

ROS

机器人编程

从基本概念到机器人应用程序编程实战!

机器人操作系统

表允哲 | 赵汉哲 | 郑黎蝠 | 林泰勋

ROS机器人编程

作者 表允哲 赵汉哲 郑黎蠎 林泰勋

发行日 2017年12月22日

出版社 ROBOTIS Co., Ltd.

地址 #1505, 145, Gasan Digital 1-ro, GeumCheon-gu, Seoul, Republic of Korea

E-mail contactus2@robotis.com

主页 www.robotis.com

ISBN 979-11-962307-2-2

ALL RIGHTS RESERVED

Copyright © 2017 ROBOTIS Co., Ltd.

本著作物受著作权法保护。禁止无端转载和复制。

使用本书的部分或全部内容，需要获得著作权者和ROBOTIS（株）的书面同意。

ROS 机器人编程

表允哲 赵汉哲 郑黎蟠 林泰勋

即使

机器人工程除了在工业机器人行业以外还没有任何商业模式，但是它凭借未来产业和下一代增长引擎的名义，被期待为蓝筹行业。问题是这些故事在10年前和现在是一样的，但还没有明确的解答。原因是什么？会有多种看法，但我认为机器人产业要形成新的商业模式还有很多不足的地方，还有很多需要解决的问题。我认为这个解决方案应该是跨越国界的超越性合作，我还认为软件平台和支持这些平台的社区可以解决这些问题。参与开发机器人操作系统ROS的有学术研究人员、工业一线开发人员和业余爱好者等人群。此外，不仅是机器人专业人士，还有网络专家、计算机科学家、计算机视觉专家也大有人参加，因此实现着机器人领域和其他诸多专业的知识的融合。机器人预计从现在开始以不同于以往的方式发展。我们认为，通过开放和合作，我们可以解决至今尚未解决的问题。我想说，这不是未来的产业，而已经是现在的产业。

本书是我们四个人在完成多项有关ROS的项目的过程中获得的经验的基础上编写的ROS机器人编程书。为了初次接触ROS的读者，尽可能涵盖了初学者所需要的大多数内容。进而包括了可以应用于嵌入式系统、移动机器人和机械手臂的ROS内容。为了尽量有助于ROS的初学者，将ROS的介绍和各部分所需要的信息都标上脚注，让读者可以自行在网上找到。真诚希望更多的人通过本书了解和参与当今日益加速的机器人工程的跨界式的知识融合和跨领域的合作。

下面我想对那些帮助我让这本书面世的人们表达我的谢意。我非常感谢ROS专家Lee Ji-Hoon博士，Ahn Byung-kyu，Jung Keun-man，Seong Chang-Hyun博士和Koo Sung-Yong博士，他们对有着诸多缺点的我总是给予新的指导。我想和他们说，我将来想和你们一起做出更多的事情。另外，感谢对这本不为人所知的ROS专业书籍的出版提供大力支持的Han Chang-Hoon代表，以及用最佳的编排使得读者更加容易理解ROS的Lee Yin-Ho，Son Will，Jason，Kim Kayla老师。我还想感谢ROBOTIS的成员们。ROBOTIS，一家以“机器人是什么？”这一哲学疑问开始的公司，我想因为有了他们才有了这本书。我还要感谢立志通过开源代码让更多人一起思考和开发机器人的“开源团队（OST）”的成员，感谢开源生态界和社区的协助者Kim Jin-Wook先生，并向本书的共同作者，“ROS英雄团”，赵汉哲、郑黎蝗和林泰勋鼓掌致意。还非常感谢正在一起编写日语版ROS书籍的我的指导教授日本九州大学的倉爪亮教授和長谷川勉教授。他们以丰富的研究经验引领我走向了研究者之路，至今也在不断地学到很多知识。借此机会，我想向教授传达我的心声“非常感谢您，给予我受益终生的敦敦教诲”。同时也向为这本书成功出版，不断激励我坚持不懈，向我积极反馈好的

建议的开源机器人社区 Park Hyeng-II和运营组以及为我加油地会员们，为开源机器人平台开发付出热情的公开项目负责人表示衷心的感谢。将来，我想继续以开源机器人为主题一起进行讨论，同时携手进行更有趣的项目。还要向facebook的机器人工程公开群运营组和为韩国机器人产业发展献计献策的群组成员表示感谢。也要感谢与我一起燃烧青春的机器人比赛团队“ROBIT”和机器人研究会“ROLAB”的师兄弟们。还有一点，本书之所以能面世的重要原因之一是本人所属的公司“ROBOTIS”，对开源代码编著活动及交流活动积极提供支持，为此向CTO Ha Yin-Yong和CEO Kim Byung-Soo表示谢意。

最后向我心爱的家人致谢。首先，向给予我无限的爱的父母说一声“我会永远敬爱你们，也时时刻刻心存对你们的感恩之心”。还有在旁边一直鼓励我，支持我的岳父母，“我对你们常怀感恩，一直爱着你们”。爱妻Ken-Mi, 你一直坚守在我的旁边，一直照顾着我，非常感谢你，爱你，以后我们要更幸福地生活。世界上最可爱的我的心脏们，儿子Ji-an还有女儿Ji-Woo, 我为了你们会努力成为使这个世界变得更美好幸福的好父亲。

2017年7月

表允哲

机器人

包含着很多技术要素，因此需要多方面的专业技术。实际上，机器人为了要进入到我们的生活当中且被广泛运用，目前尚存在着很多技术局限性，需要进行更多的研究。想要克服当前的问题，专家、相关企业、一般用户需要携手努力一起发展现状。除了机器人的制作和应用之外，我们需要一个协作和开发的平台，我认为这就是ROS平台。ROS具备着降低技术壁垒和有益于传播的各种因素。通过ROS平台，希望积累更多的知识和技术，使得更新更进步的机器人加入到我们的生活中。

嵌入式系统控制传感器和驱动器，进行实时数据处理，作为构成机器人的一部分，起着重要作用。通常为了实时地进行数据处理，会使用微控制器。这本书以在嵌入式系统中使用ROS的方法和基本示例为中心进行叙述。我希望这些内容为那些使用ROS搭建机器人嵌入式系统的用户助一臂之力。

我向Pyo Hyung-Joon, Park Hyung-II和Park Byung-Hoon为创建OpenCR而付出的努力表示感谢，他们弥补了我的很多不足之处，也共度了很多的美好时光，因此给我留下了很多回忆。也非常感谢开源项目团队（OST）的成员，与他们在一起我觉得很活跃很开心。我衷心感谢Kim In-Wook，从我刚步入社会时开始到现在，一直与我同在，在遇到困难时为给予我很多好的建议和激励。同时，也要向为我人生提供新的挑战机会的ROBOTIS的Kim Byung-Soo代表表示衷心的谢意。当我还小时候，我看到了Kim Byung-Soo代表在HiTel数码兴趣协会上的写作，通过他的文章我学到很多相关知识，也成了从事机器人行业的动机。这使得我如今在他的公司开发机器人。

总以工作繁忙为借口没能陪伴儿子，心存亏欠，在此立下决心为我心爱的儿子成为好爸爸。当笔者备受压力时，默默陪伴我，支持我的爱妻Kyung-Sun，谢谢你理解和体谅我的任性，借此机会向你表达歉意的同时也要跟你说一声我爱你。

2017年7月

赵汉哲

让 我们一起如愿以偿地制作我们想要制作的机器人吧！以前我们曾经拿着与书本配套的机器人套件来制作机器人，仅仅做出简单的动作，也会很满足地说“这就是机器人”。然而，近年来，随着计算机性能的发展、设备的成本降低、获取素材变得迅速与便捷，许多有关机器人的概念被重新定义着。机器人爱好者们已经大量开始投入到机器人制作当中，并且信息量变得庞大，随着各个品牌制作商把自己的产品进行自动化，如今汽车、飞机甚至潜艇都已成为机器人的平台。人们开始跨越科学领域，开始融合各自的技术，结果机器人终于开始被制作成梦想中的形态。我们可以说现在机器人行业正面临着全盛时期，但从中有些人可能开始跟不上日益变得高性能、高端化的潮流，从而机器人产业可能将成为只有懂机器人的人才可以生存以及得到发展的领域。

我觉得ROS对他们来说是雪中送炭，即使没有专攻该领域也能学到相关技术，从而节省时间和金钱。如果在使用过程中遇到问题，可以向制作人提问并得到反馈，将会形成互助体系。如今，像宝马等公司也正在引入ROS。通过ROS来运营公司或与其他公司合作，现可以说占据ROS的优势就会成为该领域的赢家。

我希望与本书的读者将来可以在ROS领域里相见。开源代码团队的组员们，尤其是给予我参与此书编著的表允皙博士，为诸多项目同甘共苦的赵汉哲和林泰勋，向你们表示感谢。

同时想感谢在机器人领域里慷慨提供各种建议和帮助的Song Hyeon-Jong和Kim Hyon-Seok，能使得我不断思考的Kim Jin-Wook，为主办自动驾驶比赛而一起努力的Seng Ki-Jae，共度美好时光的开源机器人论坛（OROCA）的AuTURBO项目的成员们，谢谢大家。还要想为我不断牺牲的父母说：“我会一直孝敬你们”。同时也向陪伴我的弟弟和一直与我同在的Kim Ha表示感谢，最后我把所有荣耀献给创造我的上帝。

2017年7月

郑黎焜

近些年，媒体经常报道由于高端技术机器人和人工智能而将发生变化的社会、经济和文化的新闻。随着社会发展，有些人以乐观的态度期待着不断提高改善的生活，但是可能会更早到来的对于劳动市场的负面影响的展望会使得人们更加不安。如此，我们身边正在进行的机器人和人工智能的研究和发展，会在不久的将来会对我们产生深远的影响。所以我们更要关注机器人技术的发展，试着去了解并为未来做好准备。

为了应对快速发展的技术，开源代码正在利用集体智能来促进技术的发展和普及。现在，机器人技术可以通过开源代码，与多种多样的人共同工作和共同发展，复杂的算法也可以很容易地应用在个人机器人中。从而，我们可以防止因为技术被归属于特定群体而影响整个社会的情况，也可以一步一步地揭开机器人的神秘面貌，克服恐惧心理。

我的目标是通过技术的应用和变化给人们带来感动，使他们对这些技术更感兴趣。作为第一步，我试图通过开源代码社区学习各种机器人技术，并开放自己项目的代码，努力与各种各样的人进行协作。以后，我正计划着与机器人以外其他领域的人进行交流，互相分享经验和技术。通过这种方式推广技术，并为人们提供各种体验，使得人们轻松地适应未来社会的变化。

我参与了这本书的机械手臂部分的编著，比较易懂地整理了维基里ROS、Gazebo、MoveIt!的内容，同时花了大量时间去试图总结那些没有在维基解释的部分，努力将更多更准确的信息传达给读者。我想继续精益求精，成为与人们分享所学到的知识的人。

首先想要向表允哲博士表示感谢，直到这本书出版，他作为学校前辈、公司领导、老师，教给我很多知识的同时也给予我勇气和机会。还有对Sung Chang-Hyeon博士表示感谢，在百忙之中腾出时间编审原稿，并且诚恳地回答各种各样的问题。我还要感谢从早到晚一起共事的开源代码团队的同事们和总是面带微笑，充满热情的所有ROBOTIS员工。还要向研究生院指导教授Lee Jong-Ho教授表示感谢，在他的指导下，不仅·能学到工学知识和研究过程，还能培养真诚、耐心和责任感。再次感谢您。

最后，一直在旁边默默地帮助我们、热心地守护我们的爸爸和比任何人都更有好奇心和创造力的敬爱的母亲，虽然比我小两岁，但一直像朋友般陪伴着我的唯一的弟弟，我想对你们说谢谢你们，我爱你们。而且也要感谢在过去7年时间一直陪伴着我，有时像姐姐一样包容我，使我感受幸福，有时像妹妹一样撒娇，使我心动的Kim Go-En，谢谢你！

2017年7月

林泰勋

表允哲 (Pyo)

代表作者表允哲是ROBOTIS的研究员，他是开源团队的负责人，正在研究和开发基于开源项目的服务机器人的支持系统。机器人总是会抛给我们更多需要思考的问题，让我们更加细心地接近和观察我们的生活。从光云大学电子工程专业本科毕业之后在韩国科学技术研究院（KIST）工作，之后赴日本九州大学，以人工智能工程专业获得硕士及博士学位。喜欢分享技术，比如偶尔向机器人报社、无线航模、机器人月刊、ROBOCON MAGAZINE，等报纸和杂志投稿。喜欢和拥有机器人梦想的人们畅谈，在“为机器人工程的开放社区”及以开源机器人技术为宗旨的“开源机器人技术共享社区（WWW.OROCA.ORG）”进行公开讲座并进行公开项目。一向喜欢举办活动，因此期待通过机器人及ROS相关的演讲、讲座、研讨会、展会等活动中能遇见本书的读者。

赵汉哲 (Cho)

作者赵汉哲在ROBOTIS负责机器人控制器和固件的开发。曾在LG CNS担任ATM的固件开发，对于编程和机器人有浓厚的兴趣。在中学时期参观Micro Mouse比赛成了认识机器人的契机，至今喜爱学习和分享与机器人有关的技术。尤其是对于控制硬件的固件和FPGA深感魅力，制作与此相关的机器人作品。认为技术是在分享的时候能得到更大的发展，梦想着即便时间流逝，到了老年也能过着焊硬件、编代码的生活。

郑黎蝠 (Leon)

作者郑黎蝠是ROBOTIS的研究员，负责开发自动驾驶系统及舵机控制应用程序。认为人与人之间无法互补的空缺正是机器人存在的意义所在，并在机器研发过程中也努力实践这种理念。曾获早稻田大学电气工程与生物技术系学士学位和硕士学位。他曾一边为ROBOCON杂志寄稿，一边负责大规模机器人大赛R-BIZ CHALLENGE的自动驾驶比赛。目前，正在开展“开源机器人技术共享社区（WWW.ROCA.ORG）”中的自动驾驶机器人研究项目。

林泰勋 (Darby)

作者林泰勋是ROBOTIS的一名研究员，在开源团队中负责TurtleBot3和OpenManipulator的开发，此外还担负着本团队的颜值。相信创意需要丰富的经验和广博的知识，喜欢旅行、读书以及与各行各业的人们进行对话。给更多人提供独特的体验和平实的感动正是他开发机器人的目标。努力通过与电影、展会、媒体等各领域的专家合作共事来实现自己的梦想。

从2016年起，在“开源机器人技术共享社区”参与“LooKSo in Film”活动，担任“冷冷剧作家”及软件工程师。

开源软件及硬件

本书中用到的开源软件及硬件都公开在Github及Onshape的存储库，并时刻接收用户的反馈和改进意见，持续进行版本升级。下面是与本书中用到的开源软件及硬件相关的Github及Onshape的链接。

开源软件列表

- https://github.com/ROBOTIS-GIT/robotis_tools → 第3章
- https://github.com/ROBOTIS-GIT/ros_tutorials → 第4, 第7, 第13章
- <https://github.com/ROBOTIS-GIT/DynamixelSDK> → 第8, 第10章
- <https://github.com/ROBOTIS-GIT/dynamixel-workbench> → 第8, 第13章
- <https://github.com/ROBOTIS-GIT/dynamixel-workbench-msgs> → 第8, 第13章
- https://github.com/ROBOTIS-GIT/hls_lfcd_lds_driver → 第8, 第10, 第11章
- <https://github.com/ROBOTIS-GIT/OpenCR> → 第9, 第12章
- <https://github.com/ROBOTIS-GIT/turtlebot3> → 第10, 第11章
- https://github.com/ROBOTIS-GIT/turtlebot3_msgs → 第10, 第11章
- https://github.com/ROBOTIS-GIT/turtlebot3_simulations → 第10, 第11章
- https://github.com/ROBOTIS-GIT/turtlebot3_applications → 第10, 第11章
- https://github.com/ROBOTIS-GIT/turtlebot3_deliver → 第12章
- https://github.com/ROBOTIS-GIT/open_manipulator → 第13章

开源硬件列表

- OpenCR (第9章)
 - Board: <https://github.com/ROBOTIS-GIT/OpenCR-Hardware>
- TurtleBot3 (第10章、第11章、第12章、第13章)
 - Burger: <http://www.robotis.com/service/download.php?no=676>
 - Waffle: <http://www.robotis.com/service/download.php?no=677>
 - Waffle Pi: <http://www.robotis.com/service/download.php?no=678>
 - Friends OpenManipulator Chain: <http://www.robotis.com/service/download.php?no=679>
 - Friends Segway: <http://www.robotis.com/service/download.php?no=680>
 - Friends Conveyor: <http://www.robotis.com/service/download.php?no=681>
 - Friends Monster: <http://www.robotis.com/service/download.php?no=682>
 - Friends Tank: <http://www.robotis.com/service/download.php?no=683>
 - Friends Omni: <http://www.robotis.com/service/download.php?no=684>
 - Friends Mecanum: <http://www.robotis.com/service/download.php?no=685>
 - Friends Bike: <http://www.robotis.com/service/download.php?no=686>
 - Friends Road Train: <http://www.robotis.com/service/download.php?no=687>
 - Friends Real TurtleBot: <http://www.robotis.com/service/download.php?no=688>
 - Friends Carrier: <http://www.robotis.com/service/download.php?no=689>
- OpenManipulator (第10、第13章)
 - Chain: <http://www.robotis.com/service/download.php?no=690>
 - SCARA: <http://www.robotis.com/service/download.php?no=691>
 - Link: <http://www.robotis.com/service/download.php?no=692>

开源软件下载

本书中涉及到的所有代码均保存在Github的存储库中。下载源代码的方法有利用Git命令直接下载和通过浏览器下载压缩文件的方法。两种下载方法请参考如下说明。

① 用命令下载

为了在Linux里用git命令直接下载，需要先安装git。请打开终端，如下安装git。

```
$ sudo apt-get install git
```

然后用如下命令下载相应存储库中的源代码。（例：ros_tutorials功能包）

```
$ git clone https://github.com/ROBOTIS-GIT/ros_tutorials.git
```

② 从浏览器下载

用浏览器访问相应地址（https://github.com/ROBOTIS-GIT/ros_tutorials）可以访问Github的存储库。点击右上角的“Clone or download”→“Download ZIP”按键，就可以下载压缩文件。

开源资源

本书中用作教育器材的ROS官方机器人平台“TurtleBot3”的最新信息可以从如下的公开渠道获取。通过一步步实习（持续升级的）开源程序和丰富的TurtleBot3的应用案例，想必读者的ROS机器人编程实力会大有长进。

- TurtleBot 官网 <http://www.turtlebot.com>
- TurtleBot3 Wiki网站 <http://turtlebot3.robotis.com>
- TurtleBot3 视频 <https://www.youtube.com/c/ROBOTISOpenSourceTeam>

此外，用于搭建ROS嵌入式系统的“OpenCR”控制器、学习机械手臂操控的“OpenManipulator”，等本书中介绍的项目的相关内容也都已公开。不仅如此，在TurtleBot3及OpenManipulator用作舵机的“Dynamixel”，关于它的信息和它的必用软件“Dynamixel SDK”及“Dynamixel Workbench”也在如下链接公开最新信息和例子。

- OpenCR [http://emanual.robotis.com/] > [PARTS] > [Controller] > [OpenCR]
- OpenManipulator [http://emanual.robotis.com/] > [PLATFORM] > [OpenManipulator]

- Dynamixel SDK http://wiki.ros.org/dynamixel_sdk
[http://emanual.robotis.com/] > [SOFTWARE] > [DYNAMIXEL] > [Dynamixel SDK]
- Dynamixel Workbench http://wiki.ros.org/dynamixel_workbench
[http://emanual.robotis.com/] > [SOFTWARE] > [DYNAMIXEL] > [Dynamixel Workbench]

最后，读者也可以参考公开的ROS学习参考资料。这些资料由本书的各章要点和实战实例组成，因此如果与本书一起使用，想必会有助于团队学习和研讨会。

- 讲义资料 https://github.com/ROBOTIS-GIT/ros_seminar
- 参考资料 https://github.com/ROBOTIS-GIT/ros_book
- 源代码资料 https://github.com/ROBOTIS-GIT/ros_tutorials

相关社区及提问

- Robot Source Community http://www.robotsource.org/
- ROS Discourse https://discourse.ros.org/
- ROS Answers http://answers.ros.org/
- ROS Wiki http://wiki.ros.org/

专业术语

本书中的ROS相关术语尽量使用了国内常用的ROS术语，但有些术语则为了更清楚地表达，直接采用了英文原词。

中文术语表

英文术语	中文术语	英文术语	中文术语
Node	节点	Private Name	私有名称
Master	主节点	Relative Name	相对名称
Package	功能包	NodeHandle	节点句柄
Metapackage	元功能包	Timer	计时器
Dependent Package	依赖包	Transform	变换
Message	消息	Master PC	总机
Service	服务	Host PC	主机
Topic	话题	Build	构建
Service Server	服务服务器	Motion Planning	运动规划
Service Client	服务客户端	Odometry	测位
Parameter Server	参数服务器	Pose	姿态
Publisher	发布者	Play/Replay	回放
Subscriber	订阅者	Introspection	自检
Launch	启动	Icon	图标
Client Library	客户端库	Dead Reckoning	导航推测
Repositories	存储库	Base	基座
Namespace	命名空间	Link	连杆
Base Name	基本名称	Joint	关节
Global Name	全局名称	End Effector	末端执行器

许可说明

- 本书中使用的开源代码遵循分别指定的许可，著作权者或贡献者对因使用本软件而发生的直接或间接的损害、偶发或后果性损害赔偿，特殊或一般损害赔偿，对任何原因，责任，合同，义务，责任或侵权行为（包括疏忽）不承担责任。
- 本书中使用的开源代码会根据读者使用的时间，可能会有版本变化，因此在运行时可能不会有相异的结果。
- 本书中出现的公司名称和产品名称通常为各公司的注册商标，文中将省略TM，©，®标记。
- 如果您对本书内容有任何疑问，请联系出版社或上述社区。

第1章 / 机器人软件平台

1.1. 平台的组件	2
1.2. 机器人软件平台	3
1.3. 机器人软件平台的必要性	5
1.4. 机器人软件平台将带来的未来	6

第2章 / 机器人操作系统ROS

2.1. ROS简介	10
2.2. 元操作系统	10
2.3. ROS的目的	12
2.4. ROS的组件	13
2.5. ROS的生态系统	14
2.6. ROS的历史	15
2.7. ROS的版本	16
2.7.1. 版本规则	18
2.7.2. 版本周期	19
2.7.3. 选择版本	20

第3章 / 搭建ROS开发环境

3.1. 安装ROS	24
3.1.1. 常规安装	24
3.1.2. 简易安装	29
3.2. 搭建ROS开发环境	29
3.2.1. ROS配置	29
3.2.2. 集成开发环境 (IDE)	33
3.3. ROS操作测试	36

第4章 / ROS的重要概念

4.1. ROS术语	41
4.2. 消息通信	50
4.2.1. 话题 (topic)	51
4.2.2. 服务 (service)	52
4.2.3. 动作 (action)	52
4.2.4. 参数 (parameter)	54
4.2.5. 消息通信的过程	54
4.3. 消息	60
4.3.1. msg文件	63
4.3.2. srv文件	63
4.3.3. action文件	64
4.4. 名称 (name)	64
4.5. 坐标变换 (TF)	67
4.6. 客户端库	68
4.7. 异构设备间的通信	69
4.8. 文件系统	70
4.8.1. 文件组织结构	70
4.8.2. 安装目录	71
4.8.3. 工作目录	73
4.9. 构建系统	75
4.9.1. 创建功能包	75
4.9.2. 修改功能包配置文件 (package.xml)	76
4.9.3. 修改构建配置文件 (CMakeLists.txt)	79
4.9.4. 编写源代码	88
4.9.5. 构建功能包	90
4.9.6. 运行节点	90

第5章 / ROS命令

5.1. ROS命令概述	93
5.2. ROS shell命令	95
5.2.1. roscd: 移动ROS目录	95
5.2.2. roslibs: ROS文件列表	96
5.2.3. rosed: ROS编辑命令	96
5.3. ROS执行命令	97
5.3.1. roscore: 运行roscore	97
5.3.2. rosrun: 运行ROS节点	99
5.3.3. roslaunch: 运行多个ROS节点	99
5.3.4. rosclean: 检查及删除ROS日志	100
5.4. ROS信息命令	101
5.4.1. 运行节点	101
5.4.2. rosnode: ROS节点	102
5.4.3. rostopic: ROS话题	104
5.4.4. rosservice: ROS服务	108
5.4.5. rosparam: ROS参数	111
5.4.6. rosmsg: ROS消息信息	114
5.4.7. rossrv: ROS服务信息	116
5.4.8. rosbag: ROS日志信息	118
5.5. ROS catkin命令	122
5.6. ROS功能包命令	125

第6章 / ROS工具

6.1. 三维可视化工具 (RViz)	130
6.1.1. RViz安装与运行	133
6.1.2. RViz画面布局	134
6.1.3. RViz显示屏	136

6.2 ROS GUI开发工具 (rqt)	137
6.2.1. rqt安装与运行	138
6.2.2. rqt插件	139
6.2.3. rqt_image_view	142
6.2.4. rqt_graph	143
6.2.5. rqt_plot	145
6.2.6. rqt_bag	147
第7章 / ROS编程基础	
 7.1. ROS编程前须知事项	150
7.1.1. 标准单位	150
7.1.2. 坐标表现方式	151
7.1.3. 编程规则	152
 7.2. 发布者节点和订阅者节点的创建和运行	153
7.2.1. 创建功能包	153
7.2.2. 修改功能包配置文件	153
7.2.3. 修改构建配置文件 (CMakeLists.txt)	154
7.2.4. 创建消息文件	156
7.2.5. 创建发布者节点	157
7.2.6. 创建订阅者节点	158
7.2.7. 构建 (build) 节点	159
7.2.8. 运行发布者	160
7.2.9. 运行订阅者	161
7.2.10. 检查运行中的节点的通信状态	162
 7.3. 创建和运行服务服务器与客户端节点	163
7.3.1. 创建功能包	164
7.3.2. 修改功能包配置文件 (package.xml)	164
7.3.3. 修改构建配置文件 (CMakeLists.txt)	165

7.3.4. 创建服务文件	166
7.3.5. 创建服务服务器节点	167
7.3.6. 创建服务客户端节点	168
7.3.7. 构建节点	170
7.3.8. 运行服务服务器	170
7.3.9. 运行服务客户端	171
7.3.10. rosservice call命令的用法	172
7.3.11. GUI工具Service Caller的用法	172
7.4. 创建和运行动作服务器和客户端节点	174
7.4.1. 生成功能包	174
7.4.2. 修改功能包配置文件 (package.xml)	174
7.4.3. 修改构建配置文件 (CMakeLists.txt)	175
7.4.4. 创建动作文件	176
7.4.5. 创建动作服务节点	177
7.4.6. 创建客户端节点	180
7.4.7. 构建节点	181
7.4.8. 运行动作服务器	182
7.4.9. 运行动作客户端	184
7.5. 参数的用法	185
7.5.1. 利用参数创建节点	185
7.5.2. 设置参数	187
7.5.3. 读取参数	188
7.5.4. 构建节点和运行节点	188
7.5.5. 查看参数目录	188
7.5.6. 参数的用例	189
7.6. roslaunch的用法	190
7.6.1. roslaunch的应用	190
7.6.2. Launch标签	193

第8章 / 机器人、传感器和电机

8.1. 机器人功能包	196
8.2 传感器功能包	199
8.2.1. 传感器的类型	199
8.2.2. 传感器功能包的分类	200
8.3. 相机	201
8.3.1. USB摄像头相关功能包	202
8.3.2. USB摄像头测试	202
8.3.3. 查看图像信息	204
8.3.4. 远程传输图像	207
8.3.5. 相机校准	208
8.4. 深度相机 (Depth Camera)	214
8.4.1. Depth Camera的类型	215
8.4.2. Depth Camera测试	217
8.4.3. Point Cloud Data (点云数据) 的可视化	218
8.4.4. Point Cloud Data相关库	219
8.5. 激光距离传感器	219
8.5.1. LDS传感器距离测量原理	220
8.5.2. LDS测试	221
8.5.3. 可视化LDS的距离值	223
8.5.4. LDS的应用	224
8.6. 电机功能包	226
8.6.1. Dynamixel舵机	226
8.7. 已公开的功能包的用法	227
8.7.1. 搜索功能包	228
8.7.2. 安装依赖包	231
8.7.3. 安装功能包	232
8.7.4. 运行功能包	233

第9章 / 嵌入式系统

9.1. OpenCR	238
9.1.1. 特点	239
9.1.2. 控制板规格	241
9.1.3. 搭建开发环境	244
9.1.4. OpenCR例程	253
9.2. rosserial	258
9.2.1. rosserial server	259
9.2.2. rosserial client	259
9.2.3. rosserial协议	260
9.2.4. rosserial的约束条件	262
9.2.5. 安装rosserial	263
9.2.6. rosserial例程	265
9.3. TurtleBot3的固件	276
9.3.1. TurtleBot3 Burger固件	276
9.3.2. TurtleBot3 Waffle和Waffle Pi固件	277
9.3.3. TurtleBot3配置固件	278

第10章 / 移动机器人

10.1. ROS支持的机器人	283
10.2. TurtleBot3系列机器人	283
10.3. TurleBot3的硬件	284
10.4. TurtleBot3软件	287
10.5. TurtleBot3的开发环境	288
10.6. TurtleBot3远程控制	291
10.6.1. 遥控TurtleBot3	291
10.6.2. 可视化TurtleBot3	293

10.7. Turtlebot3话题	294
10.7.1. 订阅话题	296
10.7.2. 通过订阅话题控制机器人	296
10.7.3. 发布话题	297
10.7.4. 通过发布话题识别机器人状态	298
10.8. 使用RViz 仿真TurtleBot3	301
10.8.1. 仿真	301
10.8.2. 运行虚拟机器人	302
10.8.3. Odometry和TF	303
10.9. 利用Gazebo 仿真TurtleBot3	307
10.9.1. Gazebo仿真器	307
10.9.2. 启动虚拟机器人	309
10.9.3. 虚拟SLAM和导航	312

第11章 / SLAM和导航

11.1. 导航及其组成要素	317
11.1.1. 移动机器人的导航	317
11.1.2. 地图	318
11.1.3. 测量或估计机器人姿态的功能	318
11.1.4. 识别障碍物，如墙壁和物体	321
11.1.5. 计算最优路径和行驶功能	321
11.2. SLAM实习篇	321
11.2.1. 对于使用SLAM的机器人的硬件限制	322
11.2.2. SLAM的实验环境	323
11.2.3. 用于SLAM的ROS功能包	324
11.2.4. 运行SLAM	324
11.2.5. 利用预先准备好的bag文件运行的SLAM	327

11.3.SLAM应用篇	328
11.3.1.地图	328
11.3.2. SLAM所需的信息	330
11.3.3. SLAM的处理过程	331
11.3.4. 坐标变换 (TF)	333
11.3.5. turtlebot3_slam功能包	334
11.4.SLAM理论篇	337
11.4.1.SLAM	337
11.4.2. 多种位置估计 (localization) 方法论	338
11.5.导航实战篇	342
11.5.1.用于导航的ROS功能包	342
11.5.2.运行导航	342
11.6.导航应用程序	345
11.6.1.导航	345
11.6.2. 导航所需的信息	346
11.6.3. turtlebot3_navigation的各节点和话题状态	348
11.6.4. turtlebot3_navigation设置	349
11.6.5. 设置turtlebot3_navigation的详细参数	354
11.7.导航理论篇	361
11.7.1.Costmap	361
11.7.2.AMCL	363
11.7.3. Dynamic Window Approach(DWA)	365

第12章 / 服务机器人

12.1. 配送服务机器人	368
12.2. 配送服务机器人的结构	368
12.2.1. 系统结构	368
12.2.2. 系统设计	369
12.2.3. 服务核心节点	373
12.2.4. 服务主节点	383
12.2.5. 服务从节点	390
12.3. 用ROS Java进行Android平板PC编程	396

第13章 / 机械手臂

13.1. 机械手臂介绍	405
13.1.1. 机械手臂的结构和控制	405
13.1.2. 机械手臂和ROS	408
13.2. OpenManipulator建模和仿真	409
13.2.1. OpenManipulator	410
13.2.2. 机械手臂建模	410
13.2.3. Gazebo设置	428
13.3. MoveIt!	436
13.3.1. move_group	436
13.3.2. MoveIt! Setup Assistant	437
13.3.3. Gazebo仿真	452
13.4. 应用于实际平台	455
13.4.1. 准备和控制OpenManipulator	456
13.4.2. OpenManipulator与TurtleBot3 Waffle及Waffle Pi	460
索引	463

第1章

机器人软件平台

1.1. 平台的组件



图 1-1 PC 与智能手机

“这两个产品群的共同点是什么？”

IT产品群包括个人电脑（PC）和个人电话（PP）。顾名思义，这是每个人都有 的个人产品。如图1-2所示，该产品群的共同点在于它由可与各种硬件组合的硬件（Hardware）模块组成，具有管理这些硬件的操作系统（Operating System），在由操作系统提供的基于硬件抽象的软件开发环境中，存在提供各种服务的应用程序（Application），以及有大量的用户（User）使用它们。

硬件、操作系统、应用程序和用户等四个要素在IT行业中通常被称为平台的生态系统（Ecosystem）的四个要素，如图1-2所示。当具有这些要素，并且它们之间形成不可见的分工和协作时，我们说这个平台成功实现了大众化和个人化。

上述PC和PP起初也并没有满足这四个要素。早期，由特定公司开发的硬件的专用固件，为了运行特定硬件设备而搭载了软件，只能使用制造商提供的服务。如果您不明白，让我们回想起苹果iPhone出现之前，众多制造商的功能手机（Feature phone）吧。这些PC和PP的一个常见的成功来源是操作系统（Windows、Linux、Android、iOS等）的出现。操作系统的引入导致了硬件和软件接口的集成，从而实现了硬件模块化，使得制造商能够通过大规模生产和专业化开发的低成本，成功实现产品的高性能化发展，从而使个人化计算机和电话成为可能。

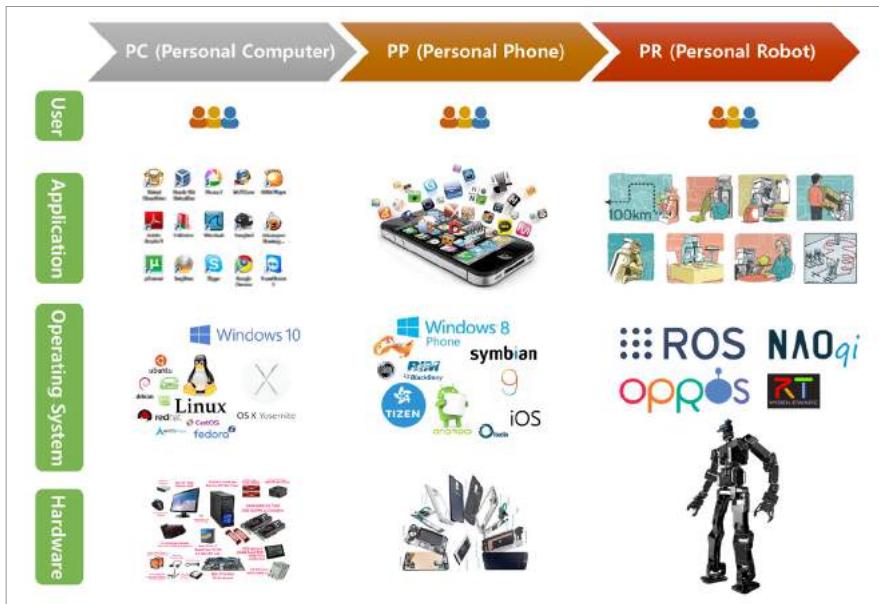


图 1-2 生态系统的4大组成要素和PC、PP、PR的历史的共同点

不仅如此，即使对硬件没有深刻的理解，也能通过操作系统提供的开发环境很容易地开发应用程序，在智能手机市场中出现了10年前没有的叫做APP开发者的新的职业群。如此的，硬件以操作系统为中心，开始向模块化发展，并且以操作系统提供的硬件抽象化为基础的应用程序开始独立分工，终于可以实现用户们真正想要的服务了。那么，与PC和PP一起被关注的PR（个人机器人）的代表性的服务机器人平台是什么？由于历史被认为是重复的，那么PR是否会像PC和PP的演变一样接近我们的生活？在下一节我们来更仔细地看一看。

1.2. 机器人软件平台

最近，“平台”在机器人领域也备受关注。平台分为软件平台和硬件平台。机器人软件平台不仅包括机器人应用中使用的硬件抽象、子设备控制，以及机器人工程中常用的传感、识别、实时自定位和绘图（SLAM）、导航（Navigation）和机械臂控制（Manipulation）等功能的实现，还包含功能包管理、开发环境所需的库、多种开发/调试工具。机器人硬件平台不仅包括移动机器人、无人机和人形硬件研究平台，还包括正在商业化的诸如SoftBank的Pepper和MIT Media Lab的Jibo等产品。

需要留意的是，这些硬件也是结合上述软件平台抽象出来的，即使没有硬件专业知识，也可以用软件平台来开发应用程序。正如您可以在不知道最新智能手机的硬件配置或细节的情况下开发APP一样。而且，与以往的机器人开发者担负从硬件设计到软件设计的整个工作不同，现在更多的非机器人领域的软件开发人员也可以参与机器人应用程序的开发。也就是说，借助软件平台，许多人可以参与机器人开发，而机器人硬件则是根据软件平台提出的界面进行设计的。

这一类的软件平台中具有代表性的有机器人操作系统ROS (Robot Operating System)¹、日本的开放式机器人技术中间件(OpenRTM)²、欧洲的实时控制为中心的OROCOS³和韩国的OProS⁴。名字各有不同，但这些机器人软件平台被开发的根本原因都是想让全世界的机器人研究人员们齐心协力解决机器人软件过于繁多和复杂而造成的问题。而其中使用人数最多的ROS正是本书要介绍的机器人软件平台。

例如，当实现机器人识别周围情况的功能的时候，硬件种类繁多的问题和要用在实际生活中的问题是会带来困难的部分。有些对人来说很简单的事情对机器人来说是很困难的问题，因为要实现机器人的传感、识别、绘制地图、动作规划等功能。因此学校的研究室或公司难以做到这些所有的功能。但如果全世界的相关领域的从事人员们都各自分享自己的专长部分，让其他研究团队使用的话情况就大不一样了。比如在众筹网Kickstarter和CES2015被人们关注的Robotbase⁵最近开发了Robotbase Personal Robot，并成功在众筹网投产。Robotbase公司集中于自己的核心技术脸部识别和物体识别，而移动机器人则采用支持ROS的Yujin Robot⁶的移动机器人基台，舵机使用ROBOTIS⁷的Dynamixel，并且障碍物识别、导航、电机驱动等均使用了ROS的公开功能包。另一个例子是 ROS-I (ROS Industrial Consortium)⁸，它是ROS工业组合。这个组合里有很多工业机器人领域的领军企业参与，一起解决着工业机器人领域中新出现的自动化、传感和协同机器人等一系列挑战。如此地，平台，尤其是在软件平台领域里通过使用共同的平台，用协作解决以往未能解决的问题，又带来了高效率。这展示了协作可能性的例子。

¹ <http://www.ros.org/>

² <http://openrtm.org>

³ <http://www.orocos.org/>

⁴ <http://oprofs.org/>

⁵ <https://www.kickstarter.com/projects/403524037/personal-robot>

⁶ <http://www.yujinrobot.com/>

⁷ <http://www.robotis.com>

⁸ <http://rosindustrial.org/>

1.3. 机器人软件平台的必要性

“为什么要使用机器人软件平台？”

我们为什么要学习ROS这个新的概念？这是在线下的ROS研讨会中经常听到的用户的提问。片面来讲是为了缩短开发时间。我们常说为了学习新的概念所花费的时间太可惜了，因为要修改已经建立好的系统和现有的程序，所以想维持现有的方式。但ROS不需要完全重新开发已有的系统和程序，而是通过加入一些标准化的代码就能对已有的非ROS程序进行ROS化的转化。并且很多通用的工具和软件都有提供，因此可以专注于自己感兴趣或想贡献的部分，这反而可以节省开发和维护所需的时间。让我们来了解一下这种方式的五种特点吧。

第一，程序的可重用性。专注于自己想要开发的部分，对剩下的功能可以下载相关功能包来使用。相反，也可以将自己开发出来的程序和其他人分享，让他们也可以使用。比如，美国的NASA为了控制宇宙空间站里使用的Robonaut2⁹机器人，除了使用自行开发的程序，还结合了可以在多种操作系统使用的ROS和具有实时控制、消息通信修复、可靠性等特点的OROCOS，得以在宇宙中执行任务。前面介绍的Robotbase公司的例子也是充分发挥可重用性的案例。

第二，是基于通信的程序。为了提供一种服务，很多时候在同一个框架里编写很多程序：从传感器或舵机的硬件驱动到传感、识别和动作等所有种类的程序。但为了重用机器人软件，根据每个处理器的用途将其分成更小的部分。根据平台的不同，我们将此称为组件化或节点化。必须由划分为最小执行单元的节点之间发送和接收数据，而平台具有关于该数据通信的所有一般信息。而且，这与最小的单位进程连接到网络的物联网（IoT）的概念一致，因此可以用作物联网平台。并且，被划分成最小执行单元的程序可以进行小单元的调试，这非常有助于找出错误。

第三，提供开发工具。ROS提供调试相关的工具-2维绘图和3维视觉化工具RViz，所以无需亲手准备机器人开发所需的开发工具，可以直接拿来使用。例如，在机器人开发中，可视化机器人的模型的情况比较多，通过遵守规定的信息格式，可以直接确认机器人的模型，并且还提供3D仿真器，因此易于扩展到仿真实验。另外，最近比较受人关注的点云

⁹ <https://robonaut.jsc.nasa.gov/R2/>

(point cloud) 形式也可以从英特尔的RealSense或微软的Kinect获得的3D距离信息转化过来。此外，实验中使用的数据可以被记录下来，因此需要的时候随时都可以重现实验当时的情况。像这样，ROS的一个重要特点是通过为机器人开发提供必要的软件工具，使开发的便利性达到最大化。

第四，活跃的开发者社区。至今比较封闭的机器人家学界和机器人业界都因为前述的功能而走向重视互相之间的合作的方向。其目的可能各自相异，但实际的合作正在通过这种软件平台发生着。其核心是开源软件平台的社区。例如，以ROS为例，到2017年自愿开发和共享的功能包数量超过了5,000个，而解释如何使用它的wiki页面超过1.7万个，这些都由用户个别参与。而在社区中非常重要的问题和答案已有超过24,000个，用户们通过这些建立着互惠互利的开发者社区。讨论超越了单纯的对于用法的议题，人们在寻找机器人工程软件的必要因素，并摸索出规则。

第五，生态系统的形成。前面提到的智能手机平台革命是由Android和iOS等软件平台创造的生态系统造成的。这一趋势在机器人领域延续着。起初，各种硬件技术泛滥，却没有能整合它们的操作系统。在这种情况下，如上所述，各种软件平台已经出现，最受瞩目的ROS现在已经开始构建生态系统。这个正在形成的生态系统里，机器人硬件领域的开发者、ROS开发运营团队、应用软件开发者以及用户也能像机器人公司和传感器公司一样从中受益。起步虽然微不足道，但考虑到逐渐增多的用户数量和机器人公司，以及急剧增加的相关工具和库，我期待在不久的将来将会形成一个圆满的生态系统。

1.4. 机器人软件平台将带来的未来

在机器人领域也发生着与前述的智能手机的案例类似的故事。当然，与智能手机操作系统相比，还有很多需要开发的工具处于正在发展的阶段，但机器人软件平台就像春秋战国时代一样处于百家争鸣的时期。其中活动比较显著的机器人软件平台和相应的群体如下。

- MSRDS¹⁰: Microsoft Robotics Developer Studio, 美国 Microsoft
- ERSP¹¹: Evolution Robotics Software Platform, 欧洲 Evolution Robotics

¹⁰ <https://www.microsoft.com/robotics/>

¹¹ https://en.wikipedia.org/wiki/Evolution_Robotics

- ROS: Robot Operating System, 美国 Open Robotics¹²
- OpenRTM: 日本工业技术综合研究所 (AIST)
- OROCOS: 欧洲
- OPRoS: 韩国 ETRI, KIST, KITECH, 江原大学
- NAOqi OS¹³: 日本软银和法国阿尔德巴兰 (Aldebaran)

除此之外, Player、YARP、MARIE、URBI、CARMEN、Orca、MOOS等也属于这个范畴。



图 1-3 多种机器人软件平台

虽然有各种各样的机器人软件平台,但是很难预测哪一个更好。这是因为它们分别提供了方便的组件添加、通信、可视化、仿真器和实时性等各自的独特功能。但是就像今天的个人电脑的操作系统一样,机器人软件的平台也将很快被压缩,具有类似的功能。我们并不是自己制作软件平台本身,所以最好把精力集中在开发通用机器人软件平台上运行的应用程序。

我收到了很多“现今的机器人软件平台中,你最喜欢哪个?”的问题。对这个问题,我的回答是:“我们从此停止消耗性的足球场建设吧!”,“让我们梦想成为球场上的伟大的球员吧!”这可以比作安卓系统。我们在既没有参与开发,也没有占据主导权的安卓生态系统中活跃为掌握硬软件的最优秀的选手,又给经济发展带来巨大的推动力一样,在机器人软件平台提供的市场中也能成为最棒的球员。

那么在现有的机器人软件平台当中,掌握哪一个比较好呢?

¹² <https://www.openrobotics.org/>

¹³ http://doc.aldebaran.com/2-1/index_dev_guide.html

对于这种提问，我认为Open Robotics在开发中的ROS是最佳的答案。尤其是如果考虑社区的活跃程度、丰富的库、扩展性和开发便利性的话ROS首当其冲。另外，供读者参考，Open Source Robotics Foundation在2017年5月改名为Open Robotics.

相比于其他任何机器人软件平台的社区，全世界的ROS社区是活动最活跃的社区。因此在使用中如果遇到疑问，可以很容易地搜索到相关信息。ROS不仅是由Open Robotics单独来开发，而且学界的科研人员、工业一线的开发人员和兴趣爱好者（Hobbyist）也都参与开发，他们在对于开发中遇到的问题也是积极地在社区里寻找答案，因此通过社区可以很方便地查到信息。不仅如此，除了机器人专业人士以外，网络、计算机科学和计算机视觉领域的人士也大有人在，因此ROS是更加让我期待的机器人软件平台。

利用机器人软件平台，即使是由不同的硬件构成的机器人，只要实现了基本功能，那么不用知道硬件专业知识，也能制作应用程序。这就像无需知道最新款智能手机的硬件和详细参数，也能开发APP。

与以往的机器人开发者担负从硬件设计到软件设计的整个工作不同，现在更多的软件开发者可以参与机器人应用产品的开发了。也就是说，多亏软件平台，许多人可以参与机器人开发，而机器人硬件则是根据软件平台提出的界面进行设计的。这形成了机器人技术快速发展的契机。

第2章

机器人操作系统ROS

2.1. ROS简介

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

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



在ROS维基中将ROS定义为“ROS是一个开放源代码的机器人元操作系统。它提供了我们对操作系统期望的服务，包括硬件抽象、低级设备控制、常用功能的实现、进程之间的消息传递以及功能包管理。它还提供了用于在多台计算机之间获取、构建、编写和运行代码的工具和库。”

换句话说，ROS包括一个类似于操作系统的硬件抽象，但它不是一个传统的操作系统，它具有可用于异构硬件的特性。此外，它是一个机器人软件平台，提供了专门为机器人开发应用程序的各种开发环境。

2.2. 元操作系统

通用计算机操作系统的种类有Windows（XP/7/8/10）、Linux（Linux Mint/Ubuntu/Fedora/Gentoo）、Mac（OS X Mavericks/Yosemite/El Capitan）等。至于智能手机，也有Android、iOS、Symbian、RiMO和Tizen等多种操作系统。

那么ROS是一种为机器人设计的新的操作系统吗？

ROS是Robot Operating System的缩写，因此会认为是一种操作系统。尤其是那些对ROS不熟悉的人会认为ROS和上面提到的操作系统一样。当我第一次遇到它时，我也认为ROS是一个新的机器人操作系统。

然而更确切地说，ROS是一个元操作系统（Meta-Operating System）¹。元操作系统不是一个明确定义的术语，而是一个利用应用程序和分布式计算资源之间的虚拟化层来运用分布式计算资源来执行调度、加载、监视、错误处理等任务的系统。

ROS不是传统的操作系统，如Windows、Linux和Android，反而是利用现有的操作系统。使用ROS前需要先安装诸如Ubuntu的Linux发行版操作系统，之后再安装ROS，以使用进程管理系统、文件系统、用户界面、程序实用程序（编译器、线程模型等）。此外，它还以库的形式提供了机器人应用程序所需的多数不同类型的硬件之间的数据传输/接收、调度和错误处理等功能。这个概念也被称为中间件（Middleware）或软件框架（Software framework）。

ROS开发、管理和提供基于元操作系统的各种用途的应用功能包，并拥有一个负责分享用户所开发的功能包的生态系统（Ecosystem）。如图2-1所示。ROS是在使用现有的传统操作系统的同时，通过使用硬件抽象概念来控制机器人应用程序所必需的机器人和传感器，同时也是开发用户的机器人应用程序的支持系统。



图 2-1 作为元操作系统的ROS

另外，如图2-2所示，ROS数据通信可以在一个操作系统中进行，但也适用于使用多种硬件的机器人开发，因为可以在不同的操作系统、硬件和程序之间交换数据。这将在下面的章节中详细讨论。

¹ <http://wiki.ros.org/ROS/Introduction>

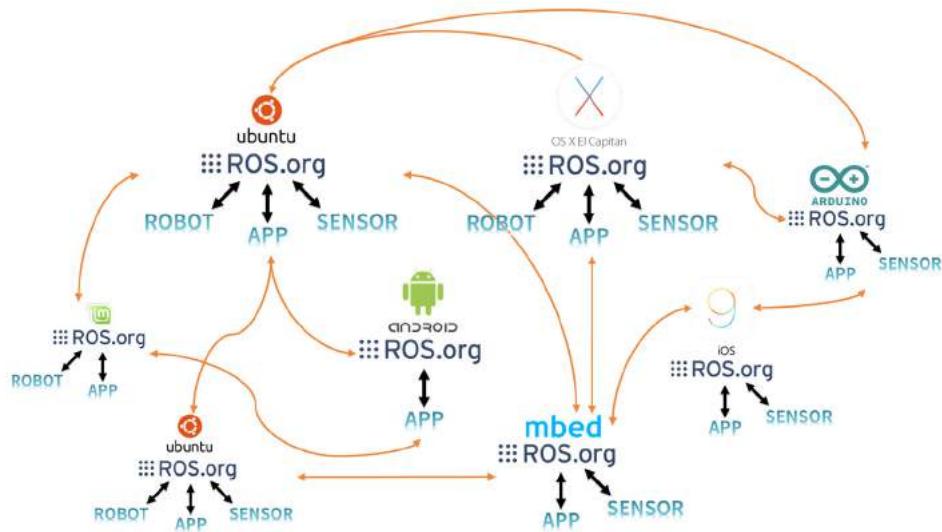


图 2-2 不同的操作系统或软件环境之间的ROS通信

2.3. ROS的目的

在我进行多年的ROS讲座的期间收到最多的提问是将ROS与其他机器人软件平台(OpenRTM、OPRoS、播放器、YARP、Orocos、CARMEN、Orca、MOOS和Microsoft Robotics Studio)进行比较。虽然可以与这些平台进行简单的比较，但是它们的比较没有什么意义，因为它们的目的各不相同。作为ROS的使用者，我觉得ROS的目标是“建立一个在全球范围内协作开发机器人软件的环境！”ROS致力于将机器人研究和开发中的代码重用做到最大化，而不是做所谓的机器人软件平台、中间件和框架。为了支持这个，ROS具有以下特征。

- 分布式进程：它以可执行进程的最小单位（节点，Node）的形式进行编程，每个进程独立运行，并有机地收发数据。
- 功能包单位管理：由于它以功能包的形式管理着多个具有相同目的的进程，所以开发和使用起来很容易，并且很容易共享、修改和重新发布。
- 公共存储库：每个功能包都将其功能包公开给开发人员首选的公共存储库（例如GitHub），并标识许可证。
- API类型：使用ROS开发程序时，ROS被设计为可以简单地通过调用API将其加载到其使用的代码中。在每章中介绍的源代码中，您会发现ROS编程与C++和Python程序没有区别。

- 支持多种编程语言：ROS程序提供客户端库（Client Library）²以支持各种语言。它可用于如JAVA、C#、Lua和Ruby等语言，也可以用于机器人中常用的编程语言，如Python、C++和Lisp。换句话说，您可以使用熟悉的语言开发ROS程序。

这种支持使得在全球范围内开发机器人软件的合作成为可能，并且ROS的终极目的-机器人研究和开发过程中的代码重用-变得越来越普遍。

2.4. ROS的组件

如图2-3所示，ROS由支持多种编程语言的客户端库、用于控制硬件的硬件接口、数据通信通道、帮助编写各种机器人应用程序的机器人应用框架（Robotics Application Framework）、基于此框架的服务应用程序-Robotics Application、在虚拟空间中控制机器人的仿真（Simulation）工具和软件开发工具（Software Development Tool）等组成。



图 2-3 ROS的组件³

² <http://wiki.ros.org/Client%20Libraries>

³ <http://wiki.ros.org/APIs>

2.5. ROS的生态系统

随着在智能手机市场出现安卓、iOS、Symbian、RiMO和Bada等多种操作系统，我们经常听到生态系统（Ecosystem）这个词。这是指将硬件制造商、操作系统公司、应用程序（APP）开发人员以及使用智能手机的用户连接起来的结构。

例如，当智能手机制造商基于操作系统的给定硬件接口生产设备时，每个操作系统公司会将它以库的形式提供。软件开发人员则无需了解硬件也可以轻松开发应用程序。而将产品投放到市场，让用户易于购买和使用。这一切统称为生态系统。

这种生态系统并不是在手机市场首次出现的。个人计算机领域也曾有各种各样的硬件制造商，而将他们结合在一起的微软的Windows操作系统和免费的Linux是典型的例子。也许这个过程就和自然界的生态系统是类似的发展过程。

机器人领域也正经历着同样的过程。起初，各种硬件技术泛滥，却没有能整合它们的操作系统。在这种情况下，如上所述，各种软件平台已经出现，最受瞩目的ROS现在已经开始构建生态系统。起步虽然微不足道，但考虑到逐渐增多的用户数量和机器人公司，以及急剧增加的相关工具和库，我期待在不久的将来将会形成一个圆满的生态系统。另外，我希望机器人硬件领域的开发者、ROS开发运营团队、应用软件开发者以及用户也能像机器人公司和传感器公司一样从中受益。

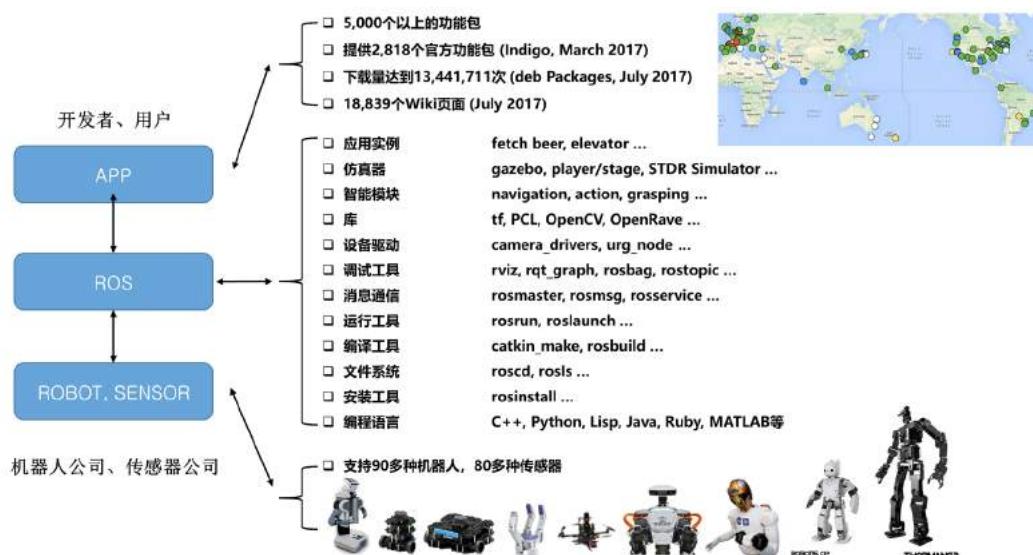


图 2-4 ROS生态系统

图 2-4是根据在ROSCon 2016⁴发布的ROS官方统计⁵和2017年度的ROS维基⁶资料整理的ROS现状。也许很多人会认为现在还势单力薄，但我认为在机器人领域没有如此活跃的机器人软件平台。我非常期待将来的发展。

2.6. ROS的历史

让我们更深入地了解ROS吧。ROS始于2007年5月份由摩根·奎格利（Morgan Quigley）⁷ 博士为了美国的斯坦福大学人工智能研究所（AI LAB）进行的STAIR（STanford AI Robot）⁸ 项目开发的Switchyard⁹ 系统。



摩根·奎格利 博士

摩根·奎格利（Morgan Quigley）博士是现在负责开发和管理ROS的Open Robotics【原OSRF (Open Source Robotics Foundation)】的创办人，也是软件开发负责人。Switchyard是当时为了用于AI研究室项目的人工智能机器人的开发而设计的程序，也是ROS的前身。此外，从2000年开始开发，对ROS的网络程序产生巨大影响的Player/Stage项目（Player网络服务器和2D stage仿真器，还影响至ROS的3D仿真器Gazebo的开发）的开发者Brian Gerkey (<http://brian.gerkey.org/>) 是Open Robotics的CEO及共同创办人。因此，ROS在2007年由Willow Garage公司获得ROS这个名字之前，受到了2000年的Player/Stage和2007年的Switchyard等项目的影响。

2007年11月开始，由美国的机器人专门公司Willow Garage承接开发ROS。Willow Garage是个人机器人（Personal robotics）及服务机器人领域中非常有名的公司。它以开发和支持我们熟知的视觉处理开源代码OpenCV和Kinect等在三维设备广泛使用的点云库（PCL， Point Cloud Library）)而著名。

⁴ <http://roscon.ros.org/2016/>

⁵ <http://wiki.ros.org/Metrics>

⁶ <http://wiki.ros.org/>

⁷ <https://www.osrfoundation.org/team/morgan-quigley/>

⁸ <http://stair.stanford.edu/>

⁹ <http://www.willowgarage.com/pages/software/ros-platform>

这家Willow Garage从2007年11月开始着手ROS的开发，在2010年1月22日向全世界发布了ROS 1.0。我们熟知的ROS Box Turtle版是2010年3月1日首次发布的。此后，和Ubuntu和安卓一样，每个版本都以C Turtle、Diamondback等按字母顺序起名。

ROS基于BSD许可证（BSD 3-Clause License）¹⁰ 及Apache License 2.0)¹¹，因此任何人可以修改、重用和重新发布。与此同时，ROS持续提供大量最新版本的软件，因此在教育及学术领域认识的参与程度非常高，一开始通过机器人相关学术会议广为传播。有面向开发者和用户的ROSDay和ROSCon¹² 学术会议，还有叫做ROS Meetup¹³的多种社区群。不仅如此，可以应用ROS的机器人平台的开发也在快速跟进。例如以Personal Robot为含义的PR2¹⁴和TurtleBot¹⁵ 机器人，有许多应用程序通过它们喷涌而出，这更加巩固了ROS作为机器人操作系统的地位。



图 2-5 OSRF 的标志 (<http://osrfoundation.org/>)



图 2-6 Open Robotics 的标志 (<https://www.openrobotics.org/>)

2.7. ROS的版本

Willow Garage承接了从2007的斯坦福大学人工智能研究所以Switchyard开始的机器人软件框架研究之后以ROS（Robot Operating System）的名称延续了开发工作。其第六次发布版ROS Groovy Galapagos是Willow Garage的最后的版本。Willow Garage在2013年进入商业服务机器人领域之后遇到诸多困难并分解为多个创业公司，此时ROS被转让给开源机器人工程基金会（OSRF，Open Source Robotics Foundation）。

¹⁰ <https://opensource.org/licenses/BSD-3-Clause>

¹¹ <https://www.apache.org/licenses/LICENSE-2.0>

¹² <http://roscon.ros.org>

¹³ <http://wiki.ros.org/Events>

¹⁴ <http://www.willowgarage.com/pages/pr2/overview>

¹⁵ <http://www.turtlebot.com/>

之后OSRF继续发布了4个新版本。从2017年5月开始OSRF更名为Open Robotics，至今开发、运营和管理ROS。在最近的2017年5月23日发布了ROS的第十一版ROS Lunar Loggerhead。ROS的每个版本的名称的首字母是按照英文字母的顺序来制定的，并将乌龟（Turtle）作为图标（图2-7）。

ROS的发布及学术会议

- 2017.12.08 - 发布ROS 2.0
- 2017.09.21 - 举办ROSCon2017（加拿大）
- 2017.05.23 - 发布Lunar Loggerhead
- 2017.05.16 - 从OSRF更名为Open Robotics
- 2016.10.08 - 举办ROSCon2016（韩国）
- 2016.05.23 - 发布Kinetic Kame
- 2015.10.03 - 举办ROSCon2015（德国）
- 2015.05.23 - 发布Jade Turtle
- 2014.09.12 - 发布ROSCon2014（美国）
- 2014.07.22 - 发布Indigo Igloo
- 2014.06.06 - 举办ROS Kong 2014（香港）
- 2013.09.04 - 发布Hydro Medusa
- 2013.05.11 - 举办ROSCon2013（德国）
- 2013.02.11 - Open Source Robotics Foundation担任开发和管理
- 2012.12.31 - 发布Groovy Galapagos
- 2012.05.19 - 举办ROSCon2012（美国）
- 2012.04.23 - 发布Fuerte
- 2011.08.30 - 发布Electric Emys
- 2011.03.02 - 发布Diamondback
- 2010.08.02 - 发布C Turtle
- 2010.03.02 - 发布Box Turtle

- 2010.01.22 - 开发ROS 1.0
- 2007.11.01 - Willow Garage起名“ROS”，并开始开发
- 2007.05.01 - Switchyard系统，Morgan Quigley，斯坦福大学AI实验室，斯坦福大学
- 2000 - Player/Stage项目，Brian Gerkey，Richard Vaughan，Andrew Howard，南加州大学

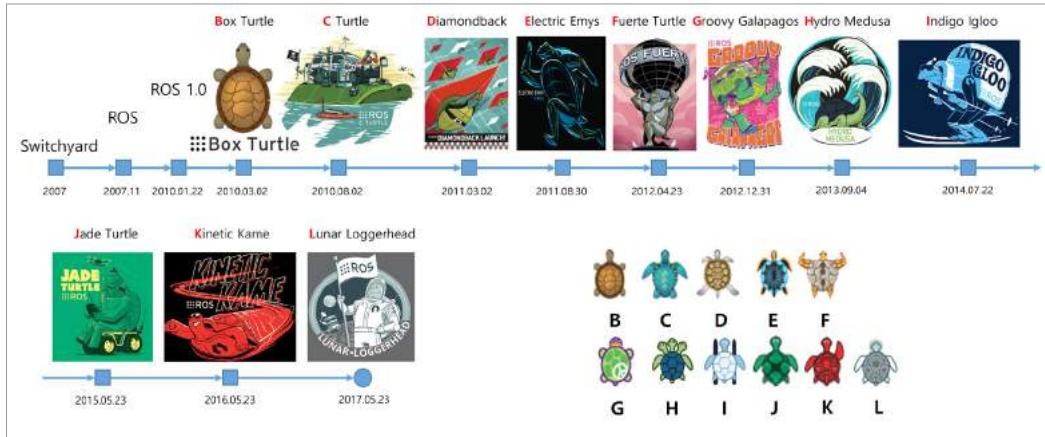


图 2-7 ROS的版本 (<http://wiki.ros.org/>)

2.7.1. 版本规则

到目前为止ROS发布了ROS 1.0、Box Turtle、C Turtle、Diamondback、Electric Emys、Fuerte、Groovy Galapagos、Hydro Medusa、Indigo Igloo、Jade Turtle、Kinetic Kame和Lunar Loggerhead等版本。

ROS除了1.0版本以外，都像Ubuntu和安卓一样，将版本名的首字母按英文字母顺序来安排。比如，Kinetic Kame是字母表的K版本，是第11个版本，也是第10个正式发布版。

此外还有一个规则。每个版本都有一个海报形式的插图和一个乌龟图标，如图2-8所示。这个乌龟图标也被用于ROS官方仿真教程turtlesim。使用海龟作为ROS的象征很大程度上是来自于20世纪60年代MIT人工智能研究所¹⁶的教育节目标志（Logo）¹⁷。在50

¹⁶ http://el.media.mit.edu/logo-foundation/what_is_logo/index.html

¹⁷ [https://en.wikipedia.org/wiki/Logo_\(programming_language\)](https://en.wikipedia.org/wiki/Logo_(programming_language))

多年前的1969年，乌龟（Turtle）机器人是使用一个叫Logo的编程语言开发的，这个机器人会根据计算机下达的命令，在地板上实际移动并可以画图。依照它，开发ROS的时候开发了叫做turtlesim的虚拟的例子程序，而这也是后来将ROS机器人成为TurtleBot的由来¹⁸。



图 2-8 各版本的乌龟图标

2.7.2. 版本周期

ROS的版本周期和ROS正式支持的操作系统Ubuntu的新版本发布周期一样，是每年发布两次（4月和10月）。但2013年很多用户由于频繁的升级，提议新的版本周期。因此在2013年接纳了用户们的建议，决定从Hydro Medusa版本开始，每年发布一次正式版本，时间点则在新的Ubuntu xx.04版本发布一个月之后的5月份。而恰好每年的5月23日是世界乌龟日（World Turtle Day）¹⁹，所以现在每年的这个有象征意义的一天会发布新的ROS版本。

ROS的支持期间根据版本而不同，一般是发布后提供两年的支持。而针对每两年发布的Ubuntu长期支持版（Long Term Support）²⁰发布的ROS版本提供与LTS一样的5年的支持期间。比如，支持2016年的Ubuntu 16.04 LTS的ROS Kinetic Kame具有直到2021年5月的5年的支持期间。非LTS版本保持最新版本的Linux核，仅仅是进行了次要的升级以及基本的维护。因此很多ROS用户们使用偶数年度发布的LTS版本的ROS。目前为止的ROS的版本升级²¹如图2-9。

¹⁸ <http://spectrum.ieee.org/automaton/robotics/diy/interview-turtlebot-inventors-tell-us-everything-about-the-robot>

¹⁹ <https://www.worldturtleday.org/>

²⁰ <https://wiki.ubuntu.com/LTS>

²¹ <http://wiki.ros.org/Distributions>

ROS发布版	发布日期	海报	乌龟图标	停止支持日期
Lunar Loggerhead	2017.05.23			2019.05
Kinetic Kame (推荐)	2016.05.23			2021.04 (Xenial EOL)
Jade Turtle	2015.05.23			2017.05
Indigo Igloo	2014.07.22			2019.04 (Trusty EOL)

图 2-9 主要ROS版本的发布及支持期限

2.7.3. 选择版本

由于ROS是元操作系统，需要选择基本的操作系统。ROS支持Ubuntu、Linux Mint、Debian、OS X、Fedora、Gentoo、OpenSUSE、ArcLinux和Windows等多种操作系统，但用的最多的操作系统是Ubuntu和Linux Mint。开发部门也是针对Ubuntu LTS版本进行测试并发布。由于这种原因我推荐Ubuntu LTS或与Linux Mint版本匹配的ROS版本。

对于每个ROS版本的Ubuntu移植信息，请访问相关信息页面²²并选择要使用的ROS版本。在这个文件中，您可以看到当前选择的ROS版本针对Linux各发行版进行的现有功能包（源代码）的迁移工作的进度（已结束或在进行中）。

您可能对Ubuntu的发布版版本比较陌生。从下面的列表可以看出14.04 Trusty是Ubuntu T版本，之后依次是U、V、W和X版本。您需要在这里进行比较并找出稳定的版本。ROS的最新版本会有很多功能包显示处于处理中的状态，如果不是重要的功能包，那

²² http://repositories.ros.org/status_page/ros_kinetic_default.html

么选择最新版本也无碍。但如果自己一直以来使用的功能包还没有完成移植，那需要再等一等。

- Ubuntu 18.04 Bionic Beaver (LTS)
- Ubuntu 17.10 Artful Aardvark
- Ubuntu 17.04 Zesty Zapus
- Ubuntu 16.10 Yakkety Yak
- **Ubuntu 16.04 Xenial Xerus (LTS)**
- Ubuntu 15.10 Wily Werewolf
- Ubuntu 15.04 Vivid Vervet
- Ubuntu 14.10 Utopic Unicorn
- **Ubuntu 14.04 Trusty Tahr (LTS)**
- Ubuntu 13.10 Saucy Salamander
- Ubuntu 13.04 Raring Ringtail
- Ubuntu 12.10 Quantal Quetzal
- **Ubuntu 12.04 Precise Pangolin (LTS)**

在2018年发布的新的Linux版本和ROS LTS版本达到稳定的2019年之前我想推荐的组合如下：

- 操作系统：Ubuntu 16.04 Xenial Xerus²³ (LTS) 或 Linux Mint 18.x
- ROS版本：ROS Kinetic Kame²⁴

²³ <http://releases.ubuntu.com/16.04/>

²⁴ <http://wiki.ros.org/kinetic>

第3章

搭建ROS开发环境

ROS支持多种操作系统，但正式支持的只有Ubuntu，而对其它版本仅提供安装方法。因此本书只说明Ubuntu及与Ubuntu兼容的Linux Mint。

本书使用的ROS应用程序开发环境如下。

- 硬件：使用INTEL及AMD芯片的台式机及笔记本电脑
- 操作系统：Ubuntu 16.04.x Xenial Xerus或Linux Mint 18.x
- ROS：Kinetic Kame

如果用户的计算机上安装了不同版本的Ubuntu，请确认官网。如果用户的操作系统是OS X¹或Windows²，可以查看相应wiki³的安装说明。而使用ARM CPU而不是INTEL或AMD CPU的单板计算机（SBC，Single Board Computer）并不单独说明ROS的安装，假如是使用Ubuntu或Linux Mint，则与以下说明相同。

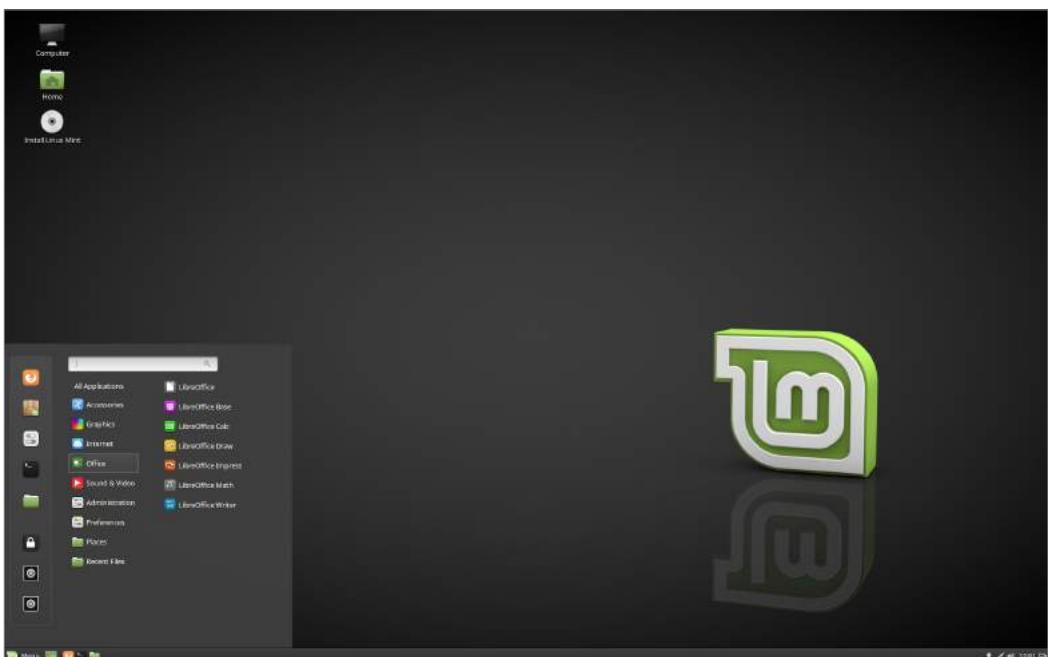


图 3-1 Linux Mint的桌面

1 <http://wiki.ros.org/kinetic/Installation/OSX/Homebrew/Source>

2 <http://wiki.ros.org/hydro/Installation/Windows>

3 <http://wiki.ros.org/kinetic/Installation>

3.1. 安装ROS

3.1.1. 常规安装

让我们安装ROS Kinetic吧。按照常规安装的说明也不难安装ROS Kinetic，但如果想更简单地进行安装，可以使用笔者在3.1.2给出的简易安装脚本。

设置网络时间协议（NTP， Network Time Protocol）

在ROS官方安装项目中虽然没有包括NTP，但为了缩小PC间通信中的ROS Time的误差，下面我们设置NTP⁴。设置方法是安装chrony之后用ntpdate命令指定ntp服务器即可。这样一来会表示服务器和当前计算机之间的时间误差，进而会调到服务器的时间。这就是通过给不同的PC指定相同的NTP服务器，将时间误差缩短到最小的方法。

```
$ sudo apt-get install -y chrony ntpdate  
$ sudo ntpdate -q ntp.ubuntu.com
```

添加代码列表

在ros-latest.list添加ROS版本库。打开新的终端窗口，输入如下命令。

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

如果用户正在使用Linux Mint版本18.x，请使用以下命令。上面提到的代码的\$(lsb_release -sc)会获得Linux发行版信息的代号，而Linux Mint 18.x使用了Ubuntu的xenial代号，所以可以添加与Ubuntu相同的源代码列表。

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

⁴ https://en.wikipedia.org/wiki/Network_Time_Protocol

设置公钥 (Key)

为了从ROS存储库下载功能包，下面添加公钥。作为参考，以下公钥可以根据服务器的操作发生变更，请参考官方wiki页面⁵。

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A389552  
3BAEEDB01FA116
```

更新软件包索引

现在用户已经将ROS版本库地址放在代码列表中，我们建议在安装ROS之前刷新软件包列表的索引并升级所有当前安装的Ubuntu软件包，但这不是必选项。

```
$ sudo apt-get update && sudo apt-get upgrade -y
```

安装ROS Kinetic Kame

使用以下命令安装台式机的ROS功能包。这包括ROS、rqt、RViz、机器人相关的库、仿真和导航等等。

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

上面的安装只包含基本的rqt，但是笔者将安装所有额外的rqt相关的功能包。用下面的命令安装所有rqt相关的功能包，可以很方便地使用各种rqt插件。

```
$ sudo apt-get install ros-kinetic-rqt*
```

⁵ <http://wiki.ros.org/kinetic/Installation/Ubuntu>

ROS功能包二进制文件

如果想安装ROS功能包，可以用下面的apt-cache命令搜索以ros-kinetic开头的所有功能包。利用这个命令可以搜索到大约1600多个功能包。

```
apt-cache search ros-kinetic
```

如果想个别安装功能包，请使用如下命令。

```
sudo apt-get install ros-kinetic-[功能包名称]
```

此外也可以使用GUI工具sysnaptic package manager。

APT (Advanced Packaging Tool) ⁶

apt-get、apt-key、apt-cache等命令中的apt (Advanced Packaging Tool) 是包括Ubuntu、Linux Mint的Debian系列的Linux中广泛使用的软件包管理命令。

http://en.wikipedia.org/wiki/Advanced_Packaging_Tool

删除旧版本的ROS及轮番使用不同版本的ROS。

sudo apt-get purge ros-indigo-* 命令可以实现设置和删除文件。在和现有版本一起使用时导入ROS配置文件（加载到‘~/.bashrc’）的命令中，将如下命令中的ROS版本部分

```
source /opt/ros/kinetic/setup.bash
```

替换为kinetic或indigo即可。

初始化rosdep

在使用ROS之前，必须初始化rosdep。rosdep是一个通过在使用或编译ros的核心组件时轻松安装依赖包来增强用户便利的功能。

```
$ sudo rosdep init  
$ rosdep update
```

⁶ https://en.wikipedia.org/wiki/Advanced_Packaging_Tool

安装rosinstall

这是安装ROS的各种功能包的程序。它是被频繁使用的有用的工具，因此务必要安装它。

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

加载环境设置文件

下面加载环境设置文件。里面定义着ROS_ROOT和ROS_PACKAGE_PATH等环境变量。

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

创建并初始化工作目录

ROS使用一个名为catkin的ROS专用构建系统。为了使用它，用户需要创建并初始化catkin工作目录，如下所示。除非用户创建工作目录，否则此设置只需设置一次。

```
$ mkdir -p ~/catkin_ws/src  
$ cd ~/catkin_ws/src  
$ catkin_init_workspace
```

如果用户已经创建了一个catkin工作目录，下面我们来进行构建。目前，只有src目录和CMakeLists.txt文件在catkin工作目录中，但让我们尝试使用catkin_make命令来构建吧。

```
$ cd ~/catkin_ws/  
$ catkin_make
```

当用户构建没有问题时，运行ls命令。除了自己创建的src目录之外，还出现了一个新的build和devel目录。catkin的构建系统的相关文件保存在build目录中，构建后的可执行文件保存在devel目录中。

```
$ ls  
build
```

```
devel  
src
```

最后，我们加载与catkin构建系统相关的环境文件。

```
$ source ~/catkin_ws/devel/setup.bash
```

测试安装结果

所有ROS安装已完成。要测试它是否正确安装，请关闭所有终端窗口并运行一个新的终端窗口。现在通过输入以下命令来运行roscore。

```
$ roscore
```

如果运行之后像下面没有出现错误，则说明成功安装。退出是[Ctrl+c]。

```
... logging to /home/pyo/.ros/log/9e24585a-60c8-11e7-b113-08d40c80c500/roslaunch-pyo-5207.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:38345/
ros_comm version 1.12.7

SUMMARY
=====

PARAMETERS
* /rosdistro: kinetic
* /rosversion: 1.12.7

NODES

auto-starting new master
process[master]: started with pid [5218]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to 9e24585a-60c8-11e7-b113-08d40c80c500
```

```
process[rosout-1]: started with pid [5231]
started core service [/rosout]
```

3.1.2. 简易安装

如果用户的Ubuntu版本是16.04.x或Linux Mint 18.x，那么可以使用下面准备好的一个脚本来简化上面描述的ROS安装。

```
$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_kinetic.sh
$ chmod 755 ./install_ros_kinetic.sh
$ bash ./install_ros_kinetic.sh
```

在上述简易安装中，由wget命令下载的install_ros_kinetic.sh shell脚本包含3.1.1中涵盖的常规安装的内容，下面将介绍的3.2.1的ROS环境设置的内容将自动建立简易的ROS安装和配置。

3.2. 搭建ROS开发环境

3.2.1. ROS配置

要加载一个配置文件，就像在ROS安装过程中使用的以下命令一样，每次打开新的终端窗口时都必须运行它。

```
$ source /opt/ros/kinetic/setup.bash
$ source ~/catkin_ws/devel/setup.bash
```

为了避免这个繁琐的任务，可以设置终端，使得每次打开新的终端窗口时，都读入配置文件。另外，配置ROS网络，还将常用的命令简化为快捷命令。

首先，使用文本编辑器gedit程序来加载bashrc文件。本书使用gedit编辑文档，但也可以使用atom、sublime text、vim、emacs、nano和visual studio code等等。

```
$ gedit ~/.bashrc
```

打开bashrc文件就可以看到已经有了很多设置。不要修改以前的设置，而是到bashrc文件的最底部添加以下内容（xxx.xxx.xxx.xxx是用户自己的IP地址，请参阅第31页的ifconfig了解IP地址设置）。输入了所有内容之后，保存用户的更改并退出gedit。

```
#!/.bashrc

# Set ROS Kinetic
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

# Set ROS Network
export ROS_HOSTNAME=xxx.xxx.xxx.xxx
export ROS_MASTER_URI=http://{$ROS_HOSTNAME}:11311

# Set ROS alias command
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'
```

为了让修改了的bashrc文件发挥作用，输入如下命令。或者，如果用户关闭当前正在运行的终端窗口并运行新的终端窗口，用户也将得到相同的效果，因为用户在bashrc中所做的设置会适用于新的终端窗口。

```
$ source ~/.bashrc
```

让我们仔细看看我们目前为止进行的设置。

加载ROS配置

“#”是表明注释的开头字符，后面是注释内容。第二行的“source /opt/ros/kinetic/setup.bash”和第三行的“source ~/catkin_ws/devel/setup.bash”是必须设置的ROS配置文件。

```
# Set ROS Kinetic
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

配置ROS网络

下面是ROS_MASTER_URI和ROS_HOSTNAME设置。此配置非常重要，因为ROS通过网络在节点之间传递消息。首先，两个项目必须输入自己的网络IP。将来，如果有专用于总机（MASTER PC）的PC，并且机器人使用主机（HOST PC），则可以通过分别输入不同的IP地址进行通信。现在让我们输入他们的网络IP。以下示例显示IP为192.168.1.100的情况的示例。用户可以在终端窗口中使用ifconfig命令检查用户的IP信息。

```
# Set ROS Network
export ROS_HOSTNAME=192.168.1.100
export ROS_MASTER_URI=http://${ROS_HOSTNAME}:11311
```

如果用户在一台PC上运行所有ROS功能包，则可以指定localhost而不是指定特定的IP。

```
# Set ROS Network
export ROS_HOSTNAME=localhost
export ROS_MASTER_URI=http://localhost:11311
```



Ifconfig⁷

在Linux中，使用ifconfig命令确认本机IP。如果在终端窗口中运行ifconfig命令（如以下示例所示），则会在wp2s0的inet addr中显示其有线网络IP地址，而在enp3s0的inet addr中显示本机的无线网IP地址。在我的情况下，我主要使用有线局域网，但我也接到了无线局域网。在以下示例中，有线连接的IP为192.168.1.100。

```
$ ifconfig
enp3s0    Link encap:Ethernet HWaddr d8:cb:8a:f1:74:2b
          inet addr:192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::60fc:7e2b:b877:f82b/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:52 errors:0 dropped:0 overruns:0 frame:0
```

⁷ <https://en.wikipedia.org/wiki/Ifconfig>

```
TX packets:81 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:10172 (10.1 KB) TX bytes:8917 (8.9 KB)
Interrupt:19

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
        RX packets:3520 errors:0 dropped:0 overruns:0 frame:0
        TX packets:3520 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:560728 (560.7 KB) TX bytes:560728 (560.7 KB)

wlp2s0   Link encap:Ethernet HWaddr 08:d4:0c:80:c5:00
        inet addr:192.168.11.19 Bcast:192.168.11.255 Mask:255.255.255.0
        inet6 addr: fe80::a60b:e157:4157:d9dc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:675821 errors:0 dropped:0 overruns:0 frame:0
        TX packets:219992 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:919561165 (919.5 MB) TX bytes:46928931 (46.9 MB)
```

快捷命令

下面将ROS开发中经常使用的命令设为快捷命令。下面的cw、cs和cm是我编写的快捷命令，我是用alias简化命令编写了它们。

- cw：进入~/catkin_ws，预设的catkin工作目录
- cs：进入catkin工作目录中保存源文件的~/catkin_ws/src目录
- cm：在移动到catkin工作目录~/catkin_ws之后，用catkin_make命令构建ROS功能包

```
# Set ROS alias command
```

```
alias cw='cd ~/catkin_ws'  
alias cs='cd ~/catkin_ws/src'  
alias cm='cd ~/catkin_ws && catkin_make'
```



检查ROS配置的方法

可以用`export | grep ROS`命令检查当前的ROS配置。

```
$ export | grep ROS  
declare -x ROSLISP_PACKAGE_DIRECTORIES="/home/pyo/catkin_ws/devel/share/common-lisp"  
declare -x ROS_DISTRO="kinetic"  
declare -x ROS_ETC_DIR="/opt/ros/kinetic/etc/ros"  
declare -x ROS_HOSTNAME="localhost"  
declare -x ROS_MASTER_URI="http://localhost:11311"  
declare -x ROS_PACKAGE_PATH="/home/pyo/catkin_ws/src:/opt/ros/kinetic/share"  
declare -x ROS_ROOT= "/opt/ros/kinetic/share/ros" ROS
```

3.2.2. 集成开发环境（IDE）

集成开发环境（IDE）是一种软件，在这个软件中完成与程序开发相关的所有任务：如编写代码、调试、编译和发布等。很多开发者应该都会有一两个他们最喜欢的IDE。

ROS还可以使用多个IDE⁸。流行的IDE包括Eclipse、CodeBlocks、Emacs、Vim、NetBeans和QtCreator⁹。我原来使用Eclipse，但Eclipse在最近的版本中感觉非常笨重，使用ROS的catkin的构建系统时感到很多不便。所以，在分析了其他IDE之后，我认为在进行轻量级的工作时Visual Studio Code比较合适，而进行GUI界面开发的时候QtCreator最合适。尤其是考虑到ROS的开发、调试，以及视觉工具rqt和RViz都是用Qt开发的；并且可以用Qt插件开发ROS工具的插件，因此可以说QtCreator非常有用。

即使不将Qt作为IDE，它也具有足够的通用编辑器功能，使用`catkin_make`非常方便，因为用户可以直接通过`CMakeLists.txt`加载项目。

⁸ <http://wiki.ros.org/IDEs>

⁹ <https://www.qt.io/ide/>

以下是使用QtCreator的ROS开发环境的描述。在此说明，即使用户使用QtCreator以外的其他IDE，理解以下内容是没有问题的。

安装QtCreator

```
$ sudo apt-get install qtcreator
```

运行QtCreator

以双击QtCreator图标的方式运行QtCreator也不影响其运行。但为了将记录到`~/.bashrc`的ROS路径等设置应用于QtCreator，我们需要打开新的终端窗口，并运行如下命令。这样才能用到`~/.bashrc`里的配置。

```
$ qtcreator
```

通过上面的命令，QtCreator如图3-2运行了。

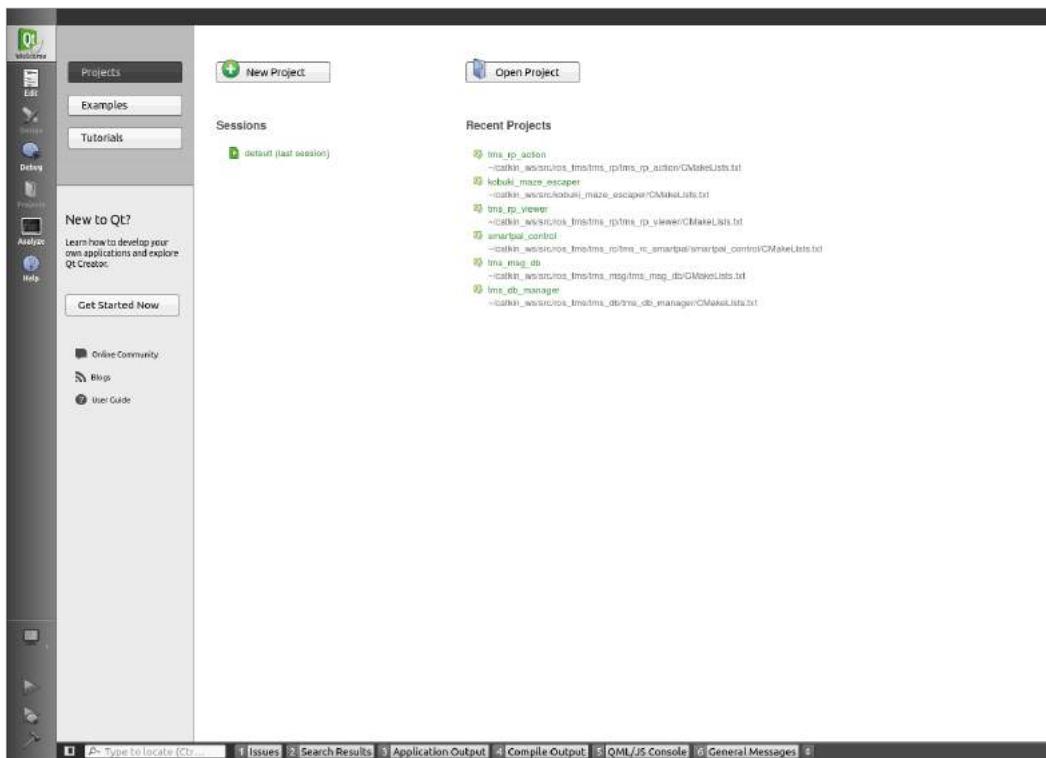


图 3-2 QtCreator IDE

加载ROS功能包

如前所述，QtCreator使用CMakeLists.txt，而ROS功能包也基于CMakeLists.txt，所以如图3-3所示，点击OpenProject键，选择相应的ROS功能包的CMakeLists.txt，就可以方便地打开工程。

可以利用构建快捷库[Ctrl+b]运行catkin_make。但是，与构建相关的文件是在与功能包相同的位置的新目录中创建的。例如，当用户编译tms_rp_action功能包时，所有与构建相关的文件都被放置在build-tms_rp_action-Desktop-Default目录中。最初，要保存在~/catkin_ws/build和~/catkin_ws/devel中的文件是被分开编译的，并放置在一个新的位置，所以用户需要在终端窗口中再次执行catkin_make。并非每次都需要反复这个过程，在开发过程中，在QtCreator开发和调试之后在运行时分别执行catkin_make即可。作为参考，还有用于ROS的Qt Creator插件（https://github.com/ros-industrial/ros_qtc_plugin/wiki），它为ROS开发环境优化了QtCreator。

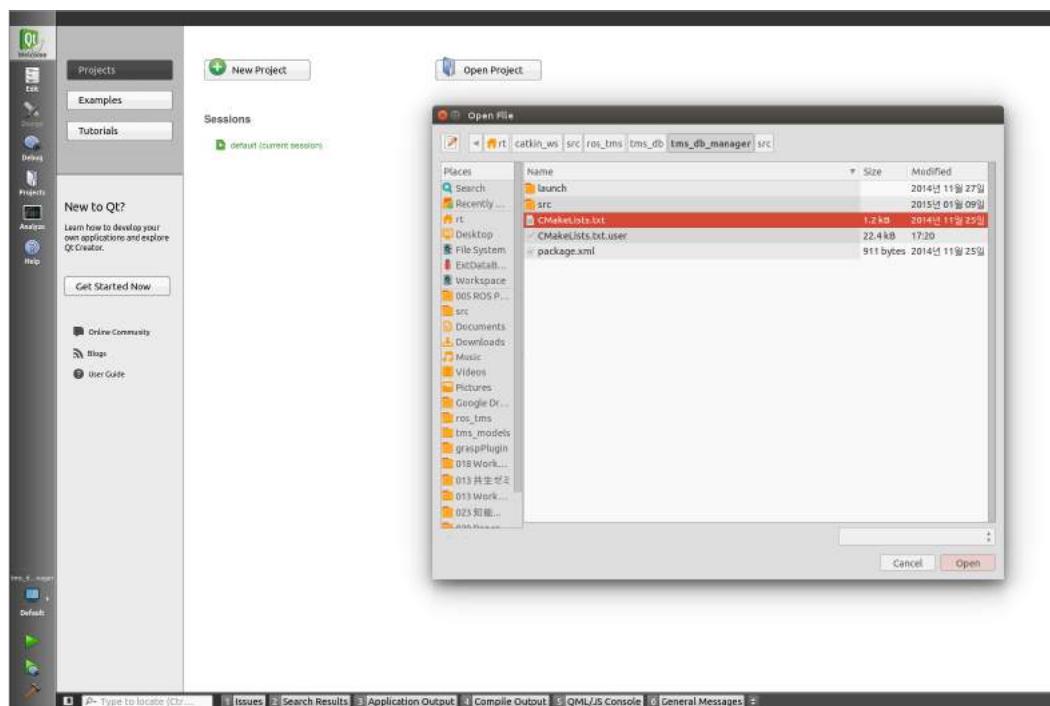


图 3-3 打开QtCreator工程

```

109     ROS_PUBLISHER()
110     {
111         ROS_ASSERT(shutdownDbPublisher());
112     }
113
114     //-----  

115     void getDocumentInformation()
116     {
117         nh_priv.getParam("is_debug", is_debug);
118
119         tms_msg_db::Tmsdb temp_dbdata;
120         char select_query[1024];
121
122         current_environment_information.tmsdb.clear();
123
124         //----  

125         for (int32_t j=0; j<7; i++)
126         {
127             sprintf(select_query, "SELECT * FROM rostmsdb.data_%s WHERE state_id=%d", target_id, j);
128             if(is_debug) ROS_INFO("%s", select_query);
129
130             mysql_query(connector, select_query);
131             result = mysql_use_result(connector);
132
133             if (result == NULL)
134             {
135                 temp_dbdata.note = "Wrong request! Try to check the query";
136                 return;
137             }
138
139             while ((row = mysql_fetch_row(result)) != NULL)
140             {
141                 for(int32_t j=0;j<25;j++) if(is_debug) ROS_INFO("%s", *row[j]);
142                 temp_dbdata.time = row[0];
143                 temp_dbdata.type = row[1];
144                 temp_dbdata.id = atoi(row[2]);
145                 temp_dbdata.name = row[3];
146                 temp_dbdata.x = atof(row[4]);
147                 temp_dbdata.y = atof(row[5]);
148                 temp_dbdata.z = atof(row[6]);
149                 temp_dbdata.rroll = atof(row[7]);
150                 temp_dbdata.pitch = atof(row[8]);
151                 temp_dbdata.yaw = atof(row[9]);
152                 temp_dbdata.offset_x = atof(row[10]);
153                 temp_dbdata.offset_y = atof(row[11]);
154                 temp_dbdata.offset_z = atof(row[12]);
155                 temp_dbdata.weight = atof(row[13]);
156                 temp_dbdata.rfid = row[14];
157                 temp_dbdata.etcdata = row[16];
158             }
159
160             if(mysql_num_rows(result) != 0)
161                 mysql_free_result(result);
162
163             //-----  

164             mysql_query(connector, "SELECT * FROM rostmsdb.id WHERE id=0");
165             result = mysql_use_result(connector);
166             while ((row = mysql_fetch_row(result)) != NULL)
167             {
168                 for(int32_t j=0;j<25;j++) if(is_debug) ROS_INFO("%s", *row[j]);
169                 now = ros::Time::now() + ros::Duration(999999); // GMT +9
170                 temp_dbdata.time = boost::posix_time::to_iso_extended_string(now);
171                 temp_dbdata.id = atoi(row[2]);
172                 temp_dbdata.name = row[3];
173                 temp_dbdata.x = atof(row[4]);
174                 temp_dbdata.y = atof(row[5]);
175                 temp_dbdata.z = atof(row[6]);
176                 temp_dbdata.rroll = atof(row[7]);
177                 temp_dbdata.pitch = atof(row[8]);
178                 temp_dbdata.yaw = atof(row[9]);
179                 temp_dbdata.offset_x = atof(row[10]);
180                 temp_dbdata.offset_y = atof(row[11]);
181                 temp_dbdata.offset_z = atof(row[12]);
182                 temp_dbdata.weight = atof(row[13]);
183                 temp_dbdata.rfid = row[14];
184                 temp_dbdata.etcdata = row[16];
185             }
186         }
187     }
188
189     //-----  

190     // Delete old data in rostmsdb.data_xxx
191     sprintf(delete_query,

