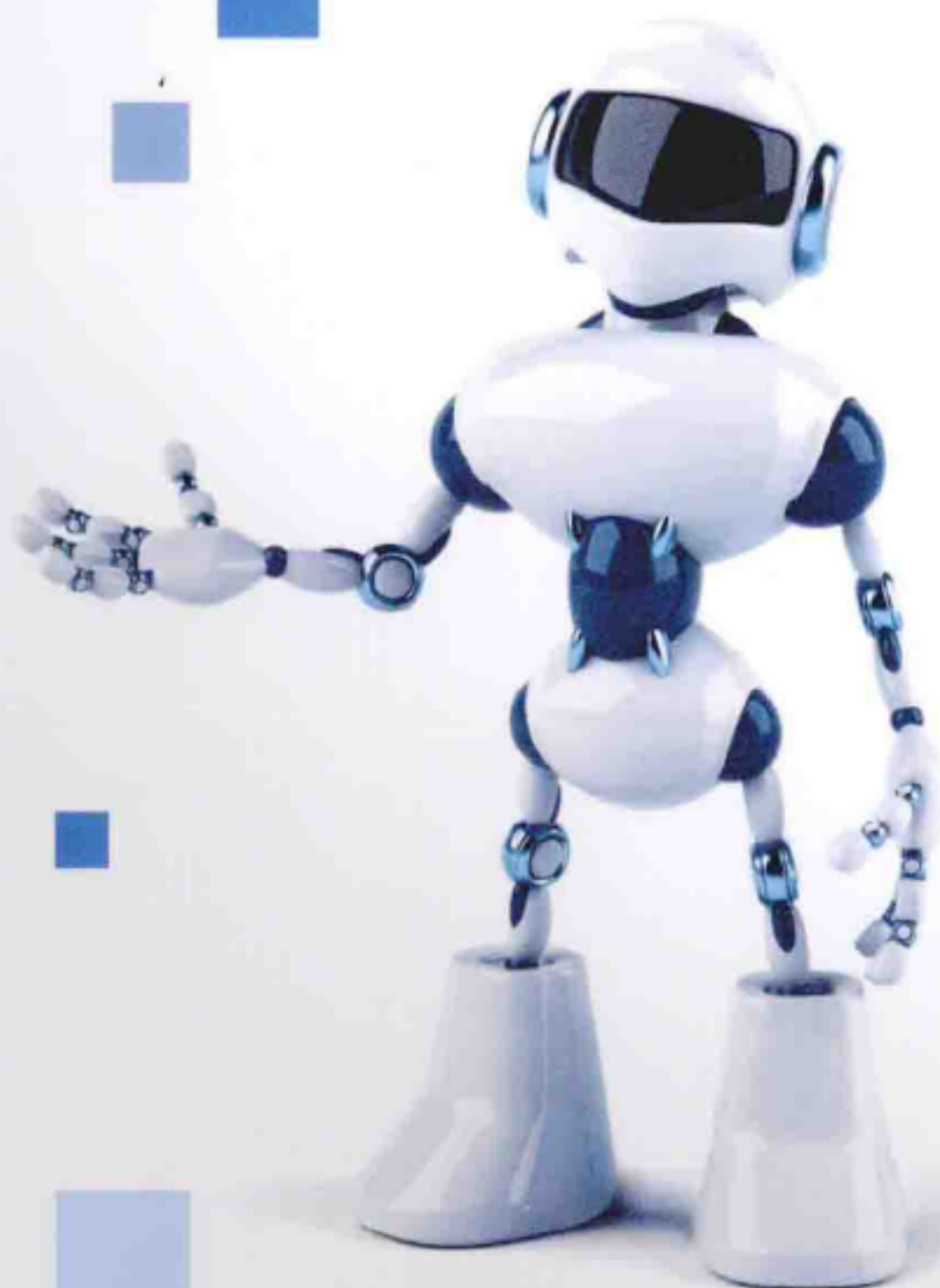


国内首本引进的ROS机器人程序设计译著，让你全面了解 ROS系统的各种工具。

提供了各种实际的示例代码供读者学习和理解ROS的软件框架。

本书可以帮助读者从对ROS一无所知到能够通过ROS系统完成小型机器人系统的开发和编程工作。



ROS机器人程序设计

Learning ROS for Robotics Programming

[西班牙] Aaron Martinez Enrique Fernández 著

刘品杰 译



机械工业出版社
China Machine Press

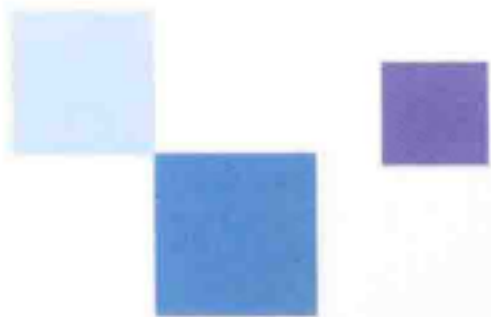
提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

备用QQ:2404062482



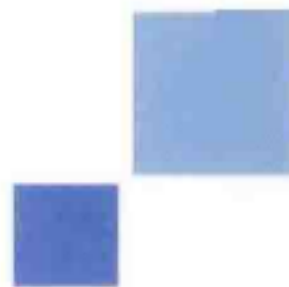
读者对象

本书主要针对希望学习机器人制作和设计，尤其是想要设计自己的机器人、真正热爱机器人的爱好者。本书将会为你提供让你的机器人可维护可升级，并且能够相互分享的途径。为了更好地学习本书的内容，你最好具备一定的C++程序设计基础以及GNU/Linux系统和计算机科学知识。而你并不需要对ROS有任何了解，因为本书正是要介绍这些基本知识和技能。还有，你最好能够了解一定的版本控制系统的知识，如SVN或GIT。在本书中会经常使用它们。

作者简介

Aaron Martinez 是一位计算机工程师、企业家和数字化制造专家。他硕士毕业于拉斯帕尔马斯大学的IUCTC（科学与网络技术研究所）。之后，参与过拉斯帕尔马斯大学AVORA项目。在这个项目中，他负责设计AUV（自主式水下机器人），并在意大利参加了欧洲学生自主式水下挑战（SAUC-E）。

Enrique Fernández 是一名计算机工程师和机器人专家。他硕士毕业于拉斯帕尔马斯大学智能系统与计算工程学院。在2012年参加了欧洲学生自主式水下挑战（SAUC-E），并作为合作者参加了2013年的比赛。2012年，他因开发水下云台视觉系统而获奖。现在，他是Pal-Robotics实验室的SLAM工程师。Enrique在博士学习期间发表了数篇学术论文和专著。其中，有两篇论文在2011年被国际机器人与自动化会议（ICRA 2011）所收录。



投稿热线: (010) 88379604
客服热线: (010) 88378991 88361066
购书热线: (010) 68326294 88379649 68995259

华章网站: www.hzbook.com
网上购书: www.china-pub.com
数字阅读: www.hzmedia.com.cn

上架指导: 计算机/程序设计/Java

ISBN 978-7-111-47396-1



9 787111 473961 >

定价: 59.00元

A collection of approximately 20 squares of various shades of gray and black, scattered across the upper half of the page.

ROS机器人程序设计

Learning ROS for Robotics Programming

[西班牙] Aaron Martinez Enrique Fernández 著

刘品杰 译



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

ROS 机器人程序设计 / (西) 马丁内斯 (Martinez, A.) 等著; 刘品杰译. —北京: 机械工业出版社, 2014.8

(电子与嵌入式系统设计丛书)

书名原文: Learning ROS for Robotics Programming

ISBN 978-7-111-47396-1

I. R… II. ① 马… ② 刘… III. 机器人—程序设计 IV. TP242

中国版本图书馆 CIP 数据核字 (2014) 第 161738 号

本书版权登记号: 图字: 01-2014-1838

Aaron Martinez, Enrique Fernández: Learning ROS for Robotics Programming (ISBN: 978-1-78216-144-8)

Copyright © 2013 Packt Publishing. First published in the English language under the title “Learning ROS for Robotics Programming”.

All rights reserved.

Chinese simplified language edition published by China Machine Press.

Copyright © 2014 by China Machine Press.

本书中文简体字版由 Packt Publishing 授权机械工业出版社独家出版。未经出版者书面许可, 不得以任何方式复制或抄袭本书内容。

ROS 是一个机器人操作系统框架, 现今已有数百个研究团体和公司将其应用在机器人技术产业中。本书是一本 ROS 的入门手册, 从 ROS 系统的安装到主要功能包里各项工具的使用方法如先进的计算机视觉和导航工具等都进行细致的介绍和说明, 并且提供全面的相关示例代码及详细的解释, 读者可以按照本书指导进行相关练习。本书适用于机械、自动化、计算机等相关专业高年级本科生及研究生学习, 也适合于准备学习 ROS 的科研人员及企业研发人员学习和使用。

ROS 机器人程序设计

[西班牙] Aaron Martinez 等著

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 朱 瑛

责任校对: 董纪丽

印 刷: 藁城市京瑞印刷有限公司

版 次: 2014 年 9 月第 1 版第 1 次印刷

开 本: 186mm × 240mm 1/16

印 张: 15

书 号: ISBN 978-7-111-47396-1

定 价: 59.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzjsj@hzbook.com

版权所有 · 侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

The Translator's Words | 译者序

在过去的几十年里，机器人主要是自动化或机械专业的研究领域，计算机往往作为辅助仿真的工具，机器人的程序设计也往往仅限于使用诸如 `matlab` 机器人工具箱之类的仿真工具。今天很多大型 IT 企业都投入大量资源开发了机器人相关的软件开发工具，例如 `Visual Studio` 的 `Robotics Developer Studio`，这些软件工具的目的是颠覆传统的机器人开发和设计模式。机器人操作系统就是基于这个目的产生的。它们能够帮助开发人员在原有机机器人的基础上不断地进行二次开发，深入拓展机器人技术。而其中最具代表性的正是开源机器人操作系统 `ROS (Robot Operating System)`。

与其说 `ROS` 是一个操作系统，不如说它是一种分布式模块化的开源软件框架。它借用标准的 `TCP(UDP)/IP` 协议实现了系统内部各个节点之间的通信，网络化的接口允许将第三方组件泛化成为其操作系统的一部分。如果你真正使用过 `ROS`，就会为其无所不包的开放性而感到惊讶。正因如此，`ROS` 社区集中了全世界顶尖的机器人研究人员。今天，`ROS` 能够无缝集成现在已知的大部分机器人操作系统；能满足最高的实时性与可靠性要求；通过使用 `EtherCAT` 等高速总线能以每秒 `Gb` 级的速度传输视频数据；能够通过集成其他开源组件及增强现实技术提供多种仿真环境；能够通过多种接口与各种硬件设备或传感器进行通信；能够替你计算繁琐的正逆运动学，通过标定和配置轻松完成视觉伺服等高难度任务；你可以方便地移植其他机器人上已经测试过的各类算法和代码，或引入其他开源库。

虽然 `ROS` 大幅度降低了开发和应用机器人的难度，但是考虑到机器人领域无所不包，想要学习和使用 `ROS` 还是有几点需要注意的。首先，你应该学习过 `C/C++` 程序设计语言，能够使用 `Linux` 编译和运行程序，这是阅读本书的基本要求。当然，如果你是一个编程新手，也可以跟着本书示例在实践中学习 `Linux` 和编程知识。其次，在学习和使用 `ROS` 的时候，往往还需要懂得程序设计之外的很多机器人知识，尤其是数学知识。但由于机器人领域过于宽泛，因此在学习过程中最好能够专注于某一方面的相关知识，不要贪多求全。再次，`ROS` 是一个开源软件，因此我们也要保持一种开放的心态。虽然本书能帮助你入门相对轻松一些，但这并不会彻底优化你的学习曲线。如果想要学好 `ROS`，一定要去 `www.ros.org` 上学习最

新的资料,尤其是 ROS wiki。最顶尖的 ROS 国际会议是 ROSCON,你可以到 Google 或者 YouTube 上搜索相关内容,里面有 ROS 最新的发展与介绍。最后,本书英文版虽然是 2013 年 9 月出版,但 ROS 和 Linux 一样半年出一个新的版本,因此书中并没有介绍最新的 ROS Hydro 和 ROS Groovy。其实这些并不妨碍你对 ROS 的学习,没有谁为了学习 Linux 编程而非要用最新出版的图书,因为软件的基本框架和组织方式不会发生改变,而且支持 ROS 的机器人硬件并不一定支持最新的 ROS。ROS Hydro 和 ROS Groovy 主要对 ROS 原有的编译系统和部分功能包有所升级和替换,如果有兴趣的话,建议在学习完本书之后尚有余力的情况下,可以通过 wiki 自学 ROS Hydro 或 ROS Groovy。

本书的两位作者并不是以英语为母语的,因此英文原著中存在着较多的语法错误和校对错误。虽然在翻译过程中参考了原著的勘误和所附代码,考虑到译者的水平亦有限,因此仍然不免存在错误,还请大家海涵。在本书之前,ROS 的中文译著甚少,因此有些专业名词的翻译亦有待商榷。为了方便读者学习,译者共享了一个 VMware 9.0 下的虚拟机,里面包含完整安装的 Vbuntu 12.04 和 ROS Fuerte(下载地址是 <http://pan.baidu.com/s/1nt6uH9R>)。这样,使用 VMware 虚拟机的读者可以下载并在 VMware 中运行,然后直接学习第 2 章到第 8 章的内容。关于本书中的各种问题,欢迎大家与我联系,我的邮箱是 liupinjie@gmail.com。

虽然 ROS 是机器人技术,但是其分布式的节点布置同样满足物联网及很多工业应用的需求,同时也有很多外国企业利用它进行机床、船舶、汽车或自动化系统的制造。记住,ROS 代表了未来的智能化时代,赋予了我们新的创造与改变世界的能力。

刘品杰

2014 年 4 月于北京

Preface | 前言

本书概括性地介绍了 ROS 系统的各种工具。ROS 是一个先进的机器人操作系统框架，现今已有数百个研究团体和公司将其应用在机器人技术产业中。对于机器人技术的非专业人士来说，它也相对容易上手。在本书中，你将了解如何安装 ROS，如何开始使用 ROS 的基本工具，以及如何最终应用先进的计算机视觉和导航工具。

在阅读本书的过程中无需使用任何特殊的设备。书中每一章都附带了一系列的源代码示例和教程，你可以在自己的计算机上运行。这是你唯一需要做的事情。当然，我们还会告诉你如何使用硬件，这样，你可以将你的算法应用到现实环境中。我们在选择设备时特意去选择一些业余用户负担得起的设备，并同时涵盖了在机器人研究中最典型的传感器或执行机构。

最后，由于 ROS 系统的存在使得整个机器人具备在虚拟环境中工作的能力。你将学习如何创建自己的机器人并结合功能强大的导航功能包集。此外如果使用 Gazebo 仿真环境，你将能够在虚拟环境中运行一切。我们会在本书的结尾提供一个能够在 ROS 虚拟环境中进行模拟试验的机器人列表。你将发现你已经可以与机器人一起工作，并理解其背后的原理。

主要内容

第 1 章简单介绍安装 ROS 系统的方法，同时还介绍 ROS 不同版本的安装包的安装，本书使用的是 ROS Fuerte。这一章还会说明如何从 Debian 软件包安装或从源代码进行编译安装，以及在虚拟机中安装。

第 2 章涉及 ROS 框架及相关的概念和工具。该章介绍节点、主题和服务，以及如何使用它们，还将通过一系列示例来说明如何调试一个节点或通过可视化方法直观地查看通过主题所发布的消息。

第 3 章进一步展示 ROS 强大的调试工具，以及通过对节点主题的图形化可以将节点间的通信数据可视化。ROS 提供一个日志记录 API，允许轻松地诊断节点的问题。事实上在使用过程中，我们会看到一些功能强大的图形化工具如 rxconsole 和 rxgraph，以及可视化接口

如 `rxplot` 和 `rviz`。最后介绍如何使用 `rosvbag` 和 `rxrbag` 记录和回放消息。

第 4 章介绍 ROS 系统与真实世界如何连接。这一章介绍在 ROS 下使用的一些常见传感器和执行机构，如激光雷达、伺服电动机、摄像头、RGB-D 传感器等设备。此外，还会解释如何使用嵌入式系统与微控制器，例如非常流行的 `Arduino`。

第 5 章介绍我们在 ROS 系统中实现机器人的第一步是在 ROS 中创建一个机器人模型，包括在 `Gazebo` 仿真环境中如何从头开始对一个机器人进行建模和仿真，并使其在仿真环境中运行。这是后续学习如何使用 ROS 的导航功能包集和其他工具的前提条件。

第 6 章介绍 ROS 对摄像头和计算机视觉任务的支持。首先使用 `FireWire` 和 `USB` 摄像头驱动程序将摄像头连接到计算机并采集图像。然后，你就可以使用 ROS 的标定工具标定你的摄像头。我们会详细介绍和说明什么是图像管道，学习如何使用集成了 `OpenCV` 的多个机器视觉 API。最后，安装并使用一个视觉测距软件。

第 7 章是本书关于 ROS 导航功能包集的两章中的第 1 章。介绍如何对你的机器人进行使用导航功能包集所需的初始化配置。然后用几个例子对导航功能包集进行说明。

第 8 章延续第 7 章的内容，介绍如何使用导航功能包集使我们的机器人有效地自主导航。本章介绍使用 ROS 的 `Gazebo` 仿真环境和 `rviz` 创建一个虚拟的环境，并在其中构建地图、定位我们的机器人并做路径规划与避障。

