到10m之间。

　　1、显示深度图像

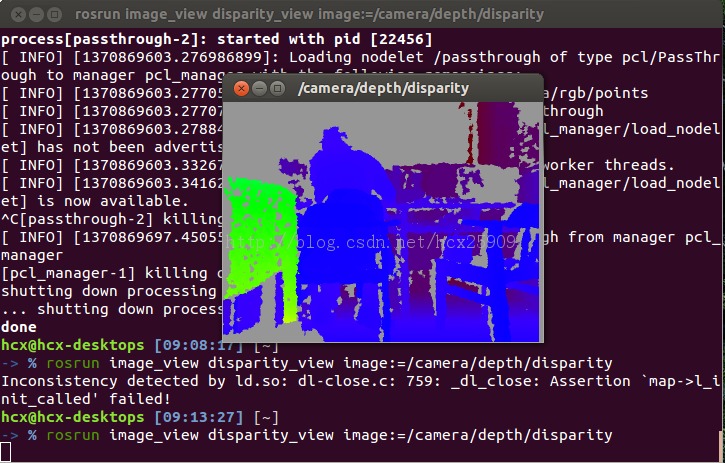
　　首先也需要运行Kinect的节点：

    01.roslaunch openni\_launch openni.launch

　　这一步我使用的是ROS安装openni包中的节点，使用与前面相同的命令运行的节点在后面无法产生深度数据。

　　然后同样适用iamge\_view包就可以简单的显示深度图像了：

    01.rosrun image\_view disparity\_view image:=/camera/depth/disparity



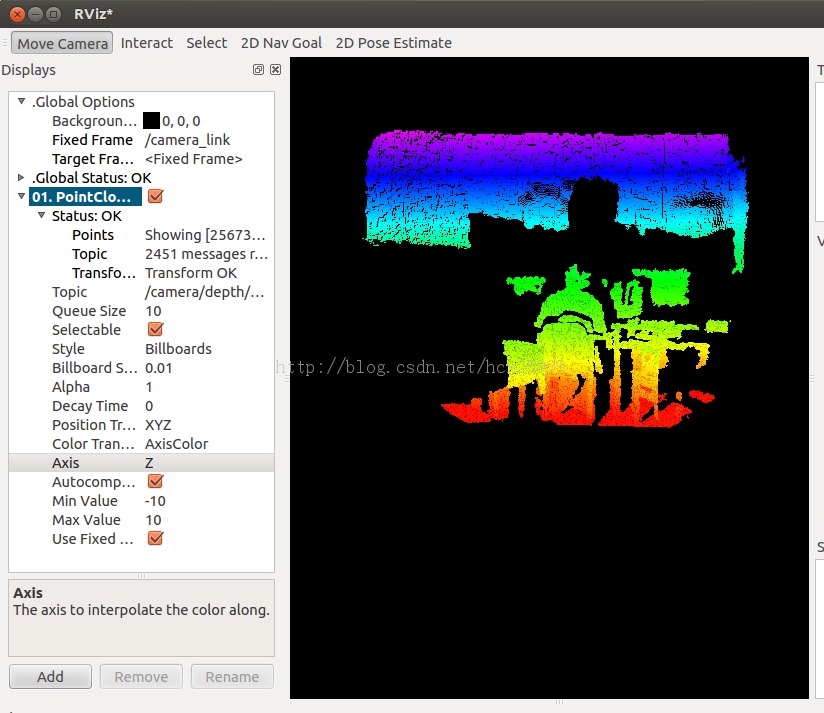
　　2、rviz中显示深度

　　首先运行rviz：

    01.rosrun rviz rviz

　　然后修改“Fixed Frame”和“Target Frame”，在add中添加PointCloud2类型，修改“topic”，具体参数如下图所示

　　：



**操作系统探索总结（十二）坐标系统**

　　在机器人的控制中，坐标系统是非常重要的，在ROS使用tf软件库进行坐标转换。

**一、tf简介**

　　我们通过一个小小的实例来介绍tf的作用。

　　1、安装turtle包

    01.$ rosdep install turtle\_tf rviz  
    02.$ rosmake turtle\_tf rviz

　　2、运行demo

　　运行简单的demo：

    01.$ roslaunch turtle\_tf turtle\_tf\_demo.launch

　　然后就会看到两只小乌龟了。



　　该例程中带有turtlesim仿真，可以在终端激活的情况下进行键盘控制。



　　可以发现，第二只乌龟会跟随你移动的乌龟进行移动。

　　3、demo分析

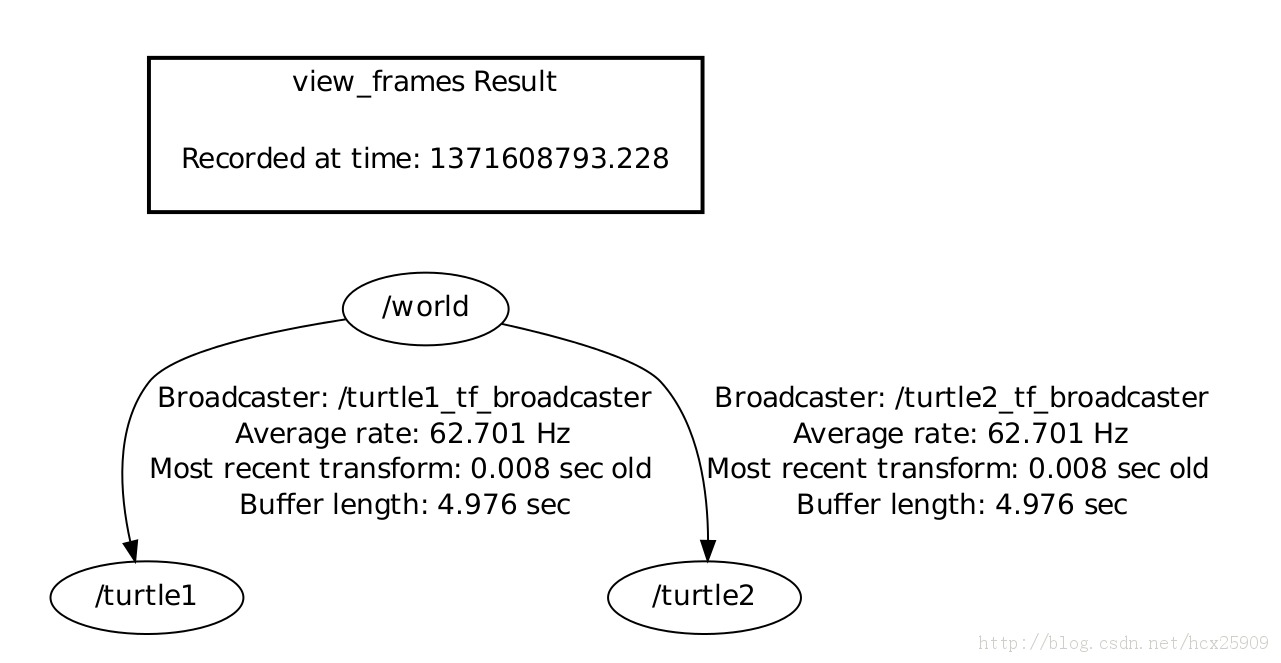
　　接下来我们就来看一看到底ROS做了什么事情。

　　这个例程使用tf建立了三个坐标帧：a world frame, a turtle1 frame, and a turtle2 frame。然后使用tf broadcaster发布乌龟的坐标帧，并且使用tf listener计算乌龟坐标帧之间的差异，使得第二只乌龟跟随第一只乌龟。

　　我们可以使用tf工具来具体研究。

    01.$ rosrun tf view\_frames

　　然后会看到一些提示，并且生成了一个frames.pdf文件。



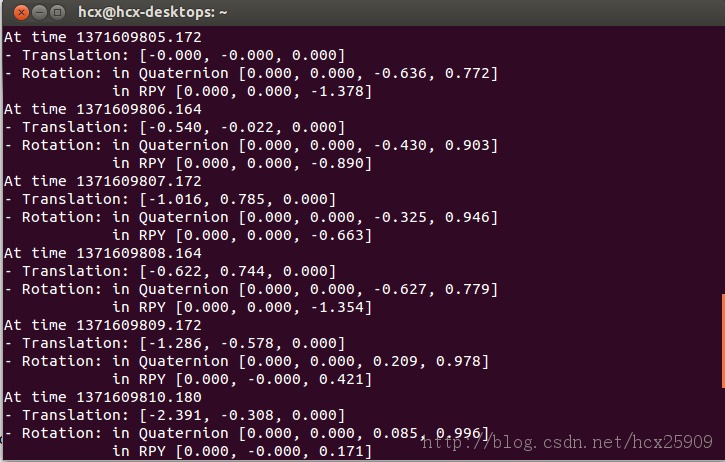
　　该文件描述了坐标系之间的联系。三个节点分别是三个坐标帧，而/world是其他两个乌龟坐标帧的父坐标帧。还包含一些调试需要的发送频率、最近时间等信息。

　　tf还提供了一个tf\_echo工具来查看两个广播帧之间的关系。我们可以看一下第二只得乌龟坐标是怎么根据第一只乌龟得出来的。

http://img.blog.csdn.net/20130705193826750?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvaGN4MjU5MDk=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

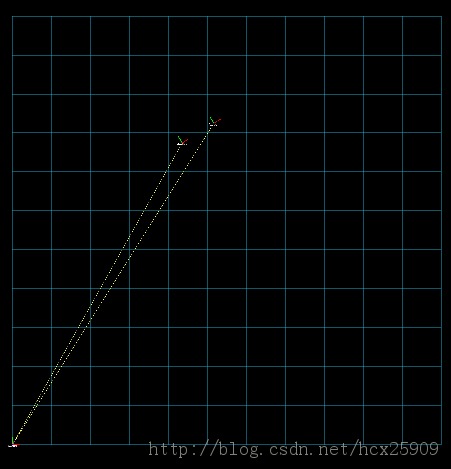
    01.$ rosrun tf tf\_echo turtle1 turtle2

　　控制一只乌龟，在终端中会看到第二只乌龟的坐标转换关系。



　　我们也可以通过rviz的图形界面更加形象的看到这三者之间的关系。

    01.$ rosrun rviz rviz -d `rospack find turtle\_tf`/rviz/turtle\_rviz.vcg



　　移动乌龟，可以看到在rviz中的坐标会跟随变化。其中左下角的是/world，其他两个是乌龟的坐标帧。

　　下面我们就来详细分析这个实例。

**二、Writing a tf broadcaster**

　　1、创建包

    01.$ roscd tutorials  
    02.$ roscreate-pkg learning\_tf tf roscpp rospy turtlesim  
    03.$ rosmake learning\_tf