```

图 3-4 QtCreator工程编程界面

3.3. ROS操作测试

如果用户已经安装了ROS，下面测试它看看是否正常工作。下面的例子是用ROS提供的turtlesim功能包（节点组合）在屏幕上显示ROS的图标-乌龟，并可以用键盘操纵它的一个节点（程序）。

在本文中，许多ROS特定的术语，如节点、功能包和roscore将在第4章的ROS术语中详细介绍。让我们先确保ROS安装没有问题吧。

运行roscore

打开一个新的终端窗口（Ctrl+Alt+t）并运行以下命令。这将启动负责所有ROS系统的roscore。

```
$ roscore
```

```
File Edit View Search Terminal Help  
pyo@pyo ~ % roscore  
... logging to /home/pyo/.ros/log/d257f518-60cc-11e7-b113-08d40c80c508/roslaunch  
-pyo-7562.log  
Checking log directory for disk usage. This may take awhile.  
Press Ctrl-C to interrupt  
Done checking log file disk usage. Usage is <1GB.  
started roslaunch server http://localhost:38881/  
ros_comm version 1.12.7  
  
SUMMARY  
=====  
PARAMETERS  
  * /rostdistro: kinetic  
  * /rosversion: 1.12.7  
NODES  
auto-starting new master  
process[master]: started with pid [7573]  
ROS_MASTER_URI=http://localhost:11311/  
setting /run_id to d257f518-60cc-11e7-b113-08d40c80c508  
process[rosout-1]: started with pid [7586]  
started core service [/rosout]
```

图 3-5 roscore的运行画面

运行turtlesim功能包的turtlesim_node

打开一个新的终端并运行以下命令。那么将显示下面的消息，并执行turtlesim功能包的turtlesim_node。可以在一个单独的蓝色窗口中看到一只乌龟（乌龟的形状会随着运行而随机变化，所以它可能与图3-6不同）。

```
$ rosrun turtlesim turtlesim_node  
[INFO] [1499182058.960816044]: Starting turtlesim with node name /turtlesim  
[INFO] [1499182058.966717811]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```



图 3-6 乌龟的移动画面

运行turtlesim功能包的turtle_teleop_key

打开一个新的终端窗口并执行以下命令。将显示下面消息，并执行turtlesim功能包中的turtle_teleop_key。如果您按下该终端窗口上的方向键（←，→，↑，↓），您将看到乌龟按照图3-6右侧所示的方向键移动。您必须在终端窗口中输入键盘。这看似只是一个简单的仿真，但实体机器人也可以通过这种方式进行远程控制。

```
$ rosrun turtlesim turtle_teleop_key  
Reading from keyboard  
-----  
Use arrow keys to move the turtle.
```



在终端窗口中使用[Tab]键

Linux环境中经常需要在终端窗口中输入命令。起初，有许多用户会不熟悉，所以表示很不方便。但积累了经验之后会发现是很快很方便的方式。但是，即使有经验的用户也不会记住所有的命令，而是使用[Tab]键。Linux终端窗口中的[Tab]键会提供命令的自动完成。有了这个，你不需要记住所有的命令，你可以快速，准确地输入命令而不会出现拼写错误。例如，让我们看看rosrun命令。在turtlesim之后使用[Tab]键来查找turtlesim包中可用的各种节点，如下所示：

```
$ rosrun turtlesim [Tab]
```

继续输入turtle_teleop之后，按[Tab]键将自动完成可用的命令。这不仅是ROS，也适用于Linux的所有命令。

```
$ rosrun turtlesim turtle_teleop[Tab]
```

```
$ rosrun turtlesim turtle_teleop_key
```

运行rqt_graph功能包的rqt_graph

在新的终端窗口中执行rqt_graph命令，这将启动rqt_graph功能包的rqt_graph节点。其结果是当前正在运行的节点（程序）的信息图，如图3-7所示。

```
$ rqt_graph
```

rqt_graph节点以GUI形式显示关于当前正在运行的节点的信息。圆圈表示节点，正方形表示话题。从图3-7可以看到，箭头从/teleop_turtle节点开始，通往/turtlesim。这表示两个节点正在运行并且这两个节点之间正在发生消息通信。

并且/turtle1/cmd_vel是turtle1话题的子话题，是两个节点之间的话题的名称，意味着在teleop_turtle节点中通过键盘输进来的速度命令通过话题将消息发送给turtlesim。

换句话说，使用先前执行的两个节点，将键盘命令传送到机器人仿真器。欲了解更多信息，请参考以下章节。如果您已圆满完成目前为止的测试，那么您做完了ROS运行测试。

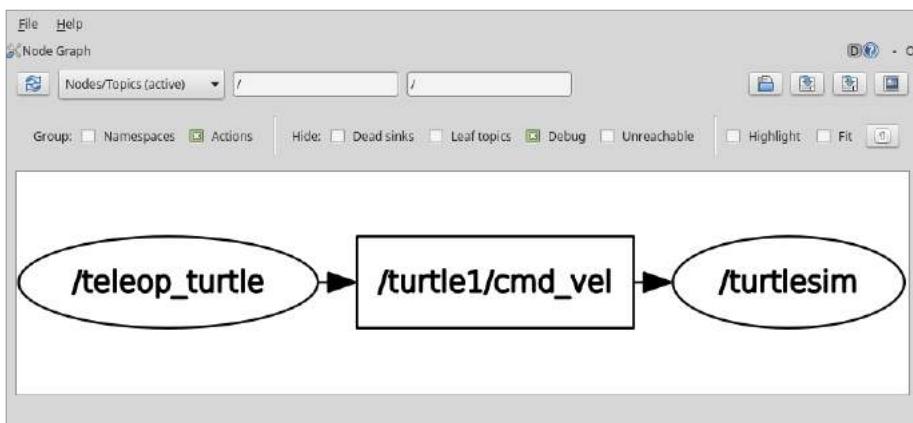


图 3-7 rqt_graph节点

退出节点

在终端窗口中按[Ctrl+c]终止所有被执行的roscore和节点。作为参考，[Ctrl+c]用于强制终止Linux/Unix中的程序。

第4章

ROS的重要概念

为了开发与ROS有关的机器人程序，有必要了解ROS的重要概念¹。首先让我们看一下ROS中使用的术语，接下来我们来看看ROS的重要概念：消息通信、消息文件、名称（Name）、坐标变换（TF）、客户端库、不同设备之间的通信、文件系统和构建系统。

4.1. ROS术语

本节总结了常用的ROS术语，所以读者可以用作ROS术语词典。大部分应该是初次遇到的术语。即使有不明白的术语，让我们先略过。这些可以通过后面各章的例子和实习掌握到。

ROS

ROS是一个用于开发机器人应用程序的、类似操作系统的机器人软件平台。ROS提供开发机器人应用程序时所需的硬件抽象、子设备控制，以及机器人工程中广泛使用的传感、识别、绘图、运动规划等功能。此外ROS还提供进程之间的消息解析、功能包管理、库和丰富的开发及调试工具。

主节点

主节点（master）²负责节点到节点的连接和消息通信，类似于名称服务器（Name Server）。roscore是它的运行命令，当您运行主节点时，可以注册每个节点的名字，并根据需要获取信息。没有主节点，就不能在节点之间建立访问和消息交流（如话题和服务）。

主节点使用XML远程过程调用（XMLRPC， XML-Remote Procedure Call）³与节点进行通信。XMLRPC是一种基于HTTP的协议，主节点不与连接到主节点的节点保持连接。换句话说，节点只有在需要注册自己的信息或向其他节点发送请求信息时才能访问主节点并获取信息。通常情况下，不检查彼此的连接状态。由于这些特点，ROS可用于非常大而复杂的环境。XMLRPC也非常轻便，支持多种编程语言，使其非常适合支持各种硬件和语言的ROS。

¹ <http://wiki.ros.org/ROS/Concepts>

² <http://wiki.ros.org/Master>

³ <https://en.wikipedia.org/wiki/XML-RPC>

当启动ROS时，主节点将获取用户设置的ROS_MASTER_URI变量中列出的URI地址和端口。除非另外设置，默认情况下，URI地址使用当前的本地IP，端口使用11311。

节点

节点（node）⁴是指在ROS中运行的最小处理器单元。可以把它看作一个可执行程序。在ROS中，建议为一个目的创建一个节点，建议设计时注重可重用性。例如，在移动机器人的情况下，为了驱动机器人，将每个程序细分化。也就是说，使用传感器驱动、传感器数据转换、障碍物判断、电机驱动、编码器输入和导航等多个细分节点。

节点在运行的同时，向主节点注册节点的名称，并且还注册发布者（publisher）、订阅者（subscriber）、服务服务器（service server）、服务客户端（service client）的名称，且注册消息形式、URI地址和端口。基于这些信息，每个节点可以使用话题和服务与其他节点交换消息。

节点使用XMLRPC与主站进行通信，并使用TCP/IP通信系列的XMLRPC或TCPROS⁵进行节点之间的通信。节点之间的连接请求和响应使用XMLRPC，而消息通信使用TCPROS，因为它是节点和节点之间的直接通信，与主节点无关。URI地址和端口则使用存储于运行当前节点的计算机上的名为ROS_HOSTNAME的环境变量作为URI地址，并将端口设置为任意的固有值。

功能包

功能包（package）⁶是构成ROS的基本单元。ROS应用程序是以功能包为单位开发的。功能包包括至少一个以上的节点或拥有用于运行其他功能包的节点的配置文件。它还包含功能包所需的所有文件，如用于运行各种进程的ROS依赖库、数据集和配置文件等。目前注册为官方功能包的数量以2017年7月为准为：ROS Indigo多达约2500个（http://repositories.ros.org/status_page/_ros_indigo_default.html），ROS Kinetic多达约1600个（http://repositories.ros.org/status_page/_ros_kinetic_default.html）。除此之外，用户们开发并共享的功能包虽然会有一些重复，但也有约4600个（<http://rosindex.github.io/stats/>）。

⁴ <http://wiki.ros.org/Nodes>

⁵ <http://wiki.ros.org/ROS/TCPROS>

⁶ <http://wiki.ros.org/Packages>

元功能包

元功能包 (metapackage)⁷ 是一个具有共同目的的功能包的集合。例如，导航元功能包包含AMCL、DWA、EKF和map_server等10余个功能包。

消息

节点之间通过消息 (message)⁸ 来发送和接收数据。消息是诸如integer、floating point和boolean等类型的变量。用户还可以使用诸如消息里包括消息的简单数据结构或列举消息的消息数组的结构。使用消息的通信方法包括TCPROS, UDPROS等，根据情况使用单向消息发送/接收方式的话题 (topic) 和双向消息请求 (request) /响应 (response) 方式的服务 (service)。

话题

话题 (topic)⁹ 就是“故事”。在发布者 (publisher) 节点关于故事向主节点注册之后，它以消息形式发布关于该故事的广告。希望接收该故事的订阅者 (subscriber) 节点获得在主节点中以这个话题注册的那个发布者节点的信息。基于这个信息，订阅者节点直接连接到发布者节点，用话题发送和接收消息。

发布与发布者

发布 (publish) 是指以与话题的内容对应的消息的形式发送数据。为了执行发布，发布者 (publisher) 节点在主节点上注册自己的话题等多种信息，并向希望订阅的订阅者节点发送消息。发布者在节点中声明自己是执行发布的个体。单个节点可以成为多个发布者。

订阅与订阅者

订阅是指以与话题内容对应的消息的形式接收数据。为了执行订阅，订阅者节点在主节点上注册自己的话题等多种信息，并从主节点接收那些发布此节点要订阅的话题的发布者节点的信息。基于这个信息，订阅者节点直接联系发布者节点来接收消息。订阅者在节点中声明自己执行订阅的个体。单个节点可以成为多个订阅者。

⁷ <http://wiki.ros.org/Metapackages>

⁸ <http://wiki.ros.org/Messages>

⁹ <http://wiki.ros.org/Topics>

发布和订阅概念中的话题是异步的，这是一种根据需要发送和接收数据的好方法。另外，由于它通过一次的连接，发送和接收连续的消息，所以它经常被用于必须连续发送消息的传感器数据。然而，在某些情况下，需要一种共同使用请求和响应的同步消息交换方案。因此，ROS提供叫做服务（service）的消息同步方法。服务分为响应请求的服务服务器和请求后接收响应的服务客户端。与话题不同，服务是一次性的消息通信。当服务的请求和响应完成时，两个节点的连接被断开。

服务

服务（service）¹⁰消息通信是服务客户端（service client）与服务服务器（service server）之间的同步双向消息通信。其中服务客户端请求对应于特定目的任务的服务，而服务服务器则负责服务响应。

服务服务器

服务服务器（service server）是以请求作为输入，以响应作为输出的服务消息通信的服务器。请求和响应都是消息，服务器收到服务请求后，执行指定的服务，并将结果下发给服务客户端。服务服务器用于执行指定命令的节点。

服务客户端

服务客户端（service client）是以请求作为输出并以响应作为输入的服务消息通信的客户端。请求和响应都是消息，并发送服务请求到服务服务器后接收其结果。服务客户端用于传达给定命令并接收结果值的节点。

动作

动作（action）¹¹是在需要像服务那样的双向请求的情况下使用的消息通信方式，不同点是在处理请求之后需要很长的响应，并且需要中途反馈值。动作文件也非常类似于服务，目标（goal）和结果（result）对应于请求和响应。此外，还添加了对应于中途的反馈（feedback）。它由一个设置动作目标（goal）的动作客户端（action client）和一个动作服务器（action server），动作服务器根据目标执行动作，并发送反馈和结果。

¹⁰ <http://wiki.ros.org/Services>

¹¹ <http://wiki.ros.org/actionlib>

动作客户端和动作服务器之间进行异步双向消息通信。

动作服务器

动作服务器（action server）以从动作客户端接收的目标作为输入并且以结果和反馈值作为输出的消息通信的服务器。在接收到来自客户端的目标值后，负责执行实际的动作。

动作客户端

动作客户端（action client）是以目标作为输出并以从动作服务器接收待结果和反馈值作为输入的消息通信的客户端。它将目标交付给动作服务器，收到结果和反馈，并给出下一个指示或取消目标。

参数

ROS中的参数（parameter）¹²是指节点中使用的参数。可以把它想象成一个Windows程序中的*.ini配置文件。这些参数是默认（default）设置的，可以根据需要从外部读取或写入。尤其是，它可以通过使用外部的写入功能实时更改设置值，因此非常有用。例如，您可以指定与外部设备连接的PC的USB端口、相机校准值、电机速度或命令的最大值和最小值等设置值。

参数服务器

参数服务器（parameter server）¹³是指在功能包中使用参数时，注册各参数的服务器。参数服务器也是主节点的一个功能。

catkin

catkin¹⁴是指ROS的构建系统。ROS的构建系统基本上使用CMake（Cross Platform Make），并在功能包目录中的CMakeLists.txt文件中描述构建环境。在ROS中，我们将CMake修改成专为ROS定制的catkin构建系统。catkin从ROS Fuerte版本开始进行

¹² <http://wiki.ros.org/Parameter%20Server#Parameters>

¹³ <http://wiki.ros.org/Parameter%20Server>

¹⁴ <http://wiki.ros.org/catkin>

alpha测试，并从Groovy版本开始核心功能包转换为catkin，且从Hydro版本开始应用于大部分功能包。catkin 构建系统让用户方便使用与ROS相关的构建、功能包管理以及功能包之间的依赖关系等。现在使用ROS的话，需要使用catkin而不是rosbuild。

rosbuild

ROS构建（rosbuild）¹⁵是在构建catkin构建系统之前使用的构建系统，虽然仍有一些用户可以使用，但这只是为ROS版本兼容性保留的，并不是官方推荐的。如果您必须使用rosbuild构建系统使用旧的功能包，我们建议您将rosbuild更改为catkin。

roscore

roscore¹⁶是运行ROS主节点的命令。也可以在另一台位于同一个网络内的计算机上运行它。但是，除了支持多roscore的某些特殊情况，roscore在一个网络中只能运行一个。运行ROS时，将使用您在ROS_MASTER_URI变量中列出的URI地址和端口。如果用户没有设置，会使用当前本地IP作为URI地址并使用端口11311。

rosrun

rosrun¹⁷是ROS的基本运行命令。它用于在功能包中运行一个节点。节点使用的URI地址将存储在当前运行节点的计算机上的ROS_HOSTNAME环境变量作为URI地址，端口被设置为任意的固有值。

roslaunch

如果rosrun是执行一个节点的命令，那么roslaunch¹⁸是运行多个节点的概念。该命令允许运行多个确定的节点。其他功能还包括一些专为执行具有诸多选项的节点的ROS命令，比如包括更改功能包参数或节点名称、配置节点命名空间、设置ROS_ROOT和ROS_PACKAGE_PATH以及更改环境变量¹⁹等。

¹⁵ <http://wiki.ros.org/rosbuild>

¹⁶ <http://wiki.ros.org/roscore>

¹⁷ <http://wiki.ros.org/roshash#rosrun>

¹⁸ <http://wiki.ros.org/roslaunch>

¹⁹ <http://wiki.ros.org/ROS/EnvironmentVariables>

roslaunch使用*.launch文件来设置可执行节点，它基于可扩展标记语言（XML），并提供XML标记形式的多种选项。

bag

用户可以保存ROS中发送和接收的消息的数据，这时用于保存的文件格式称为bag²⁰，是以*.bag作为扩展名。在ROS中，这个功能包可以用来存储信息并在需要时可以回放以前的情况。例如，当使用传感器执行机器人实验时，使用bag将传感器值以消息形式保存。有了这些保存的信息，即使不重复执行之前的实验，也能通过回放保存的bag文件来反复利用当时的传感器值。特别的，如果利用rosbag的记录和回放功能，在开发那些需要反复修改程序的算法的时候会非常有用。

ROS Wiki

ROS的基本说明是一个基于wiki的页面 (<http://wiki.ros.org/>)，它解释了ROS提供的每个功能包和功能。这个维基页面描述了ROS的基本用法、每个功能包的简要说明、用到的参数、作者、许可证、主页、存储库和教程等内容。目前，ROS Wiki拥有超过17,000页的内容。

存储库

每一个公开的功能包在该功能包的wiki上指定一个存储库(repository)。存储库是存储功能包的网站的URL地址，并使用源代码管理系统（如svn、hg和git）来管理问题、开发、下载等。许多当前可用的ROS功能包将github²¹用作存储库。如果您对每个功能包的源代码内容感兴趣，则可以在相应的存储库中进行查阅。

状态图

上面描述的节点、话题、发布者和订阅者之间关系可以通过状态图(graph)直观地表示。它是当前正在运行的消息通信的图形表示。但不能为一次性服务创建状态图。执行它是通过运行rqt_graph功能包的rqt_graph节点完成的。有两种执行命令：rqt_graph和rosrun rqt_graph rqt_graph。

²⁰ <http://wiki.ros.org/Bags>

²¹ <http://www.github.com/>

名称

节点、参数、话题和服务都有名称（name）²²。当使用主节点的参数、话题和服务时，向主节点注册该名称并根据名称进行搜索，然后发送消息。此外，名称非常灵活，因为它们可以在运行时被更改。另外，对于一个节点、参数、话题和服务，也能给其设定多个不同的名称。这种取名规则使得ROS适用于大型项目和复杂系统。

客户端库

ROS是一个客户端库（client library）²³，它为各种语言提供开发环境，以减少对所用语言的依赖性。主要的客户端库包括C++、Python和Lisp。其他语言包括Java、Lua、.NET、EusLisp和R。为此，开发了诸如roscpp、rospy、roslisp、rosjava、roslua、roscs、roseus、PhaROS、rosR等客户端库。

URI

统一资源标识符（URI，Uniform Resource Identifier）是代表Internet上资源的唯一地址。该URI被用作Internet协议中的标识符，是在Internet上所需的基本条件。

MD5

MD5（Message-Digest algorithm 5）²⁴是128位密码散列函数。它主要用于检查程序或文件的完整性，以查看它是否保持原样。在使用ROS消息的通信中，使用MD5来检查消息发送/接收的完整性。

RPC

远程过程调用（RPC，Remote Procedure Call）²⁵意味着远程（Remote）计算机上的程序调用（Call）另一台计算机中的子程序（Procedure）。这个利用TCP/IP、IPX等传输协议的技术在不需要程序员一一进行编程的情况下也能允许计算机在另一个地址空间通过远程控制运行函数或子程序。

²² <http://wiki.ros.org/Names>

²³ <http://wiki.ros.org/Client%20Libraries>

²⁴ <https://en.wikipedia.org/wiki/Md5sum>

²⁵ <http://wiki.ros.org/ROS/Technical%20Overview>

XML

可扩展标记语言（XML，Extensible Markup Language）是W3C推荐用于创建其他特殊用途标记语言的通用标记语言。它是通过使用标签来指定数据结构的语言之一。在ROS中用于*.launch、*.urdf和package.xml等各个部分。

XMLRPC

XML-Remote Procedure Call (XMLRPC) 是一种RPC协议，其编码形式采用XML编码格式，而传输方式采用既不保持连接状态、也不检查连接状态的请求和响应方式的HTTP协议。XMLRPC是一个非常简单的约定，仅用于定义小数据类型或命令。所以它比较简单。有了这个特点，XMLRPC非常轻便，支持多种编程语言，因此非常适合支持各种硬件和语言的ROS。

TCP/IP

传输控制协议（TCP，Transmission Control Protocol）是一种传输控制协议，通常被称为TCP/IP。从互联网协议层的角度来看，它基于IP（Internet Protocol）且使用传输控制协议TCP，以此保证数据传输，并按照发送顺序进行发送/接收。

TCPROS消息和服务中使用的基于TCP/IP的消息方式称为TCPROS，而UDPROS消息及服务中使用的基于UDP的消息方式称为UDPROS。在ROS中，常用的是TCPROS。

CMakeLists.txt

ROS 构建系统的catkin基本上使用了CMake，因此在功能包目录的CMakeLists.txt²⁶文件中描述着构建环境。

package.xml

包含功能包信息的XML文件²⁷，描述功能包名称、作者、许可证和依赖包。

²⁶ <http://wiki.ros.org/catkin/CMakeLists.txt>

²⁷ <http://wiki.ros.org/catkin/package.xml>

4.2. 消息通信

到目前为止，还没有详细解释ROS实际上是如何工作的。在本节中，我们来看看ROS的核心功能和概念。对ROS的概念描述中使用的各个术语的详细描述，请参阅前面介绍的术语表，本节仅涉及概念。实际的编程方法将在第7章ROS编程基础中介绍。

如第2章所述，为了最大化用户的可重用性，ROS是以节点的形式开发的，而节点是根据其目的细分的可执行程序的最小单位。节点则通过消息（message）与其他的节点交换数据，最终成为一个大型的程序。这里的关键概念是节点之间的消息通信，它分为三种。单向消息发送/接收方式的话题（topic）；双向消息请求/响应方式的服务（service）；双向消息目标（goal）/结果（result）/反馈（feedback）方式的动作（action）。另外，节点中使用的参数可以从外部进行修改。这在大的框架中也可以被看作消息通信。消息通信可以用一张图来说明，如图4-1所示，而不同之处总结在了表4-1中。在对ROS进行编程时，为每个目的使用合适的话题、服务、动作和参数是很重要的。

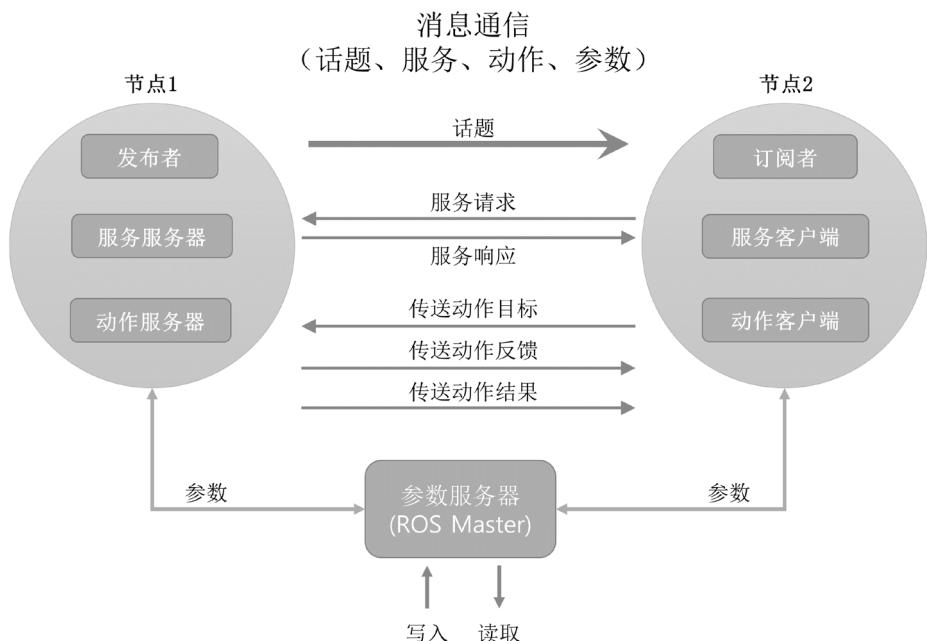


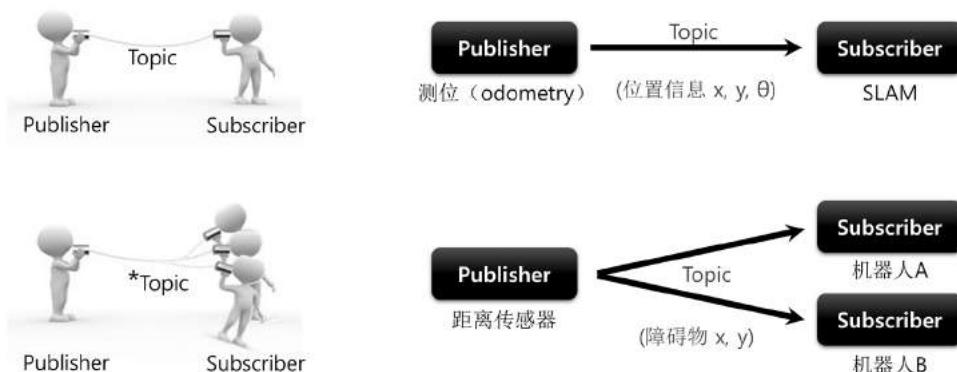
图 4-1 节点间的消息通信

种类	区别		
话题	异步	单向	连续单向地发送/接收数据的情况
服务	同步	双向	需要对请求给出即时响应的情况
动作	异步	双向	请求与响应之间需要太长的时间，所以难以使用服务的情况，或需要中途反馈值的情况

表 4-1 话题、服务和动作之间的差异

4.2.1. 话题 (topic)

如图4-2所示，话题消息通信是指发送信息的发布者和接收信息的订阅者以话题消息的形式发送和接收信息。希望接收话题的订阅者节点接收的是与在主节点中注册的话题名称对应的发布者节点的信息。基于这个信息，订阅者节点直接连接到发布者节点来发送和接收消息。例如，通过计算移动机器人的两个车轮的编码器值生成可以描述机器人当前位置的测位 (odometry)²⁸ 信息，并以话题信息 (x, y, θ) 传达，以此实现异步单向的连续消息传输。话题是单向的，适用于需要连续发送消息的传感器数据，因为它们通过一次的连接连续发送和接收消息。另外，单个发布者可以与多个订阅者进行通信，相反，一个订阅者可以在单个话题上与多个发布者进行通信。当然，这两家发布者都可以和多个订阅者进行通信。



*利用Topic可以实现1:1 的Publisher、Subscriber通信，也可以根据目的实现1:N、N:1和N:N通信。

图 4-2 话题消息通信

²⁸ <http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom>

4.2.2. 服务 (service)

服务消息通信是指请求服务的服务客户端与负责服务响应的服务服务器之间的同步双向服务消息通信，如图4-3所示。前述的发布和订阅概念的话题通信方法是一种异步方法，是根据需要传输和接收给定数据的一种非常好的方法。然而，在某些情况下，需要一种同时使用请求和响应的同步消息交换方案。因此，ROS提供叫做服务的消息同步方法。

一个服务被分成服务服务器和服务客户端，其中服务服务器只在有请求（request）的时候才响应（response），而服务客户端会在发送请求后接收响应。与话题不同，服务是一次性消息通信。因此，当服务的请求和响应完成时，两个连接的节点将被断开。该服务通常被用作请求机器人执行特定操作时使用的命令，或者用于根据特定条件需要产生事件的节点。由于它是一次性的通信方式，又因为它在网络上的负载很小，所以它也被用作代替话题的手段，因此是一种非常有用的通信手段。例如，如图4-3所示，当客户端向服务器请求当前时间时，服务器将确认时间并作出响应。

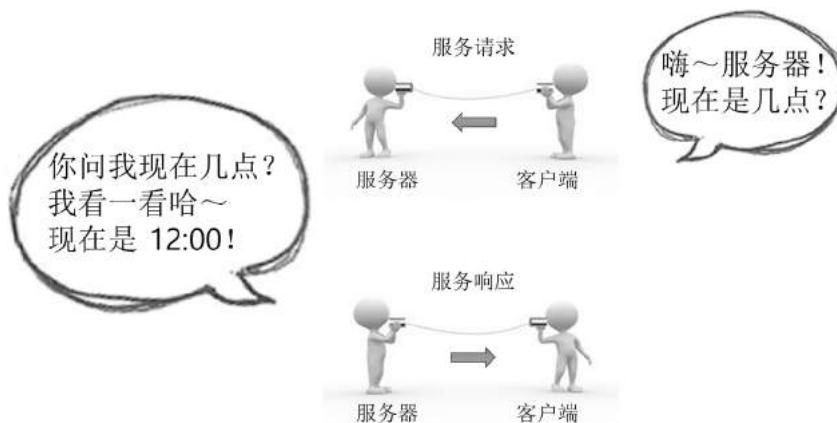


图 4-3 服务消息通信

4.2.3. 动作 (action)

动作²⁹ 消息通信是在如下情况使用的消息通信方式：服务器收到请求后直到响应所需的时间较长，且需要中途反馈值。这与服务非常相似，服务具有与请求和响应分别对应

²⁹ <http://wiki.ros.org/actionlib>

的目标 (goal) 和结果 (result)。除此之外动作中还多了反馈 (feedback)。收到请求后需要很长时间才能响应，又需要中间值时，使用这个反馈发送相关的数据。消息传输方案本身与异步方式的话题 (topic) 相同。反馈在动作客户端 (action client) 和动作服务器 (action server) 之间执行异步双向消息通信，其中动作客户端设置动作目标 (goal)，而动作服务器根据目标执行指定的工作，并将动作反馈和动作结果发送给动作客户端。例如，如图4-4所示，当客户端将家庭服务器设置为服务器时，服务器会时时地通知客户端洗碗、洗衣和清洁等进度，最后将结果值发送给客户端。与服务不同，动作通常用于指导复杂的机器人任务，例如发送一个目标值之后，还可以在任意时刻发送取消目标的命令。

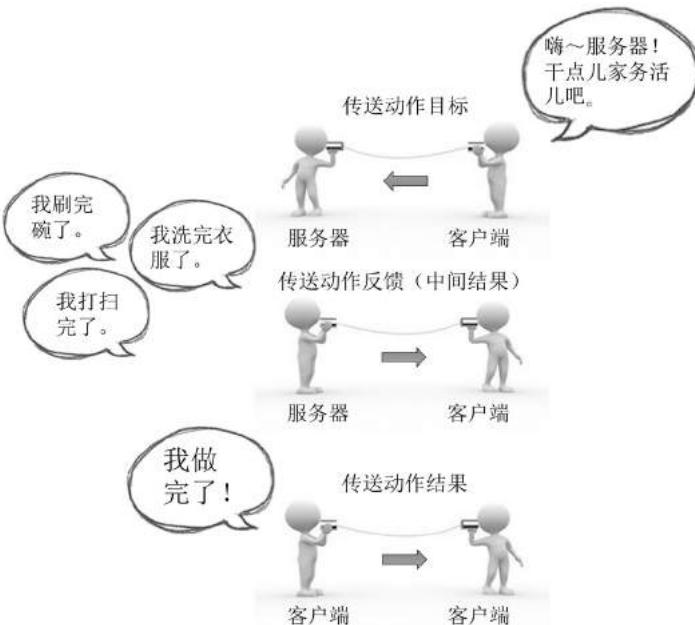


图 4-4 动作消息通信

在这种情况下，发布者、订阅者、服务服务器、服务客户端、动作服务器和动作客户端都存在于不同的节点中，这些节点需要连接才能进行消息通信。这时候，主节点是帮助节点之间的连接。**主节点就像节点名称、话题和服务、动作名称、URI地址和端口以及参数们的名称服务器。换句话说，节点同时向主节点注册自己的信息，并从主节点获取其他节点希望通过主节点访问的节点的信息。然后，节点和节点直接连接进行消息通信。**如图4-5所示。

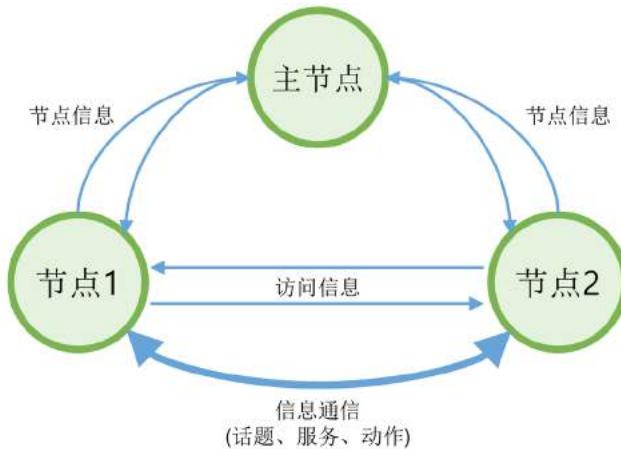


图 4-5 消息通信

4.2.4. 参数 (parameter)

信息通信主要分为话题、服务和动作，而从大的框架来看，参数也可以看作一种消息通信。可以认为参数是节点中使用的全局变量。参数的用途与Windows程序中的*.ini配置文件非常类似。默认情况下，这些设置值是指定的，有需要时可以从外部读取或写入参数。特别是，由于可以通过使用来自外部的写入功能来实时地改变设置值，因此它是非常有用的，因为它可以灵活地应对多变的情况。

尽管参数严格的来说并不是消息通信，但笔者认为它们属于消息通信的范畴，因为它们使用消息。例如，用户可以设置要连接的USB端口、摄像机色彩校正值以及速度和命令的最大值和最小值。

4.2.5. 消息通信的过程

主节点管理节点信息，每个节点根据需要与其他节点进行连接和消息通信。在这里，我们来看看最重要的主节点、节点、话题、服务和动作信息的过程。

运行主节点

节点之间的消息通信当中，管理连接信息的主节点是为使用ROS必须首先运行的必需元素。ROS 主节点使用roscore命令来运行，并使用XMLRPC运行服务器。主节点为了

节点与节点的连接，会注册节点的名称、话题、服务、动作名称、消息类型、URI地址和端口，并在有请求时将此信息通知给其他节点。

```
$ roscore
```



图 4-6 运行主节点

运行订阅者节点

订阅者节点使用rosrun或roslaunch命令来运行。订阅者节点在运行时向主节点注册其订阅者节点名称、话题名称、消息类型、URI地址和端口。主节点和节点使用XMLRPC进行通信。

```
$ rosrun PACKAGE_NAME NODE_NAME  
$ roslaunch PACKAGE_NAME LAUNCH_NAME
```

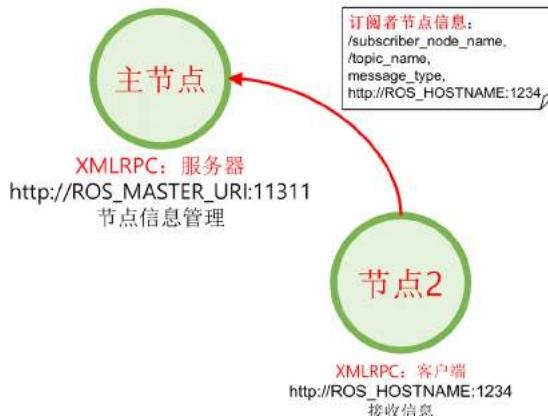


图 4-7 运行订阅者节点

运行发布者节点

发布者节点（与订阅者节点类似）使用rosrun或roslaunch命令来运行。发布者节点向主节点注册发布者节点名称、话题名称、消息类型、URI地址和端口。主节点和节点使用XMLRPC进行通信。

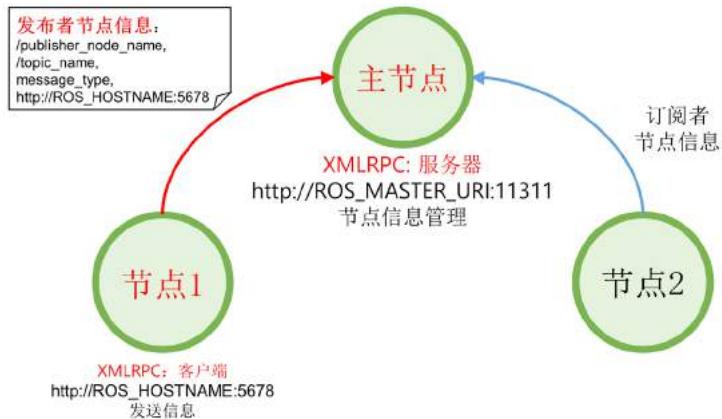


图 4-8 运行发布者节点

通知发布者信息

主节点向订阅者节点发送此订阅者希望访问的发布者的名称、话题名称、消息类型、URI地址和端口等信息。主节点和节点使用XMLRPC进行通信。

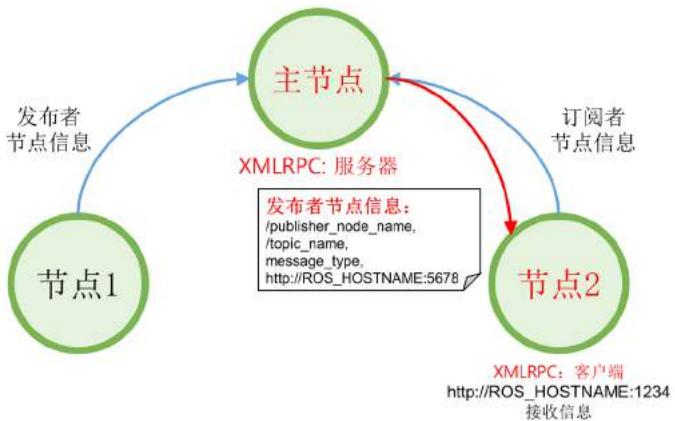


图 4-9 给订阅者节点通知发布者信息

订阅者节点的连接请求

订阅者节点根据从主节点接收的发布者信息，向发布者节点请求直接连接。在这种情况下，要发送的信息包括订阅者节点名称、话题名称和消息类型。发布者节点和订阅者节点使用XMLRPC进行通信。

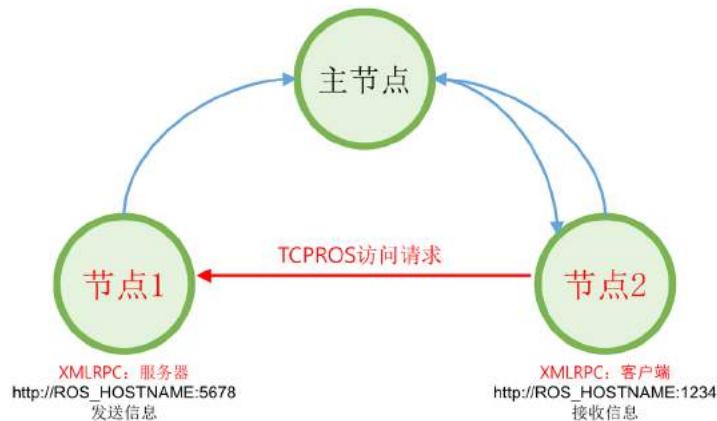


图 4-10 向发布者节点请求连接

发布者节点的连接响应

发布者节点将TCP服务器的URI地址和端口作为连接响应发送给订阅者节点。发布者节点和订阅者节点使用XMLRPC进行通信。

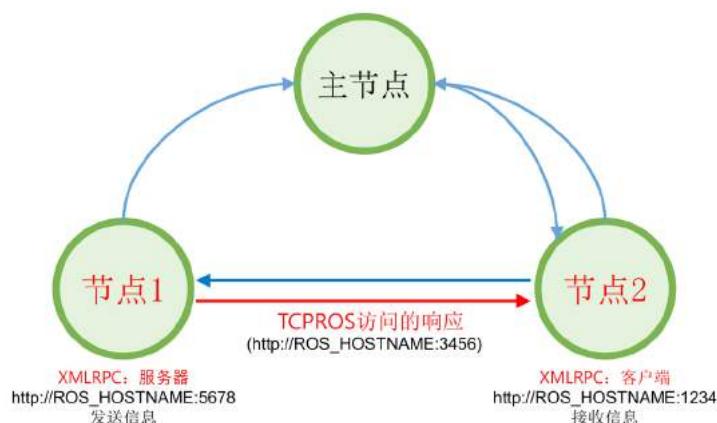


图 4-11 发布者节点的连接响应

TCPROS连接

订阅者节点使用TCPROS创建一个与发布者节点对应的客户端，并直接与发布者节点连接。节点间通信使用一种称为TCPROS的TCP/IP方式。

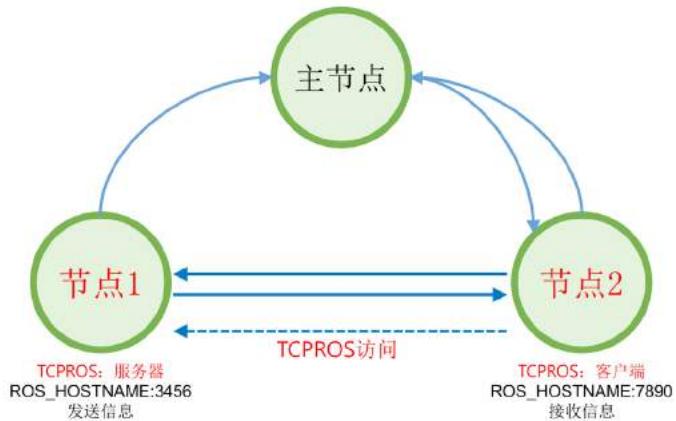


图 4-12 TCPROS连接

发送消息

发布者节点向订阅者节点发送消息。节点间通信使用一种称为TCPROS的TCP/IP方式。



图 4-13 发送话题消息

服务请求及响应

上述内容相当于消息通信中的话题。话题消息通信是只要发布者或订阅者不停止，会持续地发布和订阅。服务分为下面两种。

- 服务客户端：请求服务后等待响应
- 服务服务器：收到服务请求后执行指定的任务，并发送响应。

服务服务器和客户端之间的连接与上述发布者和订阅者之间的TCPROS连接相同，但是与话题不同，服务只连接一次，在执行请求和响应之后彼此断开连接。如果有必要，需要重新连接。



图 4-14 服务请求及响应

动作的目标、结果和反馈

动作 (action) 在执行的方式上好像是在服务 (service) 的请求 (goal) 和响应 (result) 之间仅仅多了中途反馈环节，但实际的运作方式与话题相同。事实上，如果使用rostopic命令来查阅话题，那么可以看到该动作的goal、status、cancel、result和feedback等五个话题。动作服务器和客户端之间的连接与上述发布者和订阅中的TCPROS连接相同，但某些用法略有不同。例如，动作客户端发送取消命令或服务器发送结果值会中断连接，等。

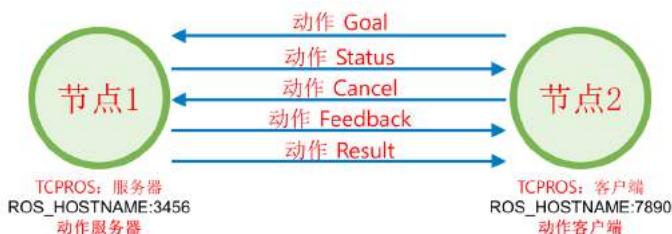


图 4-15 动作消息通信

在前面的内容中，我们用turtlesim测试了ROS的操作。在这个测试中，使用了主节点和两个节点，并且在两个节点之间，使用/turtle1/cmd_vel话题将平移和旋转消息传送

给虚拟海龟。如果按照上面描述的ROS概念思考它，可以以图4-16表达出来。让我们回顾一下之前的ROS操作测试，并再次用ROS概念思考一下吧。

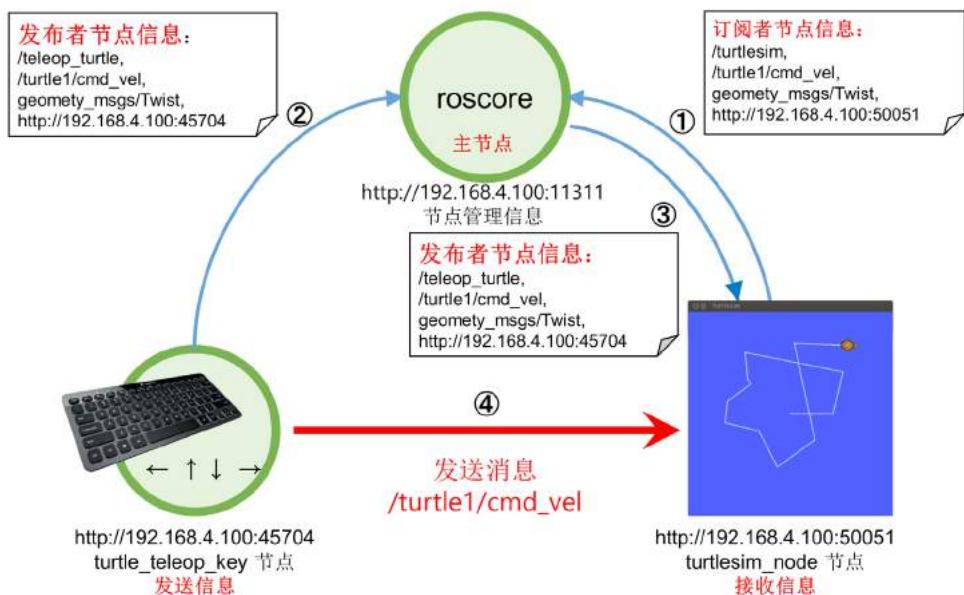


图 4-16 消息通信例子

4.3. 消息

消息（message）³⁰是用于节点之间的数据交换的一种数据形式。前述的话题、服务和动作都使用消息。消息可以是简单的数据结构，如整数（integer）、浮点（floating point）和布尔值（boolean），或者是像“geometry_msgs/PoseStamped”³¹一样消息包含消息的简单数据结构，或者也可以是像“float32[] ranges”或“Point32[10] points”之类的消息数组结构。另外，ROS中常用的头（header、std_msgs/Header）也可以作为消息来使用。这些消息由两种类型组成：字段类型（fieldtype）和字段名称（fieldname）。

³⁰ <http://wiki.ros.org/msg>

³¹ http://docs.ros.org/api/geometry_msgs/html/msg/PoseStamped.html

```
fieldtype1 fieldname1  
fieldtype2 fieldname2  
fieldtype3 fieldname3
```

字段类型应填入ROS数据类型，如表4-2所示。字段名称要填入指示数据的名称。例如，您可以按照下面两行填入。这只是最简单的消息形式，如果要添加更多的消息，则可以将字段类型作为如表4-3所示的数组，并且常常使用消息包含消息的形式。

```
int32 x  
int32 y
```

ROS数据类型	序列化(Serialization)	C++数据类型	Python数据类型
bool	unsigned 8-bit int	uint8_t	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int
int16	signed 16-bit int	int16_t	int
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	std::string	str
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

表 4-2 ROS的基本消息数据类型、序列化方法和与此对应的C++和Python的数据类型

ROS数据类型	序列化(Serialization)	C++数据类型	Python数据类型
fixed-length	no extra serialization	boost::array, std::vector	tuple
variable-length	uint32 length prefix	std::vector	tuple
uint8[]	uint32 length prefix	std::vector	bytes
bool[]	uint32 length prefix	std::vector<uint8_t>	list of bool

表 4-3 ROS的消息数据类型中与数组类似的用法，与此对应的C++和Python的数据类型

前面讲到ROS中常用的头（header、std_msgs/Header）可以作为消息来使用。更具体来说，正如std_msgs³²的Header.msg文件中描述，会记录序列号、时间戳和框架ID，利用它们在消息中记录消息的计数以及时间的计算。

std_msgs/Header.msg

```
# 序列号：是连续增加的ID，每个消息中依次递增+1。  
uint32 seq  
  
# 时间戳：具有两个子属性：以秒为单位的stamp.sec和以纳秒为单位的stamp.nsec。  
time stamp  
  
# 记录框架ID。  
string frame_id
```

消息在程序中的实际用法如下。在第3章的ROS操作测试中执行的turtlesim功能包的teleop_turtle_key节点的例子中，根据方向键（←、→、↑、↓）将turtlesim节点的转速（meter/sec）和旋转速度（radian/sec）用消息发送给turtlesim_node节点。乌龟机器人使用接收到的速度值在屏幕上移动。此时使用的消息是geometry_msgs中的Twist³³消息，是如下形式。

```
Vector3 linear  
Vector3 angular
```

它以Vector3消息的形式声明了linear和angular。它是消息包含消息的形式，其中，Vector3消息是geometry_msgs³⁴之一。这个Vector3³⁵具有以下形式。

```
float64 x  
float64 y  
float64 z
```

换句话说，从teleop_turtle_key节点发布的话题是linear.x、linear.y、linear.z、angular.x、angular.y和angular.z。所有这些都是float64格式，这是上面描述的ROS基

³² http://wiki.ros.org/std_msgs

³³ http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html

³⁴ http://docs.ros.org/api/geometry_msgs/html/index-msg.html

³⁵ http://docs.ros.org/api/geometry_msgs/html/msg/Vector3.html

本类型之一。利用它，键盘的箭头以平移速度（meter/sec）和旋转速度（radian/sec）消息传送，从而驱动TurtleBot。

上一节中描述的基于消息的话题、服务和动作都使用消息，这些消息在形式和概念上都是相似的，但根据其用法分为三类。这将在下一节中更详细地讨论。

4.3.1. msg文件

msg文件是用于话题的消息文件，扩展名为*.msg。例如，上述geometry_msgs中的Twist³⁶消息是有代表性的。这种msg文件只包含一个字段类型和一个字段名称。

```
geometry_msgs/Twist.msg
```

```
Vector3 linear  
Vector3 angular
```

4.3.2. srv文件

srv文件是服务使用的消息文件，扩展名为*.srv。例如，sensor_msgs的SetCameraInfo³⁷消息是典型的srv文件。与msg文件的主要区别在于三个连字符（---）作为分隔符，上层消息是服务请求消息，下层消息是服务响应消息。

```
sensor_msgs/SetCameraInfo.srv
```

```
sensor_msgs/CameraInfo camera_info  
---  
bool success  
string status_message
```

³⁶ http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html

³⁷ http://docs.ros.org/api/sensor_msgs/html/srv/SetCameraInfo.html

4.3.3. action文件

action消息³⁸文件是动作³⁹中使用的消息文件，它使用*.action扩展名。与msg和srv不同，它不是一个比较常见的消息文件，所以没有典型的官方消息文件，但是可以像下面的例子一样使用它。与msg和srv文件的主要区别在于，三个连字符（---）在两个地方用作分隔符，第一部分是goal消息，第二部分是result消息，第三部分是feedback消息。最大的区别是来自action文件的feedback信息。action文件的goal消息和result消息与上述srv文件的请求消息和响应消息角色相同，但action文件的feedback消息用于传输指定进程执行过程中的中途值。在下面的例子中，当机器人的出发点start_pose和目标点goal_pose的位置和姿态作为请求值被传送时，机器人移动到预定的目标位置并且将发送最终到达的result_pose的位置和姿态。另外，用percent_complete消息，周期性地发送到达目标地点进程的百分比。

```
geometry_msgs/PoseStamped start_pose
geometry_msgs/PoseStamped goal_pose
---
geometry_msgs/PoseStamped result_pose
---
float32 percent_complete
```

4.4. 名称 (name)

ROS有一个称为图（graph）的抽象数据类型作为其基本概念⁴⁰。这显示了每个节点的连接关系以及通过箭头表达发送和接收消息（数据）的关系。为此，服务中使用的节点、话题、消息以及ROS中使用的参数都具有唯一的名称（name）⁴¹。让我们仔细看看话题的名称。话题名称分为相对的方法、全局方法和私有方法，如表4-4所示。

以下代码显示了常用的话题的声明。这将在第7章中详细介绍。在这里，我们通过修改话题名称来理解名称的用法。

³⁸ http://wiki.ros.org/actionlib_msgs

³⁹ <http://wiki.ros.org/actionlib>

⁴⁰ <http://wiki.ros.org/ROS/Concepts>

⁴¹ <http://wiki.ros.org/Names>

```

int main(int argc, char **argv)           // 节点主函数
{
    ros::init(argc, argv, "node1");        // 初始化节点
    ros::NodeHandle nh;                  // 声明节点句柄
    // 声明发布者，话题名 = bar
    ros::Publisher node1_pub = nh.advertise<std_msgs::Int32>("bar", 10);

```

这里的节点名称是/node1。如果您用一个没有任何字符的相对形式的bar来声明一个发布者，这个话题将和/bar具有相同的名字。如果以如下所示使用斜杠 (/) 字符用作全局形式，话题名也是/bar。

```
ros::Publisher node1_pub = nh.advertise<std_msgs::Int32>("/bar", 10);
```

但是，如果使用波浪号 (~) 字符将其声明为私有，则话题名称将变为/node1/bar。

```
ros::Publisher node1_pub = nh.advertise<std_msgs::Int32>("~/bar", 10);
```

这可以按照表4-4所示的各种方式使用。其中/wg意味着命名空间的修改。这在下面的描述中更详细地讨论。

Node	Relative (基本)	Global	Private
/node1	bar → /bar	/bar → /bar	~bar → /node1/bar
/wg/node2	bar → /wg/bar	/bar → /bar	~bar → /wg/node2/bar
/wg/node3	foo/bar → /wg/foo/bar	/foo/bar → /foo/bar	~foo/bar → /wg/node3/foo/bar

表 4-4 名称规则

如果要运行两个摄像头应该如何做？简单地两次执行相关节点将导致之前执行的节点因ROS的性质而终止，因为ROS必须具有唯一的名称。用户可以在运行时更改节点的名称，而不需要运行额外的程序或更改源代码。方法包括命名空间（namespace）和重新映射（remapping）。

为了帮助理解，假设有一个虚拟的camera_package。假设执行camera_package功能包的camera_node会运行camera节点，此时的运行方法如下。

```
$rosrun camera_package camera_node
```

当该节点将相机的图像值发送给image话题时，可以通过rqt_image_view传送该image话题，如下所示。

```
$ rosrun rqt_imgae_view rqt_imgae_view
```

现在让我们通过重新映射来修改这些节点的话题值。如果执行如下命令，只有话题名称将被改为/front/image。其中，image是camera_node的话题名，而这些命令是在运行两个节点的同时，顺便修改名称的例子。

```
$ rosrun camera_package camera_node image:=front/image
$ rosrun rqt_imgae_view rqt_imgae_view image:=front/image
```

例如，当有前、左、右，三个摄像头，且当多次执行同名的节点时，由于节点名重复，该节点将被重复执行，因此节点会被停止。为了避免这种情况，可以采取用同一个名称运行多个不同的节点的方法。下面的命令示例中，name选项使用了两个下划线（__）。附加地说明，选项__ns、__name、__log、__ip、__hostname和__master是运行节点时使用的特殊选项。我们在话题名称选项中使用了一个下划线（_），如果它是private名称，则在现有名称前加上一个下划线。

```
$ rosrun camera_package camera_node __name:=front_device:=/dev/video0
$ rosrun camera_package camera_node __name:=left_device:=/dev/video1
$ rosrun camera_package camera_node __name:=right_device:=/dev/video2
$ rosrun rqt_imgae_view rqt_imgae_view
```

如果想绑定到单一的命名空间，可以如下操作。这样会将指定的节点和话题都绑定到一个命名空间，因此会改变所有的名称。

```
$ rosrun camera_package camera_node __ns:=back
$ rosrun rqt_imgae_view rqt_imgae_view __ns:=back
```

上面讲到了名称的多种用法。名称使得整个ROS系统的灵活运转。在本节中，我们通过节点运行命令rosrun了解了修改名称值的方法。同样，可以使用roslaunch，用一次执行来指定它们的选项。这将在第7章中通过实习例子更详细地讨论。

4.5. 坐标变换 (TF)

描述如图4-17所示的机器人的手臂位置时，可以将其描述为每个关节（joint）的相对坐标变换⁴²。下面举一个人形机器人的手的例子。手连接到手腕，再连接到肘部，而肘部又连接到肩膀。以此逆推，在机器人行走和移动时，利用相对坐标关系，可以将肘部的状态表示为肩部关节状态的函数，继而可以表示为腰部状态的函数，再可以表示为各腿关节状态的函数，最终可以表示为机器人双脚的中心状态的函数。换句话说，当机器人步行移动时，机器人手的坐标会根据各个相关关节的相对坐标变换而移动。另外，除了机器人以外，假设有一个机器人要抓取的物体，那么机器人的原点会相对的位于特定的地图上，而这个物体也是在这个地图上。机器人为了抓取这个物体，需要以自己在地图上的相对位置计算出物体对自己的相对位置，最终抓取物体。在机器人编程中，经过坐标变换的机器人的关节（或带有旋转轴的车轮）和物体的位置是非常重要的，在ROS中将它以坐标变换（Transform）⁴³来表达。

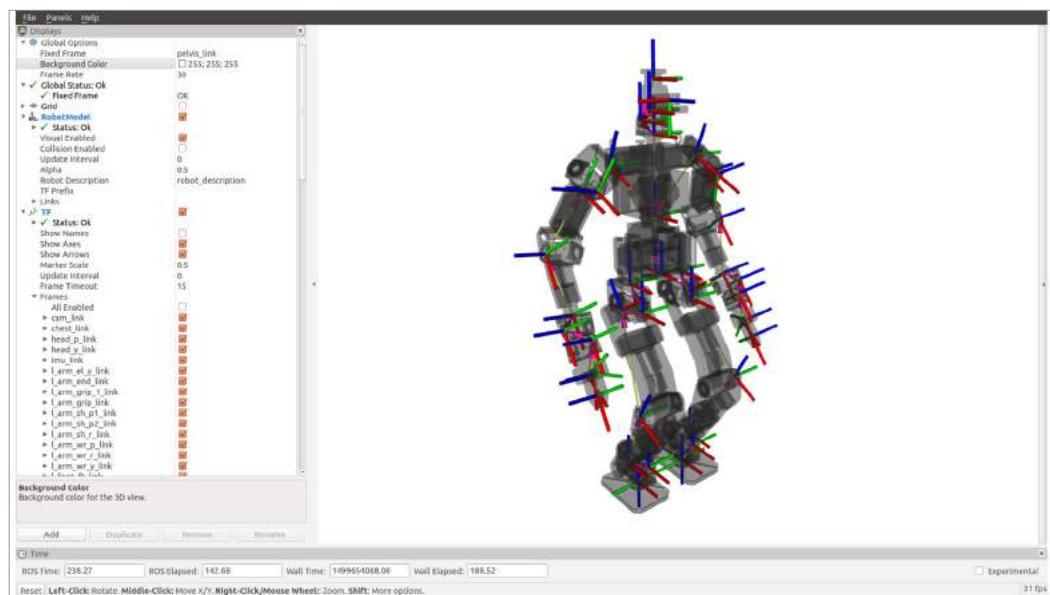


图 4-17 机器人各坐标的表达 (THORMANG3, <http://robots.ros.org/thormang/>)

⁴² <http://wiki.ros.org/geometry/CoordinateFrameConventions>

⁴³ <http://wiki.ros.org/>

ROS中的坐标转换TF在描述组成机器人的每个部分、障碍物和外部物体时是最有用的概念之一。这些可以被描述为位置（position）和方向（direction），统称为姿态（pose）。在此，位置由x、y、z这3个矢量表示，而方向是用四元数（quaternion）x、y、z、w表示。四元数并不直观，因为它们没有使用我们在日常生活中使用的三元数的角度表达方式：滚动角（roll）、俯仰角（pitch）和偏航角（yaw）。但这种四元数方式不存在滚动、俯仰和偏航矢量的欧拉（Euler）方式具有的万向节死锁（gimbal lock）问题或速度问题，因此在机器人工程中人们更喜欢用四元数（quaternion）的形式。因为同样的原因，ROS中也大量使用四元数。当然，考虑到方便，它也提供将欧拉值转换成四元数的功能。

类似上面说明的消息（message），TF使用以下格式⁴⁴。Header用于记录转换的时间，并使用名为child_frame_id的消息来表示下位的坐标。并且为了表达坐标的转换值，使用transformTranslation.x / transform.translation.y / transform.translation.z / transform.rotation.x / transform.rotation.y / transform.rotation.z / transform.rotation.w 的数据形式描述对方的位置和方向。

geometry_msgs/TransformStamped.msg

```
Header header
string child_frame_id
Transform transform
```

到这里，我们简单地了解了TF。我们后面将在移动机器人和机械手臂的建模部分详细讨论TF的实例。

4.6. 客户端库

编程语言多如繁星。为了更快的性能和更好地控制硬件，会选择C++；为了更高的工作效率，会选择Python和Ruby。除了这些之外还有人工智能领域常用的LISP、数值分析等工程用软件MATLAB、安卓中使用的Java，此外还有C#、Go、Haskell、Node.js、Lua、R、EusLisp和Julia等很多种编程语言。很难说哪些更重要。因为目的不同，

⁴⁴ http://docs.ros.org/api/geometry_msgs/html/msg/TransformStamped.html

最合适的语言也会不同。由于ROS需要支持具有多重目的特点的机器人，因此ROS提供面向多种语言的便利接口。具体来说，各个节点可以用各自的语言来编写，而节点间通过消息通信交换信息。其中，允许用各自的语言编写的软件模块就是客户端库（client library）⁴⁵。常用且具有代表性的库有支持C++的roscpp、支持Python的 rospy、支持LISP的roslisp、支持Java的rosjava。此外还有roscs、roseus、rosogo、roshask、rosnodejs、RobotOS.jl、roslua、PhaROS、rosR、rosruby和Unreal-Ros-Plugin等。不仅如此，各个客户端库此时此刻也在为丰富ROS的语言多样性而被开发和改进中。

在本书中，我们将重点介绍C++语言的roscpp。但即便使用不同的语言，也只不过是编程用法相异，概念都是一样的，所以可以选择自己熟悉的语言，通过参考每个客户端库的wiki⁴⁶来编写相应的客户端库。

4.7. 异构设备间的通信

第2章的元操作系统那一节说明了消息通信、消息、名称、坐标变换和客户端库等概念，ROS因为具有这些概念和属性，因此基本上支持异构设备间的通信（参见图4-18）。节点间的通信不受各节点所在的ROS底层的操作系统的种类的影响，也不受编程语言的影响，因此可以很容易地进行通信。例如，即使机器人安装了Linux的发行版之一的Ubuntu，开发者可以在MacOS上监测机器人的状态，同时，用户可以通过基于Android的应用程序（APP）来向机器人发送指令。相应的实习例子将通过第8章关于USB摄像头的内容中的两个不同的PC之间的视频流传输例子讲到。此外，即使是无法安装ROS的那些基于微控制器的嵌入式系统设备，只要提供收发消息的功能，也能在机器人控制器级别上进行消息通信。与此相关的内容将在第9章的嵌入式系统部分进行更详细的讨论。

⁴⁵ <http://wiki.ros.org/Client%20Libraries>

⁴⁶ <http://wiki.ros.org/Client%20Libraries>

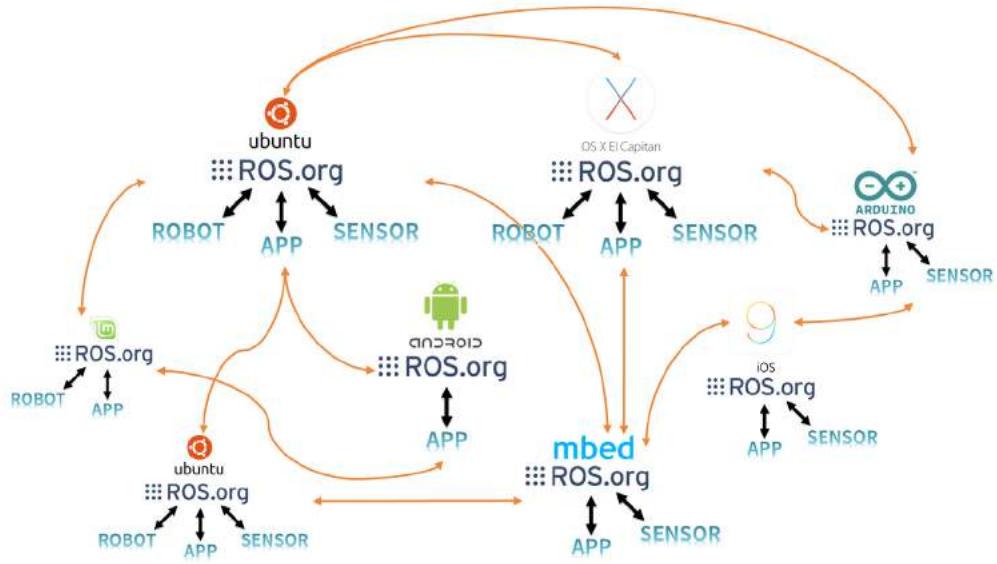


图 4-18 异构设备间的通信

4.8. 文件系统

4.8.1. 文件组织结构

我们来看一下ROS文件的组织结构。在ROS中，组成软件的基本单位是功能包（package），因此ROS应用程序是以功能包为单位开发的。功能包包含一个以上的节点（node，ROS中最小的执行处理器）或包含用于运行其他节点的配置文件。截至2017年7月，ROS Indigo拥有约2500个功能包，而ROS Kinetic拥有约1,600个官方功能包。用户开发和发布的功能包可能有一些重复，但也大约有5000个。这些功能包也会以元功能包（metapackage）的形式来统一管理。元功能包是具有共同目的的功能包的集合体。例如，Navigation元功能包包含10个功能包：AMCL、DWA、EKF和map_server等等。每个功能包都包含一个名为package.xml的文件，该文件是一个包含功能包信息的XML文件，包括其名称，作者，许可证和依赖包。此外，ROS构建系统catkin基本上使用CMake，并在功能包目录中的CMakeLists.txt文件中描述构建环境。另外，它由节点的源代码和消息文件组成，用于节点之间的消息通信。

ROS的文件系统分为安装目录和用户工作目录。安装ROS desktop版本后，在/opt目录中会自动生成名为ros的安装目录，里面会安装有roscore、rqt、RViz、机器人相关库、仿真和导航等核心实用程序。用户很少需要修改这个区域的文件。如果要修改以二进制文件形式分发的功能包，请找到包含源代码的功能包存储库之后利用在“~/catkin_ws/src”目录下用“git clone [存储库地址]”命令直接复制源代码，而不是用“sudo apt-get install ros-kinetic-xxx”形式的功能包安装命令。

用户的工作目录可以在用户想要的位置创建，下面我们就使用Linux用户目录“~/catkin_ws/（在Linux中，‘~’指‘/home/用户名/’目录）”。接下来，让我们来看看ROS安装目录和用户工作目录。



二进制文件安装和源代码安装

ROS功能包的安装方式有二进制文件安装和源代码安装两种。二进制文件安装是使用二进制形式的文件，无需额外的构建，而源代码安装是下载该功能包的源代码之后由用户进行构建之后使用的方式。根据功能包使用的目的不同选择不同的方式。如果需要修改安装包或者要检查源代码的内容，可以使用后一种安装方法。下面用turtlebot3功能包的例子，介绍两种安装方法的不同之处。

1. 二进制文件安装

```
$ sudo apt-get install ros-kinetic-turtlebot3
```

2. 源代码安装

```
$ cd ~/catkin_ws/src  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git  
$ cd ~/catkin_ws/  
$ catkin_make
```

4.8.2. 安装目录

ROS安装在“/opt/ros/[版本名称]”目录中。例如，如果您安装了ROS Kinetic Kame版本，则ROS安装路径为：

- ROS 安装目录路径：/opt/ros/kinetic

文件组织结构

如图4-19所示，“/opt/ros/kinetic”目录下包含bin、etc、include、lib、share目录和一些配置文件。

Name	Size	Type	Date Modified
▶ bin	67 items	Folder	Tue 04 Jul 2017 11:45:38 PM KST
▶ etc	2 items	Folder	Fri 28 Apr 2017 12:14:20 PM KST
▶ include	122 items	Folder	Sun 30 Apr 2017 11:33:11 AM KST
▶ lib	389 items	Folder	Tue 04 Jul 2017 11:45:48 PM KST
▶ share	263 items	Folder	Tue 23 May 2017 01:02:43 PM KST
.catkin	0 bytes	Text	Sat 18 Feb 2017 02:30:25 PM KST
.rosinstall	56 bytes	Text	Sat 18 Feb 2017 02:30:24 PM KST
_setup_util.py	12.4 kB	Text	Sat 18 Feb 2017 02:30:24 PM KST
env.sh	506 bytes	Program	Sat 18 Feb 2017 02:30:24 PM KST
setup.bash	260 bytes	Program	Sat 18 Feb 2017 02:30:24 PM KST
setup.sh	2.5 kB	Program	Sat 18 Feb 2017 02:30:24 PM KST
setup.zsh	270 bytes	Program	Sat 18 Feb 2017 02:30:24 PM KST

图 4-19 ROS文件组织结构

详细内容

ROS目录包含用户在安装ROS时选择的功能包和ROS运行程序。详情如下。

- /bin 可执行的二进制文件
- /etc 与ROS和catkin相关的配置文件
- /include 头文件
- /lib 库文件
- /share ROS功能包
- env.* 配置文件
- setup.* 配置文件

4.8.3. 工作目录

用户可以在任意位置创建工作目录，但是为了方便，本书中将Linux用户目录“`~/catkin_ws/`”用作工作目录。也就是说，您将使用“`/home/用户名/catkin_ws`”目录。例如，如果用户名是“`oroca`”，而catkin目录名设为“`catkin_ws`”，则路径是：

- 工作目录路径 `/home/oroca/catkin_ws/`

文件组织结构

如图4-20所示，在“`/home/用户名/`”目录下有一个名为`catkin_ws`的目录，由目录`build`、`devel`和`src`组成。请注意，`build`和`devel`目录是在`catkin_make`之后创建的。

Name	Size	Type	Date Modified
build	12 items	Folder	Mon 10 Jul 2017 11:55:20 AM KST
devel	9 items	Folder	Mon 10 Jul 2017 11:55:18 AM KST
src	2 items	Folder	Mon 10 Jul 2017 12:05:39 PM KST
.catkin_workspace	98 bytes	Text	Fri 28 Apr 2017 12:48:18 PM KST

图 4-20 catkin workspace的文件组织结构

详细内容

工作目录是对用户创建的功能包和其他开发人员公开的功能包进行存储和构建的空间。用户在该目录中执行与ROS有关的大部分操作。详情如下。

- `/build` 构建相关的文件
- `/devel` msg、srv头文件、用户包库、可执行文件
- `/src` 用户功能包

用户功能包

目录“`~/catkin_ws/src`”是用户源代码的空间。在这个目录中，用户可以保存和建立自己的ROS功能包或其他开发者开发的功能包。ROS的构建系统将在下一节详细介绍。

图4-21显示了笔者编写了`ros_tutorials_topic`功能包之后的状态。下面列举了通常使用的目录和文件，但其文件组成会根据功能包的用途而有所不同。

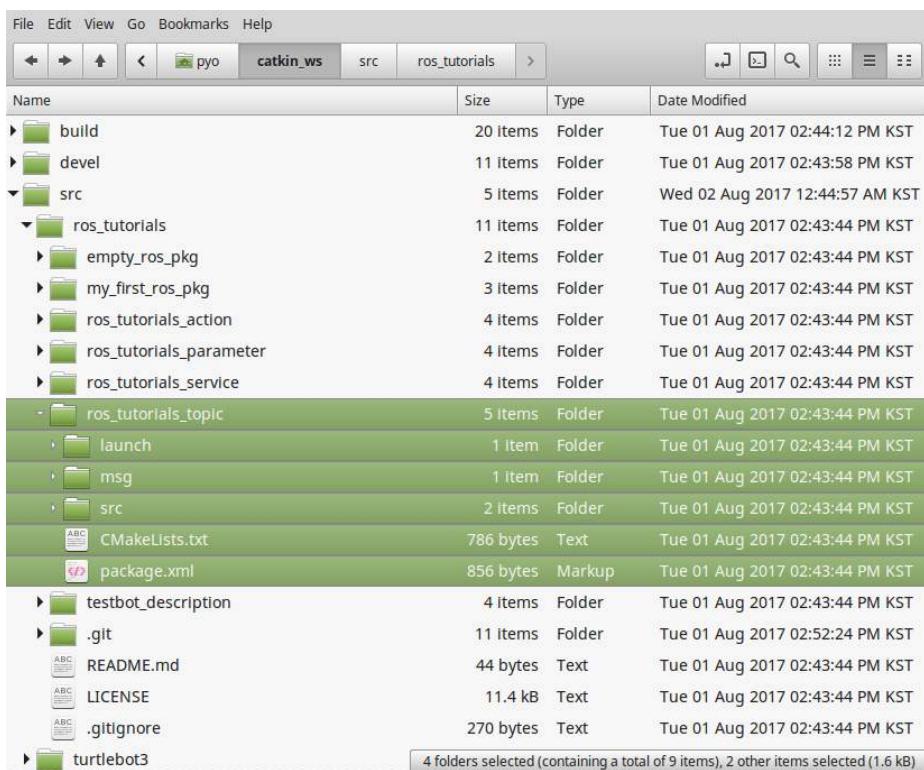


图 4-21 用户功能包的文件组成

- /include 头文件
- /launch 用于roslaunch的启动文件
- /node 用于rospy的脚本
- /msg 消息文件
- /src 源代码文件
- /srv 服务文件

- CMakeLists.txt 构建配置文件
- package.xml 功能包配置文件

4.9. 构建系统

ROS的构建系统默认使用CMake（Cross Platform Make），其构建环境在功能包目录中的CMakeLists.txt文件中描述。在ROS中，CMake被修改为适合于ROS的“catkin”构建系统。

在ROS中使用CMake的是为了在多个平台上构建ROS功能包。因为不同于只支持Unix系列的Make，CMake支持Unix类的Linux、BSD和OS X以外，还支持Windows系列。并且，它还支持Microsoft Visual Studio，也还可以轻松应用于Qt开发。此外，catkin构建系统可以轻松使用与ROS相关的构建、功能包管理和功能包之间的依赖关系。

4.9.1. 创建功能包

创建ROS功能包的命令如下。

```
$ catkin_create_pkg [功能包名称] [依赖功能包1] [依赖功能包n]
```

“catkin_create_pkg”命令在创建用户功能包时会生成catkin构建系统所需的CMakeLists.txt和package.xml文件的包目录。让我们来创建一个简单的功能包，以巩固理解。首先打开一个新的终端窗口（Ctrl + Alt + t）并运行以下命令移至工作目录。

```
$ cd ~/catkin_ws/src
```

要创建的功能包名称是“my_first_ros_pkg”。ROS中的功能包名称全部是小写字母，不能包含空格。格式规则是将每个单词用下划线（_）而不是短划线（-）连接起来。有关ROS编程，请参阅相关页面的编码风格^{47 48}和命名约定。那么下面使用以下命令创建一个名为my_first_ros_pkg的功能包：

⁴⁷ <http://wiki.ros.org/CppStyleGuide>

⁴⁸ <http://wiki.ros.org/PyStyleGuide>

```
$ catkin_create_pkg my_first_ros_pkg std_msgs roscpp
```

上面用“std_msgs”和“roscpp”作为前面命令格式中的依赖功能包的选项。这意味着为了使用ROS的标准消息包std_msgs和客户端库roscpp（为了在ROS中使用C/C++），在创建功能包之前先进行这些选项安装。这些相关的功能包的设置可以在创建功能包时指定，但是用户也可以在创建之后直接在package.xml中输入。

如果已经创建了功能包，“~/catkin_ws/src”会创建“my_first_ros_pkg”功能包目录、ROS功能包应有的内部目录以及CMakeLists.txt和package.xml文件。用户可以用下面的“ls”命令来检查内容，并使用类似Windows资源管理器的基于GUI的Nautilus来检查功能包的内部。

```
$ cd my_first_ros_pkg  
$ ls  
include      → include目录  
src          → 源代码目录  
CMakeLists.txt → 构建配置文件  
package.xml   → 功能包配置文件
```

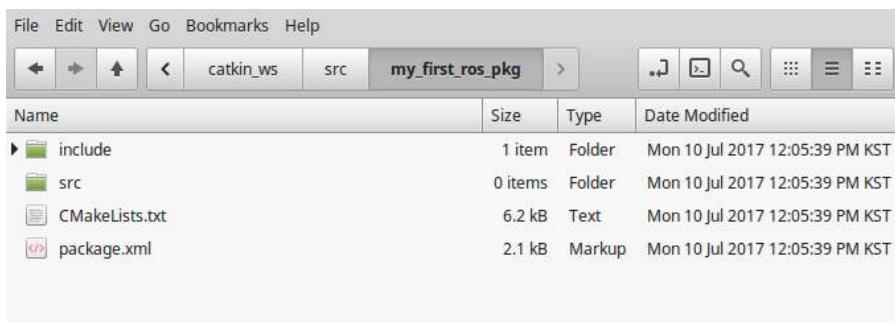


图 4-22 创建功能包时自动生成的目录及文件

4.9.2. 修改功能包配置文件 (package.xml)

必要的ROS配置文件之一的package.xml是一个包含功能包信息的XML文件，包括功能包名称、作者、许可证和依赖功能包。最初没有做任何修改的原始文件如下：

```
<?xml version="1.0"?>
<package>
  <name>my_first_ros_pkg</name>
  <version>0.0.0</version>
  <description>The my_first_ros_pkg package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example: -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="oroca@todo.todo">pyo</maintainer>

  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!-- BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>

  <!-- Url tags are optional, but mutiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/my_first_ros_pkg</url> -->

  <!-- Author tags are optional, mutiple are allowed, one per tag -->
  <!-- Authors do not have to be maintianers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->

  <!-- The *_depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!-- <build_depend>message_generation</build_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!-- <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use run_depend for packages you need at runtime: -->
  <!-- <run_depend>message_runtime</run_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!-- <test_depend>gtest</test_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
```

```
<build_depend>roscpp</build_depend>
<build_depend>std_msgs</build_depend>
<run_depend>roscpp</run_depend>
<run_depend>std_msgs</run_depend>

<!-- The export tag contains other, unspecified, tags -->
<export>
    <!-- Other tools can request additional information be placed here -->

</export>
</package>
```

下面是对每个语句的说明。

- <?xml> 这是一个定义文档语法的语句，随后的内容表明在遵循xml版本1.0。
- <package> 从这个语句到最后</package>的部分是ROS功能包的配置部分。
- <name> 功能包的名称。使用创建功能包时输入的功能包名称。正如其他选项，用户可以随时更改。
- <version> 功能包的版本。可以自由指定。
- <description> 功能包的简要说明。通常用两到三句话描述。
- <maintainer> 提供功能包管理者的姓名和电子邮件地址。
- <license> 记录版权许可证。写BSD、MIT、Apache、GPLv3或LGPLv3即可。
- <url> 记录描述功能包的说明，如网页、错误管理、存储库的地址等。根据功能包的类型，用户可以填写网站、错误跟踪（bugtracker）或存储库的地址。
- <author> 记录参与功能包开发的开发人员的姓名和电子邮件地址。如果涉及多位开发人员，只需在下一行添加<author>标签。
- <buildtool_depend> 描述构建系统的依赖关系。我们正在使用catkin 构建系统，因此输入catkin。
- <build_depend> 在编写功能包时写下您所依赖的功能包的名称。
- <run_depend> 填写运行功能包时依赖的功能包的名称。
- <test_depend> 填写测试功能包时依赖的功能包名称。

- <export> 在使用ROS中未指定的标签名称时会用到<export>。最广泛使用的情况是元功能包的情况，这时用<export> <metapackage/> </export>格式表明是元功能包。
- <metapackage> 在export标签中使用的官方标签声明，当前功能包为一个元功能包时声明它。

笔者修改了功能包配置文件（package.xml），如下所示。读者们也可以根据自己的环境进行修改。如果还不熟悉，可以键入以下内容：

```
package.xml

<?xml version="1.0"?>
<package>
  <name>my_first_ros_pkg</name>
  <version>0.0.1</version>
  <description>The my_first_ros_pkg package</description>
  <license>Apache License 2.0</license>
  <author email="pyo@robotis.com">Yoonseok Pyo</author>
  <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer>
  <url type="bugtracker">https://github.com/ROBOTIS-GIT/ros_tutorials/issues</url>
  <url type="repository">https://github.com/ROBOTIS-GIT/ros_tutorials.git</url>
  <url type="website">http://www.robotis.com</url>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>roscpp</build_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>roscpp</run_depend>
  <export></export>
</package>
```

4.9.3. 修改构建配置文件（CMakeLists.txt）

ROS的构建系统catkin基本上使用CMake，并在功能包目录中的CMakeLists.txt文件中描述构建环境。在这个文件中设置可执行文件的创建、依赖包优先构建、连接器(linker)的创建等等。最初没有做任何修改的原始文件如下：

```
cmake_minimum_required(VERSION 2.8.3)
project(my_first_ros_pkg)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)

## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

#####
## Declare ROS messages, services and actions ##
#####

## To declare and build messages, services or actions from within this
## package, follow these steps:
## * Let MSG_DEPENDENCIES be the set of packages whose message types you use in
##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
## * In the file package.xml:
##   * add a build_depend tag for "message_generation"
##   * add a build_depend and a run_depend tag for each package in MSG_DEPENDENCIES
##   * If MSG_DEPENDENCIES isn't empty the following dependency has been pulled in
##     but can be declared for certainty nonetheless:
##   * add a run_depend tag for "message_runtime"
## * In this file (CMakeLists.txt):
##   * add "message_generation" and every package in MSG_DEPENDENCIES to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * add "message_runtime" and every package in MSG_DEPENDENCIES to
```

```
## catkin_package(CATKIN_DEPENDS ...)
## * uncomment the add_*_files sections below as needed
## and list every .msg/.srv/.action file to be processed
## * uncomment the generate_messages entry below
## * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
# FILES
# Message1.msg
# Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
# FILES
# Service1.srv
# Service2.srv
# )

## Generate actions in the 'action' folder
# add_action_files(
# FILES
# Action1.action
# Action2.action
# )

## Generate added messages and services with any dependencies listed here
# generate_messages(
# DEPENDENCIES
# std_msgs
# )

#####
## Declare ROS dynamic reconfigure parameters ##
#####

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
```

```
## * In the file package.xml:  
## * add a build_depend and a run_depend tag for "dynamic_reconfigure"  
## * In this file (CMakeLists.txt):  
## * add "dynamic_reconfigure" to  
## find_package(catkin REQUIRED COMPONENTS ...)  
## * uncomment the "generate_dynamic_reconfigure_options" section below  
## and list every .cfg file to be processed  
  
## Generate dynamic reconfigure parameters in the 'cfg' folder  
# generate_dynamic_reconfigure_options(  
# cfg/DynReconf1.cfg  
# cfg/DynReconf2.cfg  
# )  
  
#####  
## catkin specific configuration ##  
#####  
## The catkin_package macro generates cmake config files for your package  
## Declare things to be passed to dependent projects  
## INCLUDE_DIRS: uncomment this if you package contains header files  
## LIBRARIES: libraries you create in this project that dependent projects also need  
## CATKIN_DEPENDS: catkin_packages dependent projects also need  
## DEPENDS: system dependencies of this project that dependent projects also need  
catkin_package(  
# INCLUDE_DIRS include  
# LIBRARIES my_first_ros_pkg  
# CATKIN_DEPENDS roscpp std_msgs  
# DEPENDS system_lib  
)  
  
#####  
## Build ##  
#####  
  
## Specify additional locations of header files  
## Your package locations should be listed before other locations  
# include_directories(include)  
include_directories(  
${catkin_INCLUDE_DIRS}  
)
```

```

## Declare a C++ library
# add_library(my_first_ros_pkg
#   src/${PROJECT_NAME}/my_first_ros_pkg.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(my_first_ros_pkg ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
# add_executable(my_first_ros_pkg_node src/my_first_ros_pkg_node.cpp)

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(my_first_ros_pkg_node ${${PROJECT_NAME}_EXPORTED_TARGETS}
# ${catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(my_first_ros_pkg_node
#   ${catkin_LIBRARIES}
# )

#####
## Install ##
#####

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv\_user\_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables and/or libraries for installation
# install(TARGETS my_first_ros_pkg my_first_ros_pkg_node

```

```

# ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
# LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
# RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
# DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
# FILES_MATCHING PATTERN "*.h"
# PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
# # myfile1
# # myfile2
# DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#####
## Testing ##
#####

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_my_first_ros_pkg.cpp)
# if(TARGET ${PROJECT_NAME}-test)
# target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)

```

构建配置文件（CMakeLists.txt）中的每一项如下所示。第一条是操作系统中安装的cmake的最低版本。由于它目前被指定为版本2.8.3，所以如果使用低于此版本的cmake，则必须更新版本。

```
cmake_minimum_required(VERSION 2.8.3)
```

project项是功能包的名称。只需使用用户在package.xml中输入的功能包名即可。请注意，如果功能包名称与package.xml中的<name>标记中描述的功能包名称不同，则在构建时会发生错误，因此需要注意。

```
project(my_first_ros_pkg)
```

find_package项是进行构建所需的组件包。目前，roscpp和std_msgs被添加为依赖包。如果此处没有输入功能包名称，则在构建时会向用户报错。换句话说，这是让用户先创建依赖包的选项。

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
)
```

以下是使用ROS以外的功能包时使用的方法。例如，使用Boost时，必须安装system功能包。功能如前面的说明，是让用户先创建依赖功能包的选项。

```
find_package(Boost REQUIRED COMPONENTS system)
```

catkin_python_setup()选项是在使用Python，也就是使用 rospy时的配置选项。其功能是调用Python安装过程setup.py。

```
catkin_python_setup()
```

add_message_files是添加消息文件的选项。FILES将引用当前功能包目录的msg目录中的*.msg文件，自动生成一个头文件 (*.h)。在这个例子中，我们将使用消息文件Message1.msg和Message2.msg。

```
add_message_files(
  FILES
  Message1.msg
  Message2.msg
)
```

add_service_files是添加要使用的服务文件的选项。使用FILES会引用功能包目录中的srv目录中的*.srv文件。在这个例子中，用户可以选择使用服务文件Service1.srv和Service2.srv。

```
add_service_files(  
    FILES  
    Service1.srv  
    Service2.srv  
)
```

generate_messages是设置依赖的消息的选项。此示例是将DEPENDENCIES选项设置为使用std_msgs消息包。

```
generate_messages(  
    DEPENDENCIES  
    std_msgs  
)
```

generate_dynamic_reconfigure_options是使用dynamic_reconfigure时加载要引用的配置文件的设置。

```
generate_dynamic_reconfigure_options(  
    cfg/DynReconf1.cfg  
    cfg/DynReconf2.cfg  
)
```

以下是catkin 构建选项。INCLUDE_DIRS表示将使用INCLUDE_DIRS后面的内部目录include的头文件。LIBRARIES表示将使用随后而来的功能包的库。

CATKIN_DEPENDS后面指定如roscpp或std_msgs等依赖包。目前的设置是表示依赖于roscpp和std_msgs。DEPENDS是一个描述系统依赖包的设置。

```
catkin_package(  
    INCLUDE_DIRS include  
    LIBRARIES my_first_ros_pkg  
    CATKIN_DEPENDS roscpp std_msgs  
    DEPENDS system_lib  
)
```

include_directories是可以指定包含目录的选项。目前设定为\${catkin_INCLUDE_DIRS}，这意味着将引用每个功能包中的include目录中的头文件。当用户想指定一个额外的include目录时，写在\${catkin_INCLUDE_DIRS}的下一行即可。

```
include_directories(  
    ${catkin_INCLUDE_DIRS}  
)
```

add_library声明构建之后需要创建的库。以下是引用位于my_first_ros_pkg功能包的src目录中的my_first_ros_pkg.cpp文件来创建my_first_ros_pkg库的命令。

```
add_library(my_first_ros_pkg  
    src/${PROJECT_NAME}/my_first_ros_pkg.cpp  
)
```

add_dependencies是在构建该库和可执行文件之前，如果有需要预生成的有依赖性的消息或dynamic_reconfigure，则要先执行。以下内容是优先生成my_first_ros_pkg库依赖的消息及dynamic reconfigure的设置。

```
add_dependencies(my_first_ros_pkg ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
```

add_executable是对于构建之后要创建的可执行文件的选项。以下内容是引用src/my_first_ros_pkg_node.cpp文件生成my_first_ros_pkg_node可执行文件。如果有多个要引用的*.cpp文件，将其写入my_first_ros_pkg_node.cpp之后。如果要创建两个以上的可执行文件，需追加add_executable项目。

```
add_executable(my_first_ros_pkg_node src/my_first_ros_pkg_node.cpp)
```

如前面描述的add_dependencies一样，add_dependencies是一个首选项，是在构建库和可执行文件之前创建依赖消息和dynamic reconfigure的设置。下面介绍名为my_first_ros_pkg_node的可执行文件的依赖关系，而不是上面提到的库。在建立可执行文件之前，先创建消息文件的情况下会经常用到。

```
add_dependencies(my_first_ros_pkg_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
```

target_link_libraries是在创建特定的可执行文件之前将库和可执行文件进行链接的选项。

```
target_link_libraries(my_first_ros_pkg_node  
${catkin_LIBRARIES}  
)
```

此外，还提供了创建官方发行版ROS功能包时使用的Install项目和用于单元测试的Testing项目。

笔者如下修改了构建配置文件（CMakeLists.txt）。读者也可以根据情况进行修改。有关如何使用它的更多信息，请参阅<https://github.com/ROBOTIS-GIT>上发布的TurtleBot3和OP3功能包，相信会对大家有用。

```
CMakeLists.txt  
  
cmake_minimum_required(VERSION 2.8.3)  
project(my_first_ros_pkg)  
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs)  
catkin_package(CATKIN_DEPENDS roscpp std_msgs)  
include_directories(${catkin_INCLUDE_DIRS})  
add_executable(hello_world_node src/hello_world_node.cpp)  
target_link_libraries(hello_world_node ${catkin_LIBRARIES})
```

4.9.4. 编写源代码

在上述CMakeLists.txt文件的可执行文件创建部分（add_executable）中，进行了以下设置。

```
add_executable(hello_world_node src/hello_world_node.cpp)
```

换句话说，是引用功能包的src目录中的hello_world_node.cpp源代码来生成hello_world_node可执行文件。由于这里没有hello_world_node.cpp源代码，我们来写一个简单的例子。

首先，用cd命令转到功能包目录中包含源代码的目录（src），并创建hello_world_node.cpp文件。这个例子使用gedit编辑器，但是您可以使用自己的编辑器，比如vi、gedit、qtcreator、vim或者emacs。

```
$ cd ~/catkin_ws/src/my_first_ros_pkg/src/  
$ gedit hello_world_node.cpp
```

之后如下修改代码。

```
hello_world_node.cpp
```

```
#include <ros/ros.h>  
#include <std_msgs/String.h>  
#include <iostream>  
  
int main(int argc, char **argv)  
{  
    ros::init(argc, argv, "hello_world_node");  
    ros::NodeHandle nh;  
    ros::Publisher chatter_pub = nh.advertise<std_msgs::String>("say_hello_world", 1000);  
    ros::Rate loop_rate(10);  
    int count = 0;  
  
    while (ros::ok())  
    {  
        std_msgs::String msg;  
        std::stringstream ss;  
        ss << "hello world!" << count;  
        msg.data = ss.str();  
        ROS_INFO("%s", msg.data.c_str());  
        chatter_pub.publish(msg);  
        ros::spinOnce();  
        loop_rate.sleep();  
        ++count;  
    }  
    return 0;  
}
```

4.9.5. 构建功能包

所有构建功能包的准备工作都已完成。在构建之前，使用以下命令更新ROS功能包的配置文件。这是一个将之前创建的功能包反映在ROS功能包列表的命令，这并不是必选操作，但在创建新功能包后更新的话使用时会比较方便。

```
$ rospack profile
```

下面是catkin构建。移动到catkin工作目录后进行catkin构建。

```
$ cd ~/catkin_ws && catkin_make
```



快捷命令

如第3.2节“搭建ROS开发环境”中所述，如果在.bashrc文件中设置了“alias cm =’ cd~/ catkin_ws && catkin_make’”，则可以用终端窗口中的cm命令替换以前的命令。这种有用的命令，最好通过复习以前的章节来掌握它。

4.9.6. 运行节点

如果构建无误，那么将在“~/catkin_ws/devel/lib/my_first_ros_pkg”中生成“hello_world_node”文件。

下一步是运行该节点，打开一个终端窗口（Ctrl + Alt + t）并在运行该节点之前先运行roscore。请注意，运行roscore后，ROS中的所有节点都可用，除非退出了roscore，否则只需运行一次。

```
$ roscore
```

最后，打开一个新的终端窗口（Ctrl + Alt + t）并使用以下命令运行节点。这是在名为my_first_ros_pkg的功能包中运行名为hello_world_node的节点的命令。

```
$ rosrun my_first_ros_pkg hello_world_node
```

```
[INFO] [1499662568.416826810]: hello world!0
[INFO] [1499662568.516845339]: hello world!1
[INFO] [1499662568.616839553]: hello world!2
[INFO] [1499662568.716806374]: hello world!3
[INFO] [1499662568.816807707]: hello world!4
[INFO] [1499662568.916833281]: hello world!5
[INFO] [1499662569.016831357]: hello world!6
[INFO] [1499662569.116832712]: hello world!7
[INFO] [1499662569.216827362]: hello world!8
[INFO] [1499662569.316806268]: hello world!9
[INFO] [1499662569.416805945]: hello world!10
```

当您运行这个节点的时候，您可以在终端窗口中看到以hello world! 0,1,2,3 ... 作为字符串发送的消息。这不是一个实际的消息传递，但可以看作是本节讨论的构建系统的成果。由于本节旨在说明ROS的构建系统，因此对于消息和节点的源代码将在下面的章节中将更详细地讨论。

第5章

ROS命令

5.1. ROS命令概述



ROS 维基

ROS命令在<http://wiki.ros.org/ROS/CommandLineTools>上的维基页面上有详细描述。另外，在<https://github.com/ros/cheatsheet/releases>的存储库中，总结了本章中的一些重要命令。这些是本章内容的有用的辅助资料。

ROS可以通过在shell环境中输入命令来进行文件系统的使用、源代码编辑、构建、调试和功能包管理等。为了正确使用ROS，除了基本的Linux命令之外，还需要熟悉ROS专用命令。

为了熟练掌握ROS的各种命令，我们对每个命令的功能进行了简单的描述，并给出了例子。在介绍每条命令时，考虑到使用的频率和重要性，标了星级评分。虽然很难从一开始就很熟练地使用所有的命令，但是随着使用的次数增多，读者会发现越来越方便快捷地使用各个ROS命令。