第 9 章结合前面几章所学的内容，介绍能够支持 ROS 并使用 `Gazebo` 仿真环境的一些机器人。在该章中，你将看到如何在仿真环境中运行这些机器人，并执行几项本书中介绍过的任务，尤其是与导航功能包集相关的。

预备知识

我们写作本书的目的是希望尽可能让每位读者都可以完成本书的学习并运行示例代码。基本上，你只需要在计算机上安装一个 `Linux` 发行版。虽然每个 `Linux` 发行版本应该都能使用，但还是建议你使用 `Ubuntu` 的最新版。这样你可以根据第 1 章的内容安装 `ROS Fuerte`。对于 ROS 的这一版本，你将需要 `Ubuntu 12.10` 之前的版本，因为之后的版本已经不再支持 `Fuerte` 了。

对于硬件要求，一般来说任何台式计算机或笔记本电脑都满足。但是，最好使用独立显卡来运行 `Gazebo` 仿真环境。此外，如果能够有足够的外围接口将会更好，因为这样你可以连接几个传感器和执行机构，包括摄像头和 `Arduino`。

你还需要 `Git` (`git-core Debian` 软件包)，以便从本书提供的软件源下载源代码。同样，你需要具备 `Bash` 命令行、`GNU/Linux` 工具的基本知识和一些 `C/C++` 编程技巧。

目标读者

本书的目标读者包括所有机器人开发人员，可以是初学者也可以是专业人员。它涵盖了整个机器人系统的各个方面，展示了 ROS 系统如何帮助完成机器人真正自主化的任务。对于听说过 ROS 却从未使用过的机器人领域的学生或科研人员来说，本书将是非常有益的。ROS 初学者能从本书中学习很多 ROS 软件框架的先进理念和工具。不仅如此，经常使用 ROS 的用户也可能从某些章节中学习一些新东西。当然，只有前 3 章是纯粹为初学者准备的，所以那些已经使用过 ROS 的人可以跳过这部分直接阅读后面的章节。

排版说明

在本书中，你会发现一些文本的样式与其他正文内容不同。下面是一些样式的示例及含义。当我们希望能够着重显示示例代码中的某些内容的时候，相关的代码行会如下表示：

```
<launch>
<node pkg="chapter3_tutorials" type="example1"
      name="example1" output="screen"
      launch-prefix="xterm -e gdb --args"/>
</launch>
```

Bash 命令行下输入的命令或显示的结果如下表示：

```
$ rosrun book_tutorials tutorialX _param:=9.0
```



警告或注意事项会这样显示。



提示与小窍门会这样显示。

读者反馈

我们非常欢迎读者能够提出意见与建议。让我们知道你关于本书的一些想法，例如：你喜欢什么，不喜欢什么。读者的反馈对我们是非常重要的，这让我们知道我们最需要做什么。

向我们反馈信息只需发送电子邮件至 feedback@packtpub.com，并在邮件的主题中说明书籍名称。

如果你在某项上有所专长，或者你有兴趣写一本书或者为书籍的出版做出贡献，你可以浏览 www.packtpub.com/authors 阅读我们的作者指南。

读者支持

你现在自豪地拥有了 Packt 出版的书籍，我们会帮助你使你购买的产品物超所值。

源代码下载

可以访问 <http://www.packtpub.com/support>，完成网站注册并通过 E-mail 接收所有代码文件。

彩色图片下载

我们同时提供包含了本书所有彩色的屏幕截图、对话框的 PDF 文件，这些彩色图片能够更好地帮助你理解输出的变化。可以从这里下载这个文件：http://www.packtpub.com/sites/default/files/downloads/1448OS_Graphics.pdf。

勘误

虽然我们已尽力确保本书内容的准确性，但错误在所难免。如果你在我们的书中发现错误，即使是一个错字或一段错误的代码，我们将非常感谢你把错误告知我们。这样可以避免其他读者发生困扰并帮助我们提高本书后续版本的质量。如果你发现任何错误，请访问 <http://www.packtpub.com/submit-errata> 报告这些问题。访问网页，选择你的书，点击勘误提交表格的链接，并输入你的勘误细节。一旦确认了你的勘误，我们会告知你的提交被接受并在我们的网站上更新勘误表。你可以在 <http://www.packtpub.com/support> 根据你选择的标题查看现有的勘误表。

问题

如果关于本书你有任何疑问，请联系我们 questions@packtpub.com，我们会竭尽所能为你解答。

Contents | 目 录

译者序

前 言

第 1 章 ROS 系统入门 1

1.1 使用软件源安装 ROS Electric 3

1.1.1 添加软件源到 sources.list 文件中 4

1.1.2 设置密码 4

1.1.3 安装 4

1.1.4 环境配置 5

1.2 使用软件源安装 ROS Fuerte 6

1.2.1 配置 Ubuntu 软件源 6

1.2.2 配置 source.list 文件 6

1.2.3 设置密码 7

1.2.4 安装 7

1.2.5 环境配置 8

1.2.6 独立工具 9

1.3 如何安装 VirtualBox 和 Ubuntu 9

1.3.1 下载 VirtualBox 9

1.3.2 创建虚拟机 10

1.4 本章小结 12

第 2 章 ROS 系统架构及示例 13

2.1 理解 ROS 文件系统级 13

2.1.1 功能包 14

2.1.2 功能包集 16

2.1.3 消息类型 16

2.1.4 服务类型 17

2.2 理解 ROS 计算图级 18

2.2.1 节点 19

2.2.2 主题 20

2.2.3 服务 21

2.2.4 消息 22

2.2.5 消息记录包 22

2.2.6 节点管理器 22

2.2.7 参数服务器 22

2.3 理解 ROS 开源社区级 23

2.4 ROS 系统试用练习 23

2.4.1 ROS 文件系统导览 24

2.4.2 创建工作空间 24

2.4.3 创建 ROS 功能包 25

2.4.4 编译 ROS 功能包 26

2.4.5 使用 ROS 节点 26

2.4.6 使用主题与节点交互 28

2.4.7 学习如何使用服务 31

2.4.8 使用参数服务器 33

2.4.9 创建节点 34

2.4.10 编译节点 36

2.4.11 创建 msg 和 srv 文件	37	3.5.2 另一个画图工具 rxtools	60
2.4.12 使用新建的 srv 和 msg 文件	38	3.6 图像可视化	61
2.5 本章小结	42	3.6.1 显示单一图片	61
第 3 章 调试和可视化	43	3.6.2 FireWire 接口摄像头	62
3.1 调试 ROS 节点	44	3.6.3 使用双目立体视觉	63
3.1.1 使用 GDB 调试器调试 ROS 节点	45	3.7 3D 可视化	64
3.1.2 ROS 节点启动时调用 GDB 调试器	46	3.7.1 使用 rviz 在 3D 世界中 实现数据可视化	64
3.1.3 设置 ROS 节点 core 文件 转存	47	3.7.2 主题与坐标系的关系	66
3.2 调试信息	47	3.7.3 可视化坐标变换	67
3.2.1 输出调试信息	47	3.8 保存与回放数据	68
3.2.2 设置调试信息级别	48	3.8.1 什么是消息记录包文件	69
3.2.3 为特定节点配置调试 信息级别	48	3.8.2 使用 rosbag 在包文件中 记录数据	69
3.2.4 信息命名	50	3.8.3 回放消息记录文件	70
3.2.5 条件显示信息与过滤信息	50	3.8.4 使用 rxbag 检查消息记录 包的主题和消息	71
3.2.6 信息的更多功能——单次 显示、可调、组合	51	3.9 rqt 插件与 rx 应用	72
3.2.7 使用 rosconsole 和 rxconsole 在运行时修改调试级别	52	3.10 本章小结	73
3.3 监视系统状态	56	第 4 章 在 ROS 下使用传感器 和执行机构	74
3.3.1 节点、主题与服务列表	56	4.1 使用游戏杆或游戏手柄	74
3.3.2 使用 rxgraph 在线监视 节点状态图	56	4.1.1 joy_node 如何发送游戏杆 动作消息	75
3.4 当奇怪的事情发生——使用 roswtf	58	4.1.2 使用游戏杆数据在 turtlesim 中移动海龟	76
3.5 画标量数据图	58	4.2 使用激光雷达——Hokuyo URG-04lx	79
3.5.1 用 rxplot 画出时间趋势 曲线	59	4.2.1 了解激光雷达如何在 ROS 中发送数据	80
		4.2.2 访问和修改激光雷达 数据	82

4.3 使用 Kinect 传感器查看 3D 环境	84	5.2.3 加载图形到机器人模型	109
4.3.1 如何发送和查看 Kinect 数据	85	5.2.4 使机器人模型运动	109
4.3.2 创建和使用 Kinect 示例	86	5.2.5 物理和碰撞属性	110
4.4 使用伺服电动机——Dynamixel	88	5.3 xacro——一个写机器人模型的更好方法	111
4.4.1 Dynamixel 如何发送和接收运动命令	89	5.3.1 使用常量	111
4.4.2 创建和使用伺服电动机示例	90	5.3.2 使用数学方法	112
4.5 使用 Arduino 添加更多的传感器和执行机构	91	5.3.3 使用宏	112
4.6 使用惯性测量模组——Xsens MTi	94	5.3.4 使用代码移动机器人	112
4.6.1 Xsens 如何在 ROS 中发送数据	95	5.3.5 使用 SketchUp 进行 3D 建模	116
4.6.2 创建和使用 Xsens 示例	96	5.4 在 ROS 中仿真	117
4.7 使用低成本惯性测量模组 IMU-10 自由度	98	5.4.1 在 Gazebo 中使用 URDF3D 模型	117
4.7.1 下载加速度传感器库	99	5.4.2 在 Gazebo 中添加传感器	120
4.7.2 Arduino Nano 和 10 自由度传感器编程	99	5.4.3 在 Gazebo 中加载和使用地图	121
4.7.3 创建 ROS 节点并使用 10 自由度传感器数据	101	5.4.4 在 Gazebo 中移动机器人	123
4.8 本章小结	103	5.5 本章小结	125
第 5 章 3D 建模与仿真	104	第 6 章 机器视觉	126
5.1 自定义机器人在 ROS 中的 3D 模型	104	6.1 连接和运行摄像头	128
5.2 创建第一个 URDF 文件	104	6.1.1 FireWire IEEE1394 摄像头	128
5.2.1 解释文件格式	106	6.1.2 USB 摄像头	132
5.2.2 在 rviz 里查看 3D 模型	107	6.2 使用 OpenCV 制作 USB 摄像头驱动程序	133
		6.2.1 创建 USB 摄像头驱动功能包	134
		6.2.2 使用 ImageTransport API 发布摄像头帧	135
		6.2.3 使用 cv_bridge 进行 OpenCV 和 ROS 图像处理	138

6.2.4 使用 ImageTransport 发布 图像	139	7.6.1 使用 map_server 保存 地图	181
6.2.5 在 ROS 中使用 OpenCV	139	7.6.2 使用 map_server 加载 地图	182
6.2.6 显示摄像头输入的图像	140	7.7 本章小结	183
6.3 如何标定摄像头	140	第 8 章 导航功能包集进阶	184
6.4 ROS 图像管道	147	8.1 创建功能包	184
6.5 对于计算机视觉任务有用的 ROS 功能包	152	8.2 创建机器人配置	184
6.6 使用 viso2 执行视觉测距	153	8.3 配置全局和局部代价地图	187
6.6.1 摄像头位姿标定	154	8.3.1 基本参数的配置	187
6.6.2 运行 viso2 在线演示	156	8.3.2 全局代价地图的配置	188
6.6.3 使用低成本双目摄像头 运行 viso2	158	8.3.3 局部代价地图的配置	189
6.7 本章小结	159	8.4 基本局部规划器配置	189
第 7 章 导航功能包集入门	160	8.5 为导航功能包集创建启动文件	190
7.1 ROS 导航功能包集	160	8.6 为导航功能包集设置 rviz	191
7.2 创建转换	161	8.6.1 2D 位姿估计	191
7.2.1 创建广播机构	162	8.6.2 2D 导航目标	192
7.2.2 创建侦听器	162	8.6.3 静态地图	193
7.2.3 查看坐标变换树	164	8.6.4 点云	193
7.3 发布传感器信息	165	8.6.5 机器人立足点	193
7.4 发布里程数据	168	8.6.6 障碍	194
7.4.1 Gazebo 如何获取里程 数据	169	8.6.7 膨胀障碍	194
7.4.2 创建自定义里程数据	171	8.6.8 全局规划	195
7.5 创建基础控制器	175	8.6.9 局部规划	195
7.5.1 使用 Gazebo 创建里程 数据	176	8.6.10 规划器规划	196
7.5.2 创建基础控制器	178	8.6.11 当前目标	196
7.6 使用 ROS 创建地图	180	8.7 自适应蒙特卡罗定位	197
		8.8 避免障碍	199
		8.9 发送目标	200
		8.10 本章小结	202

第 9 章 在实践中学习	203	型人形机器人	217
9.1 REEM——类人形 PAL		9.3.1 从软件源安装	
机器人	204	Robonaut 2	217
9.1.1 从官方软件源安装		9.3.2 在国际空间站的固定支座	
REEM	205	上运行 Robonaut2	218
9.1.2 使用 Gazebo 仿真环境		9.4 Husky——Clearpath 的轮式	
运行 REEM	208	机器人	222
9.2 PR2——柳树车库机器人	210	9.4.1 安装 Husky 仿真环境	222
9.2.1 安装 PR2 仿真环境	210	9.4.2 运行 Husky 仿真环境	222
9.2.2 在仿真环境中运行 PR2	211	9.5 TurtleBot——低成本移动	
9.2.3 生成地图与定位	214	机器人	224
9.2.4 在仿真环境中运行 PR2		9.5.1 安装 TurtleBot 仿真环境	224
演示程序	216	9.5.2 运行 TurtleBot 仿真环境	224
9.3 Robonaut 2——NASA 的敏捷		9.6 本章小结	225

第 1 章

ROS 系统入门

欢迎来到本书的第 1 章。本章会介绍如何安装 ROS 系统。与其说 ROS 是一个操作系统，不如说它是一种新的标准化机器人软件框架。通过 ROS，你可以使用大量的示例代码和开源程序轻松地完成机器人编程和控制。同时，你还能够理解如何使用各种传感器与执行机构，并为你的机器人增加如自动导航和视觉定位等新的功能。在这里，我们要感谢相关的基金会及社区对开源软件的贡献，并不断地开发出最新的算法和功能使 ROS 系统不断进步。

本书包含以下知识：

- 在特定版本的 Ubuntu 系统下安装 ROS 框架
- 学习 ROS 的基本操作
- 调试数据和数据可视化
- 在 ROS 框架下对机器人编程
- 创造 3D 模型并进行仿真
- 使用导航功能包集使机器人进行自动导航

在本章中，首先要在 Ubuntu 系统中安装完整版本的 ROS。Ubuntu 不仅能够全面支持 ROS，而且它也是 ROS 官方推荐的操作系统。当然，你也可以在其他的操作系统中安装 ROS。但是在其他操作系统中，ROS 大多是实验性质的移植版本，因此容易出现各种安装和程序执行错误。因此在学习和使用本书的过程中，推荐使用 Ubuntu。

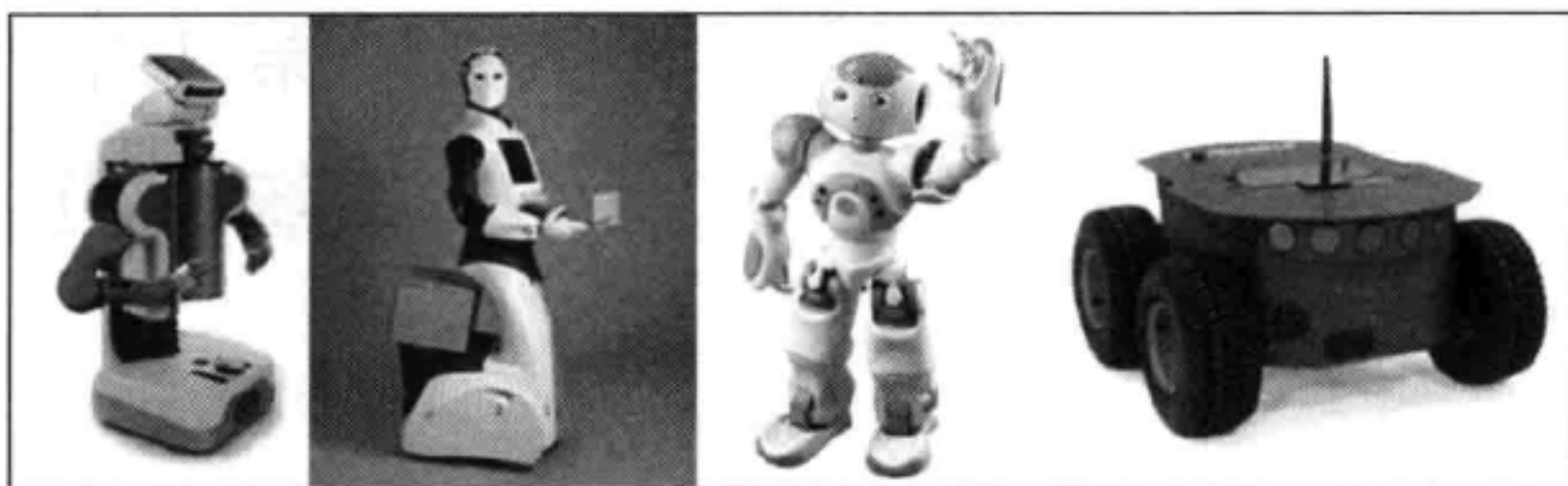
在开始安装之前，首先了解一下 ROS 的历史。

Robot Operating System (ROS) 是一种得到广泛使用的机器人操作与控制系统软件框架。该框架使用了当前最流行的面向服务 (SOA) 的软件技术，通过网络协议将节点间数据通信解耦。这样就能够轻松地集成不同语言不同功能的代码。ROS 的基本原理是无需改动就能够在不同的机器人上复用代码。基于这些，我们就可以在不同的机器人上分享和复用已经实现的功能，而不需要做太多的工作，避免了重复劳动。