　　2、broadcast transforms

　　我们首先看一下如何把坐标帧发布到tf。

　　代码文件：/nodes/turtle\_tf\_broadcaster.py

    01.#!/usr/bin/env python  
    02.import roslib  
    03.roslib.load\_manifest（'learning\_tf'）  
    04.import rospy  
    05.  
    06.  
    07.import tf  
    08.import turtlesim.msg  
    09.  
    10.  
    11.def handle\_turtle\_pose（msg, turtlename）：  
    12.    br = tf.TransformBroadcaster（）  
    13.    br.sendTransform（（msg.x, msg.y, 0），  
    14.                     tf.transformations.quaternion\_from\_euler（0, 0, msg.theta），  
    15.                     rospy.Time.now（），  
    16.                     turtlename,  
    17.                     "world"）  #发布乌龟的平移和翻转  
    18.  
    19.  
    20.if \_\_name\_\_ == '\_\_main\_\_':  
    21.    rospy.init\_node（'turtle\_tf\_broadcaster'）  
    22.    turtlename = rospy.get\_param（'~turtle'）   #获取海龟的名字（turtle1,turtle2）  
    23.    rospy.Subscriber（'/%s/pose' % turtlename,  
    24.                     turtlesim.msg.Pose,  
    25.                     handle\_turtle\_pose,  
    26.                     turtlename）   #订阅 topic "turtleX/pose"  
    27.    rospy.spin（）

　　创建launch文件start\_demo.launch：

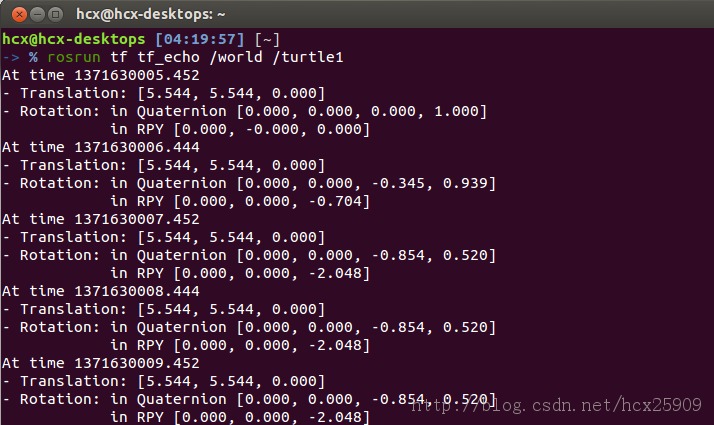
    01.<launch>  
    02.    <!-- Turtlesim Node-->  
    03.    <node pkg="turtlesim" type="turtlesim\_node" name="sim"/>  
    04.    <node pkg="turtlesim" type="turtle\_teleop\_key" name="teleop" output="screen"/>  
    05.  
    06.  
    07.    <node name="turtle1\_tf\_broadcaster" pkg="learning\_tf" type="turtle\_tf\_broadcaster.py" respawn="false" output="screen" >  
    08.      <param name="turtle" type="string" value="turtle1" />  
    09.    </node>  
    10.    <node name="turtle2\_tf\_broadcaster" pkg="learning\_tf" type="turtle\_tf\_broadcaster.py" respawn="false" output="screen" >  
    11.      <param name="turtle" type="string" value="turtle2" />  
    12.    </node>  
    13.  
    14.  
    15.  </launch>

　　运行：

    01.$ roslaunch learning\_tf start\_demo.launch

　　可以看到界面中只有移植乌龟了，打开tf\_echo的信息窗口：

    01.$ rosrun tf tf\_echo /world /turtle1



　　world坐标系的原点在最下角，对于turtle1的转换关系，其实就是turtle1在world坐标系中所在的坐标位置以及旋转角度。

**三、Writing a tf listener**

　　这一步，我们将看到如何使用tf进行帧转换。首先写一个tf listener(nodes/turtle\_tf\_listener.py)：

    01.#!/usr/bin/env python  
    02.import roslib  
    03.roslib.load\_manifest（'learning\_tf'）  
    04.import rospy  
    05.import math  
    06.import tf  
    07.import turtlesim.msg  
    08.import turtlesim.srv  
    09.  
    10.if \_\_name\_\_ == '\_\_main\_\_':  
    11.    rospy.init\_node（'tf\_turtle'）  
    12.  
    13.    listener = tf.TransformListener（） #TransformListener创建后就开始接受tf广播信息，最多可以缓存10s  
    14.  
    15.    rospy.wait\_for\_service（'spawn'）  
    16.    spawner = rospy.ServiceProxy（'spawn', turtlesim.srv.Spawn）  
    17.    spawner（4, 2, 0, 'turtle2'）  
    18.  
    19.    turtle\_vel = rospy.Publisher（'turtle2/command\_velocity', turtlesim.msg.Velocity）  
    20.  
    21.    rate = rospy.Rate（10.0）  
    22.    while not rospy.is\_shutdown（）：  
    23.        try:  
    24.            （trans,rot） = listener.lookupTransform（'/turtle2', '/turtle1', rospy.Time（0））  
    25.        except （tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException）：  
    26.            continue  
    27.  
    28.        angular = 4 \* math.atan2（trans[1], trans[0]）  
    29.        linear = 0.5 \* math.sqrt（trans[0] \*\* 2 + trans[1] \*\* 2）  
    30.        turtle\_vel.publish（turtlesim.msg.Velocity（linear, angular））  
    31.  
    32.        rate.sleep（）

　　在launch文件中添加下面的节点：

    01.<launch>  
    02.    …  
    03.    <node pkg="learning\_tf" type="turtle\_tf\_listener.py"  
    04.          name="listener" />  
    05.</launch>

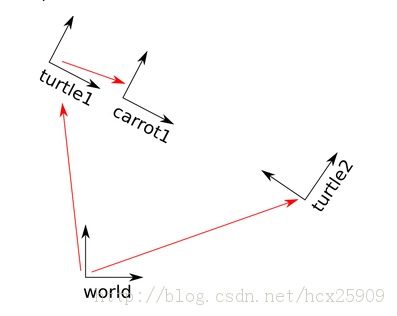
　　然后在运行，就可以看到两只turtle了，也就是我们在最开始见到的那种跟随效果。

**四、Adding a frame**

　　在很多应用中，添加一个帧是很有必要的，比如在一个world坐标系下，有很一个激光扫描节点，tf可以帮助我们将激光扫描的信息坐标装换成全局坐标。

　　1、帧结构

　　tf中的坐标帧是一个树状的结构，world坐标系统是最顶端的父坐标帧，其他的坐标帧都需要向下延伸。如果我们在上文的基础上添加一个帧，就需要让这个新的帧成为已有三个帧中的一个的子帧。



　　2、建立固定帧(fixed frame)

　　我们以turtle1作为父帧，建立一个新的帧“carrot1”。代码如下(nodes/fixed\_tf\_broadcaster.py)：

    01.#!/usr/bin/env python  
    02.import roslib  
    03.roslib.load\_manifest（'learning\_tf'）  
    04.  
    05.import rospy  
    06.import tf  
    07.  
    08.if \_\_name\_\_ == '\_\_main\_\_':  
    09.    rospy.init\_node（'my\_tf\_broadcaster'）  
    10.    br = tf.TransformBroadcaster（）  
    11.    rate = rospy.Rate（10.0）  
    12.    while not rospy.is\_shutdown（）：  
    13.        br.sendTransform（（0.0, 2.0, 0.0），  
    14.                         （0.0, 0.0, 0.0, 1.0），  
    15.                         rospy.Time.now（），  
    16.                         "carrot1",  
    17.                         "turtle1"） #建立一个新的帧，父帧为turtle1,并且距离父帧2米  
    18.        rate.sleep（）

　　在launch文件中添加节点：

    01.<launch>  
    02.  …  
    03.  <node pkg="learning\_tf" type="fixed\_tf\_broadcaster.py"  
    04.        name="broadcaster\_fixed" />  
    05.</launch>

　　运行，还是看到两只乌龟和之前的效果一样。新添加的帧并没有对其他帧产生什么影响。打开nodes/turtle\_tf\_listener.py文件，将turtle1改成carrot1：

    01.（trans,rot） = self.tf.lookupTransform（"/turtle2", "/carrot1", rospy.Time（0））

　　重新运行，现在乌龟之间的跟随关系就改变了：



　　3、建立移动帧(moving frame)

　　我们建立的新帧是一个固定的帧，在仿真过程中不会改变，如果我们要把carrot1帧和turtle1帧之间的关系设置可变的，可以修改代码如下：

    01.#!/usr/bin/env python  
    02.import roslib  
    03.roslib.load\_manifest（'learning\_tf'）  
    04.  
    05.import rospy  
    06.import tf  
    07.import math  
    08.  
    09.if \_\_name\_\_ == '\_\_main\_\_':  
    10.    rospy.init\_node（'my\_tf\_broadcaster'）  
    11.    br = tf.TransformBroadcaster（）  
    12.    rate = rospy.Rate（10.0）  
    13.    while not rospy.is\_shutdown（）：  
    14.        t = rospy.Time.now（）。to\_sec（） \* math.pi  
    15.        br.sendTransform（（2.0 \* math.sin（t）， 2.0 \* math.cos（t）， 0.0），  
    16.                         （0.0, 0.0, 0.0, 1.0），  
    17.                         rospy.Time.now（），  
    18.                         "carrot1",  
    19.                         "turtle1"）  
    20.        rate.sleep（）