ROS shell 命令

命令	重要度	命令释义	详细说明
roscd	★★★	ros+cd(changes directory)	移动到指定的ROS功能包目录
rosls	★☆☆	ros+ls(lists files)	显示ROS功能包的文件和目录
rosed	★☆☆	ros+ed(editor)	编辑ROS功能包的文件
roscp	★☆☆	ros+cp(copies files)	复制ROS功能包的文件
rosdp	☆☆☆	ros+pushd	添加目录至ROS目录索引
rosd	☆☆☆	ros+directory	显示ROS目录索引中的目录

ROS执行命令

命令	重要度	命令释义	详细说明
roscore	★★★	ros+core	master (ROS名称服务) + rosout (日志记录) + parameter server (参数管理)
rosrun	★★★	ros+run	运行节点
roslaunch	★★★	ros+launch	运行多个节点及设置运行选项
rosclean	★★☆	ros+clean	检查或删除ROS日志文件

ROS信息命令

命令	重要度	命令释义	详细说明
rostopic	★★★	ros+topic	确认ROS话题信息
rosservice	★★★	ros+service	确认ROS服务信息
rosnode	★★★	ros+node	确认ROS节点信息
rosparam	★★★	ros+param(parameter)	确认和修改ROS参数信息
rosbag	★★★	ros+bag	记录和回放ROS消息
rosmsg	★★☆	ros+msg	显示ROS消息类型
rossrv	★★☆	ros+srv	显示ROS服务类型
rosversion	★★☆	ros+version	显示ROS功能包的版本信息
roswhf	☆☆☆	ros+wtf	检查ROS系统

ROS catkin命令

命令	重要度	详细说明
catkin_create_pkg	★★★	自动生成功能包
catkin_make	★★★	基于catkin构建系统的构建
catkin_eclipse	★★☆	对于用catkin构建系统生成的功能包进行修改，使其能在Eclipse环境中使用
catkin_prepare_release	★★☆	发布时用到的日志整理和版本标记
catkin_generate_changelog	★★☆	在发布时生成或更新CHANGELOG.rst文件
catkin_init_workspace	★★☆	初始化catkin构建系统的工作目录
catkin_find	★★☆	搜索catkin

ROS功能包命令

命令	重要度	命令释义	详细说明
rospack	★★★	ros+pack(age)	查看与ROS功能包相关的信息
rosinstall	★★☆	ros+install	安装ROS附加功能包
rosdep	★★☆	ros+dep(endencies)	安装该功能包的依赖性文件
roslocate	★★☆	ros+locate	ROS功能包信息相关命令
roscreate-pkg	★★☆	ros+create-pkg	自动生成ROS功能包（用于旧的rosbuild系统）
rosmake	★★☆	ros+make	构建ROS功能包（用于旧的rosbuild系统）

5.2. ROS shell 命令

ROS shell命令又被称为rosbash¹。这使我们可以在ROS开发环境中使用Linux中常用的bash shell命令。我们主要使用前缀是ros且带有多种后缀的命令，例如cd、pd、d、ls、ed、cp和run。相关命令如下。

命令	重要度	命令释义	详细说明
roscl	★★★	ros+cd(changes directory)	移动到指定的ROS功能包目录
rosls	★☆☆	ros+ls(lists files)	显示ROS功能包的文件和目录
rosed	★☆☆	ros+ed(editor)	编辑ROS功能包的文件
roscp	★☆☆	ros+cp(copies files)	复制ROS功能包的文件
rosdp	☆☆☆	ros+pushd	添加目录至ROS目录索引
rosd	☆☆☆	ros+directory	显示ROS目录索引

我们来看看其中相对常见的roscl、rosls和rosed命令。



ROS shell命令的使用环境

想要使用ROS shell命令，需要用以下命令安装rosbash，并且只能在设置了source /opt/ros/<ros distribution>/setup.bash的终端窗口中可以使用。这不需要单独安装，只要完成了第3章的ROS开发环境的搭建，则可以使用它。

```
$ sudo apt-get install ros-<ros distribution>-rosbash
```

5.2.1. roscl：移动ROS目录

```
roscl [功能包名称]
```

这是一个移动到保存有功能包的目录的命令。该命令的基本用法是在roscl命令之后将功能包名称写入参数。在以下示例中，turtlesim功能包位于安装ROS的目录中，但是，如果将创建的功能包名称（例如，在第4章中创建的my_first_ros_pkg）作为参数，则会移至您指定的功能包的目录。这是在使用基于命令行的ROS时常用的命令。

¹ <http://wiki.ros.org/rosbash>

```
$ roscd turtlesim  
/opt/ros/kinetic/share/turtlesim$  
$ roscd my_first_ros_pkg  
~/catkin_ws/src/my_first_ros_pkg $
```

请注意，要运行此示例并获得相同的结果，必须安装相关功能包ros-kinetic-turtlesim。如果未安装，请使用以下命令进行安装。

```
$ sudo apt-get install ros-kinetic-turtlesim
```

如果已经安装，将可以看到如下所示的功能包的消息。

```
$ sudo apt-get install ros-kinetic-turtlesim  
[sudo] password for USER:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
ros-kinetic-turtlesim is already the newest version (0.7.1-0xenial-20170613-170649-0800).  
ros-kinetic-turtlesim set to manually installed.  
0 upgraded, 0 newly installed, 0 to remove and 18 not upgraded.
```

5.2.2. rosfs: ROS文件列表

```
rosfs [功能包名称]
```

该命令查看指定的ROS功能包的文件列表。您可以使用roscd命令移动到功能包，然后使用正常的ls命令执行相同的功能，但有时需要立即查看。实际中并不经常使用。

```
$ rosfs turtlesim  
cmake images msg srv package.xml
```

5.2.3. rosed: ROS编辑命令

```
rosed [功能包名称] [文件名称]
```

该命令用于编辑功能包中的特定文件。运行时，它会用用户设置的编辑器打开文件。用于快速修改相对简单的内容。这时用到的编辑器可以在`~/.bashrc`文件中进行指定，如：`export EDITOR= ‘emacs -nw’`。如前所述，它用于需要在命令窗口中直接修改的简单任务，因此不推荐用于除此之外的编写程序的任务。这不是一个经常使用的命令。

```
$ rosed turtlesim package.xml
```

5.3. ROS执行命令

ROS执行命令管理ROS节点的运行。最重要的是，`roscore`被用作节点之间的名称服务器。执行命令是`rosrun`和`roslaunch`。`rosrun`运行一个节点，当运行多个节点或设置各种选项时使用`roslaunch`。`rosclean`是删除节点执行时记录的日志的命令。

命令	重要度	命令释义	详细说明
<code>roscore</code>	★★★	<code>ros+core</code>	master (ROS名称服务) + <code>rosout</code> (日志记录) + parameter server (参数管理)
<code>rosrun</code>	★★★	<code>ros+run</code>	运行节点
<code>roslaunch</code>	★★★	<code>ros+launch</code>	运行多个节点或设置运行选项
<code>rosclean</code>	★★☆	<code>ros+clean</code>	检查或删除ROS日志文件

5.3.1. `roscore`: 运行`roscore`

```
roscore [选项]
```

`roscore`命令会运行主节点，主节点管理节点之间的消息通信中的连接信息。主节点是使用ROS时必须首先被运行的必要元素。ROS 主节点由`roscore`运行命令来驱动，并作为XMLRPC服务器运行。主节点接收多种信息的注册，如节点的名称、话题和服务名称、消息类型、URI地址和端口号，并在收到节点的请求时将此信息通知给其他节点。此外，会运行`rosout2`，这个命令用于记录ROS中使用的ROS标准输出日志，例如DEBUG、INFO、WARN、ERROR和FATAL。它还运行一个管理参数的参数服务器。

² <http://wiki.ros.org/roscpp/Overview/Logging>

当执行roscore时，将用户设置的ROS_MASTER_URI作为主URI，并且驱动主节点。如第3章ROS配置中介绍，用户可以在~/.bashrc设置ROS_MASTER_URI。

```
$ roscore
... logging to /home/pyo/.ros/log/c2d0b528-6536-11e7-935b-08d40c80c500/roslaunch-pyo-20002.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:43517/
ros_comm version 1.12.7

SUMMARY
=====
PARAMETERS
  * /rosdistro: kinetic
  * /rosversion: 1.12.7

NODES
auto-starting new master
process[master]: started with pid [20013]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to c2d0b528-6536-11e7-935b-08d40c80c500
process[rosout-1]: started with pid [20027]
started core service [/rosout]
```

从结果可以看出如下信息：日志保存在/home/xxx/.ros/log/目录中；可以使用[Ctrl+c]退出roscore；roslaunch server、ROS_MASTER_URI等信息；/rosdistro和/rosversion的参数服务器；/rosout节点正在运行。



Log 保存位置

在上面的运行结果中，保存日志的位置是“/home/xxx/.ros/log/”，但实际上它被记录在设置ROS_HOME环境变量的地方。如果ROS_HOME环境变量未设置，则默认值为“~/.ros/log/”。

5.3.2. rosrun：运行ROS节点

```
rosrun [功能包名称] [节点名称]
```

rosrun是执行指定的功能包中的一个节点的命令。以下例子运行turtlesim功能包的turtlesim_node节点。请注意，屏幕上出现的乌龟图标是随机选择并运行的。

```
$ rosrun turtlesim turtlesim_node
[INFO] [1499667389.392898079]: Starting turtlesim with node name /turtlesim
[INFO] [1499667389.399276453]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

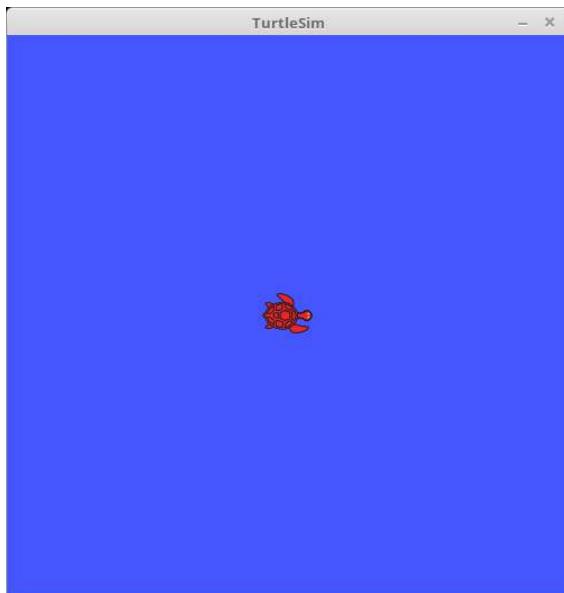


图 5-1 运行turtlesim_node节点的画面

5.3.3. roslaunch：运行多个ROS节点

```
roslaunch [功能包名称] [launch文件名]
```

roslaunch是运行指定功能包中的一个或多个节点或设置执行选项的命令。通过运行openni_launch功能包，可以运行20个以上的节点和10个以上的参数服务器，如camera_nodelet_manager、depth_metric、depth_metric_rect和depth_points。

因此，使用launch文件启动的方式对于运行多个节点非常有用，这是ROS中常用的执行方法。有关创建“*.launch”文件的更多信息，请参阅第7.6节 roslaunch用法。

```
$ roslaunch openni_launch openni.launch  
~省略~
```

请注意，要运行此示例并获得相同的结果，必须安装相关功能包ros-kinetic-openni-launch。如果未安装，请使用以下命令进行安装。

```
$ sudo apt-get install ros-kinetic-openni-launch
```

5.3.4. ros clean：检查及删除ROS日志

```
ros clean [选项]
```

该命令检查或删除ROS日志文件。在运行roscore时，对所有节点的记录都会写入日志文件，随着时间的推移，需要定期使用ros clean命令删除这些记录。

以下是检查日志使用情况的示例。

```
$ ros clean check  
320K ROS node logs → 意味着ROS日志一共占320KB
```

当运行roscore时，如果显示以下警告信息，则意味着日志文件超过1GB，如果用户觉得会让系统不堪重负，请使用ros clean命令将其删除。

```
WARNING: disk usage in log directory [/xxx/.ros/log] is over 1GB.
```

以下是删除ROS日志存储库（笔者是/home/rt/.ros/log）的所有日志的示例。如果要删除它，请按y按钮将其删除。

```
$ ros clean purge  
Purging ROS node logs.  
PLEASE BE CAREFUL TO VERIFY THE COMMAND BELOW!  
Okay to perform:
```

```
rm -rf /home/pyo/.ros/log  
(y/n)?
```

5.4. ROS信息命令

ROS信息命令用于识别话题、服务、节点和参数等信息。尤其是`rostopic`、`rosservice`、`rosnode`和`rosparam`经常被使用，并且`rosbag`是ROS的主要特征之一，它具有记录数据和回放功能，务必要掌握。

命令	重要度	命令释义	详细说明
<code>rostopic</code>	★★★	<code>ros+topic</code>	确认ROS话题信息
<code>rosservice</code>	★★★	<code>ros+service</code>	确认ROS服务信息
<code>rosnode</code>	★★★	<code>ros+node</code>	确认ROS节点信息
<code>rosparam</code>	★★★	<code>ros+param(parameter)</code>	确认和修改ROS参数信息
<code>rosbag</code>	★★★	<code>ros+bag</code>	记录和回放ROS消息
<code>rosmg</code>	★★☆	<code>ros+msg</code>	显示ROS消息类型
<code>rossrv</code>	★★☆	<code>ros+srv</code>	显示ROS服务类型
<code>rosversion</code>	★☆☆	<code>ros+version</code>	显示ROS功能包的版本信息
<code>roswtf</code>	☆☆☆	<code>ros+wtf</code>	检查ROS系统

5.4.1. 运行节点

我们将使用下面的命令，利用ROS提供的`turtlesim`来了解相关的节点、话题和服务。在使用ROS信息命令进行测试之前，需要做好以下准备工作。

运行`roscore`

为确保顺利进行，关闭所有以前运行的终端。然后打开一个新的终端并运行以下命令。

```
$ roscore
```

为了运行`turtlesim`功能包中的`turtlesim_node`节点，打开一个新的终端并运行以下命令。这将从`turtlesim`功能包运行`turtlesim_node`。用户会在一个蓝色的屏幕上看到乌龟。

```
$ rosrun turtlesim turtlesim_node
```

运行turtlesim功能包中的turtle_teleop_key节点

打开一个新的终端并运行以下命令。这将在turtlesim功能包中运行turtle_teleop_key。一旦执行，可以在该终端窗口上，用键盘上的方向键控制乌龟。请自己尝试。当您按下方向键时，屏幕上的乌龟会移动，这是一个简单的仿真，但这是将驱动机器人所需的移动速度（m/s）和旋转速度（rad/s）用消息传送的。有关消息的更多信息和用法，请参阅第4.2节“消息通信”和第7章“ROS编程基础”。

```
$ rosrun turtlesim turtle_teleop_key
```

5.4.2. rosnode: ROS节点

首先，我们需要了解节点（node），所以先复习4.1的术语。

命令	详细说明
rosnode list	查看活动的节点列表
rosnode ping [节点名称]	与指定的节点进行连接测试
rosnode info [节点名称]	查看指定节点的信息
rosnode machine [PC名称或IP]	查看该PC中运行的节点列表
rosnode kill [节点名称]	停止指定节点的运行
rosnode cleanup	删除失连节点的注册信息

rosnode list：列出正在运行中的所有节点

这是列出连接到roscore的所有节点的命令。如果已经运行了roscore和之前准备好的节点（turtlesim_node, turtle_teleop_key），则可以看到终端中列出了用于在roscore进行日志记录的rosout，以及teleop_turtle和turtlesim节点。

```
$ rosnode list
/rosout
/teleop_turtle
/turtlesim
```



节点运行及实际节点的名称

在前面的例子中运行的节点是turtlesim_node和turtle_teleop_key。rosnode list列表中有teleop_turtle和turtlesim的原因是运行的节点名称与实际节点名称不同。例如，turtle_teleop_key节点在源文件中设置为“ros :: init(argc, argv, "teleop_turtle");”。笔者建议使可执行节点的名称等于实际的节点名称。

rosnode ping [节点名称]: 与指定的节点进行连接测试

以下是测试turtlesim节点是否确实连接到当前使用的计算机。如果已连接，它将从节点收到XMLRPC响应，如下所示。

```
$ rosnode ping /turtlesim
rosnode: node is [/turtlesim]
pinging /turtlesim with a timeout of 3.0s
xmlrpc reply from http://192.168.1.100:45470/    time=0.377178ms
```

如果在该节点运行出现问题或通信中断，则显示以下错误消息。

```
ERROR: connection refused to [http://192.168.1.100:55996/]
```

rosnode info [节点名称]: 检查指定节点的信息

使用rosnode info命令可以查看指定节点的信息。基本上，用户可以检查发布者、订阅者和服务等。此外，还可以检查关于节点运行URI和话题输入/输出的信息。由于会显示大量信息，因此省略了内容，所以请务必亲自运行。

```
$ rosnode info /turtlesim
-----
Node [/turtlesim] Publications:
 * /turtle1/color_sensor [turtlesim/Color]
 ~省略~
```

rosnode机器[PC名称或IP]: 查看此PC上运行的所有节点

您可以看到指定设备（PC或终端）上运行的所有节点。

```
$ rosnode machine 192.168.1.100  
/rosout  
/teleop_turtle  
/turtlesim
```

rosnode kill [节点名称]: 终止指定节点的运行

这是一个终止正在运行的节点的命令。您可以在运行节点的终端窗口中使用[Ctrl+c]直接终止节点，但也可以指定要结束的的节点，如下所示。

```
$ rosnode kill /turtlesim  
killing /turtlesim  
killed
```

如果使用该命令终止了节点，则会在运行该节点的终端窗口上显示如下警告消息，并关闭该节点。

```
[WARN] [1499668430.215002371]: Shutdown request received.  
[WARN] [1499668430.215031074]: Reason given for shutdown: [user request]
```

rosnode cleanup: 删除无法验证连接信息的虚拟节点的注册信息

删除连接信息未被确认的虚拟节点的注册信息。当节点由于意外事件而异常终止时，该命令将从节点目录中删除连接中断的节点。这个命令很少使用，但是它非常有用，因为用户不需要重新运行roscore。

```
$ rosnode cleanup
```

5.4.3. rostopic: ROS话题

首先，让我们参考第4.1节“ROS术语”，因为我们需要了解话题。

命令	详细说明
rostopic list	显示活动的话题目录
rostopic echo [话题名称]	实时显示指定话题的消息内容
rostopic find [类型名称]	显示使用指定类型的的话题

命令	详细说明
rostopic type [话题名称]	显示指定话题的消息类型
rostopic bw [话题名称]	显示指定话题的消息带宽 (bandwidth)
rostopic hz [话题名称]	显示指定话题的消息数据发布周期
rostopic info [话题名称]	显示指定话题的信息
rostopic pub [话题名称] [消息类型] [参数]	用指定的话题名称发布消息

在运行ROS话题示例之前，请先关闭所有节点。通过在不同的终端窗口中分别运行以下命令来运行turtlesim_node和turtle_teleop_key。

```
$ roscore  
$ rosrun turtlesim turtlesim_node  
$ rosrun turtlesim turtle_teleop_key
```

rostopic list: 列出活动话题

rostopic list命令显示当前正在发送和接收的所有话题的列表。

```
$ rostopic list  
/rosout  
/rosout_agg  
/turtle1/cmd_vel  
/turtle1/color_sensor  
/turtle1/pose
```

通过将“-v”选项添加到rostopic list命令，可以分开发布话题和订阅话题，并将每个话题的消息类型一起显示。

```
$ rostopic list -v  
Published topics:  
  
* /turtle1/color_sensor [turtlesim/Color] 1 publisher  
* /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher  
* /rosout [rosgraph_msgs/Log] 2 publishers  
* /rosout_agg [rosgraph_msgs/Log] 1 publisher  
* /turtle1/pose [turtlesim/Pose] 1 publisher  
Subscribed topics:
```

```
*      /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
*      /rosout [rosgraph_msgs/Log] 1 subscriber
```

rostopic echo [话题名称]: 实时显示指定话题的消息内容

以下示例实时显示组成/turtle1/pose话题的x、y、theta、linear_velocity和angular_velocity的数据。

```
$ rostopic echo /turtle1/pose
x: 5.35244464874
y: 5.544444561
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
~省略~
```

rostopic find [类型名称]: 显示使用指定类型的的话题

```
$ rostopic find turtlesim/Pose
/turtle1/pose
```

rostopic type [话题名称]: 显示指定话题的消息类型

```
$ rostopic type /turtle1/pose
turtlesim/Pose
```

rostopic bw [话题名称]: 显示指定话题的消息数据带宽 (bandwidth)

在以下示例中，用于/turtle1/pose话题的数据带宽平均为每秒1.27 KB。

```
$ rostopic bw /turtle1/pose
subscribed to [/turtle1/pose]
average: 1.27KB/s
mean: 0.02KB min: 0.02KB max: 0.02KB window: 62 ...
~省略~
```

rostopic hz [话题名称]: 显示指定话题的消息数据发布周期

在以下示例中，用户可以检查/turtle1/pose数据的发布周期。从结果可以看出，该消息以大约62.5Hz（0.016秒= 16毫秒）的频率被发布。

```
$ rostopic hz /turtle1/pose
subscribed to [/turtle1/pose]
average rate: 62.502
min: 0.016s max: 0.016s std dev: 0.00005s window: 62
```

rostopic info [话题名称]: 显示指定话题的信息

在以下示例中，用户可以看到/turtle1/pose话题使用turtlesim/Pose消息类型，发布到/turtlesim节点，并且没有实际订阅的话题。

```
$ rostopic info /turtle1/pose
Type: turtlesim/Pose
Publishers:
* /turtlesim (http://192.168.1.100:42443/)
Subscribers: None
```

rostopic pub [话题名称] [消息类型] [参数]: 使用指定的话题名称发布消息

以下是使用/turtle1/cmd_vel话题名称发布类型为geometry_msgs/Twist的消息的示例。

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
publishing and latching message for 3.0 seconds
```

每个选项的描述如下。

- -1只发布一次消息（实际上只运行一次，但会像以前的结果一样运行3秒）。
- /turtle1/cmd_vel 指定的话题名称
- geometry_msgs/Twist 要发布的消息类型名称
- -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' 在x轴坐标上以每秒2.0 m的速度移动，以z轴为中心，每秒旋转1.8rad



图 5-2 反映了发布的消息的画面

5.4.4. rosservice: ROS服务

由于读者需要了解该服务（service），请参阅第4.1节“ROS术语”。

命令	详细说明
rosservice list	显示活动的服务信息
rosservice info [服务名称]	显示指定服务的信息
rosservice type [服务名称]	显示服务类型
rosservice find [服务类型]	查找指定服务类型的服务
rosservice uri [服务名称]	显示ROSRPC URI服务
rosservice args [服务名称]	显示服务参数
rosservice call [服务名称] [参数]	用输入的参数请求服务

在运行ROS服务相关例子之前先关闭所有节点。通过在不同的终端窗口中运行以下命令来运行turtlesim_node和turtle_teleop_key。

```
$ roscore  
$ rosrun turtlesim turtlesim_node  
$ rosrun turtlesim turtle_teleop_key
```

rosservice list: 显示活动的服务信息

显示活动中的服务的信息。会显示在同一网络中使用的所有服务。

```
$ rosservice list
/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
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

rosservice info [服务名称]: 显示指定服务的信息

以下是使用rosservice的info选项查看/turtle1/set_pen服务的节点名称、URI、类型和参数的示例。

```
$ rosservice info /turtle1/set_pen
Node: /turtlesim
URI: rosrpc://192.168.1.100:34715
Type: turtlesim/SetPen
Args: r g b width off
```

rosservice type [服务名称]: 显示服务类型

在以下示例中，可以看到/turtle1/set_pen服务是turtlesim/SetPen类型。

```
$ rosservice type /turtle1/set_pen
turtlesim/SetPen
```

rosservice find [服务类型]: 查找指定服务类型的服务

以下示例搜索turtlesim/SetPen类型的服务。因此，可以看到搜索出/turtle1/set_pen。

```
$ rosservice find turtlesim/SetPen  
/turtle1/set_pen
```

rosservice uri [服务名称]: 显示ROSRPC uri服务

用户也可以使用rosservice的uri选项来检查/turtle1/set_pen服务的ROSRPC URI，如下所示：

```
$ rosservice uri /turtle1/set_pen  
rosrpc://192.168.1.100:50624
```

rosservice args [服务名称]: 服务参数输出

我们来看看/turtle1/set_pen服务的每个参数，如下例所示，该命令显示在/turtle1/set_pen服务中使用r、g、b、width和off参数。

```
$ rosservice args /turtle1/set_pen  
r g b width off
```

rosservice call [服务名称] [参数]: 用输入的参数服务请求

以下示例是请求/turtle1/set_pen服务的命令。所使用的“255 0 0 5 0”是对应于用于/turtle1/set_pen服务的参数(r, g, b, width, off)的值。红色的r的最大值是255，因为g和b都是0，所以笔的颜色是红色的。width设置为5，off为0（假）。rosservice call是一个非常有用命令，通常用于测试服务。

```
$ rosservice call /turtle1/set_pen 255 0 0 5 0
```

通过使用前面的命令，发送了服务请求，更改了turtlesim中使用的笔的属性，并且从turtle_teleop_key中下达了下移命令。作为结果，可以从下面图中看到原来是白色的笔色显示为红色。



图 5-3 rosservice call 示例

5.4.5. rosparam：ROS参数

由于读者需要了解这些参数，请参阅第4.1节“ROS术语”。

命令	详细说明
rosparam list	查看参数列表
rosparam get [参数名称]	获取参数值
rosparam set [参数名称]	设置参数值
rosparam dump [文件名称]	将参数保存到指定文件
rosparam load [文件名称]	获取保存在指定文件中的参数
rosparam delete [参数名称]	删除参数

在运行ROS参数相关示例之前先关闭所有节点。通过在不同的终端窗口中运行以下命令来运行turtlesim_node和turtle_teleop_key。

```
$ roscore  
$ rosrun turtlesim turtlesim_node  
$ rosrun turtlesim turtle_teleop_key
```

rosparam list: 查看参数列表

显示在同一网络中使用的参数列表。

```
$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_192_168_1_100__39536
/rosversion
/run_id
```

rosparam get [参数名称]: 获取参数值

如果要查看特定参数的值，可以将该参数名称指定为rosparam get命令的选项。

```
$ rosparam get /background_b
255
```

如果要检查所有参数的值，而不是某一特定的参数，可以使用“/”作为选项来显示所有参数的值，如下所示。

```
$ rosparam get /
background_b: 255
background_g: 86
background_r: 69
rosdistro: 'kinetic'
roslaunch:
    uris: {host_192_168_1_100__43517: 'http://192.168.1.100:43517/'}
    rosversion: '1.12.7'
run_id: c2d0b528-6536-11e7-935b-08d40c80c500
```

rosparam dump [文件名]: 将参数保存到指定的文件

以下示例将当前参数值保存到parameters.yaml文件中。因为它保存了每次使用的参数值，并且可以在下次执行时使用（“~/”表示用户的home目录），所以很方便。

```
$ rosparam dump ~/parameters.yaml
```

rosparam set [参数名称]: 设置参数值

这是设置参数值的命令。在以下示例中，将turtlesim节点的background_b参数（与背景色相关的参数）设置为0。

```
$ rosparam set background_b 0  
$ rosservice call clear
```

RGB从255,86,69变为0,86,69，所以变成深绿色，如图5-4右图所示。但是，由于turtlesim节点并不是每次读取参数，因此需要使用“rosparam set background_b 0”命令修改参数之后，用“rosservice call clear”命令刷新屏幕。参数的反映结果取决于节点的运行方式。



图 5-4 rosparam set 示例

rosparam load [文件名称]: 将参数保存到指定的文件

此命令与rosparam dump相反，是读取parameters.yaml文件并将其用作当前参数值。如下例所示，运行“rosservice call clear”命令，参数值将会变为所加载的文件的参数值，并且在执行dump命令时，图5-4中变为绿色的背景将变为蓝色背景。rosparam load是一个非常有用的命令，会经常使用，读者需要掌握。

```
$ rosparam load ~/parameters.yaml  
$ rosservice call clear
```

rosparam delete [参数名称]: 删除参数

该命令删除指定的参数。

```
$ rosparam delete /background_b
```

5.4.6. rosmsg: ROS消息信息

由于读者需要了解消息（message），所以我们参考第4.1节“ROS术语”。

命令	详细说明
rosmsg list	显示所有消息
rosmsg show [消息名称]	显示指定消息
rosmsg md5 [消息名称]	显示md5sum
rosmsg package [功能包名称]	显示用于指定功能包的所有消息
rosmsg packages	显示使用消息的所有功能包

在运行ROS消息信息示例之前，先关闭所有节点。通过在不同的终端窗口中运行以下命令来运行turtlesim_node和turtle_teleop_key。

```
$ roscore  
$ rosrun turtlesim turtlesim_node  
$ rosrun turtlesim turtle_teleop_key
```

rosmsg list: 显示所有消息

该命令显示当前ROS中安装的功能包的所有消息。根据当前ROS中包含的功能包，显示结果可能会有所不同。

```
$ rosmsg list  
actionlib/TestAction  
actionlib/TestActionFeedback  
actionlib/TestActionGoal  
actionlib/TestActionResult
```

```
actionlib/TestFeedback  
actionlib/TestGoal  
sensor_msgs/Joy  
sensor_msgs/JoyFeedback  
sensor_msgs/JoyFeedbackArray  
sensor_msgs/LaserEcho  
zeroconf_msgs/DiscoveredService  
~省略~
```

rosmsg show [消息名称]: 显示指定的消息信息

显示指定的消息信息。以下是显示turtlesim/Pose消息信息的例子。float32是一个浮点变量，可以确认它是一个包含5条信息（x、y、theta、linear_velocity和angular_velocity）的消息。

```
$ rosmsg show turtlesim/Pose  
float32 x  
float32 y  
float32 theta  
float32 linear_velocity  
float32 angular_velocity
```

rosmsg md5 [消息名称]: 显示md5sum

以下是查看turtlesim/Pose消息的md5信息的示例。有时如果在消息通信期间遇到MD5问题，则需要检查md5sum。这时会用到该命令，一般不常用。有关md5sum的解释，请参见第4.1节“ROS术语”。

```
$ rosmsg md5 turtlesim/Pose  
863b248d5016ca62ea2e895ae5265cf9
```

rosmsg package [功能包名称]: 显示用于指定功能包的所有消息

可以看到特定功能包中使用的消息。

```
$ rosmsg package turtlesim  
turtlesim/Color  
turtlesim/Pose
```

rosmsg packages: 显示使用消息的所有功能包

```
$ rosmsg packages
actionlib
actionlib_msgs
actionlib_tutorials
base_local_planner
bond
control_msgs
costmap_2d
~省略~
```

5.4.7. rossrv: ROS服务信息

因为读者需要了解服务，所以请参考4.1节“ROS术语”中的服务（service）。

命令	详细说明
rossrv list	显示所有服务
rossrv show [服务名称]	显示指定的服务信息
rossrv md5 [服务名称]	显示md5sum
rossrv package [功能包名称]	显示指定的功能包中用到的所有服务
rossrv packages	显示使用服务的所有功能包

在运行ROS服务信息相关示例之前，先关闭所有节点。通过在不同的终端窗口中运行以下命令来运行turtlesim_node和turtle_teleop_key。

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

rossrv list: 显示所有服务

该命令显示了ROS上当前安装的功能包的所有服务。根据目前包含在ROS中的功能包，显示结果可能会有所不同。

```
$ rossrv list
control_msgs/QueryCalibrationState
control_msgs/QueryTrajectoryState
diagnostic_msgs/SelfTest
dynamic_reconfigure/Reconfigure
gazebo_msgs/ApplyBodyWrench
gazebo_msgs/ApplyJointEffort
gazebo_msgs/BodyRequest
gazebo_msgs/DeleteModel
~省略~
```

rossrv show [服务名称]: 显示指定服务的信息

以下示例显示turtlesim/SetPen服务信息。可以确认uint8是包含r、g、b、width和off等5种信息的服务。请注意，“---”在服务文件中用作请求和响应的分隔符。对于turtlesim/SetPen，用户可以看到只有请求，没有任何响应。对于服务文件，参见4.3节的说明和7.3节的例子。

```
$ rossrv show turtlesim/SetPen
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

rossrv md5 [服务名称]: 显示md5sum

在以下示例中，使用该命令来查看turtlesim/SetPen服务的md5信息。有时如果在服务请求和响应期间遇到MD5问题，则需要检查md5sum。这时会用到该命令，一般不常用。

```
$ rossrv md5 turtlesim/SetPen
9f452acce566bf0c0954594f69a8e41b
```

`rossrv package [功能包名称]`: 显示用于指定功能包的所有服务

可以看到指定功能包中用到的服务。

```
$ rossrv package turtlesim
turtlesim/Kill
turtlesim/SetPen
turtlesim/Spawn
turtlesim/TeleportAbsolute
turtlesim/TeleportRelative
```

`rossrv packages`: 显示使用服务的所有功能包

```
$ rossrv packages
control_msgs
diagnostic_msgs
dynamic_reconfigure
gazebo_msgs
map_msgs
nav_msgs
navfn nodelet
oroca_ros_tutorials
roscpp
sensor_msgs
std_srvs
tf
tf2_msgs
turtlesim
~省略~
```

5.4.8. rosbag: ROS日志信息

如第4.1节“ROS术语”中所述，在ROS中用bag格式保存各种消息，并在需要时将其回放，以便我们可以重现以前的情况。rosbag是一个实现生成、播放和压缩等功能的程序，它具有以下几种功能。

命令	详细说明
rosbag record [选项] [话题名称]	将指定话题的消息记录到bag文件
rosbag info [文件名称]	查看bag文件的信息
rosbag play [文件名称]	回放指定的bag文件
rosbag compress [文件名称]	压缩指定的bag文件
rosbag decompress [文件名称]	解压指定的bag文件
rosbag filter [输入文件] [输出文件] [选项]	生成一个删除了指定内容的新的bag文件
rosbag reindex bag [文件名称]	刷新索引
rosbag check bag [文件名称]	检查指定的bag文件是否能在当前系统中回放
rosbag fix [输入文件] [输出文件] [选项]	将由于版本不同而无法回放的bag文件修改成可以回放的文件

在运行ROS日志信息例子之前，先关闭所有节点。通过在不同的终端窗口中运行以下命令来运行turtlesim_node和turtle_teleop_key。

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

rosbag record [选项][话题名称]: 记录指定话题的消息

首先，使用rostopic list命令查看ROS网络上当前正在使用的话题列表。

```
$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

如以下示例所示，把要进行记录的话题作为record命令的选项来输入，则会开始记录于bag文件中。在开始记录之后，在运行turtle_teleop_key节点的终端窗口中用键盘的方向键移动乌龟，则会记录选定的/turtle1/cmd_vel话题。然后按[Ctrl + c]结束记录，则会生成一个文件名为“2017-07-10-14-16-28.bag”的bag文件，如下所示。

```
$ rosbag record /turtle1/cmd_vel
[INFO] [1499663788.499650818]: Subscribing to /turtle1/cmd_vel
[INFO] [1499663788.502937962]: Recording to 2017-07-10-14-16-28.bag.
```

如果要同时记录所有话题，而不是特定话题，请在命令中添加“-a”选项。

```
$ rosbag record -a  
[WARN] [1499664121.243116836]: --max-splits is ignored without --split  
[INFO] [1499664121.248582681]: Recording to 2017-07-10-14-22-01.bag.  
[INFO] [1499664121.248879947]: Subscribing to /turtle1/color_sensor  
[INFO] [1499664121.252689657]: Subscribing to /rosout  
[INFO] [1499664121.257219911]: Subscribing to /rosout_agg  
[INFO] [1499664121.260671283]: Subscribing to /turtle1/pose
```

rosbag info [bag文件名]: 查看bag文件的信息

用户可以检查bag文件的信息。以下示例记录了/turtle1/cmd_vel话题，共记录了373条消息。使用的消息类型是geometry_msgs/Twist。此外，还可以检查路径，bag版本和时间等信息。

```
$ rosbag info 2017-07-10-14-16-28.bag  
path: 2017-07-10-14-16-28.bag  
version: 2.0  
duration: 17.4s  
start: Jul 10 2017 14:16:30.36 (1499663790.36)  
end: Jul 10 2017 14:16:47.78 (1499663807.78)  
size: 44.5 KB  
messages: 373  
compression: none [1/1 chunks]  
types: geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]  
topics: /turtle1/cmd_vel 373 msgs :geometry_msgs/Twist
```

rosbag play [bag文件名]: 回放指定的bag文件

下面的例子是一个回放之前记录的2017-07-10-14-16-28.bag文件的命令。如此一来，当时记录的/turtle1/cmd_vel消息会原原本本地传输，因此在屏幕上可以看到乌龟移动的情况。但是，只有重新执行turtlesim_node，使得优先初始化机器人轨迹和机器人位置，才可以获得与图5-5中相同的结果。

```
$ rosbag play 2017-07-10-14-16-28.bag  
[INFO] [1499664453.406867251]: Opening 2017-07-10-14-16-28.bag
```

```
Waiting 0.2 seconds after advertising topics... done.  
Hit space to toggle paused, or 's' to step.  
[RUNNING] Bag Time: 1499663790.357031 Duration: 0.000000 / 17.419737  
[RUNNING] Bag Time: 1499663790.357031 Duration: 0.000000 / 17.419737  
[RUNNING] Bag Time: 1499663790.357163 Duration: 0.000132 / 17.419737  
~省略~
```

如下图所示，可以看到回放的数据与原始数据相同。

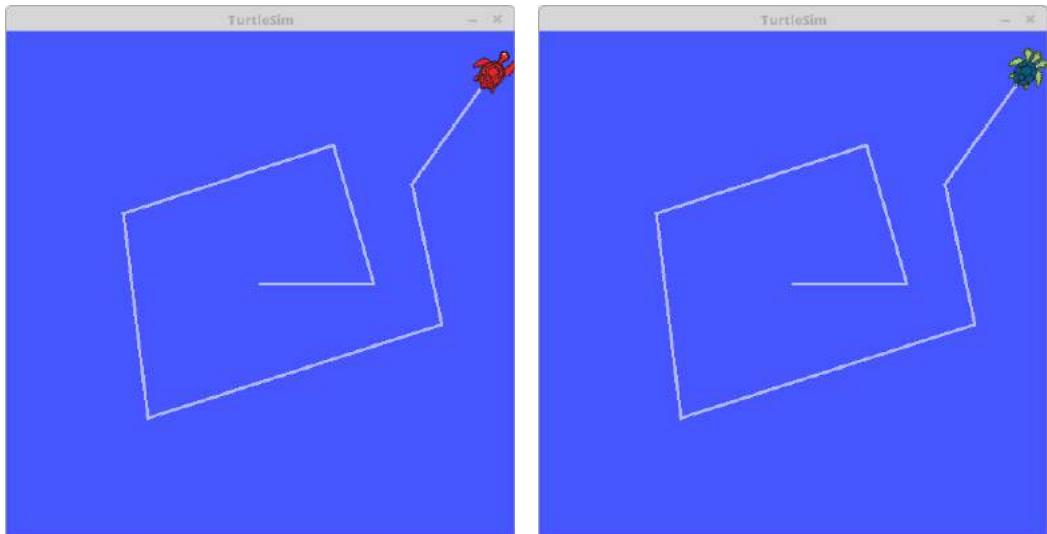


图 5-5 rosbag play 示例

rosbag compress [bag文件名]: 压缩指定的bag文件

短时间记录的bag文件因为不是很大，所以不成问题，但是长时间记录数据时，bag文件会占用大量的硬盘存储空间。如果使用本例中使用的命令对其进行压缩，则将占用很小的存储空间。

```
$ rosbag compress 2017-07-10-14-16-28.bag  
2017-07-10-14-16-28.bag 0% 0.0 KB 00:00  
2017-07-10-14-16-28.bag 100% 35.0 KB 00:00
```

如下所示，前面的例子中的bag文件被缩减为1/4。压缩前的原始文件会以文件名添加“orig”的文件另行存储。

```
2017-07-10-14-16-28.bag 12.7kB  
2017-07-10-14-16-28.orig.bag 45.5kB
```

rosbag decompress [bag文件名]: 对指定的bag文件解压

要解压缩，则需使用如下命令。这个命令会将文件恢复到压缩之前的状态。

```
$ rosbag decompress 2017-07-10-14-16-28.bag  
2017-07-10-14-16-28.bag 0% 0.0 KB 00:00  
2017-07-10-14-16-28.bag 100% 35.0 KB 00:00
```

5.5. ROS catkin命令

ROS的catkin命令用于使用catkin 构建系统来构建功能包。

命令	重要度	详细说明
catkin_create_pkg	★★★	自动生成功能包
catkin_make	★★★	基于catkin 构建系统的构建
catkin_eclipse	★★☆	把用catkin 构建系统生成的功能包修改，使得可以在Eclipse 环境中使用
catkin_prepare_release	★★☆	发布时用到的日志整理和版本标记
catkin_generate_changelog	★★☆	发布时生成或更新CHANGELOG.rst文件
catkin_init_workspace	★★☆	初始化catkin 构建系统的工作目录
catkin_find	★☆☆	搜索catkin

catkin_create_pkg: 自动生成功能包

```
catkin_create_pkg [功能包名称] [依赖性功能包1] [依赖性功能包2] ...
```

catkin_create_pkg是创建一个包含CMakeLists.txt和package.xml文件的空功能包的命令。有关ROS的构建系统的更多信息，请参考第4.9节。以下例子显示了使用catkin_create_pkg命令创建一个依赖于roscpp和std_msgs的my_package功能包。

```
$ catkin_create_pkg my_package roscpp std_msgs
```

catkin_make：基于catkin 构建系统的构建

```
catkin_make [选项]
```

catkin_make是构建用户创建的功能包或构建下载的功能包的命令。以下示例是构建~/catkin_ws/src目录中所有功能包的示例。

```
$ cd ~/catkin_ws  
$ catkin_make
```

如果要只构建一部分功能包，而不是全部功能包，请使用“--pkg [包名]”选项来运行，如下所示：

```
$ catkin_make --pkg user_ros_tutorials
```

catkin_eclipse：将以catkin构建系统生成的功能包修改成可以在Eclipse环境中使用的功能包

Eclipse是集成开发环境（IDE）之一，而catkin_eclipse命令用于构建一个可以使Eclipse来管理和编程功能包的环境。这将为Eclipse创建一个~/catkin_ws/build/.cproject和~/catkin_ws/build/.project等项目文件。您可以通过从Eclipse菜单中选择[Makefile Project with Existing Code]并选择~/catkin_ws/build/来管理Eclipse中~/catkin_ws/src中的所有功能包。

```
$ cd ~/catkin_ws  
$ catkin_eclipse
```

catkin_generate_changelog：生成CHANGELOG.rst文件

catkin_generate_changelog命令在更新功能包的版本时创建一个描述更新记录的CHANGE LOG.rst文件。

catkin_prepare_release：准备发布时用到的更新记录和版本标记

catkin_prepare_release是用于更新由catkin_generate_changelog命令生成的CHANGELOG.rst文件的命令。在把创建的功能包注册到官方ROS存储库或更新功能包的版本时，会使用catkin_generate_changelog和catkin_prepare_release命令。

catkin_init_workspace: 初始化catkin构建系统的工作目录

`catkin_init_workspace`是初始化用户工作目录 (`~/catkin_ws/src`) 的命令。如3.1节所述，除了特殊情况外，这个命令在ROS安装期间只执行一次。

```
$ cd ~/catkin_ws/src  
$ catkin_init_workspace
```

catkin_find: 搜索catkin，找到并显示工作空间

`catkin_find`是一个显示各项目的工作目录的命令。

```
catkin_find [功能包名称]
```

用户可以通过运行`catkin_find`命令来找出正在使用的所有工作目录。此外，如果执行“`catkin_find [功能包名称]`”，则会看到选项中指定的与功能包相关的工作目录，如下所示。

```
$ catkin_find  
/home/pyo/catkin_ws/devel/include  
/home/pyo/catkin_ws/devel/lib  
/home/pyo/catkin_ws/devel/share  
/opt/ros/kinetic/bin  
/opt/ros/kinetic/etc  
/opt/ros/kinetic/include  
/opt/ros/kinetic/lib  
/opt/ros/kinetic/share
```

```
$ catkin_find turtlesim  
/opt/ros/kinetic/include/turtlesim  
/opt/ros/kinetic/lib/turtlesim  
/opt/ros/kinetic/share/turtlesim
```

5.6. ROS功能包命令

ROS功能包命令用于操作ROS功能包，比如显示功能包信息、安装相关功能包，等。

命令	重要度	命令释义	详细说明
rospack	★★★	ros+pack(age)	显示与ROS功能包相关的信息
rosinstall	★★☆	ros+install	安装ROS附加功能包
rosdep	★★☆	ros+dep(endencies)	安装该功能包的依赖性文件
roslocate	☆☆☆	ros+locate	与ROS功能包信息有关的命令
roscreate-pkg	☆☆☆	ros+create-pkg	自动生成ROS功能包（用于旧的rosbuild系统）
rosmake	☆☆☆	ros+make	构建ROS功能包（用于旧的rosbuild系统）

rospack：显示指定的ROS功能包的相关信息

```
rospack [选项] [功能包名称]
```

rospack是一个命令，用于显示与指定的ROS功能包相关的信息，如存储位置、依赖关系和整个功能包列表。可以使用find、list、depends-on、depends和profile等选项。如果在rospack查找命令之后指定了功能包名称，会显示该功能包的存储位置，如以下示例所示。

```
$ rospack find turtlesim  
/opt/ros/kinetic/share/turtlesim
```

rospack list命令显示PC上的所有功能包。用户可以结合rospack list命令与Linux搜索命令grep来轻松找到该功能包。例如，“rospack list | grep turtle”将显示所有功能包中只与turtle相关的功能包。

```
$ rospack list  
actionlib /opt/ros/kinetic/share/actionlib  
actionlib_msgs /opt/ros/kinetic/share/actionlib_msgs  
actionlib_tutorials /opt/ros/kinetic/share/actionlib_tutorials  
amcl /opt/ros/kinetic/share/amcl  
angles /opt/ros/kinetic/share/angles  
base_local_planner /opt/ros/kinetic/share/base_local_planner  
bfl /opt/ros/kinetic/share/bfl
```

```
$ rospack list | grep turtle
turtle_actionlib /opt/ros/kinetic/share/turtle_actionlib
turtle_tf /opt/ros/kinetic/share/turtle_tf
turtle_tf2 /opt/ros/kinetic/share/turtle_tf2
turtlesim /opt/ros/kinetic/share/turtlesim
```

如果在rospack depends-on命令之后指定了一个功能包名称，则仅显示使用指定功能包的功能包列表，如以下示例所示。

```
$ rospack depends-on turtlesim
turtle_tf2
turtle_tf
turtle_actionlib
```

如果在rospack depends命令之后指定了功能包名称，则会看到运行该功能包所需的依赖性功能包的列表，如以下示例所示。

```
$ rospack depends turtlesim
cpp_common
rostime
roscpp_traits
roscpp_serialization
genmsg
genpy
message_runtime
std_msgs
geometry_msgs
catkin
gencpp
genlisp
message_generation
rosbuild
rosconsole
rosgraph_msgs
xmlrpcpp
roscpp
rospack
roslib
std_srvs
```

`rospack profile`命令通过检查存储功能包的工作目录（例如“/opt/ros/kinetic/share”或“~/catkin_ws/src”）和功能包的信息来重建功能包索引。当新添加的功能包使用“`rosdep`”等时，在列表中没有显示时可以使用该命令来更新索引。

```
$ rospack profile
Full tree crawl took 0.021790 seconds.
Directories marked with (*) contain no manifest. You may
want to delete these directories.
To get just of list of directories without manifests,
re-run the profile with --zombie-only
-----
0.020444 /opt/ros/kinetic/share
0.000676 /home/pyo/catkin_ws/src
0.000606 /home/pyo/catkin_ws/src/ros_tutorials
0.000240 * /opt/ros/kinetic/share/OpenCV-3.2.0-dev
0.000054 * /opt/ros/kinetic/share/OpenCV-3.2.0-dev/haarcascades
0.000035 * /opt/ros/kinetic/share/doc
0.000020 * /opt/ros/kinetic/share/OpenCV-3.2.0-dev/lbpcascades
0.000008 * /opt/ros/kinetic/share/doc/liborocos-kdl
```

rosinstall: 安装ROS附加功能包

`rosinstall`是一个自动安装或更新由源代码管理软件（SCM，如SVN、Mercurial、Git和Bazaar）管理的ROS包的命令。一旦像3.1节那样运行它一次，之后当功能包有更新时，会自动安装需要的功能包或更新。

rosdep: 安装该功能包的依赖性文件

```
rosdep [选项]
```

`rosdep`是安装指定功能包的依赖性文件的命令。选项包括`check`、`install`、`init`和`update`。如下例所示，执行“`rosdep check [功能包名]`”，会检查指定功能包的依赖关系。执行“`rosdep install package name`”，将安装指定功能包的依赖功能包。还有“`rosdep init`”和“`rosdep update`”，但请参阅第3.1节的实际用法。

```
$ rosdep check turtlesim  
All system dependencies have been satisfied  
$ rosdep install turtlesim  
All required rosdeps installed successfully
```

roslocate：显示ROS功能包的信息

```
roslocate [选项] [功能包名称]
```

roslocate是显示功能包相关信息的命令，例如功能包正在使用的ROS的版本、SCM类型和存储库位置等。可用的选项是info、vcs、type、uri和repo等。在这里，我们来看看一次显示所有这些信息的info。

```
$ roslocate info turtlesim  
Using ROS_DISTRO: kinetic  
- git:  
local-name: turtlesim  
uri: https://github.com/ros/ros_tutorials.git  
version: kinetic-devel
```

roscreate-pkg：自动生成ROS功能包（用于旧的rosbuild系统）

roscreate-pkg是一个像catkin_create_pkg命令一样自动创建一个功能包的命令。该命令是在catkin构建系统之前的旧的rosbuild系统中使用的命令。它被保留仅仅是为了版本兼容性，现在基本不会用到。

rosmake：构建ROS功能包（用于旧的rosbuild系统）

rosmake是一个像catkin_make命令一样构建功能包的命令。该命令是在catkin构建系统之前的旧的rosbuild系统中使用的命令。它被保留仅仅是为了版本兼容性，现在基本不会用到。

第6章

ROS工具

除了第5章介绍的输入型命令外，还有各种其他工具可以帮助用户使用ROS。应该指出，这些GUI工具是对输入型命令工具的补充。如果包括ROS用户个人发布的工具，那么ROS工具的数量很庞大。其中，本章讨论的工具是对于ROS编程非常有用的辅助工具。

本章涵盖的工具是：

- RViz 三维可视化工具
- rqt 基于Qt的ROS GUI开发工具
- rqt_image_view 图像显示工具（rqt的一类）
- rqt_graph 以图形显示节点和消息之间的相关关系的工具（rqt的一类）
- rqt_plot 二维数据绘图工具（rqt的一类）
- rqt_bag 基于GUI的bag数据分析工具（rqt的一类）

6.1. 三维可视化工具（RViz）

RViz¹是ROS的三维可视化工具。它的主要目的是以三维方式显示ROS消息，可以将数据进行可视化表达。例如，可以无需编程就能表达激光测距仪（LRF）传感器中的传感器到障碍物的距离，RealSense、Kinect或Xtion等三维距离传感器的点云数据（PCD，Point Cloud Data），从相机获取的图像值等。

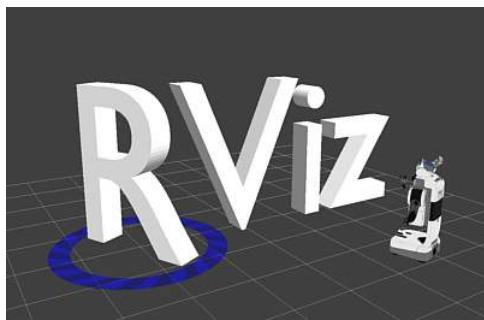


图 6-1 ROS三维可视化工具RViz的启动画面

¹ <http://wiki.ros.org/rviz>

另外，利用用户指定的多边形（polygon）支持各种表现形式，交互标记（Interactive Markers）²可以表达接收来自用户节点的命令和数据并互交的过程。在ROS中，机器人以URDF（Unified Robot Description Format，统一机器人描述格式）³描述，它可以表示为三维模型，并且每个模型可以根据自由度进行移动或驱动，因此可以用于仿真或控制。例如，如图6-2所示，可以显示移动机器人模型，同时可以通过接收来自LDS（激光距离传感器）的距离值，并用于导航。而且，安装在机器人上的照相机的图像可以如左下方那样显示。另外，如图6-3、6-4和6-5所示，可以从Kinect、LRF和RealSense等各种传感器获取数据，并以三维图像显示。

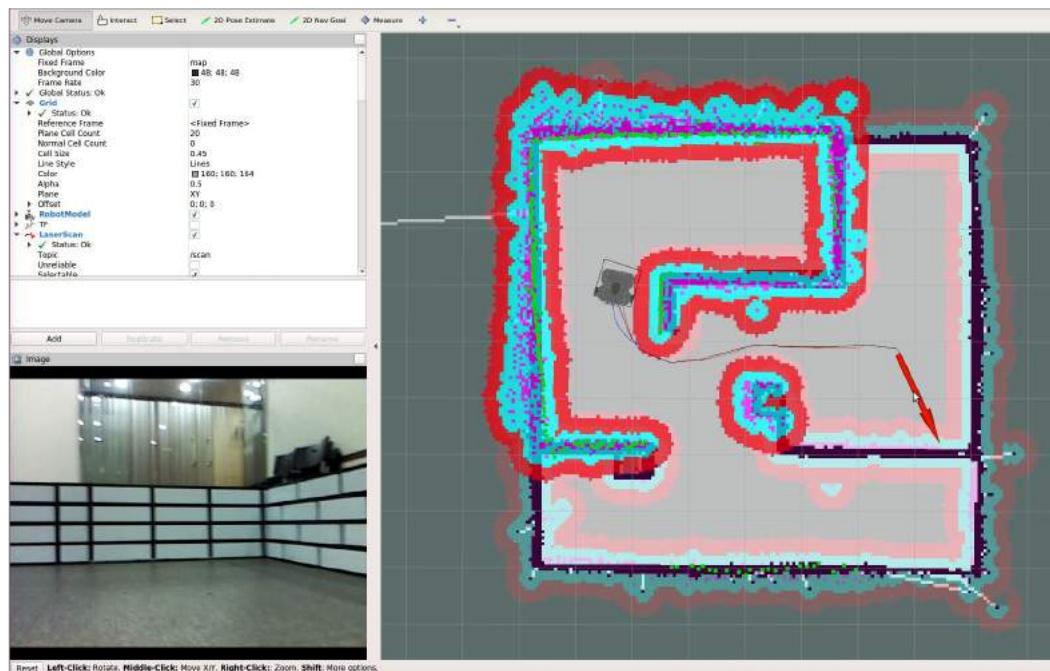


图 6-2 RViz用例一：使用TurtleBot3与LDS传感器的导航

² <http://wiki.ros.org/rviz/Tutorials/Interactive%20Markers%3A%20Getting%20Started>

³ <http://wiki.ros.org/urdf>

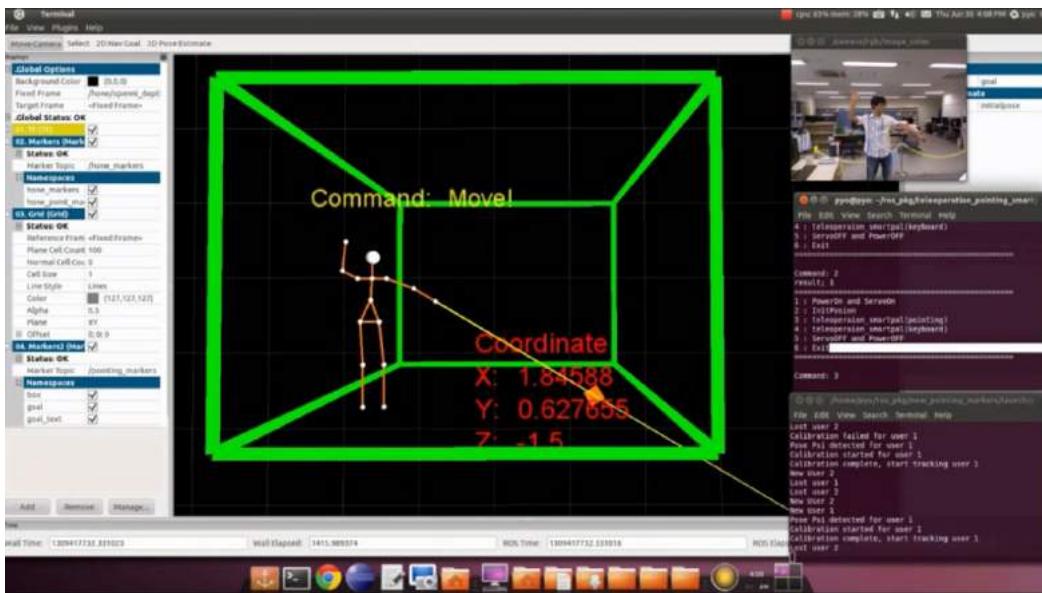


图 6-3 RViz用例二：从Kinect获取人的骨骼后控制机器人的画面

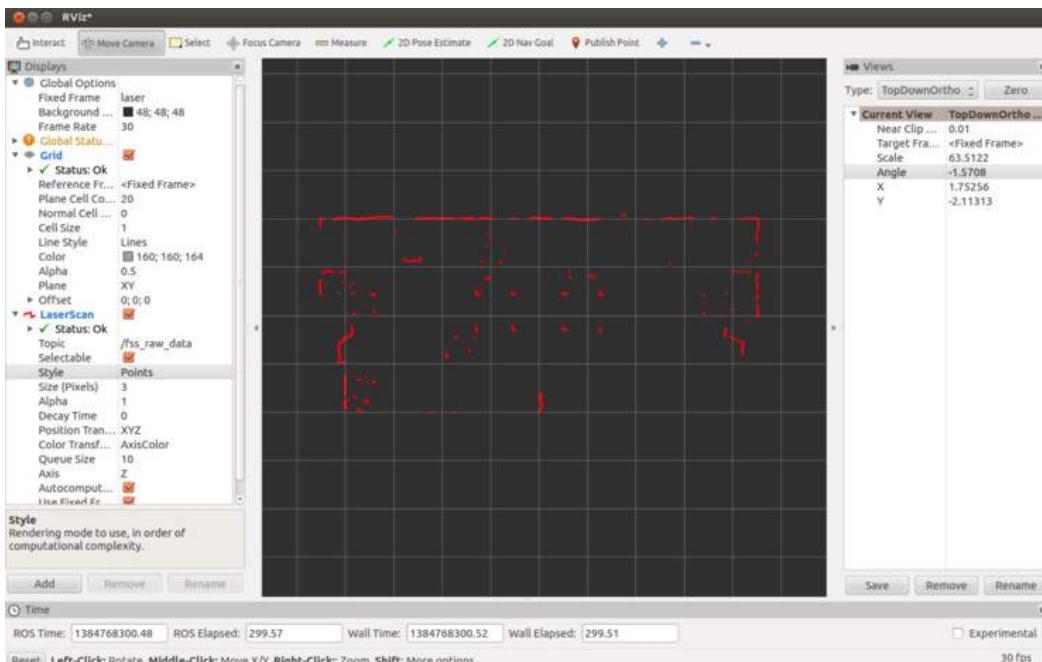


图 6-4 RViz用例三：利用LDS测距

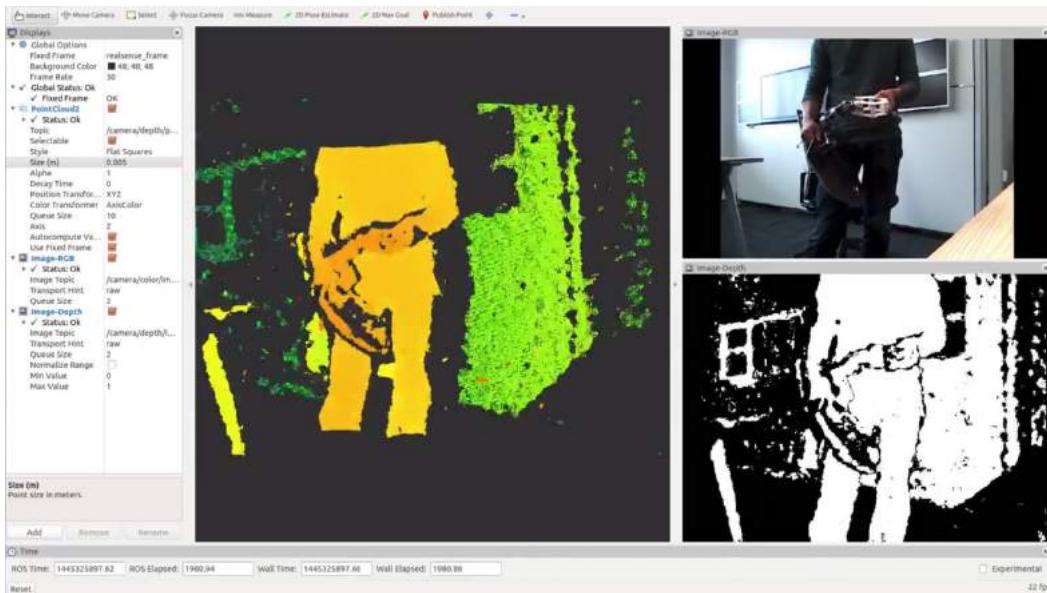


图 6-5 RViz用例四：从Intel RealSense获取的距离、红外线数据、颜色图像值

6.1.1. RViz安装与运行

以“ros- [ROS_DISTRO] -desktop-full”命令安装ROS时，RViz会默认被安装。如果未安装“desktop-full”或未安装RViz，请使用以下命令进行安装。

```
$ sudo apt-get install ros-kinetic-rviz
```

RViz的运行命令如下。就像任何其他的ROS工具一样，roscore必须运行。作为参考，您也可以使用节点运行命令“rosrun rviz rviz”运行它。

```
$ rviz
```

6.1.2. RViz画面布局

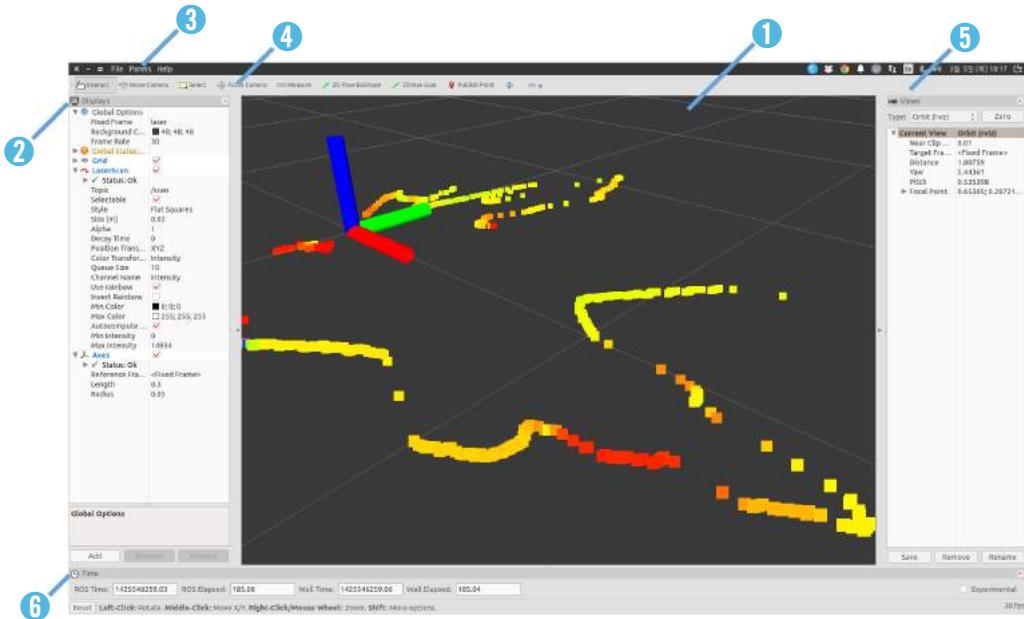


图 6-6 RViz画面布局

- ① 3D视图（3D view）：指屏幕的黑色部分。它是可以用三维方式查看各种数据的主屏幕。3D视图的背景颜色、固定框架、网格等可以在左侧显示的全局选项（Global Options）和网格（Grid）项目中进行详细设置。
- ② 显示屏（Displays）：左侧的显示屏是从各种话题当中选择用户所需的数据的视图的区域。如果单击屏幕左下方的[Add]，选择屏幕将如图6-7所示。目前有大约30种不同的显示屏可供选择，我们将在下面的描述中详细介绍。
- ③ 菜单（Menu）：菜单位于顶部。用户可以选择保存或读取显示屏状态的命令，还可以选择各种面板。
- ④ 工具（Tools）：工具是位于菜单下方的按钮，允许用户用各种功能按键选择多种功能的工具，例如Interact、Move Camera、Select，Focus Camera、Measure、2D Pose Estimate、2D Navigation Goal以及Publish Point等。
- ⑤ 视图（Views）：设定三维视图的视点

4 <http://wiki.ros.org/rviz/DisplayTypes>

- Orbit: 以指定的视点（在这里称为Focus）为中心旋转。这是默认情况下最常用的基本视图。
 - FPS（第一人称）：显示第一人称视点所看到的画面。
 - ThirdPersonFollower: 显示以第三人称的视点尾追特定目标的视图。
 - TopDownOrtho: 这是Z轴的视图，与其他视图不同，以直射视图显示，而非透视法。
 - XYOrbit: 类似于Orbit的默认值，但焦点固定在Z轴值为0的XY平面上。
- ⑥ 时间 (Time) : 显示当前时刻 (wall time) 、ROS Time以及他们各自经过的时间。这主要用于仿真，如果需要重新启动，请点击底部的[Reset]按钮。

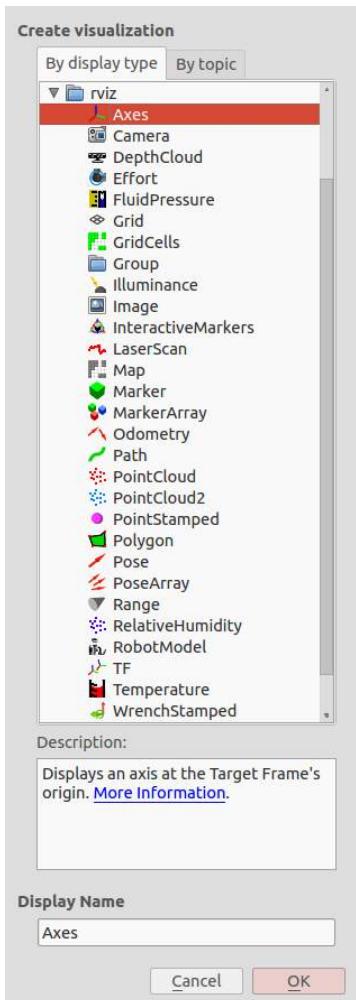


图 6-7 RViz显示屏的选择画面

6.1.3. RViz显示屏

使用RViz的过程中最常用的菜单应该是显示屏⁵菜单。该显示屏菜单用于选择三维视图（3D View）画面所显示的信息，各项目的说明请参照表6-1。

图标	名称	说明
	Axes	显示x、y、z轴。
	Camera	从相机的角度创建一个新的渲染窗口，并将图像覆盖在窗口上。
	DepthCloud	显示基于Depth map的点云。将从相机获取的彩色图像覆盖在“点”上面，这里的“点”是通常基于DepthMap和ColorImage话题的传感器（如Kinect和Xtion）的距离值。
	Effort	显示施加在机器人的每个旋转关节的力。
	FluidPressure	显示流体的压力，如空气或水。
	Grid	显示二维或三维网格。
	Grid Cells	显示网格的每个单元格。主要用于显示导航的costmap中的障碍物。
	Group	这是一个对显示屏进行分组的容器。要使用的显示屏可以作为一个组进行管理。
	Illuminance	指示亮度。
	Image	在新的渲染窗口中显示图像。与camera显示屏不同，它不覆盖相机。
	InteractiveMarkers	显示一个或多个Interactive Marker。可以使用鼠标更改位置（x, y, z）和姿态（roll, pitch, yaw）。
	LaserScan	显示激光扫描值。
	Map	将导航中使用的占用地图（occupancy map）显示在地平面（ground plane）上。
	Marker	显示RViz提供的箭头、圆圈、三角形、矩形和圆柱等标志。
	MarkerArray	显示多个标记显示屏。
	Odometry	将按照时间推移的测位（odometry）信息以箭头的形式显示。例如，将随着机器人的移动所产生的连续路径（机器人的位置和方向），沿着时间间隔以箭头的形式显示。
	Path	显示导航中使用的机器人的路径。

⁵ <http://wiki.ros.org/rviz/DisplayTypes>

图标	名称	说明
	Point Cloud	显示点云 (point cloud) 数据。它用于表示Depth camera系列的传感器数据，如RealSense、Kinect和Xtion等。由于PointCloud和PointCloud2是有区别的。由于PointCloud2是遵循最新的PCL (Point Cloud Library) 中使用的格式，因此一般情况下使用PointCloud2即可。
	Point Cloud2	
	PointStamped	显示圆形点。
	Polygon	显示一个线形式的多边形轮廓。主要用于在二维平面上简单显示机器人的轮廓。
	Pose	显示三维姿态 (pose, 位置+方向)。姿态是一个箭头，箭头的起点是位置 (x, y, z)，箭头的方向是机器人的方向 (roll, pitch, yaw)。例如，机器人模型的位置和方向可以用姿态表示，而导航中可以用目标 (goal) 点表示。
	Pose Array	显示多个姿态。
	Range	它是圆锥喇叭形状，用来可视化超声波或红外传感器的测量范围。
	RelativeHumidity	显示相对湿度。
	RobotModel	显示机器人模型。
	TF	显示ROS中使用的坐标转换TF。显示方法与上述轴 (Axes) 一样用xyz轴表示，各轴根据相对坐标用箭头表现分层结构。
	Temperature	显示温度。
	WrenchStamped	用“箭头 (力)” 和 “箭头+圆形 (转矩)” 表示扭转 (wrench) 的动作。

表 6-1 Rviz显示屏

6.2. ROS GUI开发工具 (rqt)

除了三维可视化工具RViz之外，ROS还为机器人开发提供各种GUI工具。例如，有一个将每个节点的层次结构显示为图形，且显示当前节点和话题状态的graph；将消息显示为二维图形的plot，等。从ROS Fuerte版本开始，这些GUI开发工具被称为rqt⁶，它集成了30多种工具，可以作为一个综合的GUI工具来使用。另外，RViz也被集成到rqt的插件中，这使rqt成为ROS的一个不可缺少的GUI工具。

6 <http://wiki.ros.org/rqt>

另外，顾名思义，rqt是基于Qt开发的，而Qt是一个广泛用于计算机编程的GUI编程的跨平台框架，用户可以方便自由地添加和开发插件。本节介绍rqt插件中的rqt_image_view、rqt_graph、rqt_plot和rqt_bag。

6.2.1. rqt安装与运行

以“ros-[ROS_DISTRO]-desktop-full”命令安装ROS时，rqt会默认安装。如果未安装“desktop-full”或未安装rqt，请使用以下命令进行安装。

```
$ sudo apt-get install ros-kinetic-rqt*
```

运行rqt的命令如下。只需键入rqt。作为参考，用户可以使用节点执行命令“rosrun rqt_gui rqt_gui”执行它。

```
$ rqt
```

运行rqt将显示rqt的GUI界面，如图6-8所示。如果是第一次，它将只显示菜单，此外没有任何内容。这是因为还没有指定rqt直接运行的插件程序。

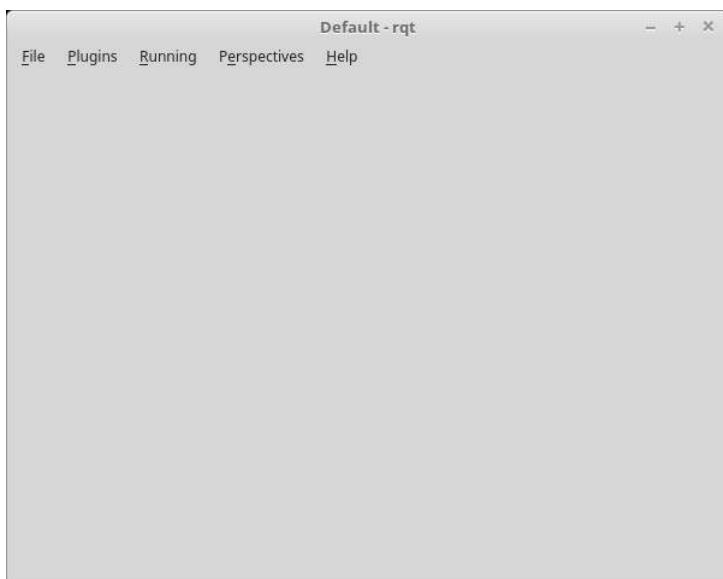


图 6-8 rqt初始画面

rqt的各菜单如下。

- 文件 (File) 只有一个退出rqt的子菜单。
- 插件 (Plugins) 有30多个插件。可以选择并使用它。
- 动作 (Running) 显示当前运行的插件，在不需要的时候可以停止。
- 全景 (Perspectives) 用于保存当前运行的插件组，并在下次运行相同的插件组。

6.2.2. rqt插件

如果从rqt的顶部菜单中选择[插件 (Plugins)]⁷⁸，则可以看到大约30个插件。该插件具有以下功能。大部分是非常有用的rqt的默认插件。非官方的插件也可以添加到此，需要的话用户也可以添加自己开发的rqt插件。

动作 (Action)

- Action Type Browser: 查看动作类型的数据结构的插件。

配置 (Configuration)

- Dynamic Reconfigure: 这是用于更改节点参数值的插件。
- Launch: roslaunch的GUI插件，当不记得roslaunch的名称或配置时，它非常有用。

自检 (Introspection)

- Node Graph: 一种图形视图类型的插件，可以检查当前运行中的节点间的关系图与消息的流动。
- Package Graph: 这是一个图形视图插件，显示功能包的依赖关系。
- Process Monitor: 可以检查当前正在运行的节点的PID（进程ID）、CPU利用率、内存使用情况和线程数。

⁷ <http://wiki.ros.org/rqt/Plugins>

⁸ http://wiki.ros.org/rqt_common_plugins

日志 (Logging)

- Bag: 是一个与ROS数据记录相关的插件。
- Console: 它是一个允许用户在一个屏幕中查看来自节点的警告 (Warning) 和错误 (Error) 等消息的插件。
- Logger Level: 通过选择负责发布日志的记录器节点来设置Debug、Info、Warn、Error和 Fatal等日志信息 (称为记录器级别⁹) 的工具。调试时选择Debug会非常方便。

多种工具 (Miscellaneous Tools)

- Python Console: Python控制台屏幕插件。
- Shell: 它是一个运行shell的插件。
- Web: 运行Web浏览器的插件。

机器人工具 (Robot Tools)

- Controller Manager: 这是一个允许用户检查机器人控制器的状态、类型和硬件接口信息插件。
- Diagnostic Viewer: 这是一个检查机器人设备和错误的插件。
- Moveit! Monitor: 用于查看运动规划的MoveIt!数据的插件。
- Robot Steering: 手动控制机器人的GUI工具。在远程控制时，利用此GUI工具进行机器人遥控会非常有用。
- Runtime Monitor: 它是一个可以实时查看节点中发生的警告或错误的插件，。

服务 (Services)

- Service Caller: 它是一个GUI插件，可以连接到正在运行中的服务服务器，并请求服务。这对测试服务 (Service) 很有用。
- Service Type Browser: 这是一个用于检查服务类型的数据结构的插件。

⁹ <http://wiki.ros.org/roscpp/Overview/Logging>

话题 (Topics)

- Easy Message Publisher: 这是一个允许用户在GUI环境中发布话题的插件，。
- Topic Publisher: 这是一个可以发布话题的GUI插件。这对话题测试很有用。
- Topic Type Browser: 这是检查话题类型的数据结构的插件。这对于检查话题类型很有用。
- Topic Monitor: 这是一个列出当前正在使用的话题，并确认用户选择的话题信息的插件。

可视化 (Visualization)

- Image View: 这是一个可以检查相机的图像数据的插件。这对于简单的照相机数据测试非常有用。
- Navigation Viewer: 这是一个用于在导航中检查机器人的位置和目标点的插件。
- Plot: 这是一个绘图二维数据的GUI 插件。这对于二维数据绘图非常有用。
- Pose View: 它是一个显示机器人模型和TF的姿态 (pose, 位置和方向) 的插件。
- RViz: 这是RViz插件，是一个3D可视化工具插件。
- TF Tree: 这是一个图形视图插件，它用树形图显示了通过TF收集的每个坐标之间的关系。

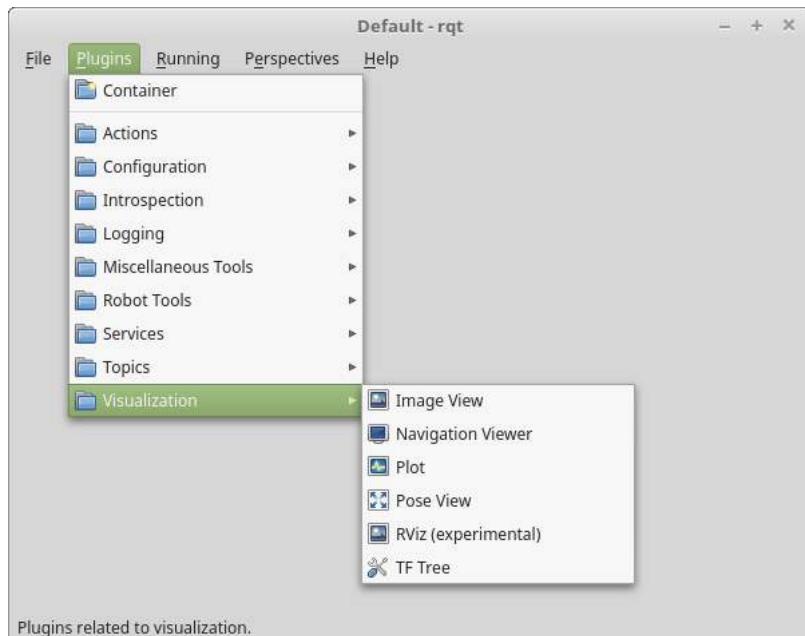


图 6-9 rqt插件

很难说明所有的插件。在本章中，我们将了解一下最常用的rqt_image_view、rqt_bag、rqt_graph和rqt_plot插件。

6.2.3. rqt_image_view

这是一个显示相机的图像数据的插件¹⁰。这不是一个图像处理过程，它只是在简单地查看图像时非常有用。一般的USB摄像头支持UVC，所以用户可以使用ROS的uvc_camera功能包。首先，使用以下命令安装uvc_camera功能包。

```
$ sudo apt-get install ros-kinetic-uvc-camera
```

将USB摄像头连接到计算机的USB接口，然后使用以下命令运行uvc_camera功能包中的uvc_camera_node节点。

```
$ rosrun uvc_camera uvc_camera_node
```

然后用“rqt”命令运行rqt，之后从菜单中选择[Plugins]→[Image View]。如果在左上方的消息选择下拉列表中选择“/image_raw”，则可以看到如图6-10所示的图像。相机传感器在第8.3节中有更详细的讨论。

```
$ rqt
```

除了从rqt菜单中选择插件之外，还可以使用专用的运行命令，如下所示。

```
$ rqt_image_view
```

¹⁰ http://wiki.ros.org/rqt_image_view

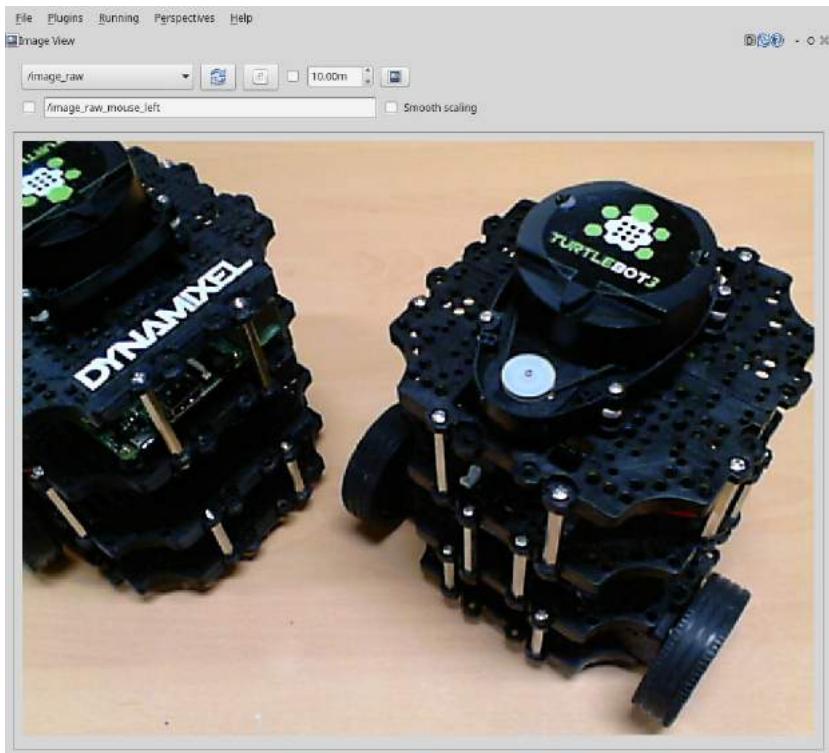


图 6-10 用Image View查看USB摄像头的图像数据的结果

6.2.4. rqt_graph

rqt_graph¹¹是用图形表示当前活动中的节点与在ROS网络上传输的消息之间的相关性的工具。这对了解当前ROS网络情况非常有用。用法很简单。如以下示例所示，为了查看第3.3节中描述的turtlesim功能包中的turtlesim_node和turtle_teleop_key，以及第6.2.3节中描述的uvc_camera功能包中的uvc_camera_node节点，将他们分别在不同的终端中运行。

```
$ rosrun turtlesim turtlesim_node  
$ rosrun turtlesim turtle_teleop_key  
$ rosrun uvc_camera uvc_camera_node  
$ rosrun image_view image_view image:=image_raw
```

¹¹ http://wiki.ros.org/rqt_graph

然后如下例所示，使用命令“rqt”运行rqt并从菜单中选择[Plugins]→[Node Graph]即可。请注意，用户也可以在终端中运行“rqt_graph”，而无需直接从菜单中选择插件。

运行rqt_graph时，节点和话题的相关性会如图6-11所示显示。

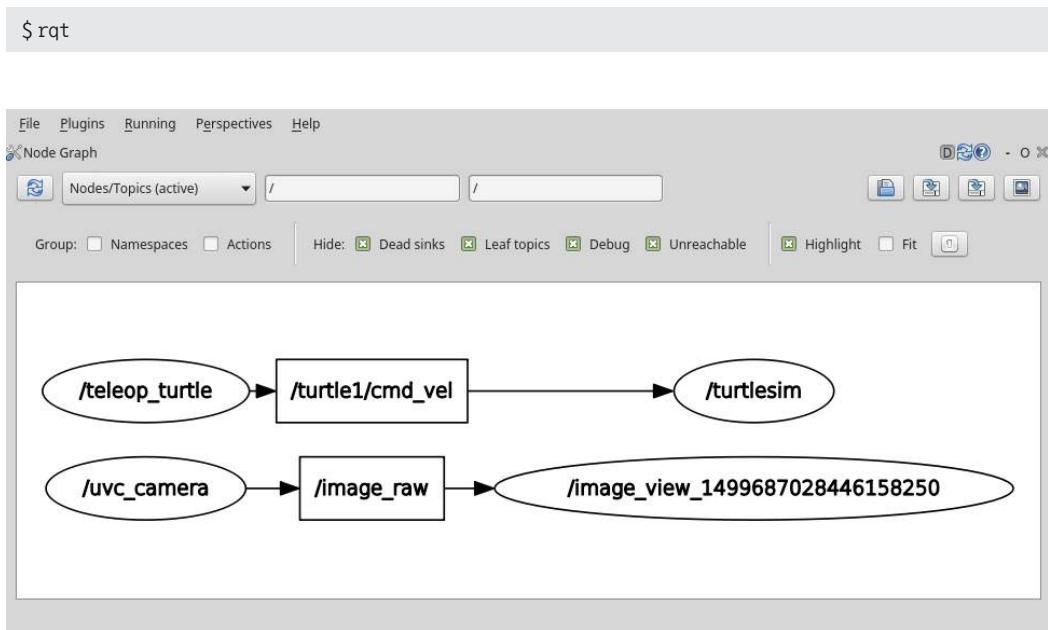


图 6-11 rqt_graph示例

在图6-11中，椭圆表示节点（/teleop_turtle、/turtlesim），方块（/turtle1/cmd_vel）表示话题消息。箭头表示发送和接收消息。在前面的例子中，运行turtle_teleop_key时，会运行teleop_turtle节点；运行turtlesim_node节点时，会运行turtlesim节点。可以看到，这两个节点正在以平移速度和旋转速度的消息类型（话题名称：/turtle1/cmd_vel）发送和接收键盘的方向键值。

uvc_camera功能包也可以通过rqt_graph确认uvc_camera节点在发出/image_raw话题消息，并且image_view_xxx节点在订阅它。这已经通过简单的几个节点确认过，但在实际的ROS编程中，会有数十个节点发送和接收各种话题消息。此时，rqt_graph对于检查当前ROS网络上节点的相关性会非常的有用。

6.2.5. rqt_plot

这一次，我们使用下面的命令运行rqt_plot，而不是在rqt界面中选择插件。作为参考，用户可以使用节点执行命令rosrun rqt_plot rqt_plot运行它。

```
$ rqt_plot
```

rqt_plot运行后，点击程序右上角的齿轮形状的选项图标。可以如图6-12所示选择一个选项，其默认设置为“MatPlot”。除MatPlot外，还提供了PyQtGraph和QwtPlot。请参阅相关的安装说明并使用所需的图形库。

例如，如果要使用PyQtGraph作为默认绘图而不是MatPlot，请从以下下载地址下载并安装最新的python-pyqtgraph_0.9.xx-x_all.deb文件。安装PyQtGraph之后，用户可以使用PyQtGraph。

- <http://www.pyqtgraph.org/downloads/>

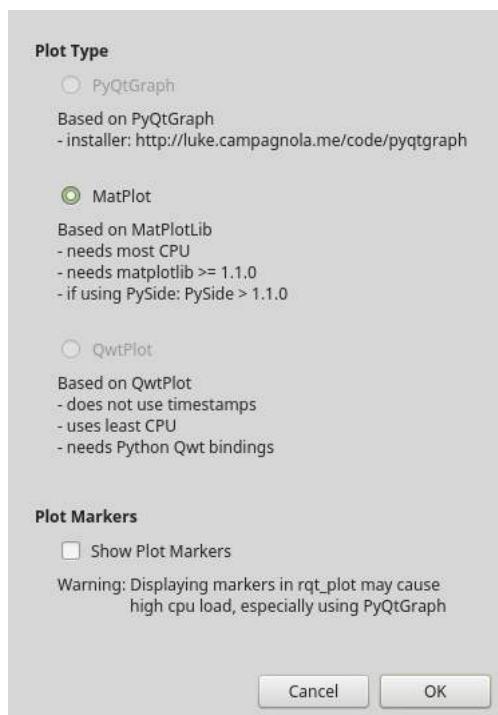


图 6-12 rqt_plot的安装选项

`rqt_plot`¹²是一个二维数据绘图工具。绘图意味着绘制坐标。换句话说，它接收到ROS消息并将其撒在坐标系上。例如，假设要标记turtlesim节点pose消息的x和y坐标。首先，运行turtlesim功能包中的turtlesim_node。

```
$ rosrun turtlesim turtlesim_node
```

然后在rqt_plot上方的Topic栏中输入/turtle1/pose/，则会在二维（x轴：数据值，y轴：时间）坐标系中绘制/turtle1/pose/节点。或者，您可以使用下一个命令立即运行它，包括指定要图示的话题。

```
$ rqt_plot /turtle1/pose/
```

接下来，运行turtlesim功能包中的turtle_teleop_key来移动屏幕上的乌龟。

```
$ rosrun turtlesim turtle_teleop_key
```

如图6-13所示，可以看到在显示龟的x位置、y位置、theta方向和平移转速。这是显示二维数据坐标的有用的工具。这里表示了turtlesim功能包，但rqt_plot也可以用于表达用户开发的节点的二维数据。特别地，适合于随着时间的推移显示传感器值，例如速度和加速度。

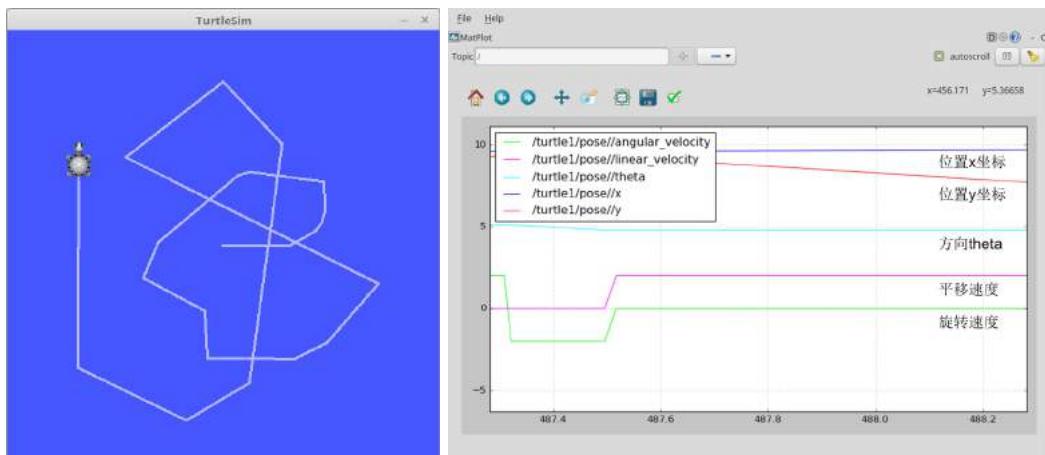


图 6-13 rqt_plot用例

¹² http://wiki.ros.org/rqt_plot

6.2.6. rqt_bag

rqt_bag是一个可以将消息进行可视化的GUI工具。在5.4.8节中，ROS日志信息中的rosbag是基于文本的，但是rqt_bag对于图像数据类的消息管理是非常有用的，因为rqt_bag多了可视化功能，因此可以立即查看摄像机的图像值。在测试之前，运行rqt_image_view和rqt_graph的工具说明中提到的turtlesim和uvc_camera中的所有相关节点。接下来，用如下命令生成为一个bag文件，记录相机的/image_raw和turtlesim的/turtle1/cmd_vel值。

在第5.4节中，曾使用rosbag程序将ROS上的各种话题消息作为bag文件进行保存、回放和压缩。rqt_bag是rosbag的GUI版本，和rosbag一样，它可以存储、回放和压缩话题消息。另外，由于它是一个GUI程序，所有的命令都是用按钮制作的，所以它很容易使用，并且用户可以用类似使用视频编辑器一样，在时间轴上来看摄像机图像。

为了利用rqt_bag的特点，将USB摄像头图像保存为一个bag文件，然后使用rqt_bag进行播放。

```
$ rosrun uvc_camera uvc_camera_node  
$ rosbag record /image_raw  
$ rqt
```

使用“rqt”命令运行rqt，然后从菜单中选择[插件（Plugins）]→[日志（Logging）]→[包（Bag）]。然后选择左上方的文件夹图标（Load Bag）加载刚才录制的*.bag文件。然后，如图6-14所示，用户可以在时间轴上查看相机图像的变化。还可以进行放大、回放和查看各时间点的数据值。如果右键单击鼠标按钮，则会出现“Publish”选项来重新发送消息。



图 6-14 rqt_bag用例

以上解释了如何安装和使用rqt工具。尽管我们没有描述本节中的所有插件，但我们建议读者尝试一下这里举的几个例子。它并不像ROS节点一样直接处理机器人或传感器，但是可以用作帮助进行数据的存储、修改和分析的辅助工具。

第7章

ROS编程基础

目前为止，我们算是学习了ROS的概略，现在开始让我们学习真正的ROS编程吧。之前的内容中出现最多的词是消息、话题、服务、动作和参数。这是因为他们是ROS的核心。作为最小执行单元的节点通过消息通信在节点之间交换I/O消息，此时使用的方法是话题、服务、动作和参数。在本章中，我们通过实例来了解ROS编程。

7.1. ROS编程前须知事项

7.1.1. 标准单位

对ROS中所使用的消息（message），推荐使用世界上最广泛运用的标准单位SI。为了确保这一点，REP-0103¹也明确了各物理量的单位。例如，长度（Length）使用米（meter）、质量（Mass）使用千克（Kilogram）、时间（Time）使用秒（Second）、电流（Current）使用安培（Ampere）、角度（Angle）使用弧度（Radian）、频率（Frequency）使用赫兹（Hertz）、力（Force）使用牛顿（Newton）、功率（Power）使用瓦（Watt）、电压（Voltage）使用伏特（Volt）、温度（Temperature）使用摄氏度（Celsius）。其他所有单位都是这些单位的组合。

例如，平移速度以米/秒表示，旋转速度以弧度/秒表示。消息鼓励重用ROS提供的方法，但也可以根据需要使用用户全新定义的新的类型的消息。然而，消息用到的单位却必须要遵守使用SI单位，这是为了让其他用户使用这种消息的时候不需要转换单位。

物理量	单位
Length	Meter
Mass	Kilogram
Time	Second
Current	Ampere
Angle	Radian

物理量	单位
Frequency	Hertz
Force	Newton
Power	Watt
Voltage	Volt
Temperature	Celsius

¹ <http://www.ros.org/reps/rep-0103.html>



REP (ROS Enhancement Proposals)

REP是一份建议书，它的内容包含由用户们在ROS社区提出的规则、新功能和管理方法。它用于以民主方式创建ROS的规则的情况，还用于在协商ROS的开发、运营和管理所需的内容的情况。收到建议书后，许多ROS用户可以查看，并通过互相协商继续修改。REP就是通过这样的过程成为ROS标准文档的。REP文件的目录可以在<http://www.ros.org/reps/rep-0000.html>找到。

7.1.2. 坐标表现方式

如图7-1左侧所示，ROS中的旋转轴²使用x, y和z轴。正面是x轴的正方向，轴是红色（R）。左边是y轴的正方向，轴用绿色（G）表示。最后，上方是z轴的正方向，轴用蓝色（B）表示。为了便于记忆，您可以将x轴视为食指，将y轴视为中指，将z轴视为拇指。顺序是x、y、z，且颜色是RGB颜色顺序。

机器人的旋转方向是右手定则³，用右手卷住的方向是正（+）方向。例如，如果机器人在原地从12点钟方向开始向9点钟方向旋转，则由于旋转角度的单位用弧度，所以我们说机器人在z轴上旋转+1.5708弧度。

这种坐标表示法在ROS编程中经常使用，必须以x: forward, y: left, z: up的形式进行编程。

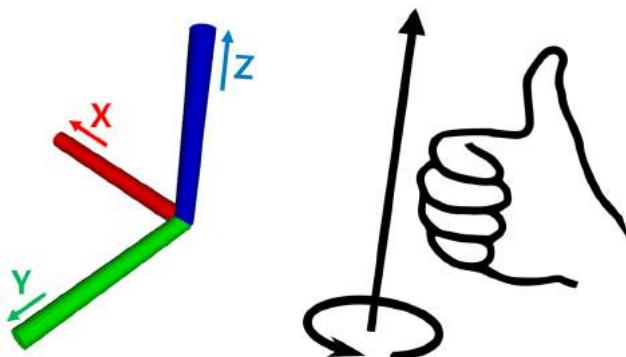


图 7-1 x, y, z轴坐标系和右手定则

² <http://www.ros.org/reps/rep-0103.html#coordinate-frame-conventions>

³ http://en.wikipedia.org/wiki/Right-hand_rule

7.1.3. 编程规则

为了最大化每个程序的源代码的可重用性，ROS指定了编程风格指南，并建议开发者遵守该指南。这减少了开发人员在处理源代码时频繁发生的额外的选项，也提高了其他协作开发人员和用户们的代码理解程度，并降低了他们之间的代码分析难度。这不是一个要求，但为了代码共享，笔者想鼓励ROS的许多用户遵守这一规则。

编程规则在wiki ([C++⁴](#), [Python⁵](#)) 中按各个编程语言有详细解释。下面的表格中整理了基本的命名规则⁶，在开始ROS编程之前熟悉这些吧。

对象	命名规则	举例
功能包	under_scored	Ex) first_ros_package
话题、服务	under_scored	Ex) raw_image
文件	under_scored	Ex) turtlebot3_fake.cpp

注意，使用ROS消息和服务时，放置在/msg和/srv目录中的消息、服务和动作文件的名称遵循CamelCased规则。这是因为*.msg、*.srv和*.action被转换为头文件后用作结构体和数据类型（例如TransformStamped.msg, SetSpeed.srv）。

命名空间	under_scored	Ex) ros_awesome_package
变量	under_scored	Ex) string table_name;
数据类型	CamelCased	Ex) typedef int32_t PropertiesNumber;
类	CamelCased	Ex) class UrlTable
结构体	CamelCased	Ex) struct UrlTableProperties
枚举型	CamelCased	Ex) enum ChoiceNumber
函数	camelCased	Ex) addTableEntry();
函数方法	camelCased	Ex) void setNumEntries(int32_t num_entries)
常数	ALL_CAPITALS	Ex) const uint8_t DAYS_IN_A_WEEK = 7;
宏定义	ALL_CAPITALS	Ex) #define PI_ROUNDED 3.0

⁴ <http://wiki.ros.org/CppStyleGuide>

⁵ <http://wiki.ros.org/PyStyleGuide>

⁶ http://wiki.ros.org/ROS/Patterns/Conventions#Naming_ROS_Resources

7.2. 发布者节点和订阅者节点的创建和运行

ROS消息通信中使用的发布者（Publisher）和订阅者（Subscriber）可以被发送和接收所代替。在ROS中，发送端称为发布者，接收端称为订阅者。本节旨在创建一个简单的msg文件，并创建和运行发布者和订阅者节点。

7.2.1. 创建功能包

以下命令是创建ros_tutorials_topic功能包的命令。这个功能包依赖于message_generation、std_msgs和roscpp功能包，因此将这些用作依赖选项。第二行命令意味着将使用创建新的功能包时用到的message_generation表示将使用创建新消息的功能包std_msgs（ROS标准消息功能包）和roscpp（在ROS中使用C/C++的客户端程序库）必须在创建功能包之前安装。用户可以在创建功能包时指定这些相关的功能包设置，但也可以在创建功能包之后直接在package.xml中修改。

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg ros_tutorials_topic message_generation std_msgs roscpp
```