2007 年，斯坦福大学人工智能实验室 (SAIL) 在斯坦福 AI 机器人项目 (Stanford AI Robot project) 的支持下开发了 ROS。2008 年之后，则主要在 Willow Garage 公司支持下与其他 20 多家研究机构进行联合研发 ROS。

现在已经有很多家研究机构通过增加 ROS 支持的硬件或开放软件源代码的方式加入 ROS 系统的开发中。同样，也有很多家公司将其产品逐步进行软件迁移并在 ROS 系统中应

用。部分 ROS 系统支持的平台如下图所示。这些平台往往会开放大量的代码、示例和仿真环境，以便开发人员轻松地开展工作。

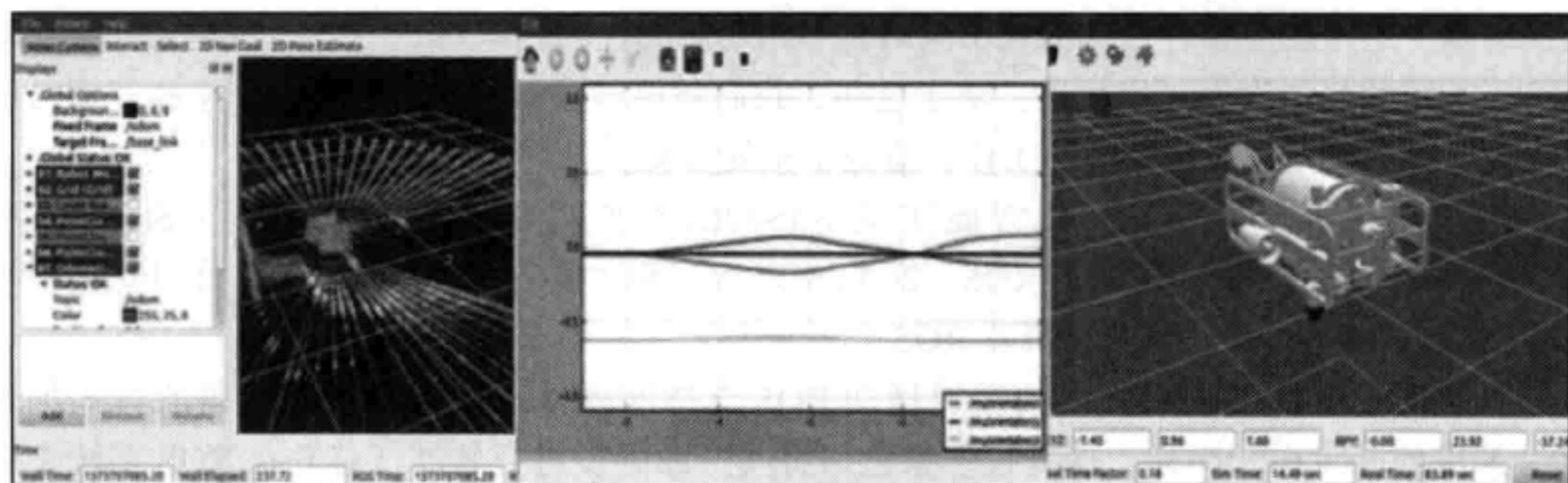


ROS 系统已经支持这些机器人中的传感器和执行机构，同时每天 ROS 软件框架所支持的设备也在大量增加。

ROS 提供了一个标准的操作系统环境，包括硬件抽象、底层设备控制、通用功能的实现、进程间消息转发和功能包管理等。它将多路复用传感器、控制器、运动状态、规划目标、执行机构及其他设备全部抽象成节点 (node)，并能对各个节点进程间消息的发生和接受的处理过程通过节点状态图展示。它的各种库都是面向类 Unix 系统的 (ROS 主要支持 Ubuntu Linux，而其他系统如 Fedora 和 Mac OS X 则是实验性质的)。



*-ros-pkg 包主要作为开发高级库的基础软件源。其很多功能是和 ROS 系统绑定的，如导航库和 rviz 可视化界面都基于这个源。其中的一些库包含很多强大的工具，可以帮助我们方便使用 ROS 并了解机器人的实时状态。其中，可视化工具、仿真环境和调试工具是最重要的几个。



ROS 是一个使用 BSD (Berkeley Software Distribution) 开源协议的开源软件。无论是商业应用, 还是科学研究, 它都是免费的。*-ros-pkg 包受到了多个开源协议的限制。

ROS 致力于机器人的代码复用, 这样每次研发新的机器人时, 开发人员和科学家们再也不必全部推倒重来。通过 ROS, 你可以集中精力做更多的事情, 应用不同代码库中的代码, 再完善, 并再次分享。

ROS 已经发布了多个版本, 最新的版本是 Groovy。在本书中, 我们使用的版本是 Fuerte, 因为这个版本更加稳定。因此需要说明的是, 本书中的示例和教程不一定都能够在 Groovy 下使用。

下面我们会介绍如何安装 Electric 版本和 Fuerte 版本的 ROS。不同的 ROS 版本对应不同的 Ubuntu 版本, 所以某些机器人可能仍然使用老版本的 Ubuntu。即使在本书中我们使用 Fuerte, 但是在实际工作中, 你仍然可能需要安装 Electric 以便运行一些老版本的代码。

如前文所述, 本书中所使用的操作系统是 Ubuntu, 如果你习惯使用其他操作系统又想完成本书的学习, 最好的选择就是安装一个带有 Ubuntu 的虚拟机。因此, 我会在最后介绍虚拟机的安装方法。

当然, 如果你想在其他系统中安装 ROS, 你可以根据链接 <http://wiki.ros.org/fuerte/Installation> 中的指导来完成。

1.1 使用软件源安装 ROS Electric

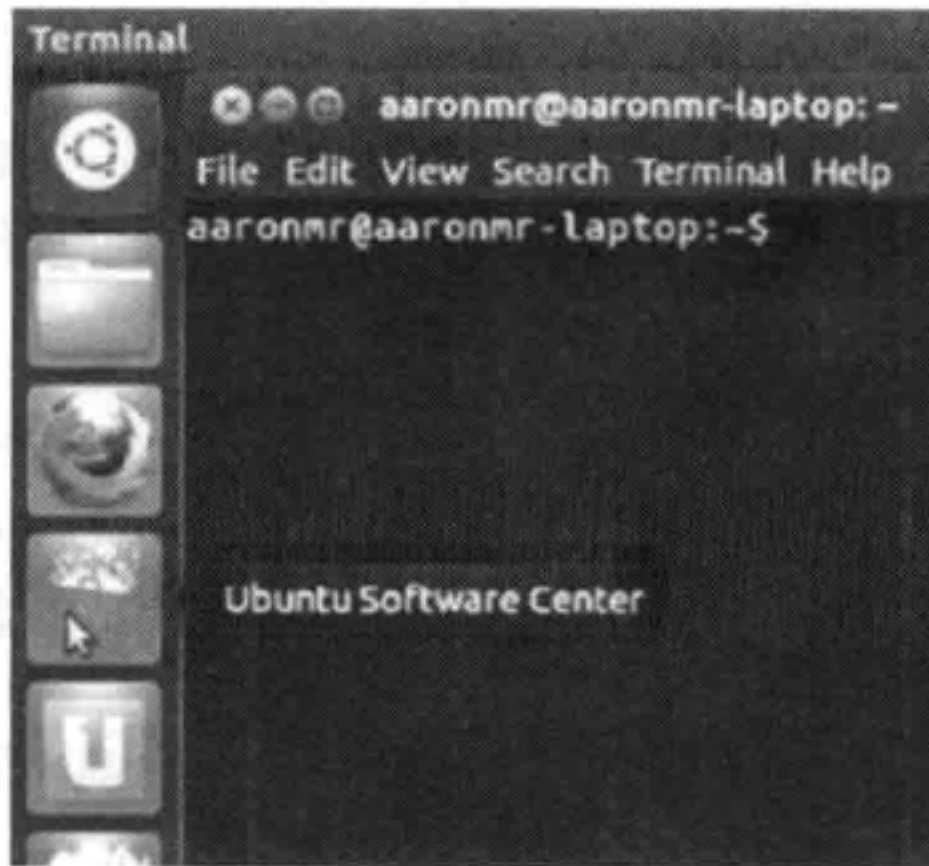
安装 ROS 有多种方法, 你可以按照本书的方法直接使用软件源, 也可以使用源文件并编译后安装。使用软件源的方法相对安全, 不容易出现错误。

按照本节介绍的步骤, 你可以在自己的电脑上逐步安装 ROS Electric, 安装过程详见官网页面 <http://wiki.ros.org/electric/Installation>。

我们假设你知道 Ubuntu 软件源 (repository) 的含义, 并且知道如何管理它。如果你有任何疑问请查询 <https://help.ubuntu.com/community/Repositories/Ubuntu> 了解相关知识。

在开始安装之前, 需要首先配置软件源, 为此需要先把软件源属性设为 restricted、universal、multiverse, 为了检查你的 Ubuntu 版本是否支持这些软件源, 请双击打开你桌面左面的 Ubuntu 软件中心 (Ubuntu Software Center), 如右图所示。

打开 Edit | Software Sources 标签页, 你能看到以下界面, 你要保证各个选项与下图中一致。





通常情况下，这些选项都是选中的，因此这一步骤没有什么问题。

1.1.1 添加软件源到 sources.list 文件中

在这一步中，你应该先选择 Ubuntu 的版本，在多种版本的操作系统中都可以安装 ROS Electric。你可以使用任何一个版本，但是推荐最新版本，以避免发生问题。

- 基于 Ubuntu 的发行版，如 Ubuntu Lucid Lynx (10.04)，安装软件源的具体方法如下：

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu lucid  
main" > /etc/apt/sources.list.d/ros-latest.list'
```

- 安装任意 Ubuntu 发行版的一种通用方法是使用所有基于 Debian 的 Linux 发行版都支持的 `lsb_release` 命令：

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu `lsb_  
release -cs` main" > /etc/apt/sources.list.d/ros-latest.list'
```

一旦添加了正确的软件源，操作系统就知道去哪里下载程序，并根据命令自动安装软件。

1.1.2 设置密码

这一步是为了确认原始的代码是正确的，并且没有人在未经所有者授权的情况下修改任何程序代码。通常情况下，当添加完软件源，你就已经添加了软件源的密码，并将其添加到操作系统的可信任列表中。

```
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

现在我们能够确定代码来自授权网站。

1.1.3 安装

现在准备开始安装。在开始之前，最好先升级一下软件，避免错误的库版本或软件版本产生各种问题，输入以下命令：


```
$ sudo apt-get update
```

ROS 非常大，有时候你会安装一些永远也用不到的库和程序。通常情况下，根据用途有四种不同的安装方式。例如，你是一个高级用户，你只需要为你的机器人进行基本安装，而不需要在硬盘上留过多的空间。在本书中，我们推荐完全安装，因为这样能够保证包含本书中所有示例和教程需要的库。不用担心，如果你不知道正在安装什么，它可能是 `rviz`、仿真环境或导航功能。你将会在后续章节中学习这些内容。

- 最简单的安装方式（并且是推荐的安装方式，但你需要有足够大的硬盘空间）就是桌面完整安装（`desktop-full`）。这将安装 ROS、Rx 工具箱、`rviz` 可视化环境（3D）、通用机器人库、2D（如 `stage`）和 3D（如 `Gazebo`）仿真环境、导航功能包集（移动、定位、地图绘制、机械臂控制）和其他感知库如视觉、激光雷达和 RGB-D 摄像头（深度摄像头）：

```
$ sudo apt-get install ros-electric-desktop-full
```

- 如果你没有足够的硬盘空间，或更喜欢安装特定部分功能包集，那么第一次安装可以仅安装桌面安装文件，包括 ROS、Rx 工具箱、`rviz` 和其他通用机器人库。之后在需要的时候，再安装其他功能包集（使用 `apt` 命令并查找 `ros-electric-*` 功能包集）：

```
$ sudo apt-get install ros-electric-desktop
```

- 如果你仅想尝试一下，请安装 `ROS-base`。`ROS-base` 通常是直接安装在机器人上，尤其是机器人并没有屏幕和人机界面，只能 TTY 远程登录的情况下。它只安装 ROS 的编译和通信库，而没有任何的 GUI 工具：

```
$ sudo apt-get install ros-electric-ros-base
```

- 最后，无论你选择哪一个选项进行安装，你都可以独立安装特定的 ROS 功能包集（将 `STACK` 替换成给定功能包集的名称）：

```
$ sudo apt-get install ros-electric-STACK
```

1.1.4 环境配置

恭喜你！能到这一步，说明你已经安装了某个版本的 ROS。为了使用 ROS，必须让操作系统知道可执行文件或二进制文件与系统命令的关联，这就需要执行下面的脚本。如果你已经安装了某一版本的 ROS，并想安装另一个 ROS 发行版，你可以每一次都调用你所需要的对应脚本。因为这个脚本能够非常方便地设置你的环境变量。在这里，我们将会使用 ROS Electric，但如果你想尝试其他的发行版，只需要把下面的命令中的 `electric` 替换成 `fuerte` 或 `groovy` 即可：

```
$ source /opt/ros/electric/setup.bash
```

如果你在 shell 中输入 `roscore`，那么将会看到有程序启动。这是测试是否完成 ROS 安

装和 ROS 安装是否正确的最佳方法。

请注意，如果你再打开一个 shell 窗口，并输入 `roscore` 或其他 ROS 命令，就行不通了。这是因为需要你再一次执行脚本来配置全局变量和 ROS 的安装路径。

如何能够让系统记住你的这些配置呢？你只需要在命令行脚本末尾处增加 `.bashrc` 文件，这样当你再次启动 shell 的时候，会自动执行这些命令并对环境进行配置。命令脚本如下：

```
$ echo "source /opt/ros/electric/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

如果安装了多个 ROS 的发行版，`~/.bashrc` 文件必须来自当前正在使用版本的 `setup.bash`。这是因为最后一次调用将覆盖环境中的其他设置。正如我们前面所提到的，在同一个系统中安装多个发行版时需要在它们之间进行切换。

1.2 使用软件源安装 ROS Fuerte

在本节中，我们将在计算机上安装 ROS Fuerte。你可以在同一台计算机上安装不同版本，这都没有问题。只需要在 `.bashrc` 文件中选择你想要使用的版本。你将在本节学习如何做到这一点。

如果你想在官方页面查看对这个过程的解释，可以访问 <http://wiki.ros.org/fuerte/Installation>。

你可以使用两种方法安装 ROS：使用软件源和使用源代码。普通用户可以使用软件源来完成 ROS 的安装。你也可以使用源代码安装 ROS，但这个方法仅建议高级用户使用，不建议普通用户这样做。

1.2.1 配置 Ubuntu 软件源

在开始安装之前，首先配置软件源，为此先把软件源属性设为 `restricted`、`universal`、`multiverse`。具体安装方法请参考 1.1 节的相关内容。

通常情况下，Ubuntu 默认支持这些选项，并不需要进行特殊设置。

1.2.2 配置 source.list 文件

现在我们需要添加下载软件的 URL。由于 ROS Fuerte 并不支持 Ubuntu 10.10 Maverick 和 11.04 Natty，因此我们需要在电脑上安装 Ubuntu 10.04、11.10 或 12.04。

在本书中，我们选择 Ubuntu 12.04。在这个版本下，所有的示例程序都被检查、编译和执行过。

打开 shell 命令行，输入以下命令：

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu `lsb_release
-cs` main" > /etc/apt/sources.list.d/ros-latest.list'
```

这些命令在任何版本的 Ubuntu 下都应该能够正确运行。

1.2.3 设置密码

为软件源增加密码是非常重要的，因为这样能够保证源代码是正确的，并且没有人在未经所有者知晓的情况下修改代码或程序。

如果你已经按照步骤安装完成 ROS Electric，那么你并不需要重新设置密码；如果你没有安装过 ROS Electric，那么你需要输入如下命令：