　　这次carrot1的位置现对于turtle1来说是一个三角函数关系了。



**操作系统探索总结（十三）导航与定位框架**

　　导航与定位是机器人研究中的重要部分。

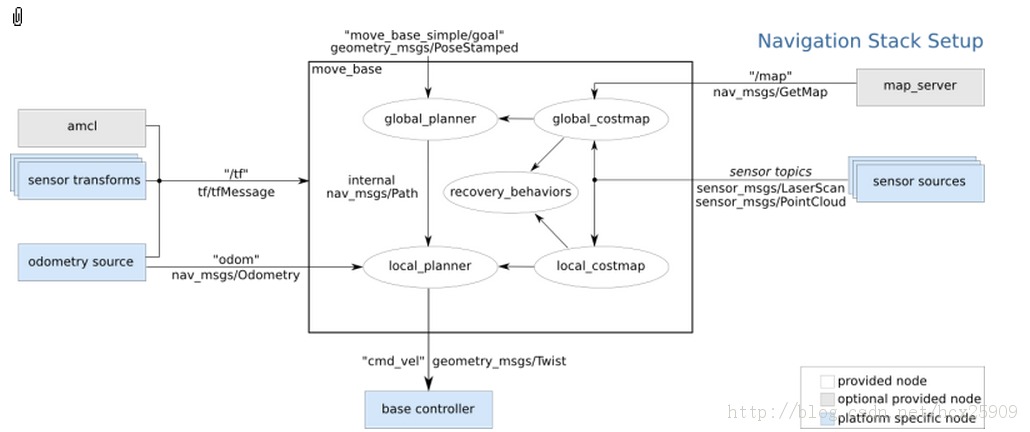
　　一般机器人在陌生的环境下需要使用激光传感器(或者深度传感器转换成激光数据)，先进行地图建模，然后在根据建立的地图进行导航、定位。在ROS中也有很多完善的包可以直接使用。

　　在ROS中，进行导航需要使用到的三个包是：

　　(1) move\_base：根据参照的消息进行路径规划，使移动机器人到达指定的位置;

　　(2) gmapping：根据激光数据(或者深度数据模拟的激光数据)建立地图;

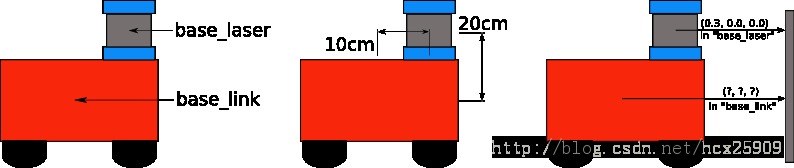
　　整体导航包的格局如下图所示：



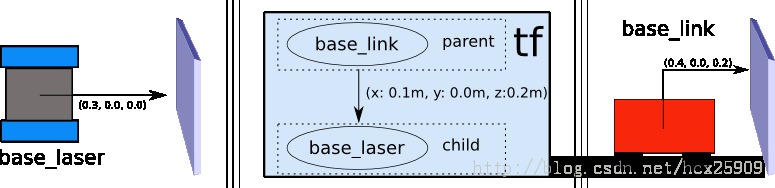
　　其中白色框内的是ROS已经为我们准备好的必须使用的组件，灰色框内的是ROS中可选的组件，蓝色的是用户需要提供的机器人平台上的组件。

　　1、sensor transforms

　　其中涉及到使用tf进行传感器坐标的转换，因为我们使用的机器人的控制中心不一定是在传感器上，所以要把传感器的数据转换成在控制中心上的坐标信息。如下图所示，传感器获取的数据是在base\_laser的坐标系统中的，但是我们控制的时候是以base\_link为中心，所以要根据两者的位置关心进行坐标的变换。



　　变换的过程不需要我们自己处理，只需要将base\_laser和base\_link两者之间的位置关系告诉tf，就可以自动转换了。具体的实现可以参见：http://blog.csdn.net/hcx25909/article/details/9255001。



　　2、sensor sources

　　这里是机器人导航传感器数据输入，一般只有两种：

　　(1) 激光传感器数据

　　(2) 点云数据

　　3、odometry source

　　机器人的导航需要输入里程计的数据(tf，nav\_msgs/Odometry\_message)，

　　4、base controller

　　在导航过程中，该部分负责将之前得出来的数据转封装成具体的线速度和转向角度信息(Twist)，并且发布给硬件平台。

　　5、map\_server

　　在导航过程中，地图并不是必须的，此时相当于是在一个无限大的空地上进行导航，并没有任何障碍物。但是考虑到实际情况，在我们使用导航的过程中还是要一个地图的。

　　在ROS的导航中，costmap\_2d这个包主要负责根据传感器的信息建立和更新二维或三维的地图。ROS的地图(costmap)采用网格(grid)的形式，每个网格的值从0~255，分为三种状态：占用(有障碍物)、无用(空闲的)、未知。具体状态和值的对应关系如下：



　　上图共分为五个部分：(下面的红色框图是机器人的轮廓，旁边的黑框是上图的映射位置)

　　(1) Lethal(致命的):机器人的中心与该网格的中心重合，此时机器人必然与障碍物冲突。

　　(2) Inscribed(内切)：网格的外切圆与机器人的轮廓内切，此时机器人也必然与障碍物冲突。

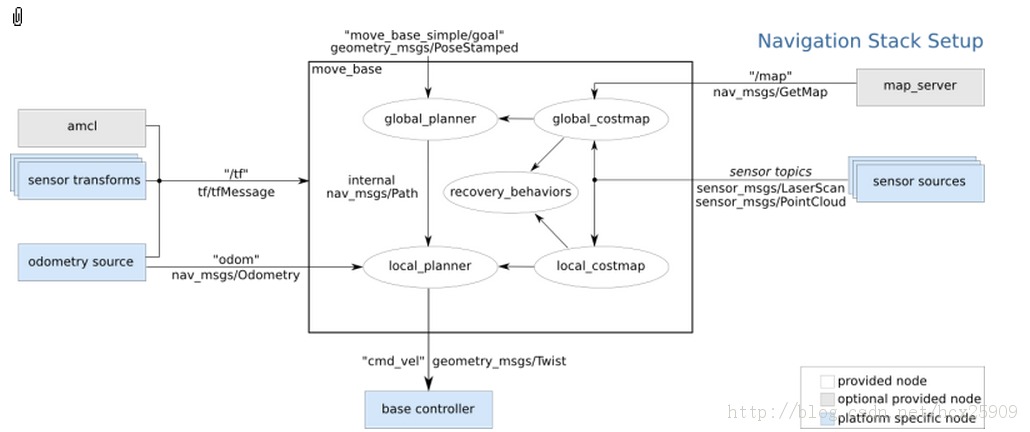
　　(3) Possibly circumscribed(外切)：网格的外切圆与机器人的轮廓外切，此时机器人相当于靠在障碍物附近，所以不一定冲突。

　　(4) Freespace(自由空间)：没有障碍物的空间。

　　(5) Unknown(未知)：未知的空间。

**操作系统探索总结（十四）move\_base（路径规划）**

　　在上一篇文章中，我们一起学习了ROS定位于导航的总体框架，这一篇我们主要研究其中最重要的move\_base包。



　　在总体框架图中可以看到，move\_base提供了ROS导航的配置、运行、交互接口，它主要包括两个部分：

　　(1) 全局路径规划(global planner)：根据给定的目标位置进行总体路径的规划;

　　(2) 本地实时规划(local planner)：根据附近的障碍物进行躲避路线规划。

**一、数据结构**

　　ROS中定义了MoveBaseActionGoal数据结构来[存储](http://www.storworld.com/)导航的目标位置数据，其中最重要的就是位置坐标(position)和方向(orientation)。

    01.rosmsg show MoveBaseActionGoal  
    02.  
    03.[move\_base\_msgs/MoveBaseActionGoal]:  
    04.std\_msgs/Header header  
    05.  uint32 seq  
    06.  time stamp  
    07.  string frame\_id  
    08.actionlib\_msgs/GoalID goal\_id  
    09.  time stamp  
    10.  string id  
    11.move\_base\_msgs/MoveBaseGoal goal  
    12.  geometry\_msgs/PoseStamped target\_pose  
    13.    std\_msgs/Header header  
    14.      uint32 seq  
    15.      time stamp  
    16.      string frame\_id  
    17.    geometry\_msgs/Pose pose  
    18.      geometry\_msgs/Point position  
    19.        float64 x  
    20.        float64 y  
    21.        float64 z  
    22.      geometry\_msgs/Quaternion orientation  
    23.        float64 x  
    24.        float64 y  
    25.        float64 z  
    26.        float64 w

**二、配置文件**

　　move\_base使用前需要配置一些参数：运行成本、机器人半径、到达目标位置的距离，机器人移动的速度，这些参数都在rbx1\_nav包的以下几个配置文件中：

　　• base\_local\_planner\_params.yaml

　　• costmap\_common\_params.yaml

　　• global\_costmap\_params.yaml

　　• local\_costmap\_params.yaml

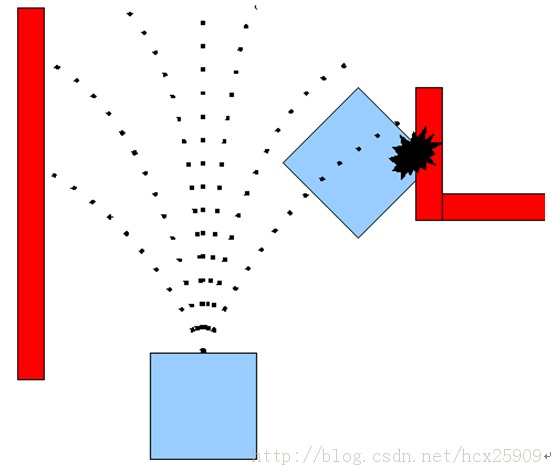
**三、全局路径规划(global planner)**

　　在ROS的导航中，首先会通过全局路径规划，计算出机器人到目标位置的全局路线。这一功能是navfn这个包实现的。

　　navfn通过Dijkstra最优路径的算法，计算costmap上的最小花费路径，作为机器人的全局路线。将来在算法上应该还会加入A\*算法。

**四、本地实时规划(local planner)**

　　本地的实时规划是利用base\_local\_planner包实现的。该包使用Trajectory Rollout 和Dynamic Window approaches算法计算机器人每个周期内应该行驶的速度和角度(dx，dy，dtheta velocities)。



　　base\_local\_planner这个包通过地图数据，通过算法搜索到达目标的多条路经，利用一些评价标准(是否会撞击障碍物，所需要的时间等等)选取最优的路径，并且计算所需要的实时速度和角度。

　　其中，Trajectory Rollout 和Dynamic Window approaches算法的主要思路如下：

　　(1) 采样机器人当前的状态(dx,dy,dtheta);