创建功能包时，将在~/catkin_ws/src目录中创建ros_tutorials_topic功能包目录，并在该功能包目录中创建ROS功能包的默认目录和CMakeLists.txt和package.xml文件。可以使用下面的ls命令检查它，并使用基于GUI的Nautilus（类似Windows资源管理器）来检查功能包的内部。

```
$ cd ros_tutorials_topic  
$ ls  
include          → 头文件目录  
src              → 源代码目录  
CMakeLists.txt   → 构建配置文件  
package.xml      → 功能包配置文件
```

7.2.2. 修改功能包配置文件

ROS的必备配置文件package.xml是一个包含功能包信息的XML文件，其中包含用于描述功能包名称、作者、许可证和依赖包的信息。使用以下命令，利用编辑器（gedit、vim、emacs等）打开文件，并修改它以匹配当前节点。

```
$ gedit package.xml
```

以下代码显示如何修改package.xml文件以匹配我们这次要创建功能包。笔者的个人信息包含在如下文件内容中，您可以修改它，以适应您的需求。有关每个选项的详细说明，请参见第4.9节。

ros_tutorials_topic/package.xml

```
<?xml version="1.0"?>
<package>
  <name>ros_tutorials_topic</name>
  <version>0.1.0</version>
  <description>ROS tutorial package to learn the topic</description>
  <license>Apache License 2.0</license>
  <author email="pyo@robotis.com">Yoonseok Pyo</author>
  <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer>
  <url type="bugtracker">https://github.com/ROBOTIS-GIT/ros_tutorials/issues</url>
  <url type="repository">https://github.com/ROBOTIS-GIT/ros_tutorials.git</url>
  <url type="website">http://www.robotis.com</url>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>message_runtime</run_depend>
  <export></export>
</package>
```

7.2.3. 修改构建配置文件（CMakeLists.txt）

ROS的构建系统catkin基本上使用CMake，它在功能包目录中的CMakeLists.txt文件中描述了构建环境。该文件设置可执行文件的创建、依赖包优先构建、链接创建等。

```
$ gedit CMakeLists.txt
```

以下是为了匹配正在创建的功能包而修改的CMakeLists.txt。同样，有关每个选项的详细说明，请参见第4.9节。

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_topic)

## catkin构建时需要的组件包。
## 是依赖包，是message_generation、std_msgs和roscpp。
## 如果这些功能包不存在，在构建过程中会发生错误。
find_package(catkin REQUIRED COMPONENTS message_generation std_msgs roscpp)

## 消息声明：MsgTutorial.msg
add_message_files(FILES MsgTutorial.msg)

## 这是设置依赖性消息的选项。
## 如果未安装std_msgs，则在构建过程中会发生错误。
generate_messages(DEPENDENCIES std_msgs)

## catkin功能包选项，描述了库、catkin构建依赖项和系统依赖的功能包。
catkin_package(
    LIBRARIES ros_tutorials_topic
    CATKIN_DEPENDS std_msgs roscpp
)

## 设置包含目录。
include_directories(${catkin_INCLUDE_DIRS})

## topic_publisher节点的构建选项。
## 配置可执行文件、目标链接库和其他依赖项。
add_executable(topic_publisher src/topic_publisher.cpp)
add_dependencies(topic_publisher ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_publisher ${catkin_LIBRARIES})

## topic_subscriber节点的构建选项。
add_executable(topic_subscriber src/topic_subscriber.cpp)
add_dependencies(topic_subscriber ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_subscriber ${catkin_LIBRARIES})
```

7.2.4. 创建消息文件

在上述的CMakeLists.txt文中添加了如下选项。

```
add_message_files(FILES MsgTutorial.msg)
```

这意味着在构建时要包含消息MsgTutorial.msg，该消息将在此节点中被使用。现在我们还没有创建MsgTutorial.msg，因此按以下顺序创建它：

\$ rosdep install ros_tutorials_topic	→ 移动到功能包目录
\$ mkdir msg	→ 功能包中创建新的msg消息目录
\$ cd msg	→ 转到创建的msg目录
\$ gedit MsgTutorial.msg	→ 创建新的MsgTutorial.msg文件并修改内容

内容很简单。如下所示，创建一个time消息类型的stamp消息和一个int32消息类型的数据消息，除了这些消息类型之外，还有一些如bool、int8、int16、float32、string、time、duration和common_msgs等基本消息类型⁷，以及在ROS中收集常用消息的common_msgs⁸。这里我们只是为了创建一个简单的例子，因此用了time和int32。

```
ros_tutorials_topic/msg/MsgTutorial.msg
time stamp
int32 data
```



消息（msg、srv、action）功能包的独立化

一般情况下，建议将消息文件msg和服务文件srv制作成一个只包含消息文件的单独的包，而不是将其包含在可执行节点中。其原因是，假设订阅者节点和发布者节点在不同的计算机上运行，存在的问题是必须安装不必要的节点，因为这两个节点只有在它们具有相互依赖性时才能使用。如果您独立创建消息功能包，则只需将独立于消息的功能包添加到依赖性选项，从而消除功能包之间不必要的依赖关系。但是，在本书中，我们为了简化代码已经将消息文件包含在可执行节点中。

⁷ http://wiki.ros.org/std_msgs

⁸ http://wiki.ros.org/common_msgs

7.2.5. 创建发布者节点

在前面的CMakeLists.txt文件中，给了生成以下可执行文件的选项。

```
add_executable(topic_publisher src/topic_publisher.cpp)
```

换句话说，是在src目录中构建topic_publisher.cpp文件以创建topic_publisher可执行文件。那么我们按如下顺序创建一个执行发布者节点函数的源代码。

```
$ roscd ros_tutorials_topic/src          → 移至src目录，该目录是功能包的源代码目录  
$ gedit topic_publisher.cpp              → 新建源文件并修改内容
```

```
ros_tutorials_topic/src/topic_publisher.cpp

#include "ros/ros.h"                      // ROS默认头文件  
#include "ros_tutorials_topic/MsgTutorial.h" // MsgTutorial消息头文件（构建后自动生成）

int main(int argc, char **argv)           // 节点主函数
{
    ros::init(argc, argv, "topic_publisher"); // 初始化节点名称
    ros::NodeHandle nh;                      // 声明一个节点句柄来与ROS系统进行通信

    // 声明发布者，创建一个使用ros_tutorials_topic功能包的MsgTutorial 消息文件的
    // 发布者ros_publisher。话题名称是"ros_publisher_msg"，
    // 消息文件发布者队列(queue) 的大小设置为100
    ros::Publisher ros_publisher =
    nh.advertise<ros_tutorials_topic::MsgTutorial>("ros_publisher_msg", 100);

    // 设定循环周期。"10"是指10Hz，是以0.1秒间隔重复
    ros::Rate loop_rate(10);

    ros_tutorials_topic::MsgTutorial msg;      // 以MsgTutorial消息文件格式声明一个叫做msg的消息

    int count = 0;                            // 声明要在消息中使用的变量

    while (ros::ok())
    {
        msg.stamp = ros::Time::now();          // 把当前时间传给msg的下级消息stamp
        msg.data = count;                     // 将变量count的值传给下级消息data
```

```

ROS_INFO("send msg = %d", msg.stamp.sec);      // 显示stamp.sec消息
ROS_INFO("send msg = %d", msg.stamp.nsec);      // 显示stamp.nsec消息

ROS_INFO("send msg = %d", msg.data);           // 显示data消息

ros_tutorial_pub.publish(msg);                  // 发布消息。

loop_rate.sleep();                            // 按照上面定义的循环周期进行暂停

++count;                                     // 变量count增加1
}

return 0;
}

```

7.2.6. 创建订阅者节点

在CMakeLists.txt文件中添加以下选项来生成可执行文件。

```
add_executable(topic_subscriber src/topic_subscriber.cpp)
```

也就是说，通过构建topic_subscriber.cpp文件来创建topic_subscriber可执行文件。我们创建一个按照以下顺序执行订阅节点功能的源代码。

\$ roscd ros_tutorials_topic/src	→ 移动到src目录，该目录是功能包的源代码目录
\$ gedit topic_subscriber.cpp	→ 创建和修改新的源代码文件

ros_tutorials_topic/src/topic_subscriber.cpp

```

#include "ros/ros.h"                                // ROS的默认头文件
#include "ros_tutorials_topic/MsgTutorial.h"        // MsgTutorial消息头文件（构建后自动生成）

// 这是一个消息后台函数,
// 此函数在收到一个下面设置的名为ros_tutorial_msg的话题时候被调用。
// 输入的消息是从ros_tutorials_topic功能包接收MsgTutorial消息。
void msgCallback(const ros_tutorials_topic::MsgTutorial::ConstPtr& msg)
{

```

```

ROS_INFO("recieve msg = %d", msg->stamp.sec);           // 显示stamp.sec消息
ROS_INFO("recieve msg = %d", msg->stamp.nsec);           // 显示stamp.nsec消息
ROS_INFO("recieve msg = %d", msg->data);                 // 显示data消息
}

int main(int argc, char **argv)                           // 节点主函数
{
    ros::init(argc, argv, "topic_subscriber");           // 初始化节点名称

    ros::NodeHandle nh;                                  // 声明用于ROS系统和通信的节点句柄

    // 声明订阅者，创建一个订阅者ros_tutorial_sub,
    // 它利用ros_tutorials_topic功能包的的MsgTutorial消息文件。
    // 话题名称是"ros_tutorial_msg"，订阅者队列（queue）的大小设为100。
    ros::Subscriber ros_tutorial_sub = nh.subscribe("ros_tutorial_msg", 100, msgCallback);

    // 用于调用后台函数，等待接收消息。在接收到消息时执行后台函数。
    ros::spin();

    return 0;
}

```

```

$ cd ~/catkin_ws      → 移动到catkin目录
$ catkin_make         → 执行catkin构建

```

7.2.7. 构建（build）节点

现在使用以下命令在ros_tutorials_topic功能包中构建消息文件、发布者节点和订阅者节点。ros_tutorials_topic功能包的源代码位于“~/catkin_ws/src/ros_tutorials_topic/src”中，而ros_tutorials_topic功能包中的消息文件位于“~/catkin_ws/src/ros_tutorials_topic/msg”中。

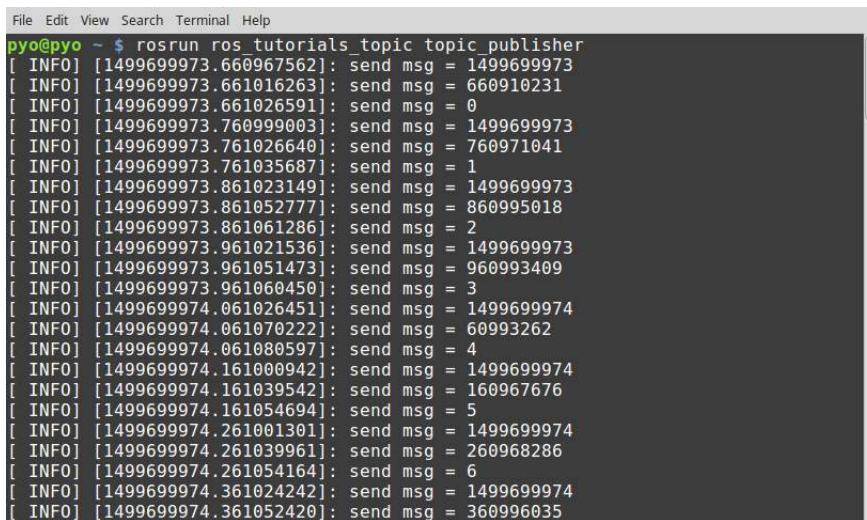
基于此的构建将分别在“~/catkin_ws”的“/build”和“/devel”目录中生成文件。“/build”目录中保存catkin构建用到的配置内容，而“/devel/lib/ros_tutorials_topic”目录中保存可执行文件。另外，“/devel/include/ros_tutorials_topic”目录存储着从消息文件自动生成的消息头文件。如果您想了解生成文件，请根据目录查看文件。

7.2.8. 运行发布者

让我们运行发布者吧。以下是使用rosrun命令运行ros_tutorials_topic功能包的rosTutorialMsgPublisher节点的命令。运行节点之前，请确保要从另一个终端运行roscore。在以下示例中，即使没有解释，roscore也必须在运行节点之前运行。

```
$ roscore  
$ rosrun ros_tutorials_topic topic_publisher
```

运行发布者时，您会看到如图7-2所示的输出屏幕。但是，屏幕上显示的内容仅仅是使用ROS专用的ROS_INFO()函数将信息显示在屏幕上，这个函数与常见编程语言中使用的printf()函数类似。实际上，为了发布话题，您必须使用一个订阅者节点或rostopic等能做订阅者节点角色的命令。



```
File Edit View Search Terminal Help  
pyo@pyo ~ $ rosrun ros_tutorials_topic topic_publisher  
[ INFO] [1499699973.660967562]: send msg = 1499699973  
[ INFO] [1499699973.661016263]: send msg = 660910231  
[ INFO] [1499699973.661026591]: send msg = 0  
[ INFO] [1499699973.760999003]: send msg = 1499699973  
[ INFO] [1499699973.761026640]: send msg = 760971041  
[ INFO] [1499699973.761035687]: send msg = 1  
[ INFO] [1499699973.861023149]: send msg = 1499699973  
[ INFO] [1499699973.861052777]: send msg = 860995018  
[ INFO] [1499699973.861061286]: send msg = 2  
[ INFO] [1499699973.961021536]: send msg = 1499699973  
[ INFO] [1499699973.961051473]: send msg = 960993409  
[ INFO] [1499699973.961060450]: send msg = 3  
[ INFO] [1499699974.061026451]: send msg = 1499699974  
[ INFO] [1499699974.061070222]: send msg = 60993262  
[ INFO] [1499699974.061080597]: send msg = 4  
[ INFO] [1499699974.161000942]: send msg = 1499699974  
[ INFO] [1499699974.161039542]: send msg = 160967676  
[ INFO] [1499699974.161054694]: send msg = 5  
[ INFO] [1499699974.261001301]: send msg = 1499699974  
[ INFO] [1499699974.261039961]: send msg = 260968286  
[ INFO] [1499699974.261054164]: send msg = 6  
[ INFO] [1499699974.361024242]: send msg = 1499699974  
[ INFO] [1499699974.361052420]: send msg = 360996035
```

图 7-2 topic_publisher节点的运行画面

下面，我们使用rostopic命令获取topic_publisher发布的话题吧。首先，我们来看一下ROS网络当前正在使用的话题列表。通过将list选项添加到rostopic命令来查看是否存在rosTutorialMsg话题。

```
$ rostopic list  
/ros_tutorial_msg  
/rosout  
/rosout_agg
```

接下来，让我们看看我们运行的发布者节点发布的消息。换句话说，是检查ros_tutorial_msg话题消息。您可以看到发布的消息，如图7-3所示。

```
$ rostopic echo /ros_tutorial_msg
```

```
File Edit View Search Terminal Help  
pyo@pyo ~ $ rostopic echo /ros_tutorial_msg  
stamp:  
  secs: 1499700351  
  nsecs: 684514825  
data: 1713  
---  
stamp:  
  secs: 1499700351  
  nsecs: 784542724  
data: 1714  
---  
stamp:  
  secs: 1499700351  
  nsecs: 884544453  
data: 1715  
---  
stamp:  
  secs: 1499700351  
  nsecs: 984543934  
data: 1716  
---  
stamp:  
  secs: 1499700352  
  nsecs: 84543178
```

图 7-3 接收ros_tutorial_msg话题的明细

7.2.9. 运行订阅者

以下是为了运行订阅者使用ROS节点命令rosrun来运行ros_tutorials_topic功能包的topic_subscriber节点的过程。

```
$ rosrun ros_tutorials_topic topic_subscriber
```

当运行订阅者时，可以看到如图7-4所示的输出屏幕。订阅者接收到了发布者发布的ros_tutorial_msg话题的消息，并在屏幕上显示该值。

```

File Edit View Search Terminal Help
pyo@pyo ~ $ rosrun ros_tutorials topic topic_subscriber
[ INFO] [1499700485.184875537]: recieve msg = 1499700485
[ INFO] [1499700485.184946471]: recieve msg = 184567102
[ INFO] [1499700485.184957742]: recieve msg = 3048
[ INFO] [1499700485.284812298]: recieve msg = 1499700485
[ INFO] [1499700485.284836776]: recieve msg = 284574255
[ INFO] [1499700485.284844492]: recieve msg = 3049
[ INFO] [1499700485.384811804]: recieve msg = 1499700485
[ INFO] [1499700485.384839629]: recieve msg = 384569171
[ INFO] [1499700485.384849957]: recieve msg = 3050
[ INFO] [1499700485.484795619]: recieve msg = 1499700485
[ INFO] [1499700485.484824179]: recieve msg = 484569717
[ INFO] [1499700485.484838747]: recieve msg = 3051
[ INFO] [1499700485.584792760]: recieve msg = 1499700485
[ INFO] [1499700485.584820628]: recieve msg = 584569677
[ INFO] [1499700485.584830560]: recieve msg = 3052
[ INFO] [1499700485.684824324]: recieve msg = 1499700485
[ INFO] [1499700485.684852121]: recieve msg = 684581217
[ INFO] [1499700485.684861556]: recieve msg = 3053
[ INFO] [1499700485.785495346]: recieve msg = 1499700485
[ INFO] [1499700485.785527583]: recieve msg = 785156898
[ INFO] [1499700485.785552403]: recieve msg = 3054
[ INFO] [1499700485.884855517]: recieve msg = 1499700485
[ INFO] [1499700485.884885781]: recieve msg = 884544763

```

图 7-4 topic_subscriber节点的运行画面

7.2.10. 检查运行中的节点的通信状态

让我们用6.2节中介绍的rqt命令来查看运行中的节点的通信状态。您可以使用rqt_graph或rqt，如下所示。执行rqt时，在菜单中选择[Plugins]→[Introspection]→[Node Graph]。则如图7-5所示，可以确认当前在ROS上运行的节点和消息。



图 7-5 利用rqt_graph看到的两个节点之间的关系

在当前的ROS网络上，发布者节点（topic_publisher）正在传输话题（ros_tutorial_msg），并且可以确认它正在接收订阅者节点（topic_subscriber）。

在本节中，我们创建了话题中用到的发布者和订阅者节点，并执行它以了解如何在节点之间进行通信。相关代码可以在以下github地址找到：

- https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/ros_tutorials_topic

如果您想马上应用它，可以在“catkin_ws/src”目录中用以下命令来克隆源代码，并进行构建。然后运行topic_publisher和topic_subscriber节点。

```
$ cd ~/catkin_ws/src  
$ git clone https://github.com/ROBOTIS-GIT/ros_tutorials.git  
$ cd ~/catkin_ws  
$ catkin_make
```

```
$ rosrun ros_tutorials_topic topic_publisher
```

```
$ rosrun ros_tutorials_topic topic_subscriber
```

7.3. 创建和运行服务服务器与客户端节点

服务由服务服务器（service server）和服务客户端（service client）组成，其中服务服务器仅在收到请求（request）时才会响应（response），而服务客户端则会发送请求并接收响应。与话题不同，服务是一次性消息通信。因此，当服务的请求和响应完成时，两个节点的连接会被断开。

这种服务通常在让机器人执行特定任务时用到。或者用于需要在特定条件下做出反应的节点。由于它是一次性的通信方式，因此对网络的负载很小，所以是一种非常有用的通信手段，例如被用作一种代替话题的通信手段。

本节旨在创建一个简单的服务文件，并创建和运行一个服务服务器（server）节点和一个服务客户端（client）节点。

7.3.1. 创建功能包

以下命令创建ros_tutorials_service功能包。这个功能包依赖于message_generation、std_msgs和roscpp功能包，因此将他们用作依赖性选项。其中，message_generation是用于创建新消息的功能包，std_msgs是ROS的标准消息功能包，roscpp是在ROS中使用C/C++的客户端程序库，这些都必须在创建功能包之前安装好。用户可以在创建功能包时指定这些依赖关系，但也可以在创建功能包之后直接在package.xml中对其进行修改。

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg ros_tutorials_service message_generation std_msgs roscpp
```

在创建功能包的时候，会在“~/catkin_ws/src”目录中生成ros_tutorials_service功能包目录，在这个功能包目录中会生成ROS功能包所要配备的默认目录，以及CMakeLists.txt和package.xml文件。下面使用ls命令检查内容。

```
$ cd ros_tutorials_service  
$ ls  
include          → 头文件目录  
src              → 源代码目录  
CMakeLists.txt   → 构建配置文件  
package.xml      → 功能包配置文件
```

7.3.2. 修改功能包配置文件（package.xml）

ROS所必需的配置文件之一package.xml是一个包含功能包信息的XML文件，它描述了功能包名称、作者、许可证和依赖包。使用以下命令使用编辑器（gedit、vim、emacs等）打开文件，并修改它，以匹配当前节点。

```
$ gedit package.xml
```

以下代码显示如何修改package.xml文件以匹配您正在创建的功能包。我的个人信息包含在内容中，所以请将其更改为您的个人信息。有关每个选项的详细说明，请参见第4.9节。

ros_tutorials_service/package.xml

```
<?xml version="1.0"?>
<package>
  <name>ros_tutorials_service</name>
  <version>0.1.0</version>
  <description>ROS tutorial package to learn the service</description>
  <license>Apache License 2.0</license>
  <author email="pyo@robotis.com">Yoonseok Pyo</author>
  <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer>
  <url type="bugtracker">https://github.com/ROBOTIS-GIT/ros_tutorials/issues</url>
  <url type="repository">https://github.com/ROBOTIS-GIT/ros_tutorials.git</url>
  <url type="website">http://www.robotis.com</url>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>message_runtime</run_depend>
  <export></export>
</package>
```

7.3.3. 修改构建配置文件（CMakeLists.txt）

ROS的构建系统catkin以CMake为基础，它在功能包目录中的CMakeLists.txt文件里描述了构建环境。该文件设置可执行文件的创建、依赖包优先构建、链接生成等。与上述ros_tutorials_topic不同，如果添加了发布者节点、订阅者节点和msg文件，则ros_tutorials_service功能包将添加新的服务服务器节点、服务客户端节点和服务文件 (*.srv)。

```
$ gedit CMakeLists.txt
```

ros_tutorials_service/CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_service)
```

```
## 这是进行catkin构建时所需的组件包。  
## 依赖包是message_generation、std_msgs和roscpp。如果这些包不存在，在构建过程中会发生错误。  
find_package(catkin REQUIRED COMPONENTS message_generation std_msgs roscpp)  
  
## 服务声明：SrvTutorial.srv  
add_service_files(FILES SrvTutorial.srv)  
  
## 这是一个设置依赖消息的选项。  
## 如果未安装std_msgs，则在构建过程中会发生错误。  
generate_messages(DEPENDENCIES std_msgs)  
  
## 这是catkin功能包选项，它描述了库、catkin构建依赖和依赖系统的功能包。  
catkin_package(  
    LIBRARIES ros_tutorials_service  
    CATKIN_DEPENDS std_msgs roscpp  
)  
  
## 设置包含目录。  
include_directories(${catkin_INCLUDE_DIRS})  
  
## 这是service_server节点的构建选项。  
## 设置可执行文件、目标链接库和附加依赖项。  
add_executable(service_server src/service_server.cpp)  
add_dependencies(service_server ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})  
target_link_libraries(service_server ${catkin_LIBRARIES})  
  
## 这是节点的构建选项。  
add_executable(service_client src/service_client.cpp)  
add_dependencies(service_client ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})  
target_link_libraries(service_client ${catkin_LIBRARIES})
```

7.3.4. 创建服务文件

CMakeLists.txt文件中加了下面的选项。

```
add_service_files(FILES SrvTutorial.srv)
```

这是在构建本次的节点中使用的SrvTutorial.srv时所包含的内容。现在您还没有创建SrvTutorial.srv，请按以下顺序创建它。

```
$ roscd ros_tutorials_service      → 移动到功能包目录  
$ mkdir srv                         → 在功能包中创建一个名为srv的新服务目录  
$ cd srv                            → 转到创建的srv目录  
$ gedit SrvTutorial.srv             → 新建和修改SrvTutorial.srv文件
```

内容很简单。让我们以int64格式设计服务请求（request）a、b，和结果服务响应（response）result，如下所示。“---”是分隔符，用于分隔请求和响应。除了请求和响应之间有一个分隔符之外，它与上述话题的消息相同。

```
ros_tutorials_service/srv/SrvTutorial.srv
```

```
int64 a
int64 b
---
int64 result
```

7.3.5. 创建服务服务器节点

在CMakeLists.txt文件中添加了如下选项来生成可执行文件。

```
add_executable(service_server src/service_server.cpp)
```

换句话说，是构建service_server.cpp文件来创建service_server可执行文件。我们按以下顺序编写一个具有服务服务器节点功能的程序吧。

```
$ roscd ros_tutorials_service/src      → 移动到功能包的源代码目录src
$ gedit service_server.cpp              → 创建和修改源文件
```

```
ros_tutorials_service/src/service_server.cpp
```

```
#include "ros/ros.h"                      // ROS的基本头文件
#include "ros_tutorials_service/SrvTutorial.h" // SrvTutorial服务头文件（构建后自动生成）

// 如果有服务请求，将执行以下处理
// 将服务请求设置为req，服务响应则设置为res。
```

```

bool calculation(ros_tutorials_service::SrvTutorial::Request &req,
                 ros_tutorials_service::SrvTutorial::Response &res)
{
    // 在收到服务请求时，将a和b的和保存在服务响应值中
    res.result = req.a + req.b;

    // 显示服务请求中用到的a和b的值以及服务响应result值
    ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
    ROS_INFO("sending back response: %ld", (long int)res.result);

    return true;
}

int main(int argc, char **argv)           // 节点主函数
{
    ros::init(argc, argv, "service_server"); // 初始化节点名称
    ros::NodeHandle nh;                      // 声明节点句柄

    // 声明服务服务器
    // 声明利用ros_tutorials_service功能包的SrvTutorial服务文件的
    // 服务服务器ros_tutorials_service_server
    // 服务名称是ros_tutorial_srv，且当有服务请求时，执行calculation函数。
    ros::ServiceServer ros_tutorials_service_server = nh.advertiseService("ros_tutorial_srv",
    calculation);

    ROS_INFO("ready srv server!");

    ros::spin(); // 等待服务请求

    return 0;
}

```

7.3.6. 创建服务客户端节点

在CMakeLists.txt文件中添加了一个选项来生成可执行文件。

```
add_executable(service_client src/service_client.cpp)
```

换句话说，是通过构建service_client.cpp文件来创建service_client可执行文件。我们按以下顺序编写一个执行服务客户端节点功能的程序吧。

```
$ roscd ros_tutorials_service/src          → 移动到功能包的源代码目录src  
$ gedit service_client.cpp                 → 创建和修改源文件
```

```
ros_tutorials_service/src/service_client.cpp
```

```
#include "ros/ros.h"                      // ROS的基本头文件  
#include "ros_tutorials_service/SrvTutorial.h"    // SrvTutorial服务头文件（构建后自动生成）  
  
#include <cstdlib>                         // 使用atoll函数所需的库  
  
int main(int argc, char **argv)             // 节点主函数  
{  
    ros::init(argc, argv, "service_client");   // 初始化节点名称  
  
    if (argc != 3)                            // 处理输入值错误  
    {  
        ROS_INFO("cmd : rosrun ros_tutorials_service service_client arg0 arg1");  
        ROS_INFO("arg0: double number, arg1: double number");  
        return 1;  
    }  
  
    ros::NodeHandle nh;           // 声明与ROS系统通信的节点句柄  
  
    // 声明客户端，声明利用ros_tutorials_service功能包的SrvTutorial服务文件的  
    // 服务客户端ros_tutorials_service_client。  
    // 服务名称是"ros_tutorial_srv"  
    ros::ServiceClient ros_tutorials_service_client =  
    nh.serviceClient<ros_tutorials_service::SrvTutorial>("ros_tutorial_srv");  
  
    // 声明一个使用SrvTutorial服务文件的叫做srv的服务  
    ros_tutorials_service::SrvTutorial srv;  
  
    // 在执行服务客户端节点时用作输入的参数分别保存在a和b中  
    srv.request.a = atol(argv[1]);  
    srv.request.b = atol(argv[2]);
```

```
// 请求服务，如果请求被接受，则显示响应值
if (ros_tutorials_service_client.call(srv))
{
    ROS_INFO("send srv, srv.Request.a and b: %ld, %ld", (long int)srv.request.a, (long int)srv.request.b);
    ROS_INFO("receive srv, srv.Response.result: %ld", (long int)srv.response.result);
}
else
{
    ROS_ERROR("Failed to call service ros_tutorial_srv");
    return 1;
}
return 0;
}
```

7.3.7. 构建节点

使用以下命令构建ros_tutorials_service功能包的服务文件、服务服务器节点和客户端节点。ros_tutorials_service功能包的源文件位于“~/catkin_ws/src/ros_tutorials_service/src”目录中，服务文件位于“~/catkin_ws/src/ros_tutorials_service/srv”目录中。

```
$ cd ~/catkin_ws && catkin_make      → 转到catkin目录并运行catkin构建
```

生成的文件位于“~/catkin_ws/build”目录和“~/catkin_ws/devel”目录。目录“~/catkin_ws/build”包含catkin构建中使用的配置，“~/catkin_ws/devel/lib/ros_tutorials_service”包含可执行文件，“~/catkin_ws/devel/include/ros_tutorials_service”保存从消息文件自动生成的服务头文件。如果您想知道这些文件，请进入各自的目录查看。

7.3.8. 运行服务服务器

在前一节中编写的服務服务器被设定为一直等待，不做任何处理，直到有服务请求。因此，执行以下命令时，服务服务器将等待服务请求。运行节点之前一定要运行roscore。

```
$ roscore  
$ rosrun ros_tutorials_service service_server  
[INFO] [1495726541.268629564]: ready srv server!
```

7.3.9. 运行服务客户端

如果已经运行了服务服务器，请使用以下命令运行服务客户端：

```
$ rosrun ros_tutorials_service service_client 2 3  
[INFO] [1495726543.277216401]: send srv, srv.Request.a and b: 2, 3  
[INFO] [1495726543.277258018]: receive srv, srv.Response.result: 5
```

从上面编写的代码可知，在运行服务客户端时输入的执行参数2和3会被作为服务请求值。结果，2和3分别请求服务作为a和b值，后来作为结果值，作为响应值收到了这两者的总和。在这种情况下，它只是用作执行参数，但在实际使用中，可以用指令代替，可以将要计算的值和触发变量用作服务请求值。

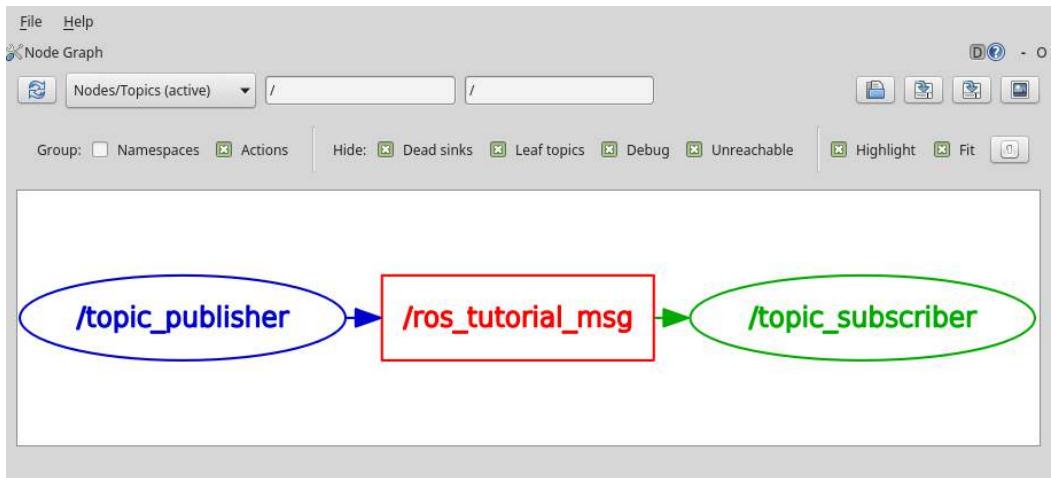


图 7-6 话题发布者（左）和话题订阅者（右）

请注意，服务与图7-6的话题发布者和订阅者不同，是一次性的，因此在rqt_graph中不可用。

7.3.10. rosservice call命令的用法

服务请求可以由service_client等服务客户端节点来执行，但有一种使用“rosservice call”或者rqt的serviceCaller的方法。我们来看看如何使用rosservice call吧。

在下面的命令中执行rosservice call命令之后，写入相应的服务名称，例如/ros_tutorial_srv，然后写入服务请求所需的参数即可。

```
$ rosservice call /ros_tutorial_srv 10 2  
result: 12
```

在前面的例子中，我们如下面的服务文件，将int64类型的a和b设置为请求，所以我们输入了10和2作为参数。服务响应的结果是int64的result以12的值返回。

```
int64 a  
int64 b  
---  
int64 result
```

7.3.11. GUI工具Service Caller的用法

最后，介绍使用rqt的ServiceCaller的方法，它使用一个GUI形式的界面。首先，运行ROS的GUI工具rqt。

```
$ rqt
```

然后从rqt程序的菜单中选择[插件[Plugins]→[Service]→[Service Caller]，然后出现如下屏幕。

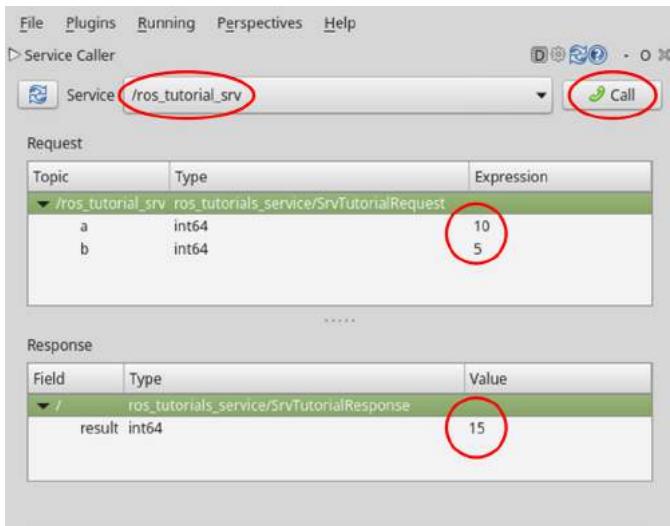


图 7-7 通过rqt的Service Caller插件的服务请求

如果在顶部的“Service”项目中选择服务名称，则会在Request中看到服务请求所需的信息。要请求服务，请在每个请求信息的Expression中输入信息。我给a输入了10，给b输入了5。然后点击右上角绿色电话的<Call>图标，服务请求将被执行，屏幕下方的Response将显示服务响应的结果。

前面描述的rosservice call具有直接在终端上运行的优点，但对于不熟悉使用Linux或ROS命令的用户，我们推荐rqt的Service Caller。

在本节中，我们创建了一个服务服务器和一个客户端节点，并尝试执行它，还学习了如何在节点之间进行服务通信。相关资源可以在以下github地址找到：

- https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/ros_tutorials_service

如果您想马上应用它，可以用catkin_ws/src目录中的以下命令来克隆源代码，并运行构建。然后运行service_server和service_client节点。

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ROBOTIS-GIT/ros_tutorials.git
$ cd ~/catkin_ws
$ catkin_make
```

```
$ rosrun ros_tutorials_service service_server
```

```
$ rosrun ros_tutorials_service service_client 2 3
```

7.4. 创建和运行动作服务器和客户端节点

在本节中，我们将创建并运行一个动作服务器和动作客户端节点，我们将了解第4.2节中讨论的第三种消息通信方法-动作⁹。动作与话题和服务不同，在异步、双向，以及请求和响应之间需要很长的时间的情况下，以及需要中途结果值等需要更复杂的编程的情况下显得格外有用。这里我们将使用ROS Wiki中介绍的actionlib示例¹⁰。

7.4.1. 生成功能包

以下命令创建ros_tutorials_action功能包。这个功能包依赖于message_generation、std_msgs、actionlib_msgs、actionlib和roscpp功能包，因此将这些作为依赖选项。

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg ros_tutorials_action message_generation std_msgs actionlib_msgs actionlib roscpp
```

7.4.2. 修改功能包配置文件（package.xml）

包括修改功能包配置文件（package.xml）的大部分过程与上述话题和服务中描述的过程非常相似。除了本节的例子中的细节之外，我只会提到源代码并跳过细节。

```
$ roscd ros_tutorials_action  
$ gedit package.xml
```

⁹ <http://wiki.ros.org/actionlib>

¹⁰ http://wiki.ros.org/actionlib_tutorials/Tutorials

ros_tutorials_action/package.xml

```
<?xml version="1.0"?>
<package>
  <name>ros_tutorials_action</name>
  <version>0.1.0</version>
  <description>ROS tutorial package to learn the action</description>
  <license>BSD</license>
  <author>Melonee Wise</author>
  <maintainer email="pyo@robotis.com">pyo</maintainer>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>actionlib</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>actionlib_msgs</build_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>actionlib</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>actionlib_msgs</run_depend>
  <run_depend>message_runtime</run_depend>
  <export></export>
</package>
```

7.4.3. 修改构建配置文件（CMakeLists.txt）

与上面介绍的ros_tutorials_topic和ros_tutorials_service节点的构建配置文件不同的是，如果这些节点的构建过程中生成了msg和srv文件，那么ros_tutorials_action功能包会生成动作文件（*.action）。此外，还添加了一个新的动作服务器节点和一个动作客户端节点作为使用它的示例节点。另外，由于我们使用了一个ROS之外的名为Boost的库，所以多了一个单独的依赖项选项。

```
$ gedit CMakeLists.txt
```

ros_tutorials_action/CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_action)
```

```
find_package(catkin REQUIRED COMPONENTS
message_generation
std_msgs
actionlib_msgs
actionlib
roscpp
)

find_package(Boost REQUIRED COMPONENTS system)

add_action_files(FILES Fibonacci.action)
generate_messages(DEPENDENCIES actionlib_msgs std_msgs)

catkin_package(
LIBRARIES ros_tutorials_action
CATKIN_DEPENDS std_msgs actionlib_msgs actionlib roscpp
DEPENDS Boost
)

include_directories(${catkin_INCLUDE_DIRS} ${Boost_INCLUDE_DIRS})

add_executable(action_server src/action_server.cpp)
add_dependencies(action_server ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
target_link_libraries(action_server ${catkin_LIBRARIES})

add_executable(action_client src/action_client.cpp)
add_dependencies(action_client ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
target_link_libraries(action_client ${catkin_LIBRARIES})
```

7.4.4. 创建动作文件

在CMakeLists.txt文件中加了如下选项。

```
add_action_files(FILES Fibonacci.action)
```

这意味着在构建时包含服务Fibonacci.action，这个服务将用于本次的节点中。由于我们目前还没有创建Fibonacci.action，因此我们按以下顺序创建它。

```
$ roscd ros_tutorials_action          → 移动到功能包目录  
$ mkdir action                         → 在功能包中创建一个名为action的动作目录  
$ cd action                            → 移至创建的action目录  
$ gedit Fibonacci.action              → 创建Fibonacci.action文件并修改内容
```

在动作文件中，三个连字符（---）用作分隔符，第一个是goal消息，第二个是result消息，第三个是feedback消息。goal消息和result消息之间的关系与上述srv文件相同，但主要区别在于feedback消息用于指定进程执行过程中的中间值传输。

```
Fibonacci.action  
  
#goal definition  
int32 order  
---  
#result definition  
int32[] sequence  
---  
#feedback  
int32[] sequence
```

动作的5种基本消息

除了可以在动作文件中找到的目标（goal）、结果（result）和反馈（feedback）之外，动作基本上还使用两个额外的消息：取消（cancel）和状态（status）。取消（cancel）消息使用actionlib_msgs/GoalID，它在动作运行时可以取消动作客户端和单独节点上的动作的执行。状态（status）消息可以根据状态转换¹¹（如PENDING、ACTIVE、PREEMPTED和SUCCEEDED¹²）检查当前动作的状态。

7.4.5. 创建动作服务器节点

在CMakeLists.txt文件中如下添加了生成可执行文件的选项。

```
add_executable(action_server src/action_server.cpp)
```

¹¹ <http://wiki.ros.org/actionlib/DetailedDescription>

¹² http://docs.ros.org/kinetic/api/actionlib_msgs/html/msg/GoalStatus.html

也就是说，是通过构建action_server.cpp文件来创建action_server可执行文件。我们按以下顺序编写一个执行Action Server节点的功能的程序吧。

```
$ roscd ros_tutorials_action/src          → 移动到功能包的源代码目录src  
$ gedit action_server.cpp                 → 创建和编辑源代码文件
```

```
ros_tutorials_action/src/action_server.cpp

#include <ros/ros.h>                      // ROS的基本头文件
#include <actionlib/server/simple_action_server.h> // 动作库头文件
#include <ros_tutorials_action/FibonacciAction.h> // FibonacciAction动作头文件（生成后自动生成）

class FibonacciAction
{
protected:

    // 声明节点句柄
    ros::NodeHandle nh_;

    // 声明动作服务器
    actionlib::SimpleActionServer<ros_tutorials_action::FibonacciAction> as_;

    // 用作动作名称
    std::string action_name_;

    // 声明用于发布的反馈及结果
    ros_tutorials_action::FibonacciFeedback feedback_;
    ros_tutorials_action::FibonacciResult result_;

public:

    // 初始化动作服务器（节点句柄、动作名称、动作后台函数）
    FibonacciAction(std::string name):
        as_(nh_, name, boost::bind(&FibonacciAction::executeCB, this, _1), false),
        action_name_(name)
    {
        as_.start();
    }
}
```

```

~FibonacciAction(void)
{
}

// 接收动作目标 (goal) 消息并执行指定动作 (此处为斐波那契数列) 的函数。
void executeCB(const ros_tutorials_action::FibonacciGoalConstPtr &goal)
{
    ros::Rate r(1); // 循环周期: 1 Hz
    bool success = true; // 用作保存动作的成功或失败的变量
    // 斐波那契数列的初始化设置，也添加了反馈的第一个 (0) 和第二个消息 (1)
    feedback_.sequence.clear();
    feedback_.sequence.push_back(0);
    feedback_.sequence.push_back(1);

    // 将动作名称、目标和斐波那契数列的两个初始值通知给用户
    ROS_INFO("%s: Executing, creating fibonacci sequence of order %i with seeds %i,%i",
action_name_.c_str(), goal->order, feedback_.sequence[0], feedback_.sequence[1]);

    // 动作细节
    for(int i=1; i<=goal->order; i++)
    {
        // 从动作客户端得知动作取消
        if (as_.isPreemptRequested() || !ros::ok())
        {
            ROS_INFO("%s: Preempted", action_name_.c_str()); // 通知动作取消
            as_.setPreempted(); // 取消动作
            success = false; // 看作动作失败并保存到变量
            break;
        }

        // 除非有动作取消或已达成动作目标
        // 将当前斐波纳契数字加上前一个数字的值保存到反馈值。
        feedback_.sequence.push_back(feedback_.sequence[i] + feedback_.sequence[i-1]);
        as_.publishFeedback(feedback_); // 发布反馈。
        r.sleep(); // 按照上面定义的循环周期调用暂歇函数。
    }

    // 如果达到动作目标值，则将当前斐波那契数列作为结果值传输。
    if(success)
    {

```

```
result_.sequence = feedback_.sequence;
ROS_INFO("%s: Succeeded", action_name_.c_str());
as_.setSucceeded(result_);
}

};

int main(int argc, char** argv) // 节点主函数
{
    ros::init(argc, argv, "action_server"); // 初始化节点名称
    FibonacciAction fibonacci("ros_tutorial_action"); // 声明Fibonacci (动作名:ros_tutorial_action)
    ros::spin(); // 等待动作目标
    return 0;
}
```

7.4.6. 创建动作客户端节点

与动作服务器节点一样，客户端节点的设置内容也作为选项加到了CMakeLists.txt文件中。

```
add_executable(action_client src/action_client.cpp)
```

也就是说，通过构建一个名为action_client.cpp的文件来创建action_client可执行文件。我们按照以下顺序编写一个执行动作客户端节点功能的程序。

```
$ roscd ros_tutorials_action/src          → 移动到功能包的源代码目录src
$ gedit action_client.cpp                  → 创建并修改源代码文件
```

```
ros_tutorials_action/src/action_client.cpp
```

```
#include <ros/ros.h>                      // ROS的基本头文件
#include <actionlib/client/simple_action_client.h> // 动作库头文件
#include <actionlib/client/terminal_state.h> // 动作目标状态头文件
#include <ros_tutorials_action/FibonacciAction.h> // FibonacciAction动作头文件 (构建后自动生成)

int main (int argc, char **argv) // 节点主函数
{
    ros::init(argc, argv, "action_client"); // 初始化节点名称
```

```

// 声明动作客户端（动作名称：ros_tutorial_action）
actionlib::SimpleActionClient<ros_tutorials_action::FibonacciAction> ac("ros_tutorial_action",
true);

ROS_INFO("Waiting for action server to start.");
ac.waitForServer(); // 等待动作服务器启动

ROS_INFO("Action server started, sending goal.");
ros_tutorials_action::FibonacciGoal goal; // 声明动作目标
goal.order = 20; // 指定动作目标（进行20次斐波那契运算）
ac.sendGoal(goal); // 发送动作目标

// 设置动作完成时间限制（这里设置为30秒）
bool finished_before_timeout = ac.waitForResult(ros::Duration(30.0));

// 在动作完成时限内收到动作结果值时
if(finished_before_timeout)
{
    // 获取动作目标状态值并将其显示在屏幕上
    actionlib::SimpleClientGoalState state = ac.getState();
    ROS_INFO("Action finished: %s",state.toString().c_str());
}
else
    ROS_INFO("Action did not finish before the time out."); // 超过了动作完成时限的情况
//exit
return 0;
}

```

7.4.7. 构建节点

使用以下命令构建ros_tutorials_action功能包的动作文件、动作服务器节点和动作客户机节点。ros_tutorials_action功能包的源文件位于“~/catkin_ws/src/ros_tutorials_action/src”目录中，而动作文件位于“~/catkin_ws/src/ros_tutorials_action/src/action”目录中。

\$ cd ~/catkin_ws && catkin_make

→ 转到catkin目录并运行catkin构建

7.4.8. 运行动作服务器

我们之前编写的动作服务器在指定动作目标（goal）前不做任何行动，只会等待。因此，执行以下命令时，动作服务器会等待来自动作客户端的目标（goal）指定。运行节点之前不要忘记运行roscore。

```
$ roscore  
$ rosrun ros_tutorials_action action_server
```

如4.3节所述，动作中的动作目标（goal）和结果（result）的用法与服务中的请求和响应类似，这是动作和服务的相似之处。但不同之处在于，动作还有作为中途值的反馈（feedback）消息。这与服务类似，但其消息的实际通信方式与话题非常类似。因此，可以通过rqt_graph和rostopic list命令来查看当前动作消息的使用情况。

```
$ rostopic list  
/ros_tutorial_action/cancel  
/ros_tutorial_action/feedback  
/ros_tutorial_action/goal  
/ros_tutorial_action/result  
/ros_tutorial_action/status  
/rosout  
/rosout_agg
```

如果想了解更多有关每条消息的信息，请将-v选项添加到rostopic list中。这将单独显示发布和订阅的话题，如下所示：

```
$ rostopic list -v  
Published topics:  
* /ros_tutorial_action/feedback [ros_tutorials_action/FibonacciActionFeedback] 1 publisher  
* /ros_tutorial_action/status [actionlib_msgs/GoalStatusArray] 1 publisher  
* /rosout [rosgraph_msgs/Log] 1 publisher  
* /ros_tutorial_action/result [ros_tutorials_action/FibonacciActionResult] 1 publisher  
* /rosout_agg [rosgraph_msgs/Log] 1 publisher  
  
Subscribed topics:  
* /ros_tutorial_action/goal [ros_tutorials_action/FibonacciActionGoal] 1 subscriber  
* /rosout [rosgraph_msgs/Log] 1 subscriber  
* /ros_tutorial_action/cancel [actionlib_msgs/GoalID] 1 subscriber
```

为了查看可视化信息，请使用以下命令。动作消息、动作服务器和客户端之间的关系如图7-8所示，是双向发送和接收。在这里，动作信息由名字ros_tutorial_action/action_topics统一表示，当关闭菜单中的Actions时，可以看到所有5个消息，如图7-9所示。在这里我们可以看到，这个动作由5个话题以及发布和订阅这些话题的节点组成。

```
$ rqt_graph
```

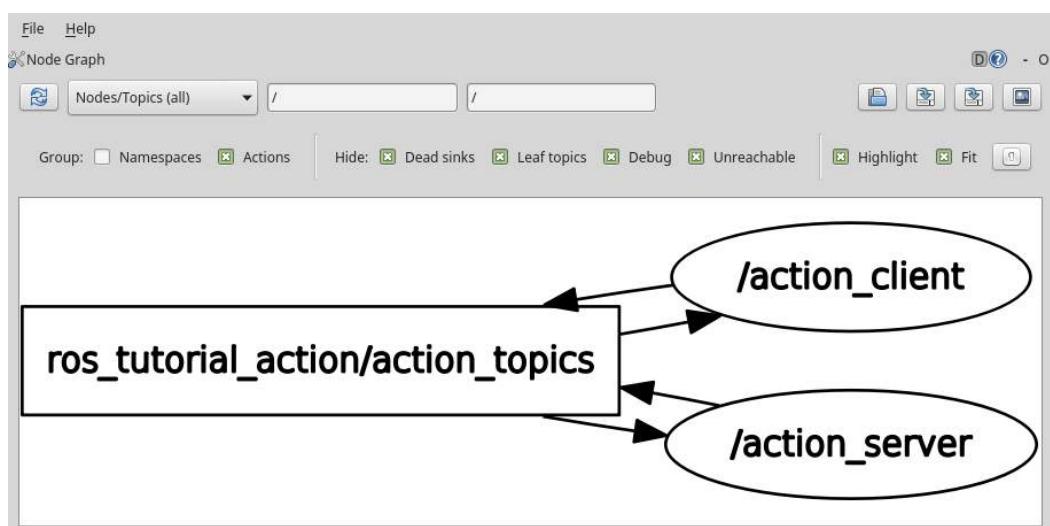


图 7-8 双向收发的动作消息、动作服务器和客户端之间的关系图

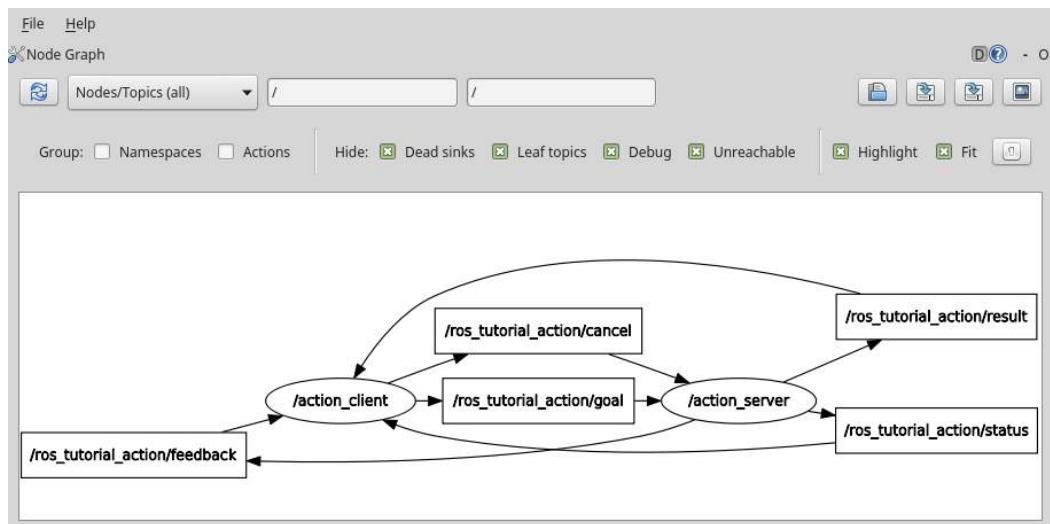


图 7-9 用于动作的5种消息

7.4.9. 运行动作客户端

使用以下命令运行动作客户端。在动作客户端启动的同时，会通过动作目标消息将动作目标设为20。

```
$ rosrun ros_tutorials_action action_client
```

通过设置这个目标值，动作服务器将如下启动斐波那契数列。如果想详细了解中途值或结果值，可以使用像“rostopic echo /ros_tutorial_action/feedback”这样的rostopic命令。

```
$ rosrun ros_tutorials_action action_server
[INFO] [1495764516.294367721]: ros_tutorial_action: Executing, creating fibonacci sequence of order 20
with seeds 0, 1
[INFO] [1495764536.294488991]: ros_tutorial_action: Succeeded
```

```
$ rosrun ros_tutorials_action action_client
[INFO] [1495764515.999158825]: Waiting for action server to start.
[INFO] [1495764516.293575887]: Action server started, sending goal.
[INFO] [1495764536.295139830]: Action finished: SUCCEEDED
```

```
$ rostopic echo /ros_tutorial_action/feedback
header:
  seq: 42
  stamp:
    secs: 1495764700
    nsecs: 413836908
  frame_id: ''
status:
goal_id:
  stamp:
    secs: 1495764698
    nsecs: 413136891
  id: /action_client-1-1495764698.413136891
  status: 1
  text: This goal has been accepted by the simple action server
feedback:
  sequence: [0, 1, 1, 2, 3]
---
```

在本节中，我们创建和运行了动作服务器和客户端节点，了解了节点间的服务通信方法。相关资源可以在下面的github地址找到。

- https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/ros_tutorials_action

如果想马上应用它，读者可以用catkin_ws/src目录中的以下命令来克隆源代码，并运行构建。然后运行action_server和action_client节点即可。

```
$ cd ~/catkin_ws/src  
$ git clone https://github.com/ROBOTIS-GIT/ros_tutorials.git  
$ cd ~/catkin_ws  
$ catkin_make  
  
$ rosrun ros_tutorials_action action_server  
  
$ rosrun ros_tutorials_action action_client
```

7.5. 参数的用法

到目前为止，我们已经多次讲到了对于参数的说明和概念。因此，在本节中，让我们通过实习学会如何使用参数。有关参数的术语，请参阅第4.1节，而rosparam命令请参考第5.4节。

7.5.1. 利用参数创建节点

修改第7.3节中创建的服务服务器和客户端节点中的service_server.cpp，让通过服务请求输入的a和b不光进行加法运算，还会利用参数让a和b进行四则运算。我们按以下顺序修改service_server.cpp源代码。

```
$ roscd ros_tutorials_service/src          → 移至功能包的源代码目录src  
$ gedit service_server.cpp                  → 修改源文件的内容
```

ros_tutorials_service/src/service_server.cpp

```
#include "ros/ros.h"                                // ROS的基本头文件
#include "ros_tutorials_service/SrvTutorial.h"      // SrvTutorial服务头文件

#define PLUS          1                // 加
#define MINUS         2                // 减
#define MULTIPLICATION 3              // 乘
#define DIVISION       4              // 除

int g_operator = PLUS;

// 当有服务请求时，会处理以下内容。
// 服务请求设置为req，服务响应设置为res。
bool calculation(ros_tutorials_service::SrvTutorial::Request &req,
                  ros_tutorials_service::SrvTutorial::Response &res)
{
    // 根据g_operator参数值进行a和b的运算
    // 计算后将结果保存到服务响应值中。
    switch(g_operator)
    {
        case PLUS:
            res.result = req.a + req.b; break;
        case MINUS:
            res.result = req.a - req.b; break;
        case MULTIPLICATION:
            res.result = req.a * req.b; break;
        case DIVISION:
            if(req.b == 0)
            {
                res.result = 0; break;
            }
            else
            {
                res.result = req.a / req.b; break;
            }
        default:
            res.result = req.a + req.b; break;
    }
    // 显示服务请求中使用的a和b值，以及相当于服务响应的result值。
    ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
}
```

```

ROS_INFO("sending back response: [%ld]", (long int)res.result);
return true;
}

int main(int argc, char **argv) // 节点主函数
{
    ros::init(argc, argv, "service_server"); // 初始化节点名称
    ros::NodeHandle nh; // 声明节点句柄
    nh.setParam("calculation_method", PLUS); // 初始化参数
    // 声明服务服务器，创建使用ros_tutorials_service功能包中的SrvTutorial服务文件的
    // 服务服务器service_server。服务名称是"ros_tutorial_srv"，
    // 此服务器当收到服务请求时，会执行一个叫做calculation的函数。
    ros::ServiceServer ros_tutorial_service_server = nh.advertiseService("ros_tutorial_srv",
    calculation);
    ROS_INFO("ready srv server!");
    ros::Rate r(10); // 10hz
    while (1)
    {
        // 将运算符改为通过参数收到的值。
        nh.getParam("calculation_method", g_operator);
        ros::spinOnce(); // 后台函数处理进程
        r.sleep(); // 为了反复进入进程而添加的sleep（暂歇）函数
    }
    return 0;
}

```

他们中的大多数与之前的内容类似，所以我们只看看为了利用参数而添加的部分。特别是，粗体的“setParam”和“getParam”在参数利用中是最重要的函数。但它非常简单，所以只看函数的应用例子，就可以理解它。

7.5.2. 设置参数

下面的代码是将calculate_method参数设置为PLUS值。由于PLUS在7.5.1源中被定义为1，所以calculate_method参数变为1，因此对于通过服务请求收到的值，进行加法运算来做出服务响应。

```
nh.setParam( "calculation_method" , PLUS);
```

作为参考，参数可以设置为integers、floats、boolean、string、dictionaries和list等。例如，1是一个integer，1.0是一个floats，“internetofthings”是一个string，true是一个boolean，[1,2,3]是一个integers的list，a: b和c: d是一个dictionary。

7.5.3. 读取参数

以下是调取calculation_method参数并设置为g_operator的值的部分代码。因此，在7.5.1的源代码中，g_operator每0.1秒检查一次参数的值，并判断对于通过服务请求收到的值进行何种四则运算。

```
nh.getParam( "calculation_method" , g_operator);
```

7.5.4. 构建节点和运行节点

使用以下命令重新构建ros_tutorials_service功能包中的服务服务器节点。

```
$ cd ~/catkin_ws && catkin_make
```

完成后，使用以下命令运行ros_tutorials_service功能包的service_server节点：

```
$ roscore  
$ rosrun ros_tutorials_service service_server  
[INFO] [1495767130.149512649]: ready srv server!
```

7.5.5. 查看参数目录

可以使用“rosparam list”命令查看当前用于ROS网络的参数列表。在显示的列表中，/calculation_method是我们使用的参数。

```
$ rosparam list  
/calculation_method  
/rosdistro  
/rosversion  
/run_id
```

7.5.6. 参数的用例

让我们按照以下命令来设置参数，并观察每次做相同的请求服务但会得到不同的服务处理。

```
$ rosservice call /ros_tutorial_srv 10 5          → 输入要进行四则运算的变量a和b  
result: 15                                         → 默认运算加法的结果值  
$ rosparam set /calculation_method 2              → 减法  
$ rosservice call /ros_tutorial_srv 10 5  
result: 5  
$ rosparam set /calculation_method 3              → 乘法  
$ rosservice call /ros_tutorial_srv 10 5  
result: 50                                         → 除法  
$ rosparam set /calculation_method 4  
$ rosservice call /ros_tutorial_srv 10 5  
result: 2
```

您可以使用“rosparam set”命令更改calculation_method参数。通过更改参数，您可以看到每次做相同的输入“rosservice call /ros_tutorial_srv 10 5”，却得到不同的结果值。通过这种方式，ROS中的参数可以改变节点外部的节点的流程、设置和处理。这是一个非常有用的功能，所以就算现在不使用它，但需要记住。

在本节中，我们讨论了如何修改现有的服务服务器和参数的用法。为了与以前创建的服务源代码区分开来，相关的源代码已经被重命名为ros_tutorials_parameter功能包，并且可以在以下github地址找到。

- https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/ros_tutorials_parameter

如果您想马上应用它，可以用catkin_ws/src目录中的以下命令克隆源代码并运行构建。然后运行service_server和服务_client节点即可。

```
$ cd ~/catkin_ws/src  
$ git clone https://github.com/ROBOTIS-GIT/ros_tutorials.git  
$ cd ~/catkin_ws  
$ catkin_make
```

```
$ rosrun ros_tutorials_parameter service_server_with_parameter
```

```
$ rosrun ros_tutorials_parameter service_client_with_parameter 2 3
```

7.6. roslaunch的用法

如果rosrun是执行一个节点的命令，那么roslaunch可以运行多个节点。除此之外，它还是一种在运行节点时可以附带如下各种选项的ROS命令：修改参数或节点的名称，设置节点的命名空间，设置ROS_ROOT及ROS_PACKAGE_PATH，以及环境变量修改等选项的ROS命令，等。

roslaunch使用“*.launch”文件来设置可执行节点，它基于XML，并提供各个标签的选项。执行命令是“roslaunch [功能包名称] [roslaunch文件]”。

7.6.1. roslaunch的应用

为了解如何使用roslaunch，下面先重命名之前创建的topic_publisher和topic_subscriber节点。只改变名字没有意义，所以让我们运行两个独立的发布者和两个独立的订阅者节点来进行各自的消息通信。

首先，我们编写一个*.launch文件。用于roslaunch的文件具有*.launch文件名，您需要在该功能包目录中创建一个launch目录，并将launch文件放在该目录中。使用以下命令创建一个目录，并创建一个名为union.launch的新文件。

```
$ roscd ros_tutorials_topic  
$ mkdir launch  
$ cd launch  
$ gedit union.launch
```

如下编辑union.launch文件的内容。

```
union.launch  
<launch>  
  <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher1"/>  
  <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber1"/>  
  <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher2"/>  
  <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber2"/>  
</launch>
```

在<launch>标签中，描述了使用roslaunch命令运行节点所需的标签。<node>描述了roslaunch运行的节点。选项包括pkg、type和name。

- pkg 功能包的名称
- type 实际运行的节点的名称（节点名）
- name 与上述type对应的节点被运行时，起的名称（运行名）。一般情况下使用与type相同的名称，但可以根据需要，在运行时更改名称。

如果创建了roslaunch文件，可以如下运行union.launch。请注意，当用roslaunch命令运行多个节点时，运行中的节点的输出（info、error等）不会显示在终端屏幕上，这会使调试变得困难。如果此时添加了--screen选项，终端上运行的所有节点的输出将显示在终端屏幕上。

```
$ rosrun ros_tutorials_topic union.launch --screen
```

运行结果会如何？首先，让我们使用以下命令看看当前正在运行的节点吧。

```
$ rosnode list  
/rosout  
/topic_publisher1  
/topic_publisher2  
/topic_subscriber1  
/topic_subscriber2
```

可以看到，topic_publisher节点已被重命名为topic_publisher1和topic_publisher2，而topic_subscriber节点已被重命名为topic_subscriber1和topic_subscriber2。这四个节点均在运行中。

问题在于，我们通过rqt_graph（图7-10）看到，与“两个发布者节点和两个订阅者节点中的每一个发布者和订阅者都只与一个订阅者和发布者进行单独的通信”的当初的目的不符，两个订阅者都在订阅两个发布者的消息。这是因为我们只是改变了节点的名字，而没有改变要使用的消息的名字。我们用另一个roslaunch命名空间标记来解决这个问题。

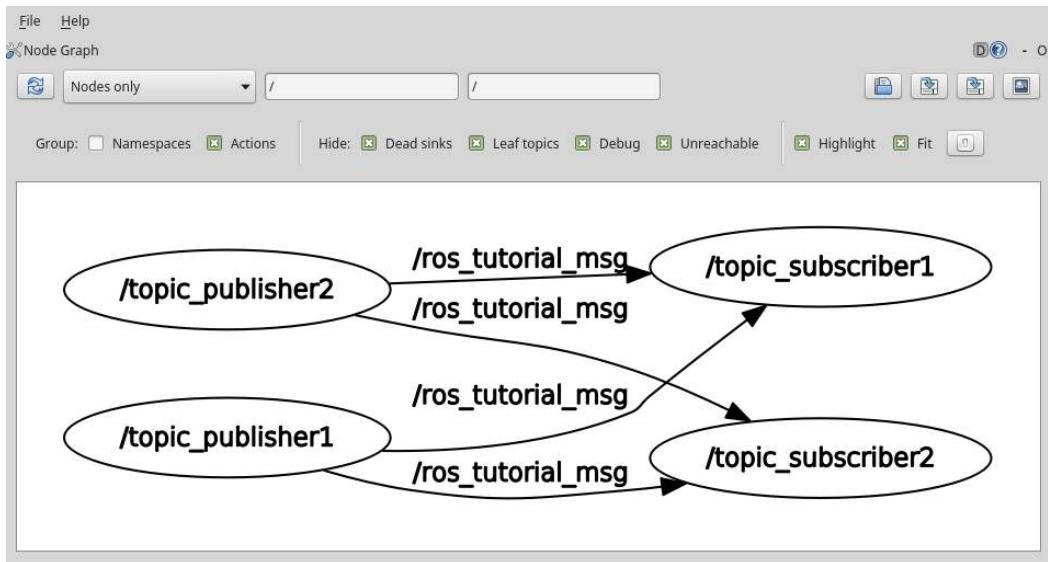


图 7-10 利用roslaunch运行多个节点时的节点图

让我们修改之前创建的union.launch文件，如下所示。

```
$ roscd ros_tutorials_topic/launch
$ gedit union.launch
```

```
ros_tutorials_topic/launch/union.launch
```

```
<launch>
<group ns="ns1">
<node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
<node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber"/>
</group>
<group ns="ns2">
<node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
<node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber"/>
</group>
</launch>
```

<group>是对指定节点进行分组的标签。选项有ns。这是命名空间（name space），是组的名称，属于该组的节点和消息都包含在由ns指定的名称中。

再一次，我们来检查rqt_graph节点之间的连接和消息发送/接收状态。这一次，如图7-11所示，我们可以看到，我们实现了我们最初的目的。



图 7-11 利用命名空间时的消息通信

7.6.2. Launch标签

在launch文件中根据XML¹³的编写方式可以实现多种功能。launch中使用的标签如下所示。

- <launch> 指roslaunch语句的开始和结束。
- <node> 这是对于节点运行的标签。您可以更改功能包、节点名称和执行名称。
- <machine> 可以设置运行该节点的PC的名称、address、ros-root和ros-package-path。
- <include> 您可以加载属于同一个功能包或不同的功能包的另一个launch，并将其作为一个launch文件来运行。
- <remap> 可以更改节点名称、话题名称等等，在节点中用到的ROS变量的名称。

¹³ <http://wiki.ros.org/roslaunch/XML>

- <env> 设置环境变量，如路径和IP（很少使用）。
- <param> 设置参数名称、类型、值等
- <rosparam> 可以像rosparam命令一样，查看和修改load、dump和delete等参数信息。
- <group> 用于分组正在运行的节点。
- <test> 用于测试节点。类似于<node>，但是有可以用于测试的选项。
- <arg> 可以在launch文件中定义一个变量，以便在像下面这样运行时更改参数。

如下面的例子所示，利用其中的参数设置<param>和launch文件中的变量<arg>，可以在运行launch时从外部修改内部变量，因此甚至可以在运行的同时修改节点内部的参数。这是一个非常有用和广泛使用的方法，因此需要掌握。

```
<launch>
<arg name="update_period" default="10" />
<param name="timing" value="${arg update_period}" />
</launch>
```

```
$ roslaunch my_package my_package.launch update_period:=30
```

第8章

机器人、传感器和电机

8.1. 机器人功能包

机器人主要分为硬件和软件。机械、电机、齿轮、电路和传感器被归类为硬件。直接驱动或控制机器人硬件的微控制器级别的固件，以及利用从传感器获得的信息进行识别、制图、导航和动作规划的应用软件均被分类为软件。

ROS属于应用软件的范畴，根据功能，ROS的功能包被分类为机器人功能包¹、专为传感器的传感器功能包²和专为驱动部的电机功能包³。这些功能包由Willow Garage、ROBOTIS、Yujin Robot和Fetch Robotics等机器人公司提供。或者由Open Robotics（曾经是Opensource Robot Foundation，OSRF）、机器人专业的大学实验室和个人开发者开发并发表自己开发的ROS机器人、传感器和电机的相关功能包。

如果要选机器人功能包的代表作，那么绝对是图8-1的PR2和TurtleBot。其中，PR2是负责ROS开发的Willow Garage以科研用机器人为目的开发的移动人形机器人。即使现在，PR2功能包还是具有代表性的机器人功能包，因为其他机器人的核心功能包很多还是使用PR2功能包的衍生包。

虽然PR2的通用性和性能非常优秀，但因为价格过高，无法实现刺激和传播ROS的目的，TurtleBot机器人正是由于这个原因而开发的机器人，旨在普及ROS。第一版Turtlebot是将基于iRobot公司的扫地机器人Roomba的create作为基础而开发的。而TurtleBot2是将韩国的服务机器人公司Yujin Robot的iCLEBO的改进版KOBUKI作为移动的基础而开发的。此外，本书将重点介绍的TurtleBot3是由ROBOTIS、Intel和Open Robotics合作开发的一款基于Dynamixel舵机的移动机器人。在后续的第10章里会介绍与TurtleBot有关的机器人功能包的用法，同时会进行对TurtleBot的更详细的说明。



图 8-1 PR2（左1）、TurtleBot2（左2）和TurtleBot3（右侧的三种机器人）

¹ <http://robots.ros.org/>, <http://wiki.ros.org/Robots>

² <http://wiki.ros.org/Sensors>

³ <http://wiki.ros.org/Motor%20Controller%20Drivers>

除了这两种有代表性的机器人之外，还有180多种机器人的功能包也已公开，如图8-2所示。这是以开源形式的ROS功能包公开的机器人数量，再加上机器人相关的公司、研究所、大学和个人使用的机器人的话数量会更多。

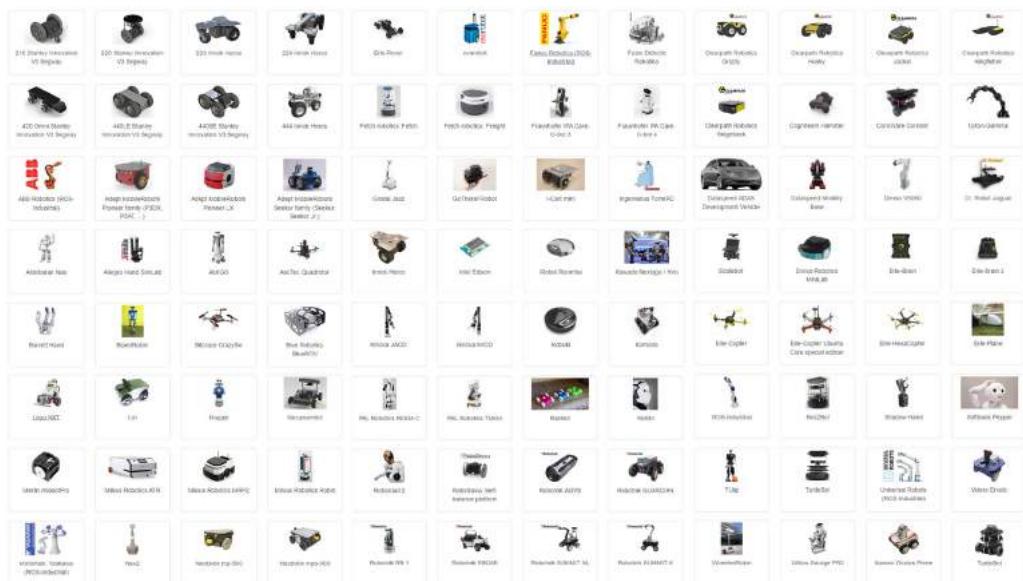


图 8-2 ROS引入的机器人 (<http://robots.ros.org/>)

注册的机器人来自各种领域，如下所示。公共机器人功能包可以在<http://robots.ros.org/>找到。

- 机械手臂 (Manipulator)
 - 移动机器人 (Mobile robot)
 - 自动驾驶汽车 (Autonomous car)
 - 人形机器人 (Humanoid)
 - 无人驾驶飞机 (UAV: Unmanned Aerial Vehicle)
 - 无人潜艇 (UUV: Unmanned Undersea Vehicle)
 - 无人水面艇 (UWV: Unmanned Surface Vehicle)

如果您要使用的机器人功能包是ROS官方功能包，那么安装方法非常简单。首先，请检查您要使用的机器人功能包是否在ROS Wiki (<http://robots.ros.org/>) 上公开可用，或者您可以使用以下命令在所有ROS功能包列表中找到它。

```
$ apt-cache search ros-kinetic
```

还有一种方法是运行Linux的GUI功能包管理器程序synaptic，并搜索单词“ros-kinetic”。如果您想使用的机器人功能包是一个官方的功能包，安装很简单。下面举几个例子。以下命令安装PR2功能包。

```
$ sudo apt-get install ros-kinetic-pr2-desktop
```

以下是安装TurtleBot3功能包的命令。

```
$ sudo apt-get install ros-kinetic-turtlebot3 ros-kinetic-turtlebot3-msgs ros-kinetic-turtlebot3-simulations
```

* 由于TurtleBot3在持续地进行升级，因此推荐您获取最新的源代码，而不是进行二进制文件安装。这在第10章移动机器人中有说明。

即使机器人功能包没有正式提供，也可以按照机器人功能包的维基页面的说明来进行安装。例如，要安装以移动机器人而闻名的Pioneer，可以移动到catkin构建系统的用户源程序目录，并从wiki存储库(repository)中下载最新的机器人功能包。

```
$ cd ~/catkin_ws/src  
$ hg clone http://code.google.com/p/amor-ros-pkg/
```

→ 移动到catkin构建系统的用户源程序目录
→ 从存储库下载

如上所述，机器人功能包可以安装ROS官方功能包，或按照维基中说明的安装方法从开源存储库中下载机器人功能包，然后经过构建过程之后就可以使用。对于功能包中的各节点的使用说明请参照相应机器人功能包的说明。机器人功能包主要包括机器人驱动节点、获取安装上的传感器的数据的节点、应用安装上的传感器的数据的节点，以及远程控制节点。如果机器人是多关节机器人，则还包括逆运动学节点，如果是移动机器人则包括导航节点。

8.2. 传感器功能包

传感器是与机器人无法分离的。有许多研究从传感器数据提供的无数环境信息中提取有意义的信息，或利用这些信息认识环境并传输给机器人。这样的环境信息有位置、空间、天气、声音、惯性、振动、气体、电流量、RFID，物体和外力识别等很多种。该信息被用作机器人执行实际任务的重要数据。

在制作机器人时，通过驱动轮或机器人手臂让机器人移动，且通过智能手机等进行遥控并不意味着开发完成。如果您在这里停下来，只能说您搭建了一个移动的机器。机器人只有在自己认识到周围环境，只提取有意义的信息，并能够计划和做出思考和判断，才能被视为机器人。这就是为什么传感器很重要。

8.2.1. 传感器的类型

环境信息有很多种类，而传感器的种类也不亚于此。其中，机器人使用的典型传感器有距离传感器。比较常见的是红外线传感器和以激光为基础的多种激光距离传感器。激光距离传感器有LDS（Laser Distance Sensor，激光距离传感器）、LiDAR（Light Detection And Ranging，光检测和测距）和LRF（Laser Range Finders，激光测距仪）。最近，三维距离传感器RealSense、Kinect和Xtion也广泛地被用作距离传感器。另外还有用于识别用户或物体的彩色照相机、用于位置估计的惯性传感器、用于语音识别的麦克风以及用于转矩控制的转矩传感器等处理各种信息的多种传感器。

问题是，如图8-3所示，有太多的传感器可以使用。而在微处理器中以ADC（模拟数字转换器）方式接收数据的传感器是有限的。其中，LDS、3D传感器、相机等传感器有大量需要处理的信息，处理起来需要较高的配置，因此无法用微处理器实现，需要用PC。因此需要驱动程序，还需要如OpenNI和OpenCV的Point Cloud处理，以及图像处理所需的库。

ROS提供了可以使用上述传感器的驱动程序和库的开发环境。目前，还并不是提供所有的传感器功能包，但传感器功能包正变得越来越多。而且功能相同但通信方式不同的传感器们的用法也趋向于统一。传感器制造商正在积极支持ROS传感器功能包，这将加速ROS对未来传感器的支持。



图 8-3 ROS中可用的传感器示例

8.2.2. 传感器功能包的分类

在ROS传感器wiki页面⁴上公开着多种传感器功能包。这里按照种类将传感器分为1D range finders、2D range finders、3D Sensors、Pose Estimation(GPS+IMU)、Cameras、Sensor Interfaces、Audio/Speech Recognition、Environmental、Force/Torque/Touch Sensors、Motion Capture、Power Supply和RFID等，并介绍属于每个类别的传感器。有关传感器功能包的更多详细用法，请参阅上述ROS传感器wiki页面。以下是笔者尤其重视的功能包。

- 1D Range Finders 可用于制作低成本机器人的红外线方式的直线距离传感器。
- 2D Range Finders 亦被称为LDS，是常用于导航的传感器。
- 3D Sensors 有英特尔的RealSense、微软的Kinect和华硕的Xtion，以及各种3D测量所需的传感器。
- Audio/Speech Recognition 目前，与语音识别相关的部分很少，但估计会不断增加。
- Cameras 这里收集了广泛用于物体识别、人脸识别和字符识别的相机驱动程序和各种应用功能包。
- Sensor Interfaces 很少有传感器支持USB和Web协议。仍然有许多传感器可以很容易地从微处理器获取信息。这些传感器支持微处理器的UART和微型PC中的ROS接口。下面将介绍这些接口。

⁴ <http://wiki.ros.org/Sensors>

这里公开了各种传感器功能包，读者可以找到适合自己的项目的传感器，并将其应用到自己的项目。最常用的相机（camera）、深度相机（depth camera）和激光距离传感器（LDS）将在下面的章节中详细讨论。

8.3. 相机

相机相当于机器人的眼睛。从相机获得的图像对于识别机器人周围的环境非常有用。例如，利用相机图像的对象识别和脸部识别；使用两台相机（立体相机）从两个不同图像之间的差异获得的距离值；利用距离值生成3维地图的Visual-SLAM；单眼相机Visual-SLAM；利用从彩色图像获得的颜色信息的颜色识别；跟踪特定对象的对象跟踪。

这些场合中用到的相机种类非常多，本节中将用USB摄像头来进行说明。USB摄像头意味着它是支持USB的视频录制设备。另一个名称是USB video device class（UVC）⁵。所以，官方的名字是“UVC相机”，但本节中将其称为常用的“USB摄像头”。

截至2017年7月，UVC发布到1.5版⁶。UVC 1.5版本支持最新的USB 3.0，可在几乎所有的操作系统上使用，包括Linux、Windows和OS X。它比其他相机易于使用、要求高、价格便宜。在本章中，我们将实习运行USB摄像头的操作和检查数据的操作。



相机接口

相机的接口并不只有USB。某些相机具有可连接到网络的功能。通常连接到局域网或WiFi，将视频数据以视频流形式传输到网络。这些相机应该被称为网络摄像头。此外，有些摄像机使用FireWire(IEEE 1394接口)进行高速传输，主要用于需要高速传输图像的研究目的。FireWire标准在大多数常见的电路板上无法找到，但它是由苹果公司开发的，因此主要用于苹果产品。

⁵ https://en.wikipedia.org/wiki/USB_video_device_class

⁶ http://www.usb.org/developers/docs/devclass_docs/

8.3.1. USB摄像头相关功能包

ROS提供了与USB摄像头相关的各种功能包。有关更多信息，请参阅ROS Wiki的“传感器/摄像机”类别 (<http://wiki.ros.org/Sensors/Cameras>)。在此让我们看看几种功能包。

- libuvc-camera 这是用于采用UVC标准的相机的接口功能包。（开发者：Ken Tossell）
- uvc-camera 因为有相对详细的相机设置功能，所以非常方便。此外，如果您因为有两个相机，所以考虑使用立体相机，那么这将是一个比较合适的功能包。
- usb-cam 这是Bosch使用的非常简单的摄像头驱动程序。（开发者：Benjamin Pitzer）
- freenect-camera, openni-camera, openni2-camera 所有这三个功能包名称中都有相机，但它们都是深度相机（如Kinect或Xtion）的功能包。这些传感器也被称为RGB-D相机，因为它们也包含彩色相机。如果要利用彩色图像，则需要使用这些功能包。
- camera1394 它是使用FireWire（IEEE 1394接口）的相机的驱动程序。
- prosilica-camera 它被用于AVS的prosilica相机，它被广泛用于研究目的。
- pointgrey-camera-driver 它是Point Grey Research公司的Point Gray相机的一个驱动程序，被广泛用于科研。
- camera-calibration James Bowman和Patrick Mihelich开发了一个应用了OpenCV的校准功能的相机校准功能包。许多相机相关的功能包需要这个功能包。

8.3.2. USB摄像头测试

在本节中，我们来使用Ken Tossell发布的uvc-camera⁷。它是最常用的USB摄像头功能包。其他相关功能包的用法类似，所以如果您想使用另一个功能包，请检查本功能包的wiki页面。

- USB摄像头：将准备好的USB摄像头连接到电脑的USB端口。
- 相机连接信息：打开一个新的终端窗口，并按如下所示使用“lsusb”命令检查连接是否正确。如果您有一个通用的UVC系统，则可以检查摄像机是否连接成带下划线的消息。

⁷ http://wiki.ros.org/uvc_camera

```
$ lsusb
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 002: ID 2109:0812 VIA Labs, Inc. VL812 Hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 005: ID 046d:c52b Logitech, Inc. Unifying Receiver
Bus 001 Device 006: ID 05e3:0608 Genesys Logic, Inc. Hub
Bus 001 Device 013: ID 046d:08ce Logitech, Inc. QuickCam Pro 5000
Bus 001 Device 012: ID 0c45:7603 Microdia
Bus 001 Device 002: ID 2109:2812 VIA Labs, Inc. VL812 Hub
Bus 001 Device 007: ID 8087:0a2a Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

安装uvc camera功能包

```
$ sudo apt-get install ros-kinetic-uvc-camera
```

安装image相关功能包

```
$ sudo apt-get install ros-kinetic-image-*
$ sudo apt-get install ros-kinetic-rqt-image-view
```

运行uvc_camera节点

如果您运行uvc_camera节点，将收到关于相机校准的警告，例如 “[WARN] [1423194481.257752159]: Camera calibration file /home/xxx/.ros/camera_info/camera.yaml not found.”。可以先忽略它。在下一节中，我们将详细介绍校准。

```
$ roscore
$ rosrn uvc_camera uvc_camera_node
```

查看话题消息

以下话题消息显示正在发布相机信息 (/camera_info) 和图像信息 (/image_raw)。

```
$ rostopic list
/camera_info
/image_raw
/image_raw/compressed
/image_raw/compressed/parameter_descriptions
/image_raw/compressed/parameter_updates
/image_raw/compressedDepth
/image_raw/compressedDepth/parameter_descriptions
/image_raw/compressedDepth/parameter_updates
/image_raw/theora
/image_raw/theora/parameter_descriptions
/image_raw/theora/parameter_updates
/rosout
/rosout_agg
```

8.3.3. 查看图像信息

在上一节中，我们确认过在执行uvc_camera_node节点时会发布图像信息。在本节中，将使用可视化工具image_view和RViz实际地查看图像信息。如果在这里没有看到图像，则说明摄像头驱动程序或连接有问题，则请转到下一节查看是否有这些问题。

使用image_view节点查看图像

首先，我们运行image_view节点来查看图像信息。最后面的附加选项“image:=/image_raw”是把话题列表中的话题以图像形式查看的选项。运行下面命令后，摄像头图像将在一个小窗口中显示，如图8-4所示。

```
$ rosrun image_view image_view image:=/image_raw
```



图 8-4 利用image_view节点查看图像视图

用rqt_image_view节点检查

我们来利用6.2节中介绍的rqt_image_view。rqt_image_view中有GUI元素的rqt插件image_view。运行rqt_image_view节点将显示如图8-5所示的图像。与image_view不同，您还可以在运行后从图像查看器GUI中选择一个话题。

```
$ rqt_image_view image:=/image_raw
```

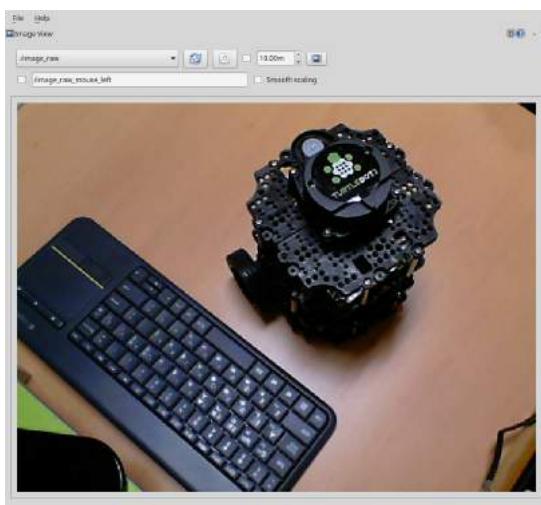


图 8-5 利用rqt_image_view节点查看图像

用RViz查看

我们运行一个可视化工具RViz。有关RViz的详细说明，请参见第6.1节。

```
$ rviz
```

RViz运行后，先更改“Displays”选项。单击RViz左下方的[Add]，在[By display type]选项卡中选择[Image]，以此加载图像显示功能，如图8-6所示。

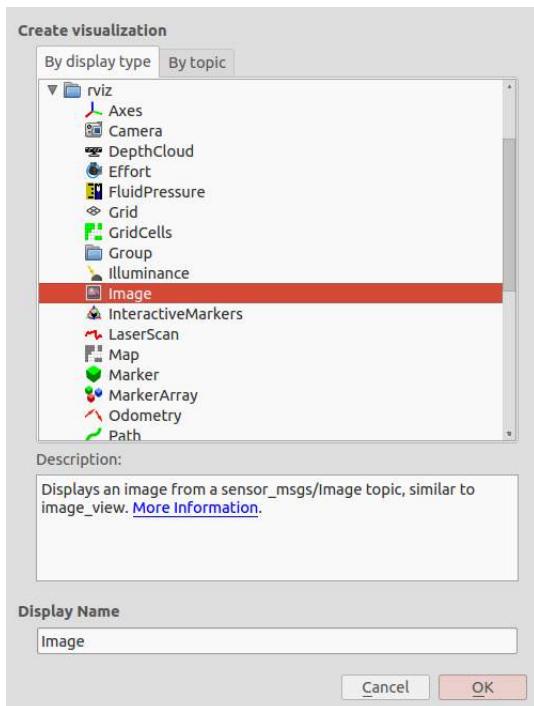


图8-6 将Image Display添加到RViz

然后将[Image] → [Image Topic]的值更改为“/image_raw”。则会如图8-7所示显示图像。如果图像看起来很小，则可以用鼠标调整该image视图的边缘，以增加视图尺寸。

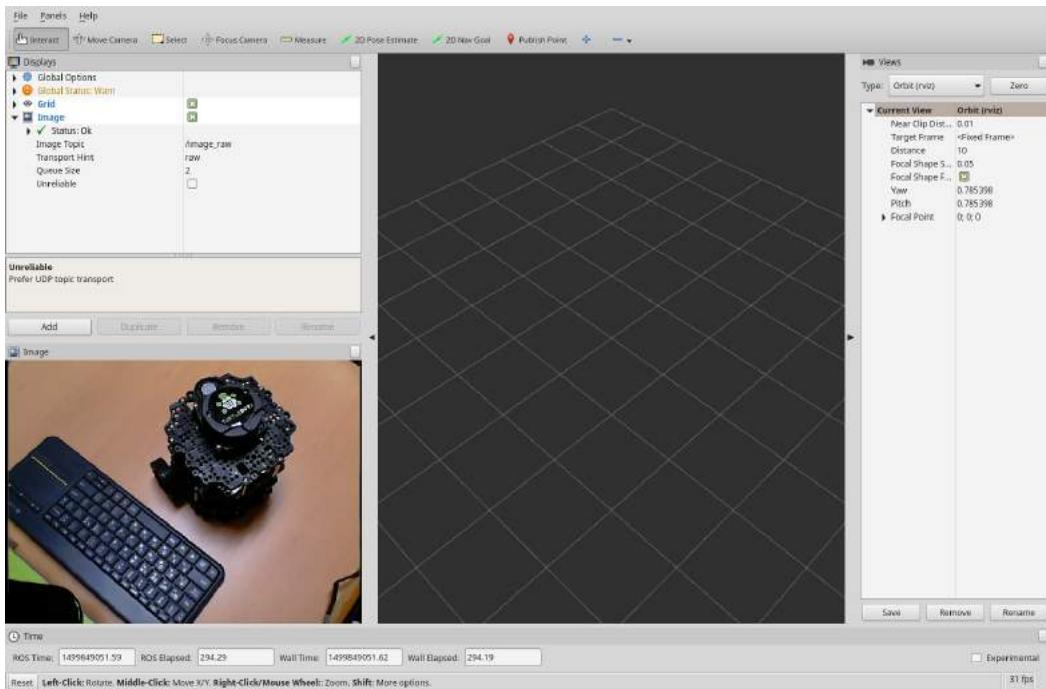


图 8-7 使用RViz查看图像

8.3.4. 远程传输图像

在前一节中，我们在一台计算机上连接USB摄像头并查看了图像。但是，由于机器人在移动，所以安装在机器人上的相机在跟随机器人移动时无法看到图像。在本节中，我将解释从另一台远程计算机查看安装在机器人上的相机的图像信息的方法。请务必掌握下述说明并亲自操作。

连接了相机的计算机

ROS主节点可以在任何一台计算机上运行，但是在这个例子中，将一台连有相机的计算机用作运行ROS主节点的计算机。您需要做的第一件事就是修改网络变量，比如ROS_MASTER_URI和ROS_HOSTNAME。首先，使用类似gedit的文档编辑程序（sublime text、vim、emacs和nano），使用以下命令打开bashrc文件。

```
$ gedit ~/.bashrc
```

加载bashrc文件后可以发现已经有很多设置。保持现有的设置不变，到bashrc文件的底部，并按如下所示修改ROS_MASTER_URI和ROS_HOSTNAME变量。请注意，以下示例中的IP地址（192.168.1.100是相机所连接的计算机的IP地址）仅为示例。需要将它修改为自己的IP。检查IP的命令是ifconfig，这在上面的3.2节中描述。

```
export ROS_MASTER_URI=http://192.168.1.100:11311  
export ROS_HOSTNAME=192.168.1.100
```

然后运行roscore，并在另一个终端窗口中运行uvc_camera_node节点

```
$ roscore  
$ rosrun uvc_camera uvc_camera_node
```

远程计算机

同样地，在远程计算机上，打开bashrc文件并修改ROS_MASTER_URI和ROS_HOSTNAME变量。将ROS_MASTER_URI设置为连有相机的计算机的IP，并将ROS_HOSTNAME变量更改为远程计算机本身的IP（192.168.1.120是远程计算机的IP）。在这里，192.168.1.120也是一个示例。需要用ifconfig检查自己的IP，并记录到bashrc文件。然后只运行image_view。

```
export ROS_MASTER_URI=http://192.168.1.100:11311  
export ROS_HOSTNAME=192.168.1.120
```