```
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

1.2.4 安装

我们现在准备安装 ROS Fuerte。在安装之前，需要先升级一下 ROS 可能使用到的程序。这样可以避免出现兼容性问题。

输入以下命令并等待：

```
$ sudo apt-get update
```

这个命令会花一定时间才能完成，具体取决于你是否升级过系统。

ROS 同样有很多个组件，在一个机器人上完整安装全部系统会略显笨重而没有一点个性。基于这个原因，你可以根据需要选择安装。

在本书中，我们还是会安装完整版本。这个版本会安装所有的示例、功能包集和程序。考虑到我们在本书的后续章节会使用大量的工具，完整安装是一个不错的选择。当然，如果你并不想现在就完整安装，可以之后再继续进行。

- 最简单的安装方式（并且是推荐的安装方式，但你需要有足够大的硬盘空间）就是桌面完整安装（desktop-full）。这将安装 ROS、Rx 工具箱、rviz 可视化环境（3D）、通用机器人库、2D（如 stage）和 3D（如 Gazebo）仿真环境、导航功能包集（移动、定位、地图绘制、路径规划、机械臂控制）和其他感知库如视觉、激光雷达和 RGB-D 摄像头（深度摄像头）：

```
$ sudo apt-get install ros-fuerte-desktop-full
```

- 如果你没有足够的硬盘空间，或更喜欢安装部分功能包集，那么第一次仅安装桌面安装文件，包括 ROS、Rx 工具箱、rviz 和其他通用机器人库。你可以在需要的时候，再安装其他的功能包集（使用 apt 命令并查找 ros-fuerte-* 功能包集）：

```
$ sudo apt-get install ros-fuerte-desktop
```

- 如果你仅想尝试一下，请安装 ROS-base。它通常是直接安装在机器人上，尤其是机器人并没有屏幕和人机界面，只能 TTY 远程登录的情况下。只安装 ROS 的编译和通信库，而没有任何的 GUI 工具：

```
$ sudo apt-get install ros-fuerte-ros-comm
```

- 最后，无论选择哪一个选项进行安装，你都可以独立安装特定的 ROS 功能包集（将

STACK 替换成给定功能包集的名称):

```
$ sudo apt-get install ros-fuerte-STACK
```

在安装过程中, 请不要担心你对所安装的软件并不了解。在后面的章节, 你会学到包括如何使用在内的所有相关知识。

当你逐渐积累 ROS 系统的经验之后, 就能够在机器人上仅安装 ROS 内核, 使用更少的资源, 并且逐项添加你所需要的工具。

1.2.5 环境配置

现在 ROS 安装完毕, 并准备开始使用。这时你需要向 Ubuntu 提供 ROS 的安装路径。打开一个新的 shell 并输入以下命令:

```
$ roscore
roscore: command not found
```

之所以会看到这样的消息, 主要是因为 Ubuntu 并不知道去哪里找这个命令对应的执行程序。为了解决这个问题, 需要在 shell 环境中输入以下命令:

```
$ source /opt/ros/fuerte/setup.bash
```

然后, 再次输入 roscore 命令, 你会看到以下输出:

```
...
started roslaunch server http://localhost:45631/
ros_comm version 1.8.11
```

SUMMARY

=====

PARAMETERS

```
* /rostdistro
* /rosversion
```

NODES

```
auto-starting new master
```

```
....
```

这意味着 Ubuntu 已经知道了到哪里去寻找这条命令所对应的执行程序, 并启动了 ROS。而如果你重新打开一个 shell 命令行, 并再次输入 roscore, 就又不管用了。这是因为你必须将这个命令行脚本加入 .bashrc 文件中。这样在 shell 打开时 .bashrc 会自动运行, 所以每次你打开一个新的 shell, 这个配置环境变量的脚本都会运行。

在命令行中使用以下脚本命令:


```
$ echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

和前面提到的一样，`setup.bash` 只能为一个 ROS 发行版进行初始化配置。试想一下，当你同时安装了 ROS Electric 和 Fuerte，而平时需要使用 Fuerte 作为常用的版本。如果你想在 shell 环境中更改版本，只需要输入以下命令：

```
$ source /opt/ros/electric/setup.bash
```

如果你希望使用其他版本作为常用的版本，只需要修改 `.bashrc` 文件并将所需版本的相关脚本加入文件中。

1.2.6 独立工具

ROS 中的很多工具是需要在 ROS 安装之后进行独立安装的。这些工具能够帮助我们对编程和编译的安装依赖项进行管理、下载和安装 ROS 程序等。这些工具包括 `roscpp` 和 `roscpp`。我们推荐将这两个工具都安装，因为在后续的章节中需要使用它们。我们需要在 shell 命令行中输入以下命令：

```
$ sudo apt-get install python-roscpp python-roscpp
```

现在我们已经安装在系统中安装了一个完整版本的 ROS。如你所见，只有部分步骤是必须做的。

你可以在机器上安装两个甚至更多的 ROS 发行版。另外，如果你的电脑并不使用 Ubuntu 系统，你可以在虚拟机中安装 ROS。下一节我们将介绍如何安装虚拟机和使用镜像而且这可能是对于初学者来说最好的方法。

1.3 如何安装 VirtualBox 和 Ubuntu

VirtualBox 是一个通用的完整的，适用于 x86 硬件，定位于服务器、台式机和嵌入式应用的虚拟机。VirtualBox 是免费的，支持所有主流的操作系统。几乎每一个 Linux 爱好者都使用它。

由于我们推荐使用 Ubuntu，你可能不希望更改计算机现有的操作系统。而如 VirtualBox 之类的工具就可以满足此类需求。它能帮助我们在电脑上建立虚拟的新操作系统，而无需对电脑硬件做任何改动。

在后面的章节中，我们将展示如何安装 VirtualBox 和 Ubuntu。此外，通过安装虚拟机，你可以在一个干净的操作系统中完成开发。如果你遇到任何问题，能够快速重启计算机解决；也可以备份虚拟机及所有必要的机器人安装文件。

1.3.1 下载 VirtualBox

第一步是下载 VirtualBox 的安装文件。在编写本书时，以下链接能够下载最新的可用版本：

- <https://www.virtualbox.org/wiki/Downloads>
- <http://download.virtualbox.org/virtualbox/4.2.0/VirtualBox-4.2.1-80871-OSX.dmg>

一旦安装完成，你就需要下载 Ubuntu 的镜像文件。在本教程中，我们使用了一个已经安装了 ROS Ruerte 的 Ubuntu 镜像文件。你可以通过以下链接下载：

<http://nootrix.com/wp-content/uploads/2012/08/ROS.ova>。

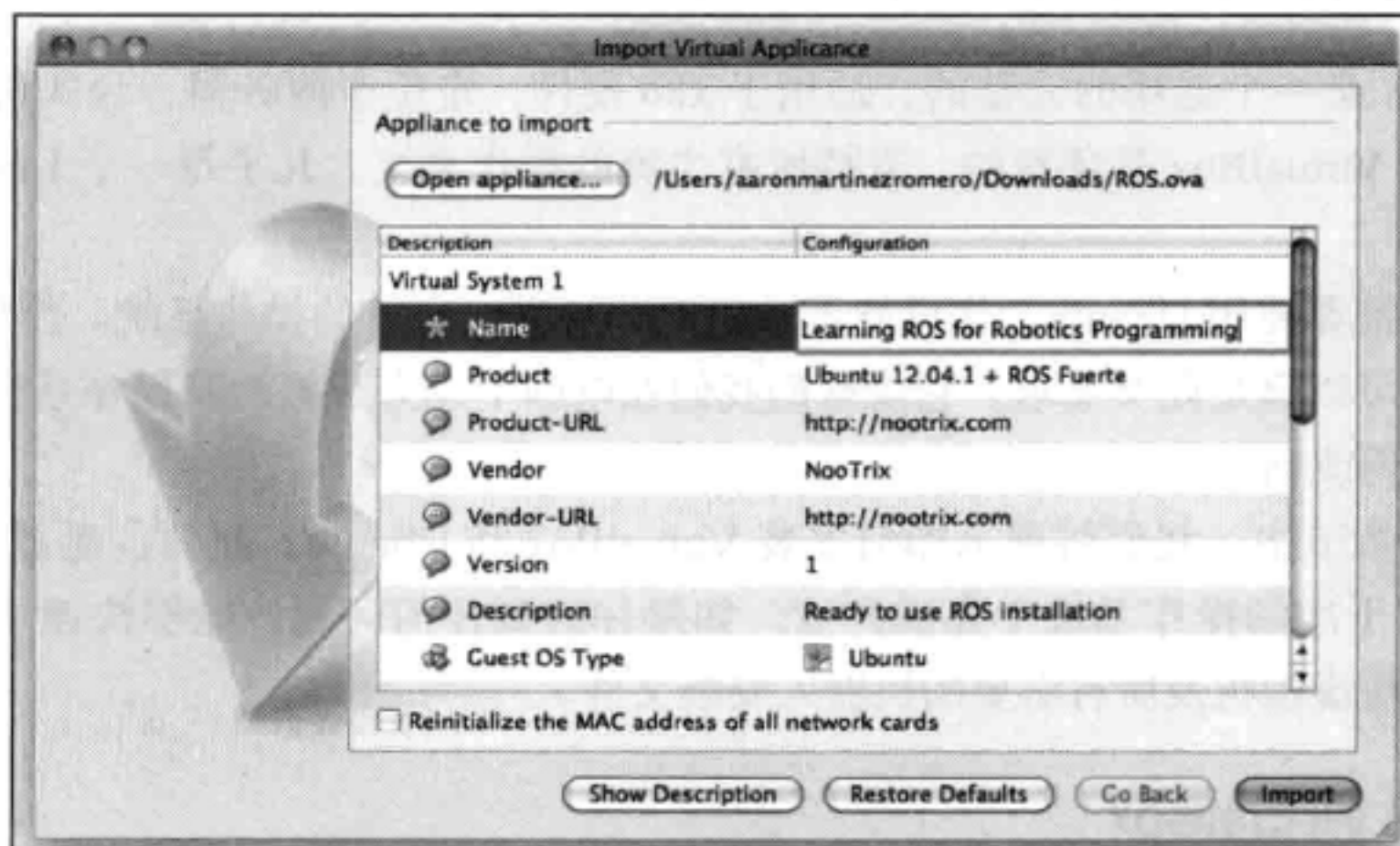
你也可以找到带有 Ubuntu 和 ROS 的其他虚拟机，但是我们还是将使用这个 ROS 官网推荐的版本。

1.3.2 创建虚拟机

通过下载好的文件创建一个虚拟机非常简单，只需要按照本节的内容一步一步进行即可。打开 VirtualBox 虚拟机软件并打开 File | Import Appliance...，然后点击 Open appliance 并选择之前下载好的 ROS.ova 文件。



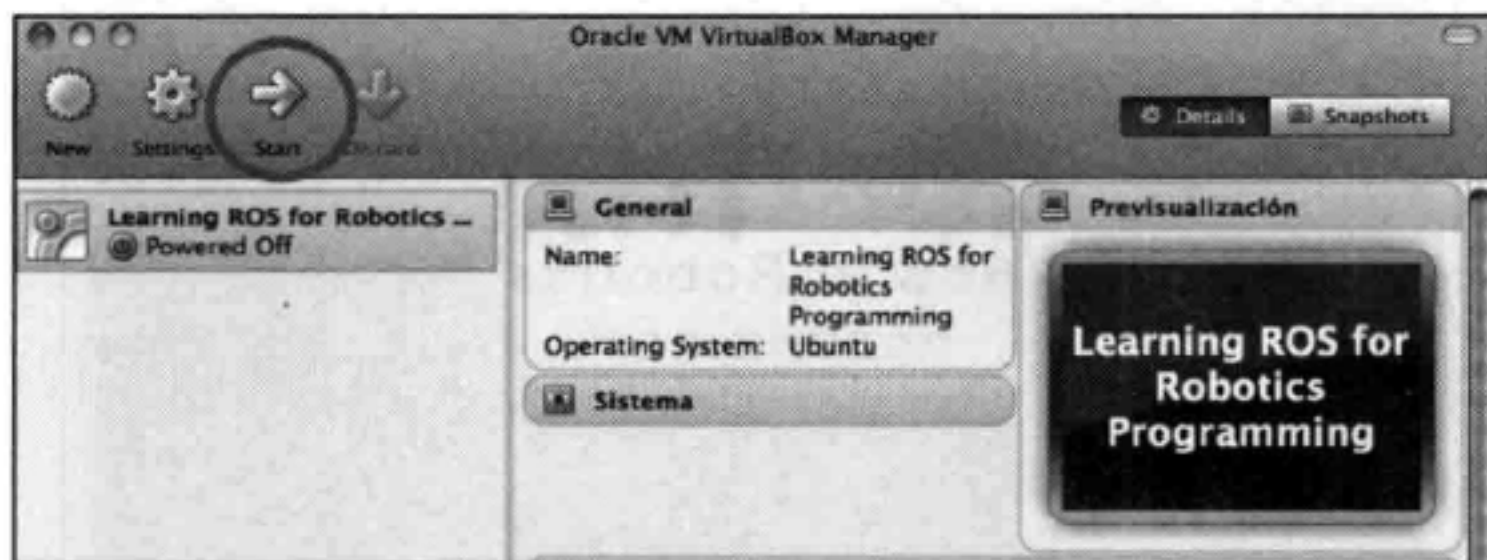
在下一个窗口中，可以配置新虚拟机的参数。我们保持默认配置并且仅仅改变虚拟机的名称。这个名称帮助我们区分不同的虚拟机。我们推荐给它起一个容易理解的名称，在这里我们使用本书的名称。



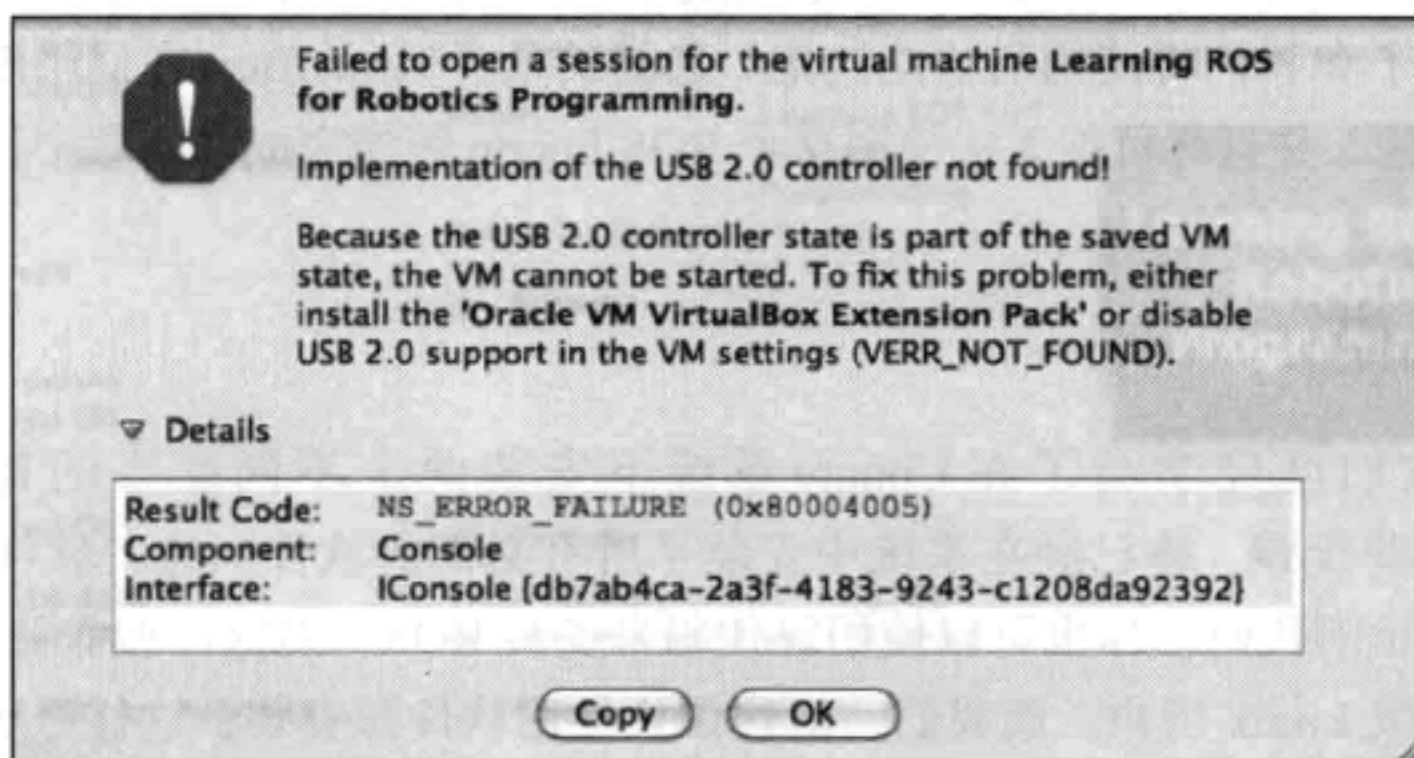
点击 Import 按钮，并在下一个窗口中接受软件授权许可。然后，你将看到一个进度条。这表明 VirtualBox 正在复制虚拟机镜像文件。

需要说明的是，这个过程并不会影响原有的 ROS.ova 文件，并且你可以通过对原文件进行多次复制创建多个虚拟机。

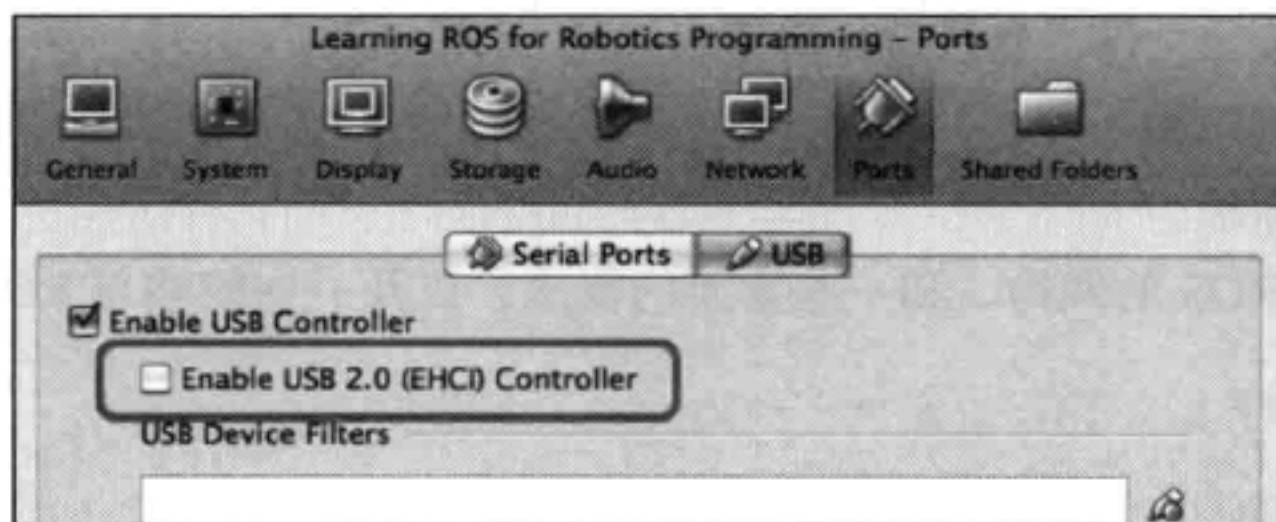
复制过程所需要的时间取决于你计算机的执行速度。当它完成时，你可以点击 Start 按钮启动你的虚拟机。需要注意的是，如果你的机器上有多个虚拟机，在启动前应该选择正确的那一个。当然，在我们这个例子里只有一个虚拟机。



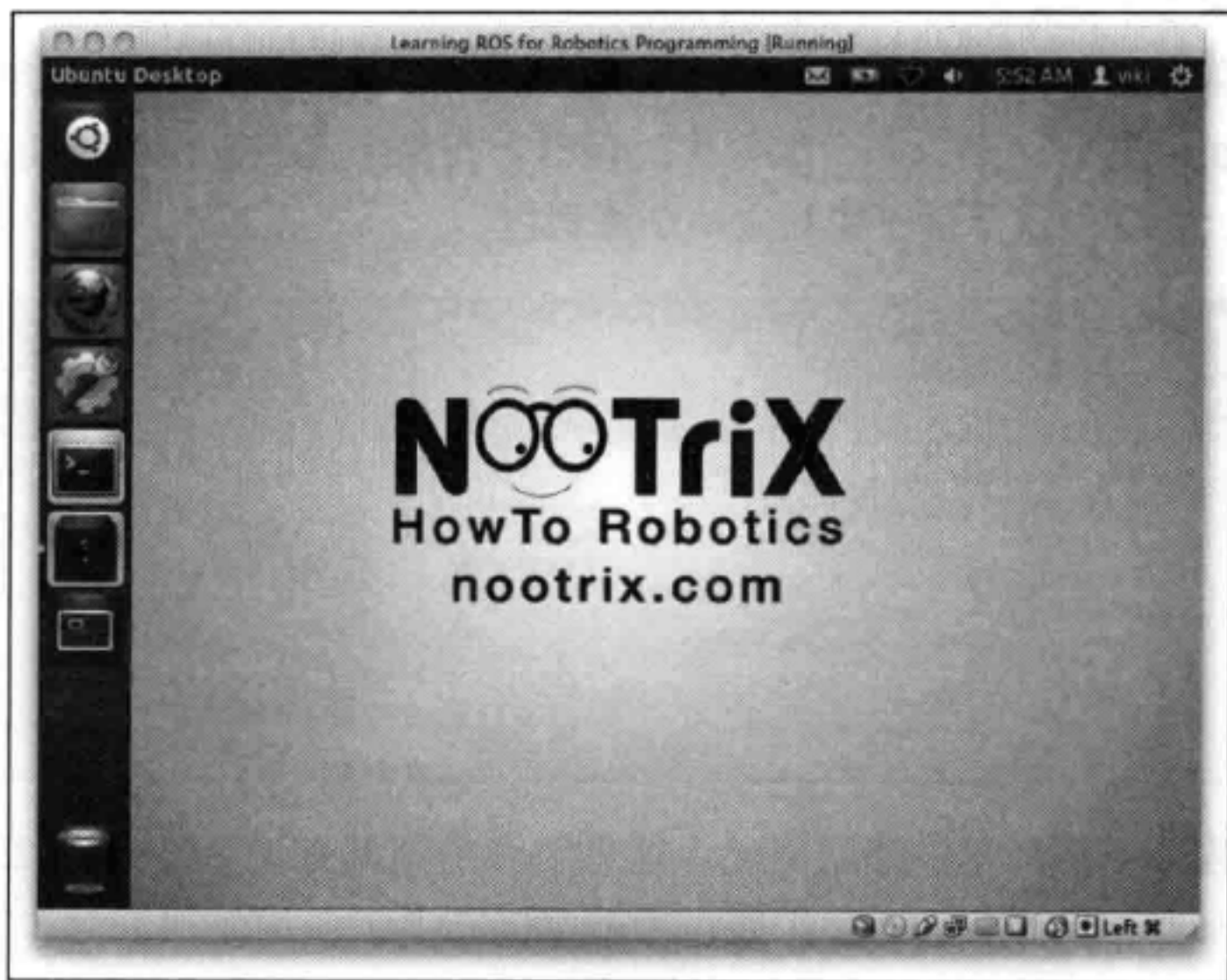
有时候会出现如下图所示的错误提示。这是因为电脑没有正确的 USB 2.0 驱动程序。你可以通过安装 Oracle VM VirtualBox 扩展包（Extension Pack）来解决问题，当然你也可以通过在虚拟机中禁用 USB 来解决。



为了禁用 USB，在虚拟机上右键单击并选择 Settings。在工具栏中，选择 Ports | USB，并取消勾选 Enable USB 2.0 (EHCI) Controller。重启虚拟机，就不会再出现任何问题。



虚拟机启动之后，你能看到完成 ROS 安装之后的 Ubuntu 12.04 界面，如下图显示：



当完成这些步骤后，你就有了一个能够在这本书中使用的完整版本的 ROS Fuerte。你可以运行所有的例子和我们将使用的功能包集。遗憾的是，VirtualBox 在使用部分实际外接设备的时候会有问题，并且你可能无法使用这个 ROS Fuerte 镜像完成第 4 章中列出的步骤。

1.4 本章小结

本章中，我们已经学会了在 Ubuntu 系统下安装两个不同版本的 ROS (Electric 和 Fuerte)。通过这些步骤，你已经在系统中安装了所有必需的软件，这些软件能够支持 ROS 和这本书的所有示例代码。你也可以使用源代码来安装 ROS。但这样做需要编译所有代码，因此只适用于高级 Linux 用户。而我们一般建议你使用软件源安装，这样做更加通用，且通常不会发生任何错误或问题。

如果你对 Ubuntu 系统不是很熟悉，那么建立一个虚拟机并在虚拟机上学习使用 ROS，会更加方便。这样，如果你在安装和使用过程中发生任何问题，都无需重新安装操作系统，只需要恢复虚拟机镜像文件，然后就可以重新开始。

通常情况下，虚拟机不能访问扩展出的实际硬件，如传感器和执行机构。尽管如此，你仍可以用它来测试算法。

下一章将学习 ROS 的架构、和一些重要的概念，以及一些能够与 ROS 进行直接交互的工具。

第 2 章

ROS 系统架构及示例

一旦你完成了 ROS 系统的安装，你肯定会想“好了，我已经安装完成，那么下一步是什么呢？”本章我们会学习 ROS 系统架构及它都包含哪些部分。然后，我们会开始创建节点和功能包，并使用 ROS 系统自带的 TurtleSim 示例。

ROS 系统的架构主要被设计和划分成了三部分，每一部分都代表一个层级的概念：

- 文件系统级 (Filesystem level)
- 计算图级 (Computation Graph level)
- 开源社区级 (Community level)

第一级是文件系统级。在这一级，我们将会使用一组概念来解释 ROS 的内部构成、文件夹结构，以及工作所需的核心文件。

第二级是计算图级，体现的是进程和系统之间的通信。在相关章节中，我们将看到 ROS 的各个概念和功能，包括建立系统、处理各类进程、与多台计算机通信等。

第三级是开源社区级，我们将解释一系列的工具和概念，其中包括在开发人员之间如何共享知识、算法和代码。这个层级是非常重要的，因为在开源社区的大力支持下 ROS 系统才得以快速成长。

2.1 理解 ROS 文件系统级

如果你刚接触 ROS，无论是准备使用 ROS 系统还是准备开发 ROS 项目，你都会觉得 ROS 中的各种概念非常奇怪。而一旦你已经驾轻就熟，那么这些概念就会变得亲切了。



与其他操作系统相类似，一个 ROS 程序的不同组件要被放在不同的文件夹下。这些文件夹是根据功能的不同来对文件进行组织的：

- **功能包 (Package)** 功能包是 ROS 中软件组织的基本形式。一个功能包具有最小的结构和最少的内容，用于创建 ROS 程序。它可以包含 ROS 运行的进程 (节点)、配置文件等。
- **功能包清单 (Manifest)** 功能包清单提供关于功能包、许可信息、依赖关系、编译标志等的信息。功能包清单是一个 `manifests.xml` 文件，通过这个文件能够实现对功能包的管理。
- **功能包集 (Stack)** 如果你将几个具有某些功能的功能包组织在一起，那么你将会获得一个功能包集。在 ROS 系统中，存在大量的不同用途的功能包集，例如导航功能包集。
- **功能包集清单 (Stack manifest)** 功能包集清单 (`stack.xml`) 提供一个关于功能包集的清单，包括开源代码的许可证信息、与其他功能包集的依赖关系等。
- **消息类型 (Message /msg type)** 消息是一个进程发送到其他进程的信息。ROS 系统有很多的标准类型消息。消息类型的说明存储在 `my_package/msg/MyMessageType.msg` 中，也就是对应功能包的 `msg` 文件夹下。
- **服务类型 (Service/srv type)** 对服务的类型进行描述说明的文件在 ROS 系统中定义了服务的请求和响应的数据结构。这些描述说明存储在 `my_package/srv/MyServiceType.srv` 中，也就是对应功能包的 `srv` 文件夹下。

在右面的截屏中，你能看到本书第 3 章文件夹下的内容。该文件夹就是一个功能包，其中包含了本章示例代码、功能包清单等多项内容。第 3 章的文件中没有消息和服务。但如果有，你还可以看到 `srv` 和 `msg` 文件夹。