　　(2) 针对每个采样的速度，计算机器人以该速度行驶一段时间后的状态，得出一条行驶的路线。

　　(3) 利用一些评价标准为多条路线打分。

　　(4) 根据打分，选择最优路径。

　　(5) 重复上面过程。

　　五、ArbotiX仿真——手动设定目标

　　在这一步，我们暂时使用空白地图(blank\_map.pgm)，就在空地上进行无障碍仿真。

　　首先运行ArbotiX节点，并且加载机器人的URDF文件。

    01.roslaunch rbx1\_bringup fake\_turtlebot.launch

　　然后运行move\_base和加载空白地图的launch文件(fake\_move\_base\_blank\_map.launch)：

    01.roslaunch rbx1\_nav fake\_move\_base\_blank\_map.launch

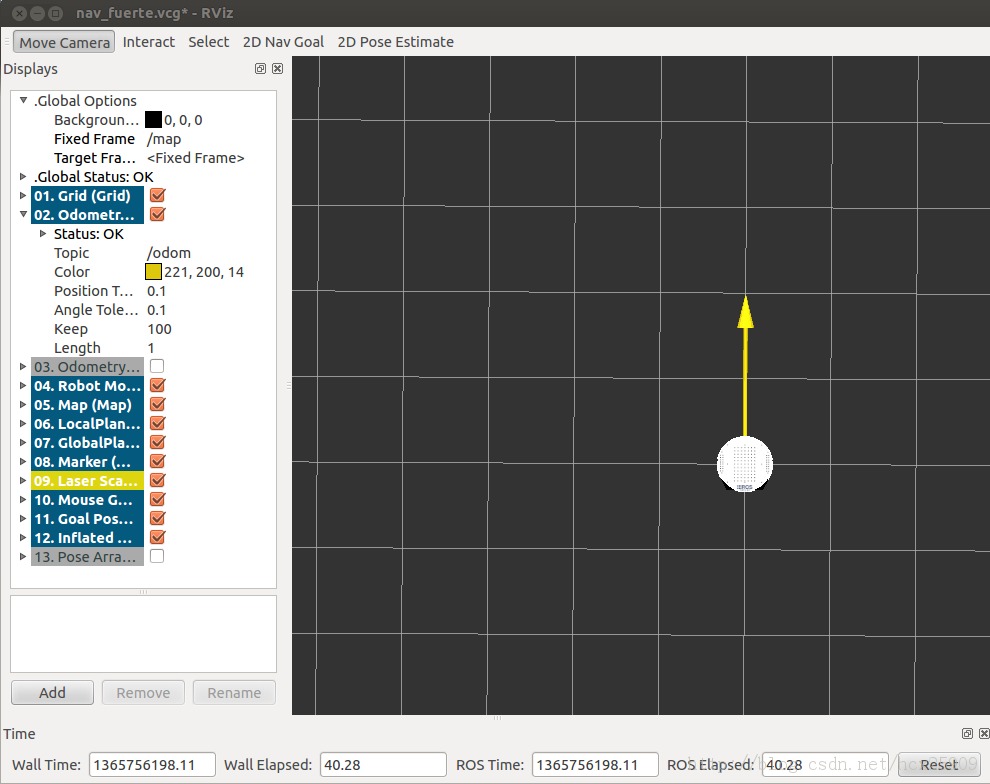
　　该文件的具体内容如下：

    01.<launch>  
    02.  <!-- Run the map server with a blank map -->  
    03.  <node name="map\_server" pkg="map\_server" type="map\_server" args="$（find rbx1\_nav）/maps/blank\_map.yaml"/>  
    04.  
    05.  <include file="$（find rbx1\_nav）/launch/fake\_move\_base.launch" />  
    06.  
    07.  <!-- Run a static transform between /odom and /map -->  
    08.  <node pkg="tf" type="static\_transform\_publisher" name="odom\_map\_broadcaster" args="0 0 0 0 0 0 /map /odom 100" />  
    09.  
    10.</launch>

　　其中调用了fake\_move\_base.launch文件，是运行move\_base节点并进行参数配置。

　　然后调用rviz就可以看到机器人了。

    01.rosrun rviz rviz -d `rospack find rbx1\_nav`/nav\_fuerte.vcg



　　我们先以1m的速度进行一下[测试](http://softtest.chinaitlab.com/)：

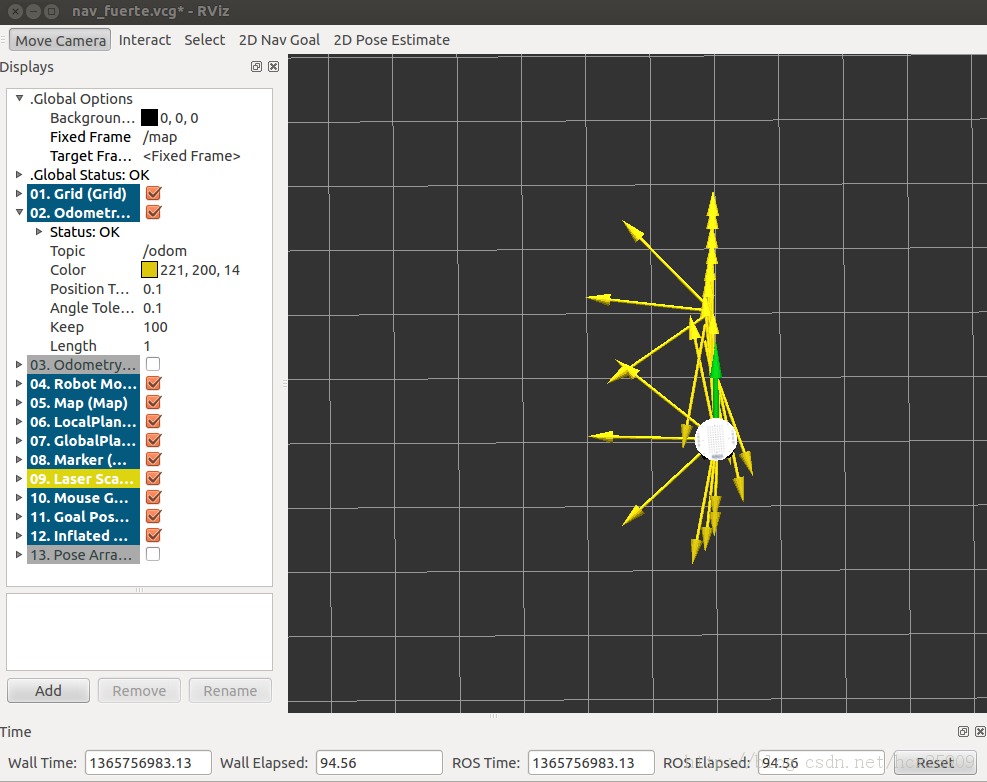
　　让机器人前进一米：

    01.rostopic pub /move\_base\_simple/goal geometry\_msgs/PoseStamped \  
    02.'{ header: { frame\_id: "base\_link" }, pose: { position: { x: 1.0, y: 0, z: 0 }, orientation: { x: 0, y: 0, z: 0, w: 1 } } }'

　　让机器人后退一米，回到原来的位置：

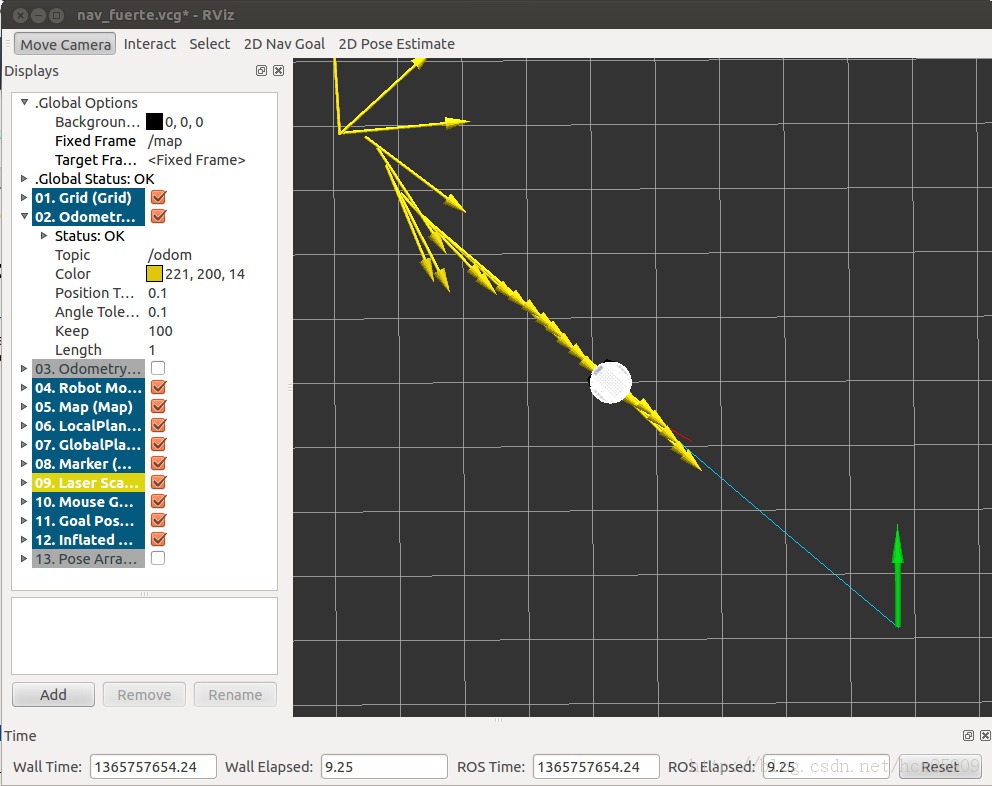
    01.rostopic pub /move\_base\_simple/goal geometry\_msgs/PoseStamped \  
    02.'{ header: { frame\_id: "map" }, pose: { position: { x: 0, y: 0, z: 0 }, orientation: { x: 0, y: 0, z: 0, w: 1 } } }'

　　在rviz中的轨迹图如下：



　　在机器人移动过程中，有一条蓝色的线(被黄线挡住了)就是机器人的全局规划的路径;红色的箭头是实施规划的路线，会不断更新，有的时候会呈现很大的弧线，那是因为机器人在转向的过程中尽量希望保持平稳的角度。如果觉得路径规划的精度不够，可以修改配置文件中的pdist\_scale参数进行修正。