```
$ rosrun image_view image_view image:=/image_raw
```

在本节中，我们介绍了如何在远程的另一台计算机上检查安装在机器人上的摄像机的图像信息。由于可以远程查看机器人周围环境，所以可以用作遥感机器人、视频会议机器人或者可以将图像信息实时地传输到网络的网络摄像机（即监控系统）。

8.3.5. 相机校准

当运行uvc_camera节点时，会得到关于相机校准的警告，例如 “[WARN] [1423194481.257752159]: Camera calibration file /home/xxx/.ros/camera_info/”。

camera.yaml not found.”。这可以忽略。但是，当您使用立体相机或用图像测量距离值，或在处理图像（如物体识别）时，需要进行校准。

为了从相机图像信息获得准确的距离信息，则需要多种附加信息，如每台相机各不相同的镜头特性、镜头与图像传感器之间的距离以及扭转角度等信息。这是因为相机是一个将我们生活的三维空间世界投影到二维空间的图像投影设备，所以在投影过程中，由于每台相机的固有特性，这些参数都会不同。

例如，每台相机的镜头和图像传感器彼此不同，并且由于相机的硬件结构不同，镜头和图像传感器之间的距离也各不相同。并且，在相机制作过程中，镜头和图像传感器必须水平组装，但由于细微的偏差，图像中心（image center）与主点（principal point）会有细微的偏离，且图像传感器的倾度也会略有差异。

校准这些部件的过程称为相机校准（Calibration），目的是查找相机的固有参数。摄像机的校准是非常重要的，但在本书中很难处理，所以请参考OpenCV图像处理书籍。

ROS提供的是利用OpenCV相机校准的校准功能包⁸。接下来，按照以下章节所述校准相机。

安装相机校准功能包

如下所示安装相机校准功能包并运行uvc_camera_node节点。

```
$ sudo apt-get install ros-kinetic-camera-calibration  
$ rosrn uvc_camera uvc_camera_node
```

接下来，我们来检查当前的相机信息。由于目前还没有关于相机校准的信息，因此将全部显示为默认值。

```
$ rostopic echo /camera_info  
header:  
seq: 7609  
stamp:  
secs: 1499873386  
nsecs: 558678149
```

⁸ http://wiki.ros.org/camera_calibration

```
frame_id: camera
height: 480
width: 640
distortion_model: ''
D: []
K: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
R: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
P: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
binning_x: 0
binning_y: 0
roi:
  x_offset: 0
  y_offset: 0
  height: 0
  width: 0
do_rectify: False
---
```

准备棋盘

摄像机的校准是以一个由黑白方块组成的棋盘为基准进行的，如图8-8所示。从下面的地址下载8x6国际象棋棋盘，并打印出来后将其贴到一个平坦的纸箱。有时也会打印成超过1米的棋盘，但这里用的是A4纸。作为参考，8x6棋盘横向有9个方块，所以有8个交叉点，而竖向有7个方块，有6个交叉点，所以它被称为8x6棋盘。

- http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration?action=AttachFile&do=view&target=check-108.pdf

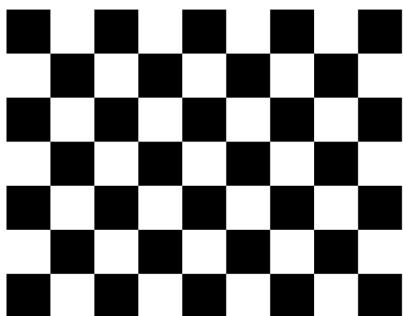


图 8-8 用于校准的国际象棋棋盘 (8×6)

校准

下面进行相机校准。--size是上述棋盘的宽度和高度，--square 0.024是棋盘的一个小方格的实际尺寸。这个小方块是长宽相同的正方形，但是打印出来的尺寸可能会各不相同。笔者打印出来是24毫米，所以下面命令中写了0.024。

```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.024 image:=/image_raw camera:=/camera
```

校准节点运行后会运行GUI，如图8-9所示。此时，如果用相机对准棋盘，校准将立即开始。在GUI屏幕的右侧，可以看到一个标有X、Y、Size和Skew的条形控件。这是校准的进展状态，都以绿色填满意味着校准完成。在校准过程中需要将棋盘对着相机朝着左/右/上/下/前/后移动，还需要倾斜棋盘。

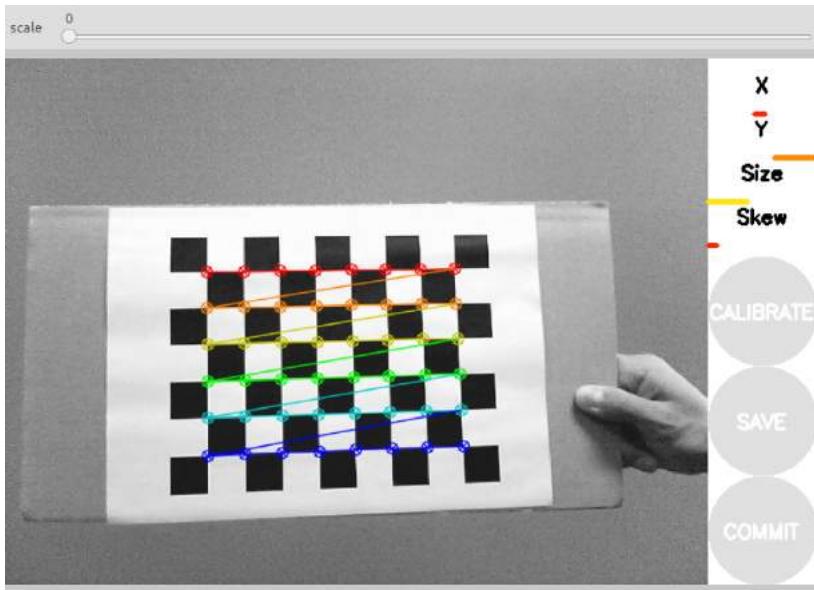


图8-9 校准GUI的初始状态

如图8-10所示，校准所需的所有图像都记录下来之后，CALIBRATE按钮会被激活。点击这个按钮后会进行实际的校准计算，这需要大约1到5分钟。计算完成后，点击SAVE按钮保存校准信息。存储的地址显示在执行校准的终端窗口中，是存储在某一个/tmp目录（如“/tmp/calibrationdata.tar.gz”）中。

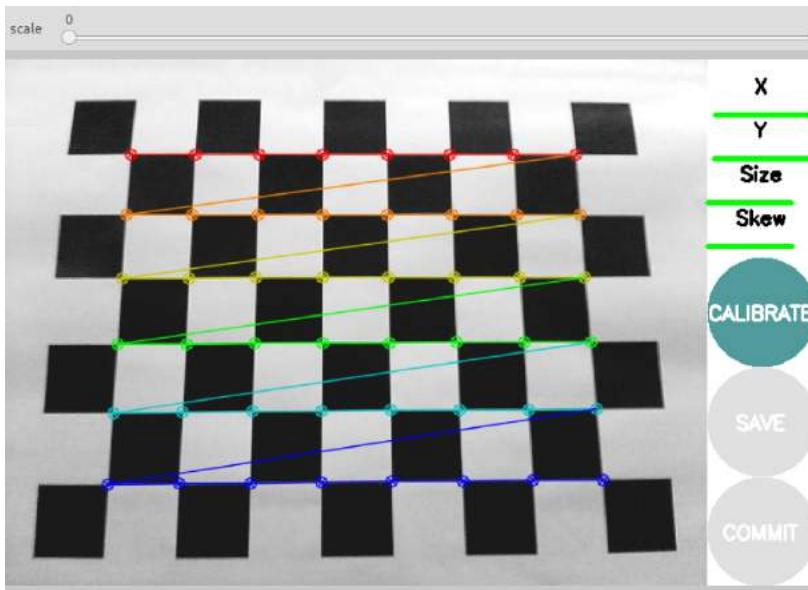


图8-10 利用校准GUI进行的校准过程

创建相机参数文件

下面，我们来创建一个包含相机校准参数的相机参数文件（camera.yaml）。如下例所示，解压缩calibrationdata.tar.gz文件以查看图像文件 (*.png) 和记录了校准中使用的校准参数的ost.txt文件。

```
$ cd /tmp  
$ tar -xvf calibrationdata.tar.gz
```

接下来，将ost.txt文件改名为ost.ini，并使用camera_calibration_parsers功能包的convert节点创建相机参数文件（camera.yaml）。创建完成后，将其保存在~/.ros/camera_info/目录中，则ROS中使用的相机相关功能包会引用此信息。

```
$ mv ost.txt ost.ini  
$ rosrun camera_calibration_parsers convert ost.ini camera.yaml  
$ mkdir ~/.ros/camera_info  
$ mv camera.yaml ~/.ros/camera_info/
```

接下来，打开camera.yaml文件后可以看到如下设置内容。其中，可以将相机名称（camera_name）改为任意值。一般来说，与相机相关的功能包通常被称为carmera，所以笔者将相机名称从原来的narrow_stereo改为camera来使用。

~/.ros/camera_info/camera.yaml

```
image_width: 640
image_height: 480
camera_name: camera
camera_matrix:
  rows: 3
  cols: 3
  data: [778.887262, 0, 302.058565, 0, 779.885146, 221.545303, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.195718, -0.419555, -0.002234, -0.016098, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [794.464417, 0, 294.819501, 0, 0, 805.005371, 220.404173, 0, 0, 0, 1, 0]
```

在这个camera.yaml中，记录着相机内部矩阵camera_matrix、失真系数distortion_coefficients、立体相机的整流矩阵rectification_matrix和投影矩阵projection matrix。各个参数的含义在“http://wiki.ros.org/image_pipeline/CameraInfo”中有详细的描述。最后再次运行uvc_camera_node节点。这一次，确保您目前没有出现关于校准文件的警告。

```
$ rosrun uvc_camera uvc_camera_node
[INFO] [1499873830.472050095]: using default calibration URL
[INFO] [1499873830.472116471]: camera calibration URL:
file:///home/xxx/.ros/camera_info/camera.yaml
```

另外，如果查看/camera_info话题，则可以看到D、K、R和P参数已被填充，如以下示例所示。

```
$ rostopic echo /camera_info
header:
seq: 2213
stamp:
  secs: 1499874042
  nsecs: 898227060
frame_id: camera
height: 480
width: 640
distortion_model: plumb_bob
D: [0.195718, -0.419555, -0.002234, -0.016098, 0.0]
K: [778.887262, 0.0, 302.058565, 0.0, 779.885146, 221.545303, 0.0, 0.0, 1.0]
R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P: [794.464417, 0.0, 294.819501, 0.0, 0.0, 805.005371, 220.404173, 0.0, 0.0, 0.0, 1.0, 0.0]
binning_x: 0
binning_y: 0
roi:
  x_offset: 0
  y_offset: 0
  height: 0
  width: 0
do_rectify: False
---
```

8.4. 深度相机 (Depth Camera)

Depth Camera（深度相机）有多种名称，在类似LDS（laser distance sensor，激光距离传感器）的范畴内被称为Depth sensor，可以获得彩色图像时也被称为RGB-D camera，而微软公司成功普及的深度相机被称为Kinect Camera。在本节中，为了防止产生混淆，将其统一称为Depth camera。

8.4.1. Depth Camera的类型

根据获取信息的方法，Depth camera可以被分成多种类型，诸如ToF（Time of flight、飞行时间）⁹、结构光（Structured Light）¹⁰和立体（Stereo）¹¹方法、等。

ToF (Time of Flight)

ToF方法是发送红外线后利用返回所需的时间测量距离。通常，IR发光部和收光部是成对的（例如使用红外相机的产品。但在另一些产品中却不是如此），并读取由每个像素测量的距离。ToF方法比后面将介绍的利用相干辐射模式的结构光方式更昂贵的原因是这种结构方面的原因提高了硬件的价格（最近，引入了使用相位差的距离计算方法，因此价格在下降）。

采用ToF方式的传感器有Panasonic的D-IMAGER、MESA Imaging的SwissRanger、Fotonic的FOTONIC-B70、pmdtechnologies的CamCube和CamBoard、SoftKinetic的DepthSense DS系列以及微软最新发布的Kinect 2。

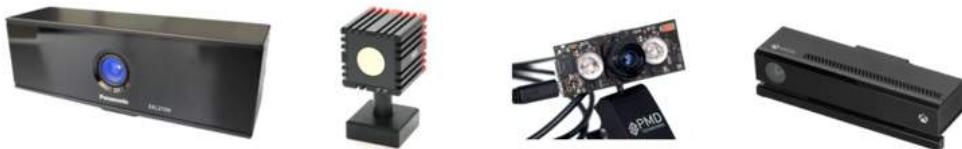


图 8-11 从左边起，D-IMager、SwissRanger、CamBoard和Kinect2

结构光 (Structured Light)

结构光方式的代表性产品是微软的Kinect和华硕的Xtion，它们使用相干辐射模式（pattern of coherent radiation，利用US20100225746专利）。此外，还有PrimeSense的Carmine和Capri以及最近的Occipital的Structure Sensor。这些传感器的共同点是都使用PrimeSense公司的PrimeSense片上系统（SoC）。

⁹ https://en.wikipedia.org/wiki/Time-of-flight_camera

¹⁰ https://en.wikipedia.org/wiki/Structured-light_3D_scanner

¹¹ https://en.wikipedia.org/wiki/Range_imaging



图8-12 从左边起，Kinect、Xtion、Carmine和Structure Sensor

使用PrimeSense公司的PrimeSense SoC的Depth Camera是一款由一个红外投影仪和一个红外相机组成的传感器，它使用了现有的ToF方法中从未使用的相干辐射模式。该技术解决了现有ToF方式的硬件昂贵的问题和外部干扰等问题，因此备受关注。而且搭载了PrimeSense公司的PrimeSense SoC的Carmine和Capri也面世了。另外，采用了相同SoC芯片的微软Kinect成为了Xbox的控制器之后人气飞涨。也因为有这款SoC，华硕也能推出面向普通PC用户的Xtion了。这些都是配备了PrimeSense SoC的传感器。

但是，苹果在2013年12月收购PrimeSense时出现了问题。PrimeSense的Carmine和Capri产品已经不再可用，而且微软的Kinect也停产，而华硕的Xtion也即将停产（库存除外）。Occipital公司的Structure Sensor是采用PrimeSense SoC的最后一款产品，目前是将此产品作为苹果的附件出售，但无法知道未来会发生什么。以低价流行的产品已经隐藏在历史中。

立体 (Stereo) 相机

作为Depth Camera类型之一的立体相机（见图8-13）是比前两种类型研究了更长时间的相机，其距离是使用双眼视差来计算的，如人的左眼和右眼。顾名思义，立体相机配置了相隔一定距离的两个图像传感器，并利用这两个图像传感器捕获的两幅图像之间的差异来计算距离值。代表性的产品包括Point Grey的Bumblebee相机和韩国InRobot公司的OjOcamStereo。

立体照相机的类型很多，最近脱颖而出的有两种，分别是双红外图像传感器，以及内置一个红外线投影仪和两个图形传感器的传感器。其中后一种利用红外线投影仪以一定的模式发射肉眼看不见的红外线，并且用两个图形传感器接收之后通过三角测量法计算出距离。为了与上述一般的立体相机区分开，将前者称为无源（passive）立体相机，而后者也称为有源（active）立体相机。后一种类型的相机的主要产品是Intel的RealSense。R200型号是约100美元左右，在目前的Depth Camera产品中是最便宜的产品，而且尺寸

小，性能也和前面介绍的Xtion类似。此外，被认为是RealSense的下一代产品的D400系列尽管属于低价产品，但因为小尺寸、广视角、野外可用、测距范围提升等原因，在机器人工程领域广为所用。



图 8-13 从左边起，Bumblebee、OjOcamStereo 和 RealSense

8.4.2. Depth Camera 测试

为了进行安装和运行Depth camera的驱动程序，本节将使用英特尔的RealSense R200。

安装与RealSense相关的功能包

下载并安装与RealSense相关的驱动程序和可执行功能包。

```
$ sudo apt-get install ros-kinetic-librealsense ros-kinetic-realsense-camera
```

运行r200_nodelet_default启动文件

运行realsense_camera功能包中的r200_nodelet_default.launch文件。

```
$ roscore
```

```
$ roslaunch realsense_camera r200_nodelet_default.launch
```

如上所述运行后，如果功能包未能安装或工作异常，则有时可能需要对不同的Linux内核进行不同的设置。这在下面的维基地址详细描述。

- <http://wiki.ros.org/librealsense>

8.4.3. Point Cloud Data（点云数据）的可视化

Depth Camera与对象物体的三维空间距离被表示为一个点，而这些点的集合体与云相似，因此这些数据被称为Point Cloud Data（点云数据）。下面为了在GUI环境中检查点云数据，运行RViz并按以下顺序更改显示选项。

- ① 将[Global Options] → [Fixed Frame]更改为“camera_depth_frame”。
- ② 单击RViz左下角的[Add]按钮，然后选择[PointCloud2]来添加它。在详细设置中将Topic设为camera/depth/points，然后选择所需形状的大小和颜色。
- ③ 完成所有设置后，可以看到PCD值，如图8-14所示。由于颜色基准设为X轴，因此离X轴越远，越接近紫色。

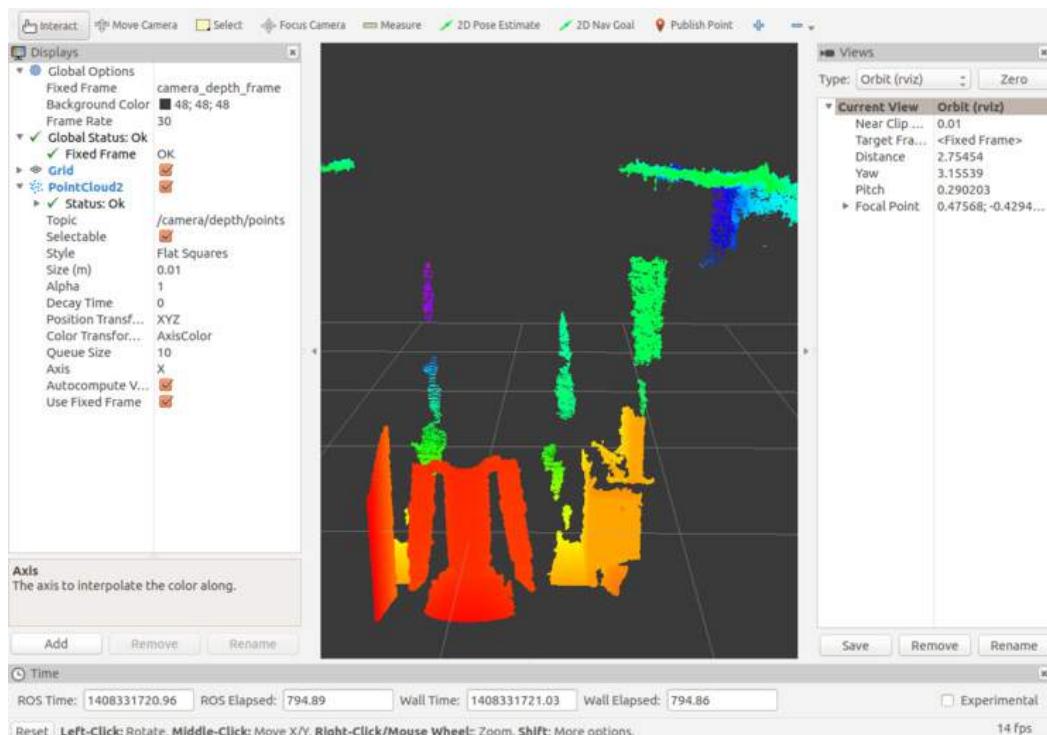


图 8-14 从RViz的PointCloud2显示屏观察到的点云数据

如果使用的是其他深度相机，请查看以下地址的wiki，了解各种相机的操作方法和功能包的用法。

- http://wiki.ros.org/Sensors#A3D_Sensors_.28range_finders_.26_RGB-D_cameras.29

8.4.4. Point Cloud Data相关库

Point Cloud Library

Depth Sensor根据获取信息的方法的不同，分为LDS和Depth Camera，这个类别的所有距离传感器都将与物体相距的距离显示为一个点，并处理点的集合Point Cloud。作为使用该点云的API的集合，用的最多的是叫做PCL（Point Cloud Library）¹²的库，功能包括滤波、分割、表面重构、用模型拟合或提取特征，等。

OpenNI

OpenNI（Open Natural Interaction，开放自然交互）¹³是以PrimeSense公司为中心，与Willow Garage和ASUS一起，为了使用这几家公司的产品而开发的驱动程序和多种API库。在这里，NI（Natural Interaction，自然交互）是指人与机器之间的交流，这个词意味着这个交流基于人的感觉而非键盘和鼠标的交互。带有PrimeSense SoC的大多数传感器都使用此驱动程序。

类似的形式有微软的Kinect Windows SDK和Libfreenect，后者曾经是第一次破解Kinect并免费发布的驱动程序。除了Ponit Cluud Data的基本驱动程序之外，OpenNI还包括处理人体骨架的中间件，如NITE。PrimeSense被苹果公司收购后，OpenNI一度被置于废弃的边缘，但现在Occipital公司在其Github存储库¹⁴中提供OpenNI¹⁵。

8.5. 激光距离传感器

激光距离传感器（Laser Distance Sensor，LDS）有多种名称，比如激光雷达（LIDAR）、激光测距仪（Laser Range Finder，LRF）和激光扫描仪（Laser Scanner）。LDS是利用激光光源来测量与物体的距离的传感器。LDS传感器具有高性能、高速度和实时数据采集的优点，因此在距离测量方面有着广泛的应用。由于这些优点，它是在机器人领域被广泛使用的传感器，比如用于使用距离传感器的SLAM

¹² <http://pointclouds.org/>

¹³ <http://en.wikipedia.org/wiki/OpenNI>

¹⁴ <https://github.com/occipital/openni2>

¹⁵ <https://structure.io/openni>

(Simultaneous Localization and Mapping) 或用于识别人或物体识别。由于其优越的实时性能，最近还被广泛用于无人驾驶车辆。

典型的产品是在室内广泛使用的Hokuyo的URG系列，如图8-15所示。多用于室外的产品有SICK和Velodyne的配有很多激光传感器的HDL系列。这些传感器最大的问题是价格。一般来说，不同产品的价格不尽相同，但大多是几千美元左右，而其中Velodyne的HDL系列是几万美元的产品。弥补这些缺点的中国产品（如RPLIDAR）以400美元左右的低价进入了市场，而近期则出现了韩国的一家公司推出的一款100多美元的LDS（HLS-LFCD2）¹⁶。



图8-15 从左边起，SICK LMS 210、Hokuyo UTM-30LX、Velodyne HDL-64e和HLS-LFCD LDS

8.5.1. LDS传感器距离测量原理

LDS传感器在测量距离时利用激光被物体反射时出现的波长。问题在于这里使用的激光器受控制和价格问题，所以大多数制造商只使用一个激光源。作为参考，价格高达数万美元或更多的Velodyne公司的HDL系列使用的激光器少则16个，多达64个。除此之外大多只使用一个激光器。为了克服这个问题，典型的LDS由一个激光器、一个反射镜和电机组成。当使用LDS时，会听到电机声音，因为它会旋转内部的反射镜，以在水平面内发射激光。测量范围通常是从180度到360度，取决于具体产品。

图8-16的左侧是LDS的内部结构，可以看到内部有一个激光器和一个倾斜的反射镜，在选择反射镜的过程中测量激光的返回时间（准确地说是波长的差异）。这样，如中间所示，可以扫描到以LDS为中心的水平面上的物体。但缺点是，如右图所示，随着距离变长，准确度降低。

¹⁶ http://wiki.ros.org/hls_lfcd_lds_driver

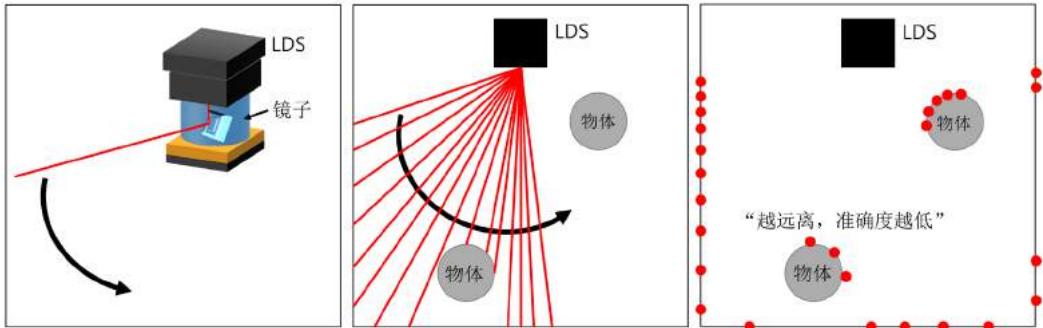


图8-16 使用LDS进行距离测量

用户不需要知道LDS的工作原理。这里包含了这个内容的原因是，因为这种原理，在测量时有一些要注意的地方。

第一，由于激光被用作光源，强烈的激光束可能会损伤眼睛。不过由于产品分为不同的等级，购买产品时需要注意。一般来说，激光等级分为1级到4级，数字越高越危险。1级是安全的产品，与眼睛直接接触也没有问题，2级和2级以上会在长时间接触时有风险。上述的LDS对应于1级。

第二，由于LDS利用反射光，因此如果不发生反射，则无法测量。换句话说，如透明玻璃、PET瓶和玻璃杯等不发生反射而散射到许多方向的物体是无法用LDS准确识别的。而且，由于激光在镜面上会发生镜面反射，因此无法获得准确的值。

第三，由于LDS在水平面发射激光，所以传感器仅检测水平面上的物体。换句话说，您需要理解这是2D数据（某些LDS会旋转传感器本身，并以3D方式进行测量）。

8.5.2. LDS测试

支持LDS的典型的ROS功能包包括支持SICK LDS的sicks300、sicktoolbox和sicktoolbox_wrapper功能包，还有支持Hokuyo公司的LDS的hokuyo_node和urg_node功能包，以及支持velodyne公司的LDS的velodyne功能包。还有支持RPLIDAR的rplidar功能包，以及支持TurtleBot3所配备的LDS的hls_lfcd_lds_drive功能包。

安装hls_lfcd_lds_driver功能包

在本节中，我们将使用HLDS（日立-LG数据存储）公司的LDS（HLS-LFCD2）进行测试，因此安装hls_lfcd_lds_driver功能包¹⁷。

```
$ sudo apt-get install ros-kinetic-hls-lfcd-lds-driver
```

LDS连接和更改使用权限

HLDS的LDS是USB类型的，因此可以通过将其插入计算机的USB端口来进行连接。当连接建立时，它会被识别为“ttyUSB*”，所以让我们如下设置权限（注意，在这里被识别为ttyUSB0）。

```
$ ls -l /dev/ttyUSB*
crw-rw--- 1 root dialout 188, 0 Jul 13 23:25 /dev/ttyUSB0
```

在上面的结果中可以看到，ttyUSB0的使用权限还未被赋予。我们用chmod命令设置权限，如下所示。

```
$ sudo chmod a+rw /dev/ttyUSB0
$ ls -l /dev/ttyUSB*
crw-rw-rw- 1 root dialout 188, 0 Jul 13 23:25 /dev/ttyUSB0
```

您可以使用chmod命令验证是否已更改使用权限。

运行hlds_laser启动文件

为了运行hlds_laser启动文件，在运行了roscore的情况下运行以下命令。

```
$ roslaunch hls_lfcd_lds_driver hlds_laser.launch
```

检查scan数据

运行hlds_laser节点，则会将LDS值发送到“/scan”话题。我们用rostopic echo命令检查这个值，如下所示。

¹⁷ http://wiki.ros.org/hls_lfcd_lds_driver

```
$ rostopic echo /scan
header:
  seq: 49
  stamp:
    secs: 1499956463
    nsecs: 667570534
  frame_id: laser
angle_min: 0.0
angle_max: 6.28318548203
angle_increment: 0.0174532923847
time_increment: 2.98899994959e-05
scan_time: 0.0
range_min: 0.119999997318
range_max: 3.5
ranges: [0.0, 0.47200000286102295, 0.4779999852180481, 0.48399999737739563, 0.4909999966621399,
0.49700000088214874, 0.0, 0.5099999904632568,
```

在扫描信息中，frame_id设置为laser，测量角度为6.28318548203弧度（= 360°）。您还可以看到增量为1°（0.0174532923847 rad = 1°），测量距离范围是0.11米到3.5米，而且将距离值按测量角度的顺序记录在发布的数组中。

8.5.3. 可视化LDS的距离值

现在为了在GUI环境中检查LDS距离信息，运行RViz，并按以下顺序更改显示选项。

```
$ rviz
```

- ① 将RViz右上方的Views的Type设置为“TopDownOrtho”，将其变成XY平面视图，以便轻松查看二维距离信息。
- ② 将RViz左上角的[Global Options] → [Fixed Frame]更改为“laser”。
- ③ 单击RViz左下角的[Add]按钮，然后从显示中选择[Axes]来添加它。如图8-17所示，更改细节设置(Length和Radius)。
- ④ 单击RViz左下角的[Add]按钮，然后从显示屏中选择[LaserScan]来添加它。如图8-17所示，更改细节设置(Topic、Color Transformer和Color)。

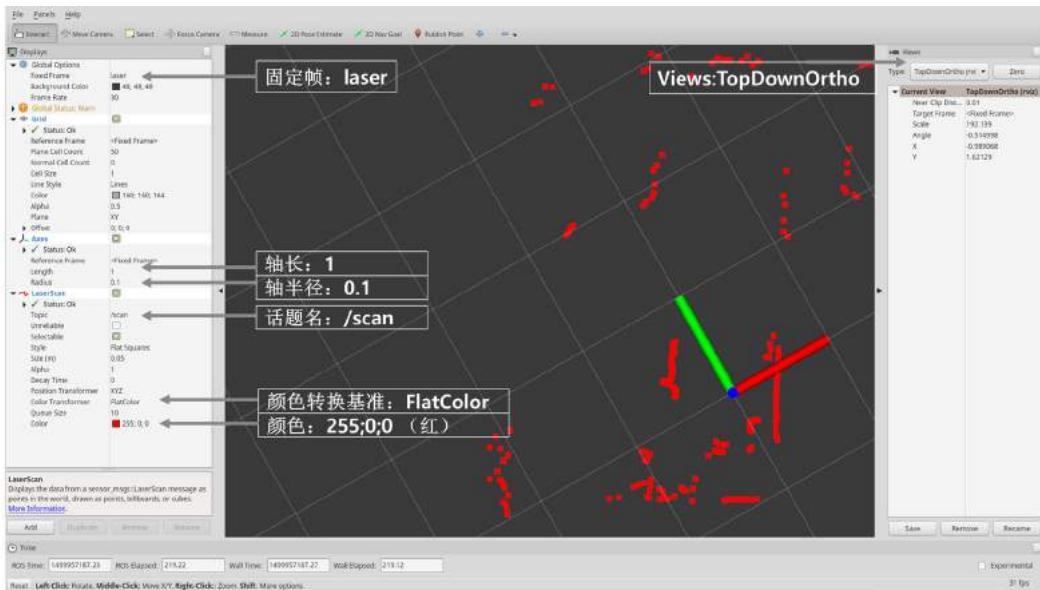


图8-17 在RViz上显示LaserScan

完成所有设置后，可以看到在中间红色（x轴）和绿色（y轴）所示坐标系的z轴周围扫描到了物体，并以点的形式显示出来，如图8-17所示。由于灰色网格的边长设为1米，因此可以通过与现实的比较来确认。

将上述操作作为预先配置的启动文件来运行，效果也是相同的。下面使用命令一次检查RViz中的激光值。

```
$ roslaunch hls_lfcd_lds_driver view_hlds_laser.launch
```

8.5.4. LDS的应用

LDS有着无限的应用，一个典型的例子是SLAM（Simultaneous Localization And Mapping）¹⁸。SLAM是在机器人上安装LDS，用它识别机器人周围的障碍物，并通过估计自身的位置来创建地图，如图8-18所示。在第11章里将详细介绍SLAM。

¹⁸ https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

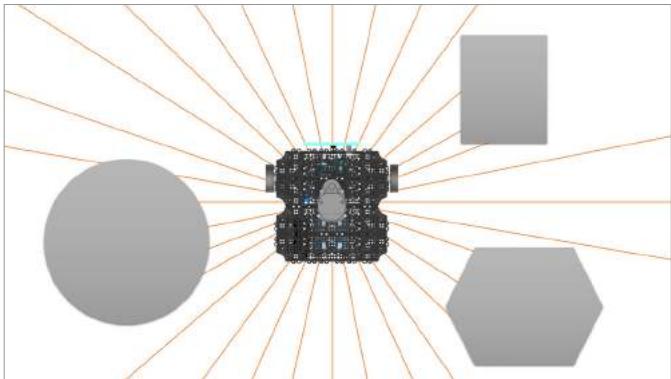


图8-18 LDS的应用：移动机器人的障碍物检测

作为LDS的另一个例子，LDS可以检测周围的各种物体，感知人的脚或身体的位置，如图8-19所示，并通过匹配周围情况来预测当前的行为。在第10章和第11章中，将利用安装了LDS的机器人更详细地介绍LDS的实际应用。

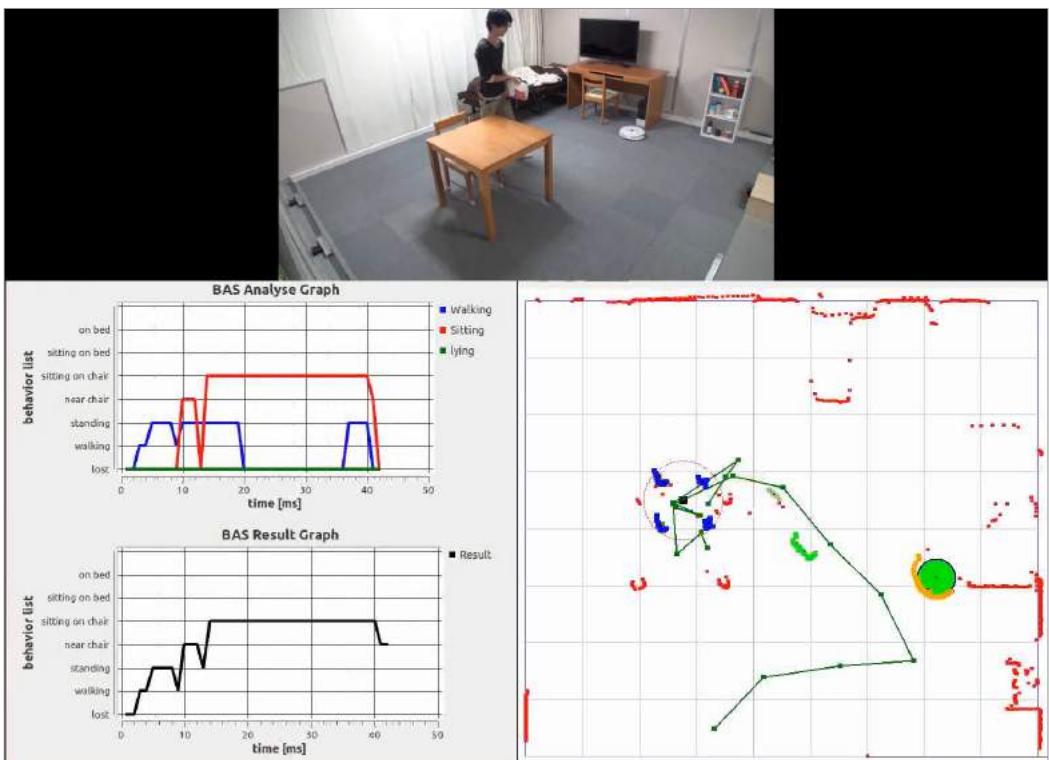


图8-19 使用LDSF的应用：检测人员和移动物体

8.6. 电机功能包

最近添加到ROS Wiki中的Motors页面¹⁹是ROS支持的所有电机和伺服控制器的说明页面。目前有支持PhidgetMotorControl HC、Roboteq AX2550 Motor Controller和ROBOTIS Dynamixel的功能包。

8.6.1. Dynamixel舵机

Dynamixel系列是由减速箱、控制器、驱动单元和通信单元组成的模块，并且可以向主机反馈位置、速度、温度、负载、电压和电流等物理量。因为它可以通过总线方法连接和控制，从而可以非常简单地设计机器人。除了基本的位置控制之外，还可以使用广泛应用于机器人的速度控制和转矩控制（部分系列）。

因为有这些丰富的功能，Dynamixel被广泛应用于机器人领域。将Dynamixel应用于机器人的方法有两种：一种是通过U2D2（将在第13章讨论的通信转换设备）将控制命令从计算机传送到舵机的方法，还有一种是直接通过OpenCR（将在第9章讨论）等嵌入式控制器来控制舵机的方法。为了在如此多样的环境中使用Dynamixel，此舵机支持叫做DynamixelSDK²⁰的开发环境。这个开发环境支持三种主要的操作系统（Linux、Windows和MacOS）且支持C、C++、C#、Python、Java、MATLAB和LabVIEW等多种编程语言。此外还支持Arduino和ROS功能包，因此在ROS中也可以很容易地使用Dynamixel。支持Dynamixel的典型功能包是dynamixel_motor、arbotix和dynamixel_workbench²¹。前两个是社区用户提供的功能包，后者是由ROBOTIS官方提供的功能包。dynamixel_workbench使用官方DynamixelSDK，通过ROS中的GUI工具支持电机设置和位置/速度/转矩控制。本书中将要详细讨论的TurtleBot3也将Dynamixel作为舵机来使用。有关这些电机的说明，请参见第9章“嵌入式系统”和第10章“移动机器人”。

¹⁹ <http://wiki.ros.org/Motor%20Controller%20Drivers>

²⁰ http://wiki.ros.org/dynamixel_sdk

²¹ http://wiki.ros.org/dynamixel_workbench



图8-20 Dynamixel系列舵机

8.7. 已公开的功能包的用法

ROS上已经发布了多少个功能包呢？截至2017年7月，ROS Kinetic提供了大约1600个功能包（http://repositories.ros.org/status_page/ros_kinetic_default.html），用户开发和发布的功能包可能有一些重复，但也有大约5,000个（<http://rosindex.github.io/stats/>）。在本节中，您将学习如何从公开了的功能包中搜索并安装和使用所需的功能包。

首先访问下面网页，点击上方的ROS版本中的“kinetic”，可以看到ROS的最新版本“kinetic”的功能包列表，如图8-21所示。

- <http://www.ros.org/browse/list.php>

The screenshot shows the ROS.org website's software browse interface for the 'kinetic' version. The 'kinetic' link in the top navigation bar and the 'packages' link in the left sidebar are highlighted with red circles. A large red arrow points down to the first row of the table below.

Name	Maintainers / Authors	Description
acc_finder	Martin Guenther	This package contains two small tools to help configure the navigation pipeline. The node min_max_f...
ackermann_msgs	Jack O'Quin	ROS messages for robots using Ackermann steering.
actionlib	Mikael Arguedas, Vijay Pradeep	The actionlib stack provides a standardized interface for interfacing with preemptable tasks. Ex...
actionlib_lisp	Lorenz Moesenlechner, Georg Bartels	actionlib_lisp is a native implementation of the famous actionlib in Common Lisp. It provides a c...
actionlib_msgs	Tully Foote	actionlib_msgs defines the common messages to interact with an action server and an action clie...
actionlib_tutorials	Daniel Stonier	The actionlib_tutorials package
alexandria	Lorenz Moesenlechner, Georg Bartels	3rd party library: Alexandria
amcl	David V. Lu!!, Michael Ferguson	<p> amcl is a probabilistic localization system for a robot moving in 2D. It...
angles	Ivan Sucan	This package provides a set of simple math utilities to work with angles. The utilities cove...
aniso8601	AlexV	Another ISO 8601 parser for Python
ar_track_alvar	Scott Niekum, Isaac I.Y. Saito	This package is a ROS wrapper for Alvar, an open source AR tag tracking library.

图8-21 ROS功能包列表

这个列表是为ROS Kinetic版本发布的功能包。数一数有大约1600个。之前的LTS版本Indigo发布了2900多个功能包。作为参考，有些是同一个功能包在ROS版本升级时持续提供兼容版，也有一些是由于开发终止，所以在新版本不提供兼容的功能包。但是，即使ROS的版本不同，也会有一些兼容性，所以只要对旧版本的功能包进行一点修改，也可以在新版本的ROS中使用。那么如何使用这些功能包呢？我们来看下一节。

8.7.1. 搜索功能包

为了在公开的ROS功能包中找到所需的功能包，请在网页<http://wiki.ros.org/>上的搜索框中输入关键词，则会在网站上显示搜索词的搜索结果。例如，如果您键入“find object”并单击“Submit”按钮，则可以查看与您输入的关键词匹配的各种功能包的信息和提问。



图8-22 如何搜索功能包

如果关键词合适，则会显示相关的功能包，如图8-22所示。有很多相关的功能包，但是在图中我们使用了“find_object_2d-ROS Wiki”中的“find_object_2d”功能包。点击“find_object_2d-ROS Wiki”将打开find_object_2d功能包的wiki页面，如图8-23所示。在这个页面上，您可以看到加载该功能包时使用的构建系统是catkin还是rosbuild，是谁创建的，以及它的开源许可证的类型。先点击上方的kinetic按钮，查看kinetic版本的信息。这个页面列出了依赖包（点击右边的Dependencies）、项目的Web页面的链接（External website）、功能包的存储库地址以及功能包的用法。其中，一定要检查功能包的依赖关系。

find_object_2d

hydro indigo jade kinetic Documentation Status

Package Summary

✓ Released ✓ Continuous integration ✓ Documented

The find_object_2d package

- Maintainer status: maintained
- Maintainer: Mathieu Labbe <matlabbe AT gmail DOT com>
- Author: Mathieu Labbe <matlabbe AT gmail DOT com>
- License: BSD
- External website: <http://find-object.googlecode.com>
- Source: git <https://github.com/introlab/find-object.git> (branch: kinetic-devel)

依赖包

外部链接

存储库

Package Links

- Code API
- Msg API
- find_object_2d website
- FAQ
- Change List
- File News
- Dependencies (12)**
- Jenkins jobs (12)**

Contents

- Overview
 - Citing
- Quick start
- Description
- 3D position of the objects
- Nodes
 - find_object_2d
 - Subscribed Topics
 - Published Topics
 - Parameters

1. Overview

Simple Qt interface to try OpenCV implementations of SIFT, SURF, FAST, BRIEF and other feature detectors and descriptors. Using a webcam, objects can be detected and published on a ROS topic with ID and position (pixels in the image). This package is a ROS integration of the [Find-Object](#) application.

图8-23 功能包的信息

8.7.2. 安装依赖包

可以在find_object_2d功能包的wiki页面²²中查看功能包相关性（Dependencies），则可以看到此功能包总共依赖于12个不同的功能包。

- catkin
- cv_bridge
- genmsg
- image_transport
- message_filters
- pcl_ros
- roscpp
- rospy
- sensor_msgs
- std_msgs
- std_srvs
- tf

使用rospack list命令或rospack find命令确认是否安装了必要的功能包。

使用rospack list命令确认

```
$ rospack list
actionlib /opt/ros/kinetic/share/actionlib
actionlib_msgs /opt/ros/kinetic/share/actionlib_msgs
actionlib_tutorials /opt/ros/kinetic/share/actionlib_tutorials
```

使用rospack find命令确认（如果已安装）

```
$ rospack find cv_bridge
/opt/ros/kinetic/share/cv_bridge
```

²² http://wiki.ros.org/find_object_2d

使用`rospack find`命令确认（如果尚未安装）

```
$ rospack find cv_bridge  
[rospack] Error: package 'cv_bridge' not found
```

如果未安装，请检查相应功能包的wiki页面上的安装方法并按如下所示进行安装。

```
$ sudo apt-get install ros-kinetic-cv-bridge
```

此外，维基页面 (http://wiki.ros.org/find_object_2d) 中的“2. Quick start”中的“`find_object_2d`”功能包被描述为能够使用`uvc_camera`功能包 (http://wiki.ros.org/uvc_camera)，所以我们也安装`uvc_camera`功能包。

```
$ sudo apt-get install ros-kinetic-uvc-camera
```

8.7.3. 安装功能包

如果您已经安装了所有的依赖包，那么接下来安装`find_object_2d`。典型的安装方法有二进制安装和下载后构建源代码的方式。在图8-23的功能包信息中，单击存储库中的链接，转到Github地址，这里有安装方法的说明。

二进制安装

```
$ sudo apt-get install ros-kinetic-find-object-2d
```

安装源程序

```
$ cd ~/catkin_ws/src  
$ git clone https://github.com/introlab/find-object.git  
$ cd ~/catkin_ws/  
$ catkin_make
```

以下功能包与ROS没有直接关系，但是它们使用`find_object_2d`功能包中的OpenCV和Qt库，因此您需要在安装之前安装它们。

```
$ sudo apt-get install libopencv-dev          // 安装OpenCV  
$ sudo apt-get install libqt4-dev            // 安装Qt
```

8.7.4. 运行功能包

按照find_object_2d功能包的说明运行功能包。首先启动roscore，然后在另一个终端窗口中使用以下命令启动相机节点。

```
$ roscore
```

```
$ rosrun uvc_camera uvc_camera_node
```

然后打开另一个终端窗口并运行find_object_2d节点，如下所示。

```
$ rosrun find_object_2d find_object_2d image:=image_raw
```

将检测对象的图像保存为PNG、JPEG等通用的图像文件格式，并将其拖放到已执行的GUI程序中。在这里，我们将下面两个图片用作检测对象，如图8-24所示。



图 8-24 两个要检测的对象

现在我们尝试对象检测。准备图8-24的要检测的图像和其他非检测图像混在一起的图像，并将其打印后放到相机前面，如图8-25所示。您可以看到两个字符串分别被一个矩形包围，说明被正确检测到。



图8-25 两个检测到的对象

您还可以在终端窗口中使用`rostopic echo`命令来查看`/object`话题，或运行`print_objects_detected`节点以查看搜索到的对象的信息。当使用此功能包创建新功能包时，如果通过话题接收该坐标值，那么可以充分创建其他的应用功能包。

```
rostopic echo /object
```

```
rosrun find_object_2d print_objects_detected
```

随着ROS开始被广泛使用，在ROS公开的功能包正在迅速增加。正如我在本节中所解释的那样，如果您知道如何在需要的时候查找和运用功能包，那么您可以依靠那些一直专注并努力工作的人们的成果，能够向前跨越一步，将时间集中花费在真正需要的部分。这是ROS的基本理念。知识逐渐积累并上升到更高的阶段，以此带来机器人技术的发展。

至此，说明了如何使用开源的功能包的例子。有关功能包的详细的用法，请参阅ROS Wiki。

第9章

嵌入式系统

嵌入式系统的定义如下：嵌入在需要控制的系统中的专用计算机系统。

嵌入式系统¹

嵌入式系统（英文：embedded system）是指，在机器或其他需要控制的系统中，执行控制功能的计算机系统。换句话说，一个嵌入式系统是整个系统中的一部分，是特定目的的计算机系统，是作为控制系统的大脑。

如图9-1所示，为了实现机器人的功能，机器人使用了很多嵌入式设备。具体而言，为了使用机器人的舵机或传感器，采用了可以进行实时控制的微控制器，并且在利用相机的视频处理或导航过程中需要采用配备高性能处理器的计算机。

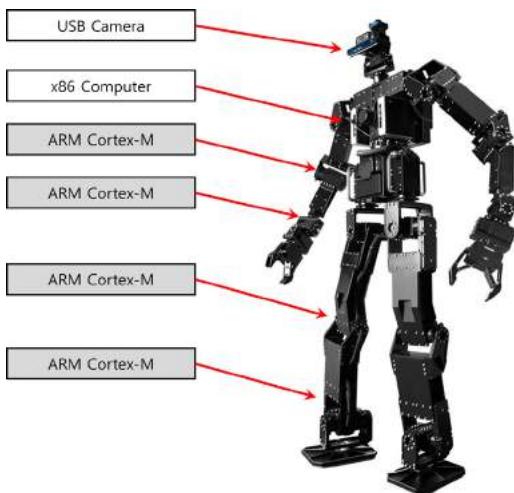


图 9-1 机器人的嵌入式系统配置

图9-2中列举²了从8位微控制器到高性能的PC，在实际应用中要根据需要选择适当性能的嵌入式系统。以ROS为例，它需要在PC或ARM Cortex-A系列的高性能CPU中运行，并且需要类似Linux的操作系统。

¹ https://en.wikipedia.org/wiki/Embedded_system

² https://roscon.ros.org/2015/presentations/ros2_on_small_embedded_systems.pdf



	8/16-bit MCU	32-bit MCU		ARM A-class	x86
		"small" 32-bit MCU	"big" 32-bit MCU		
Example Chip	Atmel AVR	ARM Cortex-M0	ARM Cortex-M7	Samsung Exynos	Intel Core i5
Example System	Arduino Leonardo	Arduino M0 Pro	SAM V71	ODROID	Intel NUC
MIPS	10's	100's	100's	1000's	10000's
RAM	1-32 KB	32 KB	384 KB	a few GB (off-chip)	2-16 GB (SODIMM)
Max power	10's of mW	100's of mW	100's of mW	1000's of mW	10000's of mW
Peripherals	UART, USB FS, ...	USB FS	Ethernet, USB HS	Gigabit Ethernet	USB SS, PCIe

图 9-2 各种嵌入式控制板

由于Linux等操作系统不能保证实时性，所以为了控制舵机或传感器，使用适合实时控制的微控制器。

TurtleBot3 Burger和TurtleBot3 Waffle中，Cortex-M7系列微控制器用于控制舵机和传感器，而使用Linux和ROS操作系统的Raspberry Pi 3控制板则通过USB连接，配置结构如图9-3。

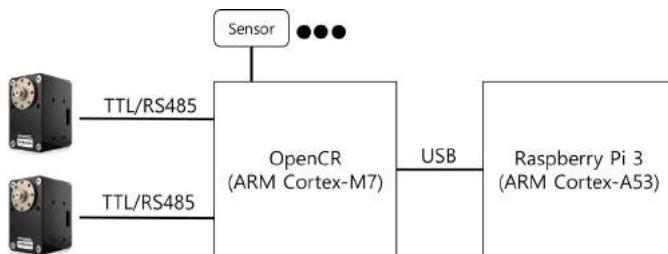


图 9-3 TurtleBot3 嵌入式系统配置

9.1. OpenCR

OpenCR (Open-source Control Module for ROS)³是一个支持ROS的嵌入式控制板，被用作TurtleBot3的主控制器。诸如电路/固件/ Gerber数据等硬件信息⁴和用于TurtleBot3的OpenCR源代码⁵均已公开，并且用户可以修改和重新分发。

主控MCU使用STM32F746⁶，它内嵌ARM Cortex-M7内核，硬件支持浮点运算，适合实现需要高性能的场合。

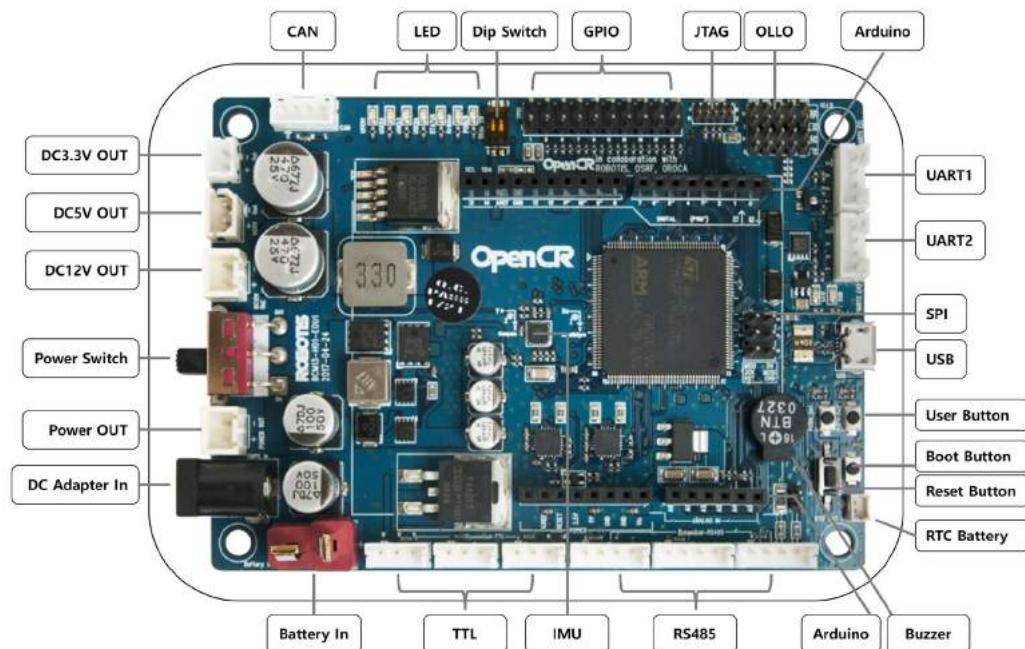


图 9-4 OpenCR的接口

³ http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_opencr1_0/

⁴ <https://github.com/ROBOTIS-GIT/OpenCR-Hardware>

⁵ <https://github.com/ROBOTIS-GIT/OpenCR>

⁶ <http://www.st.com/en/microcontrollers/stm32f746ng.html>

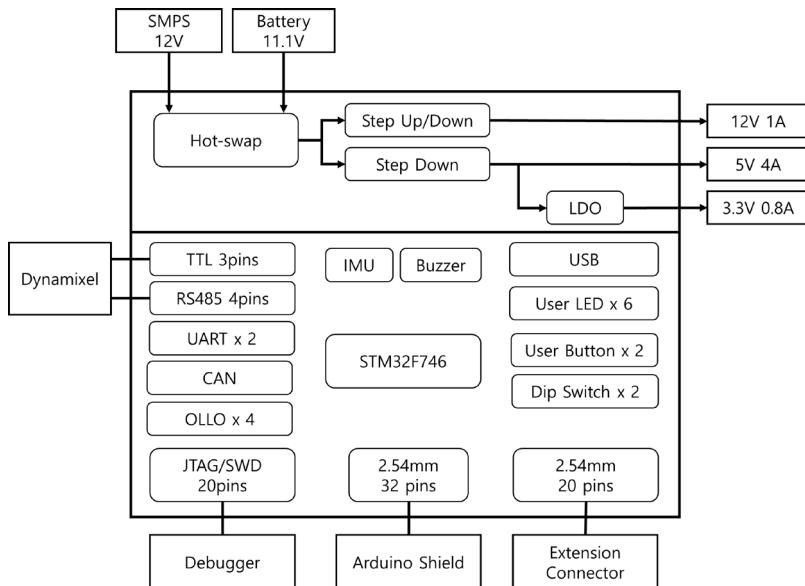


图 9-5 OpenCR 框图

9.1.1. 特点

高性能

OpenCR 是一款高性能微控制器，采用了意法半导体的 STM32F746，是 ARM 微控制器中最高端的 Cortex-M7 内核，工作频率高达 216MHz。它也可以用来实现高速运算算法或利用多种外围设备来处理大量数据的任务。

支持 Arduino

OpenCR 的基本开发环境采用 Arduino IDE⁷，因此那些不熟悉嵌入式开发环境的人们也可以容易地使用。支持与 Arduino UNO⁸ 引脚头兼容的接口，因此可以直接使用那些在现有的 Arduino 开发环境中开发出来的很多库和源代码以及大部分 Arduino 扩展模块。由于 OpenCR 控制板是通过 Arduino IDE 控制板管理器添加和管理的，所以很容易处理固件的更新。

⁷ <https://www.arduino.cc/en/Main/Software>

⁸ <https://store.arduino.cc/usa/arduino-uno-rev3>

多种接口

OpenCR支持ROBOTIS公司的舵机接口TTL和RS485，因此，可以使用ROBOTIS公司的大部分型号的舵机。此外，它还支持UART/SPI/I2C/CAN等通信接口，并具有额外的GPIO。调试接口使用JTAG⁹，因此可以使用STLink或JLink等专业JTAG设备来开发和调试固件。

IMU传感器

由于OpenCR控制板包含了具有集成陀螺仪/加速度计/地磁传感器的MPU9250¹⁰，因此无需额外的传感器，也可以实现IMU传感器的各种应用。由于通过SPI通信接口处理传感器数据，因此可以进行高速读/写操作。

电压输出

输入电压为7V~24V时提供12V/5V/3.3V等电压输出。它可以用作树莓派等SBC的电源，因为OpenCR支持5V/4A的高电流输出。

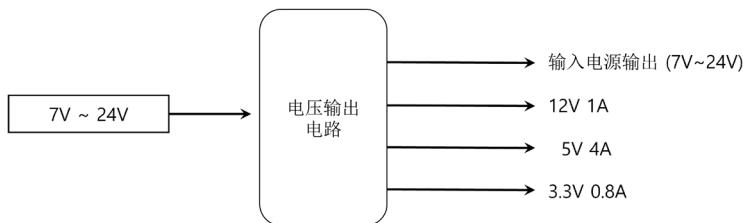


图 9-6 电源输出示意图

电源热插拔

连接了电池的情况下连接SMPS（开关模式电源，通常称为将交流电转换为直流电的电源装置），会自动将控制板电源从电池转换为开关电源。相反，如果您在使用SMPS作为电源时接上电池，之后拔出SMPS电源，则会使用电池作为供电源，而OpenCR控制板不会断电。这使得OpenCR在换电池时无需关闭电源。

⁹ <https://en.wikipedia.org/wiki/JTAG>

¹⁰ <https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/>

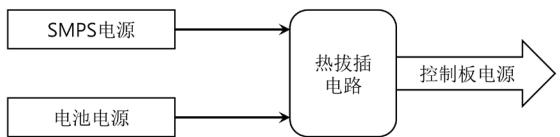


图 9-7 热插拔示意图

开源代码

OpenCR控制板生产所需的所有资料都已开放。提供了硬件生产所需的PCB Gerber和引导加载程序/固件也都已在github上公开。因此，用户可以根据需要改变它并重新制作。

- <https://github.com/ROBOTIS-GIT/OpenCR>
- <https://github.com/ROBOTIS-GIT/OpenCR-Hardware>

9.1.2. 控制板规格

硬件规格

表9-1显示了OpenCR的硬件规格。

Items	Specifications
Microcontroller	STM32F746ZGT6 / 32-bit ARM Cortex®-M7 with FPU (216MHz, 462DMIPS)
Sensors	Gyroscope 3Axis, Accelerometer 3Axis, Magnetometer 3Axis (MPU9250)
Programmer	ARM Cortex 10pin JTAG/SWD connector USB Device Firmware Upgrade (DFU) USB (Virtual COM Port)
Extension pins	32 pins (L 14, R 18) *Arduino connectivity Sensor module x 4 pins Extension connector x 18 pins
Communication circuits	USB TTL (JST 3pin / Dynamixel) RS485 (JST 4pin / Dynamixel) UART x 2 CAN SPI

Items	Specifications
LED	LD2 (red/green) : USB communication
Button	User LED x 4 : LD3 (red), LD4 (green), LD5 (blue)
Switch	User Button x 2
	User Switch x 2
Powers	External input source └ 5 V (USB VBUS), 7-24 V (Battery or SMPS) └ Default battery: LI-PO 11.1V 1,800mAh 19.98Wh └ Default SMPS: 12V 5A
	External output source └ 12V@1A, 5V@4A, 3.3V@800mA
	External battery connect for RTC (Real Time Clock)
	Power LED: LD1 (red, 3.3 V power on)
	Reset button x 1 (for power reset of board)
	Power on/off switch x 1
Dimensions	105(W) X 75(D) mm
Mass	60g

表 9-1 OpenCR 的硬件规格

闪存映射（闪存布局）

OpenCR 的闪存总共为 1MB，由引导加载程序区域、Arduino 中使用的模拟 EEPROM 的区域以及固件区域组成。为了用闪存来模拟 Arduino 中使用的 EEPROM 库，同时又为了增加闪存的写入寿命，我们使用两个闪存扇区。

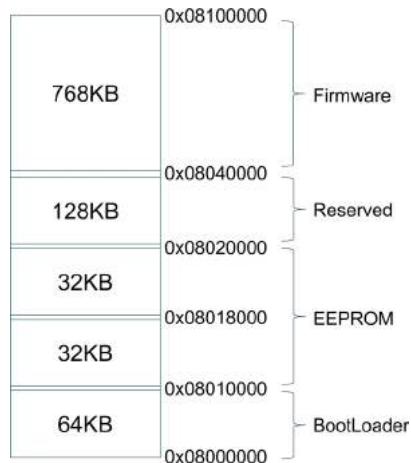


图 9-8 闪存映射（闪存布局）

IMU传感器

OpenCR使用InvenSense公司的 MPU9250传感器，为了精确的测量，将它位于OpenCR控制板的中心。MPU9250中内置陀螺仪/加速度/地磁传感器，传感器的方向和控制板的方向如下图9-9和9-10所示。

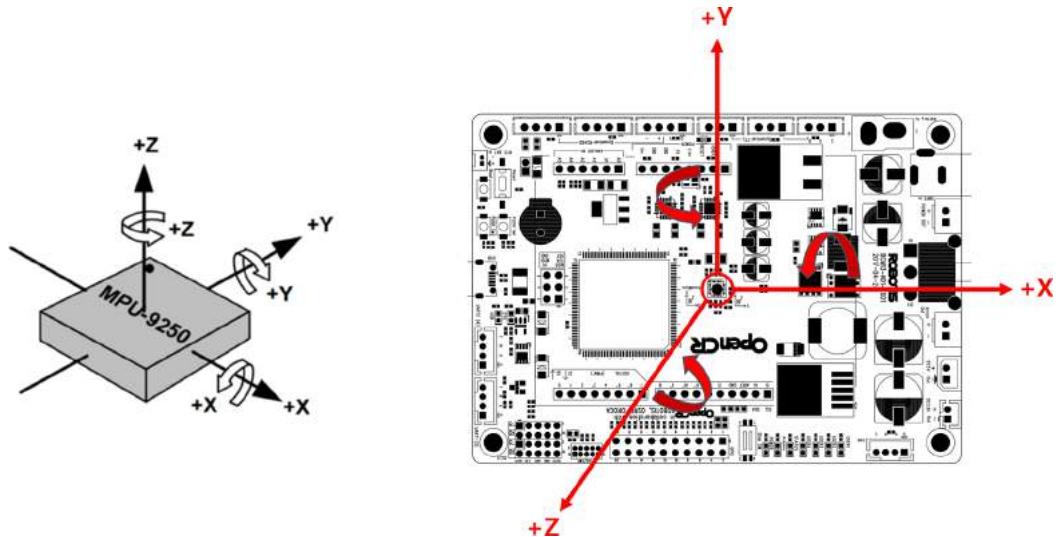


图 9-9 陀螺仪/加速度传感器方向

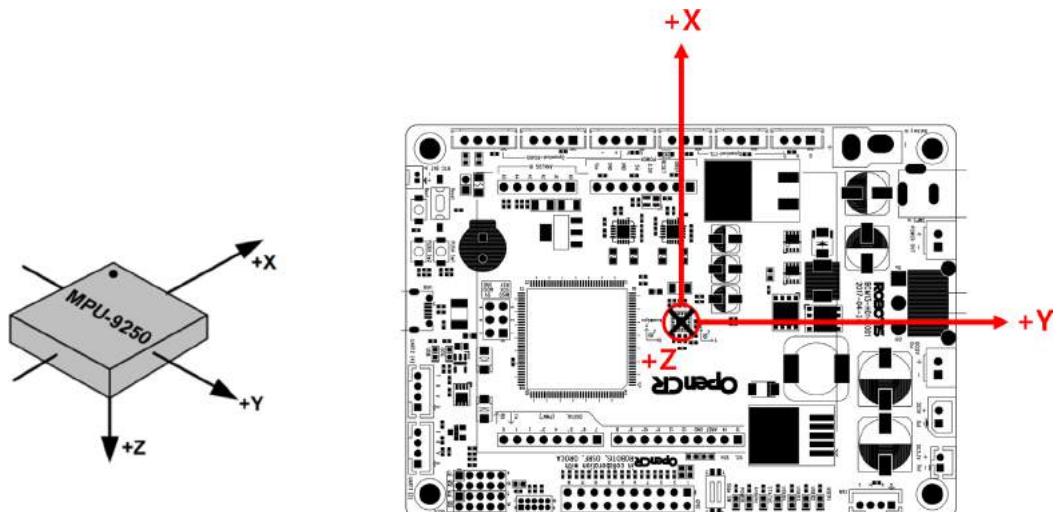


图 9-10 地磁传感器的方向

9.1.3. 搭建开发环境

OpenCR的基本开发环境是Arduino IDE。OpenCR是一个与Arduino兼容的控制板，附加的硬件通过提供附加的库函数来扩展功能。在Arduino IDE中安装OpenCR并完成设置。Arduino IDE利用Arduino.cc中发布的版本，并通过控制板管理器（Board Manager）添加和管理控制板。让我们用如下过程搭建一个开发环境。

设置USB端口权限

为了从Arduino IDE下载固件到OpenCR，可以通过以下命令更改对USB的管理员访问权限。下面的说明是在开发环境是Linux的前提下的说明。打开一个新的终端窗口（Ctrl + Alt + t），并输入以下命令。

```
$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/OpenCR/master/99-opencr-cdc.rules  
$ sudo cp ./99-opencr-cdc.rules /etc/udev/rules.d/  
$ sudo udevadm control --reload-rules  
$ sudo udevadm trigger
```

99-opencr-cdc.rules文件包含更改USB端口的访问权限的选项和不将OpenCR识别为调制解调器的选项。在Linux中，当连有串行通信设备时，Linux会向设备发送特定的命令，以判断设备是不是调制解调器。但这个命令可能会造成问题，因此通过上述选项防止发生这种情况。

```
ATTRS{idVendor}=="0483" ATTRS{idProduct}=="5740", ENV{ID_MM_DEVICE_IGNORE}="1", MODE=="0666"
```

设置编译环境

OpenCR中使用的GCC使用32位可执行文件，所以如果您安装了64位操作系统，需要添加与32位库的兼容性。

```
$ sudo apt-get install libncurses5-dev:i386
```

安装Arduino IDE

从Arduino下载站下载最新版本的Arduino IDE。OpenCR已经在版本1.6.12及以上的版本上测试过，并且已经确认它在最新版本1.8.2上运行。如果您的Arduino IDE的版本超过此版本，则仍然可以使用最新版本，因为它保持版本兼容性。

- <https://www.arduino.cc/en/Main/Software>

下载最新版本，将其解压到~/tools目录并继续安装。如果没有tools目录，请使用命令“cd ~/ && mkdir tools”创建一个新目录。

```
$ cd ~/tools/arduino-1.8.2  
$ ./install.sh
```

将Arduino IDE路径添加到您的shell配置文件，以便您可以从任何位置运行它。shell配置文件可以使用gedit或其他文本编辑程序加载。

```
$ gedit ~/.bashrc
```

将解压的路径添加到PATH中，并添加以下命令以实际反映它。

```
export PATH=$PATH:$HOME/tools/arduino-1.8.2
```

```
$ source ~/.bashrc
```

运行Arduino IDE

现在安装完成，可以使用以下命令在终端窗口中运行Arduino。

```
$ arduino
```



图 9-11 Adunino IDE运行界面

OpenCR设置

Arduino IDE安装完成后，需要添加控制板，以便可以使用OpenCR进行构建固件和下载固件。从Arduino IDE的菜单中选择File → Preferences，在图9-12的Additional Boards Manager URLs字段中输入下面的控制板配置文件链接，然后按“OK”。

- https://raw.githubusercontent.com/ROBOTIS-GIT/OpenCR/master/arduino/opencr_release/package_opencr_index.json

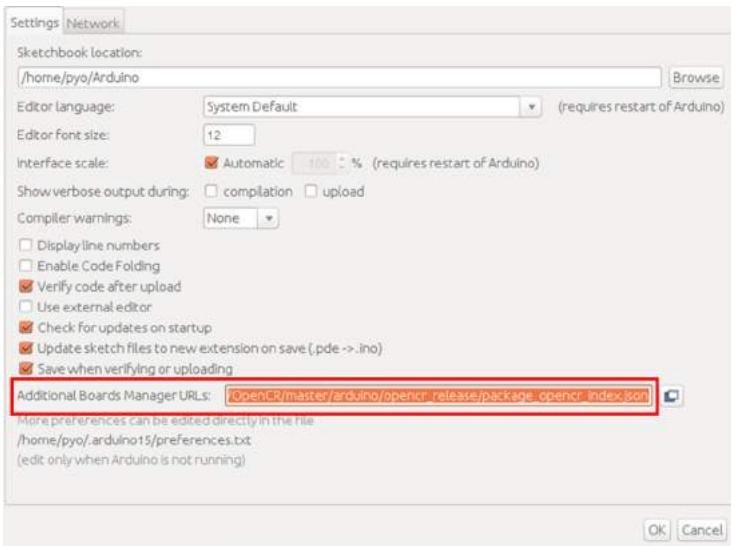


图 9-12 输入控制板管理器配置文件

输入控制板配置文件链接后，从Arduino IDE菜单中选择Tools→Board→Boards Manager，如图9-13所示。

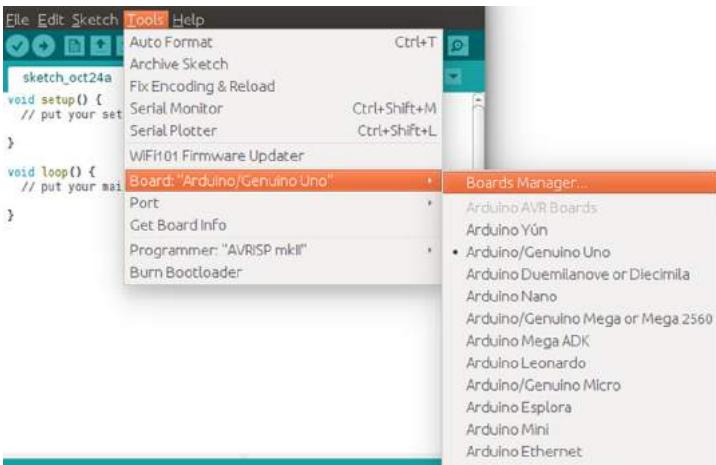


图 9-13 启动控制板管理器

可以在控制板列表的末尾看到有OpenCR。如果选择“ROBOTIS OpenCR”并安装，则控制板相关的文件将被自动安装。如果有已经安装的版本，则可以删除现有版本，或者如果版本已升级，则可以更新它。

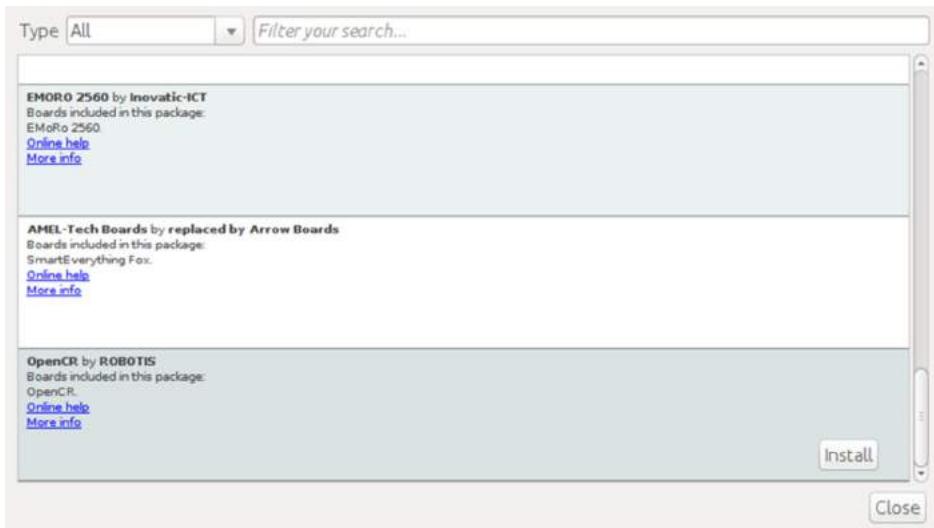


图 9-14 控制器列表

到目前为止，我们通过控制板管理器安装了OpenCR。要使用已安装的OpenCR，请在Arduino IDE的菜单中选择Tools→Board→OpenCR Board，如图9-15所示。

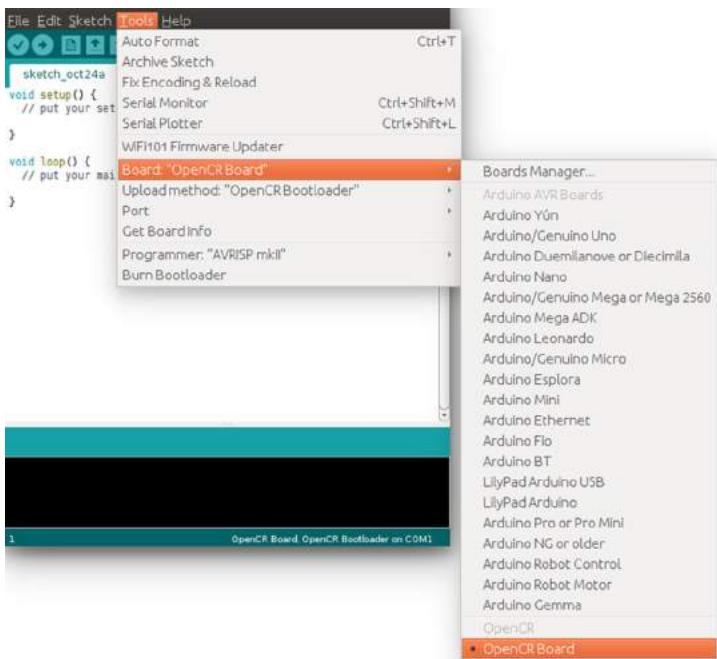


图 9-15 选择控制器

当OpenCR控制板连接到PC时，它会被识别为串行设备。如果您从Tools→Port中选择串行端口名称，如图9-16所示，那么我们已经做好了使用OpenCR控制板的准备。

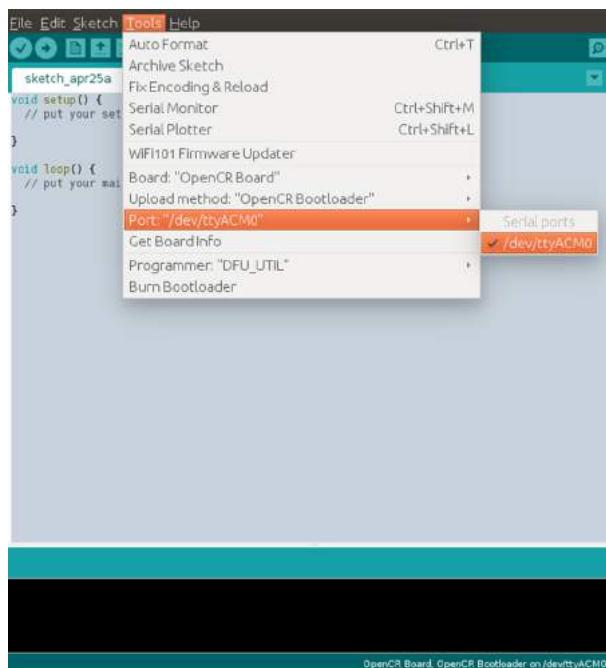


图 9-16 选择通讯端口

确认固件下载

如图9-17所示，选择File→New创建一个新文件。选择控制板和通讯端口，然后点击上方的向右箭头的图标来构建源代码并下载。

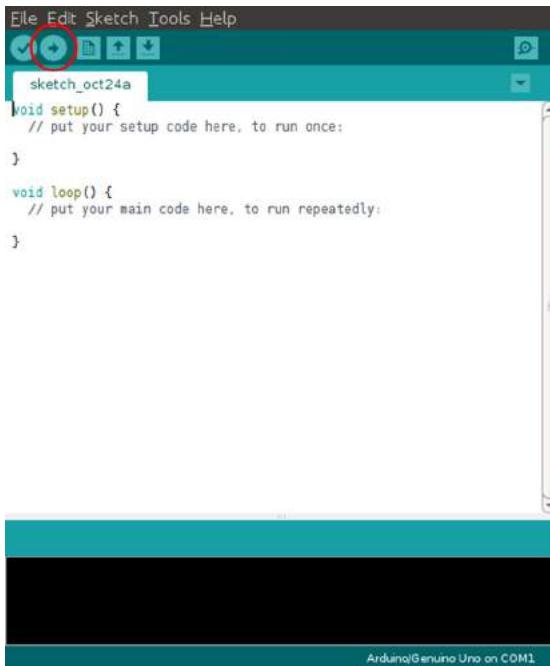


图 9-17 构建和下载固件

一旦源代码被构建完毕，Arduino IDE将调用OpenCR的下载程序并下载构建好的固件。在消息窗口的底部，将显示以下消息，并在下载完后执行下载的固件。

```
opencr_ld ver 1.0.2
opencr_ld_main
>>
file name :
/tmp/arduino_build_655974/b_Blink_LED.ino.bin
file size : 36 KB
Open port OK
Clear Buffer Start
Clear Buffer End
>>
Board Name : OpenCR R1.0
Board Ver : 0x17020800
Board Rev : 0x00000000
>>
flash_erase : 0 : 0.931000 sec
flash_write : 0 : 0.806000 sec
CRC OK 37F398 37F398 0.001000 sec
[OK] Download
jump_to_fw
```

} 下载器版本信息
} 文件信息及打开端口
} 固件版本信息
} 读/写过程与结果信息

图 9-18 下载消息

固件修复模式

如果下载的固件不正常工作，则无法用上述方法下载别的固件，则需要在强制执行引导加载程序之后下载固件。要执行引导加载程序，如图9-19所示，请按住控制板上的PUSH SW2按键，并使用RESET按键重置控制板，如此操作将运行引导加载程序。在这种状态下，您可以下载正常的固件。



图 9-19 运行固件修复模式

更新引导加载程序

如果需要更新OpenCR的引导加载程序，请使用STM32F746内置的引导加载程序的DFU功能。借助DFU功能，OpenCR的引导加载程序可以在没有任何附加设备（如JTAG）的情况下进行更新。作为参考，在制作控制板时已经下载了引导加载程序，所以用户更新它的情况很少见。在用USB将OpenCR连接到PC的情况下，同时按下BOOT0键和RESET键，则执行内置在STM32F746中的引导加载程序并使MCU进入DFU模式。



图 9-20 DFU模式按键

您可以使用lsusb命令检查是否已正常进入DFU模式。如图9-21所示，在USB设备列表中要有“STMicroelectronics STM Device in DFU Mode”。

```
$ lsusb
$ lsusb
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 003: ID 2109:0812 VIA Labs, Inc. VL812 Hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 005: ID 046d:c52b Logitech, Inc. Unifying Receiver

Bus 001 Device 020: ID 0483:df11 STMicroelectronics STM Device in DFU Mode

Bus 001 Device 013: ID 05e3:0608 Genesys Logic, Inc. Hub
Bus 001 Device 012: ID 046d:08ce Logitech, Inc. QuickCam Pro 5000
Bus 001 Device 011: ID 0c45:7603 Microdia
Bus 001 Device 010: ID 2109:2812 VIA Labs, Inc. VL812 Hub
Bus 001 Device 007: ID 8087:0a2a Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

图 9-21 确认DFU设备

为了使用DFU模式，请从Arduino IDE菜单中选择Tools→Programmer→DFU_UTIL，如图9-22所示。

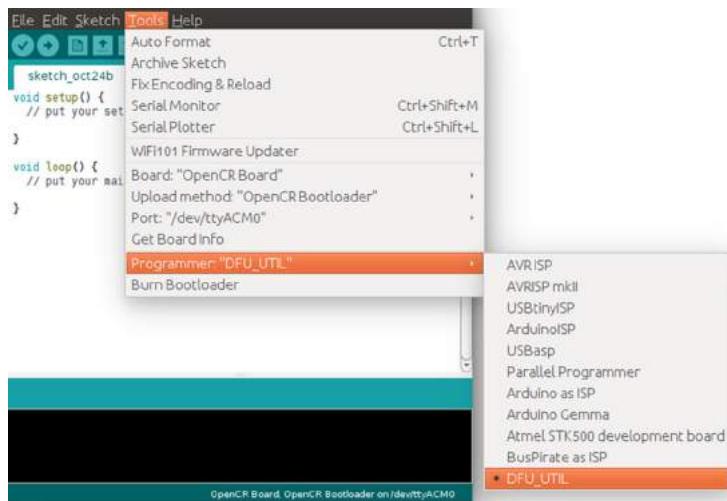


图 9-22 选择Programmer（下载程序）

选择下载程序后，通过选择Tools→Burn Bootloader来更新引导加载程序，如图9-23所示。更新完成后，您必须重置控制板。

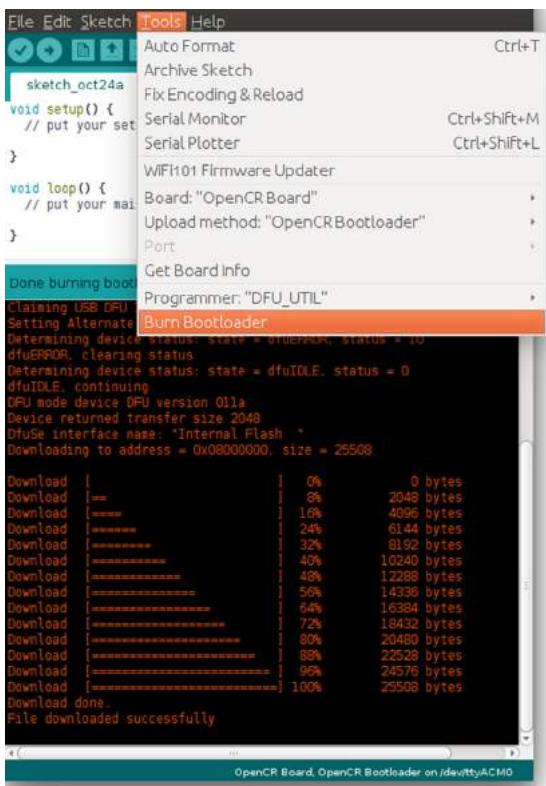


图 9-23 更新引导程序

9.1.4. OpenCR例程

通过控制板管理器将OpenCR添加到Arduino IDE后，可以使用File → Example菜单中的OpenCR例程。有很多例程可以帮助您控制和学习OpenCR的硬件。我们来看看OpenCR提供的一些主要附加组件的例程。

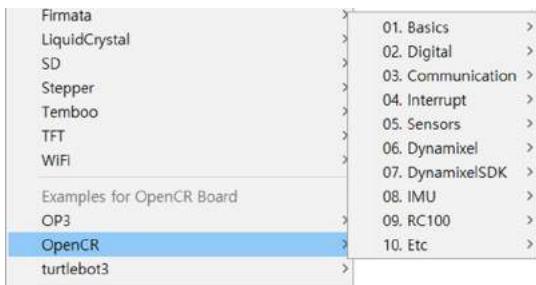


图 9-24 OpenCR例程

LED

OpenCR有4个可供用户控制的LED，位置如下图9-25所示。让我们使用用户可控的LED显示信息。

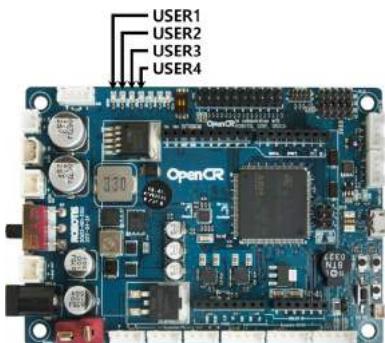


图 9-25 用户LED的位置

四个LED被定义为BDPIN_LED_USER_1 ~ BDPIN_LED_USER_4，以下例程按顺序闪烁LED。

```
blink_led

int led_pin = 13;
int led_pin_user[4] = { BDPIN_LED_USER_1, BDPIN_LED_USER_2, BDPIN_LED_USER_3, BDPIN_LED_USER_4 };

void setup() {
    pinMode(led_pin, OUTPUT);
    pinMode(led_pin_user[0], OUTPUT);
    pinMode(led_pin_user[1], OUTPUT);
```

```
pinMode(led_pin_user[2], OUTPUT);
pinMode(led_pin_user[3], OUTPUT);
}

void loop() {
    int i;

    digitalWrite(led_pin, HIGH);
    delay(100);
    digitalWrite(led_pin, LOW);
    delay(100);

    for( i=0; i<4; i++ )
    {
        digitalWrite(led_pin_user[i], HIGH);
        delay(100);
    }
    for( i=0; i<4; i++ )
    {
        digitalWrite(led_pin_user[i], LOW);
        delay(100);
    }
}
```

Buzzer

OpenCR内置有蜂鸣器。可以使用Arduino的基本功能之一的tone()函数控制蜂鸣器。Buzzer连接的引脚定义为BDPIN_BUZZER。tone()函数的参数是引脚号、频率(Hz) 和持续时间(ms)。

```
buzzer

void setup()
{
}

void loop() {
    tone(BDPIN_BUZZER, 1000, 100);
    delay(200);
}
```

电压测量

可以测量电池或SMPS等输入电源的电压。可以使用getPowerInVoltage()函数进行测量，该函数返回输入电压，单位为Voltage。

read_voltage

```
void setup() {  
    Serial.begin(115200);  
}  
  
void loop() {  
    float voltage;  
  
    voltage = getPowerInVoltage();  
  
    Serial.print("Voltage : ");  
    Serial.println(voltage);  
}
```

IMU传感器

通过传感器融合将加速度/陀螺仪传感器的值转换为控制板的Roll/Pitch/Yaw值。当创建cIMU类作为对象并调用update()函数时，定期读取加速度/陀螺仪值来计算姿态值。默认的计算周期是200赫兹，但可以更改。

read_roll_pitch_yaw

```
#include <IMU.h>  
  
cIMU IMU;  
  
void setup()  
{  
    Serial.begin(115200);  
  
    IMU.begin();  
}  
  
void loop()  
{
```

```

static uint32_t pre_time;

IMU.update();

if( (millis()-pre_time) >= 50 )
{
    pre_time = millis();

    Serial.print(IMU.rpy[0]);
    Serial.print(" ");
    Serial.print(IMU.rpy[1]);
    Serial.print(" ");
    Serial.println(IMU.rpy[2]);
}
}

```

Dynamixel SDK

为了控制ROBOTIS公司的Dynamixel舵机，OpenCR使用C++版DynamixelSDK的Arduino定制版。Arduino定制版支持原版DynamixelSDK的用法，且利用原版DynamixelSDK编写的源代码也只需修改一部分就可以在Arduino中使用。

DynamixelSDK可以在以下地址获取，出厂时OpenCR中内置修改后的版本。

- <https://github.com/ROBOTIS-GIT/DynamixelSDK>

DynamixelSDK例程中的一部分包含在OpenCR库中，并支持协议1.0和2.0。

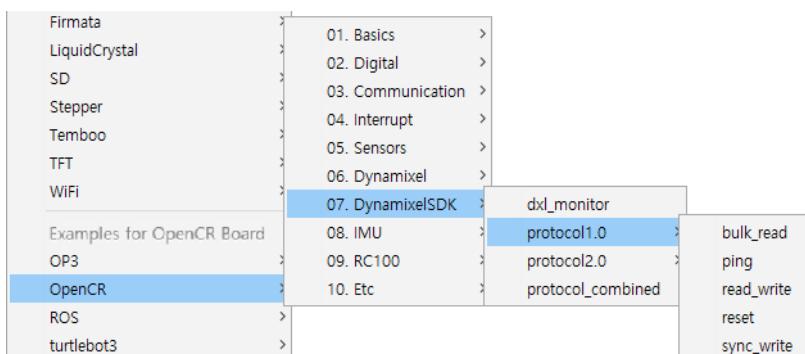


图 9-26 DynamixelSDK的协议1.0的例程

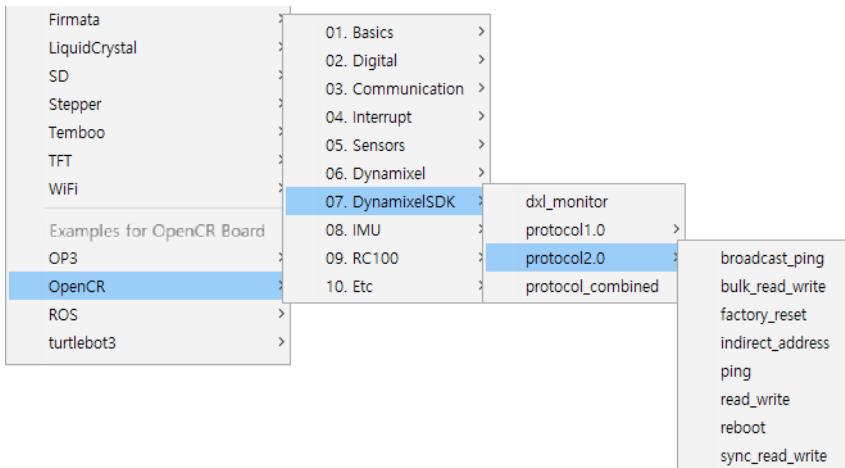


图 9-27 DynamixelSDK协议2.0的例程

9.2. rosserial

rosserial¹¹是一个可以将ROS的消息、话题和服务转换为串行通信方式的功能包。通常，微控制器使用串行通信，如UART，而不是ROS中的基本通信方式TCP / IP。因此，对于使用ROS的微控制器和计算机之间的消息通信，需要诸如rosserial的中介作用。

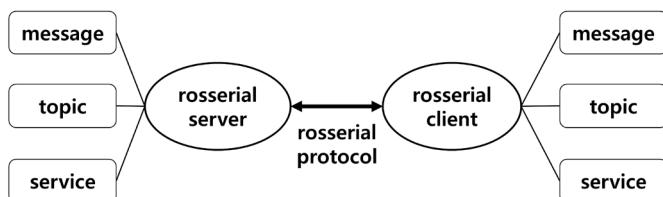


图 9-28 rosserial server和client

运行ROS的PC是一个rosserial server¹²，连接到PC的微控制器作为rosserial client¹³。由于server和client使用rosserial协议发送和接收数据，因此所有能够发送和接收数据的硬件都可以使用。因此微控制器中常用的UART也可以用于ROS消息或话题。

¹¹ <http://wiki.ros.org/rosserial>

¹² http://wiki.ros.org/rosserial_server

¹³ http://wiki.ros.org/rosserial_client

例如，如果将连接到微控制器的传感器的值先进行ADC数字化，然后通过URAT传输，那么计算机的rosserial_server节点将接收串行值并将其转换为ROS中使用的话题（Topic）。相反，如果ROS的另一个节点将电机速度控制值发送给话题，则rosserial_server节点会将其发送到单片机，并直接控制连接的电机。

包括SBC在内的通用计算机不能保证实时控制，但如果将微控制器作为辅助硬件控制器来使用，就可以保证实时性。

9.2.1. rosserial server

rosserial server是运行ROS的PC机上的一个节点，它通过rosserial protocol¹⁴做为PC与嵌入式设备之间的中介。根据所使用的编程语言，目前为止可以使用三种节点。

rosserial_python

它是用Python语言实现的，使用rosserial时经常用到。

rosserial_server

因为使用了C++语言，性能相对有所提升，但相比于rosserial_python，功能上还是有一些限制。

rosserial_java

当需要基于Java语言的模块时，或者在与Android SDK一起使用时用到。

9.2.2. rosserial client

这是作为rosserial的client功能的库，是将rosserial_client库移植到微控制器所使用的平台上。由于它支持Arduino平台，因此用户可以使用任何支持Arduino的开发板，并且源代码已公开，所以可以轻松移植到其他平台。

¹⁴ <http://wiki.ros.org/rosserial/Overview/Protocol>

rosserial_arduino

它用于Arduino板，支持Arduino UNO和Leonardo开发板，但可以通过修改源码在其他开发板上使用。TurtleBot3中使用的OpenCR控制板使用经过部分修改的rosserial_arduino。

rosserial_embeddedlinux

这是一个可以在嵌入式Linux上使用的库。

rosserial_windows

它支持Windows操作系统并支持与Windows应用程序的通信。

rosserial_mbed

支持嵌入式开发环境mbed平台，因此可以使用mbed开发板。

rosserial_tivac

这是用于TI生产的Launchpad板的库。

9.2.3. rosserial协议

rosserial server和client通过基于串行通信的数据包的形式发送和接收数据。rosserial协议是以字节为单位定义的，包含了数据包同步和数据验证所需的信息。

数据包结构

rosserial数据包通过数据包的头部分（为了进行ROS标准消息数据发送/接收而附加）来验证有效性，而各个数据则通过“校验和”来验证数据的有效性。

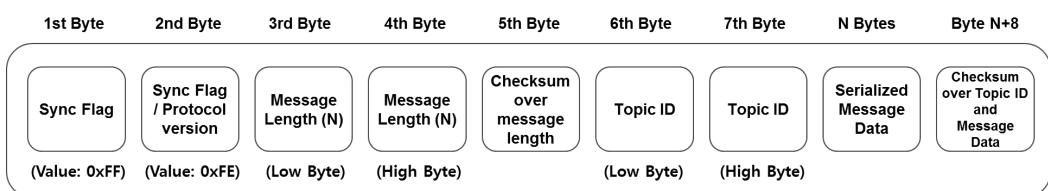


图 9-29 rosserial数据包的结构

Sync Flag

用于标志数据包的起始位置，始终是0xFF。

Sync Flag / Protocol version

协议版本，ROS Groovy是0xFF，ROS Hydro、Indigo、Jade和Kinetic是0xFE。

Message Length

通过数据包传输的消息的数据长度，由2个字节组成。低字节先传输，后接高字节。

Checksum over message length

用于验证消息长度的校验和，计算如下。

```
255 - (Message Length Low Byte + Message Length High Byte)%256
```

Topic ID

识别消息类型的ID，由2个字节组成。话题ID 0到100为用于系统功能而保留。系统使用的主要的话题ID如下。这可以在rosserial_msgs/TopicInfo¹⁵查看。

```
uint16 ID_PUBLISHER=0  
uint16 ID_SUBSCRIBER=1  
uint16 ID_SERVICE_SERVER=2  
uint16 ID_SERVICE_CLIENT=4  
uint16 ID_PARAMETER_REQUEST=6  
uint16 ID_LOG=7  
uint16 ID_TIME=10  
uint16 ID_TX_STOP=11
```

Serialized Message Data

它是以串行形式传输发送/接收消息的数据。

Checksum over topic ID and Message Data

话题ID和消息数据的校验和，计算如下。

¹⁵ https://github.com/ros-drivers/roserial/blob/jade-devel/roserial_msgs/msg/TopicInfo.msg

```
255 - ((Topic ID Low Byte + Topic ID High Byte + data byte values) % 256)
```

查询包

当rosserial server启动后，向client请求话题名称和话题类型等信息。此时发送的就是查询包。查询包的话题ID为0，数据长度为0。查询包的内容如下。

```
0xff 0xfe 0x00 0x00 0xff 0x00 0x00 0xff
```

收到查询包的client向server发送具有以下内容的消息，之后server会基于该信息发送和接收消息。

```
uint16 topic_id  
string topic_name  
string message_type  
string md5sum  
int32 buffer_size
```

9.2.4. rosserial的约束条件

使用rosserial可以发送和接收ROS标准消息，但由于嵌入式系统的硬件限制，在使用rosserial时存在一些差异。因此，创建使用rosserial的节点时，必须考虑这些约束条件。

内存约束

嵌入式系统中使用的微控制器具有有限的内存空间，并且比常规PC容量小得多。因此，考虑到内存容量，必须预先定义要使用的发布者、订阅者的数量和发送/接收缓冲区的大小。应该注意的是，超过发送/接收缓冲区大小的消息数据是不能被传送的。

Float64

将rosserial_arduino用作rosserial client时，Arduino开发板的微控制器不支持64位实数运算，因此在创建库时会自动更改为32位类型。如果微控制器支持64位实数，则需要修改make_libraries.py中的数据类型转换部分。

Strings

由于微控制器的存储空间的限制，字符串数据不存储在String消息中，而是只将外部定义的字符串数据的指针值存储在消息中。要使用String消息，需要如下操作。

```
std_msgs::String str_msg;
unsigned char hello[13] = "hello world!";
str_msg.data = hello;
```