```
chapter3_tutorials/  
├── CMakeLists.txt  
├── config  
│   └── chapter3_tutorials.config  
├── launch  
│   ├── example1_gdb.launch  
│   ├── example1.launch  
│   ├── example1_valgrind.launch  
│   ├── example2.launch  
│   └── example3.launch  
├── mainpage.dox  
├── Makefile  
├── manifest.xml  
├── output  
│   └── gdb_run_node_example1.txt  
└── src  
    ├── example1.cpp  
    ├── example2.cpp  
    └── example3.cpp  
  
4 directories, 14 files
```

2.1.1 功能包

当每次提到功能包时，指的是一种特定的文件结构和文件夹组合。这种结构如下所示：

- `bin/` 这是我们编译和链接程序后，用于存储可执行文件的文件夹。
- `include/package_name/` 此目录包含了你所需要库的头文件。不要忘记导出功能包清单，因为它们还会被其他功能包所使用。
- `msg/` 如果你要开发非标准消息，请把文件放在这里。
- `scripts/` 其中包括 Bash、Python 或任何其他脚本的可执行脚本文件。
- `src/` 这是存储程序源文件的地方。你可能会为节点创建一个文件夹或按照你希望的方式去组织它。

- `srv/` 这表示的是服务 (srv) 类型。
- `CMakeLists.txt` 这是 CMake 的生成文件。
- `manifest.xml` 这是功能包清单文件。

为了创建、修改或使用功能包，ROS 给我们提供了一些工具：

- `rospack` 使用此命令来获取信息或在系统中查找功能包。
- `roscmake` 当你想要创建一个新的功能包时，使用此命令。
- `rosmake` 使用此命令来编译功能包。
- `roscdep` 此命令安装功能包的系统依赖项。
- `roscdeps` 如果你想要查看功能包的依赖关系图，使用此命令。

若要在文件夹和功能包之间移动文件，ROS 提供了非常有用的 `roscbash` 功能包，其中包含了一些非常类似于 Linux 命令的命令。例如：

- `roscd` 此命令用于更改目录，类似于 Linux 中的 `cd` 命令。
- `roscd` 此命令用来编辑文件。
- `roscp` 此命令用于从一些功能包复制文件。
- `roscd` 此命令列出功能包的目录。
- `roscd` 此命令列出功能包下的文件，类似于 Linux 中的 `ls` 命令。

文件 `manifest.xml` 必须在功能包中，用来说明此功能包相关的各类信息。如果你发现在某个文件夹内包含有此文件，那么这个文件夹很可能是一个功能包。

打开一个 `manifest.xml` 文件，可以看到包的名称、依赖关系等信息。功能包清单的作用就是为了更容易地安装和分发这些功能包。

在功能包清单文件中使用的两个典型标记是 `<depend>` 和 `<export>`。`<depend>` 标记会显示当前功能包安装之前必须先安装哪些功能包。这是因为新的功能包会使用其他包的一些功能。`<export>` 标记告诉系统编译该功能包需要使用什么编译标志，引用哪些头文件等。

下面的代码段是此文件的示例：

```
<package>
  <description brief="short description">
    long description,
  </description>
  <author>Aaron Martinez, Enrique Fernandez</author>
  <license>BSD</license>
  <url>http://example.com/</url>
  <depend package="roscpp"/>
  <depend package="common"/>
  <depend package="otherPackage"/>
  <versioncontrol type="svn" url="https://urlpackage/trunk"/>
  <export>
    <cpp cflags="-I${prefix}/include" lflags="-L${prefix}/lib -lros"/>
  </export>
</package>
```


2.1.2 功能包集

ROS 系统中的很多功能包被组织成 ROS 功能包集。如果说功能包的目标是创建易于代码复用的最小代码集合，那么功能包集的目标则是简化代码共享的过程。

功能包集同样是一个由文件和文件夹构成的基本结构。你可以手动创建它，但 ROS 为我们提供了 `roscat-stack` 命令行工具来简化这一过程。

功能包集必需的三个文件是：`CMakeList.txt`、`Makefile` 和 `stack.xml`。如果你在文件夹中见到 `stack.xml`，那么可以确定这是一个功能包集。

下面的代码段是此文件的示例：

```
<stack>
  <description brief="Sample_Stack">Sample_Stack1</description>
  <author>Maintained by AaronMR</author>
  <license>BSD,LGPL,proprietary</license>
  <review status="unreviewed" notes="" />
  <url>http://someurl.blablabla</url>
  <depend stack="common_msgs" /> <!-- nav_msgs, sensor_msgs, geometry_msgs -->
  <depend stack="ros_tutorials" /> <!-- turtlesim -->
</stack>
```

2.1.3 消息类型

ROS 使用了一种简化的消息类型描述语言来描述 ROS 节点发布的数据值。通过这样的描述语言，ROS 能够使用多种编程语言生成不同类型消息的源代码。

ROS 提供了很多预定义消息类型。如果你创建了一种新的消息类型，那么就要把消息的类型定义放到功能包的 `msg/` 文件夹下。在该文件夹中，有用于定义各种消息的文件。这些文件都以 `.msg` 为扩展名。

消息类型必须具有两个主要部分：字段和常量。字段定义了要在消息中传输的数据的类型，例如 `int32`、`float32`、`string` 或之前创建的自定义类型，如叫做 `type1` 和 `type2` 的新类型。常量用于定义字段的名称。

一个消息文件的示例如下：

```
int32 id
float32 vel
string name
```

我们能够在下表中找到很多 ROS 消息所使用的标准数据类型：

基本类型	串行化	C++	Python
<code>bool</code>	Unsigned 8-bit int	<code>uint8_t</code>	<code>bool</code>
<code>int8</code>	Signed 8-bit int	<code>int8_t</code>	<code>int</code>
<code>uint8</code>	Unsigned 8-bit int	<code>uint8_t</code>	<code>int</code>
<code>int16</code>	Signed 16-bit int	<code>int16_t</code>	<code>int</code>

(续)

基本类型	串行化	C++	Python
uint16	Unsigned 16-bit int	uint16_t	int
int32	Signed 32-bit int	int32_t	int
uint32	Unsigned 32-bit int	uint32_t	int
int64	Signed 64-bit int	int64_t	long
uint64	Unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ASCII string (4-bit)	std::string	string
time	Secs/nsecs signed 32-bit ints	ros::Time	rospy. Time
duration	Secs/nsecs signed 32-bit ints	ros::Duration	rospy. Duration

ROS 消息中的一种特殊数据类型是报文头，主要用于添加时间戳、坐标位置等。报文头还允许对消息进行编号。通过在报文头内部附加信息，我们可以知道是哪个节点发出的消息，或者可以添加一些能够被 ROS 处理的其他功能。

报文头类型包含以下字段：