然后我们可以认为的确定目标位置，点击rviz上方的2D Nav Goal按键，然后左键选择目标位置，机器人就开始自动导航了。

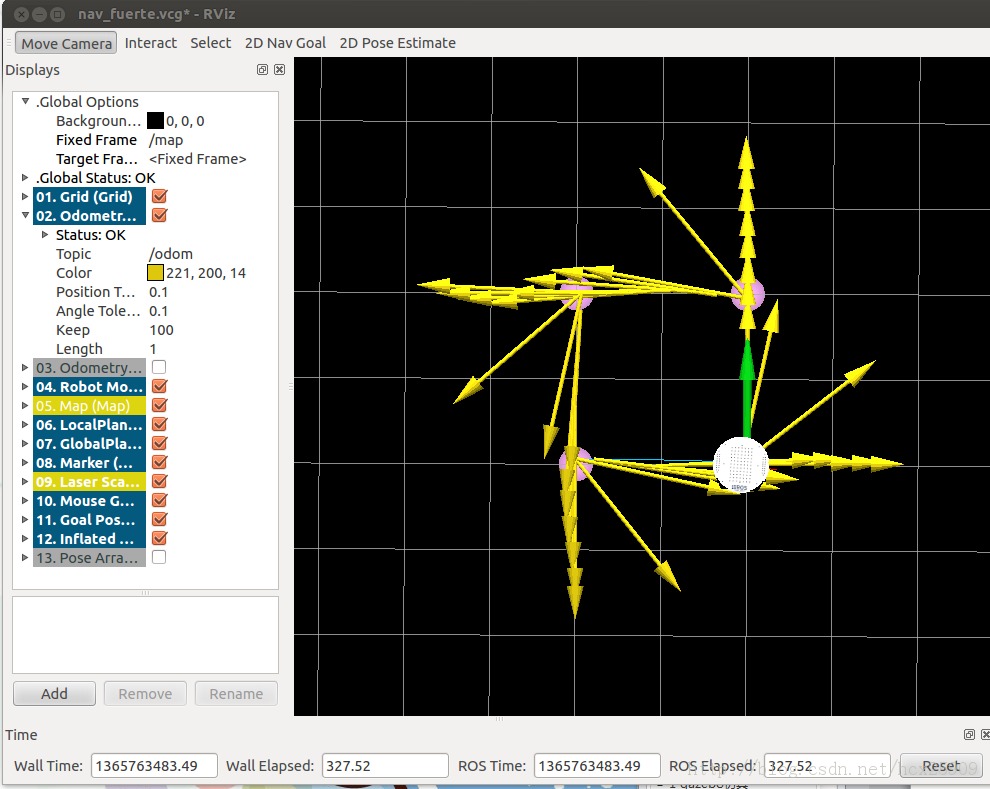


**六、ArbotiX仿真——带有障碍物的路径规划**

　　首先我们让机器人走一个正方形的路线。先通过上面的命令，让机器人回到原始位置(0，0，0)，然后按reset按键，把所有的箭头清除。接着运行走正方形路径的代码：

    01.rosrun rbx1\_nav move\_base\_square.py

　　在rviz中可以看到：



　　四个顶角的粉色圆盘就是我们设定的位置，正方形比较规则，可见定位还是比较准确的。然我们先来分析一下走正方形路线的代码：

    01.#!/usr/bin/env python  
    02.import roslib; roslib.load\_manifest（'rbx1\_nav'）  
    03.import rospy  
    04.import actionlib  
    05.from actionlib\_msgs.msg import \*  
    06.from geometry\_msgs.msg import Pose, Point, Quaternion, Twist  
    07.from move\_base\_msgs.msg import MoveBaseAction, MoveBaseGoal  
    08.from tf.transformations import quaternion\_from\_euler  
    09.from visualization\_msgs.msg import Marker  
    10.from math import radians, pi  
    11.  
    12.class MoveBaseSquare（）：  
    13.    def \_\_init\_\_（self）：  
    14.        rospy.init\_node（'nav\_test', anonymous=False）  
    15.  
    16.        rospy.on\_shutdown（self.shutdown）  
    17.  
    18.        # How big is the square we want the robot to navigate?  
    19.        # 设定正方形的尺寸，默认是一米  
    20.        square\_size = rospy.get\_param（"~square\_size", 1.0） # meters  
    21.  
    22.        # Create a list to hold the target quaternions （orientations）  
    23.        # 创建一个列表，保存目标的角度数据  
    24.        quaternions = list（）  
    25.  
    26.        # First define the corner orientations as Euler angles  
    27.        # 定义四个顶角处机器人的方向角度（Euler angles:http://zh.wikipedia.org/wiki/%E6%AC%A7%E6%8B%89%E8%A7%92）  
    28.        euler\_angles = （pi/2, pi, 3\*pi/2, 0）  
    29.  
    30.        # Then convert the angles to quaternions  
    31.        # 将上面的Euler angles转换成Quaternion的格式  
    32.        for angle in euler\_angles:  
    33.            q\_angle = quaternion\_from\_euler（0, 0, angle, axes='sxyz'）  
    34.            q = Quaternion（\*q\_angle）  
    35.            quaternions.append（q）  
    36.  
    37.        # Create a list to hold the waypoint poses  
    38.        # 创建一个列表[存储](http://www.storworld.com/)导航点的位置  
    39.        waypoints = list（）  
    40.  
    41.        # Append each of the four waypoints to the list.  Each waypoint  
    42.        # is a pose consisting of a position and orientation in the map frame.  
    43.        # 创建四个导航点的位置（角度和坐标位置）  
    44.        waypoints.append（Pose（Point（square\_size, 0.0, 0.0）， quaternions[0]））  
    45.        waypoints.append（Pose（Point（square\_size, square\_size, 0.0）， quaternions[1]））  
    46.        waypoints.append（Pose（Point（0.0, square\_size, 0.0）， quaternions[2]））  
    47.        waypoints.append（Pose（Point（0.0, 0.0, 0.0）， quaternions[3]））  
    48.  
    49.        # Initialize the visualization markers for RViz  
    50.        # 初始化可视化标记  
    51.        self.init\_markers（）  
    52.  
    53.        # Set a visualization marker at each waypoint  
    54.        # 给每个定点的导航点一个可视化标记（就是rviz中看到的粉色圆盘标记）  
    55.        for waypoint in waypoints:  
    56.            p = Point（）  
    57.            p = waypoint.position  
    58.            self.markers.points.append（p）  
    59.  
    60.        # Publisher to manually control the robot （e.g. to stop it）  
    61.        # 发布TWist消息控制机器人  
    62.        self.cmd\_vel\_pub = rospy.Publisher（'cmd\_vel', Twist）  
    63.  
    64.        # Subscribe to the move\_base action server  
    65.        # 订阅move\_base[服务器](http://server.chinaitlab.com/)的消息  
    66.        self.move\_base = actionlib.SimpleActionClient（"move\_base", MoveBaseAction）  
    67.  
    68.        rospy.loginfo（"Waiting for move\_base action server…"）  
    69.  
    70.        # Wait 60 seconds for the action server to become available  
    71.        # 等待move\_base[服务器](http://server.chinaitlab.com/)建立  
    72.        self.move\_base.wait\_for\_server（rospy.Duration（60））  
    73.  
    74.        rospy.loginfo（"Connected to move base server"）  
    75.        rospy.loginfo（"Starting navigation test"）  
    76.  
    77.        # Initialize a counter to track waypoints  
    78.        # 初始化一个计数器，记录到达的顶点号  
    79.        i = 0  
    80.  
    81.        # Cycle through the four waypoints  
    82.        # 主循环，环绕通过四个定点  
    83.        while i < 4 and not rospy.is\_shutdown（）：  
    84.            # Update the marker display  
    85.            # 发布标记指示四个目标的位置，每个周期发布一起，确保标记可见  
    86.            self.marker\_pub.publish（self.markers）  
    87.  
    88.            # Intialize the waypoint goal  
    89.            # 初始化goal为MoveBaseGoal类型  
    90.            goal = MoveBaseGoal（）  
    91.  
    92.            # Use the map frame to define goal poses  
    93.            # 使用map的frame定义goal的frame id  
    94.            goal.target\_pose.header.frame\_id = 'map'  
    95.  
    96.            # Set the time stamp to "now"  
    97.            # 设置时间戳  
    98.            goal.target\_pose.header.stamp = rospy.Time.now（）  
    99.  
    100.            # Set the goal pose to the i-th waypoint  
    101.            # 设置目标位置是当前第几个导航点  
    102.            goal.target\_pose.pose = waypoints[i]  
    103.  
    104.            # Start the robot moving toward the goal  
    105.            # 机器人移动  
    106.            self.move（goal）  
    107.  
    108.            i += 1  
    109.  
    110.    def move（self, goal）：  
    111.            # Send the goal pose to the MoveBaseAction server  
    112.            # 把目标位置发送给MoveBaseAction的服务器  
    113.            self.move\_base.send\_goal（goal）  
    114.  
    115.            # Allow 1 minute to get there  
    116.            # 设定1分钟的时间限制  
    117.            finished\_within\_time = self.move\_base.wait\_for\_result（rospy.Duration（60））  
    118.  
    119.            # If we don't get there in time, abort the goal  
    120.            # 如果一分钟之内没有到达，放弃目标  
    121.            if not finished\_within\_time:  
    122.                self.move\_base.cancel\_goal（）  
    123.                rospy.loginfo（"Timed out achieving goal"）  
    124.            else:  
    125.                # We made it!  
    126.                state = self.move\_base.get\_state（）  
    127.                if state == GoalStatus.SUCCEEDED:  
    128.                    rospy.loginfo（"Goal succeeded!"）  
    129.  
    130.    def init\_markers（self）：  
    131.        # Set up our waypoint markers  
    132.        # 设置标记的尺寸  
    133.        marker\_scale = 0.2  
    134.        marker\_lifetime = 0 # 0 is forever  
    135.        marker\_ns = 'waypoints'  
    136.        marker\_id = 0  
    137.        marker\_color = {'r': 1.0, 'g': 0.7, 'b': 1.0, 'a': 1.0}  
    138.  
    139.        # Define a marker publisher.  
    140.        # 定义一个标记的发布者  
    141.        self.marker\_pub = rospy.Publisher（'waypoint\_markers', Marker）  
    142.  
    143.        # Initialize the marker points list.  
    144.        # 初始化标记点的列表  
    145.        self.markers = Marker（）  
    146.        self.markers.ns = marker\_ns  
    147.        self.markers.id = marker\_id  
    148.        self.markers.type = Marker.SPHERE\_LIST  
    149.        self.markers.action = Marker.ADD  
    150.        self.markers.lifetime = rospy.Duration（marker\_lifetime）  
    151.        self.markers.scale.x = marker\_scale  
    152.        self.markers.scale.y = marker\_scale  
    153.        self.markers.color.r = marker\_color['r']  
    154.        self.markers.color.g = marker\_color['g']  
    155.        self.markers.color.b = marker\_color['b']  
    156.        self.markers.color.a = marker\_color['a']  
    157.  
    158.        self.markers.header.frame\_id = 'map'  
    159.        self.markers.header.stamp = rospy.Time.now（）  
    160.        self.markers.points = list（）  
    161.  
    162.    def shutdown（self）：  
    163.        rospy.loginfo（"Stopping the robot…"）  
    164.        # Cancel any active goals  
    165.        self.move\_base.cancel\_goal（）  
    166.        rospy.sleep（2）  
    167.        # Stop the robot  
    168.        self.cmd\_vel\_pub.publish（Twist（））  
    169.        rospy.sleep（1）  
    170.  
    171.if \_\_name\_\_ == '\_\_main\_\_':  
    172.    try:  
    173.        MoveBaseSquare（）  
    174.    except rospy.ROSInterruptException:  
    175.        rospy.loginfo（"Navigation test finished."）