Arrays

像字符串一样，由于内存的限制，仅用指向数组数据的指针是无法知道数组的尾端的。因此，添加了有关数组大小的信息，并用于发送和接收消息。

通讯速度

在UART通讯的情况下，如果使用115200bps的速率，当消息的数量增多时，会拖慢响应和处理速度。但是在OpenCR中，通过使用USB进行虚拟串行通信，可以实现高速通信。

9.2.5. 安装rosserial

为了使用rosserial，需要安装ROS所需的功能包，并使用您要使用的设备的平台client库。请按照以下步骤安装功能包。我们下面将了解如何在Arduino平台上使用它。

安装功能包

使用以下命令安装支持rosserial和arduino系列的功能包。此外，还有ros-kinetic-rosserial-windows、ros-kinetic-rosserial-xbee和ros-kinetic-rosserial-embeddedlinux，根据需要安装即可。

```
$ sudo apt-get install ros-kinetic-rosserial ros-kinetic-rosserial-server ros-kinetic-rosserial-arduino
```

创建库

为了在Arduino中使用它，需要创建一个用于Arduino的rosserial库。移动到Arduino IDE个人目录中的库目录，如果存在之前生成的库，则删除现有的库。运行rosserial_arduino功能包中的make_libraries.py文件来创建ros_lib。由于OpenCR基本上已经添加了用于TurtleBot3的ros_lib库，所以在使用TurtleBot3时不需要执行以下步骤。

```
$ cd ~/Arduino/libraries/  
$ rm -rf ros_lib  
$ rosrn rosserial_arduino make_libraries.py .
```

更改通讯端口

创建Arduino ROS库时，使用默认通信端口。对于一般的Arduino开发板，会使用HardwareSerial类的Serial对象，因此为了改变通信端口，必须修改生成的库的源代码。如果查看库目录中的ros.h文件的内容，可以看到是用ArduinoHardware类来定义NodeHandle。因此，用户可以在ArduinoHardware类中更改要使用的端口的硬件功能。

```
ros.h  
  
#include "ros/node_handle.h"  
#include "ArduinoHardware.h"  
  
namespace ros  
{  
    /* Publishers, Subscribers, Buffer Sizes */  
    typedef NodeHandle <ArduinoHardware, 25, 25, 1024, 1024> NodeHandle;  
}
```

在使用OpenCR的情况下，ArduinoHardware.h文件中的SIGNAL_CLASS被改为USBSerial，以便它可以通过USB进行通信。如果要改为别的端口，则可以更改串口类和对象。

```
#define SERIAL_CLASS USBSerial
__省略__
class ArduinoHardware
{
    iostream=&Serial;
__省略__
}
```

9.2.6. rosserial例程

OpenCR提供rosserial库以及利用OpenCR的基本的rosserial例程。如图9-30所示，有一些像LED和Button等输入/输出例程，也有IMU传感器的例程。例程会在后续版本中继续添加。



图 9-30 OpenCR的rosserial例程

在继续下面的例程之前，必须先运行roscore。

LED

通过使用ROS标准数据类型std_msg/Byte将四个LED定义为led_out订阅者。当订阅者的后台函数被调用时，如果传递的消息值的0~3位bit值为1，则打开相应的LED，反之关闭相应的LED。

```
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Byte.h>

int led_pin_user[4]={BDPIN_LED_USER_1,BDPIN_LED_USER_2,BDPIN_LED_USER_3,BDPIN_LED_USER_4};
```

```
ros::NodeHandle nh;

void messageCb( const std_msgs::Byte& led_msg) {
    int i;

    for (i=0; i<4; i++)
    {
        if (led_msg.data & (1<<i))
        {
            digitalWrite(led_pin_user[i], LOW);
        }
        else
        {
            digitalWrite(led_pin_user[i], HIGH);
        }
    }
}

ros::Subscriber<std_msgs::Byte> sub("led_out", messageCb );

void setup() {
    pinMode(led_pin_user[0], OUTPUT);
    pinMode(led_pin_user[1], OUTPUT);
    pinMode(led_pin_user[2], OUTPUT);
    pinMode(led_pin_user[3], OUTPUT);

    nh.initNode();
    nh.subscribe(sub);
}

void loop(){
    nh.spinOnce();
}
```

通过Arduino IDE下载创建的固件，然后使用rosserial_python运行rosserial server，如下所示。如果OpenCR被识别为不同的USB设备，请修改并执行_port:=/dev/ttyACM0部分。从此刻起，rosserial server和OpenCR client通过USB发送和接收数据包并传递消息。

```
$ rosrun rosserial_python serial_node.py __name:=opencr _port:=/dev/ttyACM0 _baud:=115200
[INFO] [1495609829,326019]: ROS Serial Python Node
[INFO] [1495609829,336151]: Connecting to /dev/ttyACM0 at 115200 baud
[INFO] [1495609831,454144]: Note: subscribe buffer size is 1024 bytes
[INFO] [1495609831,454994]: Setup subscriber on led_out [std_msgs/Byte]
```

打开一个新终端并使用rostopic列表来验证是否有led_out话题。

```
$ rostopic list
/diagnostics
/led_out
/rosout
/rosout_agg
```

如果使用rostopic pub将LED控制值输入到led_out，则LED会根据该值亮起或灭掉。

```
$ rostopic pub -1 led_out std_msgs/Byte 1    → USER1 LED On
$ rostopic pub -1 led_out std_msgs/Byte 2    → USER2 LED On
$ rostopic pub -1 led_out std_msgs/Byte 4    → USER3 LED On
$ rostopic pub -1 led_out std_msgs/Byte 8    → USER4 LED On
$ rostopic pub -1 led_out std_msgs/Byte 0    → LED all Off
```

Button

与LED例程一样，Button例程也使用std_msgs/Byte 数据类型，但为了向server发送按键的输入，将Button声明为发布者。它发布OpenCR控制板的SW1/SW2按键被按下的状态值。按键值会周期性地发布，在下面的例程中，每50ms发送一次。

```
b_Button.ino

#include <ros.h>
#include <std_msgs/Byte.h>

ros::NodeHandle nh;

std_msgs::Byte button_msg;
ros::Publisher pub_button("button", &button_msg);

void setup()
{
```

```

nh.initNode();
nh.advertise(pub_button);

pinMode(BDPIN_PUSH_SW_1, INPUT);
pinMode(BDPIN_PUSH_SW_2, INPUT);
}

void loop()
{
    uint8_t reading = 0;
    static uint32_t pre_time;

    if (digitalRead(BDPIN_PUSH_SW_1) == HIGH)
    {
        reading |= 0x01;
    }
    if (digitalRead(BDPIN_PUSH_SW_2) == HIGH)
    {
        reading |= 0x02;
    }

    if (millis() - pre_time >= 50)
    {
        button_msg.data = reading;
        pub_button.publish(&button_msg);
        pre_time = millis();
    }

    nh.spinOnce();
}

```

下载Button例程并如下运行rosserial_python，可以从输出的信息中看到button发布者已完成设置。

```

$ rosrun rosserial_python serial_node.py __name:=openrcr _port:=/dev/ttyACM0 _baud:=115200
[INFO] [1495609931.875745]: ROS Serial Python Node
[INFO] [1495609931.885488]: Connecting to /dev/ttyACM0 at 115200 baud
[INFO] [1495609934.000344]: Note: publish buffer size is 1024 bytes
[INFO] [1495609934.001180]: Setup publisher on button [std_msgs/Byte]

```

用rostopic list命令确认是否有button话题。

```
$ rostopic list  
/button  
/diagnostics  
/rosout  
/rosout_agg
```

打开一个新的终端窗口并输入以下命令，以实时显示发布的按键值。

```
$ rostopic echo button  
data: 1  
---  
data: 1
```

测量输入电压

如图9-31所示，通过分压电阻将输入电压以特定比例缩小，然后可以通过ADC进行测量。根据分压公式，输入到ADC的电压是 $V_{out} = \text{输入电压} \times (10 / 57)$ 。因此，可以测量 V_{out} 电压并将其换算为输入电压。在OpenCR中，由getPowerInVoltage()函数执行换算操作，因此用户可以使用此函数来测量输入电压。

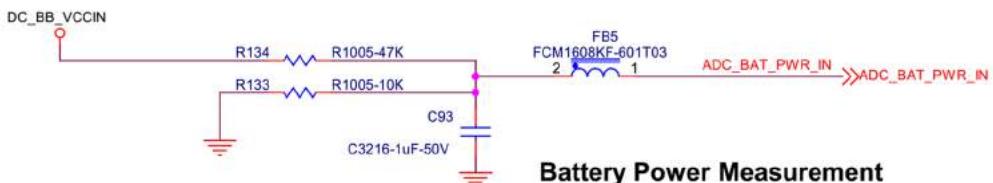


图 9-31 电压测量电路

为了表达准确的电压值，测量输入电压的例程使用std_msgs/Float32数据类型。声明一个叫做voltage的发布者，每50ms测量一次输入电压并发布。

c_Voltage.ino

```
#include <ros.h>
#include <std_msgs/Float32.h>

ros::NodeHandle nh;

std_msgs::Float32 voltage_msg;
ros::Publisher pub_voltage("voltage", &voltage_msg);

void setup()
{
    nh.initNode();
    nh.advertise(pub_voltage);
}

void loop()
{
    static uint32_t pre_time;

    if (millis()-pre_time >= 50)
    {
        voltage_msg.data = getPowerInVoltage();
        pub_voltage.publish(&voltage_msg);
        pre_time = millis();
    }

    nh.spinOnce();
}
```

使用以下命令运行rosserial server。

```
$ rosrun rosserial_python serial_node.py __name:=opencr _port:=/dev/ttyACM0 _baud:=115200
[INFO] [1495609160.098041]: ROS Serial Python Node
[INFO] [1495609160.108219]: Connecting to /dev/ttyACM0 at 115200 baud
[INFO] [1495609162.224307]: Note: publish buffer size is 1024 bytes
[INFO] [1495609162.225184]: Setup publisher on voltage [std_msgs/Float32]
```

打开一个新的终端窗口，用rostopic命令输出voltage话题值，则可以看到OpenCR的输入电压值。

```
$ rostopic echo voltage
data: 12.1300001144
---
data: 12.1099996567
---
data: 12.1300001144
---
data: 12.1099996567
```

IMU

为了用ROS的标准消息类型支持IMU传感器，这里使用sensor_msgs/Imu。以下例程将由IMU传感器库计算出的姿态值转换为sensor_msgs/Imu消息类型并发布。作为IMU传感器的参考的tf由base_link生成，并将base_link与IMU传感器连接，从base_link检查姿态值的变化。

```
d_IMU.ino

#include <ros.h>
#include <sensor_msgs/Imu.h>
#include <tf/tf.h>
#include <tf/transform_broadcaster.h>

#include <IMU.h>

ros::NodeHandle nh;

sensor_msgs::Imu imu_msg;
ros::Publisher imu_pub("imu", &imu_msg);

geometry_msgs::TransformStamped tfs_msg;
tf::TransformBroadcaster tfbroadcaster;
```

```
cIMU imu;

void setup()
{
nh.initNode();
nh.advertise(imu_pub);
tfbroadcaster.init(nh);

imu.begin();
}

void loop()
{
static uint32_t pre_time;

imu.update();

if (millis()-pre_time >= 50)
{
pre_time = millis();

imu_msg.header.stamp = nh.now();
imu_msg.header.frame_id = "imu_link";

imu_msg.angular_velocity.x = imu.gyroData[0];
imu_msg.angular_velocity.y = imu.gyroData[1];
imu_msg.angular_velocity.z = imu.gyroData[2];
imu_msg.angular_velocity_covariance[0] = 0.02;
imu_msg.angular_velocity_covariance[1] = 0;
imu_msg.angular_velocity_covariance[2] = 0;
imu_msg.angular_velocity_covariance[3] = 0;
imu_msg.angular_velocity_covariance[4] = 0.02;
imu_msg.angular_velocity_covariance[5] = 0;
imu_msg.angular_velocity_covariance[6] = 0;
imu_msg.angular_velocity_covariance[7] = 0;
imu_msg.angular_velocity_covariance[8] = 0.02;

imu_msg.linear_acceleration.x = imu.accData[0];
```

```
imu_msg.linear_acceleration.y = imu.accData[1];
imu_msg.linear_acceleration.z = imu.accData[2];
imu_msg.linear_acceleration_covariance[0] = 0.04;
imu_msg.linear_acceleration_covariance[1] = 0;
imu_msg.linear_acceleration_covariance[2] = 0;
imu_msg.linear_acceleration_covariance[3] = 0;
imu_msg.linear_acceleration_covariance[4] = 0.04;
imu_msg.linear_acceleration_covariance[5] = 0;
imu_msg.linear_acceleration_covariance[6] = 0;
imu_msg.linear_acceleration_covariance[7] = 0;
imu_msg.linear_acceleration_covariance[8] = 0.04;

imu_msg.orientation.w = imu.quat[0];
imu_msg.orientation.x = imu.quat[1];
imu_msg.orientation.y = imu.quat[2];
imu_msg.orientation.z = imu.quat[3];

imu_msg.orientation_covariance[0] = 0.0025;
imu_msg.orientation_covariance[1] = 0;
imu_msg.orientation_covariance[2] = 0;
imu_msg.orientation_covariance[3] = 0;
imu_msg.orientation_covariance[4] = 0.0025;
imu_msg.orientation_covariance[5] = 0;
imu_msg.orientation_covariance[6] = 0;
imu_msg.orientation_covariance[7] = 0;
imu_msg.orientation_covariance[8] = 0.0025;

imu_pub.publish(&imu_msg);

tfs_msg.header.stamp = nh.now();
tfs_msg.header.frame_id = "base_link";
tfs_msg.child_frame_id = "imu_link";
tfs_msg.transform.rotation.w = imu.quat[0];
tfs_msg.transform.rotation.x = imu.quat[1];
tfs_msg.transform.rotation.y = imu.quat[2];
tfs_msg.transform.rotation.z = imu.quat[3];

tfs_msg.transform.translation.x = 0.0;
tfs_msg.transform.translation.y = 0.0;
```

```
    tfs_msg.transform.translation.z = 0.0;

    tfbroadcaster.sendTransform(tfs_msg);
}

nh.spinOnce();
}
```

下载例程后，使用以下命令运行rosserial server以创建imu和tf发布者。

```
$ rosrun rosserial_python serial_node.py __name:=opencr _port:=/dev/ttyACM0 _baud:=115200
[INFO] [1495611663.941723]: ROS Serial Python Node
[INFO] [1495611663.946220]: Connecting to /dev/ttyACM0 at 115200 baud
[INFO] [1495611666.075668]: Note: publish buffer size is 1024 bytes
[INFO] [1495611666.076638]: Setup publisher on imu [sensor_msgs/Imu]
[INFO] [1495611666.146240]: Setup publisher on /tf [tf/tfMessage]
```

可以打开一个新的终端窗口，并按如下方式查看imu话题的消息数据。

```
$ rostopic echo /imu
header:
  seq: 686
  stamp:
    secs: 1495611700
    nsecs: 369472074
  frame_id: imu_link
orientation:
  x: 0.0232326872647
  y: -0.0115436725318
  z: -4.04381135013e-05
  w: 0.999659180641
orientation_covariance: [0.002499999441206455, 0.0, 0.0, 0.0, 0.002499999441206455, 0.0, 0.0, 0.0,
  0.002499999441206455]
angular_velocity:
  x: 0.0
  y: 0.0
  z: 0.0
angular_velocity_covariance: [0.01999999552965164, 0.0, 0.0, 0.0, 0.01999999552965164, 0.0, 0.0, 0.0,
  0.01999999552965164]
```

```

linear_acceleration:
  x: 370.0
  y: 754.0
  z: 16228.0
linear_acceleration_covariance: [0.03999999910593033, 0.0, 0.0, 0.0, 0.03999999910593033, 0.0, 0.0, 0.0,
  0.03999999910593033]
---

```

以图形方式检查IMU数据时可以使用RViz。输入命令如下。

```
$ rviz
```

如图9-32所示，在RViz的Displays画面中选择 Global Options → Fixed Frame → base_link。然后点击Displays画面底部的Add按钮，将轴添加到新的显示项目中，并选择Reference Frame→imu_link，则可以从屏幕上看到画面中的轴会根据OpenCR的姿态发生变化。

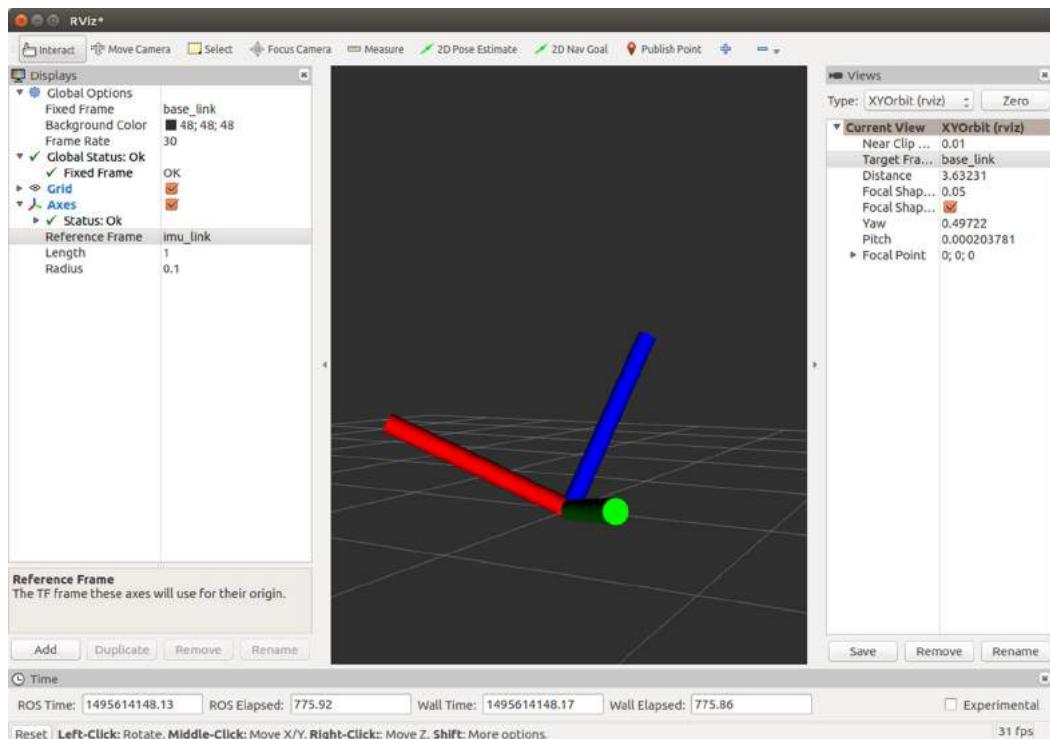


图 9-32 在RViz中验证IMU

9.3. TurtleBot3的固件

OpenCR内置了TurtleBot3的rosserial库，可以以例程形式下载TurtleBot3的固件。由于例程文件是源代码形式，因此可以由用户修改。TurtleBot3相关固件通过Arduino IDE的控制板管理器发布。因此，如果控制板管理器的版本发生变化，则只需执行更新后下载最新的固件即可。

9.3.1. TurtleBot3 Burger固件

TurtleBot3固件可以通过在Arduino IDE中选择Upload→File→Examples→turtlebot3→turtlebot3_burger→turtlebot3_core来下载，如图9-33所示。



图 9-33 下载TurtleBot3 Burger固件

TurtleBot3的Burger机器人和Waffle（Waffle Pi）机器人因为他们的Dynamixel舵机的安装位置和转动半径有差异，所以在turtlebot3_core_config.h中有适用于各机器人的参数值。如果改变了Dynamixel舵机的安装位置，这些参数值也必须改变。

```
turtlebot3_core_config.h
```

```
#define WHEEL_RADIUS      0.033    // meter
#define WHEEL_SEPARATION   0.160    // meter
#define TURNING_RADIUS     0.080    // meter
#define ROBOT_RADIUS        0.105    // meter
```

为了在TurtleBot3 Burger功能包中使用它，请使用以下命令运行rosserial_python节点。完成后，如图9-34所示，将执行turtlebot3_core节点，并会将移动命令订阅到/cmd_vel话题，并且将发布测位信息（/odom）、IMU信息（/imu）和传感器信息（/sensor_state）。TurtleBot3 Waffle和Waffle Pi的用法也相同。更详细的用法，请参阅第10章。

```

$ rosrun rosserial_python serial_node.py __name:=turtlebot3_core _port:=/dev/ttyACM0
[INFO] [1500275719.375458]: ROS Serial Python Node
[INFO] [1500275719.380338]: Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [1500275721.496849]: Note: publish buffer size is 1024 bytes
[INFO] [1500275721.497162]: Setup publisher on sensor_state [turtlebot3_msgs/SensorState]
[INFO] [1500275721.499622]: Setup publisher on imu [sensor_msgs/Imu]
[INFO] [1500275721.502328]: Setup publisher on cmd_vel_rc100 [geometry_msgs/Twist]
[INFO] [1500275721.507266]: Setup publisher on odom [nav_msgs/Odometry]
[INFO] [1500275721.511984]: Setup publisher on joint_states [sensor_msgs/JointState]
[INFO] [1500275721.568189]: Setup publisher on /tf [tf/tfMessage]
[INFO] [1500275721.571585]: Note: subscribe buffer size is 1024 bytes
[INFO] [1500275721.571865]: Setup subscriber on cmd_vel [geometry_msgs/Twist]
[INFO] [1500275721.573235]: Start Calibration of Gyro
[INFO] [1500275724.046148]: Calibration End

```

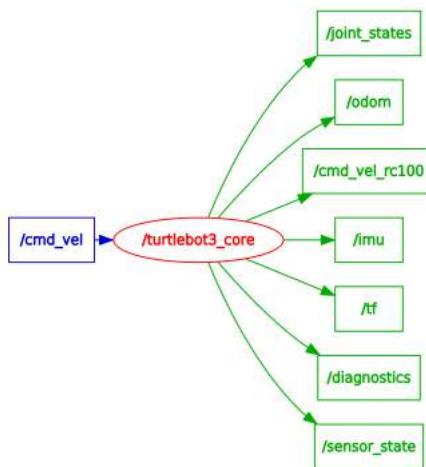


图 9-34 turtlebot3_core 节点发布和订阅的话题

9.3.2. TurtleBot3 Waffle 和 Waffle Pi 固件

如图9-35所示，用户可以从Arduino IDE中选择File→Examples→turtlebot3→turtlebot3_waffle→turtlebot3_core，然后点击Upload按钮来下载TurtleBot3 Waffle和Waffle Pi的固件。



图 9-35 TurtleBot3 Waffle和Waffle Pi的固件

```

turtlebot3_core_config.h

#define WHEEL_RADIUS      0.033          // meter
#define WHEEL_SEPARATION  0.287          // meter
#define TURNING_RADIUS   0.1435         // meter
#define ROBOT_RADIUS     0.220          // meter

```

9.3.3. TurtleBot3配置固件

TurtleBot3中使用的Dynamixel舵机在出厂时已将ID等设定值初始化为符合TurtleBot3的值。如果用户更换了舵机或将舵机改用为其他目的，那么为了重新正确用于TurtleBot3，需要恢复设定值。例程中有一个初始化舵机的例子，所以我们下面用这个例程来初始化设定值。

下载配置固件

如图9-36所示，将turtlebot3_setup→turtlebot3_setup_motor例程下载到OpenCR控制板，则会开始进行设置。完成设置后，再将TurtleBot3固件下载到OpenCR中。



图 9-36 TurtleBot3舵机设置例程

点击Adunino IDE上的Upload按钮下载，之后点击右上方的串口终端图标，如图9-37所示。之后将Dynamixel连接到OpenCR。请注意，此固件只适用于设置一个Dynamixel，因此必须连接一个Dynamixel。

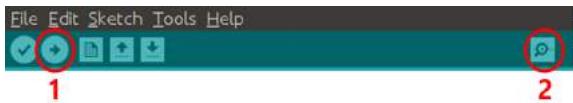


图 9-37 下载并运行串口终端

更改Dynamixel设置

运行串行终端时，将显示一个Dynamixel设置菜单，如图9-38所示。TurtleBot3中使用的Dynamixel由左右两个舵机组成，选择要更改的Dynamixel。如果要将舵机设置为左轮，请按下‘1’之后按Enter键。



图 9-38 TurtleBot3舵机设置菜单

为防止输入错误，再次显示确认菜单，如果确定要更改，请输入“Y”。

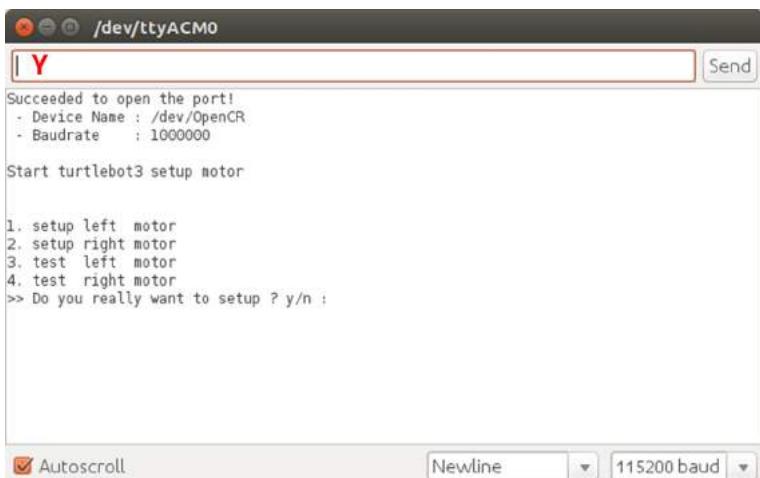
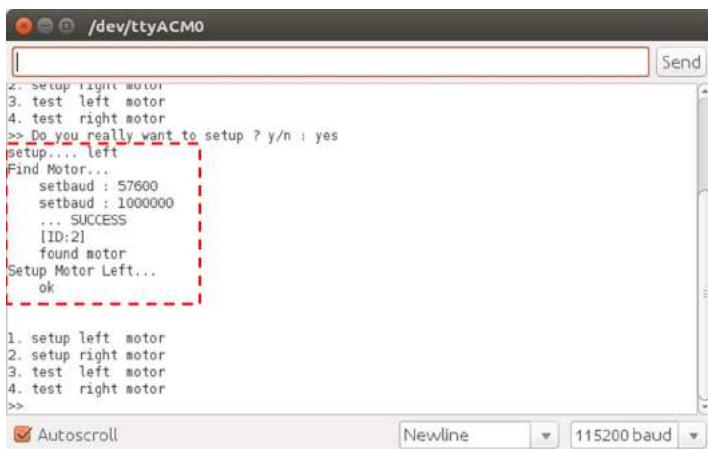


图 9-39 设置确认菜单

如果输入“Y”，则会一边改变通信速度，一边搜索连接的Dynamixel。找到舵机之后会初始化ID和其他设置值。更改完成后，最后输出消息“ok”。



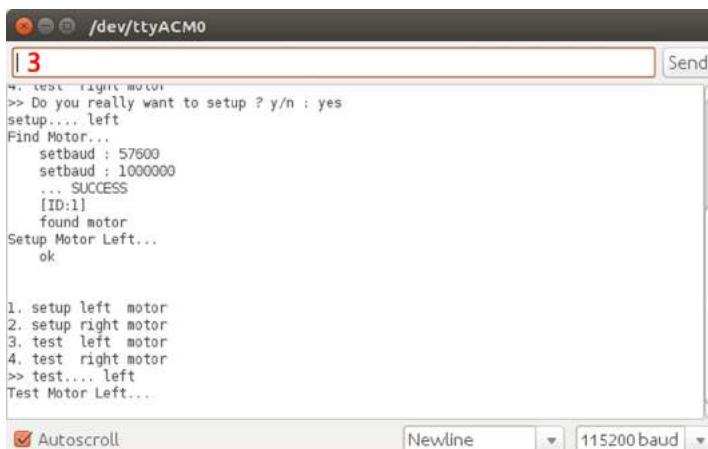
```
2. setup right motor
3. test left motor
4. test right motor
>> Do you really want to setup ? y/n : yes
setup.... left
Find Motor...
setbaud : 57600
setbaud : 1000000
... SUCCESS
[ID:2]
found motor
Setup Motor Left...
ok

1. setup left motor
2. setup right motor
3. test left motor
4. test right motor
>>
 Autoscroll
```

图 9-40 设置完成消息

Dynamixel测试

完成设置后需要检查是否正常更改。如果您在菜单中选择了test left motor/test right motor之一，则相应的Dynamixel会以顺时针方向和逆时针方向反复转动。要结束测试，请再按一次Enter键。测试左侧Dynamixel，要输入“3”，测试右侧Dynamixel，要输入“4”。



```
4. test right motor
>> Do you really want to setup ? y/n : yes
setup.... left
Find Motor...
setbaud : 57600
setbaud : 1000000
... SUCCESS
[ID:1]
found motor
Setup Motor Left...
ok

1. setup left motor
2. setup right motor
3. test left motor
4. test right motor
>> test.... left
Test Motor Left...
```

图 9-41 Dynamixel测试菜单

我们已经讨论了如何将ROS与嵌入式系统结合使用。需要实时控制的机器人与嵌入式系统是不可分割的关系。希望读者掌握与ROS结合使用的方法，用于以后的机器人开发。随后，第10、11、12和13章将介绍使用本章描述的嵌入式系统的移动机器人的实例。

第10章

移动机器人

10.1. ROS支持的机器人

ROS支持的机器人可以在相关wiki (<http://robots.ros.org/>) 上找到。有大约180种。其中一些包括由个人开发者公开发布的自制机器人，但是如果考虑到这是由单一系统支持的机器人的话，却是不少的数量。其中最著名的是Willow garage开发的PR2和TurtleBot。两者都是Willow garage或Open Robotics（前称OSRF）参与开发的机器人，也是ROS的标准平台。在本章中，我们将重点讨论这两者中以普及型为目的开发的TurtleBot3机器人。

10.2. TurtleBot3系列机器人

TurtleBot是ROS的标准平台机器人。这里的turtle来源于1967年开发的乌龟机器人。这款机器人是为了用教育计算机编程语言Logo¹控制实际机器人而开发的。另外，在ROS基础教程中首先出现的turtlesim节点是一个模仿Logo turtle²程序的命令体系而制作的程序。如图10-1所示，乌龟图标甚至成为了ROS的象征。不仅如此，ROS标志中使用的9个点也来自龟背壳。源于Logo的乌龟的Turtlebot，旨在通过TurtleBot向ROS新手轻松教授ROS的用法（这与Logo的目的类似，Logo的目的是利用Logo语言轻松教授计算机编程语言）。从第一代TurtleBot开始，近10年来，TurtleBot已经成为开发者和学生使用最多的ROS标准平台。

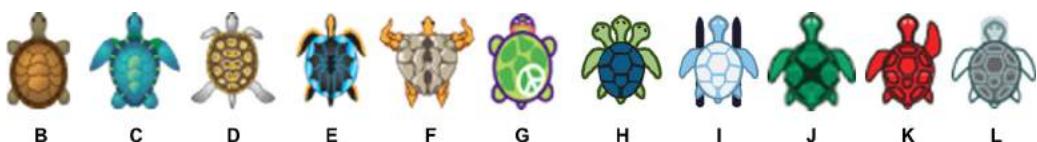


图 10-1 ROS各版本的图标

TurtleBot系列³有1、2和3版本（见图10-2）。TurtleBot1由Willow Garage的Tully（现为Open Robotics Platform Manager）和Melonee（现为Fetch Robotics CEO）

¹ <http://el.media.mit.edu/logo-foundation/index.html>

² http://el.media.mit.edu/logo-foundation/what_is_logo/logo_primer.html

³ <http://www.turtlebot.com/about>

为了普及ROS，以iRobot公司的基于Roomba的研究机器人Create为基础而开发⁴。它于2010年开发，自2011年起开始销售。然后在2012年，Yujin Robot公司开发出了以基于iCLEBO的研究机器人Kobuki为基础的TurtleBot2。在2017年开发的TurtleBot3弥补了TurtleBot1和TurtleBot2中的不足点，且采纳了用户们的新的需求。TurtleBot3的驱动部采用了ROBOTIS公司的智能舵机Dynamixel。



图 10-2 TurtleBot1（左1）、TurtleBot2（左2）和TurtleBot3（右侧3种机器人）

TurtleBot3是一个小型、低成本、可编程的基于ROS的移动机器人，其目的是用于教育、科研、爱好者作品和产品原型。TurtleBot3的目标是在不牺牲功能和品质的前提下大幅缩小平台的尺寸且降低价格，同时将机器人组件根据用户的需求更改或扩展。根据用户如何选择部件，如机械部件、计算机和传感器，TurtleBot3可以通过各种方法进行定制。此外，TurtleBot3采用了比现有的PC更经济、更小巧，并且适合嵌入式系统的SBC（单板计算机），还应用了距离传感器和3D打印等最新技术。

10.3. TurtleBot3的硬件

如图10-3，TurtleBot3⁵有TurtleBot3 Burger、Waffle和Waffle Pi三种官方型号。如不特指，本书中将以TurtleBot3 Burger为例进行说明。此外，TurtleBot3还有如TurtleBot3 Monster、Tank、Carrier等TurtleBot3 + 【后缀】的多种型号。TurtleBot3的基本组件有：用于驱动的舵机、用于运行ROS的SBC、用于SLAM和导航（Navigation）的传感器、可变形的结构件、用作中层控制器的嵌入式控制器OpenCR、兼容轮胎和履带的链轮，最后还有11.1V的锂聚合物电池。Waffle型号的特

⁴ <http://spectrum.ieee.org/automaton/robotics/diy/interview-turtlebot-inventors-tell-us-everything-about-the-robot>

⁵ <http://turtlebot3.robotis.com>

点是：其形状更易于装载物体、采用扭矩更大的舵机、采用基于Intel处理器的SBC、使用360度距离传感器LDS（Laser Distance Sensor），还有用于三维识别的深度摄像机（Depth Camera）Intel RealSense。TurtleBot3 Waffle Pi与Waffle具有相同的外形，但Waffle Pi的单板机采用了与Burger相同的Raspberry Pi，并且相机采用了Raspberry Pi Camera，因此大大提高了性价比。

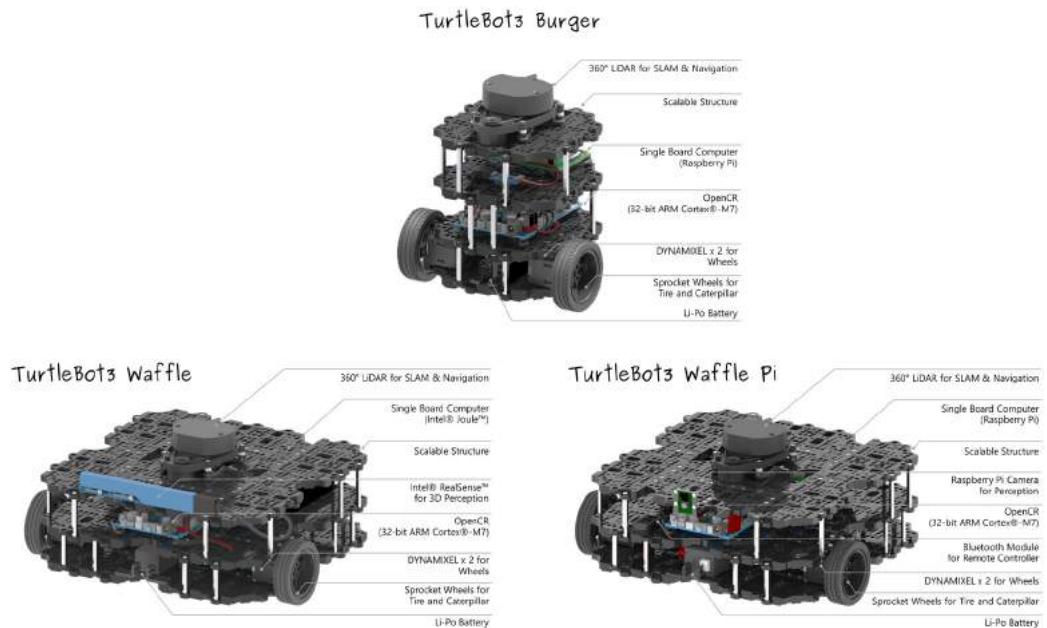


图 10-3 TurtleBot3的硬件配置

此外，TurtleBot的所有硬件结构信息都已在云共享三维CAD Onshape中公开，Onshape可供多人同时使用智能手机、平板电脑以及网页浏览器查看。您可以在网页浏览器中查看TurtleBot3的每个组件，也可以把想修改的零件下载到您的资源库，通过修改设计出自己想要的零件。也可以下载相关的STL文件后，用3D打印机打出零件。每个型号的公开文件可以在TurtleBot3的官方wiki⁶的附录提供的Open Source项目中找到。

⁶ <http://turtlebot3.robotis.com>

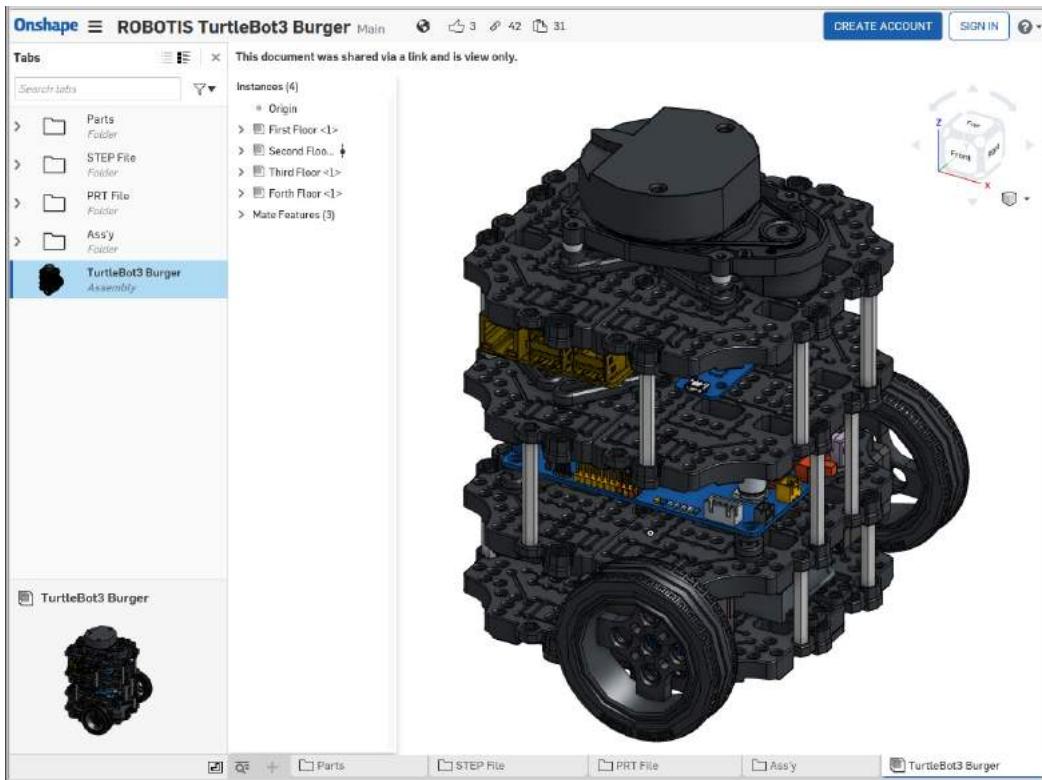


图 10-4 TurtleBot3的开源硬件



TurtleBot3官方维基

上面提到的TurtleBot3的硬件细节和本章介绍的基本内容也可以在以下链接（TurtleBot3官方wiki）中找到。如果您想用TurtleBot3学习ROS，请参考以下链接的信息。

<http://turtlebot3.robotis.com>



TurtleBot3的开源硬件

TurtleBot3的硬件设计文件像开源软件一样，都向公众开放。如果需要使用TurtleBot3的控制器OpenCR或需要TurtleBot3各型号的硬件文件，请使用以下链接地址。每个开源硬件如无特指，都遵循Open Source Hardware Statement of Principles and Definition v1.0许可证。

OpenCR: <https://github.com/ROBOTIS-GIT/OpenCR-Hardware>

TurtleBot3 Burger: <http://www.robotis.com/service/download.php?no=676>

TurtleBot3 Waffle:	http://www.robotis.com/service/download.php?no=677
TurtleBot3 Waffle Pi:	http://www.robotis.com/service/download.php?no=678
TurtleBot3 Friends OpenManipulator Chain:	http://www.robotis.com/service/download.php?no=679
TurtleBot3 Friends Segway:	http://www.robotis.com/service/download.php?no=680
TurtleBot3 Friends Conveyor:	http://www.robotis.com/service/download.php?no=681
TurtleBot3 Friends Monster:	http://www.robotis.com/service/download.php?no=682
TurtleBot3 Friends Tank:	http://www.robotis.com/service/download.php?no=683
TurtleBot3 Friends Omni:	http://www.robotis.com/service/download.php?no=684
TurtleBot3 Friends Mecanum:	http://www.robotis.com/service/download.php?no=685
TurtleBot3 Friends Bike:	http://www.robotis.com/service/download.php?no=686
TurtleBot3 Friends Road Train:	http://www.robotis.com/service/download.php?no=687
TurtleBot3 Friends Real TurtleBot:	http://www.robotis.com/service/download.php?no=688
TurtleBot3 Friends Carrier:	http://www.robotis.com/service/download.php?no=689

10.4. TurtleBot3软件

TurtleBot3的软件由OpenCR控制板的固件（FW）和4个ROS功能包组成。正如在第9章嵌入式系统中的说明，作为TurtleBot3的核心，OpenCR的固件还被称为turtlebot3_core。固件将OpenCR作为中间控制器，读取TurtleBot3的驱动舵机Dynamixel的编码器值来估算机器人的位置，或者根据上位软件的命令来控制速度。另外，固件还从安装在OpenCR上的3轴加速度和3轴陀螺仪传感器获得加速度和角加速度，以此估计机器人的方向，此外还测量电池电压并将其以话题传输。

TurtleBot3的ROS功能包包括turtlebot3、turtlebot3_msgs、turtlebot3_simulations和turtlebot3_applications。其中，turtlebot3功能包包括TurtleBot3的机器人模型、SLAM和导航功能包、遥控功能包以及与行驶相关的bringup功能包。另外，TurtleBot3的消息文件的集合turtlebot3_msgs、仿真功能包的集合turtlebot3_simulations以及应用程序的集合turtlebot3_applications构成了TurtleBot3的ROS功能包。



TurtleBot3的开源软件

TurtleBot3的软件都是开源的。在TurtleBot3中用作控制器的OpenCR的引导加载程序、用于与Arduino IDE兼容的固件、用于控制TurtleBot3的固件，等TurtleBot3的控制器的固件均已公开。此外，ROS功能包（turtlebot3、turtlebot3_msgs、turtlebot3_simulations 和turtlebot3_applications）也都以开源形式提供。这些开源软件的许可证根据每个源代码都有不同，基本上是Apache许可证2.0，有些软件使用3-clause BSD许可证和GPLv3。

<https://github.com/ROBOTIS-GIT/OpenCR>
<https://github.com/ROBOTIS-GIT/turtlebot3>
https://github.com/ROBOTIS-GIT/turtlebot3_msgs
https://github.com/ROBOTIS-GIT/turtlebot3_simulations
https://github.com/ROBOTIS-GIT/turtlebot3_applications

10.5. TurtleBot3的开发环境

如图10-5所示，TurtleBot3的开发环境可以分为远程PC（运行远程控制、SLAM和导航功能包）和TurtleBot PC（控制实际机器人且搜集传感器信息）。这两种PC在开发环境上都非常相似，但是它们使用的功能包根据PC的性能和用途进行了不同的配置。为了搭建一个基本的开发环境，两台PC都要安装Linux（Ubuntu 16.04兼容的Linux Mint和Ubuntu MATE）作为基本操作系统，且ROS安装Kinetic Kame版本即可。有关详细信息，请参阅第3章“搭建ROS开发环境”。有关PC、TurtleBot和OpenCR的信息，请参阅以下地址。

- <http://turtlebot3.robotis.com>

如果已经安装了Linux和ROS，则可以安装与TurtleBot3相关的软件。所有这些安装方法都在上面提到的维基地址中描述，但是这里我们简单地总结一下，只解释一下安装方法。在控制TurtleBot3机器人的用户PC（在这里我们称为远程PC）上，如下所示安装相关的依赖包和TurtleBot3功能包。但是，我们已经排除了包含本书中没有提到的多种示例的turtlebot3_applications功能包。

安装依赖包的方法 [Remote PC]

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation
```

安装TurtleBot3功能包 [Remote PC]

```
$ cd ~/catkin_ws/src/  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git  
$ cd ~/catkin_ws && catkin_make
```

接下来，在TurtleBot3机器人的PC（以下称为TurtleBot）中安装相关的依赖包、TurtleBot3功能包和传感器包。



安装依赖包的方法 [TurtleBot3]

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation
```

安装TurtleBot3功能包 [TurtleBot3]

```
$ cd ~/catkin_ws/src/  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git  
$ git clone https://github.com/ROBOTIS-GIT/hls_lfcd_lds_driver.git  
$ cd ~/catkin_ws && catkin_make
```



图 10-5 TurtleBot3的远程控制设置

如果已经安装了所有软件，则下一个最重要的配置是如图10-5所示的网络设置。有关如何更改ROS_HOSTNAME和ROS_MASTER_URI设置的详细说明，请参阅第3.2节和第8.3节，本节中只了解设置的顺序。作为参考，TurtleBot3将用户的个人台式机和笔记本电脑称为远程PC，这台PC将担任运行roscore的主节点，会负责远程控制、SLAM、导航等上层控制。与此PC配对的TurtleBot3配备了SBC，负责机器人行驶和传感器信息采集。以下远程控制设置示例是在远程PC上运行ROS Master时的示例。

查看远程PC的IP值

在终端窗口中，使用ifconfig命令查看远程PC的IP值（例如，192.168.7.100）。

远程PC的ROS_HOSTNAME、ROS_MASTER_URI设置

修改~/.bashrc文件中的ROS_HOSTNAME和ROS_MASTER_URI设置，如下所示：

```

export ROS_HOSTNAME=192.168.7.100
export ROS_MASTER_URI=http://${ROS_HOSTNAME}:11311

```

查看TurtleBot的IP值

在终端窗口中使用ifconfig命令查看TurtleBot的IP值。例如，假设TurtleBot的IP是192.168.7.200。需要注意，TurtleBot必须使用与远程PC相同的网络。

TurtleBot3 SBC的ROS_HOSTNAME及ROS_MASTER_URI设置

如下修改~/.bashrc文件中的ROS_HOSTNAME和ROS_MASTER_URI设置。

```
export ROS_HOSTNAME=192.168.7.200  
export ROS_MASTER_URI=http://192.168.7.100:11311
```

以此搭建了所有TurtleBot3的开发环境。在下一节中，我们将以远程控制为开始，使用TurtleBot3的各种ROS功能包控制TurtleBot3。

10.6. TurtleBot3远程控制

让我们来看一看Turtlebot的远程控制。几乎所有可连接到PC的设备，如键盘、蓝牙遥控器RC-100B、PS3游戏杆、XBOX 360游戏杆、Wii遥控器、Nunchuk、Android应用程序、LEAP Motion，和Myo等都可以用于远程遥控TurtleBot3。更多信息可以在“<http://turtlebot3.robotis.com/>”的teleoperation项目中查看。在本节中，我们将以最易于使用的键盘和通常用于机器人控制的PS3操纵杆进行说明。

10.6.1. 遥控TurtleBot3

运行roscore [Remote PC]

在远程PC上，使用以下命令启动roscore。roscore只需运行一次。

```
$ roscore
```

运行turtlebot3_robot.launch启动文件[TurtleBot]

如下所示，在TurtleBot中，运行turtlebot3_robot.launch文件。这个启动文件将运行负责turtlebot3_core和hls_lfcd_lds_driver节点，其中turtlebot3_core负责与TurtleBot3的控制器OpenCR的通信，而hls_lfcd_lds_driver节点负责运行360度距离传感器LDS。

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch --screen
```



--screen 选项

roslaunch同时运行多个节点。默认情况下，不显示来自每个节点的消息。如有必要，可以使用--screen选项查看启动过程中发生的所有隐藏的消息。在运行roslaunch时，我推荐使用这个选项。

运行turtlebot3_teleop_key.launch启动文件[Remote PC]

在远程PC上，按如下所示运行turtlebot3_teleop_key.launch启动文件。

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch --screen
```

运行该启动文件时，会运行turtlebot3_teleop_keyboard节点，并在终端窗口中显示以下消息。该节点用键盘的w、a、d和x键将平移速度（单位为m/sec）和旋转速度（单位为rad/sec）发送给机器人，并用空格键或s键将平移速度和旋转速度均设为0（停止机器人）。

```
Control Your Turtlebot3!
-----
Moving around:
 w
 a s d
 x

w/x: increase/decrease linear velocity
a/d: increase/decrease angular velocity
space key, s : force stop

CTRL-C to quit
```

如果要使用PS3操纵杆代替键盘来操作机器人，请按如下方式在远程PC上安装依赖包。然后运行teleop_twist_joy功能包中的teleop.launch启动文件，则可以用PS3游戏杆控制机器人。此时，PS3游戏杆必须通过蓝牙连接到远程PC。

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-joystick-drivers ros-kinetic-teleop-twist-joy
$ roslaunch teleop_twist_joy teleop.launch --screen
```

10.6.2. 可视化TurtleBot3

我们将利用RViz来可视化机器人的状态。为了确定要使用的模型，先使用export命令将当前的TurtleBot3的型号指定为Burger。如果是Waffle或Waffle Pi，您可以将其指定为’ waffle’ 或’ waffle_pi’ ，而不是’ burger’ 。然后运行turtlebot3_model.launch启动文件，则RViz将会被运行。

```
$ export TURTLEBOT3_MODEL=burger  
$ roslaunch turtlebot3_bringup turtlebot3_remote.launch  
$ rosrun rviz rviz -d `rospack find turtlebot3_description`/rviz/model.rviz
```

如图10-6所示，当RViz被运行时，TurtleBot3 Burger的模型和各关节的tf会以RGB坐标系的形式显示在画面中央。而且，可以看到安装在机器人上的360度距离传感器LDS接收距离值，并将障碍物显示为红色。

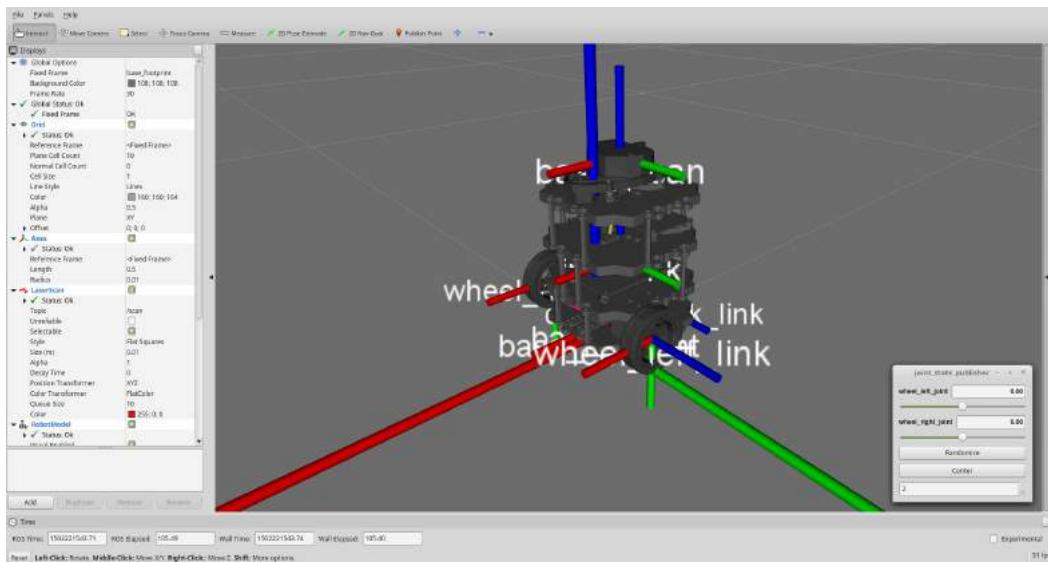


图 10-6 TurtleBot3的可视化

如上面所述，对TurtleBot3的控制需要在远程PC和TurtleBot3 SBC的计算机上回来操作，是比较麻烦的方式。为了解决这个问题，我想建议一种使用SSH来远程访问TurtleBot3 SBC的方法。这使您可以在远程PC上执行所有的命令。从远程PC远程连接到TurtleBot3 SBC的操作如下。有关更多信息，请参阅有关SSH的说明。

```
$ ssh turtlebot@192.168.7.200
```



SSH (Secure Shell, 安全外壳)

SSH是指一种应用程序或用于该应用程序的协议，它允许您登录到网络上的另一台计算机，或在远程系统上运行命令并将文件复制到另一个系统。它多用于在Linux环境中从终端窗口访问另一台计算机并发送远程指令的情况。要做到这一点，您需要按如下操作安装ssh程序。

```
$ sudo apt-get install ssh
```

要连接到另一台计算机，请在终端窗口中用如下命令访问。连接之后的用法与其他命令一样。

```
$ ssh 用户名@笔记本电脑IP
```

使用Raspberry Pi的情况下（TurtleBot3 Burger和Waffle Pi），由于Ubuntu MATE 16.04.x和Raspbian的SSH服务器默认是未激活的。如果要激活SSH，请参考如下链接的文档进行设置。

<https://www.raspberrypi.org/documentation/remote-access/ssh/>

<https://ubuntu-mate.org/raspberry-pi/>

10.7. Turtlebot3话题

如果在远程PC上只运行roscore，并且没有运行任何其他节点的情况下使用rostopic list命令查看话题列表，则只能看到/rosout和/rosout_agg。在此基础上，正如在TurtleBot3远程控制中的操作一样，可以在TurtleBot3 SBC的终端窗口中运行turtlebot3_robot.launch启动文件，以此驱动TurtleBot3。当TurtleBot3被驱动时，turtlebot3_core节点和turtlebot3_lds节点会被运行，并且可以用话题的方式接收从每个节点发布的关节的状态、电机驱动单元和IMU等内容。

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch --screen
```

例如，您可以使用rostopic list命令来查看正在发布或订阅的各种话题。

```
$ rostopic list
/cmd_vel
/cmd_vel_rc100
```

```
/diagnostics  
/imu  
/joint_states  
/odom  
/rosout  
/rosout_agg  
/rpms  
/scan  
/sensor_state  
/tf
```

进一步地，像在TurtleBot远程控制中的操作一样，在远程PC上运行turtlebot3_teleop_key.launch启动文件。

```
$ rosrun turtlebot3_teleop turtlebot3_teleop_key.launch --screen
```

要获得更详细的节点和话题的信息，请运行rqt_graph，如下例所示。则可以查看在TurtleBot中发布和订阅的话题，如图10-7所示。

```
$ rqt_graph
```

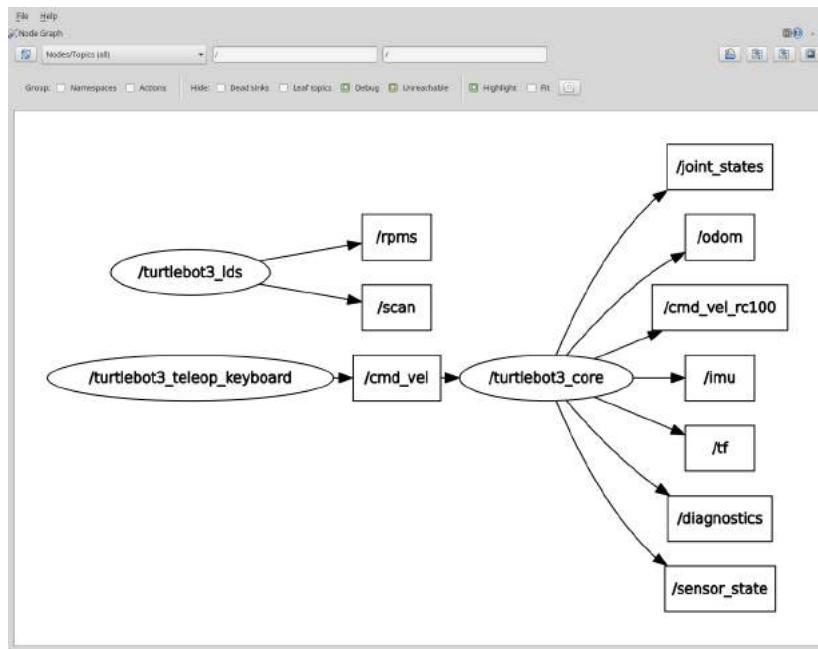


图 10-7 TurtleBot3的节点和话题

10.7.1. 订阅话题

上述话题可以分为TurtleBot3收到的订阅话题和TurtleBot3发送的发布话题。其中，订阅话题如下。您不需要知道所有的订阅话题。下面通过测试来了解几种订阅话题的用法。所有的话题当中，“cmd_vel”是最需要知道的话题。这是控制机器人的最实用的话题，用户可以通过这个话题来控制机器人的前进、后退和左右旋转。

名称	形式	功能
motor_power	std_msgs/Bool	Dynamixel舵机On/Off
reset	std_msgs/Empty	重置测位 (odometry and IMU)
sound	turtlebot3_msgs/Sound	发出“哔”声
cmd_vel	geometry_msgs/Twist	控制移动机器人的平移和旋转速度， 单位分别是米/秒和弧度/秒 (实际控制机器人运动)

*TurtleBot3中使用的话题会根据目的而有变化。

表 10-1 TurtleBot3的订阅话题

10.7.2. 通过订阅话题控制机器人

上述订阅话题的过程是，用户发布话题后由机器人接收和处理该话题。很难测试本书中的每个话题，所以下面只使用几种订阅话题作为示例。下面例子是在终端窗口中使用rostopic pub命令停止电机的例子。

```
$ rostopic pub /motor_power std_msgs/Bool "data: 0"
```

接下来，为了让TurtleBot动起来，让我们控制机器人的速度吧。这里使用的x和y是平移速度，单位是ROS标准中的m/s。z是以rad/s为单位的转速。如下面的例子所示，当x的值为0.02时，TurtleBot3在x轴方向上以0.02m/s的速度前进。

```
$ rostopic pub /cmd_vel geometry_msgs/Twist "linear:  
x: 0.02  
y: 0.0  
z: 0.0  
angular:  
x: 0.0  
y: 0.0  
z: 0.0"
```

如下例所示，当z值设为1.0时，TurtleBot3相对于z轴以逆时针1.0弧度/秒的转速旋转。

```
$ rostopic pub /cmd_vel geometry_msgs/Twist "linear:  
x: 0.0  
y: 0.0  
z: 0.0  
angular:  
x: 0.0  
y: 0.0  
z: 1.0"
```

10.7.3. 发布话题

TurtleBot3发布的主要的话题可以分为与诊断（diagnostics）相关的话题、与调试相关的话题以及与传感器相关的话题。其他话题还包括与关节（joint_states）相关的话题、与控制器信息（controller_info）相关的话题、与测位（odometry）和转换（tf）相关的话题。

您不需要知道所有的发布话题，下面通过测试来了解几种发布话题的用法。尤其要留意到是，包含测位（odometry）信息的odom、坐标变换信息tf、关节信息joint_states和与传感器相关的信息，它们是将来使用TurtleBot时必不可少的话题。

名称	形式	功能
sensor_state	turtlebot3_msgs/SensorState	这是一个可以查看安装在TurtleBot3上的传感器值的话题。
battery_state	sensor_msgs/BatteryState	可以获得诸如电池电压等状态值。
Scan	sensor_msgs/LaserScan	这是一个可以查看安装在TurtleBot3上的雷达的扫描值的话题。
Imu	sensor_msgs/Imu	这个话题包含机器人的方向值，此方向值由加速度传感器和陀螺仪传感器值计算而得。
Odom	nav_msgs/Odometry	根据编码器和IMU信息，可以获得TurtleBot3的测位（odometry）信息。
Tf	tf2_msgs/TFMessage	它包含TurtleBot3的坐标转换值，例如base_footprint和odom等。
joint_states	sensor_msgs/JointState	将左右车轮看作关节时，可以查看位置、速度和力。各单位为位置：米，速度：米/秒，力：N·米。

名称	形式	功能
Diagnostics	diagnostic_msgs/ DiagnosticArray	可以获得自检信息。
version_info	turtlebot3_msgs/ VersionInfo	可以获得TurtleBot3的硬件、固件和软件等信息。
cmd_vel_rc100	geometry_msgs/Twist	这是使用RC-100B（一种基于蓝牙的控制器）时使用的话题，用于移动机器人的速度控制并会订阅此话题。单位使用m/s和rad/s。

10.7.4. 通过发布话题识别机器人状态

上面提到的发布话题以话题的形式传输机器人的传感器值、电机状态和机器人的位置。在本节中，我们尝试接收一些话题并查看机器人的当前状态。

sensor_state话题主要涉及连接到嵌入式控制板OpenCR的仿真传感器，如下例所示，您可以获取bumper、cliff、button、left_encoder和right_encoder等信息。

```
$ rostopic echo /sensor_state
stamp:
secs: 1500378811
nsecs: 475322065
bumper: 0
cliff: 0
button: 0
left_encoder: 35070
right_encoder: 108553
battery: 12.0799999237
---
```

利用odom话题可以获取Odometry信息，这相当于行车记录器。通过此话题，可以获得基于陀螺仪和编码器的TurtleBot3的测位（odometry）信息，这在移动机器人系列中是必要的信息。这也是对导航必要的信息，因此请务必掌握。

tf话题是以相对坐标形式描述的机器人的各个关节的姿势（位置和方向）信息，例如XY平面上的机器人的中心位置base_footprint与测位（odometry）信息odom之间的坐标转换。

```
$ rostopic echo /tf
transforms:

-
  header:
    seq: 0
    stamp:
      secs: 1500379130
      nsecs: 727869913
    frame_id: odom
  child_frame_id: base_footprint
  transform:
    translation:
      x: 3.55720019341
      y: 0.655082404613
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.112961538136
      w: 0.993599355221
---
```

还可以使用rqt中的tf_tree插件（如图10-8所示），在GUI环境中查看。在图10-8中，由于未包括机器人的模型信息，所以各坐标未被连接，但是如果包含了机器人的模型信息后进行坐标变换，则可以使用机器人的每个关节的连接信息，如图10-14所示。

```
$ rosrun rqt_tf_tree rqt_tf_tree
```

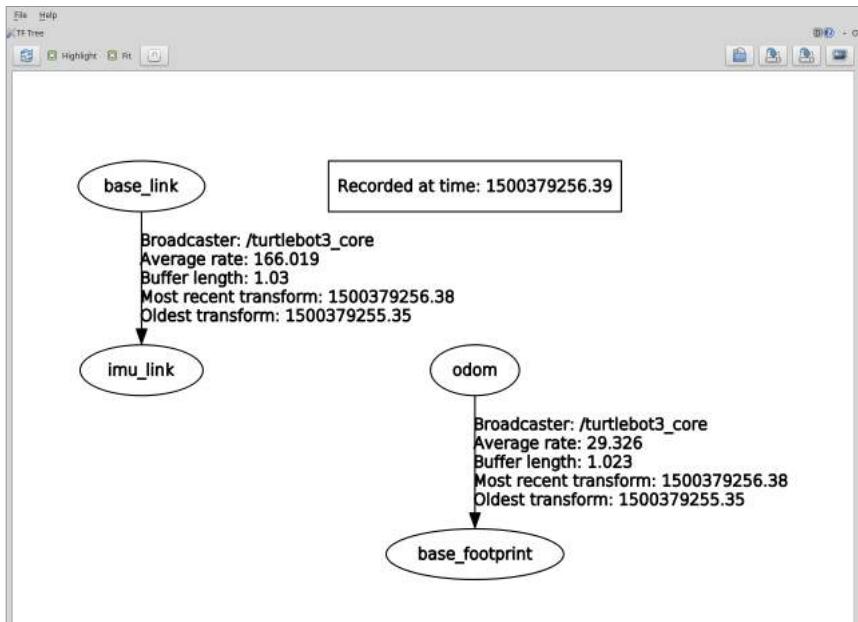


图 10-8 使用tf_tree查看坐标变换

至此，说明了关于话题的内容。在ROS中，节点作为各自独立的处理器，他们之间的通信方式有话题、服务和动作，其中话题是使用最广泛的消息通信方法，因此必须要掌握。

10.8. 使用RViz 仿真 TurtleBot3

10.8.1. 仿真

TurtleBot3提供虚拟仿真开发环境，即使没有机器人硬件，也可以通过仿真软件里的虚拟机器人进行编程和仿真。有两种方法可以做到这一点，一种是使用ROS的3D可视化工具RViz，另一种是使用3D机器人仿真器Gazebo。

在本节中，我们将了解第一个方法RViz的用法。即使您没有TurtleBot3的硬件，您也可以遥控TurtleBot3，还可以测试SLAM和导航，因此是非常有用的方法。具体方法是使用turtlebot3_simulations元功能包。要在这个元功能包中使用虚拟仿真，首先需要安装turtlebot3_fake功能包。这在“10.5. TurtleBot3开发环境”中做了说明。如果已经安装了它，请转到下面的内容。

10.8.2. 运行虚拟机器人

要运行虚拟机器人，请按如下所示运行turtlebot3_fake功能包的turtlebot3_fake.launch文件。

```
$ export TURTLEBOT3_MODEL=burger  
$ roslaunch turtlebot3_fake turtlebot3_fake.launch
```

它运行turtlebot3_fake_node节点和robot_state_publisher节点。其中，turtlebot3_fake_node节点从turtlebot3_description功能包导入TurtleBot3的三维模型，并发布实际机器人发布的话题。robot_state_publisher节点则通过接收机器人的两个车轮的旋转值，以TF形式发布两个车轮及各关节的三维位置和方向信息。但是，由于在RViz中无法使用传感器信息，所以需要使用包含物理引擎的3D仿真器Gazebo。下一节将介绍Gazebo，在本节中，我们只介绍在简单的运动和移动过程中可以确认的Odometry和TF。

本来，下一步操作是运行RViz并在左侧显示窗口中将 [Global Options] → [fixed frame] 更改为 “/odom”，然后按下显示窗口左下方的添加按钮，再点击 “RobotModel” 来添加机器人模型，但是由于我们已经从turtlebot3_fake.launch文件中加载了TurtleBot3的3D模型，所以您会在画面中央看到TurtleBot3的3D模型，如图10-9所示。

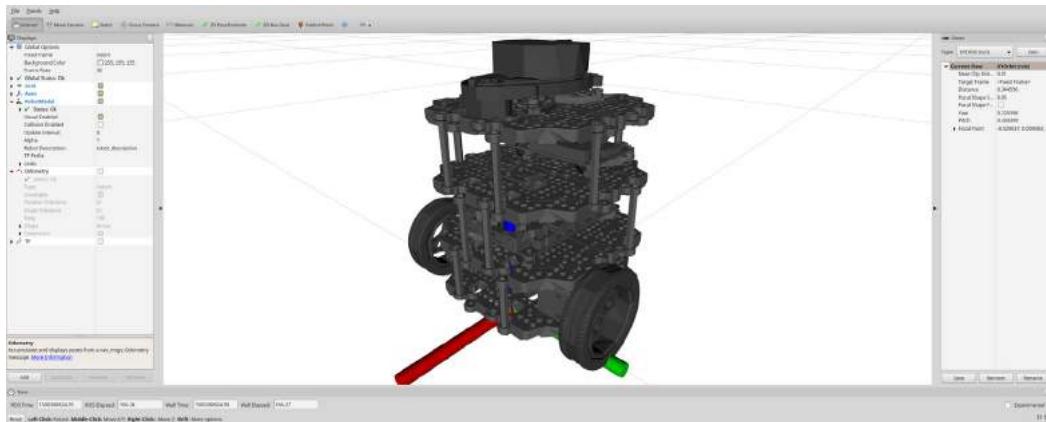


图 10-9 运行虚拟机器人

接下来，我们来运行这个虚拟机器人。运行turtlebot3_teleop功能包中的turtlebot3_teleop_key.launch文件，该文件允许用键盘操纵机器人。

```
$ rosrun turtlebot3_teleop turtlebot3_teleop_key.launch
```

运行turtlebot3_teleop_key.launch文件将启动turtlebot3_teleop_keyboard节点。turtlebot3_fake_node节点接收由turtlebot3_teleop_keyboard节点通过/cmd_vel话题发送的平移速度和旋转速度，并将其作为驱动命令，并用此命令驱动虚拟机器人。在执行turtlebot3_teleop_key.launch文件的终端窗口中，让我们直接使用下面的键运行机器人。

- w 键：前进(+0.01, 单位=m/sec)
- x 键：后退(-0.01, 单位=m/sec)
- a 键：逆时针方向旋转(+0.1, 单位=rad/sec)
- d 键：顺时针方向旋转(-0.1, 单位=rad/sec)
- 空格或者s键：初始化平移速度及旋转速度
- Ctrl + C：退出

10.8.3. Odometry和TF

我们已经尝试了虚拟机器人的驱动，下面让我们查看其他话题值。例如，如图10-10所示，turtlebot3_fake_node节点会接收速度命令以生成测位（odometry）信息并将其以话题形式发布，还可以发布joint states或tf，从而在RViz中查看TurtleBot3的移动情况。

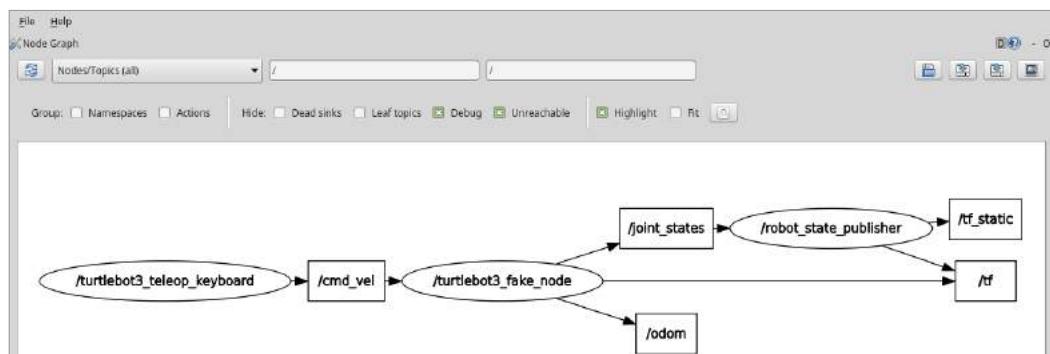


图 10-10 使用rqt_graph查看的节点和话题

首先来确认测位 (odometry) 信息是否正确地生成和发布。您可以在终端窗口中使用“rostopic echo /odom”命令来检查，但我们这次是在使用RViz，因此我们可以直观地检查它。单击RViz左下角的“Add”按钮，然后单击“By Topic”标签，如图10-11所示，选择“Odometry”来添加它。屏幕上会出现一个红色的箭头，表示TurtleBot3前方的测位 (odometry)。首先，在显示窗口中关闭“Odometry”中的“Covariance”复选框，并且因为箭头初始值比机器人大很多，因此需要适当设置“Shape/Shft Length”和“Shape/Head Length”。



图 10-11 为检查odom话题添加odometry的显示功能

现在，让我们再次使用turtlebot3_teleop_keyboard节点移动虚拟TurtleBot3。如图10-12所示，与之前不同，可以看到红色箭头会沿着机器人的移动轨迹显示。这个测位 (odometry) 信息对于移动机器人来说是表示自身位置和方向的非常基本的信息。至此，我们简单查看了测位 (odometry) 信息是否正确显示。

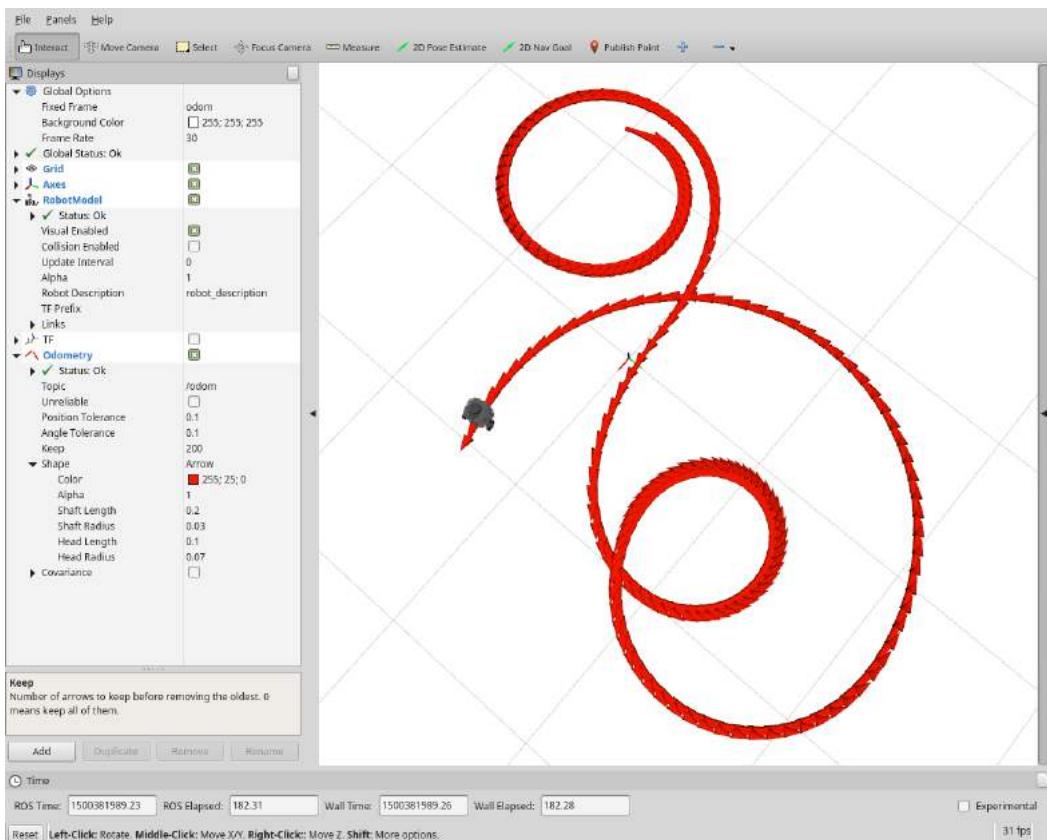


图 10-12 虚拟机器人移动和测位 (odometry) 信息

TF话题包含TurtleBot3组件的相对坐标信息，它可以像之前的操作一样用rostopic命令确认，但是我们下面将用RViz进行确认，就像查看odom的方式一样。并用rqt_tf_tree检查层次结构。

按下RViz左下角的Add按钮，然后选择“TF”。则如图10-13所示，会显示Odom、base_footprint、imu_link、wheel_left_link和wheel_right_link。让我们再次使用turtlebot3_teleop_keyboard节点移动虚拟TurtleBot3。当TurtleBot3移动时，wheel_left_link和wheel_right_link会旋转。

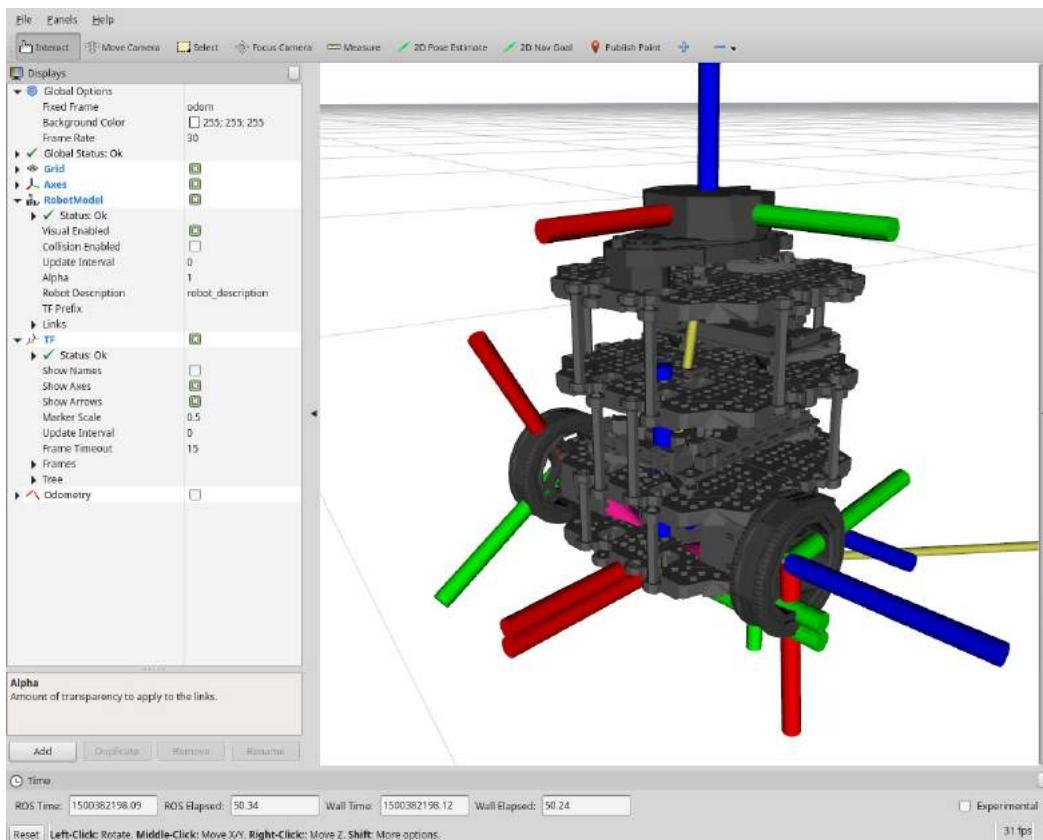


图 10-13 查看RViz中的tf话题

下面使用以下命令运行rqt_tf_tree。我们可以看到每个部分的元素的相对位置会通过tf进行变换，且彼此相关联，如图10-14所示。以后可以通过这个方法，表示安装在机器人上的传感器的位置。这在下一章中会有更详细的介绍。

```
$ rosrun rqt_tf_tree rqt_tf_tree
```

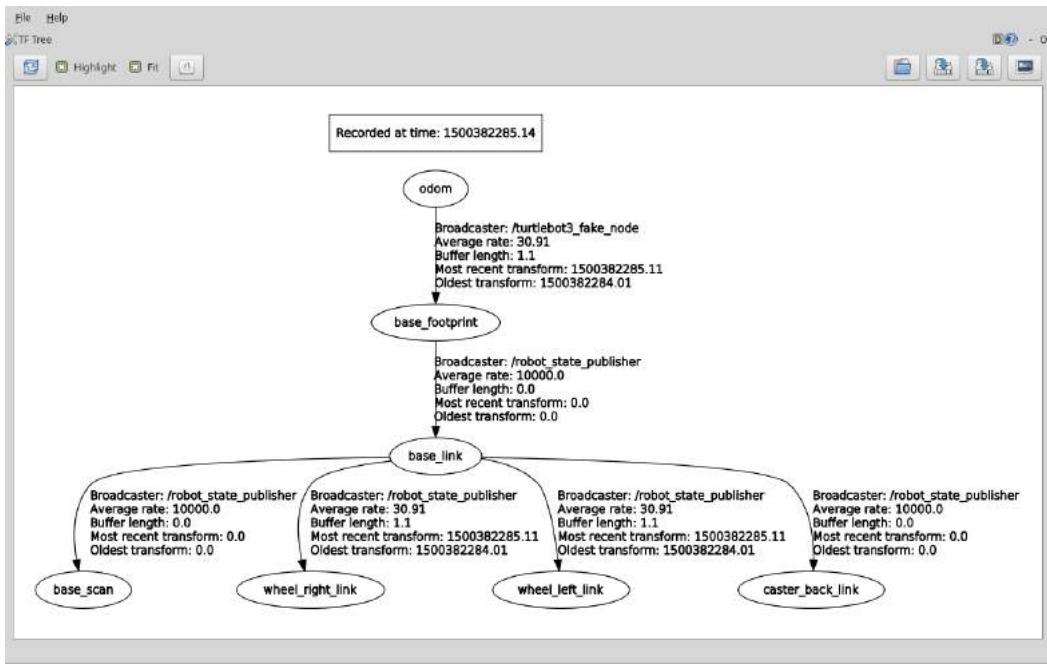


图 10-14 通过rqt_tf_tree查看tf话题

10.9. 利用Gazebo仿真TurtleBot3

10.9.1. Gazebo仿真器

Gazebo是一款3D仿真器，支持机器人开发所需的机器人、传感器和环境模型，并且通过搭载的物理引擎可以得到逼真的仿真结果。Gazebo是近年来最受欢迎的三维仿真器之一，并被选为美国DARPA机器人挑战赛⁷的官方仿真器。因此它再接再厉，即便是开源仿真器，却具有高水准的仿真性能，因此在机器工程领域中非常流行。不仅如此，负责开发和普及ROS，且担任社区的Open Robotics在开发ROS和Gazebo，因此ROS和Gazebo非常兼容。

⁷ <http://www.darpa.mil/program/darpa-robotics-challenge>

Gazebo的特征⁸如下。

- 动力学仿真：在最初的版本中，只支持ODE（开放式动力引擎），但从3.0版本开始，各种物理引擎如Bullet、Simbody和DART被用来满足不同用户的需求。
- 3D图形：Gazebo采用经常在游戏中使用的OGRE（开源图形渲染引擎），因此不仅可以实现机器人模型，还可以逼真地表达光、阴影和材质。
- 支持传感器和噪声：支持虚拟的激光测距仪（LRF）、2/3D相机、深度相机、触摸传感器、力矩传感器，并且在检测到的数据中包含与真实世界相似的噪声。
- 可添加插件：提供API，以便用户可以以插件的形式亲手创建机器人、传感器和环境控制等。
- 机器人模型：PR2、Pioneer2 DX、iRobot Create和TurtleBot已经以SDF格式存在于Gazebo中。SDF格式是一个Gazebo模型文件格式。此外，用户可以添加自己创建的SDF格式的机器人。
- TCP/IP数据传输：仿真也可以在远程服务器上执行，这是使用Google的protobufs（基于socket的消息传递）实现的。
- 云仿真：提供CloudSim云仿真环境，因此可以在Amazon、Softlayer和OpenStack等云环境中使用Gazebo。
- 命令行工具：不仅可以使用GUI界面，还可以使用CUI风格的命令行工具来查看和控制仿真过程。

在写这本书的时候，Gazebo的版本是8.0。就在五年前，它还是1.9版本，但在短短的5年当中已经更新到了8.0版本。当前版本8.0是本书使用的ROS Kinetic Kame版本的默认版本，如果已经按照“3.1 ROS安装”中的说明进行安装，则无需安装即可使用。

现在我们来运行Gazebo。运行以下命令，如果没有问题，则可以看到Gazebo正在运行，如图10-15所示。到目前为止，它可以被看作是与ROS无关的独立的仿真器。

```
$ gazebo
```

⁸ <http://gazebosim.org/>

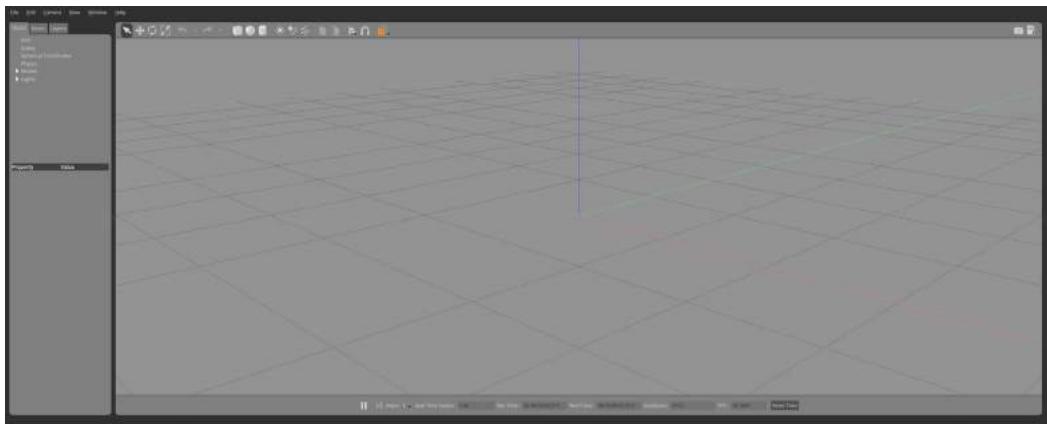


图 10-15 Gazebo初始画面

10.9.2. 启动虚拟机器人

为了在Gazebo运行TurtleBot，先安装相关的功能包。要安装的功能包有gazebo_ros_pkgs metapack和turtlebot3_gazebo。前者的功能是连接Gazebo和ROS，已经安装。后者是与TurtleBot3的三维仿真有关的功能包，也已在“10.5. TurtleBot3开发环境”中安装。

下面是在Burger、Waffle和Waffle Pi中选择型号的命令。在这里，我们使用可以检查相机信息的Waffle模型。使用以下命令将TURTLEBOT3_MODEL变量设置为waffle。作为参考，在`~/.bashrc`文件中记录以下模型变量，则不必每次输入命令。

```
$ export TURTLEBOT3_MODEL=waffle
```

下面如以下示例所示运行启动文件。那么将同时运行gazebo、gazebo_gui、mobile_base_nodelet_manager、robot_state_publisher和spawn_mobile_base节点，且TurtleBot3会出现在Gazebo屏幕上，如图10-16所示。Gazebo是一款3D仿真器，由于使用了物理引擎和图形效果，因此会占据大量的CPU、GPU和RAM的资源。根据您的PC的规格，可能需要相当长的时间来加载。

```
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

如下图所示，可以看到只显示了机器人，因为没有指定任何环境选项。

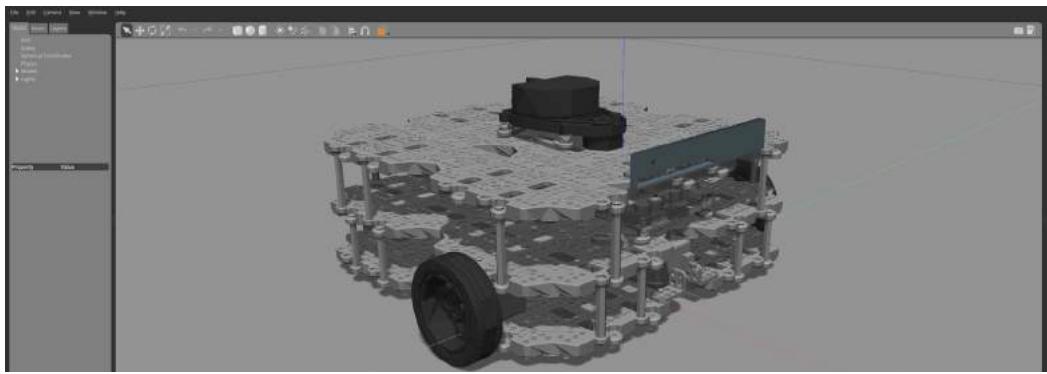


图 10-16 Gazebo上的TurtleBot3的3D形象

这只是在Gazebo加载了机器人，为了进行实际的仿真，用户可以指定环境或者加载Gazebo提供的环境模型。要从Gazebo加载环境模型，您可以通过点击屏幕顶部的“Insert”并选择一个文件来添加环境模型。除了环境模型以外，还有各种机器人和物体的模型，必要时都可以添加。

在本书中，为了使读者使用和我相同的环境，我们将使用现有的开发环境。关闭当前活动的Gazebo屏幕。如果要退出Gazebo，请点击屏幕右上角的X按钮，或者在开始Gazebo的终端窗口中键入[Ctrl+c]即可。

再次运行turtlebot3_world.launch文件，如下所示。turtlebot3_world.launch文件将加载我们已经创建的turtlebot3.world环境模型。turtlebot3.world环境模型是通过模仿TurtleBot系列图标创建的，如图10-17所示。如果您想知道如何做到这一点，请查看turtlebot3_gazebo功能包中的/models/turtlebot3.world文件来了解它是如何制作的。

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

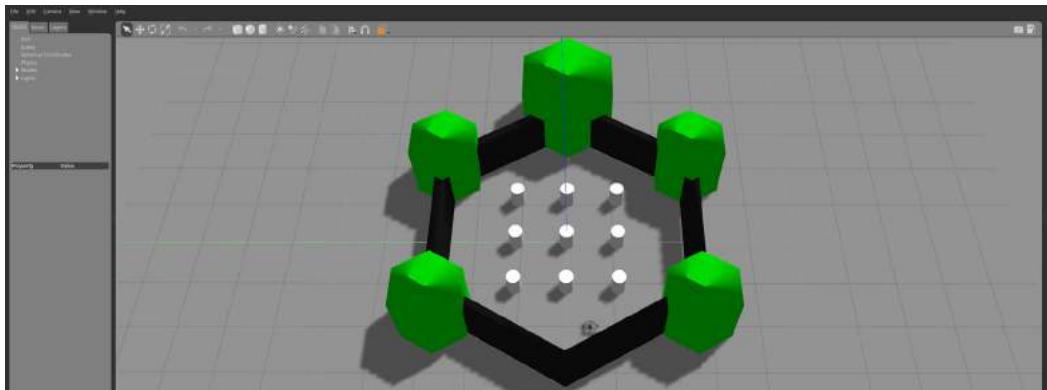


图 10-17 加载了TurtleBot3和环境模型的画面

现在运行下面例子中的远程控制launch文件，这样就可以在Gazebo环境中用键盘控制虚拟TurtleBot3。

```
$ rosrun turtlebot3_teleop turtlebot3_teleop_key.launch
```

至此，会与上一节中介绍的使用RViz的仿真相同。然而，Gazebo不仅提供虚拟机器人的外形，还可以检测机体的碰撞，也可以测量位置，还能虚拟地使用IMU传感器和摄像机传感器。一个使用这些功能的例子是下面的启动文件。启动后，虚拟TurtleBot3在规定的环境中随机移动，如图10-18所示，以避免障碍物或撞墙。这是学习Gazebo的一个很好的例子。

```
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun turtlebot3_gazebo turtlebot3_simulation.launch
```

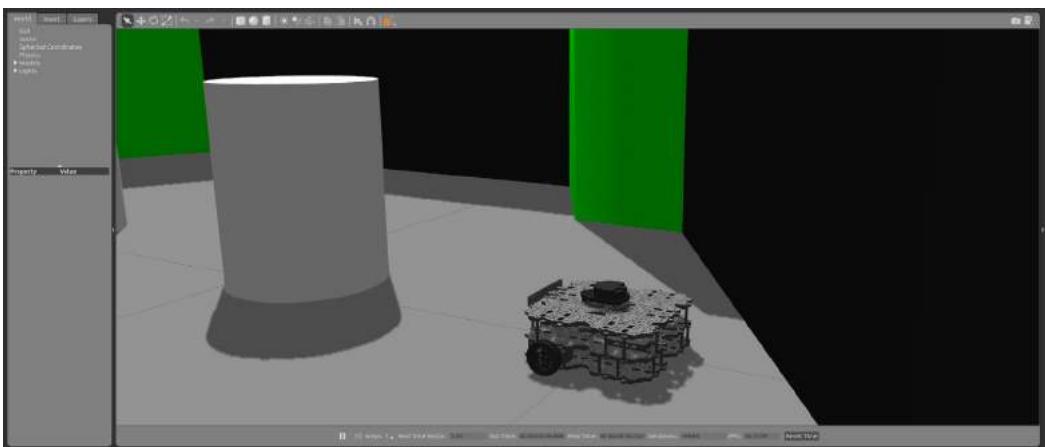


图 10-18 在Gazebo中自动避障行驶的TurtleBot3

下一步，我们用以下命令调用RViz。如图10-19所示，RViz可以检查Gazebo中运行的机器人的位置、距离传感器值和摄像机图像。这与通过RViz观察机器人的状态是一样的，就像Gazebo本身是一个环境，同时也在运行机器人一样。

```
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
```

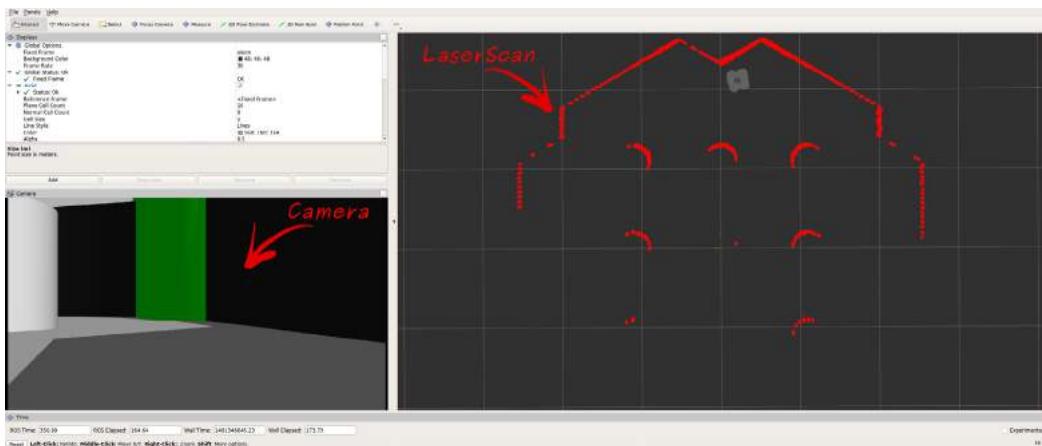


图 10-19 在Gazebo中查看影像和距离值

10.9.3. 虚拟SLAM和导航

在下一章中，我们将介绍使用TurtleBot3创建地图的SLAM和移动到地图中指定的目的地的导航，这里使用的SLAM和导航也可以在上面提到的Gazebo中实现。我建议在下一章中掌握了SLAM和导航之后再通过Gazebo实现SLAM和导航，而在这一章只列出使用方法。在学完下一章后请再自行尝试。

虚拟SLAM运行顺序

当运行了相关功能包，且在虚拟空间中移动机器人，并创建地图时，您可以创建如图10-20所示的地图，并获得类似于结果10-21的地图。

运行Gazebo

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

运行SLAM

```
$ export TURTLEBOT3_MODEL=waffle
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```

运行RViz

```
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun rviz rviz -d `rospack find turtlebot3_slam`/rviz/turtlebot3_slam.rviz
```

远程操作Turtlebot

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

显示地图

```
$ rosrun map_server map_saver -f ~/map
```

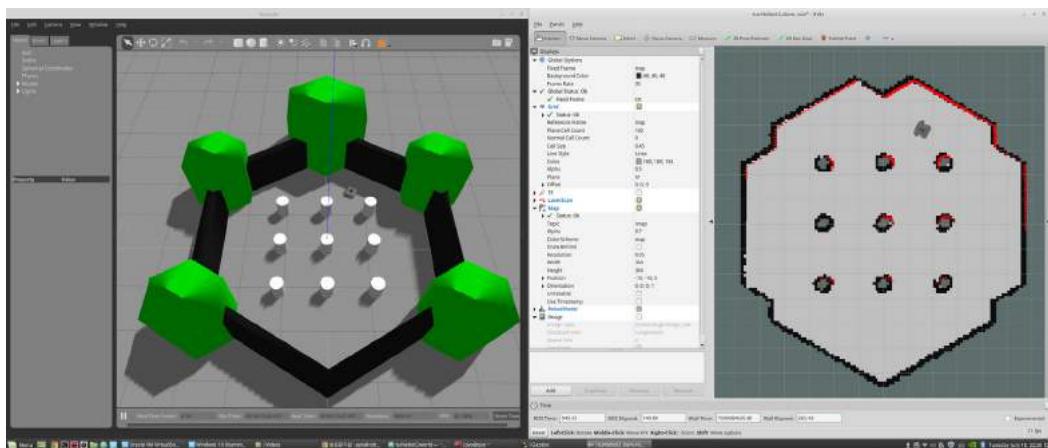


图 10-20 在Gazebo中运行SLAM的影像（左：Gazebo，右：Rviz）

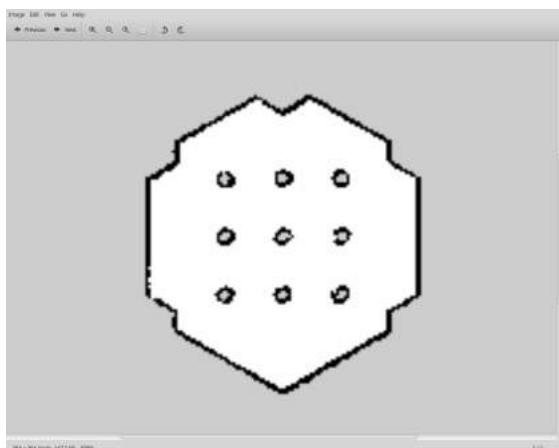


图 10-21 已生成的Gazebo环境地图

虚拟导航运行顺序

在运行导航之前退出所有执行的程序。然后，如下所示运行相关功能包时，机器人将显示在之前创建的地图上。在RViz上设置机器人的初始位置并设置目的地后，可以看到机器人移动到目的地，如图10-22所示。请注意，初始位置在开始时只能指定一次。