```
uint32 seq
time stamp
string frame_id
```

我们将在后续的章节中看到，正是通过报文头才能够记录当前机器人运行的时间戳和坐标位置。

在 ROS 中有一些处理消息的工具。例如：rosmmsg 命令行工具能够输出的消息具体信息，并可以找到使用该消息类型的源文件。

在后面的章节中，我们将会学习如何使用正确的工具创建消息。

2.1.4 服务类型

ROS 使用了一种简化的服务描述语言来描述 ROS 的服务类型。这直接借鉴了 ROS 消息的数据格式，以实现节点之间的请求 / 响应通信。服务的描述存储在功能包的 srv/ 子目录下 .srv 文件中。

若要调用服务，你需要使用该功能包的名称及服务名称。例如，对于 sample_package1/srv/sample1.srv 文件，可以将它称为 sample_package1/sample1 服务。

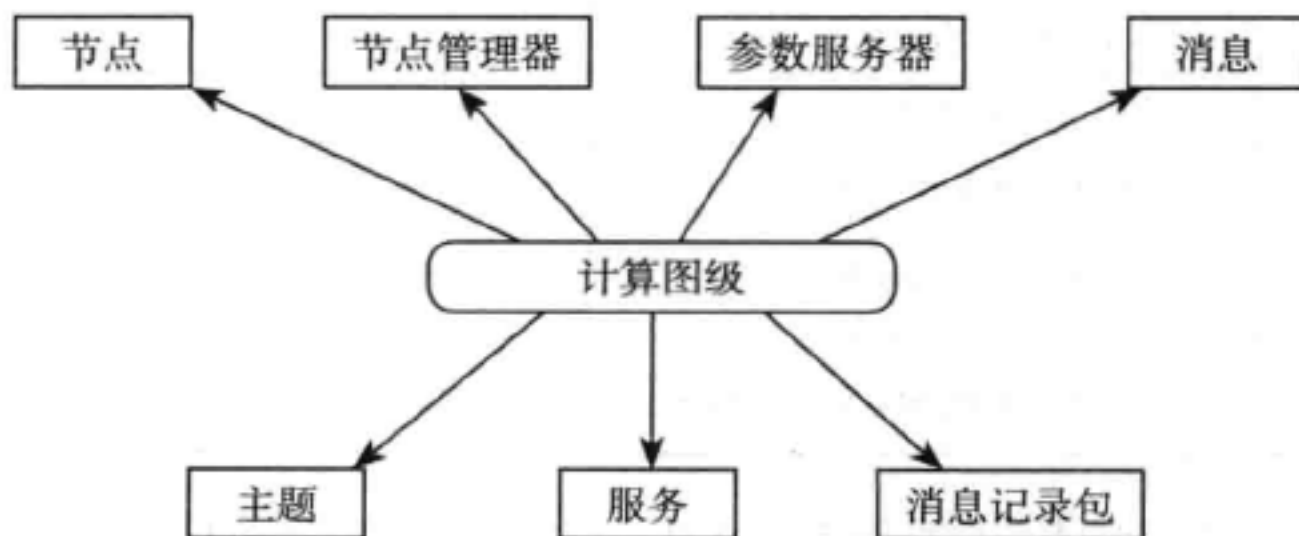
ROS 中有一些执行某些功能与服务的工具。rossrv 工具能输出服务说明、.srv 文件所在的功能包名称，并可以找到使用某一服务类型的源代码文件。

如果你想要在 ROS 中创建一个服务，可以使用服务生成器。这些工具能够从基本的服务说明中生成代码。你只需要在 CMakeLists.txt 文件中加一行 gensrv() 命令。

在后面的章节中，我们将会学习如何创建服务。

2.2 理解 ROS 计算图级

ROS 会创建一个连接到所有进程的网络。在系统中的任何节点都可以访问此网络，并通过该网络与其他节点交互，获取其他节点发布的信息，并将自身数据发布到网络上。



在这一层级中最基本的概念包括节点、节点管理器、参数服务器、消息、服务、主题和消息记录包，这些概念都以不同的方式向计算图级提供数据：

- **节点 (Node)** 节点是主要的计算执行进程。如果你想要有一个可以与其他节点进行交互的进程，那么你需要创建一个节点，并将此节点连接到 ROS 网络。通常情况下，系统包含能够实现不同功能的多个节点。你最好让每一个节点都具有特定的单一的功能，而不是在系统中创建一个包罗万象的大节点。节点需要使用如 `roscpp` 或 `rospy` 的 ROS 客户端库进行编写。
- **节点管理器 (Master)** 节点管理器用于节点的名称注册和查找等。如果在你的整个 ROS 系统中没有节点管理器，就不会有节点、服务、消息之间的通信。需要注意的是，由于 ROS 本身就是一个分布式网络系统，你可以在某一台计算机上运行节点管理器，在其他计算机上运行由该管理器管理的节点。
- **参数服务器 (Parameter Server)** 参数服务器能够使数据通过关键词存储在一个系统的核心位置。通过使用参数，就能够在运行时配置节点或改变节点的工作任务。
- **消息 (Message)** 节点通过消息完成彼此的沟通。消息包含一个节点发送到其他节点的数据信息。ROS 中包含很多种标准类型的消息，同时你也可以基于标准消息开发自定义类型的消息。
- **主题 (Topic)** 主题是由 ROS 网络对消息进行路由和消息管理的数据总线。每一条消息都要发布到相应的主题。当一个节点发送数据时，我们就说该节点正在向主题发布消息。节点可以通过订阅某个主题，接收来自其他节点的消息。一个节点可以订阅一个主题，而并不需要该节点同时发布该主题。这就保证了消息的发布者和订阅者之间相互解耦，完全无需知晓对方的存在。主题的名称必须是独一无二的，否则在同名主题之间的消息路由就会发生错误。

歧义。节点可以使用不同的库进行编写，如 `roscpp` 和 `rospy`。`roscpp` 基于 C++，而 `rospy` 基于 Python。在这本书里，我们将使用 `roscpp`。

ROS 提供了处理节点的工具，如 `roscpp`。`roscpp` 是一个用于显示节点信息的命令行工具，例如，列出当前正在运行的节点等。支持的命令如下所示：

- `roscpp info node` 输出当前节点信息。
- `roscpp kill node` 结束当前运行节点进程或发送给定信号。
- `roscpp list` 列出当前活动节点。
- `roscpp machine hostname` 列出某一特定计算机上运行的节点或列出主机名称。
- `roscpp ping node` 测试节点间的连通性。
- `roscpp cleanup` 将无法访问节点的注册信息清除。

在后面的章节中，我们将通过一些示例学习如何使用这些命令。

ROS 节点的一个强大功能是在启动该节点时更改参数。此功能使我们能够改变节点名称、主题名称和参数名称。我们无需重新编译代码就能重新配置节点，这样就可以在不同的场景中使用该节点。

一个改变主题名称的例子如下所示：

```
$ roscpp book_tutorials tutorialX topic1:=/level1/topic1
```

此命令将主题名称从 `topic1` 改为 `/level1/topic1`。我相信你现在还不是很明白，但在后面的章节中你会发现它的实用性。

更改节点中的参数和更改主题名称很类似。只需要在参数名称前添加一个下划线，例如：

```
$ roscpp book_tutorials tutorialX _param:=9.0
```

这样参数 (`param`) 就设置为浮点数 9.0。

请记住，你不能使用系统保留关键字的名称，包括：

- `_name` 为节点名称保留的一个特殊关键字。
- `_log` 为记录节点日志存储地址保留的一个关键字。
- `_ip_hostname` 表示 `ROS_IP` 和 `ROS_HOSTNAME` 的关键字。
- `_master` 表示 `ROS_MASTER_URI` 的关键字。
- `_ns` 表示 `ROS_NAMESPACE` 的关键字。

2.2.2 主题

主题是节点间用来传输数据的总线。通过主题进行消息路由不需要节点之间直接连接。这就意味着发布者和订阅者之间不需要知道彼此是否存在。同一个主题也可以有很多个订阅者。

每个主题都是强类型的，发布到主题上的消息必须与主题的消息类型相匹配，并

且节点只能接收类型匹配的消息。一个节点要想订阅一个主题，它们就必须具有相同的消息类型。

ROS 的主题可以使用 TCP/IP 和 UDP 传输。基于 TCP 传输称为 TCPROS，它使用 TCP/IP 长连接。这是 ROS 默认的传输方式。

基于 UDP 传输称为 UDPROS，它是一种低延迟高效率的传输方式，但可能产生数据丢失。所以它最适合于远程操控任务。

ROS 有一个 `rostopic` 工具用于主题操作。它是一个命令行工具，允许我们获取主题的相关信息或直接在网络上发布数据。此工具的参数如下：

- `rostopic bw /topic` 显示主题所使用的带宽。
- `rostopic echo /topic` 将消息输出到屏幕。
- `rostopic find message_type` 按照类型查找主题。
- `rostopic hz /topic` 显示主题的发布率。
- `rostopic info /topic` 输出活动主题、发布的主题、主题订阅者和服务的信息。
- `rostopic list` 输出活动主题的列表。
- `rostopic pub /topic type args` 将数据发布到主题。它允许我们直接从命令行中对任意主题创建和发布数据。
- `rostopic type /topic` 输出主题的类型，或者说主题中发布的消息类型。

我们会在后面的章节中学习如何使用这些命令。

2.2.3 服务

当你需要直接与节点通信并获得应答时，将无法通过主题实现，而需要使用服务。

服务需要由用户开发，节点并不提供标准服务。包含消息源代码的文件存储在 `srv` 文件夹中。

像主题一样，服务关联一个以功能包中 `.srv` 文件名称来命名的服务类型。与其他基于 ROS 文件系统的类型一样，服务类型是功能包名称和 `.srv` 文件名称的组合。例如，`chapter2_tutorials/srv/chapter2_srv1.srv` 文件的服务类型是 `chapter2_tutorials/chapter2_srv1`。

ROS 关于服务的命令行工具有两个：`rossrv` 和 `rosservice`。我们可以通过 `rossrv` 看到有关服务数据结构的信息，并且与 `rosmmsg` 具有完全一致的用法。通过 `rosservice` 可以列出服务列表和查询某个服务。支持的命令如下所示：

- `rosservice call /service args` 根据命令行参数调用服务。
- `rosservice find msg-type` 根据服务类型查询服务。
- `rosservice info /service` 输出服务信息。
- `rosservice list` 输出活动服务清单。
- `rosservice type /service` 输出服务类型。

- `rosservice uri /service` 输出服务的 ROSRPC URI。

2.2.4 消息

一个节点通过向特定主题发布消息，从而将数据发送到另一个节点。消息具有一定的类型和数据结构，包括 ROS 提供的标准类型和用户自定义类型。

消息的类型在 ROS 中按照以下标准命名方式进行约定：功能包名称 / .msg 文件名称。例如，`std_msgs/msg/String.msg` 的消息类型是 `std_msgs/String`。

ROS 使用命令行工具 `rosmmsg` 来获取有关消息的信息。惯用参数如下所示：

- `rosmmsg show` 显示一条消息的字段。
- `rosmmsg list` 列出所有消息。
- `rosmmsg package` 列出功能包的所有消息。
- `rosmmsg packages` 列出所有具有该消息的功能包。
- `rosmmsg users` 搜索使用该消息类型的代码文件。
- `rosmmsg md5` 显示一条消息的 MD5 求和结果。

2.2.5 消息记录包

消息记录包是由 ROS 创建的一组文件。它使用 `.bag` 格式保存消息、主题、服务和其他 ROS 数据信息。你可以在事件发生后，通过使用可视化工具调用和回放数据，检查在系统中到底发生了什么；你可以播放、停止、后退及执行其他操作。

记录包文件可以像实时会话一样在 ROS 中再现情景，在相同时间向主题发送相同的数据。通常情况下，我们可以使用此功能来调试算法。若要使用记录包文件，我们可以使用以下 ROS 工具：

- `rosbag` 用来录制、播放和执行其他操作。
- `rxbag` 用于可视化图形环境中的数据。
- `rostopic` 帮助我们看到节点发送的主题。

2.2.6 节点管理器

ROS 节点管理器向 ROS 系统中其他节点提供命名和注册服务。它像服务一样跟踪和记录主题的发布者和订阅者。节点管理器的作用是使 ROS 节点之间能够相互查找。一旦这些节点找到了彼此，就能建立一种点对点的通信方式。

节点管理器还提供了参数服务器。节点管理器通常使用 `roscore` 命令运行，它会加载 ROS 节点管理器及其他 ROS 核心组件。

2.2.7 参数服务器

参数服务器是可通过网络访问的共享的多变量字典。节点使用此服务器来存储和检索运

行时的参数。

参数服务器使用 XML-RPC 实现并在 ROS 节点管理器下运行。这意味着它的 API 可通过通用的 XMLRPC 库进行访问。

参数服务器使用 XMLRPC 数据类型为参数赋值，包括以下类型：

- 32 位整数
- 布尔值
- 字符串
- 双精度浮点
- ISO 8601 日期
- 列表 (List)
- 基于 64 位编码的二进制数据

ROS 中关于参数服务器的工具是 `rosparam`。其支持的参数如下所示：

- `rosparam list` 列出了服务器中的所有参数。
- `rosparam get parameter` 获取参数值。
- `rosparam set parameter value` 设置参数值。
- `rosparam delete parameter` 删除参数。
- `rosparam dump file` 将参数服务器保存到一个文件。
- `rosparam load file` 加载参数文件到参数服务器。

2.3 理解 ROS 开源社区级

ROS 开源社区级的概念主要关于 ROS 资源，能够通过独立的网络社区分享软件 and 知识。这些资源包括：

- **发行版 (Distribution)** ROS 发行版是可以独立安装的、带有版本号的一系列功能包集。ROS 发行版像 Linux 发行版一样发挥类似的作用。这使得 ROS 软件安装更加容易，而且能够通过一个软件集合来维持一致的版本。
- **软件源 (Repository)** ROS 依赖于共享开源代码与软件源的网站或主机服务，在这里不同的机构能够发布和分享各自的机器人软件与程序。
- **ROS Wiki** ROS Wiki 是用于记录有关 ROS 系统信息的主要论坛。任何人都可以注册账户和贡献自己的文件、提供更正或更新、编写教程以及其他信息。
- **邮件列表 (Mailing list)** ROS 用户邮件列表是关于 ROS 的主要交流渠道，能够交流从 ROS 软件更新到 ROS 软件使用中的各种疑问或信息。

2.4 ROS 系统试用练习

现在，是时候对之前学习的内容进行一些练习了。在下面的几小节中，你会看到一组试

用 TurtleSim 的练习，其中包括如何建立功能包、使用节点、使用参数服务和移动一个虚拟机器人。通过这些练习，你会明确上面这些概念的具体含义。

2.4.1 ROS 文件系统导览

我们通过命令行工具来浏览一下 ROS 的文件系统。我们将要解释最常用的部分。

为了获得功能包和功能包集的信息，或移动文件，需要使用 `rospack`、`rostack`、`roscd` 和 `rosls` 命令。

我们使用 `rospack` 和 `rostack` 来获取有关功能包、功能包集、路径和依赖性等信息。例如，如果你想要找 `turtlesim` 包的路径，可以使用以下命令：

```
$ rospack find turtlesim
```

你将获得以下信息：

```
/opt/ros/ fuerte /share /turtlesim
```

同样的，如果你想要找到你已经在系统中安装过的某个功能包集，示例如下：

```
$ rostack find 'nameofstack'
```

想要获得功能包或功能包集下面的文件列表，那么需要使用：

```
$ rosls turtlesim
```

你将获得以下信息：

```
cmake      images      mimic  srv          turtle_teleop_key  
draw_square manifest.xml msg turtlesim_node
```

如果你想进入到某个文件夹，可以使用 `roscd` 命令：

```
$ roscd turtlesim
```

```
$ pwd
```

你将获得以下新路径：

```
/opt/ros/ fuerte /share /turtlesim
```

2.4.2 创建工作空间

在开始具体工作之前，首先创建工作空间。在这个工作空间中，我们将会完成本书中所使用的所有代码。

若要查看 ROS 正在使用的工作空间，请使用下面的命令：

```
$ echo $ROS_PACKAGE_PATH
```

你将会看到如下信息：

```
/opt/ros/ fuerte /share : /opt/ros/ fuerte /stacks
```

我们将要创建的文件夹是在 `~/dev/rosbook/` 中。若要新建此文件夹，使用以下命令：

```
$ cd ~  
$ mkdir -p dev/rosbook
```

一旦我们在目录下建好了文件夹，就有必要将此新路径添加到 `ROS_PACKAGE_PATH`。要做到这一点，只需要在 `~/.bashrc` 文件的末尾添加一个新行：