　　但是，在实际情况中，往往需要让机器人自动躲避障碍物。move\_base包的一个强大的功能就是可以在全局规划的过程中自动躲避障碍物，而不影响全局路径。障碍物可以是静态的(比如墙、桌子等)，也可以是动态的(比如人走过)。

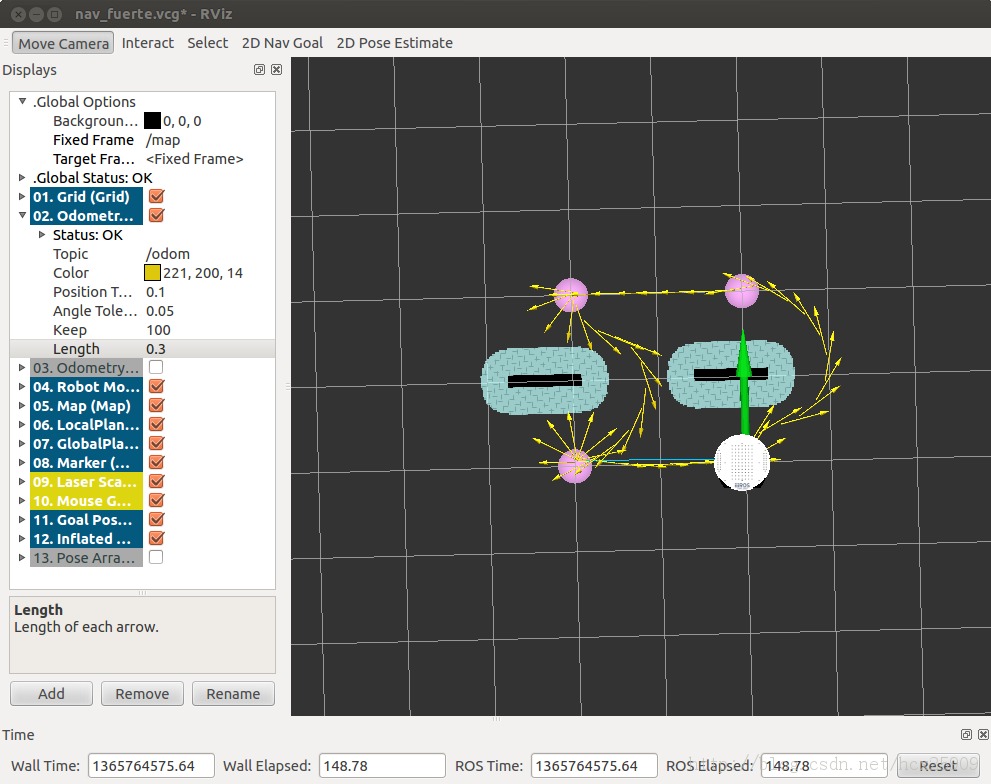
　　现在我们尝试在之前的正方形路径中加入障碍物。把之前运行fake\_move\_base\_blank\_map.launch的中断Ctrl-C掉，然后运行：

    01.roslaunch rbx1\_nav fake\_move\_base\_obstacle.launch

　　然后就会看到在rviz中出现了障碍物。然后在运行之前走正方形路线的代码：

    01.rosrun rbx1\_nav move\_base\_square.py

　　这回我们可以看到，在全局路径规划的时候，机器人已经将障碍物绕过去了，下过如下图：



在上图中，黑色的线是障碍物，周围浅色的椭圆形是根据配置文件中的inflation\_radius参数计算出来的[安全](http://security.chinaitlab.com/)缓冲区。全局规划的路径基本已经是最短路径了。在仿真的过程中也可以动态重配置那四个配置文件，修改仿真参数。

**操作系统探索总结（十五）amcl（导航与定位）**

　　在理解了move\_base的基础上，我们开始机器人的定位与导航。gmaping包是用来生成地图的，需要使用实际的机器人获取激光或者深度数据，所以我们先在已有的地图上进行导航与定位的仿真。

　　amcl是移动机器人二维环境下的概率定位系统。它实现了自适应(或kld采样)的蒙特卡罗定位方法，其中针对已有的地图使用粒子滤波器跟踪一个机器人的姿态。

**一、**[**测试**](http://softtest.chinaitlab.com/)

　　首先运行机器人节点：

    01.roslaunch rbx1\_bringup fake\_turtlebot.launch

　　然后运行amcl节点，使用[测试](http://softtest.chinaitlab.com/)地图：

    01.roslaunch rbx1\_nav fake\_amcl.launch map:=test\_map.yaml

　　可以看一下fake\_amcl.launch这个文件的内容：

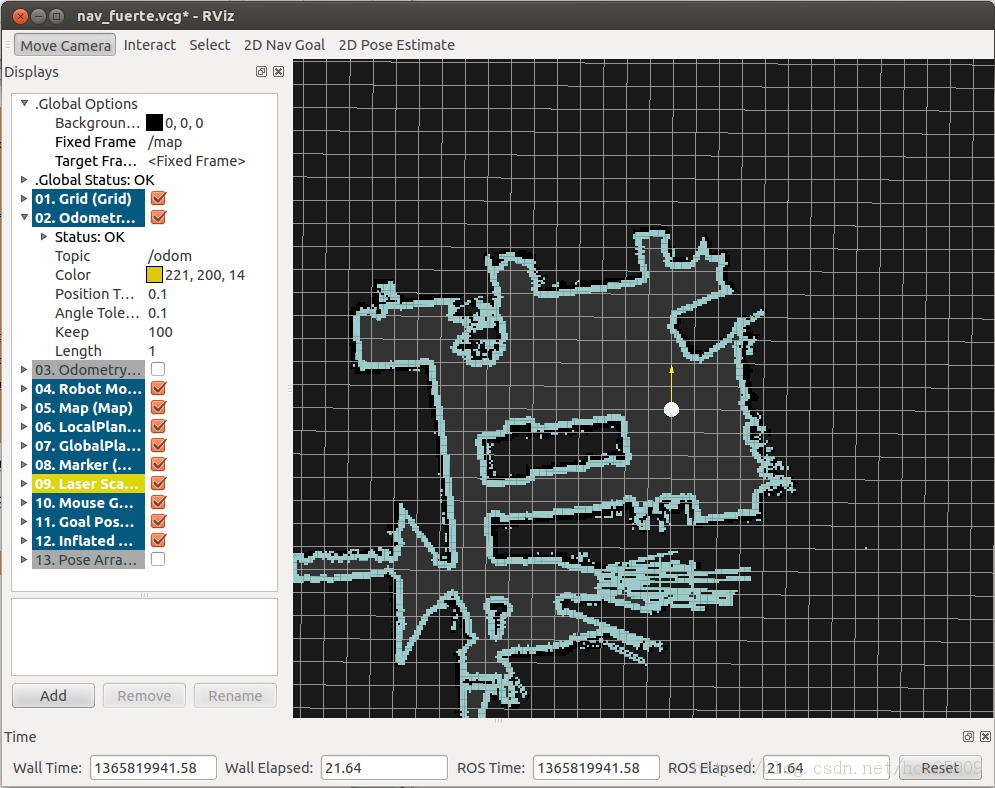
    01.<launch>  
    02.  <!-- Set the name of the map yaml file: can be overridden on the command line. -->  
    03.  <arg name="map" default="test\_map.yaml" />  
    04.  <!-- Run the map server with the desired map -->  
    05.  <node name="map\_server" pkg="map\_server" type="map\_server" args="$（find rbx1\_nav）/maps/$（arg map）"/>  
    06.  <!-- The move\_base node -->  
    07.  <include file="$（find rbx1\_nav）/launch/fake\_move\_base.launch" />  
    08.  
    09.  <!-- Run fake localization compatible with AMCL output -->  
    10.  <node pkg="fake\_localization" type="fake\_localization"  name="fake\_localization" output="screen" />  
    11.  <!-- For fake localization we need static transforms between /odom and /map and /map and /world -->  
    12.  <node pkg="tf" type="static\_transform\_publisher" name="odom\_map\_broadcaster"  
    13.args="0 0 0 0 0 0 /odom /map 100" />  
    14.</launch>