运行Gazebo

```
$ export TURTLEBOT3_MODEL=waffle  
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

运行导航

```
$ export TURTLEBOT3_MODEL=waffle  
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

运行RViz并设置目的地

```
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun rviz rviz -d `rospack find turtlebot3_navigation`/rviz/turtlebot3_nav.rviz
```

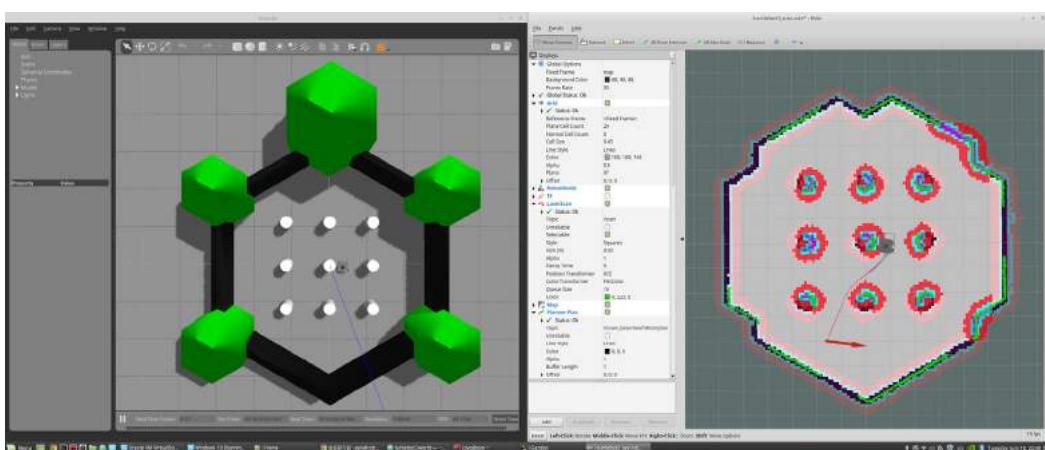


图 10-22 在Gazebo中运行导航的影像（左：Gazebo，右：RViz）

至此，我们介绍了TurtleBot3功能包中的两种仿真工具。一种是使用ROS的3D可视化工具RViz，另一种是使用3D机器人仿真器Gazebo。仿真可以在逼近实际的机器人和场景中完成，而不需要实际的机器人，所以对于需要虚拟仿真的用户来说，这将是一个很好的工具。



TurtleBot的仿真

Turtlebot支持三种类型的仿真（stage、stdr、gazebo）。当希望用虚拟机器人进行各种仿真时，请参考以下相关wiki。

http://wiki.ros.org/turtlebot_std

http://wiki.ros.org/turtlebot_gazebo

http://wiki.ros.org/turtlebot_stage

第11章

SLAM和导航

11.1. 导航及其组成要素

这里的导航（navigation）可以理解为安装在汽车里的导航仪。在驾驶汽车时，只要在导航仪中设置目的地，就可以知道从当前位置到目的地的准确距离和所需时间，另外还可以设置中途路过的地点或指定公路。

我们现今使用的导航仪使用方便，但其历史相对较短。1981年日本本田首先推出了一种基于三轴陀螺仪和胶卷地图的叫做“Electro Gyrocator”¹的模拟（非数字）方法。后来美国汽车产品公司Etak推出了利用带传感器的电子罗盘和车轮的电子导航仪（Etak Navigator）²。然而，将传感器安装在电子罗盘和车轮上对于已经很昂贵的汽车来说是大的负担，而且还具有可靠性方面的问题。自二十世纪七十年代以来，美国一直在开发用于军事的卫星定位系统，二十一世纪二十年代开放了二十四颗全球定位系统（GPS）³卫星，并开始使用三角测量导航系统。

11.1.1. 移动机器人的导航

让我们把话题转移到机器人上吧。导航是移动机器人技术的基本目的之一，同时也是一颗明珠。机器人技术中导航非常重要，是必不可少的部分。导航是指机器人运动到一个指定的目的地，这说起来很容易，但完成它所需的技术一个个都不是容易的任务：要知道机器人本身在哪里，并要有一个给定的周围环境的地图，在各种路径中找出最优路径，在行驶中避免障碍物（如墙壁、家具、物体）等。

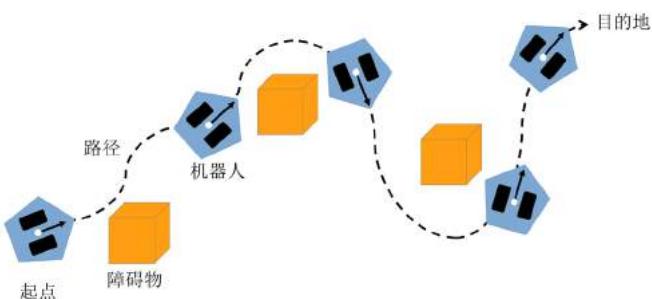


图 11-1 导航

¹ https://en.wikipedia.org/wiki/Electro_Gyrocator

² <https://en.wikipedia.org/wiki/Etak>

³ https://en.wikipedia.org/wiki/Global_Positioning_System

机器人实现自主导航都需要哪些？根据导航算法会有不同，但应该至少需要如下几种：

- ① 地图
- ② 测量或估计机器人的姿态的功能
- ③ 识别障碍物，如墙壁和物体的功能
- ④ 能够计算出最优路线并行驶的功能

11.1.2. 地图

第一是地图。导航仪从购买时起就配备有非常准确的地图，并且可以定期下载更新的地图，以便可以根据地图将汽车引导到目的地。但是在使用服务机器人的房间里是否会有地图呢？服务机器人也像导航仪一样，需要一个地图，所以需要人创建一个地图，并把它给到机器人，或者需要机器人自己创建一个地图。

SLAM (Simultaneous localization and mapping)⁴就是为了让机器人自己（或接受人的一些帮助）绘制地图而出现的技术。用中文应该是“同步定位和绘制地图”。这是在机器人移动到未知空间时通过探测周围环境来估计当前位置并同时绘制地图的方法。

11.1.3. 测量或估计机器人姿态的功能

第二，机器人需要自己能够测量和估计姿态（位置+方向）。汽车会用GPS估计自己的位置，但在室内无法使用GPS。即使说可以在室内使用，误差较大的GPS无法用于测量精细的移动。最近，虽然有DGPS⁵等高精度的定位系统，但在室内还是无法使用的。为了克服这种问题，人们引进了标志识别方式及室内定位系统等技术，但在成本或精确度方面还不足以投入实际应用。当前的室内机器人用的最多的是导航推测（dead reckoning）⁶⁷。它的缺点是只能估算相对位置，但因为仅用廉价的传感器就能实现，且已有较长时间的研究进展，因此可以得到一定水平的位置估计值，因此被广泛使用。导航推测技术用机器人的车轮的旋转量来估计机器人本身的移动量。但车轮的旋转量具有不少的误差。

⁴ https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

⁵ https://en.wikipedia.org/wiki/Differential_GPS

⁶ https://en.wikipedia.org/wiki/Dead_reckoning

⁷ <http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/s07/labs/NXTLabs/Lab%203.html>

因此还利用IMU传感器等获取惯性信息来补偿位置和方向值，以此减小误差。



姿态（位置+方向）

在ROS中，机器人的位置（position: x, y, z ）和方向（orientation: x, y, z, w ）被定义为姿态。如第4.5节TF描述中所提到的，该位置由 x 、 y 和 z 三个向量描述，而方向使用四元数形式的 x 、 y 、 z 和 w 。有关消息pose的说明，请参阅以下地址。需要注意，有一些术语隐含地包括方向信息，例如位置估计或特定物体的位置等。

http://docs.ros.org/api/geometry_msgs/html/msg/Pose.html

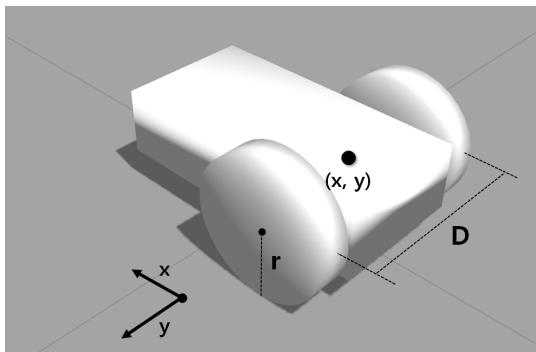


图 11-2 导航推测所需要的信息（中心位置 (x, y) ，车轮间距离 D ，车轮半径 r ）

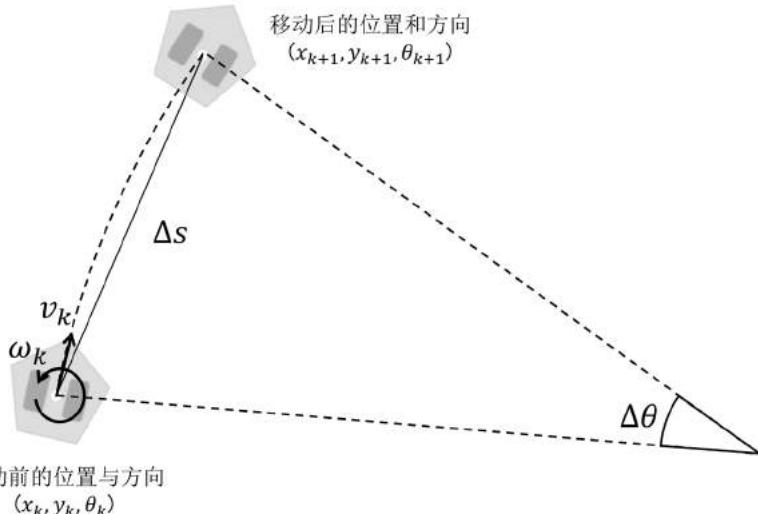


图 11-3 导航推测（dead reckoning）

下面简要介绍一下导航推测。当有如图11-2的移动机器人时，设D是车轮之间的距离，r是车轮的半径。如图11-3所示，当机器人在时间 T_e 内移动很短距离时，利用左右电机旋转量（当前编码器值 $E_{l,c}$ 、 $E_{r,c}$ 和 T_e 之前的编码器值 $E_{l,p}$ 和 $E_{r,p}$ ）来计算出左右车轮的转速 (v_l, v_r) ，如式11-1和11-2所示。

$$v_l = \frac{(E_{l,c} - E_{l,p})}{T_e} \cdot \frac{\pi}{180} \text{ (radian / sec)} \quad (\text{式 11-1})$$

$$v_r = \frac{(E_{r,c} - E_{r,p})}{T_e} \cdot \frac{\pi}{180} \text{ (radian / sec)} \quad (\text{式 11-2})$$

如式11-3和11-4求出左右车轮的移动速度 (V_l, V_r) ，并如式11-5和11-6求出机器人的平移速度（linear velocity: v_k ）和旋转速度（angular velocity: ω_k ）。

$$V_l = v_l \cdot r \text{ (meter/sec)} \quad (\text{式 11-3})$$

$$V_r = v_r \cdot r \text{ (meter/sec)} \quad (\text{式 11-4})$$

$$v_k = \frac{(V_r + V_l)}{2} \text{ (meter/sec)} \quad (\text{式 11-5})$$

$$\omega_k = \frac{(V_r - V_l)}{D} \text{ (radian/sec)} \quad (\text{式 11-6})$$

最后，通过这些值，利用式11-7至11-10的计算来求得机器人的位置 $(x_{(k+1)}, y_{(k+1)})$ 和方向 $(\theta_{(k+1)})$ 。

$$\Delta s = v_k T_e \quad \Delta \theta = \omega_k T_e \quad (\text{式 11-7})$$

$$x_{(k+1)} = x_k + \Delta s \cos \left(\theta_k + \frac{\Delta \theta}{2} \right) \quad (\text{式 11-8})$$

$$y_{(k+1)} = y_k + \Delta s \sin \left(\theta_k + \frac{\Delta \theta}{2} \right) \quad (\text{式 11-9})$$

$$\theta_{(k+1)} = \theta_k + \Delta \theta \quad (\text{式 11-10})$$

11.1.4. 识别障碍物，如墙壁和物体

第三是一种利用传感器检测墙壁和物体等障碍物的方法。此时用到距离传感器、视觉传感器等多种传感器。其中距离传感器有基于雷达的距离传感器（常用的是LDS、LRF和LiDAR）、超声波传感器和红外距离传感器等，而视觉传感器包括立体相机、单镜相机、360度相机，以及经常用作深度摄像头的RealSense、Kinect和Xtion也都用于识别障碍物。

11.1.5. 计算最优路径和行驶功能

第四是导航（Navigation）功能，这是计算到达目的地的最优路径，并且驱动机器人按照最优路径到达目的地的功能。实现这个功能的算法有很多种：称为路径搜索和规划的A*算法⁸、势场算法⁹、粒子过滤算法¹⁰和RRT（Rapidly-exploring Random Tree）算法¹¹等。

在这一节中，我们简要地总结了SLAM和导航的组成要素，但这是难解和广博的内容。四个要素中的第二个要素-测量和估计机器人的位置-已经在前面讲到，第三个要素-识别墙壁、物体等障碍物-在前面的“第8章 机器人、传感器和电机”中已说明。下面了解一下第一个要素-用于绘制地图的SLAM-和第四个要素-导航。

11.2. SLAM实习篇

在描述SLAM的理论之前，我将解释如何使用TurtleBot3来使用SLAM。在本节中，把可以用于绘制地图的bag文件也上传到了github存储库中，因此建议读者跟着做一次。下面先介绍SLAM的应用方法，理论则会在11.4节中详细介绍。

⁸ https://en.wikipedia.org/wiki/A*_search_algorithm

⁹ http://www.cs.cmu.edu/~./motionplanning/lecture/Chap4-Potential-Field_howie.pdf

¹⁰ https://en.wikipedia.org/wiki/Particle_filter

¹¹ https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

11.2.1. 对于使用SLAM的机器人的硬件限制

与SLAM相关的常用的功能包有gmapping¹²、cartographer¹³和rtabmap¹⁴。我们将在本节中使用gmapping。使用Gmapping有几种硬件限制。对于常见的移动机器人不成问题，但还是希望读者了解一下。

移动方式

机器人必须能够用X、Y轴平面上的平移速度（linear velocity）和theta旋转速度（angular velocity）指令进行操作。比如有左右两个可以单独驱动的差动驱动式移动机器人（differential drive mobile robot），或者具有三个以上的全向轮的全向移动机器人（omni-wheel robot）。

测位(Odometry)

需要能获得测位信息。换句话说，要可以通过导航推测方法（dead reckoning）来推算机器人移动的距离和旋转量，或者通过使用IMU传感器的惯性信息推定姿态补偿，或用IMU传感器测量平移速度和旋转速度，最终要可以测量及推断机器人本身的位置。

检测用传感器

为了实现SLAM和导航，机器人需要有LDS（Laser Distance Sensor）、LRF（Laser Range Finder）和LiDAR来测量XY平面上的障碍物。深度相机（如RealSense、Kinect和Xtion）也可以将3D信息转换为XY平面上的信息。换句话说，有必要安装一个能够测量二维平面的传感器。使用超声波传感器、PSD传感器和摄像机的可视SLAM也属于这个概念，但不在本书论述的范围内。

机器人的形态

只考虑正多边形、正方形和圆形机器人。不考虑沿某一方向太长的机器人、无法从房门通过的过大的机器人、双足人形机器人、多关节移动机器人和飞行机器人等。在本章中，我们将使用我们在第10章中讨论过的官方ROS平台TurtleBot3。图11-4中的

¹² <http://wiki.ros.org/gmapping>

¹³ <http://wiki.ros.org/cartographer>

¹⁴ <http://wiki.ros.org/rtabmap>

TurtleBot3满足上面提到的所有四个SLAM约束条件。

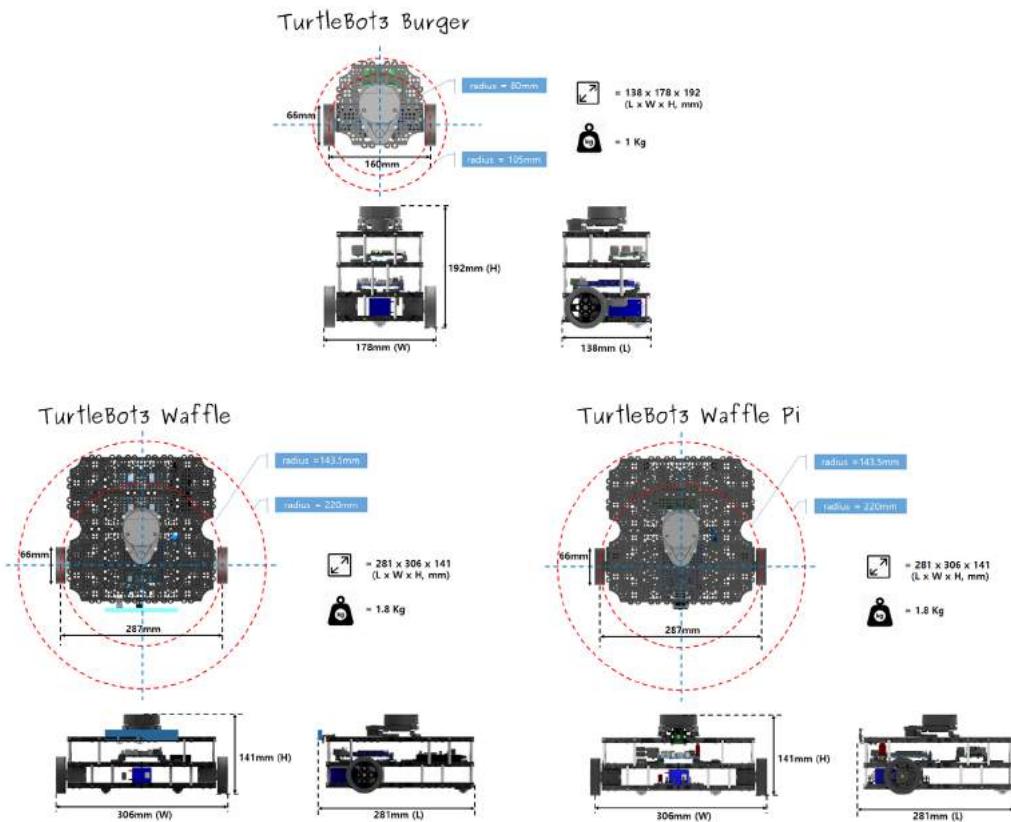


图 11-4 TurtleBot3 Burger、Waffle 和 Waffle Pi 的外形

11.2.2. SLAM的实验环境

如下几种环境从Gmapping算法的角度来看缺少特征要素，因此这些不适合应用SLAM。**①**没有任何障碍物的方形环境。**②**由两个长长而平行的墙壁形成的走廊。**③**无法反射激光或红外线的玻璃窗。**④**散射镜。**⑤**由于传感器的特性，无法获取障碍物信息的环境，如湖泊或海边等。

在本书的练习中，实验环境被设置为一个可以测量长度的网格式的迷宫型平面区域，如图11-5所示。



图 11-5 实验环境

11.2.3. 用于SLAM的ROS功能包

本节中使用的SLAM相关的ROS功能包是turtlebot3元功能包、slam_gmapping元功能包中的gmapping功能包，以及navigation元功能包中的map_server功能包。这一切都曾在“10.5 TurtleBot3开发环境”中安装过。由于这个练习是第10章的后续内容，所以我只描述运行方法。每个功能包的说明将在下一节中详细介绍。为了便于参考，本节将分开说明[Remote PC]和[TurtleBot]两种环境下运行的命令以避免混淆。

11.2.4. 运行SLAM

SLAM运行顺序如下。在这个例子中，我们将使用TurtleBot3 Waffle作为参考。如果您的是Burger，则只需改变名字。如果是在使用Burger或Waffle Pi，只需将命令中的‘TURTLEBOT3_MODEL’项目从‘waffle’改为‘burger’或‘waffle_pi’。

roscore

在[Remote PC]中，运行roscore。

```
$ roscore
```

启动机器人

在[TurtleBot]中，运行turtlebot3_robot.launch文件并运行turtlebot3_core和turtlebot3_lds节点。

```
$ rosrun turtlebot3_bringup turtlebot3_robot.launch
```

运行SLAM功能包

在[Remote PC]中，运行turtlebot3_slam.launch启动文件。turtlebot3_slam功能包只包含一个launch文件。运行后，将运行robot_state_publisher节点和slam_gmapping节点，其中robot_state_publisher节点将两个轮子和每个关节的三维位置和方向信息以TF形式发布，而slam_gmapping节点用于绘制地图。另外，描述包含机器人的外观信息的URDF的robot_model也会被设置。

```
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun turtlebot3_slam turtlebot3_slam.launch
```

运行RViz

运行RViz可视化工具RViz，以便在SLAM过程中可以直观地确认结果。运行时如果附加如下所示的“-d”选项，则从一开始就会添加有关显示(display)的插件，会比较方便。

```
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun rviz rviz -d `rospack find turtlebot3_slam`/rviz/turtlebot3_slam.rviz
```

保存话题信息

下一步，用户将直接遥控机器人并执行SLAM操作，此时发出的/scan和/tf话题存储在名为scan_data的bag文件中。您可以在后期使用此文件创建地图，也可以重现绘制地图过程中的/scan和/tf话题，而无需重复做实验。可以把它想象成来自实验的话题数据（下一个例子中的/scan和/tf话题）的副本。以下命令的-O选项是指定输出文件名称的选项，这将会把输出内容保存为叫做“scan_data.bag”的bag文件。保存话题消息在SLAM过程中不是必要功能，所以如果不需要保存消息，则可以跳过它。

```
$ rosbag record -O scan_data /scan /tf
```

遥控机器人

以下命令允许用户手动遥控机器人并执行SLAM操作。这里重要的是不要过快改变机器人的速度，也不要以过快的速度前进、后退或旋转。移动机器人时，机器人必须扫描要测量的环境的每个角落。这需要经验和技巧，因此需要在大量的SLAM实验中积累经验。

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

绘制地图

现在已经完成了所有的准备工作，让我们运行map_saver节点来创建一个地图吧。当移动机器人时，机器人会根据测位（odometry）、tf信息和传感器的扫描信息来创建地图。这可以在我们刚刚运行的RViz中看到。创建的地图保存在运行map_saver的目录中。除非指定了文件名，否则保存为实际地图文件map.pgm和包含地图信息的map.yaml文件。如下命令中的“-f”选项是指定保存地图文件的目录及文件名的选项。例如，如果指定为“~/map”，则“~”意味着用户目录，而“map”意味着要保存为map.pgm和map.yaml文件。

```
$ rosrun map_server map_saver -f ~/map
```

您可以通过上述过程创建地图。绘制地图所需的节点和话题可以通过使用rqt_graph来查看，如图11-6所示。绘制地图过程如图11-7所示，最终的地图如图11-8所示。我们可以确认，上面提到的实验环境地图已正确绘制。

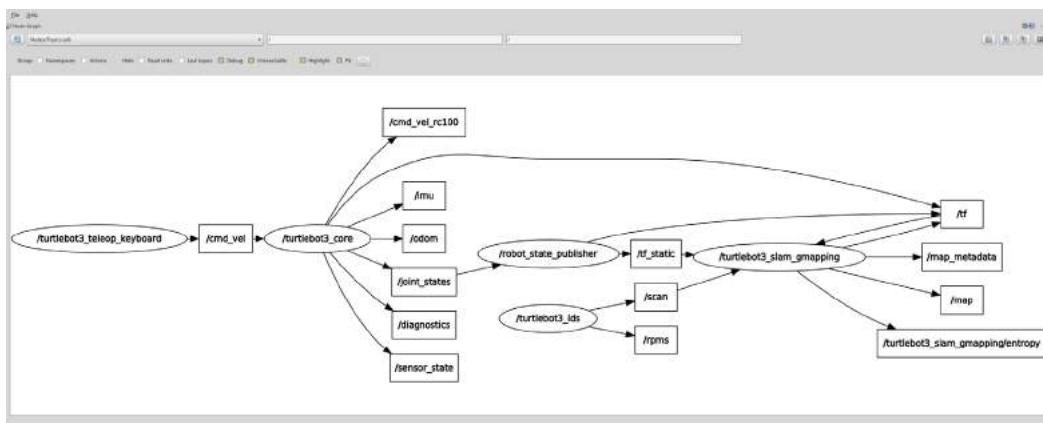


图 11-6 SLAM所需的节点和话题

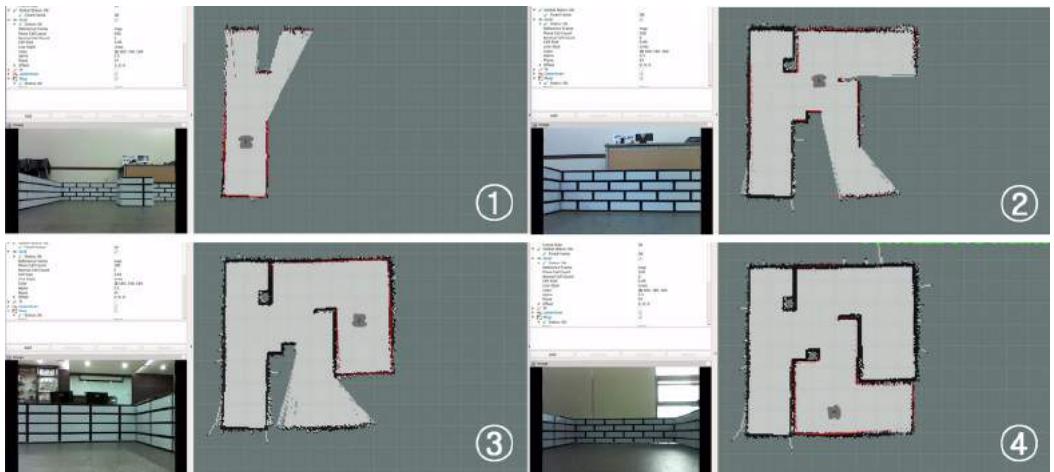


图 11-7 用于绘制地图的SLAM的运行过程

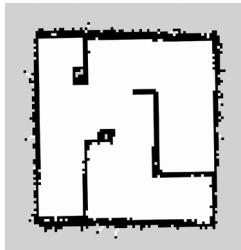


图 11-8 完成的地图

11.2.5. 利用预先准备好的bag文件运行的SLAM

为了可以在没有TurtleBot3和LDS传感器的情况下尝试SLAM，我们将利用录制的bag文件。首先，下载本节要用到的文件。

```
$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/turtlebot3/master/turtlebot3_slam/bag/
TB3_WAFFLE_SLAM.bag
```

以下内容类似于上面的SLAM运行方法。唯一的不同点是对rosbag进行回放(play)，而不是保存。如此操作就和实际实验相同。

```
$ roscore
```

```
$ export TURTLEBOT3_MODEL=waffle  
$ roslaunch turtlebot3_bringup turtlebot3_remote.launch
```

```
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun rviz rviz -d `rospack find turtlebot3_slam`/rviz/turtlebot3_slam.rviz
```

```
$ roscd turtlebot3_slam/bag  
$ rosbag play ./TB3_WAFFLE_SLAM.bag
```

```
$ rosrun map_server map_saver -f ~/map
```

在下一节中将更详细地说明前面运行过的功能包的源代码，并附加说明设置方法。

11.3. SLAM应用篇

如果11.2节是一个按部就班操作的简单的过程，那么本节的应用篇中将详细剖析SLAM中使用的ROS功能包以及创建和配置它的方法。我们将仔细看看turtlebot3元功能包、slam_gmapping元功能包中的gmapping功能包和navigation元功能包中的map_server功能包。所以，本节是将11.2节内容应用于自己的机器人的应用篇。对SLAM的理论将在11.4节中介绍。

本课程是基于TurtleBot3机器人平台和LDS传感器，但理解和掌握之后可以将SLAM应用在自己的机器人，而非受限于机器人平台和传感器。如果您想在TurtleBot3机器人平台上搭建您自己的机器人或创建自己的风格的新的机器人，本节将会带来帮助。

11.3.1. 地图

首先，由于本段课程最终要得到的结果是地图，因此我们有必要更详细了解有关地图的内容。如果我们给机器人一张我们使用的纸质地图，机器人会理解吗？应该不会。应该给机器人一个易于理解和易于计算的数字文件。人们对于这种机器人导航地图的定义已经讨论了很长时间，现在也没有结束。尤其是，近年来出现了各种形式的地图，有些不仅包括二维信息，而且还包括三维信息，或者有些地图不仅包含有关移动的信息，还包含各物体的分割（segmentation）的信息。

在这个讲座中，我们将使用在ROS社区中常用的二维占用网格地图（OGM，Occupancy Grid Map）。如上图11-9所示，白色是机器人可以移动的自由区域（free area），黑色是机器人不能移动的占用区域（occupied area），灰色是未被确认的未知区域（unknown area）。

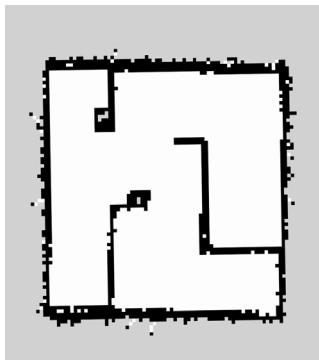


图 11-9 占用网格地图

这些区域用从0到255表达的灰度（gray scale）值表示。该值是通过贝叶斯定理的后验概率获得的，该概率代表占用状态的占用概率。占用概率 occ 表示为“ $\text{occ} = (255 - \text{color_avg}) / 255.0$ ”。如果图像是24位，则是“ $\text{color_avg} = (\text{一个单元的灰度值} / 0xFFFFFFF \times 255)$ ”。这个 occ 越接近1，它被占用的概率越大，越接近0，被占用的概率就越小。

当它以ROS消息（nav_msgs/OccupancyGrid）发布时，会被重新定义为占有度，是[0~100]之间的整数。越接近“0”就越接近移动自由的区域（free area），而越接近“100”就越是不可移动的已占用的区域（occupied area）。此外，“-1”是定义为未知区域（unknown area）。

在ROS中，地图信息以*.pgm文件格式（portable graymap format）存储和使用。它还包含一个*.yaml文件，它也包含地图信息。例如，如果我们查看我们在11.2节中所写地图信息（map.yaml），则结果类似如下所示。`image`是地图的文件名，而`resolution`是地图的分辨率，单位是meters/pixel。

```
image: map.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
```

```
negate: 0  
occupied_thresh: 0.65  
free_thresh: 0.196
```

也就是说，每个像素意味着5厘米。origin是地图的原点，origin的每个数字代表x、y和yaw。地图的左下角是 $x = -10m$, $y = -10m$ 。negate会反转黑白。每个像素的颜色如下确定：由当占用概率超过占用阈值（occupied_thresh）时表示为黑色的占用区域，而当占用概率小于自由阈值（free_thresh）时表示为白色的自由区域（free area）。

图11-10显示了使用TurtleBot3创建大型地图的结果。用了大约一个小时的时间创建了一个行程约350米的地图。

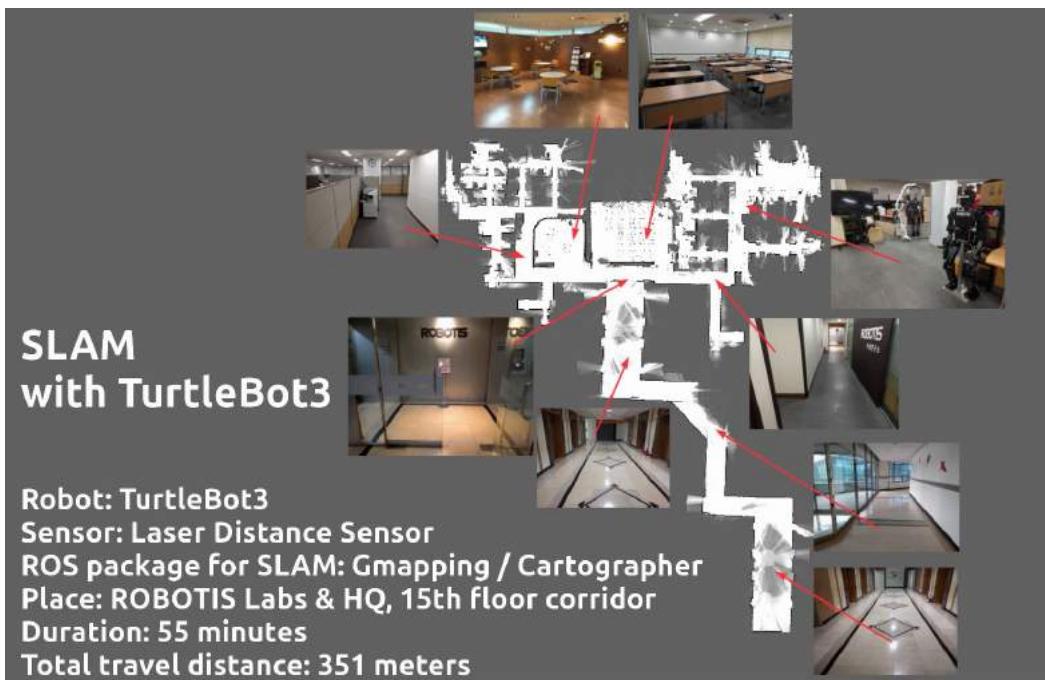


图 11-10 用TurtleBot3制作的广域占用网格地图

11.3.2. SLAM所需的信息

我们已经了解过地图了，那下面让我们来看看为了绘制地图，SLAM都需要哪些信息。首先，“制作地图时需要什么？”首要的是距离值。例如，可以以自己为中心来判

断“那个沙发离我有2m远”的距离值。可以说这个距离值是用LDS或深度摄像机来扫描XY平面的结果值。

其次是我的位置值。在这里，“我”是指“传感器”，因为这个传感器的位置固定在机器人上，所以如果机器人移动，传感器也会一起移动。因此，传感器的位置值依赖于机器人的移动量，也就是测位（odometry）。有必要计算它并将其作为位置值来提供。

这里提到的距离值在ROS中被称为scan，并且姿态（位置+方向）信息会根据相对坐标关系而改变，因此被称为tf（transform）。如图11-11所示，我们根据两个信息scan和tf来运行SLAM，并创建我们想要的地图。

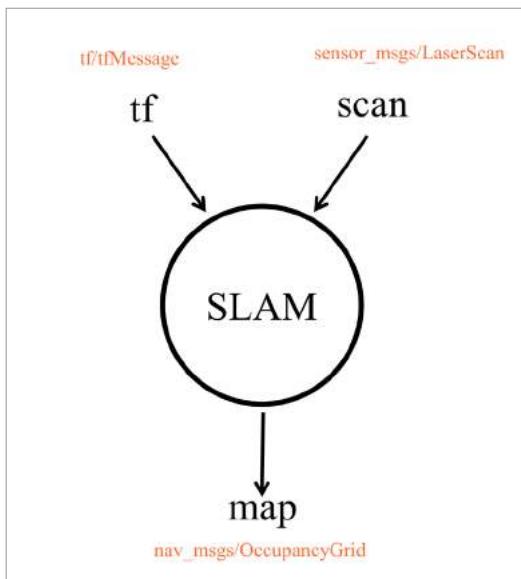


图 11-11 SLAM所需的tf、scan数据及其结果map的关系

11.3.3. SLAM的处理过程

为了创建地图，除了turtlebot3_core节点之外，笔者还为SLAM创建了turtlebot3_slam功能包。该功能包没有源文件，但是通过将需要的功能包捆绑成launch文件来运行。这个过程如图11-12所示，后面有详细的说明。

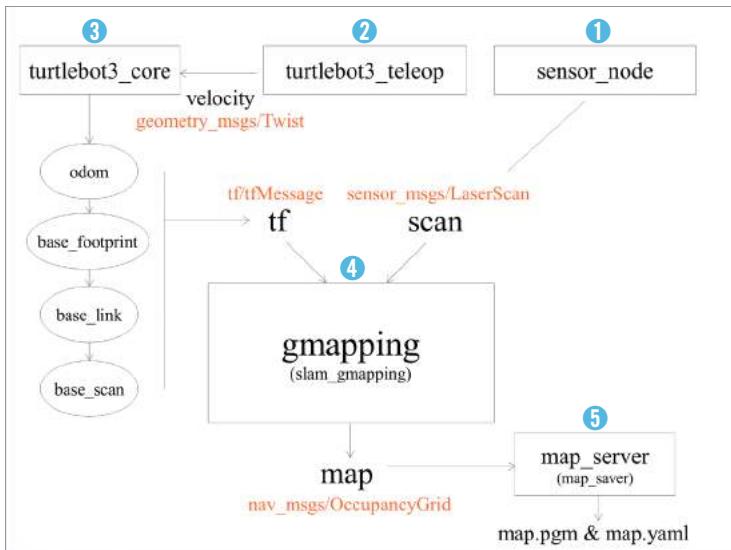


图 11-12 turtlebot3_slam流程图

① sensor_node(例: turtlebot3_lds)

turtlebot3_lds节点运行LDS传感器，并将SLAM所需的scan信息发送到slam_gmapping节点。

② turtlebot3_teleop(例: turtlebot3_teleop_keyboard)

turtlebot3_teleop_keyboard节点是可以接收键盘输入并控制机器人的节点。向turtlebot3_core节点发送移动速度和旋转速度命令。

③ turtlebot3_core

turtlebot3_core节点接收用户的命令并移动机器人。此时在内部发送测得的机器人自己的位置odom信息，且还会以odom→base_footprint→base_link→base_scan的顺序将odom的相对坐标变换信息以tf形式发布。

④ turtlebot3_slam_gmapping

在turtlebot3_slam_gmapping节点中，根据scan信息（由传感器测量的距离值）和tf值（传感器的位置值）来创建地图。

5 map_saver

map_server功能包中的map_saver节点将利用这个地图的信息生成一个可保存的map.pgm文件和一个信息文件map.yaml。

11.3.4. 坐标变换 (TF)

在SLAM中使用的两种信息是如上所述的距离值和测量该距离值的位置。距离值可以从传感器节点接收，并且距离值被测量的位置是相应传感器的位置。传感器安装在机器人的某个地方，因此机器人的移动会带动传感器移动。换句话说，机器人和传感器在物理上是固定的，并且传感器的姿态（位置+方向）根据机器人的移动而相对变化。可以将其视为相对坐标变换。在ROS中，这个过程被称为tf。我们以树的形式看看当前的相对坐标，命令如下：

```
$ rosrun rqt_tf_tree rqt_tf_tree
```

如果执行上述命令，则可以使用tf的tree查看器检查机器人和传感器的相对位置变换信息（tf），如图11-13所示。换句话说，如果仅考虑从机器人位置到安装LDS的位置这一段，则位置信息会按照odom→base_footprint→base_link→base_scan的顺序相对连接。机器人会根据从turtlebot3_teleop_keyboard节点收到的平移速度和转速命令来移动，并且按照前面所述的导航推测（dead reckoning）估计机器人的测位（odometry），如此生成的odom会以tf形式发布。之后的base_footprint→base_link→base_scan是在结构上固定的状态。这与在turtlebot3_description功能包中的/urdf/turtlebot3_waffle.urdf.xacro中描述的一样，描述了各坐标变换，并定期地通过robot_state_publisher节点发布tf。

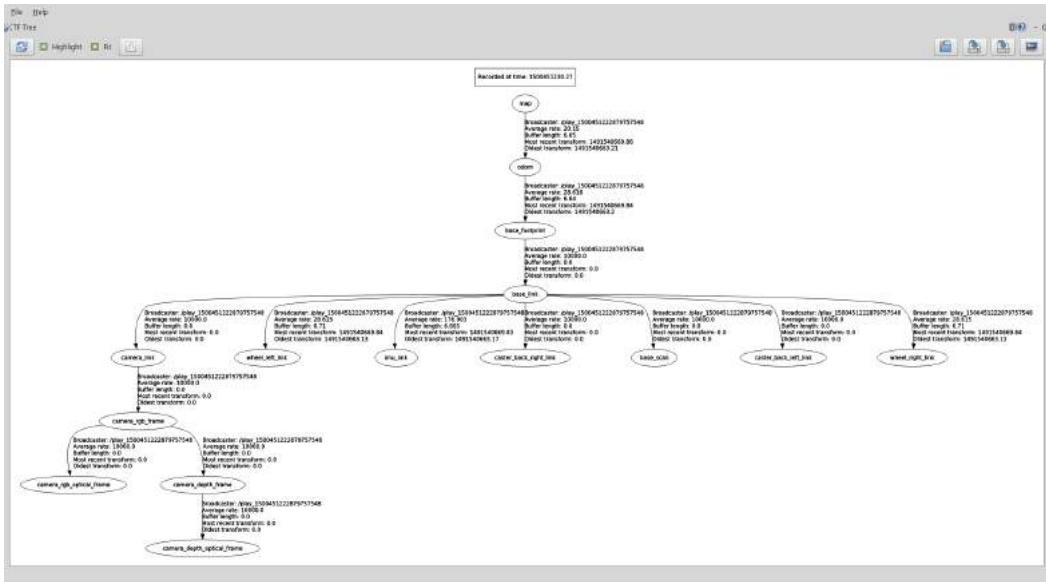


图 11-13 地图和机器人各部分的相对坐标变换状态

11.3.5. turtlebot3_slam功能包

turtlebot3_slam功能包中的turtlebot3_slam.launch的内容如下。这个启动文件主要分为两种，一种包含turtlebot3_remote.launch文件，另一种运行本章讨论的turtlebot3_slam_gmapping节点。

turtlebot3_slam/launch/turtlebot3_slam.launch

```
<launch>
  <!-- Turtlebot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch" />

  <!-- Gmapping -->
  <node pkg="gmapping" type="slam_gmapping" name="turtlebot3_slam_gmapping" output="screen">
    <param name="base_frame" value="base_footprint"/>
    <param name="odom_frame" value="odom"/>
    <param name="map_update_interval" value="2.0"/>
    <param name="maxUrange" value="4.0"/>
    <param name="minimumScore" value="100"/>
    <param name="linearUpdate" value="0.2"/>
    <param name="angularUpdate" value="0.2"/>
  </node>
</launch>
```

```

<param name="temporalUpdate" value="0.5"/>
<param name="delta" value="0.05"/>
<param name="lskip" value="0"/>
<param name="particles" value="120"/>
<param name="sigma" value="0.05"/>
<param name="kernelSize" value="1"/>
<param name="lstep" value="0.05"/>
<param name="astep" value="0.05"/>
<param name="iterations" value="5"/>
<param name="lsigma" value="0.075"/>
<param name="ogain" value="3.0"/>
<param name="srr" value="0.01"/>
<param name="srt" value="0.02"/>
<param name="str" value="0.01"/>
<param name="stt" value="0.02"/>
<param name="resampleThreshold" value="0.5"/>
<param name="xmin" value="-10.0"/>
<param name="ymin" value="-10.0"/>
<param name="xmax" value="10.0"/>
<param name="ymax" value="10.0"/>
<param name="llsamplerange" value="0.01"/>
<param name="llsamplestep" value="0.01"/>
<param name="lasamplerange" value="0.005"/>
<param name="lasamplestep" value="0.005"/>

</node>
</launch>

```

首先，我们来看看turtlebot3_remote.launch文件。该文件描述了用户指定的机器人模型的加载和robot_state_publisher节点的执行，该节点将两个轮子和每个关节的姿态信息以TF形式。

```

turtlebot3_bringup/launch/turtlebot3_remote.launch

<launch>
<arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>

<include file="$(find turtlebot3_bringup)/launch/includes/description.launch.xml">
<arg name="model" value="$(arg model)"/>
</include>

```

```

<node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher"
output="screen">
  <param name="publish_frequency" type="double" value="50.0" />
</node>
</launch>

```

剩下的一个节点turtlebot3_slam_gmapping实际上是将gmapping功能包中的slam_gmapping节点改名后运行。为了使这个节点正常工作，需要根据自己的机器人和传感器修改各种选项，如下所示。下面的设置值都是为TurtleBot3 Waffle设置的。如果想使用TurtleBot3以外的其他机器人，请参考以下说明并根据您的机器人和传感器进行修改。

设置slam_gmapping节点	
<param name="base_frame" value="base_footprint"/>	机器人基本框架
<param name="odom_frame" value="odom"/>	测位 (Odometry) 框架
<param name="map_update_interval" value="2.0"/>	地图更新时间间隔 (sec)
<param name="maxUrange" value="4.0"/>	使用的激光传感器的最大范围 (meter)
<param name="minimumScore" value="100"/>	考虑到扫描匹配结果的最低分数
<param name="linearUpdate" value="0.2"/>	处理所需的最小移动距离
<param name="angularUpdate" value="0.2"/>	处理所需的最小旋转角度
<param name="temporalUpdate" value="0.5"/>	如果从最后一次扫描时刻开始超过了此更新时
间，则执行扫描。如果这个值小于0，则不使用它。	
<param name="delta" value="0.05"/>	地图分辨率：距离/像素
<param name="lskip" value="0"/>	在每次扫描中跳过的光束数量
<param name="particles" value="120"/>	粒子滤波器中的粒子数
<param name="sigma" value="0.05"/>	激光辅助搜索的标准偏差
<param name="kernelSize" value="1"/>	激光辅助搜索的窗口大小
<param name="lstep" value="0.05"/>	初始搜索步骤（平移）
<param name="astep" value="0.05"/>	初始搜索步骤（旋转）
<param name="iterations" value="5"/>	扫描匹配迭代次数
<param name="lsigma" value="0.075"/>	光束似然估计的标准偏差
<param name="ogain" value="3.0"/>	似然估计扁平增益
<param name="srr" value="0.01"/>	测位误差（平移→移动）
<param name="srt" value="0.02"/>	测位误差（平移→旋转）
<param name="str" value="0.01"/>	测位误差（旋转→平移）
<param name="stt" value="0.02"/>	测位误差（旋转→旋转）
<param name="resampleThreshold" value="0.5"/>	重新采样阈值

<param name="xmin" value="-10.0"/>	初始地图大小 (最小x)
<param name=" ymin" value="-10.0"/>	初始地图大小 (最小y)
<param name="xmax" value="10.0"/>	初始地图大小 (最大x)
<param name="ymax" value="10.0"/>	初始地图大小 (最大y)
<param name="llsamplerange" value="0.01"/>	似然估计的范围 (平移)
<param name="llsamplestep" value="0.01"/>	似然估计的步幅 (平移)
<param name="lasamplerange" value="0.005"/>	似然估计的范围 (旋转)
<param name=" lasamplestep " value= " 0.005 " />	似然估计的步幅 (旋转)

上面解释了绘制地图所需的所有内容。下一节讨论SLAM的理论。

11.4. SLAM理论篇

11.4.1. SLAM

SLAM (Simultaneous Localization And Mapping) , 翻译成中文就是“同时定位与地图构建”。换句话说，这意味着机器人在未知的环境中探索，仅通过安装在机器人上的传感器估计机器人本身的位置的同时绘制未知环境的地图。这是导航及自动驾驶的关键技术。

通常用于位置估算的传感器有编码器 (Encoder) 和惯性测量单元 (IMU) 。编码器测量车轮的旋转量，并通过导航推测 (dead reckoning) 推算机器人的大致位置。在这种情况下会发生一定的误差，此时用惯性传感器测得的惯性信息补偿位置信息的误差。根据目的，位置也可以不用编码器，只用惯性传感器来估算。

该位置估计根据通过在创建地图时使用的距离传感器或相机获得的周围环境的信息再次校正位置。这种位置估计方法包括卡尔曼滤波 (Kalman filter) 、马尔可夫定位 (Markov localization) 、利用粒子滤波 (Particle filter) 的蒙特卡罗定位 (Monte Carlo Localization) 等等。

距离传感器广泛用于测绘，如超声波传感器、光探测器、无线电探测器、激光测位仪和红外扫描仪。除了距离传感器之外，还使用相机，诸如将立体相机当作距离传感器，或使用普通相机的视觉SLAM。

而且，有的研究者提出了通过给环境贴上标记（marker）来识别环境的方法。例如，通过将标记安装在天花板上，用相机区分标记。近来，出现了多种深度相机（RealSense、Kinect和Xtion等），利用这些相机可以获得接近距离传感器的距离值，因此有很多相关的研究。

11.4.2. 多种位置估计（localization）方法论

位置估计方法是机器人工程的一个重要研究领域，它目前也在被人们积极地研究。只要能对机器人的位置进行足够正确的估计，则能够容易地解决基于位置的地图绘制的问题，如SLAM。但是目前还有许多问题，比如，传感器捕捉到信息不确定、为了在实际环境中工作需要保证实时性，等等。为了解决这个问题，有各种位置估计方法在被研究当中。在本节中，作为位置估计的代表性例子，讨论了卡尔曼滤波器（Kalman filter）和粒子滤波器（Particle filter）方法论。

卡尔曼滤波器（Kalman filter）

由Rudolf E. Kalman博士开发的卡尔曼滤波器（Kalman filter）因其在美国宇航局的阿波罗计划中的应用而广为人知，该滤波器指，在有噪声的线性系统中，跟踪目标值状态的递归滤波器。它基于贝叶斯（Bayes）概率，它预先假定了一个模型，并使用这个模型从以前的状态预测（Prediction）当前状态。然后，使用这个预测值与由外部测量仪器获得的实际测量值之间的误差来执行一个补偿（update）过程，这个过程利用误差值推定更准确的状态值。它持续地重复迭代，以此提高准确性。这个过程的简化说明如图11-14所示。

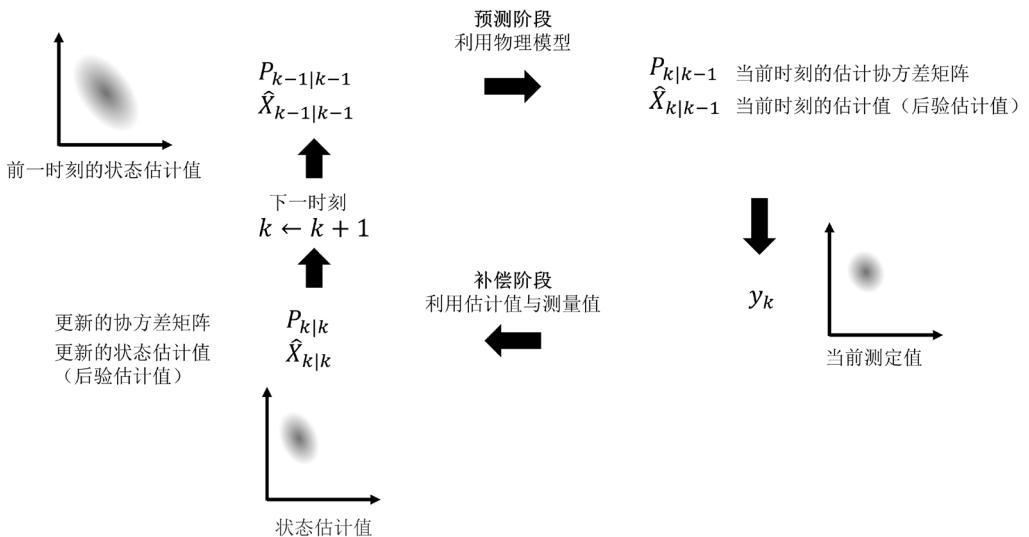


图 11-14 卡尔曼滤波器的基本概念

但是，卡尔曼滤波器仅适用于线性系统。我们的机器人和传感器大部分都是非线性系统，扩展和改进卡尔曼滤波的EKF（扩展卡尔曼滤波）被广泛应用。此外，还有许多KF变体，例如无损卡尔曼滤波器（UKF，Unscented Kalman Filter）和快速卡尔曼滤波器（Fast Kalman filter），这些都提高了EKF的精度。它也经常与其他算法一起使用，例如会与粒子滤波器一起使用的Rao-Blackwellized粒子滤波器（RBPF，Rao-Blackwellized Particle Filter）一起使用。

粒子滤波器

粒子滤波（Particle Filter）是目前最流行的目标跟踪算法。典型的例子是使用粒子滤波器的蒙特卡罗定位（Monte Carlo Localization）。前面描述的卡尔曼滤波器存在一个问题，即在具有高斯噪声的线性系统中能保证准确度，但是对于其他的系统无法保证准确度。但现实世界中的大部分问题都是非线性系统。

机器人和传感器也是如此，所以粒子滤波器通常用于位置估计。如果卡尔曼滤波器是一个分析方法，假设目标是一个线性系统，那么卡尔曼滤波器通过线性运动搜索参数，而粒子滤波器是一种基于尝试和错误（try-and-error）的方法通过仿真进行预测的技术。称为粒子滤波器是因为将由目标系统中的概率分布随机生成的估计值看作为粒子。这也被称为顺序蒙特卡罗（SMC，Sequential Monte Carlo）方法或蒙特卡罗方法。

像其他位置估计算法一样，用粒子滤波方法推定目标物体的位置时，也假设输入信息中包含误差。当使用SLAM时，使用机器人的测位（odometry）值和用距离传感器获得的环境测量值作为观测值来估计机器人的当前位置。

在粒子滤波方法中，位置不确定性是用称为样本的粒子群来描述的。我们根据机器人的运动模型和概率将粒子移动到一个新的估计位置和方向，并根据实际测量值对每个粒子的权重进行调整，逐渐降低噪声，得到比较精确的位置。在移动机器人的情况下，每个粒子（particle）是姿态（pose）和权重（weight）的函数。其中姿态（pose）是机器人的位置（ x, y ）和方向（ i ）的函数。

这个粒子滤波器经历以下5个步骤。除了第一步的初始化之外，重复执行第二至第五步，以估计机器人的位置值。换句话说，是一种用测量值反复更新粒子分布（在X, Y坐标平面上把机器人的位置用概率表达的粒子的分布），以此推测机器人的位置的方法。

① 初始化 (initialization)

由于一开始根本无法知道初始机器人的姿态（位置和方向），因此在可以用N个粒子求得的所有可能的姿态的范围内随机安排机器人的姿态。每个初始粒子权重为 $1 / N$ ，总和为1。N以经验确定，通常为数百。如果初始位置已知，则将粒子放置在其附近。

② 预测 (prediction)

根据描述机器人运动的系统模型（system model），给被观察到的移动量（如机器人的测位（Odometry）信息等）加上噪声（noise），用这种方法移动各个粒子。

③ 调整 (update)

基于所测量到的传感器信息，计算每个粒子的概率，并且通过反映这个值来更新每个粒子的权重值。

④ 姿态估计 (pose estimation)

利用所有粒子的位置、方向和权重值来计算加权平均值、中央值和最大权重的粒子值，并用这些估计机器人的姿态。

⑤ 重采样 (Resampling)

这是生成新粒子的步骤，是去除权重小的粒子，以权重大的粒子为中心创建继承了现有例子的特性（姿态信息）的新粒子。这里，必须保持粒子数量N不变。

另外，如果样本数量足够，粒子滤波器的位置估计比卡尔曼滤波器改进版的EKF或UKF更准确，但如果数目不够，则可能不准确。为了解决这个问题，同时使用粒子滤波和卡尔曼滤波的基于Rao-Blackwellized粒子滤波器（RBPF）的SLAM算法被广泛使用。



粒子滤波器（Particle Filter）

如果您想了解更多关于粒子滤波器的知识，您可以在“Probabilistic Robotics”一书中找到更多关于粒子滤波器的知识。Sebastian Thrun（斯坦福教授，Juda City创始人，谷歌研究员）的这本书被称为机器人工程领域的概率教科书。我给任何想学习机器人的人都强烈推荐这本书。此外，在Open Robotics领域中进行大量活动的KITECH的Yang Guang-Woong先生的博客和个人主页、Hwang Byung-Hoon先生在机器人工程相关的博客上写的有关粒子的文章、Choi Sung-Jun先生的enginius博客，这些都会带来帮助。最后，还可以参考Juda City的在线视频讲座“Artificial Intelligence for Robotics”。

<http://www.probabilistic-robotics.org/>

<https://www.udacity.com/course/cs373>

<http://blog.daum.net/pg365/>

<http://abipictures.tistory.com/>

<http://enginius.tistory.com/>

至此结束了对SLAM的讲解。对于gmapping的描述已被粒子滤波器的说明所取代，因此，更详细的解释请参阅以下参考文献中提到的文章。下一节将介绍导航（navigation）。



OpenSLAM와 Gmapping

如上所述，SLAM领域已经在机器人工程中进行着广泛的研究。这些信息可以在最新的学术期刊和学会的讲座中找到，其中许多研究都是开源的。这些信息由OpenSLAM小组编辑，可以在OpenSLAM.org找到。这是我们必须访问的网站。

我们在第11.4节中使用的gmapping也在这里有介绍，而ROS社区在SLAM中使用了很多gmapping。有两篇关于gmapping的论文。其中一篇发表在ICRA 2005上，另一篇发表在2007年的Robotics, IEEE Transactions on论文期刊上。

这些论文描述了如何尽量减少粒子的数量，以减少计算量且提高实时性。主要的方法是使用上述Rao-Blackwellized粒子滤波器。有关详细信息，请参阅文章，粗略描述可以理解为11.4.2节中的粒子滤波器的解释。

[1] Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard, Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling, Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp. 2432-2437, 2005.

[2] Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard, Improved techniques for grid mapping with rao-blackwellized particle filters, IEEE Transactions on Robotics, Vol.23, No.1, pp.34-46, 2007

11.5. 导航实战篇

在说明导航之前，先解释如何使用TurtleBot3进行导航。与SLAM相同，首先介绍导航应用篇，而对于导航的理论将在11.7节中介绍。

导航所需的机器人硬件与11.2节中提到的相同。移动机器人使用TurtleBot3，传感器使用LDS。测量环境也与SLAM相同。在本节中将了解的导航是指：利用前面在SLAM的章节中创建的地图，让机器人移动到指定的目的地。

11.5.1. 用于导航的ROS功能包

本节中使用的与导航相关的ROS功能包包括：turtlebot3元功能包；前一个SLAM课程中编写的turtlebot3元功能包；navigation元功能包中的move_base、amcl和map_server功能包。安装已经在前面的SLAM课程中做好了。由于这个练习是后续讲座，所以我只描述执行方法。下一节将介绍每个功能包。

11.5.2. 运行导航

运行导航的顺序如下。在这个例子中，我们将以TurtleBot3 Waffle为准进行说明。如果是用Burger模型，只需改变它的名字。如果是在使用Burger或Waffle Pi，只需将命令中的‘TURTLEBOT3_MODEL’项目从‘waffle’改为‘burger’或‘waffle_pi’。

roscore

在[Remote PC]中，运行roscore。

```
$ roscore
```

启动机器人

在[TurtleBot]中，运行turtlebot3_robot.launch文件来运行turtlebot3_core和turtlebot3_lds节点。

```
$ rosrun turtlebot3_bringup turtlebot3_robot.launch
```

运行导航功能包

在[Remote PC]中，运行turtlebot3_navigation.launch启动文件。turtlebot3_navigation功能包由几个启动文件组成。运行后，robot_state_publisher节点（将TurtleBot3的3D模型信息、双轮及各关节的三维位置和方向信息以TF形式发布）、用于加载先前创建的地图的map_server节点、AMCL（自适应蒙特卡罗定位，Adaptive Monte Carlo Localization）节点和move_base节点等4个节点会一起被运行。

```
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

运行RViz

先运行ROS的可视化工具RViz，以便在导航中可以直观地确认目标指定命令和结果。当使用以下选项运行RViz时，会从一开始添加显示插件，因此非常方便。

```
$ rosrun rviz rviz -d `rospack find turtlebot3_navigation`/rviz/turtlebot3_nav.rviz
```

运行后，可以看到如图11-15所示的画面。在右边的地图上，可以看到很多绿色的箭头，这是SLAM理论中描述的粒子滤波器的粒子。这是因为导航也使用粒子滤波器，我将在稍后再解释。可以确认机器人在绿色箭头的中间。

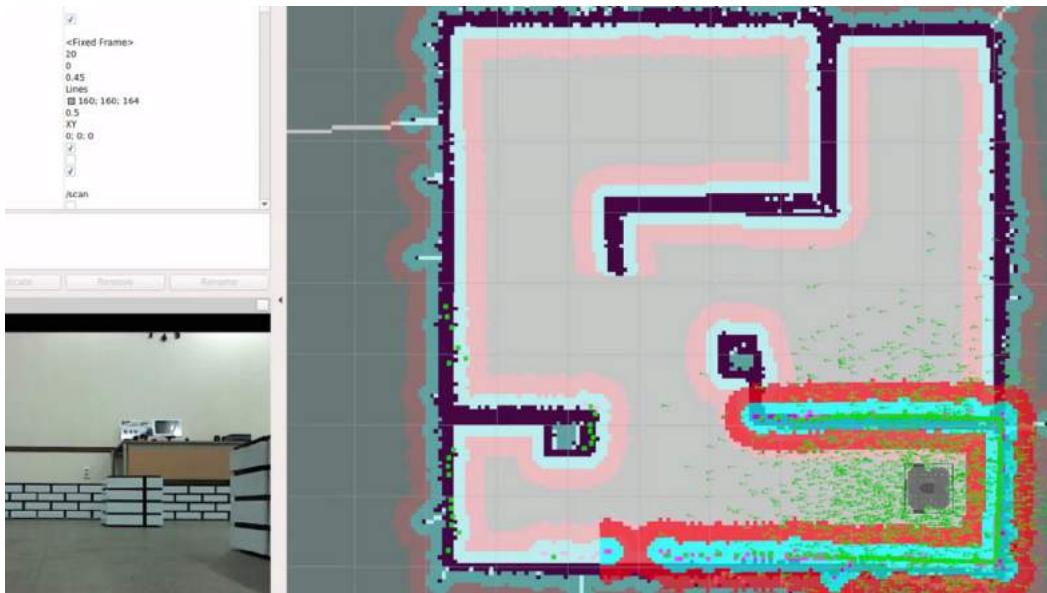


图 11-15 RViz中可以查看的粒子（机器人周围的绿色箭头）

初始位置估计

首先，要做的是估计机器人的初始位置。当在RViz的菜单中按下[2D Pose Estimate]时，会出现一个非常大的绿色箭头。将其移动到机器人在给定的地图中所在的位置，并按住鼠标键的同时，拖动绿色箭头使其指向机器人的前方。这是一种在初期为了估计机器人位置的命令。然后用turtlebot3_teleop_keyboard节点等来回移动机器人，搜集周围的环境信息，找出机器人当前位于地图上的位置。经过了这个过程后，机器人将绿色箭头指定的位置和方向作为初始位置，推定自己的位置和方向。

设置目的地且移动机器人

一切准备就绪后，下面下达移动命令。如果在RViz的菜单中按[2D Nav Goal]，会出现一个非常大的绿色箭头。该绿色箭头是指定机器人的目的地的标记，箭头的起点是机器人的x、y位置，箭头方向是机器人的i方向。将此箭头移动到机器人的目的地，然后拖动，以设置方向。机器人将根据创建的地图躲避障碍物，移动到目的地（见图11-16）。

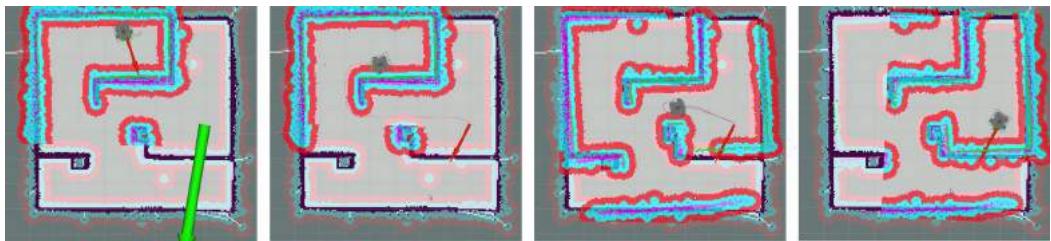


图 11-16 目的地设置（大箭头）和机器人的移动

以下部分将介绍上面运行过的功能包源代码的详细说明和设置方法。它将与SLAM的课程类似地，按实习篇和应用篇和理论篇来分步进行。

11.6. 导航应用程序

如果第11.5节只是按部就班的操作课，那么本节将探讨导航中使用的ROS功能包以及如何创建和配置它。我们将讲解turtlebot3元功能包、作为LDS驱动程序的turtlebot3_lds节点、turtlebot3的三维模型信息（turtlebot3_description）、加载先前创建的地图的map_server节点、ACML（自适应蒙特卡罗定位）节点和move_base节点。这是一个可以将导航实习篇应用于自己的机器人的应用篇。

在本节中，我们将以TurtleBot3机器人平台和LDS传感器为例进行说明，但是如果能活学活用，则可以用自己的机器人实现导航，而不限于特定的机器人平台和特定的传感器。如果读者想创建自己的机器人平台或想在TurtleBot3机器人平台上按自己的风格构建一个新的机器人，本教程将对读者有所帮助。

11.6.1. 导航

导航是在给定的环境中将机器人从当前位置移动到指定的目的地。为此，需要有包含给定环境的家具、物体和墙壁的几何信息（geometry, geo-：土地, metry：测量）的地图，正如前面的SLAM课程所述，机器人可以从自己的姿态信息和从传感器获得的距离信息来获得地图。

在导航中，机器人利用这个地图和机器人的编码器、惯性传感器和距离传感器等资源，从当前位置移动到地图上的指定目的地。这个过程如下。

传感 (sensing)

在地图上，机器人利用编码器和惯性传感器（IMU传感器）更新其测位（odometry）信息，并测量从距离传感器的位置到障碍物（墙壁、物体、家具等）的距离。

姿态估计 (localization / pose estimation)

基于来自编码器的车轮旋转量、来自惯性传感器的惯性信息以及从距离传感器到障碍物的距离信息，估计机器人在已经绘制的地图上的姿态（localization / pose estimation）。此时用到的姿态估计方法有很多种，本节中将使用粒子滤波器定位（particle filter localization），以及蒙特卡罗定位（Monte Carlo Localization）的变体ACML（Adaptive Monte Carlo Localization，自适应蒙特卡罗定位）。

运动规划

也称为路径规划，它创建一个从当前位置到地图上指定的目标点的轨迹。对整个地图进行全局路径规划，以及以机器人为中心对部分地区进行局部路径规划。我们计划使用基于一种避障算法Dynamic Window Approach（DWA）的ROS move_base和nav_core等路径规划功能包。

移动/躲避障碍物 (move / collision avoidance)

如果按照在运动规划中创建的移动轨迹向机器人发出速度命令，则机器人会根据移动轨迹移动到目的地。由于感应、位置估计和运动规划在移动时仍在被执行，因此使用动态窗口方法（DWA）算法可避免突然出现的障碍物和移动物体。

11.6.2. 导航所需的信息

图11-17显示了运行ROS导航功能包所需的必要节点和话题之间的关系。我们将重点介绍导航所需的信息（话题）。作为参考，在描述图11-17中的话题时，分别显示了话题名称和话题消息类型。例如，在测位（odometry）中，“/odom”是话题名称，“nav_msgs/Odometry”是话题消息的类型。

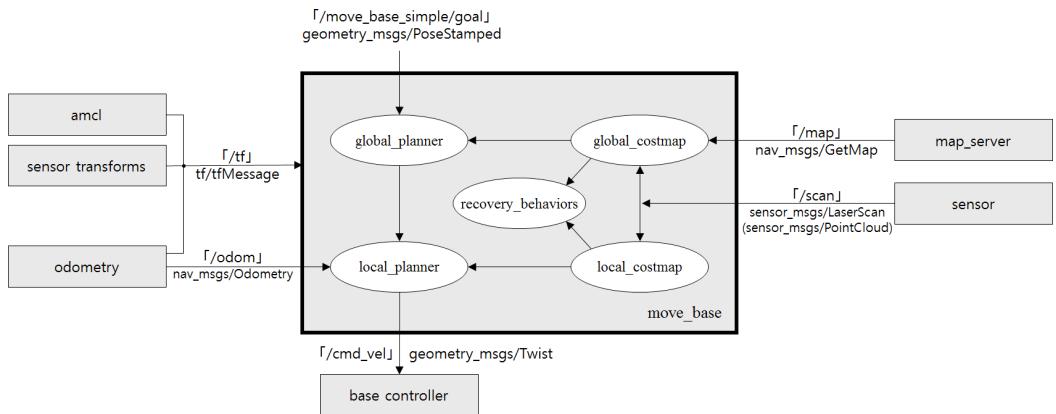


图 11-17 有关导航堆栈配置的各必要节点和话题的关系图

测位（‘/odom’，`nav_msgs/Odometry`）

机器人的测位信息用于局部路径规划，利用接收到的机器人的当前速度等信息，产生局部移动路径或避开障碍物。

坐标变换（‘/tf’，`tf/tfMessage`）

由于机器人传感器的位置根据机器人的硬件配置而变化，所以ROS使用tf相对坐标变换。这只是简单地利用测位获得机器人的位置，例如测位描述“以机器人的位置为原点，传感器在x、y、z坐标坐标系中位于某某位置”。例如，经过`odom`→`base_footprint`→`base_link`→`base_scan`的变换后以话题来发布。它从`move_base`节点接收这些信息，并根据机器人的位置和传感器的位置执行移动路径规划。

距离传感器（‘/scan’，`sensor_msgs/LaserScan` or `sensor_msgs/PointCloud`）

意味着从传感器测量得到的距离值。通常使用LDS和RealSense、Kinect、Xtion等。该距离传感器使用AMCL（adaptive Monte Carlo localization，自适应蒙特卡罗定位）来估计机器人的当前位置，且规划机器人的运动。

地图（‘/map’，`nav_msgs/GetMap`）

导航使用占用网格地图（occupancy grid map）。在本教程中，我们将使用`map_server`功能包来发布我们之前编写的“`map.pgm`”和“`map.yaml`”文件。

目标坐标（‘/move_base_simple/goal’，geometry_msgs/PoseStamped）

目标坐标由用户直接指定。可以使用如平板电脑的设备创建和使用单独的目标坐标命令功能包，但在本教程中，将在ROS的可视化工具RViz中设置目标坐标。目标坐标由二维坐标（x, y）和姿态θ组成。

速度命令（‘/cmd_vel’，geometry_msgs/Twist）

根据最终规划的移动轨迹发布移动机器人的速度命令，而机器人根据该命令移动到目的地。

11.6.3. turtlebot3_navigation的各节点和话题状态

如第11.5节所述，如下所示，在[turtlebot]中执行turtlebot3_robot.launch文件和turtlebot3_navigation.launch文件，就具有了导航的必要条件。

```
$ rosrun turtlebot3_bringup turtlebot3_robot.launch  
  
$ export TURTLEBOT3_MODEL=waffle  
$ rosrun turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

在这种状态下，运行rqt_graph，则可以在ROS环境中查看正在运行的节点和话题信息，如图11-18所示。如图所示，上述导航所需的信息分别以/odom、/tf、/scan、/map和/cmd_vel的话题名称发布和订阅，而move_base_simple/goal是在ROS的可视化工具RViz中直接指定目标坐标时才会被发布。

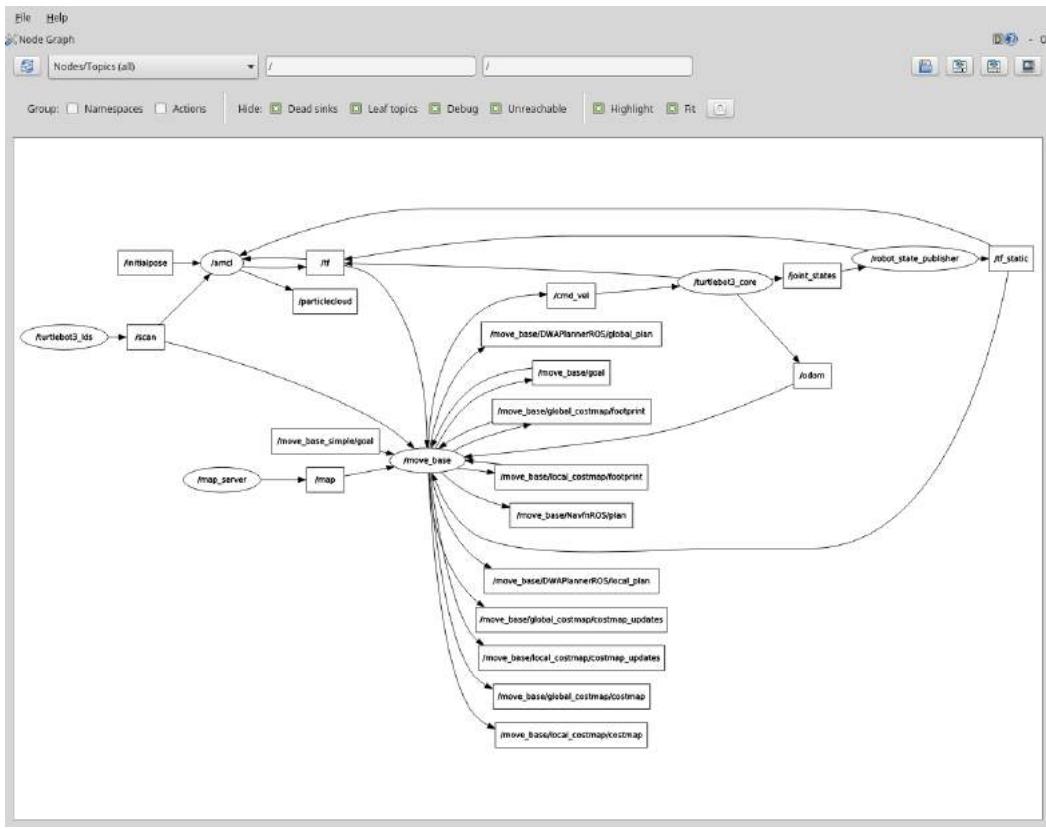


图 11-18 turtlebot3_navigation的各节点和话题的状态

11.6.4. turtlebot3_navigation设置

turtlebot3_navigation功能包需要多种文件：启动与导航节点有关的功能包的launch文件、xml文件、设置各种参数的yaml文件、地图相关文件和rviz配置文件。下面介绍这些配置文件。

- `/launch/turtlebot3_navigation.launch`

`turtlebot3_navigation.launch`一个文件可以启动与导航相关的所有功能包。

- `/launch/amcl.launch.xml`

`amcl.launch.xml`文件是一个包含自适应蒙特卡罗定位（AMCL）的各种参数设置的文件。它与`turtlebot3_navigation.launch`一起使用。

- /param/move_base_params.yaml

统筹管理运动规划的move_base的参数设置文件。

- /param/costmap_common_params_burger.yaml

- /param/costmap_common_params_waffle.yaml

- /param/global_costmap_params.yaml

- /param/local_costmap_params.yaml

导航使用11.3.1中描述的占用网格地图（occupancy grid map）。基于该占用网格地图，利用机器人姿态和从传感器获得的周围信息，将每个像素计算为障碍物、不可移动区域和可移动区域。这时用到的概念是costmap。costmap的配置参数就是这些文件，其中包括公用的costmap_common_params.yaml文件、全局区域运动规划所需的global_costmap_params.yaml文件以及本地区域所需的local_costmap_params.yaml文件。其中，costmap_common_params.yaml根据机器人的型号具有burger和waffle等两种后缀，并且各模型的内容包含不同的外观信息。需要留意的是，TurtleBot3 Waffle Pi除了相机以外其他部分与TurtleBot3 Waffle相同，因此使用waffle后缀，沿用waffle型号的设置。

- /param/dwa_local_planner_params.yaml

dwa_local_planner是最后将移动速度命令传给机器人的功能包，上面文件则是设置该功能包的参数的文件。

- base_local_planner_params.yaml

因为turtlebot3已将dwa_local_planner用作base_local_planner的设置值，因此在此不使用。这是因为已经在move_base节点中更改了参数设置，如下所示：

```
<param name=" base_local_planner" value=" dwa_local_planner/DWAPlannerROS" />
```

- /maps/map.pgm

- /maps/map.yaml

将以前创建的占用网格地图（occupancy grid map）保存在/maps目录中。

■ /rviz/turtlebot3_nav.rviz

它是一个包含ROS的可视化工具RViz的设置信息的文件。将加载RViz的显示插件中的Grid、RobotModel、TF、LaserScan、Map、Global Map、Local Map和AMCL Particles。

以下文件是turtlebot3_navigation.launch文件的详细内容，其中包括机器人模型、robot_state_publisher、map server、AMCL和move_base的执行和配置的内容。

```
/launch/turtlebot3_navigation.launch

<launch>

<arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger,waffle,waffle_pi]"/>

<!-- Turtlebot3 -->
<include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch" />

<!-- Map server -->
<arg name="map_file" default="$(find turtlebot3_navigation)/maps/map.yaml"/>
<node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)">
</node>

<!-- AMCL -->
<include file="$(find turtlebot3_navigation)/launch/amcl.launch.xml" />

<!-- move_base -->
<arg name="cmd_vel_topic" default="/cmd_vel" />
<arg name="odom_topic" default="odom" />
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
<param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />

<rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml"
command="load" ns="global_costmap" />
<rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml"
command="load" ns="local_costmap" />
<rosparam file="$(find turtlebot3_navigation)/param/local_costmap_params.yaml" command="load" />
<rosparam file="$(find turtlebot3_navigation)/param/global_costmap_params.yaml" command="load" />
<rosparam file="$(find turtlebot3_navigation)/param/move_base_params.yaml" command="load" />
<rosparam file="$(find turtlebot3_navigation)/param/dwa_local_planner_params.yaml" command="load" />
```

```
<remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
<remap from="odom" to="$(arg odom_topic)"/>
</node>
</launch>
```

机器人模型和TF

该部分从turtlebot3_description功能包中加载TurtleBot3的机器人3D模型，并通过robot_state_publisher将关节信息等机器人状态发布到相对坐标变换tf。更确切地说，测位(odomentry)的tf(例如odom)是从turtlebot3_core发布的，而其他坐标是以导入的机器人模型中描述的坐标变换值为基准进行相对坐标变换($odom \rightarrow base_footprint \rightarrow base_link \rightarrow base_scan$)后以tf形式发布的。由于这个过程，在RViz中可以看到机器人的三维模型，通过tf可以得到从传感器获得的格栅值的测量位置。

```
<!-- Turtlebot3 -->
<include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch" />
```

turtlebot3_bringup/launch/turtlebot3_remote.launch

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>

  <include file="$(find turtlebot3_bringup)/launch/includes/description.launch.xml">
    <arg name="model" value="$(arg model)" />
  </include>

  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher"
        output="screen">
    <param name="publish_frequency" type="double" value="50.0" />
  </node>
</launch>
```

地图服务器 (map server)

保存在turtlebot3_navigation/maps/目录中的地图信息(map.yaml)和地图(map.pgm)被加载后通过map_server节点以话题的形式被发布。

```
<!-- Map server -->
<arg name="map_file" default="$(find turtlebot3_navigation)/maps/map.yaml"/>
<node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)">
</node>
```

AMCL (Adaptive Monte Carlo Localization)

运行AMCL的amcl节点并设置相关参数。这将在11.6.5中有更详细地介绍。

```
<include file="$(find turtlebot3_navigation)/launch/amcl.launch.xml" />
```

move_base

设置几种变量：运动规划所需的costmap相关参数、将移动速度命令传递给机器人的dwa_local_planner相关参数以及统筹运动规划的move_base的参数。更详细的讨论将在11.6.6中给出。

```
<arg name="cmd_vel_topic" default="/cmd_vel" />
<arg name="odom_topic" default="odom" />
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
  <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />

  <rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml"
    command="load" ns="global_costmap" />
  <rosparam file="$(find turtlebot3_navigation)/param/costmap_common_params_$(arg model).yaml"
    command="load" ns="local_costmap" />
  <rosparam file="$(find turtlebot3_navigation)/param/local_costmap_params.yaml" command="load" />
  <rosparam file="$(find turtlebot3_navigation)/param/global_costmap_params.yaml" command="load" />
  <rosparam file="$(find turtlebot3_navigation)/param/move_base_params.yaml" command="load" />
  <rosparam file="$(find turtlebot3_navigation)/param/dwa_local_planner_params.yaml" command="load" />

  <remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
  <remap from="odom" to="$(arg odom_topic)"/>
</node>
```

11.6.5. 设置turtlebot3_navigation的详细参数

我们来设置前面讨论过的与turtlebot3navigation相关的详细参数。

AMCL (Adaptive Monte Carlo Localization)

amcl.launch.xml文件是一个包含AMCL参数设置的文件，它与上述的turtlebot3_navigation.launch一起使用。对AMCL的描述将在导航理论篇中进行讨论。

turtlebot3_navigation/launch/amcl.launch.xml

```
<launch>
  <!-- 如果值为真， AMCL将接收地图话题， 而不是服务调用。 -->
  <arg name="use_map_topic" default="false"/>
  <!-- 这是关于距离传感器检测值的话题名称。 -->
  <arg name="scan_topic"  default="scan"/>
  <!-- 它在初始位置估计中被用作为高斯分布的初始x坐标值。 -->
  <arg name="initial_pose_x" default="0.0"/>
  <!-- 它在初始位置估计中被用作为高斯分布的初始y坐标值。 -->
  <arg name="initial_pose_y" default="0.0"/>
  <!-- 它在初始位置估计中被用作为高斯分布的初始yaw坐标值。 -->
  <arg name="initial_pose_a" default="0.0"/>

  <!-- 参考下面的参数设置， 运行amcl节点。 -->
  <node pkg="amcl" type="amcl" name="amcl">

    <!-- 过滤器相关参数 -->
    <!-- 允许的最小粒子数量 -->
    <param name="min_particles" value="500"/>
    <!-- 允许的最大粒子数量（越大越好， 需根据PC的性能进行设置） -->
    <param name="max_particles" value="3000"/>
    <!-- 实际分布与估计分布之间的最大误差 -->
    <param name="kld_err" value="0.02"/>
    <!-- 执行滤波器更新之前所需的平移运动（以米为单位） -->
    <param name="update_min_d" value="0.2"/>
    <!-- 执行滤波器更新之前所需的旋转运动（以弧度表示） -->
    <param name="update_min_a" value="0.2"/>
    <!-- 重采样间隔 -->
    <param name="resample_interval" value="1"/>
    <!-- 允许的转换时间（以秒为单位） -->
```

```

<param name="transform_tolerance" value="0.5"/>
<!-- 指数衰减率 (slow average weight filter) , 当值为0.0则禁用 -->
<param name="recovery_alpha_slow" value="0.0"/>
<!-- 指数衰减率 (fast average weight filter) , 当值为0.0则禁用 -->
<param name="recovery_alpha_fast" value="0.0"/>
<!-- 请参考上述initial_pose_x的说明 -->
<param name="initial_pose_x" value="$(arg initial_pose_x)"/>
<!-- 请参考上述initial_pose_y的说明 -->
<param name="initial_pose_y" value="$(arg initial_pose_y)"/>
<!-- 请参考上述initial_pose_a的说明 -->
<param name="initial_pose_a" value="$(arg initial_pose_a)"/>
<!-- 可视化扫描和路径信息的最大周期 -->
<!-- 例如：10Hz = 0.1秒， -1.0则被禁用 -->
<param name="gui_publish_rate" value="50.0"/>
<!-- 同上述use_map_topic的说明 -->
<param name="use_map_topic" value="$(arg use_map_topic)"/>

<!-- 距离传感器参数 -->
<!-- 更改传感器的话题名称 -->
<remap from="scan" to="$(arg scan_topic)"/>
<!-- 要使用的激光感应最大距离 (米) -->
<param name="laser_max_range" value="3.5"/>
<!-- 滤波器更新时被用到的激光束数量上限。-->
<param name="laser_max_beams" value="180"/>
<!-- 传感器的z_hit的混合加权值 (mixture weight) -->
<param name="laser_z_hit" value="0.5"/>
<!-- 传感器的z_short的混合加权值 (mixture weight) -->
<param name="laser_z_short" value="0.05"/>
<!-- 传感器的z_max的混合加权值 (mixture weight) -->
<param name="laser_z_max" value="0.05"/>
<!-- 传感器的z_rand的混合加权值 (mixture weight) -->
<param name="laser_z_rand" value="0.5"/>
<!-- 使用传感器的z_hit的高斯模型的标准偏差 -->
<param name="laser_sigma_hit" value="0.2"/>
<!-- 传感器的z_short的指数衰减参数 -->
<param name="laser_lambda_short" value="0.1"/>
<!-- 与障碍物的最大距离, 用于likelihood_field方式的传感器 -->
<param name="laser_likelihood_max_dist" value="2.0"/>
<!-- 传感器类型 (选择 likelihood_field或beam) -->

```

```

<param name="laser_model_type" value="likelihood_field"/>

<!-- 与odometry有关的参数 -->
<!-- 机器人驱动方法可以选择"diff"或"omni"。 -->
<param name="odom_model_type" value="diff"/>
<!-- 在旋转运动中预计的测位的旋转动量噪声的估计值 -->
<param name="odom_alpha1" value="0.1"/>
<!-- 在平移运动中预计的测位的旋转动量噪声的估计值 -->
<param name="odom_alpha2" value="0.1"/>
<!-- 在平移运动中预计的测位的平移动量噪声的估计值 -->
<param name="odom_alpha3" value="0.1"/>
<!-- 在旋转运动中预计的测位的平移动量噪声的估计值 -->
<param name="odom_alpha4" value="0.1"/>
<!-- odometry框架 -->
<param name="odom_frame_id" value="odom"/>
<!-- 机器人底座 (robot base) 框架 -->
<param name="base_frame_id" value="base_footprint"/>
</node>
</launch>

```

move_base

统筹运动规划的move_base的参数设置文件。

turtlebot3_navigation/param/move_base_params.yaml

```

# move_base处于非活动状态时，是否停止costmap节点的选项
shutdown_costmaps: false
# 向机器人底座发送速度命令的控制周期 (Hz)
controller_frequency: 3.0
# 控制器在执行space-clearing操作之前等待接收控制信息的最长时间 (以秒为单位)
controller_patience: 1.0
# 全局规划的重复周期 (Hz)
planner_frequency: 2.0
# 在space-clearing操作之前等待查询可用规划的时间上限 (以秒为单位)
planner_patience: 1.0
# 在执行还原操作 (recovery behavior) 之前，允许机器人来回移动的时间 (以秒为单位)
oscillation_timeout: 10.0
# 为使机器人不被认为是在来回移动而需要挪动的最小距离
# 以meter为单位，若移动大于或等于下面的距离则oscillation_timeout会被初始化）。

```

```
oscillation_distance: 0.2  
# 在还原操作中costmap被初始化时，比该给定距离更远的障碍物将从地图中移除。  
conservative_reset_dist: 0.1
```

costmap

导航使用占用网格地图。基于这种网格地图，根据机器人的位置和从传感器获得的周围信息，将各像素计算为障碍物、不可移动区域和可移动区域。这时用到的概念就是costmap。costmap由三种文件组成：公共的costmap_common_params.yaml文件、全局区域运动规划所需的global_costmap_params.yaml文件以及本地所需的local_costmap_params.yaml文件。TurtleBot3根据型号分为costmap_common_params_burger.yaml文件和costmap_common_params_waffle.yaml文件。

首先，是对Turtlebot3 Burger的参数设定。

turtlebot3_navigation/param/costmap_common_params_burger.yaml

```
# 当物体与机器人的距离在如下距离内时，将物体视为障碍物。  
obstacle_range: 2.5  
# 传感器值大于如下距离的数据被视为自由空间（freespace）。  
raytrace_range: 3.5  
# 将机器人的外部尺寸以多边形的形式来提供。  
footprint: [[-0.110, -0.090], [-0.110, 0.090], [0.041, 0.090], [0.041, -0.090]]  
# 记录机器人的半径。这里我们使用上面的footprint设置而不是robot_radius。  
# robot_radius: 0.105  
# 膨胀区的半径，以防止接近障碍物。  
inflation_radius: 0.15  
# costmap计算中使用的缩放变量。计算公式如下。  
#  $\exp(-1.0 * \text{cost\_scaling\_factor} * (\text{distance\_from\_obstacle} - \text{inscribed\_radius}))^{(254 - 1)}$   
cost_scaling_factor: 0.5  
# 从voxel(voxel-grid)和costmap(costmap_2d)中选择要使用的costmap。  
map_type: costmap  
# tf之间相对坐标变换时间的允许的误差  
transform_tolerance: 0.2  
# 指定要使用的传感器  
observation_sources: scan  
# 激光扫描的数据类型、话题类型、是否反映到costmap、是否设置障碍物高度  
scan: {data_type: LaserScan, topic: scan, marking: true, clearing: true}
```

以下是TurtleBot3 Waffle的参数设置。与TurtleBot3 Burger不同的是，模型的外形尺寸（footprint）和膨胀区域半径（inflation_radius，防止接近障碍物）对于每个模型都是不同的。此外的值都相同，对于每个参数的描述，请参阅Burger的说明。

turtlebot3_navigation/param/costmap_common_params_waffle.yaml

```
obstacle_range: 2.5
raytrace_range: 3.5
footprint: [[-0.205, -0.145], [-0.205, 0.145], [0.077, 0.145], [0.077, -0.145]]
inflation_radius: 0.20
cost_scaling_factor: 0.5
map_type: costmap
transform_tolerance: 0.2
observation_sources: scan
scan: {data_type: LaserScan, topic: scan, marking: true, clearing: true}
```

turtlebot3_navigation/param/global_costmap_params.yaml

```
global_costmap:
  global_frame: /map          # 设置地图框架
  robot_base_frame: /base_footprint  # 设置机器人底座框架
  update_frequency: 2.0        # 更新周期
  publish_frequency: 0.1       # 发布周期
  static_map: true            # 是否使用给定地图的设置
  transform_tolerance: 1.0     # 允许的转换时间
```

turtlebot3_navigation/param/local_costmap_params.yaml

```
local_costmap:
  global_frame: /odom          # 地图框架设置
  robot_base_frame: /base_footprint  # 机器人底座框架设置
  update_frequency: 2.0        # 更新周期
  publish_frequency: 0.5       # 发布周期
  static_map: false            # 是否使用给定地图的设置
  rolling_window: true         # 局部地图窗口设置
  width: 3.5                  # 局部地图窗口宽度
  height: 3.5                 # 局部地图窗口高度
  resolution: 0.05             # 局部地图窗口分辨率（米/单元格）
  transform_tolerance: 1.0     # 允许的转换时间
```

dwa_local_planner

dwa_local_planner是一个最终将移动速度命令传给机器人的功能包，会设置其参数。

`turtlebot3_navigation/param/dwa_local_planner_params.yaml`

DWAPlannerROS:

```
# 机器人参数设定
max_vel_x: 0.18          # x轴最大速度 (meter/sec)
min_vel_x:-0.18          # x轴最小速度 (meter/sec)
max_vel_y: 0.0            # 仅适用于全向机器人，因此省略
min_vel_y: 0.0            # 仅适用于全向机器人，因此省略
max_trans_vel: 0.18       # 最大平移速度 (meter/sec)
min_trans_vel: 0.05       # 最小平移速度 (meter/sec)，当值为负数时也可以后退
# trans_stopped_vel: 0.01 # 平移停止速度 (meter/sec)
max_rot_vel: 1.8          # 最大旋转速度 (radian/sec)
min_rot_vel: 0.7          # 最小旋转速度 (radian/sec)
# rot_stopped_vel: 0.01 # 旋转停止速度 (radian/sec)
acc_lim_x: 2.0            # x轴加速度限制 (meter/sec^2)
acc_lim_y: 0.0             # y轴加速度限制 (meter/sec^2)
acc_lim_theta: 2.0         # theta轴角加速度限制 (radian/sec^2)

# 目标地点的允许误差
yaw_goal_tolerance: 0.15  # yaw轴离目标地点允许的误差 (弧度)
xy_goal_tolerance: 0.05   # x, y离目标地点允许的距离误差 (米)

# 前向仿真 (Forward Simulation) 参数
sim_time: 3.5            # 前向仿真轨迹时间
vx_samples: 20             # 在x轴速度空间中搜索的样本数
vy_samples: 0               # 在y轴速度空间中搜索的样本数
vtheta_samples: 40          # 在theta轴速度空间中搜索的样本数

# 轨迹评分参数 (轨迹评估)
# 用于轨迹评估的成本函数的分数计算如下。
# cost =
# path_distance_bias * (distance to path from the endpoint of the trajectory in meters)
# + goal_distance_bias * (distance to local goal from the endpoint of the trajectory in meters)
# + occdist_scale * (maximum obstacle cost along the trajectory in obstacle cost (0-254))
path_distance_bias: 32.0      # 衡量控制器遵循给定路径的一致程度的加权值
```

```
goal_distance_bias: 24.0          # 判断是否接近目标地点和控制速度的加权值
occdist_scale: 0.04              # 有关避障的加权值
forward_point_distance: 0.325    # 机器人中心与附加得分点之间的距离 (meter)
stop_time_buffer: 0.2            # 机器人在碰撞之前为了停止所需的最短时间 (sec)
scaling_speed: 0.25              # scaling speed (meter/sec)
max_scaling_factor: 0.2          # maximum scaling factor

# 防止震荡动作的参数
# 震荡标志复位之前机器人所需移动的距离
oscillation_reset_dist: 0.05

# 调试
publish_traj_pc: true           # 移动轨迹调试设置
publish_cost_grid_pc: true       # costmap调试设置
global_frame_id: odom            # 全局框架ID设置
```

map

将先前创建的占用网格地图 (occupancy grid map) 保存到/maps目录。没有其他配置参数。

```
/maps/map.pgm
/maps/map.yaml
```

turtlebot3_nav.rviz

它是一个包含ROS的可视化工具RViz的设置信息文件。它将加载RViz的显示插件中的Grid、Robot Model、TF、LaserScan、Map、Global Map、Local Map和Amcl Particles。建议使用以下文件，而不使用单独的配置文件。使用方法是，按如下命令运行rosrun时，可以指定为一个选项。