```
$ echo "export ROS_PACKAGE_PATH=~/dev/rosbook:${ROS_PACKAGE_PATH}" >>  
~/.bashrc  
$ . ~/.bashrc
```

现在我们已经可以在 ROS 系统中创建和配置完成自己的新文件夹。

2.4.3 创建 ROS 功能包

就像之前所说，你也可以手动创建功能包。但是为了避免那些繁琐的工作，最好使用 `roscreeate-pkg` 命令行工具。

使用以下命令在之前建立的文件夹下创建新的功能包：

```
$ cd ~/dev/rosbook  
$ roscreeate-pkg chapter2_tutorials std_msgs rospy roscpp
```

此命令的格式包括功能包的名称和依赖项；在这个示例中，依赖项包括 `std_msgs`、`rospy` 和 `roscpp`。如以下命令行所示：

```
roscreeate-pkg [package_name] [depend1] [depend2] [depend3]
```

这些依赖项包括：

- `std_msgs` 包含了常见消息类型，表示基本数据类型和其他基本的消息构造，如多维数组。
- `rospy` 一个 ROS 的纯 Python 客户端库。
- `roscpp` 使用 C++ 实现 ROS 的各种功能。它提供了一个客户端库。程序员能够调用这些接口快速完成与 ROS 的主题、服务和参数相关的开发工作。

如果所有步骤都正确执行，结果如下图所示：

```
Created package directory /home/aaronmr/dev/rosbook/chapter2_tutorials  
Created include directory /home/aaronmr/dev/rosbook/chapter2_tutorials/include/chapter2_tutorials  
Created cpp source directory /home/aaronmr/dev/rosbook/chapter2_tutorials/src  
Created package file /home/aaronmr/dev/rosbook/chapter2_tutorials/Makefile  
Created package file /home/aaronmr/dev/rosbook/chapter2_tutorials/manifest.xml  
Created package file /home/aaronmr/dev/rosbook/chapter2_tutorials/CMakeLists.txt  
Created package file /home/aaronmr/dev/rosbook/chapter2_tutorials/mainpage.dox  
  
Please edit chapter2_tutorials/manifest.xml and mainpage.dox to finish creating your package
```

正如我们之前看到的，你可以使用 `rospack`、`roscd` 和 `rosls` 命令来获取新的功能包信息：

- `rospack find chapter2_tutorials` 此命令用于查找路径。
- `rospack depends chapter2_tutorials` 此命令用于查看依赖关系。

- `rosls chapter2_tutorials` 此命令用于查看内容。
- `roscd chapter2_tutorials` 此命令会更改实际路径。

2.4.4 编译 ROS 功能包

一旦你创建了一个功能包，并且编写了一些代码，就需要编译功能包了。当你编译功能包的时候，主要是代码的编译过程。

为了编译功能包，可以使用 `rosmake` 命令：

```
$ rosmake chapter2_tutorials
```

在几秒之后，你会看到：

```
[ rosmake ] Results:
[ rosmake ] Cleaned 5 packages.
[ rosmake ] Built 5 packages with 0 failures.
[ rosmake ] Summary output to directory
[ rosmake ] /home/aaronmr/.ros/rosmake/rosmake_output-20130713-183756
```

如果没有看到错误提示信息，说明功能包编译成功。

2.4.5 使用 ROS 节点

正如我们在 2.2.1 章节中解释的，节点都是可执行程序，这些可执行文件位于 `packagename/bin` 目录中。要学习和了解有关节点的知识，我们要使用一个名为 `turtlesim` 的功能包进行练习。

如果你进行了 ROS 系统的完整安装，那么你已经有了 `turtlesim` 功能包。如果还没有，请使用以下命令安装：

```
$ sudo apt-get install ros-fuerte-ros-tutorials
```

在开始之前，必须使用如下命令启动 `roscore`：

```
$ roscore
```

为了获得节点信息，可以使用 `roscnode` 命令。为了查看命令接受哪些参数，可以输入以下命令：

```
$ roscnode
```

你会获得一个可接受参数的清单，如下图所示：

```
roscnode is a command-line tool for printing information about ROS Nodes.

Commands:
  roscnode ping      test connectivity to node
  roscnode list      list active nodes
  roscnode info      print information about node
  roscnode machine   list nodes running on a particular machine or list machines
  roscnode kill      kill a running node

Type roscnode <command> -h for more detailed usage, e.g. 'roscnode ping -h'
```

如果你想获得关于这些参数更详细的解释，请使用以下命令：

```
$ rosnode <param> -h
```

现在 roscore 已经运行，我们想要获取正在运行的节点的相关信息：

```
$ rosnode list
```

你会看到运行的节点仅有 /roscout。这是正常的，因为这个节点总是随着 roscore 的运行而运行。

通过使用参数我们可以获得此节点的所有信息。也可以使用下列命令获得更详细的信息：

```
$ rosnode info
```

```
$ rosnode ping
```

```
$ rosnode machine
```

```
$ rosnode kill
```

现在我们要用 rosrun 命令启动一个新的节点，如下所示：

```
$ rosrun turtlesim turtlesim_node
```

我们看到出现了一个新的窗口，窗口中间有一个小海龟，如下图所示：



如果我们再去查看节点列表，会看到出现了一个新的节点，叫做 /turtlesim。

你可以通过使用 rosnode info Nodename 命令查看节点信息。你可以看到很多可以用于程序调试的信息：

```
$ rosnode info /turtlesim
Node [/turtlesim]
Publications:
  * /turtle1/color_sensor [turtlesim/Color]
  * /rosout [rosgraph_msgs/Log]
  * /turtle1/pose [turtlesim/Pose]
Subscriptions:
  * /turtle1/command_velocity [unknown type]
Services:
  * /turtle1/teleport_absolute
  * /turtlesim/get_loggers
  * /turtlesim/set_logger_level
  * /reset
  * /spawn
  * /clear
  * /turtle1/set_pen
  * /turtle1/teleport_relative
  * /kill
contacting node http://aaronmr-laptop:42791/ ...
Pid: 28337
Connections:
  * topic: /rosout
  * to: /rosout
  * direction: outbound
  * transport: TCPROS
```

在以上信息中，我们可以看到发布者（及相应主题）、订阅者（及相应主题）和该节点具有的服务（srv）及它们各自唯一的名称。

接下来介绍如何使用主题和服务与该节点进行交互。

2.4.6 使用主题与节点交互

要进行交互并获取主题的信息，可以使用 `rostopic` 工具。此工具接受以下参数：

- `rostopic bw` 显示主题所使用的带宽。
- `rostopic echo` 将消息输出到屏幕。
- `rostopic find` 按照类型查找主题。
- `rostopic hz` 显示主题的发布频率。
- `rostopic info` 输出活动主题的信息。
- `rostopic list` 输出活动主题的列表。
- `rostopic pub` 将数据发布到主题。

- `rostopic type` 输出主题的类型。

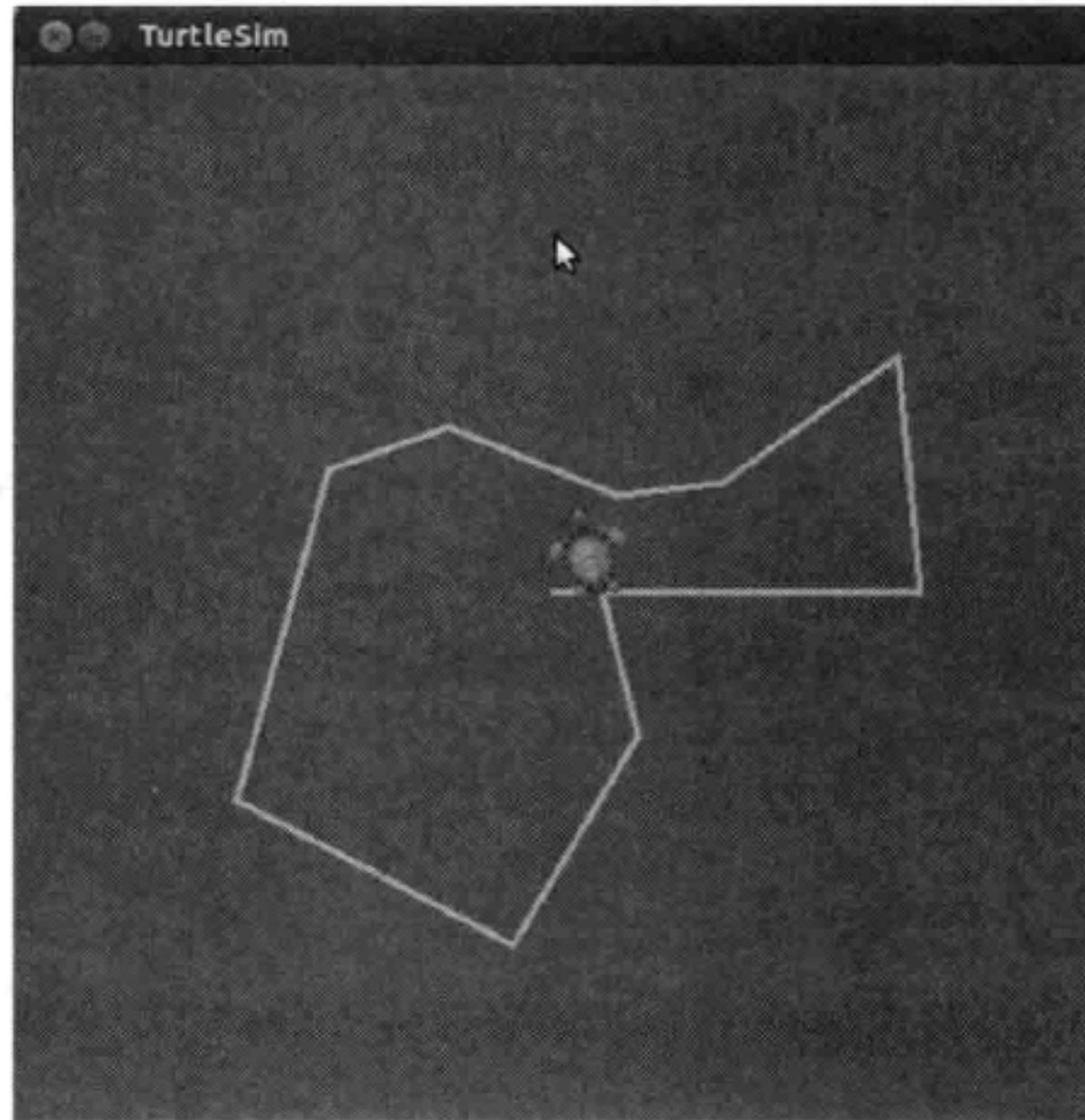
如果想要查看有关这些参数的详细信息，请使用 `-h`，如下所示：

```
$ rostopic bw -h
```

通过使用 `pub` 参数，可以发布任何节点都可以订阅的主题。我们只需要用正确的名称将主题发布出去。我们将会在以后做这个测试，现在要使用一个节点，并让节点做如下工作：

```
$ rosrun turtlesim turtle_teleop_key
```

通过节点订阅的主题，我们可以使用箭头键移动乌龟，如下图所示：



为什么 `turtle_teleop_key` 执行时，小海龟会移动呢？

如果你想要看到 `/teleop_turtle` 和 `/turtlesim` 节点的信息，可以看到在下面的代码中，在发布者 (Publications) 部分的第一个节点有一个主题叫 `*/turtle1/ command_velocity [turtlesim/Velocity]`；在订阅者 (Subscriptions) 部分的第二个节点有 `*/turtle1/command_velocity [turtlesim/Velocity]`：

```
$ rosnodetool info /teleop_turtle
```

```
Node [/teleop_turtle]
```

```
...
```

```
Publications:
```

```
* /turtle1/command_velocity [turtlesim/Velocity]
```

```
...
```

```
$ rosnode info /turtlesim
```

```
Node [/turtlesim]
```

```
...
```

```
Subscriptions:
```

```
  * /turtle1/command_velocity [turtlesim/Velocity]
```

```
...
```

这意味着前面的节点发布了一个主题，而后面的节点订阅了这个主题。

你可以使用以下命令查看主题清单：

```
$ rostopic list
```

```
/rosout
```

```
/rosout_agg
```

```
/turtle1/color_sensor
```

```
/turtle1/command_velocity
```

```
/turtle1/pose
```

通过使用 `echo` 参数，可以查看节点发出的信息。运行以下命令行并使用箭头键触发消息，查看消息产生时发送了哪些数据：

```
$ rostopic echo /turtle1/command_velocity
```

你会看到类似如下显示：

```
linear: 2.0
```

```
angular: 0.0
```

```
---
```

```
linear: 0.0
```

```
angular: 2.0
```

```
---
```

你可以使用以下命令行查看由主题发送的消息类型：

```
$ rostopic type /turtle1/command_velocity
```

```
turtlesim/Velocity
```

如果你想要看到消息字段，可以使用以下命令：

```
$ rosmmsg show turtlesim/Velocity
```

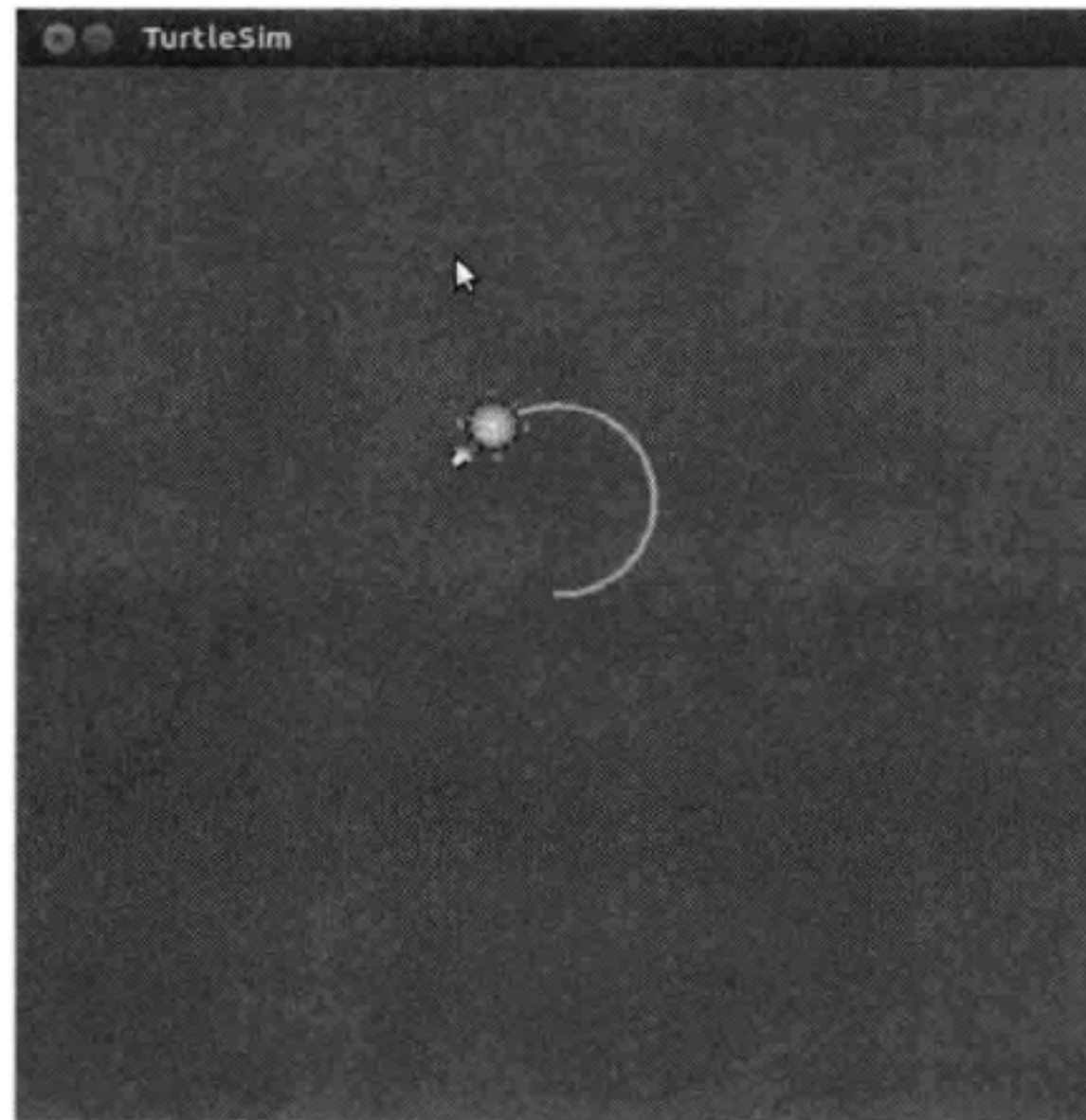
```
float32 linear
```

```
float32 angular
```

这些工具非常有用，因为我们可以通过这些工具使用 `rostopic pub [topic] [msg_type] [args]` 命令直接发布主题：

```
$ rostopic pub -1 /turtle1/command_velocity turtlesim/Velocity -- 1 1
```

你会看到小海龟做曲线运动，如下图所示。



2.4.7 学习如何使用服务

服务是能够使节点之间相互通信的另一种方法。服务允许节点发送请求和接收响应。

我们要使用 `rosservice` 工具与服务进行交互。此命令接受的参数如下所示：

- `rosservice args /service` 输出服务参数。
- `rosservice call /service args` 根据命令行参数调用服务。
- `rosservice find msg-type` 根据服务类型查询服务。
- `rosservice info /service` 输出服务信息。
- `rosservice list` 输出活动服务清单。
- `rosservice type /service` 输出服务类型。
- `rosservice uri /service` 输出服务的 ROSRPC URI。

我们要使用以下命令列出在 `turtlesim` 节点运行时系统提供的服务。如果你运行了这个命令，却没有任何反应，那么请记住以后要先运行 `roscore` 并启动 `turtlesim` 节点：