　　这个lanuch文件作用是加载地图，并且调用fake\_move\_base.launch文件打开move\_base节点并加载配置文件，最后运行amcl。

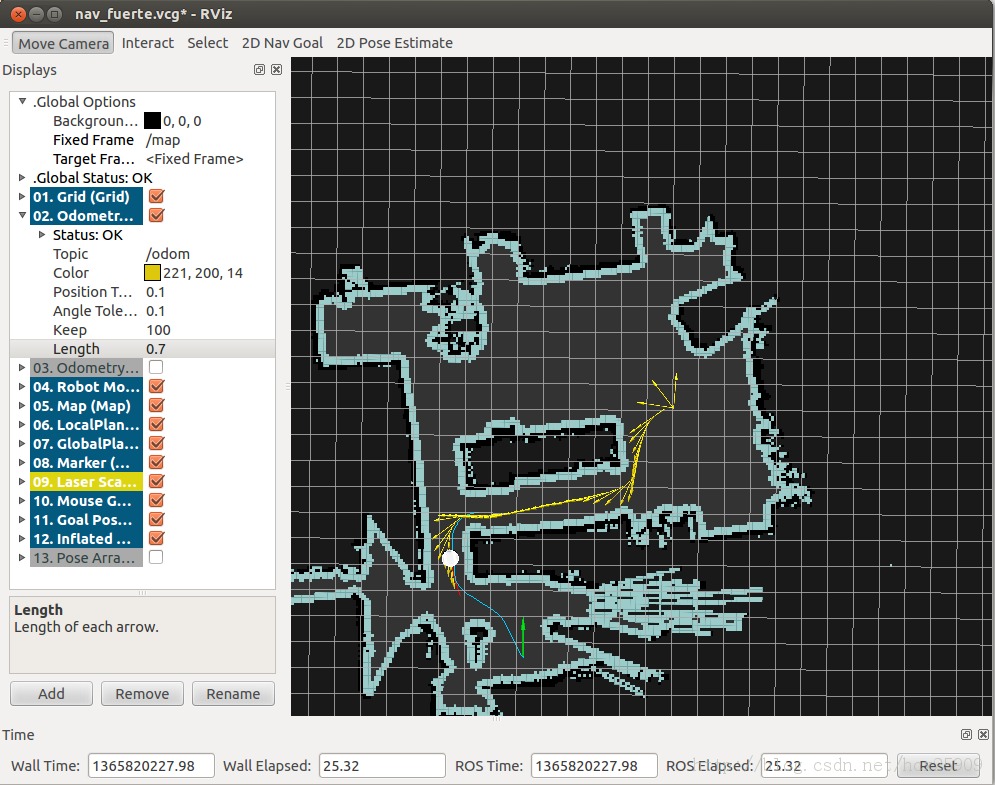
　　然后运行rviz：

    01.rosrun rviz rviz -d `rospack find rbx1\_nav`/nav\_fuerte.vcg

　　这时在rvoiz中就应该显示出了地图和机器人：



　　现在就可以通过rviz在地图上选择目标位置了，然后就会看到机器人自动规划出一条全局路径，并且导航前进：



**二、自主导航**

　　在实际应用中，我们往往希望机器人能够自主进行定位和导航，不需要认为的干预，这样才更智能化。在这一节的测试中，我们让目标点在地图中随机生成，然后机器人自动导航到达目标。

　　这里运行的主要文件是：fake\_nav\_test.launch，让我们来看一下这个文件的内容：

    01.<launch>  
    02.  
    03.  <param name="use\_sim\_time" value="false" />  
    04.  
    05.  <!-- Start the ArbotiX controller -->  
    06.  <include file="$（find rbx1\_bringup）/launch/fake\_turtlebot.launch" />  
    07.  
    08.  <!-- Run the map server with the desired map -->  
    09.  <node name="map\_server" pkg="map\_server" type="map\_server" args="$（find rbx1\_nav）/maps/test\_map.yaml"/>  
    10.  
    11.  <!-- The move\_base node -->  
    12.  <node pkg="move\_base" type="move\_base" respawn="false" name="move\_base" output="screen">  
    13.    <rosparam file="$（find rbx1\_nav）/config/fake/costmap\_common\_params.yaml" command="load" ns="global\_costmap" />  
    14.    <rosparam file="$（find rbx1\_nav）/config/fake/costmap\_common\_params.yaml" command="load" ns="local\_costmap" />  
    15.    <rosparam file="$（find rbx1\_nav）/config/fake/local\_costmap\_params.yaml" command="load" />  
    16.    <rosparam file="$（find rbx1\_nav）/config/fake/global\_costmap\_params.yaml" command="load" />  
    17.    <rosparam file="$（find rbx1\_nav）/config/fake/base\_local\_planner\_params.yaml" command="load" />  
    18.    <rosparam file="$（find rbx1\_nav）/config/nav\_test\_params.yaml" command="load" />  
    19.  </node>  
    20.  
    21.  <!-- Run fake localization compatible with AMCL output -->  
    22.  <node pkg="fake\_localization" type="fake\_localization" name="fake\_localization" output="screen" />  
    23.  
    24.  <!-- For fake localization we need static transform between /odom and /map -->  
    25.  <node pkg="tf" type="static\_transform\_publisher" name="map\_odom\_broadcaster" args="0 0 0 0 0 0 /map /odom 100" />  
    26.  
    27.  <!-- Start the navigation test -->  
    28.  <node pkg="rbx1\_nav" type="nav\_test.py" name="nav\_test" output="screen">  
    29.    <param name="rest\_time" value="1" />  
    30.    <param name="fake\_test" value="true" />  
    31.  </node>  
    32.  
    33.</launch>

　　这个lanuch的功能比较多：

　　(1) 加载机器人驱动

　　(2) 加载地图

　　(3) 启动move\_base节点，并且加载配置文件

　　(4) 运行amcl节点

　　(5) 然后加载nav\_test.py执行文件，进行随机导航

　　相当于是把我们之前实验中的多个lanuch文件合成了一个文件。

　　现在开始进行测试，先运行ROS：

    01.roscore

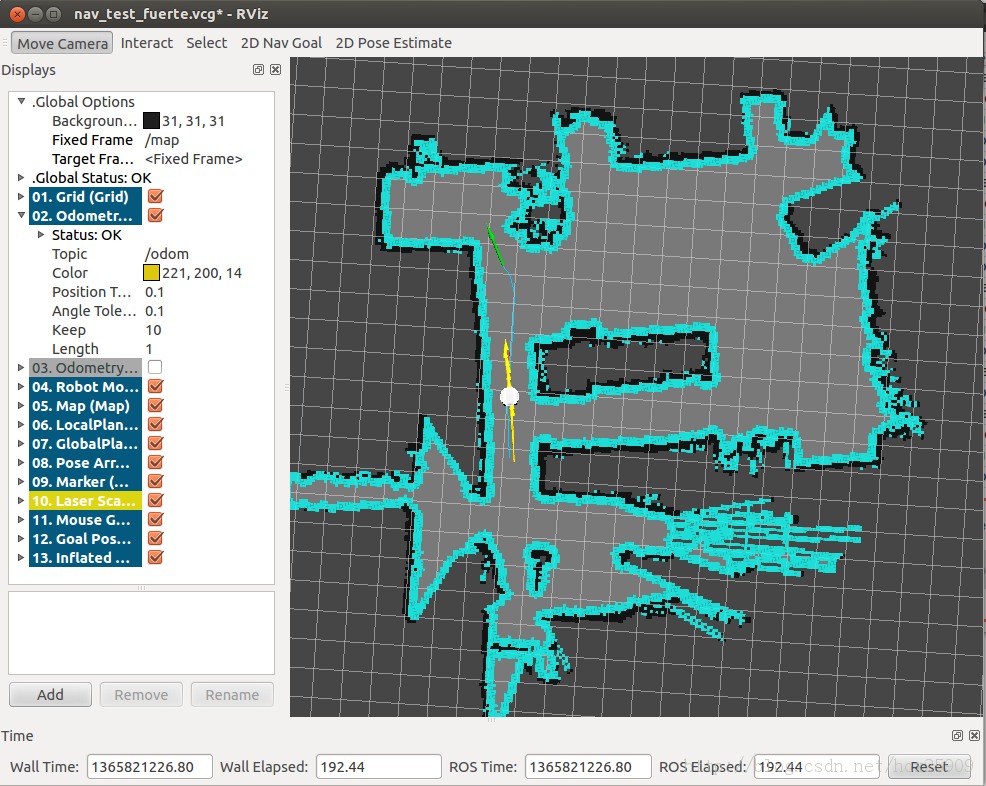
　　然后我们运行一个监控的窗口，可以实时看到机器人发送的数据：

    01.rxconsole

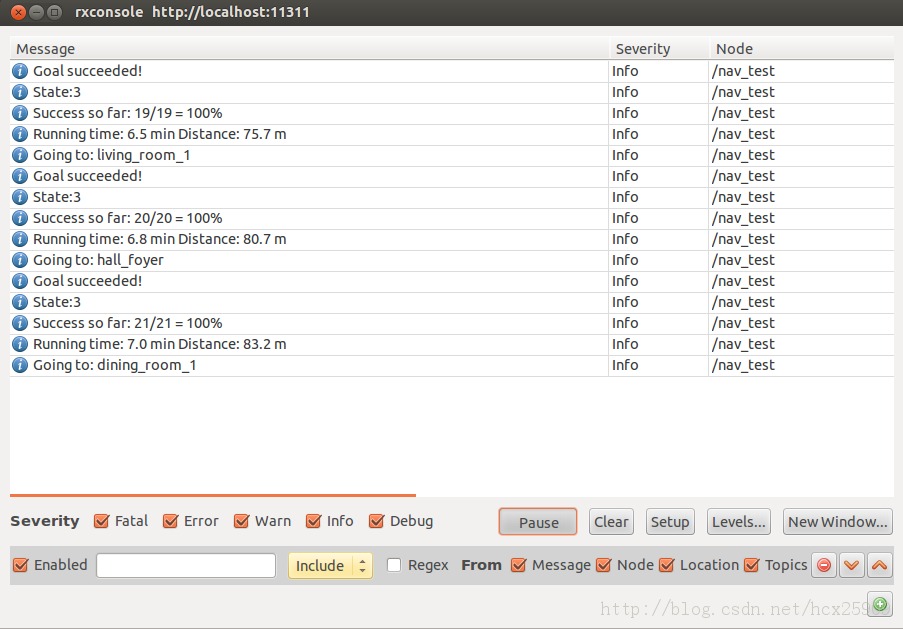
　　接着运行lanuch文件，并且在一个新的终端中打开rviz：

    01.roslaunch rbx1\_nav fake\_nav\_test.launch  
    02.（打开新终端）  
    03.rosrun rviz rviz -d `rospack find rbx1\_nav`/nav\_test\_fuerte.vcg

　　好了，此时就看到了机器人已经放在地图当中了。然后我们点击rviz上的“2D Pose Estimate”按键，然后左键在机器人上单击，让绿色的箭头和黄色的箭头重合，机器人就开始随机选择目标导航了：



　　在监控窗口中，我们可以看到机器人发送的状态信息：



　　其中包括距离信息、状态信息、目标的编号、成功率和速度等信息。

**三、导航代码分析**

    01.#!/usr/bin/env python  
    02.  
    03.import roslib; roslib.load\_manifest（'rbx1\_nav'）  
    04.import rospy  
    05.import actionlib  
    06.from actionlib\_msgs.msg import \*  
    07.from geometry\_msgs.msg import Pose, PoseWithCovarianceStamped, Point, Quaternion, Twist  
    08.from move\_base\_msgs.msg import MoveBaseAction, MoveBaseGoal  
    09.from random import sample  
    10.from math import pow, sqrt  
    11.  
    12.class NavTest（）：  
    13.    def \_\_init\_\_（self）：  
    14.        rospy.init\_node（'nav\_test', anonymous=True）  
    15.  
    16.        rospy.on\_shutdown（self.shutdown）  
    17.  
    18.        # How long in seconds should the robot pause at each location?  
    19.        # 在每个目标位置暂停的时间  
    20.        self.rest\_time = rospy.get\_param（"~rest\_time", 10）  
    21.  
    22.        # Are we running in the fake simulator?  
    23.        # 是否仿真？  
    24.        self.fake\_test = rospy.get\_param（"~fake\_test", False）  
    25.  
    26.        # Goal state return values  
    27.        # 到达目标的状态  
    28.        goal\_states = ['PENDING', 'ACTIVE', 'PREEMPTED',  
    29.                       'SUCCEEDED', 'ABORTED', 'REJECTED',  
    30.                       'PREEMPTING', 'RECALLING', 'RECALLED',  
    31.                       'LOST']  
    32.  
    33.        # Set up the goal locations. Poses are defined in the map frame.  
    34.        # An easy way to find the pose coordinates is to point-and-click  
    35.        # Nav Goals in RViz when running in the simulator.  
    36.        # Pose coordinates are then displayed in the terminal  
    37.        # that was used to launch RViz.  
    38.        # 设置目标点的位置  
    39.        # 如果想要获得某一点的坐标，在rviz中点击 2D Nav Goal 按键，然后单机地图中一点  
    40.        # 在终端中就会看到坐标信息  
    41.        locations = dict（）  
    42.  
    43.        locations['hall\_foyer'] = Pose（Point（0.643, 4.720, 0.000）， Quaternion（0.000, 0.000, 0.223, 0.975））  
    44.        locations['hall\_kitchen'] = Pose（Point（-1.994, 4.382, 0.000）， Quaternion（0.000, 0.000, -0.670, 0.743））  
    45.        locations['hall\_bedroom'] = Pose（Point（-3.719, 4.401, 0.000）， Quaternion（0.000, 0.000, 0.733, 0.680））  
    46.        locations['living\_room\_1'] = Pose（Point（0.720, 2.229, 0.000）， Quaternion（0.000, 0.000, 0.786, 0.618））  
    47.        locations['living\_room\_2'] = Pose（Point（1.471, 1.007, 0.000）， Quaternion（0.000, 0.000, 0.480, 0.877））  
    48.        locations['dining\_room\_1'] = Pose（Point（-0.861, -0.019, 0.000）， Quaternion（0.000, 0.000, 0.892, -0.451））  
    49.  
    50.        # Publisher to manually control the robot （e.g. to stop it）  
    51.        # 发布控制机器人的消息  
    52.        self.cmd\_vel\_pub = rospy.Publisher（'cmd\_vel', Twist）  
    53.  
    54.        # Subscribe to the move\_base action server  
    55.        # 订阅move\_base[服务器](http://server.chinaitlab.com/)的消息  
    56.        self.move\_base = actionlib.SimpleActionClient（"move\_base", MoveBaseAction）  
    57.  
    58.        rospy.loginfo（"Waiting for move\_base action server…"）  
    59.  
    60.        # Wait 60 seconds for the action server to become available  
    61.        # 60s等待时间限制  
    62.        self.move\_base.wait\_for\_server（rospy.Duration（60））  
    63.  
    64.        rospy.loginfo（"Connected to move base server"）  
    65.  
    66.        # A variable to hold the initial pose of the robot to be set by  
    67.        # the user in RViz  
    68.        # 保存机器人的在rviz中的初始位置  
    69.        initial\_pose = PoseWithCovarianceStamped（）  
    70.  
    71.        # Variables to keep track of success rate, running time,  
    72.        # and distance traveled  
    73.        # 保存成功率、运行时间、和距离的变量  
    74.        n\_locations = len（locations）  
    75.        n\_goals = 0  
    76.        n\_successes = 0  
    77.        i = n\_locations  
    78.        distance\_traveled = 0  
    79.        start\_time = rospy.Time.now（）  
    80.        running\_time = 0  
    81.        location = ""  
    82.        last\_location = ""  
    83.  
    84.        # Get the initial pose from the user  
    85.        # 获取初始位置（仿真中可以不需要）  
    86.        rospy.loginfo（"\*\*\* Click the 2D Pose Estimate button in RViz to set the robot's initial pose…"）  
    87.        rospy.wait\_for\_message（'initialpose', PoseWithCovarianceStamped）  
    88.        self.last\_location = Pose（）  
    89.        rospy.Subscriber（'initialpose', PoseWithCovarianceStamped, self.update\_initial\_pose）  
    90.  
    91.        # Make sure we have the initial pose  
    92.        # 确保有初始位置  
    93.        while initial\_pose.header.stamp == "":  
    94.            rospy.sleep（1）  
    95.  
    96.        rospy.loginfo（"Starting navigation test"）  
    97.  
    98.        # Begin the main loop and run through a sequence of locations  
    99.        # 开始主循环，随机导航  
    100.        while not rospy.is\_shutdown（）：  
    101.            # If we've gone through the current sequence,  
    102.            # start with a new random sequence  
    103.            # 如果已经走完了所有点，再重新开始排序  
    104.            if i == n\_locations:  
    105.                i = 0  
    106.                sequence = sample（locations, n\_locations）  
    107.                # Skip over first location if it is the same as  
    108.                # the last location  
    109.                # 如果最后一个点和第一个点相同，则跳过  
    110.                if sequence[0] == last\_location:  
    111.                    i = 1  
    112.  
    113.            # Get the next location in the current sequence  
    114.            # 在当前的排序中获取下一个目标点  
    115.            location = sequence[i]  
    116.  
    117.            # Keep track of the distance traveled.  
    118.            # Use updated initial pose if available.  
    119.            # 跟踪形式距离  
    120.            # 使用更新的初始位置  
    121.            if initial\_pose.header.stamp == "":  
    122.                distance = sqrt（pow（locations[location].position.x -  
    123.                                    locations[last\_location].position.x, 2） +  
    124.                                pow（locations[location].position.y -  
    125.                                    locations[last\_location].position.y, 2））  
    126.            else:  
    127.                rospy.loginfo（"Updating current pose."）  
    128.                distance = sqrt（pow（locations[location].position.x -  
    129.                                    initial\_pose.pose.pose.position.x, 2） +  
    130.                                pow（locations[location].position.y -  
    131.                                    initial\_pose.pose.pose.position.y, 2））  
    132.                initial\_pose.header.stamp = ""  
    133.  
    134.            # Store the last location for distance calculations  
    135.            # [存储](http://www.storworld.com/)上一次的位置，计算距离  
    136.            last\_location = location  
    137.  
    138.            # Increment the counters  
    139.            # 计数器加1  
    140.            i += 1  
    141.            n\_goals += 1  
    142.  
    143.            # Set up the next goal location  
    144.            # 设定下一个目标点  
    145.            self.goal = MoveBaseGoal（）  
    146.            self.goal.target\_pose.pose = locations[location]  
    147.            self.goal.target\_pose.header.frame\_id = 'map'  
    148.            self.goal.target\_pose.header.stamp = rospy.Time.now（）  
    149.  
    150.            # Let the user know where the robot is going next  
    151.            # 让用户知道下一个位置  
    152.            rospy.loginfo（"Going to: " + str（location））  
    153.  
    154.            # Start the robot toward the next location  
    155.            # 向下一个位置进发  
    156.            self.move\_base.send\_goal（self.goal）  
    157.  
    158.            # Allow 5 minutes to get there  
    159.            # 五分钟时间限制  
    160.            finished\_within\_time = self.move\_base.wait\_for\_result（rospy.Duration（300））  
    161.  
    162.            # Check for success or failure  
    163.            # 查看是否成功到达  
    164.            if not finished\_within\_time:  
    165.                self.move\_base.cancel\_goal（）  
    166.                rospy.loginfo（"Timed out achieving goal"）  
    167.            else:  
    168.                state = self.move\_base.get\_state（）  
    169.                if state == GoalStatus.SUCCEEDED:  
    170.                    rospy.loginfo（"Goal succeeded!"）  
    171.                    n\_successes += 1  
    172.                    distance\_traveled += distance  
    173.                    rospy.loginfo（"State:" + str（state））  
    174.                else:  
    175.                  rospy.loginfo（"Goal failed with error code: " + str（goal\_states[state]））  
    176.  
    177.            # How long have we been running?  
    178.            # 运行所用时间  
    179.            running\_time = rospy.Time.now（） - start\_time  
    180.            running\_time = running\_time.secs / 60.0  
    181.  
    182.            # Print a summary success/failure, distance traveled and time elapsed  
    183.            # 输出本次导航的所有信息  
    184.            rospy.loginfo（"Success so far: " + str（n\_successes） + "/" +  
    185.                          str（n\_goals） + " = " +  
    186.                          str（100 \* n\_successes/n\_goals） + "%"）  
    187.            rospy.loginfo（"Running time: " + str（trunc（running\_time, 1）） +  
    188.                          " min Distance: " + str（trunc（distance\_traveled, 1）） + " m"）  
    189.            rospy.sleep（self.rest\_time）  
    190.  
    191.    def update\_initial\_pose（self, initial\_pose）：  
    192.        self.initial\_pose = initial\_pose  
    193.  
    194.    def shutdown（self）：  
    195.        rospy.loginfo（"Stopping the robot…"）  
    196.        self.move\_base.cancel\_goal（）  
    197.        rospy.sleep（2）  
    198.        self.cmd\_vel\_pub.publish（Twist（））  
    199.        rospy.sleep（1）  
    200.  
    201.def trunc（f, n）：  
    202.    # Truncates/pads a float f to n decimal places without rounding  
    203.    slen = len（'%.\*f' % （n, f））  
    204.    return float（str（f）[:slen]）  
    205.  
    206.if \_\_name\_\_ == '\_\_main\_\_':  
    207.    try:  
    208.        NavTest（）  
    209.        rospy.spin（）  
    210.    except rospy.ROSInterruptException:  
    211.        rospy.loginfo（"AMCL navigation test finished."）