```
$ rosservice list
```

你会获得以下输出：

```
/clear  
/kill
```



```

/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtle2/set_pen
/turtle2/teleport_absolute
/turtle2/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level

```

如果你想查看某个服务的类型，例如 `/clear` 服务，请使用：

```
$ rosservice type /clear
```

你会获得：

```
std_srvs/Empty
```

若要调用服务，你需要使用 `rosservice call [service] [args]` 命令。所以如果想要调用 `/clear` 服务，请使用：

```
$ rosservice call /clear
```

在 `turtlesim` 的窗口中，你会看到由于小海龟移动所产生的线条消失了。

现在我们尝试其他的 service，例如 `/spawn` 服务。这项服务将在不同的方向和不同的位置创建另一只小海龟。开始之前，我们要去查看以下类型的消息：

```
$ rosservice type /spawn | rossrv show
```

我们会获得以下参数：

```

float32 x
float32 y
float32 theta
string name
---
string name

```

通过这些字段，可以知道如何来调用服务。我们需要新海龟位置的 `x` 和 `y`、方向 `theta` 和新海龟的名称：

```
$ rosservice call /spawn 3 3 0.2 "new_turtle"
```

我们会获得以下结果：



2.4.8 使用参数服务器

参数服务器用于存储所有节点均可访问的共享数据。ROS 中用来管理参数服务器的工具称为 `rosparam`。接受的参数如下所示：

- `rosparam set parameter value` 设置参数值。
- `rosparam get parameter` 获取参数值。
- `rosparam load file` 加载参数文件到参数服务器。
- `rosparam delete parameter` 删除参数。
- `rosparam dump file` 将参数服务器保存到一个文件。
- `rosparam list` 列出了服务器中的所有参数。

例如，查看被所有节点所使用的服务器参数：

```
$ rosparam list
```

我们会获得以下输出：

```
/background_b  
/background_g  
/background_r  
/roscdistro  
/roslaunch/uris/host_aaronmr_laptop__60878  
/rosversion  
/run_id
```

上面的背景 (background) 参数是 turtlesim 节点的参数。窗口初始化为蓝色，这些参数可以改变窗口的颜色。如果你想要读取某个值，可以使用 get 参数：

```
$ rosparam get /background_b
```

为了设定一个新的值，可以使用 set 参数：

```
$ rosparam set /background_b 100
```

命令行工具 rosparam 的另外一个重要特性是参数转存。通过该参数，你可以保存或加载参数服务器的内容。

我们使用 rosparam dump [file_name] 来保存参数服务器：

```
$ rosparam dump save.yaml
```

使用 rosparam load [file_name] [namespace] 向参数服务器加载一个新的数据文件：

```
$ rosparam load load.yaml namespace
```

2.4.9 创建节点

在本节中，我们要学习如何创建两个节点：一个发布一些数据，另一个接收这些数据。也就是说，两个节点之间使用最基本的方式进行通信，发送和接收数据并使用这些数据来做些工作：

```
$ roscd chapter2_tutorials/src/
```

创建两个文件并分别命名为 example1_a.cpp 和 example1_b.cpp。

example1_a.cpp 文件将会发送带有节点名称的数据，example1_b.cpp 文件会把这些数据 displays 在 shell 窗口中。

将下面的代码复制到 example1_a.cpp 文件或者从下载的本书示例代码中找到它：

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "example1_a");
    ros::NodeHandle n;
    ros::Publisher chatter_pub = n.advertise<std_
msgs::String>("message", 1000);
    ros::Rate loop_rate(10);
    while (ros::ok())
    {
        std_msgs::String msg;
        std::stringstream ss;
        ss << " I am the example1_a node ";
```



```

    msg.data = ss.str();
    //ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
}
return 0;
}

```

这里是一些关于上面代码的进一步解释。要包含的头文件是 `ros/ros.h`、`std_msgs/String.h` 和 `sstream`。其中，`ros/ros.h` 包含了使用 ROS 节点所有必要的文件，而 `std_msgs/String.h` 则包含了我们要使用的消息类型：

```

#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

```

启动该节点并设置其名称，请记住该名称必须是唯一的：

```
ros::init(argc, argv, "example1_a");
```

下面是设置节点进程的句柄：

```
ros::NodeHandle n;
```

将节点设置成发布者，并将所发布主题的类型和名称告知节点管理器。第一个参数是消息的名称，设置为 `message`；第二个参数是缓冲区的大小。如果主题发布数据速度较快，那么将缓冲区设置为 1000 个消息，如下所示：

```
ros::Publisher chatter_pub = n.advertise<std_msgs::String>("message",
1000);
```

在本例中，设置发送数据的频率为 10Hz：

```
ros::Rate loop_rate(10);
```

当收到 `Ctrl + C` 的按键消息或 ROS 停止当前节点运行时，`ros::ok()` 库会执行停止节点运行的命令：

```
while (ros::ok())
{

```

在这里，我们创建一个消息变量，变量的类型必须符合发送数据的要求：

```
std_msgs::String msg;
std::stringstream ss;
ss << " I am the example1_a node ";
msg.data = ss.str();

```

这样，消息被发布：

```
chatter_pub.publish(msg);
```

如果有一个订阅者出现，ROS 就会更新和读取所有主题：

```
ros::spinOnce();
```

按照 10Hz 的频率将程序挂起：

```
loop_rate.sleep();
```

将下面的代码复制到 example1_b.cpp 文件或者从下载的本书示例代码中找到它：

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "example1_b");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("message", 1000, chatterCallback);
    ros::spin();
    return 0;
}
```

这里是一些对上面代码的进一步解释。首先要包含头文件和主题所使用的消息类型：

```
#include "ros/ros.h"
#include "std_msgs/String.h"
```

每次该节点收到一条消息时都将调用此函数，我们就可以使用或处理数据。在本示例中，我们将收到的数据在命令行窗口中输出出来：

```
void messageCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}
```

ROS_INFO() 用于在命令行窗口中输出数据。

创建一个订阅者，并从主题获取以 message 为名称的消息数据。设置缓冲区为 1000 个消息，处理消息句柄的回调函数为 messageCallback：

```
ros::Subscriber sub = n.subscribe("message", 1000, messageCallback);
```

ros::spin() 库是节点读取数据的消息响应循环，当消息到达的时候，回调函数 messageCallback 就会被调用。当用户按下 Ctrl + C 时，节点会退出消息循环，于是循环结束：

```
ros::spin();
```

2.4.10 编译节点

当使用 chapter2_tutorials 功能包时，需要自行编辑 CMakeLists.txt 文件。你

可以使用你喜欢的编辑器或直接使用 `roscd` 工具。这里我们将会使用 `VIM` 编辑器打开这个文件：

```
$ roscd chapter2_tutorials CMakeLists.txt
```

将以下命令行复制到文件的末尾处：

```
rosbuild_add_executable(example1_a src/example1_a.cpp)
rosbuild_add_executable(example1_b src/example1_b.cpp)
```

这些命令行会在 `/bin` 文件夹下创建两个可执行文件。

现在我们使用 `rosmake` 工具编译功能包就会编译全部的节点：

```
$ rosmake chapter2_tutorials
```

如果在你的电脑上还没有启动 ROS，需要首先调用：

```
$ roscore
```

你可以使用 `rostopic list` 命令检查 ROS 是否运行：

```
$ rostopic list
```

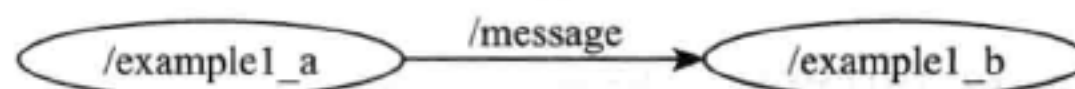
然后，在不同的 shell 窗口下分别运行以下命令：

```
$ roslaunch chapter2_tutorials example1_a
$ roslaunch chapter2_tutorials example1_b
```

如果你检查一下 `example1_b` 正在运行的 shell 窗口，你就会看到以下信息：

```
...
[ INFO] [1355228542.283236850]: I heard: [ I am the example1_a node ]
[ INFO] [1355228542.383221843]: I heard: [ I am the example1_a node ]
[ INFO] [1355228542.483249861]: I heard: [ I am the example1_a node ]
...
```

我们可以在下图中看到正在发生的消息传递。`example1_a` 节点发布消息到 `message` 主题，同时节点 `example2_b` 订阅了这个主题。



你可以使用 `rostopic` 和 `rostopic` 命令来调试和查看当前节点的运行状况。尝试使用以下命令：

```
$ rostopic list
$ rostopic info /example1_a
$ rostopic info /example1_b
$ rostopic list
$ rostopic info /message
$ rostopic type /message
$ rostopic bw /message
```

2.4.11 创建 msg 和 srv 文件

在这一节中，我们将会学习如何在节点中创建 `msg` 和 `srv` 文件。它们用于定义传输

数据的类型和数据值的文件。ROS 会根据这些文件内容自动地为我们创建所需的代码，以便 msg 和 srv 文件能够被节点所使用。第一步，我们先学习 msg 文件。

在上一节所使用的示例中，我们已经创建了两个具有标准类型 message 的节点。现在，我们要学习如何使用 ROS 工具创建自定义消息。

首先，在 chapter2_tutorials 功能包下创建 msg 文件夹，并在其中创建一个新的文件 chapter2_msg1.msg。在文件中增加以下行：

```
int32 A
int32 B
int32 C
```

现在编辑 CMakeList.txt，从 # rosbuilt_genmsg() 这一行中删除 #，然后使用 rosmake 命令编译功能包：

```
$ rosmake chapter2_tutorials
```

为了检查编译是否正确，可以使用 rosmmsg 命令：

```
$ rosmmsg show chapter2_tutorials/chapter2_msg1
```

如果你在 chapter2_msg1.msg 文件中看到一样的内容，说明编译正确。

现在创建一个 srv 文件。在 chapter2_tutorials 文件夹下创建一个名为 srv 的文件夹，并新建文件 chapter2_srv1.srv，在文件中增加以下行：

```
int32 A
int32 B
int32 C
---
int32 sum
```

编辑 CMakeList.txt 文件，从 #rosbuilt_gensrv() 这一行中删除 #，并且使用 rosmake chapter2_tutorials 命令编译功能包。

你可以通过 rossrv 工具来检查编译是否正确：

```
$ rossrv show chapter2_tutorials/chapter2_srv1
```

如果你在 chapter2_srv1.srv 文件中看到相同的内容，说明编译正确。

2.4.12 使用新建的 srv 和 msg 文件

首先，我们将会学习如何创建一个服务并且在 ROS 中使用它。该服务将会对三个整数求和。我们需要两个节点，一个服务器一个客户端。

在 chapter2_tutorials 功能包中，新建两个节点并使用 example2_a.cpp 和 example2_b.cpp 为名称。别忘了要在 src 文件夹下创建这两个文件。

在第一个文件 example2_a.cpp 中，添加以下代码：

```
#include "ros/ros.h"
#include "chapter2_tutorials/chapter2_srv1.h"
```

```

bool add(chapter2_tutorials::chapter2_srv1::Request &req,
         chapter2_tutorials::chapter2_srv1::Response &res)
{
    res.sum = req.A + req.B + req.C;
    ROS_INFO("request: A=%ld, B=%ld C=%ld", (int)req.A, (int)req.B,
(int)req.C);
    ROS_INFO("sending back response: [%ld]", (int)res.sum);
    return true;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_3_ints_server");
    ros::NodeHandle n;

    ros::ServiceServer service = n.advertiseService("add_3_ints", add);
    ROS_INFO("Ready to add 3 ints.");
    ros::spin();

    return 0;
}

```

还需要包含必要的头文件和我们创建的 srv 文件:

```

#include "ros/ros.h"
#include "chapter2_tutorials/chapter2_srv1.h"

```

这个函数会对 3 个变量求和, 并将计算结果发送给其他节点:

```

bool add(chapter2_tutorials::chapter2_srv1::Request &req,
         chapter2_tutorials::chapter2_srv1::Response &res)

```

在这里, 创建服务并在 ROS 中发布广播, 如下:

```

ros::ServiceServer service = n.advertiseService("add_3_ints", add);

```

在第 2 个文件 example2_b.cpp 中, 增加以下代码:

```

#include "ros/ros.h"
#include "chapter2_tutorials/chapter2_srv1.h"
#include <cstdlib>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_3_ints_client");
    if (argc != 4)
    {
        ROS_INFO("usage: add_3_ints_client A B C ");
        return 1;
    }

    ros::NodeHandle n;
    ros::ServiceClient client = n.serviceClient<chapter2_
tutorials::chapter2_srv1>("add_3_ints");
    chapter2_tutorials::chapter2_srv1 srv;

```

```

    srv.request.A = atoll(argv[1]);
    srv.request.B = atoll(argv[2]);
    srv.request.C = atoll(argv[3]);
    if (client.call(srv))
    {
        ROS_INFO("Sum: %ld", (long int)srv.response.sum);
    }
    else
    {
        ROS_ERROR("Failed to call service add_3_ints");
        return 1;
    }

    return 0;
}

```

使用 `add_3_ints` 为名称创建一个服务的客户端:

```

ros::ServiceClient client = n.serviceClient<chapter2_
tutorials::chapter2_srv1>("add_3_ints");

```

下面创建 `srv` 文件的一个实例, 并且加入需要发送的数据值。如果你还记得, 这个消息需要 3 个字段:

```

chapter2_tutorials::chapter2_srv1 srv;
srv.request.A = atoll(argv[1]);
srv.request.B = atoll(argv[2]);
srv.request.C = atoll(argv[3]);

```

这些代码会调用服务并发送数据。如果调用成功, `call()` 函数会返回 `true`; 如果没成功, `call()` 函数会返回 `false`:

```

if (client.call(srv))

```

为了编译节点, 在 `CMakeList.txt` 文件中增加以下行:

```

rosbuild_add_executable(example2_a src/example2_a.cpp)
rosbuild_add_executable(example2_b src/example2_b.cpp)

```

现在执行以下命令:

```

$ rosmake chapter2_tutorials

```

为了启动节点, 需要执行以下命令行:

```

$roslaunch chapter2_tutorials example2_a
$roslaunch chapter2_tutorials example2_b 1 2 3

```

并且你会看到如下显示:

```

Node example2_a
[ INFO] [1355256113.014539262]: Ready to add 3 ints.
[ INFO] [1355256115.792442091]: request: A=1, B=2 C=3
[ INFO] [1355256115.792607196]: sending back response: [6]

```


Node example2_b

```
[ INFO] [1355256115.794134975]: Sum: 6
```

现在将要用自定义的 msg 文件来创建节点。这个例子也一样，创建 example3_a.cpp 和 example3_b.cpp 文件，同时调用 chapter2_msg1.msg。

将下面的代码放在 example3_a.cpp 文件中：

```
#include "ros/ros.h"
#include "chapter2_tutorials/chapter2_msg1.h"
#include <sstream>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "example1_a");
    ros::NodeHandle n;
    ros::Publisher pub = n.advertise<chapter2_tutorials::chapter2_
msg1>("message", 1000);
    ros::Rate loop_rate(10);
    while (ros::ok())
    {
        chapter2_tutorials::chapter2_msg1 msg;
        msg.A = 1;
        msg.B = 2;
        msg.C = 3;
        pub.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

将下面的代码放在 example3_b.cpp 文件中：

```
#include "ros/ros.h"
#include "chapter2_tutorials/chapter2_msg1.h"

void messageCallback(const chapter2_tutorials::chapter2_
msg1::ConstPtr& msg)
{
    ROS_INFO("I heard: [%d] [%d] [%d]", msg->A, msg->B, msg->C);
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "example1_b");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("message", 1000, messageCallback);
    ros::spin();
    return 0;
}
```

如果现在运行这两个节点，将会看到如下信息：

```
...  
[ INFO] [1355270835.920368620]: I heard: [1] [2] [3]  
[ INFO] [1355270836.020326372]: I heard: [1] [2] [3]  
[ INFO] [1355270836.120367449]: I heard: [1] [2] [3]  
[ INFO] [1355270836.220266466]: I heard: [1] [2] [3]  
...
```

2.5 本章小结

本章介绍了 ROS 系统的基本架构及其典型的工作方式。学习了一些基本概念、工具及如何同节点、主题和服务进行交互的示例。一开始，所有这些概念可能看起来有些复杂且不太实用，但在后面的章节中，你会逐渐理解如何使用这些概念所代表的对象。

各位读者最好在继续后续章节的学习之前，对这些概念及示例进行练习，因为在后面的章节里，我们将假定你已经熟悉所有的概念及其用途。

请注意如果想查询某个名词或功能的解释，且无法在这本书中找到相关内容或答案，那么可以通过以下链接访问 ROS 官方资源 <http://www.ros.org>。而且，你还可以通过访问 ROS 社区 <http://answers.ros.org> 提出自己的问题。

在下一章中，将学习如何使用 ROS 工具查询调试信息和实现数据的可视化。这些将帮助你理解 ROS 的运行状态，发现软件运行的问题，并且指导你对它的运行进行调整。