```
$ rosrun rviz rviz -d `rospack find turtlebot3_navigation`/rviz/turtlebot3_nav.rviz
```

至此，解释了使用导航功能包所需要的一切。在下一节中，我们将讨论costmap、Adaptive Monte Carlo Localization(AMCL)和Dynamic Window Approach(DWA)的理论。

11.7. 导航理论篇

11.7.1. Costmap

机器人的位置是根据从编码器和惯性传感器（IMU传感器）获得的测位来估计的。然后，通过安装在机器人上的距离传感器来计算机器人与障碍物之间的距离。导航系统将机器人位置、传感器姿态、障碍物信息和作为SLAM地图的结果而获得的占用网格地图调用到固定地图（static map），用作占用区域（occupied area）、自由区域（free area）和未知区域（unknown area）。

在导航中，基于上述四种因素，计算障碍物区域、预计会和障碍物碰撞的区域以及机器人可移动区域，这被称为成本地图（costmap）。根据导航类型，成本地图又被分成两部分。一个是global_costmap，在全局移动路径规划中以整个区域为对象建立移动计划，其输出结果就是global_costmap。而另一个被称为local_costmap，这是在局部移动路径规划中，在以机器人为中心的部分限定区域中规划移动路径时，或在躲避障碍物时用到的地图。然而，这两种成本图的表示方法是相同的，尽管它们的目的不同。

costmap用0到255之间的值来表示。数值的含义如图11-19所示，简单地说，根据该值可以知道机器人是位于可移动区域还是位于可能与障碍物碰撞的区域。每个区域的计算取决于第11.6节中指定的costmap配置参数。

- 000：机器人可以自由移动的free area（自由区域）
- 001~127：碰撞概率低的区域
- 128~252：碰撞概率高的区域
- 253~254：碰撞区域
- 255：机器人不能移动的占用区域（occupied area）

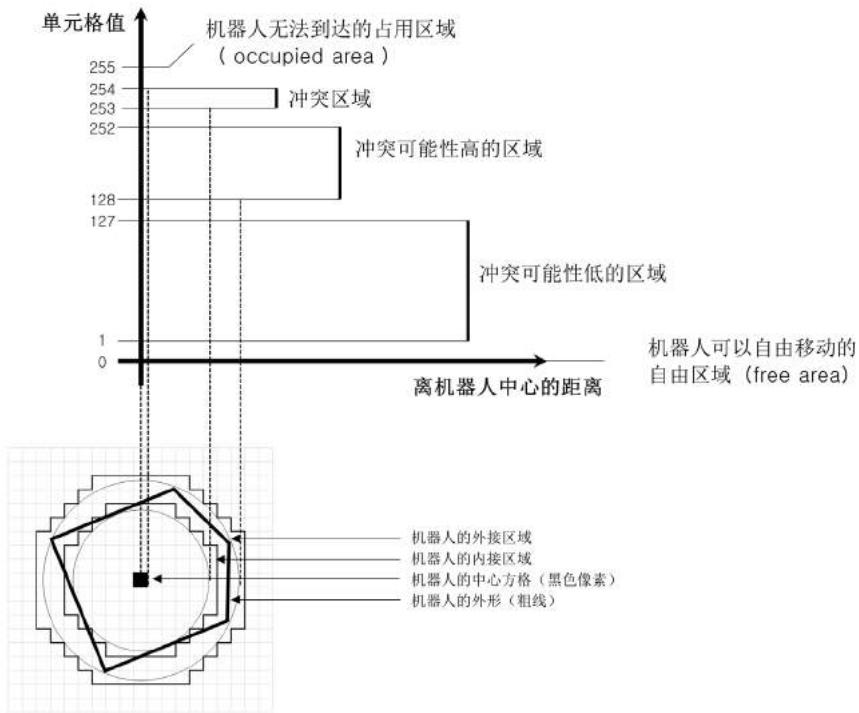


图 11-19 障碍距离与costmap值的关系

例如，实际的costmap如图11-20所示。具体的，机器人模型位于中间，其周围的黑色边框对应于机器人的外表面。当这个轮廓线碰到实际的墙壁时，意味着机器人会发生碰撞。绿色用从激光传感器获得的距离传感器值表示的障碍物，灰度的颜色越深的位置意味着碰撞的可能性越高。这同样适用于用颜色表示的情况，粉红色区域是实际的障碍物，蓝色区域是机器人中心位置进入该区域则会发生碰撞的位置，且边框用粗红色像素绘制。这些颜色可以由用户在RViz中修改，因此可以说没有太大的意义。

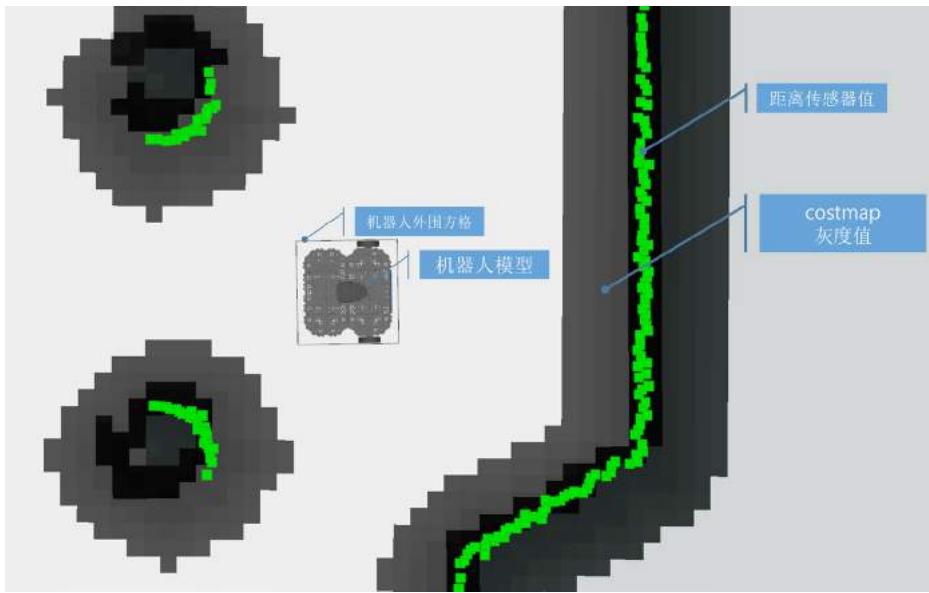


图 11-20 costmap的表示方式（灰度）

11.7.2. AMCL

正如在11.4节的SLAM理论篇的粒子滤波（particle filter）的说明，蒙特卡罗定位（MCL）位置估计算法在位置估计领域中被广泛运用。AMCL（自适应蒙特卡罗定位）可以被看作蒙特卡罗位置估计的改进版本，它通过在蒙特卡罗位置估计算法中使用少量样本来减少执行时间，以此提高实时性能。那么，我们先来看看基本的蒙特卡罗位置估计（MCL）。

蒙特卡罗位置估计（MCL）的最终目的是确定机器人在特定环境中的位置。也就是说，我们必须在地图中得到x、y和 θ 。为此，MCL计算机器人所在位置的概率。首先，机器人在t时刻的位置和方向（x, y, θ ）是 x_t ，距离传感器到t时刻为止获得的距离信息记为 $z_{0..t} = \{z_0, z_1, \dots, z_t\}$ ，编码器到t时刻为止获得的机器人的运动信息记为 $u_{0..t} = \{u_0, u_1, \dots, u_t\}$ ，则可以如下计算belief（使用贝叶斯更新公式的后验概率）。

$$bel(x_t) = p(x_t | z_{0..t}, u_{0..t}) \quad (\text{式 11-11})$$

由于机器人可能存在硬件误差，因此建立传感器模型和移动模型，并且如下执行贝叶斯滤波器（bayes filter）的预测（prediction）和更新（update）。首先，在预测阶段中，利用机器人的移动模型 $p(x_t | x_{t-1}, u_{t-1})$ 、前一个位置上的概率 $bel(x_{t-1})$ ，和从编码器获得的移动信息 u ，计算下一个时刻的机器人位置 $bel'(x_t)$ 。

$$bel'(x_t) = \int p(x_t | x_{t-1}, u_{t-1}) bel(x_{t-1}) dx_{t-1} \quad (\text{式 11-12})$$

下面是校正步骤，这次我们利用传感器模型 $p(z_t | x_t)$ 、前面求得的概率 $bel'(x_t)$ 和归一化常数 η_t ，可以求得基于传感器信息提高准确度的当前位置的概率 $bel(x_t)$ 。

$$bel(x_t) = \eta_t p(z_t | x_t) bel'(x_t) \quad (\text{式 11-13})$$

以下步骤通过上面得出的当前位置的概率 $bel(x_t)$ ，用粒子滤波器生成N个粒子来估计位置。具体请参考11.4节SLAM理论篇的粒子滤波器（particle filter）的说明。在MCL中，使用“样品”这个术语来代替“粒子”。总的来说会经过SIR（Sampling Importance weighting Re-sampling）过程。首先，是抽样（sampling）过程。这里，使用前一个位置的概率 $bel(x_{t-1})$ 中的机器人的移动模型 $p(x_t | x_{t-1}, u_{t-1})$ 来提取新的样本集合 x_t' 。利用该样品集合 x_t' 中的第*i*个样品 $x_t'^{(i)}$ 、由距离传感器获得的距离信息 z_t 和归一化常数 ηp 来计算权重值 $\omega_t^{(i)}$ 。

$$\omega_t^{(i)} = \eta p(z_t | x_t'^{(i)}) \quad (\text{式 11-14})$$

最后，在重采样过程中，我们使用样本 x_t' 和权重 $\omega_t^{(i)}$ 来创建N个新的样品（粒子）集合 X_t 。

$$X_t = \{x_t^{(j)} \mid j = 1 \dots N\} \sim \{x_t'^{(i)}, \omega_t^{(i)}\} \quad (\text{式 11-15})$$

以这种方式，在重复SIR过程的同时移动粒子，且提高机器人位置估计的准确度。例如，如图11-21所示，我们可以看到随着时间随t1、t2、t3、t4变化的过程中位置估计在逐渐收敛。所有这些过程都参考了在机器人工程中被称为概率领域的教科书的Sebastian Thrun教授的著作“Probabilistic Robotics”一书。如有时间建议读者务必参阅。

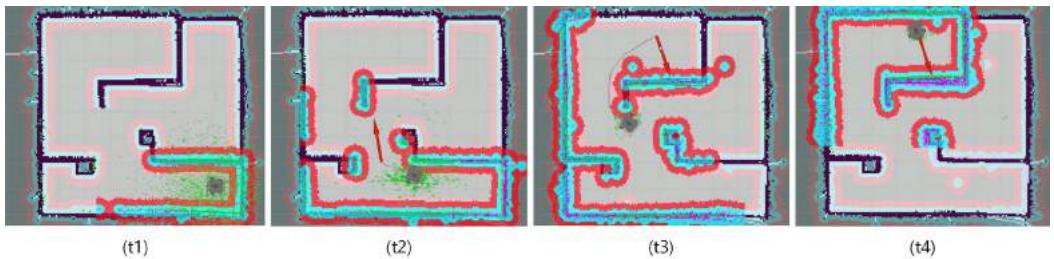


图 11-21 用于机器人位置估计的AMCL过程

11.7.3. Dynamic Window Approach(DWA)

动态窗口方法 (Dynamic Windows Approach, DWA) 是在规划移动路径和躲避障碍物时常用的方法，具体是指在机器人的速度搜索空间 (velocity search space) 中选择适当的速度，以躲避可能碰撞的障碍物的同时能迅速到达目的地。在ROS中，局部移动规划中曾广泛使用Trajectory planner，而最近由于DWA的性能优越，因此DWA在逐渐代替Trajectory planner。

首先，如图11-22所示，用平移速度 v 和旋转速度 ω 为轴的速度搜索空间 (velocity search space) 来表示机器人，而不是用x轴和y轴的位置坐标系表示。在这个空间中，由于硬件限制，机器人具有最大允许速度，这被称为动态窗口 (Dynamic Window) 。

v : 平移速度 (meter/sec)

ω : 旋转速度 (radian/sec)

V_s : 最大速度区域

V_a : 允许的速度区域

V_c : 当前速度

V_r : 动态窗口中的速度区域

a_{max} : 最大加/减速度

$G(v, \omega) = \sigma(\alpha \cdot heading(v, \omega) + \beta \cdot dist(v, \omega) + \gamma \cdot velocity(v, \omega))$: 目标函数

$heading(v, \omega)$: 180 - (机器人的方向与目标点的方向之差)

$dist(v, \omega)$: 离障碍物的距离

$velocity(v, \omega)$: 选择的速度

α, β, γ : 权重常数

$\sigma(x)$: Smooth Function

在这个动态窗口中，通过使用目标函数，获得满足条件的平移速度和旋转速度。满足条件意味着使考虑了机器人的方向、速度和碰撞的目标函数达到最大化。如果绘制出来，则如图11-23所示，我们可以在各种 和 中找到最优速度并移动机器人。

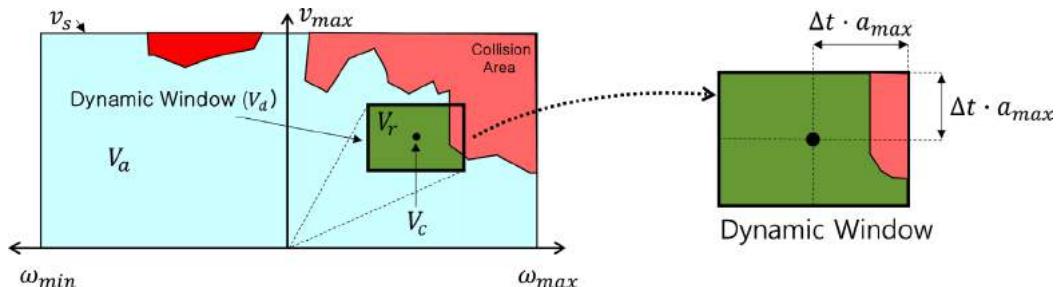


图 11-22 机器人的速度搜索空间（velocity search space）和动态窗口（Dynamic Window）

至此，结束了对SLAM和导航的实习、应用和理论的讲解。虽然这主要是用移动机器人平台Turtlebot3来说明的，但也适用于其他机器人，所以如果读者正在使用另一个机器人或开发了自己的机器人，那么也可以应用它。

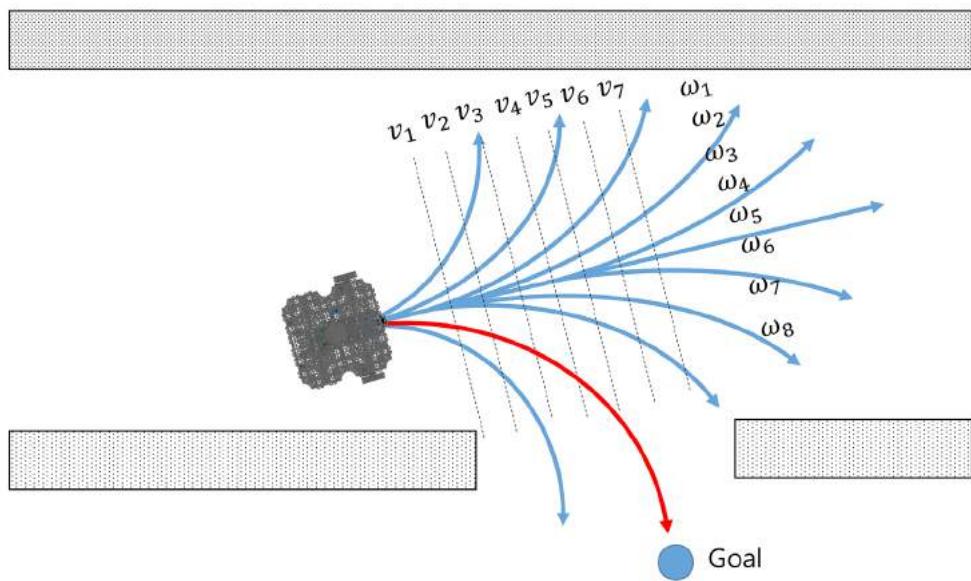


图 11-23 平移速度和转速

第12章

服务机器人

12.1. 配送服务机器人

目前，SLAM和导航技术应用于多种情况。在机器人领域，有用于车辆、工厂和生产线的机器人，还有用于配送服务的机器人，并且其技术也在迅速发展。在第10章和第11章中，我们了解了SLAM和导航功能包的组成和配置。在第12章中，我们来制作一个简单实用的配送服务机器人。

12.2. 配送服务机器人的结构

12.2.1. 系统结构

如图12-1所示，配送服务机器人的服务系统设计可以分为“服务核心”、“服务主机”和“服务从机”等区块。

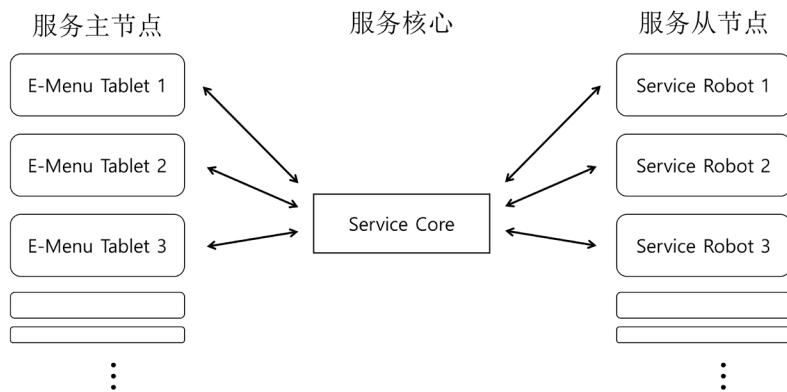


图 12-1 配送服务机器人系统设计实例

服务核心

服务核心是综合管理客户订单状态和机器人服务执行状态的数据库的一种。在接到客户的订单后，它在处理订单和调度机器人的服务处理方面起着关键作用。由于一个客户的订单和处理该订单的机器人的状态影响另一个客户的订单，所以服务核心必须作为一个单独的个体来存在。设计时，可以小到用键盘输入接单的形式，也可以大到用如平板PC的设备接单的形式。

服务主机

服务主机直接接收客户的订单并将订单内容转达给服务核心。另外，也可以列出客户可以订购的物品列表，并将机器人的服务执行状态发送给客户。为此，服务主机必须与由服务核心管理的数据库同步。

服务从机

服务从机是实际配送客户订购的产品的对象或机器人平台，在服务执行期间实时地将服务状态更新到服务核心。

12.2.2. 系统设计

服务主机、服务从机和服务核心等区块可以设计为都在单台计算机上运行或各个区块分别分配给各自的计算机来运行。但是，如果在一台计算机上运行所有工作，则该计算机的处理能力可能无法承担所有任务，而如果给每一种区块分配单独的计算机，则无线网络有可能无法承受要传输的数据量，因此要根据实情适当分配。而且，由于ROS 1.x基本上是一个旨在控制一台机器人的系统，所以如果要在多台PC上使用ROS，则应使用以下命令尽可能地同步每台计算机的时间。

```
$ sudo ntpdate ntp.ubuntu.com
```

例如，如图12-2所示，笔者使用以下设备实现了配送服务机器人系统。

- 服务核心：NUC i5（3台，包括roscore和服务核心的PC）
- 服务主机：SAMSUNG NOTE 10.4 Android OS 3台
- 服务从机：TurtleBot3 Carrier (用TurtleBot3 Waffle定制的配送机器人) 的三台英特尔Joule 570



图 12-2 配送服务机器人和运营系统的实际照片

总体构想完成后，需要决定在每个区块的节点之间应该交换什么消息，如图12-3所示。



图 12-3 配送服务机器人系统中的每个区块发送和接收消息的示例

尽管可以用一个平板控制多个机器人，但是为了简化系统结构，说明时，按照给每个机器人配备一个平板的结构来进行说明。当客户下订单时，用于下单的平板将他们的平板ID和关于所选项目的信息发送到服务核心。服务核心基于平板ID来确定应该移动哪个服务机器人来执行服务，并向服务机器人发送与订购的项目相对应的目的地，以便服务机器人可以执行服务。

服务机器人根据自己的路径规划（Path planning）算法到达目的地。此时，服务机器人会把各种情况传送给服务核心，例如包括与障碍物的碰撞、未找到路径、到达目的地等可能面临的状况，以及是否成功获取订购的物品等。服务核心综合处理实时接收到的机器人的服务状态、与订购物品相关的重复订单以及在机器人工作期间接收到的订单等情况，并通过平板向客户提供反馈。在这个过程中发送和接收的所有信息将使用由开发者根据各个目的新配置的msg文件或srv文件。

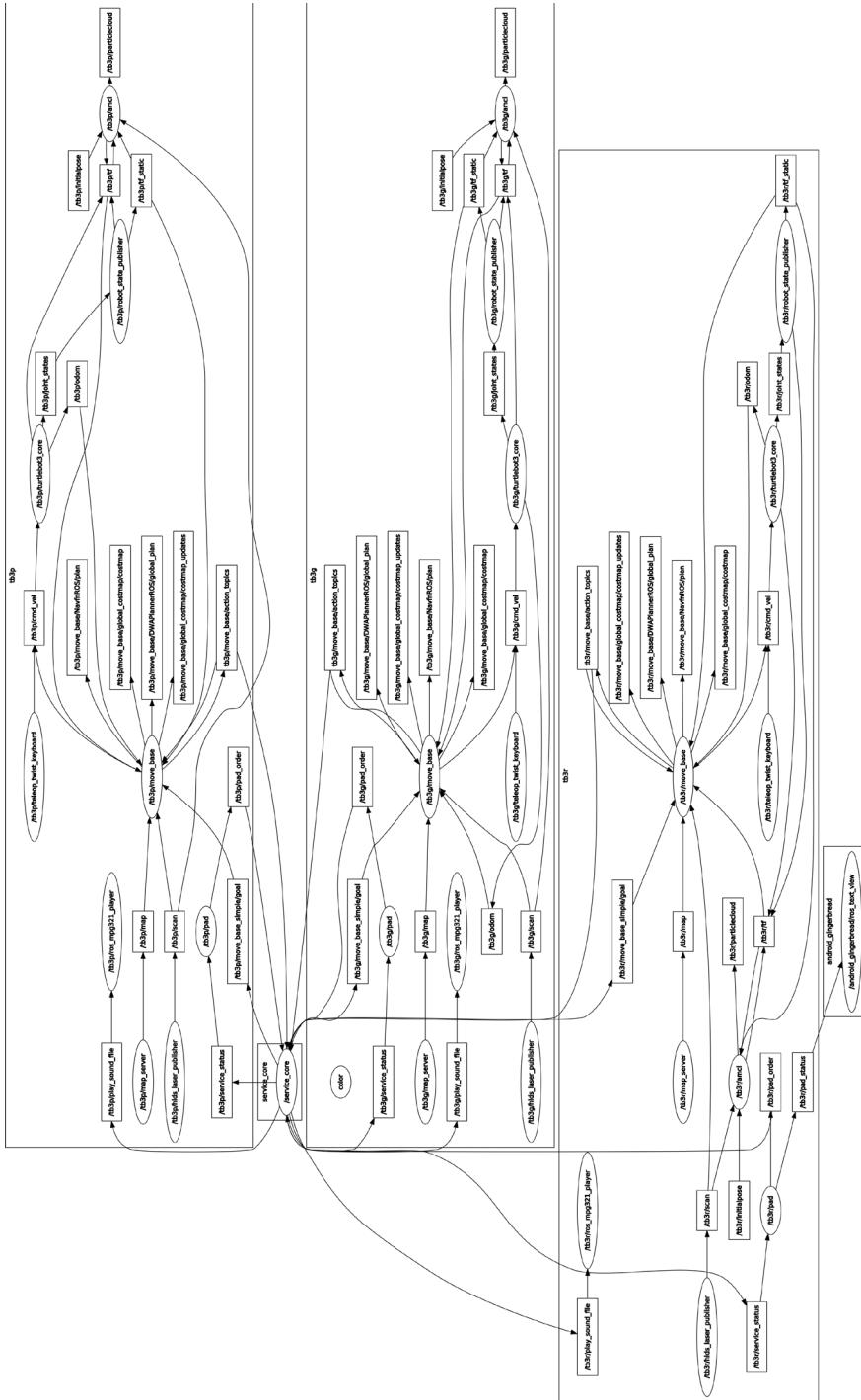


图 12-4 配送服务机器人的节点结构示例

在我们解释每个节点之前，先来了解整个大局。图12-4用关系图显示了在第12章中将创建的配送服务机器人的所有节点和话题的相关性。该图由三个组（group）和service_core组成。每个组中，与订单平板和服务机器人的所有节点都在名为“tb3p”、“tb3g”和“tb3r”的namespace中配对存在。由于service_core需要管理三个组，因此不属于任何组，它独立于三个组而存在。如果要配置成平板和机器人不配对的方式，也就是说如要采用一个平板控制多台机器人的系统的话，要为平板和每台服务机器人都创建单独的namespace下的组。

在开发服务机器人系统时使用组namespace¹的原因是，当多个机器人或计算机要同时使用一种ROS功能包时，要注册到ROS主节点的名称（如节点或话题等）会重复，因此会无法运行。

图12-5显示了一个节点图，该节点图着重表达分组到三个namespace的话题当中由service_core直接发送和接收的那些话题。service_core通过名为/pad_order的话题接收订单，并通过/move_base/action_topics话题接收机器人的路径查找情况和到达目的地与否。它还通过/service_status话题提供服务状态，通过/play_sound_file传递已录音的语音文件的位置，并通过(move_base_simple/goal)来传达机器人的目的地的坐标。



图 12-5 service_core节点发送和接收的话题列表

¹ <http://wiki.ros.org/roslaunch/XML/group>

图12-6是用于准备配送服务的流程图。导航使用SLAM制作的地图。在服务中，有必要考虑将哪个位置用作服务区，然后确认和记录该区域在生成的地图上的坐标。该坐标值将在下面要说明的service_core中通过获得和设置ROS参数来使用。在下面的示例中，需要“客户下单的位置”和“从机器人获取物品的位置”在地图上的坐标值。有关SLAM和导航的更多信息，请参阅第10章和第11章。

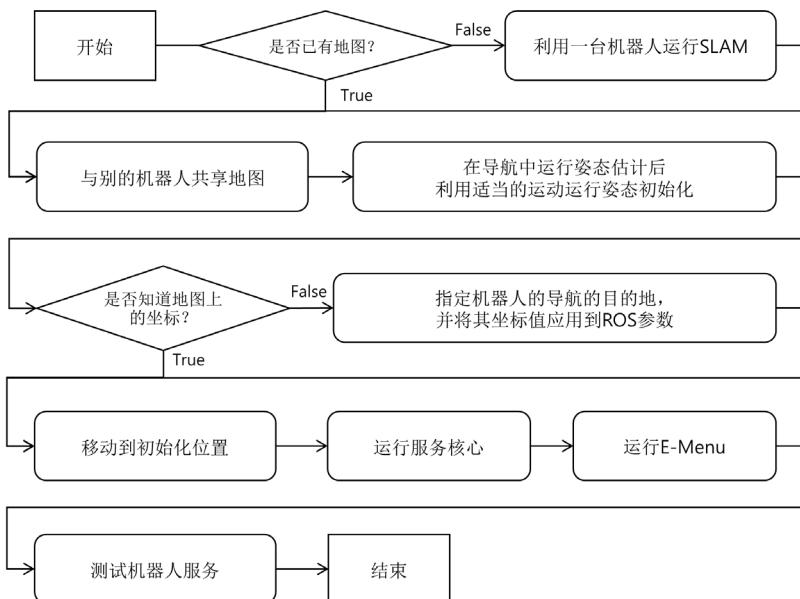


图 12-6 准备配送服务的流程图

以下链接以开源形式提供了本章描述的配送服务机器人制作所需的源代码。下面，将描述与各个区域对应的节点的配置和源代码。

- https://github.com/ROBOTIS-GIT/turtlebot3_deliver

12.2.3. 服务核心节点

图12-7表示服务核心节点的基本结构。该节点以main()函数开始，首先通过fnInitParam()设置ROS参数。之后，接收有关客户订单的数据的bReceivePadOrder()函数和接收机器人的目的地到达与否信息的cbCheckArrivalStatus()函数，会被声明为在话题订阅时被运行。并且通过fnPubServiceStatus()和fnPubPose()函数发布服务状态和机器人目的地的坐标值。这个过程在无限循环中执行，直到按下[Ctrl + c]键。

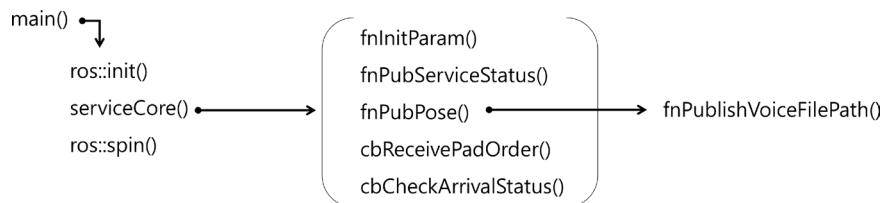


图 12-7 服务核心的基本结构

当运行 `serviceCore()` 函数时，会调用 `fnInitParam()` 函数，并定义用于收发各种数据的发布者和订阅者。`service_core` 节点需要处理来自 3 个机器人和 3 个平板的话题，所以定义了 3 种发布者和 3 种订阅者。各个发布者和订阅者的工作任务如下。

项目	说明
<code>pubServiceStatusPad</code>	向平板发布配送服务状态的发布者
<code>pubPlaySound</code>	发布录音语音文件位置的发布者
<code>subPadOrder</code>	从平板订阅客户订单的订阅者
<code>subArrivalStatus</code>	订阅机器人到达与否的订阅者

表 12-1 `service_core.cpp` 中的 `ServiceCore()` 函数

```

/turtlebot3_carrier/src/service_core.cpp (只提取一部分)

ServiceCore()
{
    fnInitParam();

    pubServiceStatusPadTb3p = nh_.advertise<turtlebot3_carrier::ServiceStatus>("/tb3p/service_status", 1);
    pubServiceStatusPadTb3g = nh_.advertise<turtlebot3_carrier::ServiceStatus>("/tb3g/service_status", 1);
    pubServiceStatusPadTb3r = nh_.advertise<turtlebot3_carrier::ServiceStatus>("/tb3r/service_status", 1);

    pubPlaySoundTb3p = nh_.advertise<std_msgs::String>("/tb3p/play_sound_file", 1);
    pubPlaySoundTb3g = nh_.advertise<std_msgs::String>("/tb3g/play_sound_file", 1);
    pubPlaySoundTb3r = nh_.advertise<std_msgs::String>("/tb3r/play_sound_file", 1);

    pubPoseStampedTb3p = nh_.advertise<geometry_msgs::PoseStamped>("/tb3p/move_base_simple/goal", 1);
    pubPoseStampedTb3g = nh_.advertise<geometry_msgs::PoseStamped>("/tb3g/move_base_simple/goal", 1);
    pubPoseStampedTb3r = nh_.advertise<geometry_msgs::PoseStamped>("/tb3r/move_base_simple/goal", 1);

    subPadOrderTb3p = nh_.subscribe("/tb3p/pad_order", 1, &ServiceCore::cbReceivePadOrder, this);
}

```

```

subPadOrderTb3g = nh_.subscribe("/tb3g/pad_order", 1, &ServiceCore::cbReceivePadOrder, this);
subPadOrderTb3r = nh_.subscribe("/tb3r/pad_order", 1, &ServiceCore::cbReceivePadOrder, this);

subArrivalStatusTb3p = nh_.subscribe("/tb3p/move_base/result", 1,
&ServiceCore::cbCheckArrivalStatusTB3P, this);
subArrivalStatusTb3g = nh_.subscribe("/tb3g/move_base/result", 1,
&ServiceCore::cbCheckArrivalStatusTB3G, this);
subArrivalStatusTb3r = nh_.subscribe("/tb3r/move_base/result", 1,
&ServiceCore::cbCheckArrivalStatusTB3R, this);

ros::Rate loop_rate(5);

while (ros::ok())
{
    fnPubServiceStatus();

    fnPubPose();
    ros::spinOnce();
    loop_rate.sleep();
}
}

```

fnInitParam()函数从给定参数文件获取机器人在地图上的目标姿态（位置+方向）数据，如下所示。在这个例子中，机器人从互不相同的三个下单区域出发并移动到订购的物品所在的位置。可以下单的物品设为三种，因此总共需要知道地图上的6个位置的坐标。fnInitParam()函数的结构如下。

项目	说明
poseStampedTable	客户下订单的位置的坐标
poseStampedCounter	载货区域的坐标

表 12-2 service_core.cpp中的fnInitParam()函数

/turtlebot3_carrier/src/service_core.cpp (只提取一部分)

```

void fnInitParam()
{
    nh_.getParam("table_pose_tb3p/position", target_pose_position);
    nh_.getParam("table_pose_tb3p/orientation", target_pose_orientation);
}

```

```
poseStampedTable[0].header.frame_id = "map";
poseStampedTable[0].header.stamp = ros::Time::now();

poseStampedTable[0].pose.position.x = target_pose_position[0];
poseStampedTable[0].pose.position.y = target_pose_position[1];
poseStampedTable[0].pose.position.z = target_pose_position[2];

poseStampedTable[0].pose.orientation.x = target_pose_orientation[0];
poseStampedTable[0].pose.orientation.y = target_pose_orientation[1];
poseStampedTable[0].pose.orientation.z = target_pose_orientation[2];
poseStampedTable[0].pose.orientation.w = target_pose_orientation[3];

nh_.getParam("table_pose_tb3g/position", target_pose_position);
nh_.getParam("table_pose_tb3g/orientation", target_pose_orientation);

poseStampedTable[1].header.frame_id = "map";
poseStampedTable[1].header.stamp = ros::Time::now();

poseStampedTable[1].pose.position.x = target_pose_position[0];
poseStampedTable[1].pose.position.y = target_pose_position[1];
poseStampedTable[1].pose.position.z = target_pose_position[2];

poseStampedTable[1].pose.orientation.x = target_pose_orientation[0];
poseStampedTable[1].pose.orientation.y = target_pose_orientation[1];
poseStampedTable[1].pose.orientation.z = target_pose_orientation[2];
poseStampedTable[1].pose.orientation.w = target_pose_orientation[3];

nh_.getParam("table_pose_tb3r/position", target_pose_position);
nh_.getParam("table_pose_tb3r/orientation", target_pose_orientation);

poseStampedTable[2].header.frame_id = "map";
poseStampedTable[2].header.stamp = ros::Time::now();

poseStampedTable[2].pose.position.x = target_pose_position[0];
poseStampedTable[2].pose.position.y = target_pose_position[1];
poseStampedTable[2].pose.position.z = target_pose_position[2];
```

```
poseStampedTable[2].pose.orientation.x = target_pose_orientation[0];
poseStampedTable[2].pose.orientation.y = target_pose_orientation[1];
poseStampedTable[2].pose.orientation.z = target_pose_orientation[2];
poseStampedTable[2].pose.orientation.w = target_pose_orientation[3];

nh_.getParam("counter_pose_bread/position", target_pose_position);
nh_.getParam("counter_pose_bread/orientation", target_pose_orientation);

poseStampedCounter[0].header.frame_id = "map";
poseStampedCounter[0].header.stamp = ros::Time::now();

poseStampedCounter[0].pose.position.x = target_pose_position[0];
poseStampedCounter[0].pose.position.y = target_pose_position[1];
poseStampedCounter[0].pose.position.z = target_pose_position[2];

poseStampedCounter[0].pose.orientation.x = target_pose_orientation[0];
poseStampedCounter[0].pose.orientation.y = target_pose_orientation[1];
poseStampedCounter[0].pose.orientation.z = target_pose_orientation[2];
poseStampedCounter[0].pose.orientation.w = target_pose_orientation[3];

nh_.getParam("counter_pose_drink/position", target_pose_position);
nh_.getParam("counter_pose_drink/orientation", target_pose_orientation);

poseStampedCounter[1].header.frame_id = "map";
poseStampedCounter[1].header.stamp = ros::Time::now();

poseStampedCounter[1].pose.position.x = target_pose_position[0];
poseStampedCounter[1].pose.position.y = target_pose_position[1];
poseStampedCounter[1].pose.position.z = target_pose_position[2];

poseStampedCounter[1].pose.orientation.x = target_pose_orientation[0];
poseStampedCounter[1].pose.orientation.y = target_pose_orientation[1];
poseStampedCounter[1].pose.orientation.z = target_pose_orientation[2];
poseStampedCounter[1].pose.orientation.w = target_pose_orientation[3];

nh_.getParam("counter_pose_snack/position", target_pose_position);
nh_.getParam("counter_pose_snack/orientation", target_pose_orientation);
```

```
poseStampedCounter[2].header.frame_id = "map";
poseStampedCounter[2].header.stamp = ros::Time::now();

poseStampedCounter[2].pose.position.x = target_pose_position[0];
poseStampedCounter[2].pose.position.y = target_pose_position[1];
poseStampedCounter[2].pose.position.z = target_pose_position[2];

poseStampedCounter[2].pose.orientation.x = target_pose_orientation[0];
poseStampedCounter[2].pose.orientation.y = target_pose_orientation[1];
poseStampedCounter[2].pose.orientation.z = target_pose_orientation[2];
poseStampedCounter[2].pose.orientation.w = target_pose_orientation[3];
}
```

在target_pose.yaml文件中显示的参数值是机器人为了执行其服务所需的地图上的坐标值。在地图上获取坐标值的方法有很多种，其中有一种简单的方法是在导航过程中通过“rostopic echo”命令获取姿态（pose）值。但是，由于每次通过SLAM执行地图绘制时该坐标值都发生变化，因此建议在实际导航期间尽可能避免重新绘制地图，并且减少物体和障碍物的位置变化以便使用同一个地图。

/turtlebot3_carrier/param/target_pose.yaml

```
table_pose_tb3p:
position: [-0.338746577501, -0.85418510437, 0.0]
orientation: [0.0, 0.0, -0.0663151963596, 0.997798724559]

table_pose_tb3g:
position: [-0.168751597404, -0.19147400558, 0.0]
orientation: [0.0, 0.0, -0.0466624033917, 0.998910716786]

table_pose_tb3r:
position: [-0.251043587923, 0.421476781368, 0.0]
orientation: [0.0, 0.0, -0.0600887022438, 0.998193041382]

counter_pose_bread:
position: [-3.60783815384, -0.750428497791, 0.0]
orientation: [0.0, 0.0, 0.999335763287, -0.0364421763375]
```

```

counter_pose_drink:
position: [-3.48697376251, -0.173366710544, 0.0]
orientation: [0.0, 0.0, 0.998398746904, -0.0565680314445]

counter_pose_snack:
position: [-3.62247490883, 0.39046728611, 0.0]
orientation: [0.0, 0.0, 0.998908838216, -0.0467026009308]

```

接下来的fnPubPose()函数的功能是，根据机器人当前所处的服务状态，在机器人到达目的地时设定下一个目的地。当机器人完成服务时，所有的参数都会被初始化。

项目	说明
is_robot_reached_target	机器人是否已经到达导航目的地
is_item_available	项目是否可以订购
item_num_chosen_by_pad	订购的物品的编号
robot_service_sequence	机器人的服务状态 0- 等待客户的订单 1- 刚收到客户的订单 2- 正在去取客户订购的物品 3- 正在装载订购的物品 4- 正在移动到客户的位置 5- 正将物品递给客户
fnPublishVoiceFilePath()	发布已经录制的语音文件的位置的函数
ROBOT_NUMBER	机器人的编号（开发者指定）

表 12-3 service_core.cpp中的fnPubPose()函数

/turtlebot3_carrier/src/service_core.cpp (只提取一部分)

```

void fnPubPose()
{
    if (is_robot_reached_target[ROBOT_NUMBER])
    {
        if (robot_service_sequence[ROBOT_NUMBER] == 1)
        {
            fnPublishVoiceFilePath(ROBOT_NUMBER, "~/voice/voice1-2.mp3");

            robot_service_sequence[ROBOT_NUMBER] = 2;
        }
    }
}

```

```

else if (robot_service_sequence[ROBOT_NUMBER] == 2)
{
    pubPoseStampedTb3p.publish(poseStampedCounter[item_num_chosen_by_pad[ROBOT_NUMBER]]);

    is_robot_reached_target[ROBOT_NUMBER] = false;

    robot_service_sequence[ROBOT_NUMBER] = 3;
}

else if (robot_service_sequence[ROBOT_NUMBER] == 3)
{
    fnPublishVoiceFilePath(ROBOT_NUMBER, "~/voice/voice1-3.mp3");

    robot_service_sequence[ROBOT_NUMBER] = 4;
}

else if (robot_service_sequence[ROBOT_NUMBER] == 4)
{
    pubPoseStampedTb3p.publish(poseStampedTable[ROBOT_NUMBER]);

    is_robot_reached_target[ROBOT_NUMBER] = false;

    robot_service_sequence[ROBOT_NUMBER] = 5;
}

else if (robot_service_sequence[ROBOT_NUMBER] == 5)
{
    fnPublishVoiceFilePath(ROBOT_NUMBER, "~/voice/voice1-4.mp3");

    robot_service_sequence[ROBOT_NUMBER] = 0;

    is_item_available[item_num_chosen_by_pad[ROBOT_NUMBER]] = 1;

    item_num_chosen_by_pad[ROBOT_NUMBER] = -1;
}
}
}
...
省略 ...

```

cbReceivePadOrder()函数接收订购时使用的平板的编号和订购商品的编号，以确定是否可以提供服务。如果可能，则将robot_service_sequence设置为“1”，以启动服务。这个函数的源代码如下。

项目	说明
pad_number	用于下单的平板的编号（要提供服务的机器人的编号）
item_number	订购的物品的编号

表 12-4 service_core.cpp中的cbReceivePadOrder()函数

```

/turtlebot3_carrier/src/service_core.cpp ( 只提取一部分 )

void cbReceivePadOrder(const turtlebot3_carrier::PadOrder padOrder)
{
    int pad_number = padOrder.pad_number;
    int item_number = padOrder.item_number;

    if (is_item_available[item_number] != 1)
    {
        ROS_INFO("Chosen item is currently unavailable");
        return;
    }

    if (robot_service_sequence[pad_number] != 0)
    {
        ROS_INFO("Your TurtleBot is currently on servicing");
        return;
    }

    if (item_num_chosen_by_pad[pad_number] != -1)
    {
        ROS_INFO("Your TurtleBot is currently on servicing");
        return;
    }

    item_num_chosen_by_pad[pad_number] = item_number;

    robot_service_sequence[pad_number] = 1; // just left from the table

    is_item_available[item_number] = 0;
}

```

下面显示的cbCheckArrivalStatus()函数确认订阅的机器人的移动状态。在由else处理的部分中，服务核心将对于机器人处于困境的情况作出响应，比如当机器人在路径寻找中遇到障碍时或者在路径寻找算法中不能找到路径时的情况。

/turtlebot3_carrier/src/service_core.cpp (只提取一部分)

```
void cbCheckArrivalStatusTB3P(const move_base_msgs::MoveBaseActionResult &rcvMoveBaseActionResult)
{
    if (rcvMoveBaseActionResult.status.status == 3)
    {
        is_robot_reached_target[ROBOT_NUMBER_TB3P] = true;
    }
    else
    {
        ...省略...
    }
}

void cbCheckArrivalStatusTB3G(const move_base_msgs::MoveBaseActionResult &rcvMoveBaseActionResult)
{
    ...省略...
}

void cbCheckArrivalStatusTB3R(const move_base_msgs::MoveBaseActionResult &rcvMoveBaseActionResult)
{
    ...省略...
}
```

以下源代码中显示的fnPublishVoicePath()函数将预先录制的语音文件的位置以字符串形式发布。实际上，为了在ROS上播放声音，需要用于播放语音文件的ROS功能包。

/turtlebot3_carrier/src/service_core.cpp (只提取一部分)

```
void fnPublishVoiceFilePath(int robot_num, const char* file_path)
{
    std_msgs::String str;

    str.data = file_path;

    if (robot_num == ROBOT_NUMBER_TB3P)
```

```
{  
    pubPlaySoundTb3p.publish(str);  
}  
else if (robot_num == ROBOT_NUMBER_TB3G)  
{  
    pubPlaySoundTb3g.publish(str);  
}  
else if (robot_num == ROBOT_NUMBER_TB3R)  
{  
    pubPlaySoundTb3r.publish(str);  
}  
}
```

12.2.4. 服务主节点

在本例中，服务主节点在安装了Android操作系统的平板电脑上运行。但是，也可以使节点从终端接受订单。下一节将讨论使用Android OS平台的ROS Java编程。下面描述的服务主节点的源代码是使用ROS Java²基本提供的“android_tutorial_pubsub”例子的源代码创建的，这是一个发布和订阅话题的简单的例子。

turtlebot3_carrier_pad/ServicePad.java

```
package org.ros.android.android_tutorial_pubsub;  
  
import org.ros.concurrent.CancellableLoop;  
import org.ros.message.MessageListener;  
import org.ros.namespace.GraphName;  
import org.ros.node.AbstractNodeMain;  
import org.ros.node.ConnectedNode;  
import org.ros.node.topic.Publisher;  
import org.ros.node.topic.Subscriber;  
  
import javax.security.auth.SubjectDomainCombiner;  
  
public class ServicePad extends AbstractNodeMain {
```

² <http://wiki.ros.org/rosjava>

```
private String pub_pad_order_topic_name;
private String sub_service_status_topic_name;
private String pub_pad_status_topic_name;

private int robot_num = 0;
private int selected_item_num = -1;

private boolean jump = false;
private int[] item_num_chosen_by_pad = {-1, -1, -1};
private int[] is_item_available = {1, 1, 1};
private int[] robot_service_sequence = {0, 0, 0};

public boolean[] button_pressed = {false, false, false};

public ServicePad() {
    this.pub_pad_order_topic_name = "/tb3g/pad_order";
    this.sub_service_status_topic_name = "/tb3g/service_status";
    this.pub_pad_status_topic_name = "/tb3g/pad_status";
}

public GraphName getDefaultNodeName() {
    return GraphName.of("tb3g/pad");
}

public void onStart(ConnectedNode connectedNode) {
    final Publisher pub_pad_order = connectedNode.newPublisher(this.pub_pad_order_topic_name,
        "turtlebot3_carrier/PadOrder");
    final Publisher pub_pad_status = connectedNode.newPublisher(this.pub_pad_status_topic_name,
        "std_msgs/String");

    final Subscriber<turtlebot3_carrier.ServiceStatus> subscriber =
        connectedNode.newSubscriber(this.sub_service_status_topic_name, "turtlebot3_carrier/
        ServiceStatus");

    subscriber.addMessageListener(new MessageListener<turtlebot3_carrier.ServiceStatus>() {
        @Override
        public void onNewMessage(turtlebot3_carrier.ServiceStatus serviceStatus)
        {
            item_num_chosen_by_pad = serviceStatus.item_num_chosen_by_pad;
        }
    });
}
```

```
is_item_available = serviceStatus.is_item_available;
robot_service_sequence = serviceStatus.robot_service_sequence
}
});

connectedNode.executeCancellableLoop(new CancellableLoop() {
protected void setup() {}

protected void loop() throws InterruptedException
{
str_msgs.String padStatus = (str_msgs.String)pub_pad_status.newMessage();
turtlebot3_carrier.PadOrder padOrder = (turtlebot3_carrier.PadOrder)pub_pad_order.newMessage();

String str = "";

if (button_pressed[0] || button_pressed[1] || button_pressed[2])
{
jump = false;

if (button_pressed[0])
{
selected_item_num = 0;
str += "Burger was selected";

button_pressed[0] = false;
}
else if (button_pressed[1])
{
selected_item_num = 1;
str += "Coffee was selected";

button_pressed[1] = false;
}
else if (button_pressed[2])
{
selected_item_num = 2;
str += "Waffle was selected";

button_pressed[2] = false;
}
}
```

```
else
{
    selected_item_num = -1;
    str += "Sorry, selected item is now unavailable. Please choose another item.";
}

if (is_item_available[selected_item_num] != 1)
{
    str += ", but chosen item is currently unavailable.";
    jump = true;
}
else if (robot_service_sequence[robot_num] != 0)
{
    str += ", but your TurtleBot is currently on servicing";
    jump = true;
}
else if (item_num_chosen_by_pad[robot_num] != -1)
{
    str += ", but your TurtleBot is currently on servicing";
    jump = true;
}

padStatus.setData(str);
pub_pad_status.publish(padStatus);

if(!jump)
{
    padOrder.pad_number = robot_num;
    padOrder.item_number = selected_item_num;
    pub_pad_order.publish(padOrder);
}
}

Thread.sleep(1000L);
}
});
}
}
```

在整个源代码中， MainActivity类接收和使用ServicePad类的一个实例(instance)，但是除此之外的部分与一般的MainActivity类一样，所以省略了详细的解释，下面只了解与配送服务对应的部分。

将上传本例的平板电脑要控制的机器人的编号指定给robot_num变量，并将平板电脑上被选商品的编号初始化为“-1”。

```
private int robot_num = 0;  
private int selected_item_num = -1;
```

为了避免重复的订单，平板电脑必须在收到客户的订单之前与保存在服务核心中的订单状态同步。以下源代码的格式声明与在服务核心中记录订单状态的数组保持一致的格式声明。

```
private int[] item_num_chosen_by_pad = {-1, -1, -1};  
private int[] is_item_available = {1, 1, 1};  
private int[] robot_service_sequence = {0, 0, 0};
```

在MainActivity类中创建ServicePad类的实例时，按如下方式指定话题名称。在这个例子中，每对平板和机器人都被指定于同一组namespace，所以在每个话题名称之前要写入namespace中使用的名称才能进行通信。

```
public ServicePad(){  
    this.pub_pad_order_topic_name = "/tb3g/pad_order";  
    this.sub_service_status_topic_name = "/tb3g/service_status";  
    this.pub_pad_status_topic_name = "/tb3g/pad_status";  
}
```

指定在ROS中显示的节点名称。节点名称也应该标记group namespace。

```
public GraphName getDefaultNodeName() {  
    return GraphName.of("tb3g/pad");  
}
```

从服务核心接收如下服务状态：从各平板订购的物品编号、是否可以选择该物品、机器人的服务状态。

```
subscriber.addMessageListener(new MessageListener<turtlebot3_carrier.ServiceStatus>() {
    @Override
    public void onNewMessage(turtlebot3_carrier.ServiceStatus serviceStatus)
    {
        item_num_chosen_by_pad = serviceStatus.item_num_chosen_by_pad;
        is_item_available = serviceStatus.is_item_available;
        robot_service_sequence = serviceStatus.robot_service_sequence;
    }
});
```

这是根据与服务核心同步的整体服务情况来处理客户订单的部分。同样，判断订单是否重复，如果可以下单，则发布订单。图12-8和12-9分别显示了按下一个画有物品图片的按键后下单成功和下单失败的示例。

```
protected void loop() throws InterruptedException
{
    std_msgs.String padStatus = (std_msgs.String) pub_pad_status.newMessage();
    turtlebot3_carrier.PadOrder padOrder = (turtlebot3_carrier.PadOrder) pub_pad_order.newMessage();

    String str = "";

    if (button_pressed[0] || button_pressed[1] || button_pressed[2])
    {
        jump = false;

        if (button_pressed[0])
        {
            selected_item_num = 0;
            str += "Burger was selected";

            button_pressed[0] = false;
        }
        else if (button_pressed[1])
        {
            selected_item_num = 1;
            str += "Coffee was selected";

            button_pressed[1] = false;
        }
    }
}
```

```

else if (button_pressed[2])
{
    selected_item_num = 2;
    str += "Waffle was selected";

    button_pressed[2] = false;
}
else
{
    selected_item_num = -1;
    str += "Sorry, selected item is now unavailable. Please choose another item.";
}

if (is_item_available[selected_item_num] != 1)
{
    str += ", but chosen item is currently unavailable.";
    jump = true;
}
else if (robot_service_sequence[robot_num] != 0)
{
    str += ", but your TurtleBot is currently on servicing";
    jump = true;
}
else if (item_num_chosen_by_pad[robot_num] != -1)
{
    str += ", but your TurtleBot is currently on servicing";
    jump = true;
}

padStatus.setData(str);
pub_pad_status.publish(padStatus);

if(!jump)
{
    padOrder.pad_number = robot_num;
    padOrder.item_number = selected_item_num;
    pub_pad_order.publish(padOrder);
}
}

```

```
    Thread.sleep(1000L);  
}  
}
```

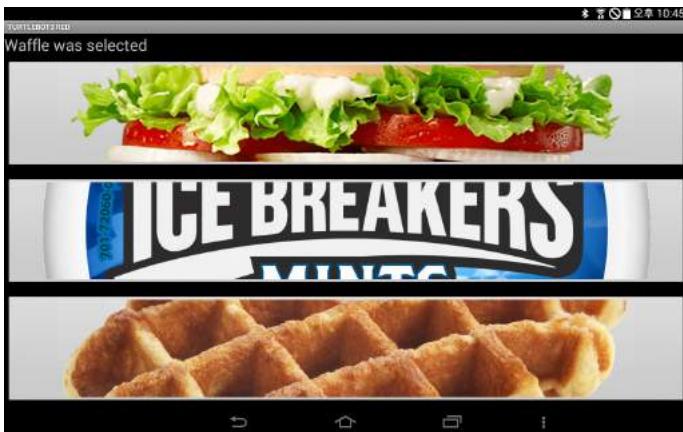


图 12-8 在平板上运行的菜单示例1（订单已成功接收的情况）

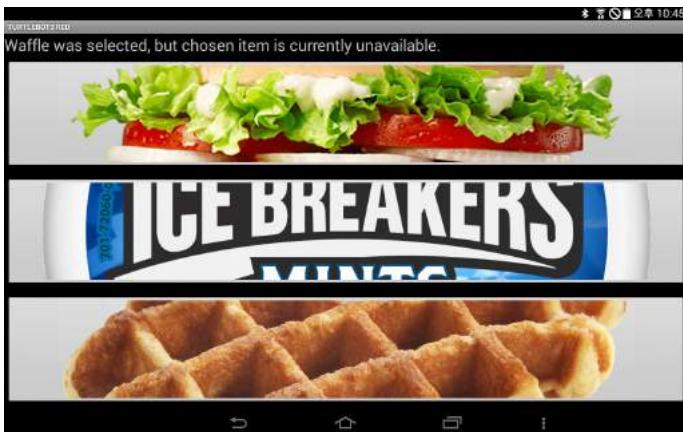


图 12-9 在平板上运行的菜单示例2（选择了无法下单的物品的情况）

12.2.5. 服务从节点

服务从节点管理的节点都与机器人的控制直接相关。例如，这个例子使用了 TurtleBot3 Carrier，并且在第10章和第11章中所描述的SLAM和导航运行时运行的节点

是主要节点。在这里，我们描述了为了开发TurtleBot3 Carrier³而改装的TurtleBot3的源代码。在这个例子中，主要使用图12-10所示的功能包。每个箭头代表一个子功能包。为了制作配送服务机器人，其中一些功能包需要进行修改。

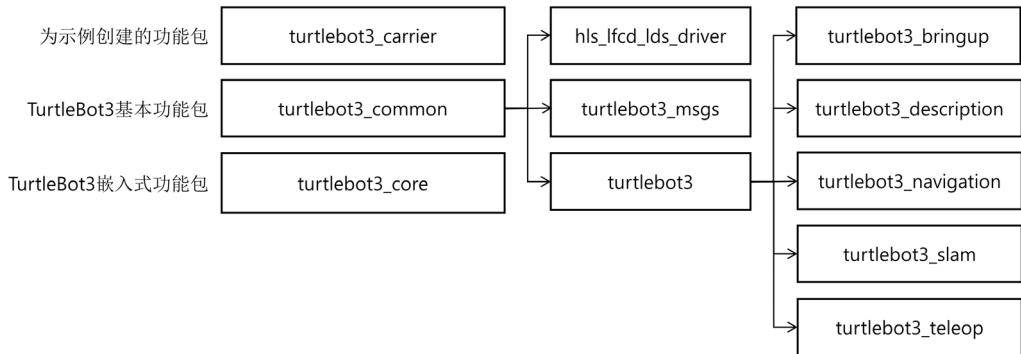


图 12-10 要使用的功能包列表

以下说明了为了制作配送服务机器人而修改的源代码。poll()函数处理附近物体之间的距离，而这个距离是通过LDS（HLS-LFCD2）的激光获得的。由于TurtleBot3 Carrier在LDS周围立了一些柱子，并且为了配送，被设计成有几层结构的形态，所以当LDS运转时，识别柱子，因此会对SLAM或导航的结果产生不利影响。因此，TurtleBot3 Carrier在检测到小于一定的距离的物体时，将与此物体相同角度的反射光的强度强制改为“0”。之后，在导航算法中，距离“0”的值被识别为“无对象”，因此柱子不影响SLAM或导航。

```
hld_lfcd_lds_driver/src/hlds_laser_publisher.cpp
void LFCDLaser::poll(sensor_msgs::LaserScan::Ptr scan)
{
    ...
    while (!shutting_down_ && !got_scan)
    {
        ...
        if(start_count == 0)
```

³ <http://emanual.robotis.com/docs/en/platform/turtlebot3/friends/#turtlebot3-friends-carrier>

```
{  
    if(raw_bytes[start_count] == 0xFA)  
    {  
        start_count = 1;  
    }  
}  
else if(start_count == 1)  
{  
    if(raw_bytes[start_count] == 0xA0)  
    {  
...省略...  
  
    //read data in sets of 6  
    for(uint16_t i = 0; i < raw_bytes.size(); i=i+42)  
    {  
        if(raw_bytes[i] == 0xFA && raw_bytes[i+1] == (0xA0 + i / 42))  
        {  
...省略...  
  
        for(uint16_t j = i+4; j < i+40; j=j+6)  
        {  
            index = (6*i)/42 + (j-6-i)/6;  
  
            // Four bytes per reading  
            uint8_t byte0 = raw_bytes[j];  
            uint8_t byte1 = raw_bytes[j+1];  
            uint8_t byte2 = raw_bytes[j+2];  
            uint8_t byte3 = raw_bytes[j+3];  
  
            // Remaining bits are the range in mm  
            uint16_t intensity = (byte1 << 8) + byte0;  
            uint16_t range = (byte3 << 8) + byte2;  
  
            scan->ranges[359-index] = range / 1000.0;  
            scan->intensities[359-index] = intensity;  
        }  
    }  
}
```

```

/// 添加部分开始 ///

for(uint16_t deg = 0; deg < 360; deg++)
{
    if(scan->ranges[deg] < 0.15)
    {
        scan->ranges[deg] = 0.0;
        scan->intensities[deg] = 0.0;
    }
}

/// 添加部分结束 ///

scan->time_increment = motor_speed/good_sets/1e8;
}
else
{
    start_count = 0;
}
}
}
}
}

```

turtlebot3_core是TurtleBot3使用的控制板OpenCR的专用固件，turtlebot3_motor_driver.cpp是直接控制TurtleBot3中使用的舵机的源程序。实际的服务机器人在装载物体的情况下移动，因此为了安全的搬运，需要适当的控制。因此，我们在下面的源代码中添加了不包含在turtlebot3_motor_driver.cpp源程序的Dynamixel的轮廓控制(profile control) 代码。这里，ADDR_X_PROFILE_ACCELERATION的值是108。关于舵机的更多信息，请参阅Dynamixel手册 (<http://emanual.robotis.com/>) 。

turtlebot3_core(for TurtleBot3 waffle)/turtlebot3_motor_driver.cpp

```

bool Turtlebot3MotorDriver::init(void)TurtleBot3载体
{
    ...省略...

    // Enable Dynamixel Torque
    setTorque(left_wheel_id_, true);
    setTorque(right_wheel_id_, true);
}

```

```
/// 添加部分开始 ///

// Set Dynamixel Profile Acceleration
setProfileAcceleration(left_wheel_id_, 15);
setProfileAcceleration(right_wheel_id_, 15);

/// 添加部分结束 ///

...省略...

return true;
}

bool Turtlebot3MotorDriver::setTorque(uint8_t id, bool onoff)
{
    ...省略...
}

bool Turtlebot3MotorDriver::setProfileAcceleration(uint8_t id, uint32_t value)
{
    uint8_t dxl_error = 0;
    int dxl_comm_result = COMM_TX_FAIL;

    dxl_comm_result = packetHandler_->write4ByteTxRx(portHandler_, id, ADDR_X_PROFILE_ACCELERATION,
value, &dxl_error);
    if(dxl_comm_result != COMM_SUCCESS)
    {
        packetHandler_->printTxRxResult(dxl_comm_result);
    }
    else if(dxl_error != 0)
    {
        packetHandler_->printRxPacketError(dxl_error);
    }
}
```

`turtlebot3_navigation.launch`启动在TurtleBot3中进行导航时运行的节点。如前所述，节点和ROS话题必须分组到同一组namespace中，以便在多个机器人中同时使用一种ROS功能包。在上面的launch代码中，我们将节点分组到叫做“tb3g”的namespace，并为了从未绑定在同一个namespace的其他节点接收消息，使用了remap功能。这种方法在无法修改源代码的话题名称时也可以使用。请注意，这种修改不仅应该在launch文件中完成，而且还应该在需要同样的分组（如RViz文件）的所有地方进行。

```
turtlebot3/turtlebot3_navigation/turtlebot3_navigation.launch
```

```
<launch>

<!-- 添加部分开始-->

<group ns="tb3g">
<remap from="/tf" to="/tb3g/tf"/>
<remap from="/tf_static" to="/tb3g/tf_static"/>

<!-- 添加部分结束-->
<arg name="model" default="waffle" doc="model type [burger, waffle, waffle_pi]"/>

...省略...

</node>

<!-- 添加部分开始 -->

</group>

<!-- 添加部分结束 -->

</launch>
```

如果已操作到这里，那么已经成功搭建了图12-11所示的系统。

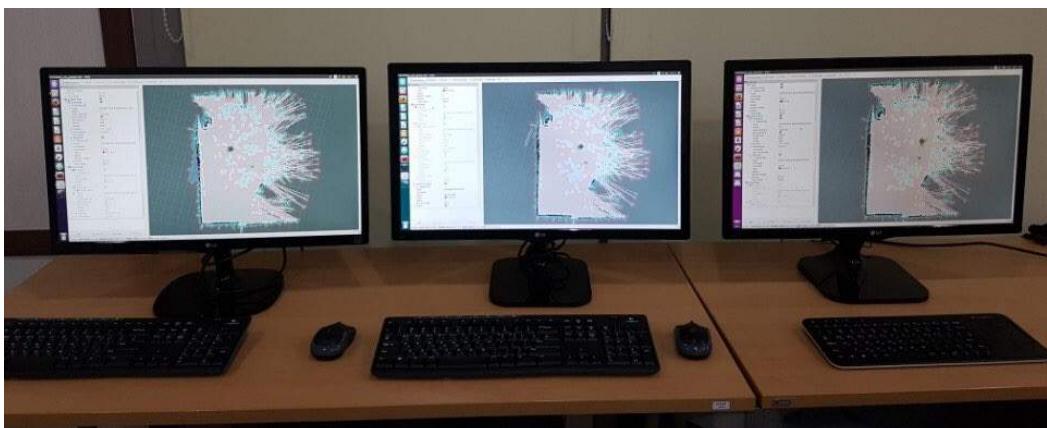


图 12-11 在各个计算机上运行的RViz显示的各机器人的导航场景

12.3. 用ROS Java进行Android平板PC编程

在前一节中，我们以图12-8中的平板上运行的菜单为例，说明了利用平板的订购。在本节中，我将在Linux上安装Android Studio IDE⁴，并构建ROS Java开发环境。然后，将介绍一个简单的ROS Java示例。

下面介绍Android Studio IDE的安装方法和ROS Java环境设置所需的功能包的安装方法。ROS Java是指以Java语言运行的ROS客户端库。我们先设置运行Java所需的选项。需要的功能包是Java SE Development Kit (JDK)，您需要指定其他的选项，如运行位置等。本书介绍了如何下载JDK 8，但在JDK版本更新后，您应该修改它。

```
$ sudo apt-get install openjdk-8-jdk
$ echo export PATH=${PATH}:/opt/android-sdk/tools:/opt/android-sdk/platform-tools:/opt/android-studio/bin >> ~/.bashrc
$ echo export ANDROID_HOME=/opt/android-sdk >> ~/.bashrc
$ source ~/.bashrc
```

⁴ <https://developer.android.com/studio/index.html>

以下命令下载用于ROS Java中的构建所需的工具。之后，安装并构建一个包含ROS Java系统和示例的功能包。这里的android_core目录是和前面出现的catkin_ws目录具有同样意义的目录。

```
$ sudo apt-get install ros-kinetic-rosjava-build-tools  
  
$ mkdir -p ~/android_core  
$ wstool init -j4 ~/android_core/src https://raw.githubusercontent.com/rosjava/rosjava/kinetic/  
    android_core.rosinstall  
$ source /opt/ros/kinetic/setup.bash  
$ cd ~/android_core  
$ catkin_make
```

下面说明如何安装Android Studio IDE。为了避免混淆，下面说明时使用与ROS Android中说明的目录和文件名一样的名称。这些功能包是在Android Studio IDE中安装mksdcard所需的附加功能包，其中mksdcard是利用SD卡实现Virtual Device的功能。如果不安装，mksdcard功能将无法安装。

```
$ sudo apt-get install lib32z1 lib32ncurses5 lib32stdc++6
```

本书中将在ROS Android⁵推荐的/opt目录安装Android Studio IDE和SDK。为了安装到/opt目录，需要将/opt的用户权限更改为可写。

```
$ sudo chown -R $USER:$USER /opt
```

Android Studio IDE的安装文件可以从<https://developer.android.com/studio/index.html#download>获取。下载安装文件后，将其解压到/opt/android-studio。解压后，目录的配置如图12-12所示。

⁵ <http://wiki.ros.org/android>



图 12-12 下载的Android Studio IDE已正确解压缩

解压完成后，输入以下命令开始IDE安装。

```
$ /opt/android-studio/bin/studio.sh
```

当出现如图12-13所示的窗口时，单击“无导入运行”继续进行“自定义安装”。

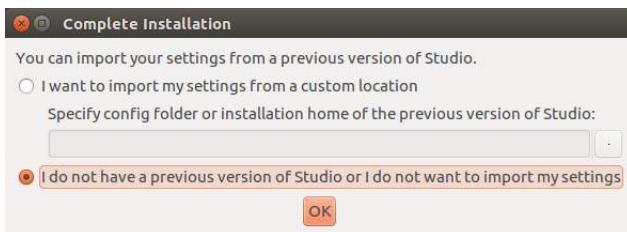


图 12-13 在安装过程中出现的第一个窗口

之后，您将看到如图12-14所示的安装Android SDK的窗口。请注意，您需要将安装位置设置为/opt/android-sdk。如果您没有“android-sdk”目录，请在该位置创建并继续。

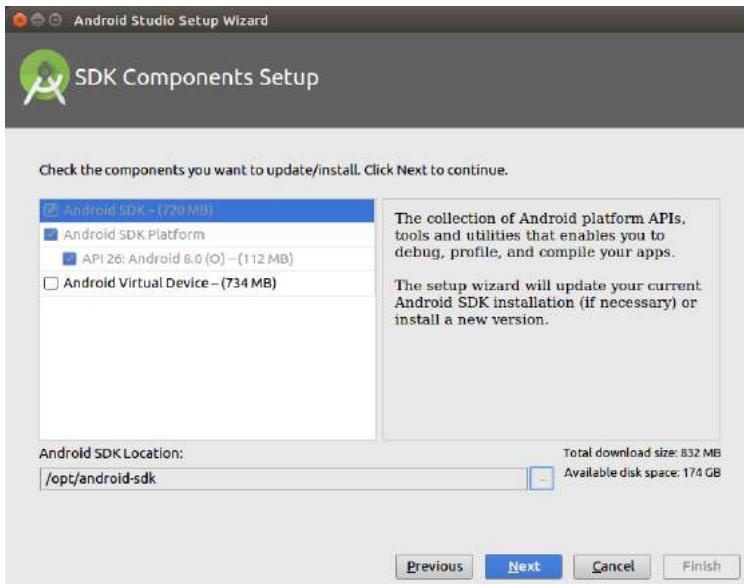


图 12-14 Android SDK安装画面

如果安装正常完成，则按照图12-15所示完成安装。

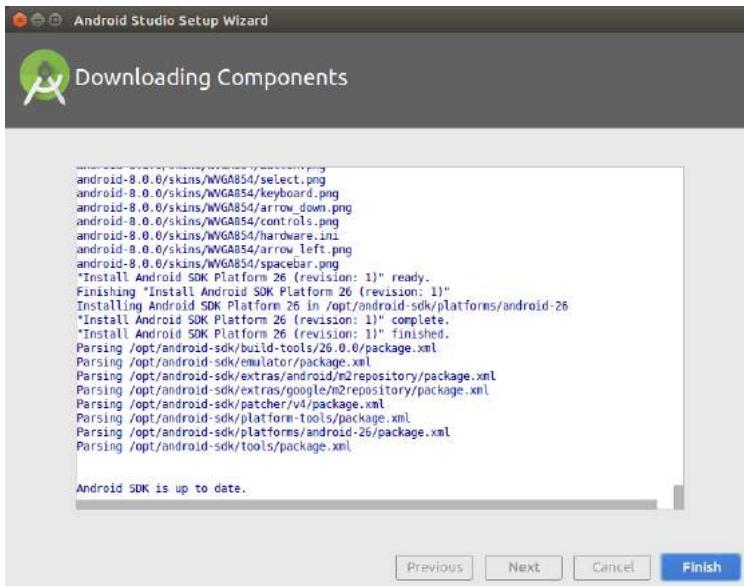


图 12-15 安装正常结束的画面

当Android Studio IDE正常安装时，将出现“Welcome to Android Studio”窗口，如图12-16所示。



图 12-16 Welcome to Android Studio

此时，请通过Configure→SDK Manager更新Android SDK。在图12-17中，可用的SDK包括10（Gingerbread）、13（Honeycomb）、15（Ice cream）、18（Jellybean）。

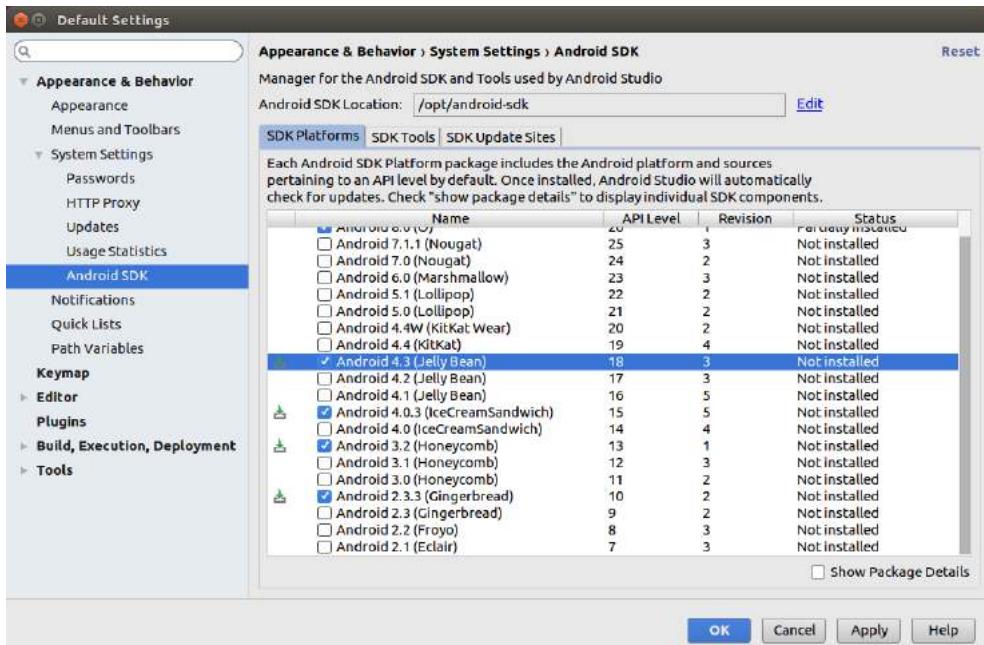


图 12-17 Android SDK设置

配置和安装完成后，从窗口中点击Open an existing Android Studio project，import之前安装的android_core，如图12-18所示。当点击OK后，出现如图12-19所示的IDE窗口并开始构建import的源代码。

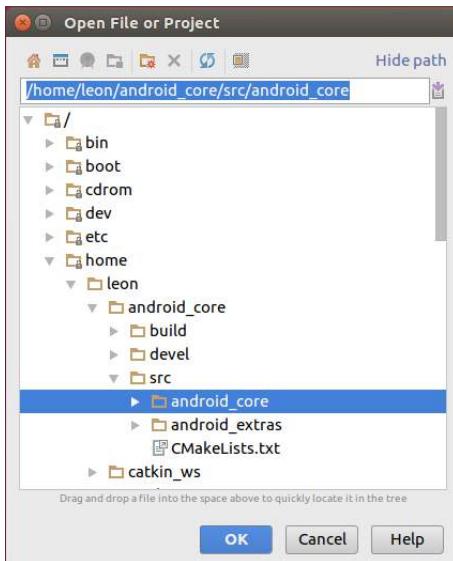


图 12-18 Project import

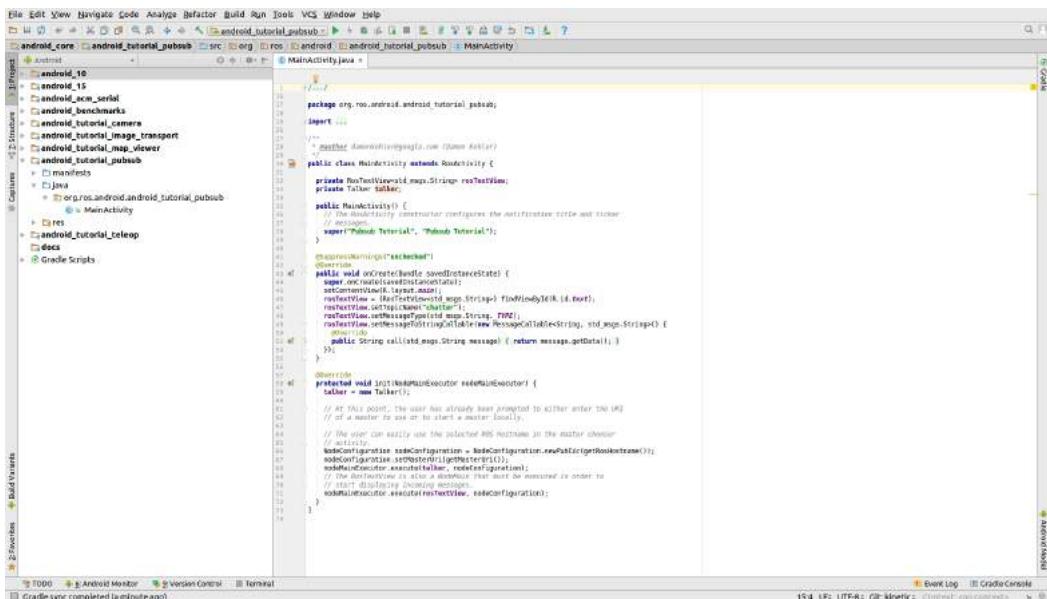


图 12-19 Android Studio IDE的import画面

接下来，我们运行android_tutorial_pubsub示例。在窗口顶部的project选择窗口中选择android_tutorial_pubsub，然后点击旁边的play键，则会查找执行程序的终端。选择适当的终端，如图12-20所示，然后按OK。

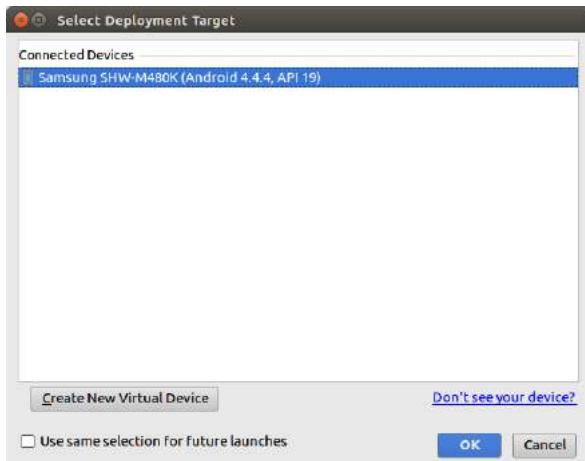


图 12-20 终端选择窗口

将程序成功安装到终端上时，需要将ROS主节点所在的PC（运行roscore的PC）的IP输入到终端，如图12-21所示。输入正确的IP，然后按Connect。

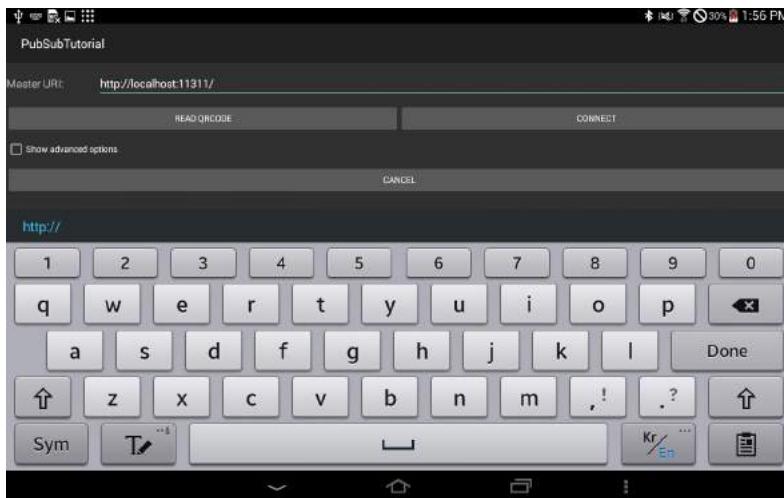


图 12-21 ROS IP设置窗口

当android_tutorial_pubsub运行时，终端会发布std_msgs::String类型的“Hello world! n”。现在，让我们在您的计算机上订阅此终端发布的字符串吧。如果您使用上述的“rostopic echo”命令查看“/chatter”话题，则可以看到该话题正在发布，如下所示。

```
$ rostopic echo /chatter
data: Hello world! 96
---
data: Hello world! 97
---
data: Hello world! 98
---
data: Hello world! 99
---
data: Hello world! 100
---
data: Hello world! 101
---
```

我们在上一章中介绍了应用SLAM和导航的服务机器人。如本章所述，创建一个在您周围可见的服务机器人并不困难。在本章的结尾，我衷心希望读者可以通过本书可以开发出更好的机器人。

第13章

机械手臂

13.1. 机械手臂介绍

机械手臂（Manipulator）是为了在工厂里执行简单重复任务而设计的机器人。它的目的是取代危险的工作或取代重复的任务，最近有许多有关机械手臂和人的协作的研究。^{1 2}

随着人机交互（Human Robot Interaction, HRI）³的研究活跃起来，机械手臂不仅应用到了工厂，还与多种领域（Media Arts⁴、VR⁵等）结合，为大众带来了新的体验。数字舵机和3D打印技术的结合正在提高机械手臂对公众的接近度，这给了制造商和教育行业巨大的期待。^{6 7 8}一方面，机械手臂和人工智能的结合使很多人带来了大规模失业的恐惧。^{9 10}但是，机械手臂长期以来一直是使社会富饶的工具之一，当今也在许多不同的领域帮助人们。^{11 12}未来如果机器人的发展能够渗透到我们的生活中，而不会摆脱其本质，那么像扫地机器人一样，机器人将成为我们生活的一部分。

今后，我们将介绍机械手臂的结构和ROS中提供的机械手臂的库。ROBOTIS的OpenManipulator是支持ROS的机械手臂之一，其优点是能够使用Dynamixel和3D打印部件，因此可以低成本轻松制作。我将以本机械手臂为例，介绍可以与ROS一起使用的Gazebo 3D仿真器，还会介绍用于机械手臂的集成库MoveIt!，并且讲解他们的使用方法。最后，我将讨论如何配置和控制实际的平台，以及OpenManipulator与TurtleBot3 Waffle、Waffle Pi之间的兼容性。

13.1.1. 机械手臂的结构和控制

机械手臂的基本结构由基座（Base）、连杆（Link）、关节（Joint）和末端执行器（End-effector）组成，如图13-1所示。

1 <https://www.automationworld.com/inside-human-robot-collaboration-trend>

2 <https://www.kuka.com/en-us/technologies/human-robot-collaboration>

3 https://en.wikipedia.org/wiki/Human%E2%80%93robot_interaction

4 <https://youtu.be/lx6jcybgDFo>

5 <http://www.asiae.co.kr/news/view.htm?idxno=2016100416325879220>

6 <http://www.littlearmrobot.com/>

7 <https://niryo.com/products/>

8 <http://www.ufactory.cc/#/en/>

9 <http://time.com/4742543/robots-jobs-machines-work/>

10 <http://adage.com/article/digitalnext/5-jobs-robots/308094/>

11 <https://www.bostonglobe.com/magazine/2015/09/24/this-robot-going-take-your-job/paj3zwznSXMSvQiQ8pdBjK/story.html>

12 <https://www.automationworld.com/article/abb-unveils-future-human-robot-collaboration-yumi>

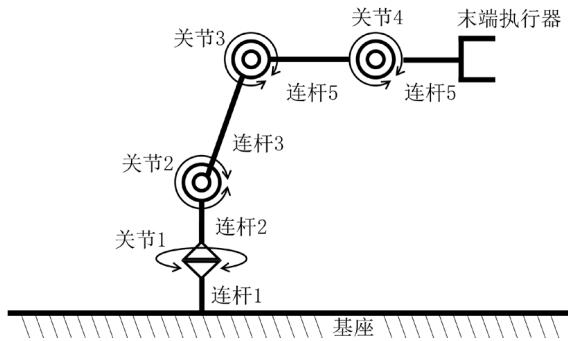


图 13-1 机械手臂的基本结构

机械手臂一般是固定一端，而固定部分称为基座。因为机械手臂的长度和端部的移动速度与施加到基座的力的大小成比例，所以基座在机械手臂的整个结构中是面积最大且最坚固的部分。基座不仅是一个固定的物体，它也可以是一个像移动机器人那样运动的物体，因此可以作为机械手臂自由度的补充。以“基座”作为基础而制作出来的机械手臂还由连杆和关节的级联组成。连杆通常有一个关节，但有时也有不止一个关节。关节代表旋转轴，主要由电机组成。通过这个电机的转动，关节产生连杆的运动。关节根据其运动方式可以分为：旋转关节（Revolute）、平移关节（Prismatic）、螺丝关节（Screw）、圆筒关节（Cylindrical）、混合关节（Universal）和球形关节（Spherical）等。近年来，出现了很多使用液压而非电机的关节，而且学界正在积极进行关于可以代替电机的新型关节的研究。

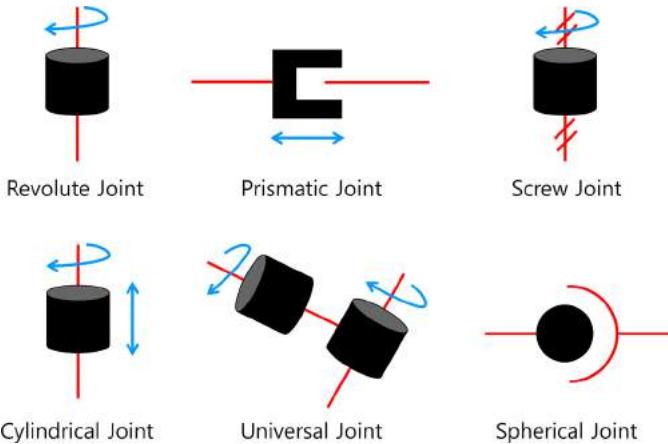


图 13-2 关节 (joint) 的类型

基座上有一连串的连杆和关节，最终端是一个“末端执行器”。由于机械手臂的固有目的是抓取并搬运物体，因此其“末端执行器”往往是一个抓手（Gripper）。如图13-3所示，为了实现抓取不同形状物体的任务，研究者们尝试着多种形态的末端执行器。



图 13-3 抓手的类型（从左到右 ROBOTIQ、ROBOTIS、Cornell University）

控制机械手臂的方法可分为关节空间控制（Joint Space Control）和任务空间控制（Task Space Control）。

关节空间控制是通过输入每个关节的旋转角度来输出机械手臂坐标值的方法，如图13-4所示。根据各关节的旋转程度而变化的末端坐标值（X，Y，Z， θ ， ϕ 和 ψ ）可以通过正向运动学获得。

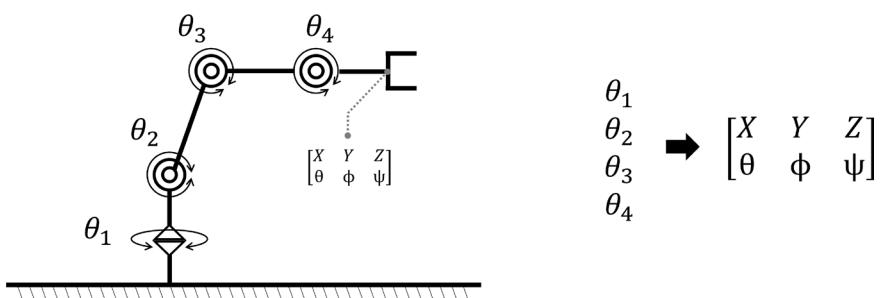


图 13-4 正向运动学

如图13-5所示，任务空间控制是输入机械手臂的末端的坐标值，以此获得各关节的角度位置，其输入和输出与关节空间控制正好相反。工作空间中的物体姿态（Pose）包括其位置（Position）和方向（Orientation）。我们生活在一个三维的世界，因此可以用X、Y、Z轴来表示，而方向可以表示为 θ （roll）、 ϕ （pitch）、 ψ （yaw）。以桌子上的杯子（假设该坐标系的原点是杯子的中心）为例，即使它的位置不变，但可以通过使其躺

下或改变它的手把的方向来改变杯子在三维空间中的姿态。换句话说，如果以数学语言描述，则意味着有6个未知数，所以如果有6个方程就可以找到唯一的解。根据机械手臂的自由度特点，当有6个关节时，才可以把桌子上的杯子以任何角度举到任何可能的位置。但是，并不是所有的机械手臂都需要六个以上的自由度。根据机械手臂使用的目的和环境来调整自由度会更有效率。通过逆运动学原理，可以根据机械手臂末端的坐标值获得每个关节的角度。

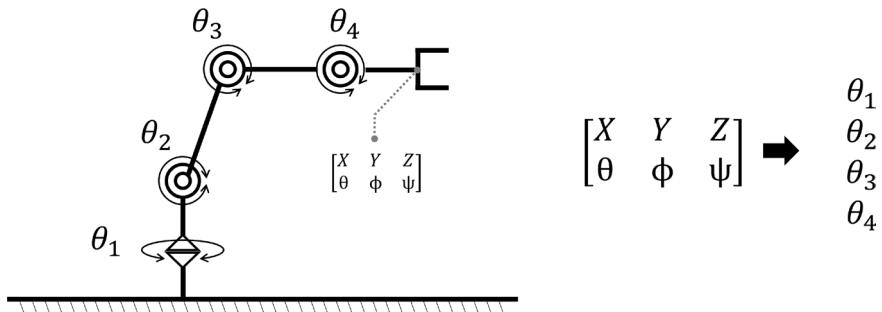


图 13-5 逆运动学

13.1.2. 机械手臂和ROS

ROS通过开源的可扩展性和灵活性吸引了许多用户。随着越来越多的用户使用ROS，出现了更多的支持ROS的平台，也出现了收集和销售这些平台的公司¹³。此外，个人开发者为了研究或爱好的目的而创建的平台也有在ROS官方功能包中注册，并被介绍给许多用户。ROS Robots¹⁴支持大约180个ROS平台。

典型的有ROS-INDUSTRIAL¹⁵支持的ABB的工业机械手臂¹⁶，而被广泛用于研究的Kinova的JACO¹⁷也支持ROS。作为韩国公司，ROBOTIS的MANIPULATOR-H¹⁸支持ROS。各机械手臂参见图13-6。

¹³ <https://www.roscomponents.com/en/>

¹⁴ <http://robots.ros.org/>

¹⁵ <http://rosindustrial.org/>

¹⁶ <http://wiki.ros.org/abb/>

¹⁷ <http://wiki.ros.org/Robots/JACO/>

¹⁸ <http://wiki.ros.org/ROBOTIS-MANIPULATOR-H/>



图 13-6 支持ROS的各种机械手臂（从左开始，ABB，ROBOTIS，Kinova）

13.2. OpenManipulator建模和仿真

ROS为机械手臂提供了有用的工具。

第一种工具是可以通过可扩展标记语言（XML）很方便地创建一个统一机器人描述格式（URDF¹⁹，Unified Robot Description Format）文件，该文件可以在ROS的机器人建模可视化工具RViz（ROS Visualization）中加载并使用。

第二种工具是3D仿真器Gazebo²⁰，可以仿真实际的操作环境。与URDF类似，Gazebo仿真环境也可以使用XML和仿真描述格式（SDF，Simulation Description Format²¹）文件轻松创建。Gazebo还支持ROS-CONTROL²²和plugin²³功能来控制机器人和各种传感器。

第三种是MoveIt!²⁴，一个用于机械手臂的集成库。MoveIt!提供Kinematics and Dynamics Library（KDL）²⁵和The Open Motion Planning Library（OMPL）²⁶等开源库。它是功能强大的机械手臂工具，可以查看机械手臂的多种功能，比如碰撞计算、运动规划和Pick and Place演示，等。

让我们看看如何使用上面的三个工具，并用示例代码来实现它。

¹⁹ <http://wiki.ros.org/urdf>

²⁰ <http://gazebosim.org/>

²¹ <http://sdformat.org/>

²² http://wiki.ros.org/ros_control

²³ http://gazebosim.org/tutorials?tut=ros_gzplugins

²⁴ <http://moveit.ros.org/>

²⁵ <http://www.orocos.org/kdl>

²⁶ <http://ompl.kavrakilab.org/>

13.2.1. OpenManipulator

OpenManipulator是由ROBOTIS开发的基于开源软件和硬件的机械手臂。OpenManipulator支持Dynamixel X系列²⁷，您可以通过选择您所需的规格的舵机来制作机器人。而且，因为它由基本连接件和3D打印连接件组成，因此可以根据用户的环境或目的制作一种新型的机械手臂。由于这些特点，除了四关节机械手臂之外，还将提供SCARA、Planar、Delta等各种形状和功能的机械手臂。OpenManipulator支持ROS的同时还支持OpenCR²⁸、Arduino IDE²⁹和Processing³⁰。

本章中将要使用的OpenManipulator Chain具有最基本的机械手臂形式，并且末端执行器具有由3D打印连接件构成的线性抓手形状。OpenManipulator Chain设计文件都已在Onshape³¹上公开，而源代码可以从ROBOTIS的Github³²下载。源代码同时支持ROS、Arduino和Processing，而且源代码包括OpenManipulator Chain的MoveIt!功能包和Gazebo功能包。此外，OpenManipulator Chain与TurtleBot3 Waffle及Waffle Pi在机械方面兼容，所以可以弥补自由度的不足。

下面我们将看OpenManipulator Chain的源代码、URDF、Gazebo和MoveIt!。以下是使用上述三种工具所需的ROS功能包。我们预先安装它们。

```
$ sudo apt-get install ros-kinetic-ros-controllers ros-kinetic-gazebo* ros-kinetic-moveit* ros-kinetic-dynamixel-sdk ros-kinetic-dynamixel-workbench-toolbox ros-kinetic-robotis-math ros-kinetic-industrial-core
```

13.2.2. 机械手臂建模

为了在虚拟空间中仿真机械手臂，我们先来了解一下各个组件的仿真方法。在查看OpenManipulator Chain中的URDF之前，让我们为由三个关节和四个连杆组成的机械手臂创建一个简单的URDF。

²⁷ http://www.robotis.com/index/product.php?cate_code=131810

²⁸ <http://emanual.robotis.com/docs/en/parts/controller/opencr10/>

²⁹ <https://www.arduino.cc/en/main/software>

³⁰ <https://processing.org/>

³¹ <https://goo.gl/NsqJMu>

³² https://github.com/ROBOTIS-GIT/open_manipulator

首先，按如下所示创建testbot_description功能包，然后创建urdf目录。然后使用编辑器创建一个testbot.urdf文件，并输入下面的URDF例程。

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg testbot_description urdf  
$ cd testbot_description  
$ mkdir urdf  
$ cd urdf  
$ gedit testbot.urdf
```

testbot_description/urdf/testbot.urdf

```
<?xml version="1.0" ?>  
<robot name="testbot">  
  
<material name="black">  
  <color rgba="0.0 0.0 0.0 1.0"/>  
</material>  
<material name="orange">  
  <color rgba="1.0 0.4 0.0 1.0"/>  
</material>  
  
<link name="base"/>  
<joint name="fixed" type="fixed">  
  <parent link="base"/>  
  <child link="link1"/>  
</joint>  
  
<link name="link1">  
  <collision>  
    <origin xyz="0 0 0.25" rpy="0 0 0"/>  
    <geometry>  
      <box size="0.1 0.1 0.5"/>  
    </geometry>  
  </collision>  
  <visual>  
    <origin xyz="0 0 0.25" rpy="0 0 0"/>  
    <geometry>  
      <box size="0.1 0.1 0.5"/>
```

```
</geometry>
<material name="black"/>
</visual>
<inertial>
<origin xyz="0 0 0.25" rpy="0 0 0"/>
<mass value="1"/>
<inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
</inertial>
</link>

<joint name="joint1" type="revolute">
<parent link="link1"/>
<child link="link2"/>
<origin xyz="0 0 0.5" rpy="0 0 0"/>
<axis xyz="0 0 1"/>
<limit effort="30" lower="-2.617" upper="2.617" velocity="1.571"/>
</joint>

<link name="link2">
<collision>
<origin xyz="0 0 0.25" rpy="0 0 0"/>
<geometry>
<box size="0.1 0.1 0.5"/>
</geometry>
</collision>
<visual>
<origin xyz="0 0 0.25" rpy="0 0 0"/>
<geometry>
<box size="0.1 0.1 0.5"/>
</geometry>
<material name="orange"/>
</visual>
<inertial>
<origin xyz="0 0 0.25" rpy="0 0 0"/>
<mass value="1"/>
<inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
</inertial>
</link>
```

```

<joint name="joint2" type="revolute">
  <parent link="link2"/>
  <child link="link3"/>
  <origin xyz="0 0 0.5" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <limit effort="30" lower="-2.617" upper="2.617" velocity="1.571"/>
</joint>

<link name="link3">
  <collision>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <geometry>
      <box size="0.1 0.1 1"/>
    </geometry>
  </collision>
  <visual>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <geometry>
      <box size="0.1 0.1 1"/>
    </geometry>
    <material name="black"/>
  </visual>
  <inertial>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
  </inertial>
</link>

<joint name="joint3" type="revolute">
  <parent link="link3"/>
  <child link="link4"/>
  <origin xyz="0 0 1.0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <limit effort="30" lower="-2.617" upper="2.617" velocity="1.571"/>
</joint>

<link name="link4">
  <collision>

```

```
<origin xyz="0 0 0.25" rpy="0 0 0"/>
<geometry>
  <box size="0.1 0.1 0.5"/>
</geometry>
</collision>
<visual>
  <origin xyz="0 0 0.25" rpy="0 0 0"/>
  <geometry>
    <box size="0.1 0.1 0.5"/>
  </geometry>
  <material name="orange"/>
</visual>
<inertial>
  <origin xyz="0 0 0.25" rpy="0 0 0"/>
  <mass value="1"/>
  <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
</inertial>
</link>

</robot>
```

URDF使用XML标签来描述机器人的每个组件。以URDF形式先描述机器人的名称、基座（在URDF中将基座看作一个固定的连杆）的名称和类型、连接到基座的连杆，之后逐一说明连杆和关节的内容。连杆描述连杆的名称、大小、重量和惯性等。关节描述每个关节的名称、类型和连接的连杆。并且可以很容易地设置机器人的动力学元素、可视化和碰撞模型。URDF是以`<robot>`标签来开始，详细内容中通常会反复交替出现`<link>`标签和`<joint>`标签，这两种标签都用于定义机器人的组件-连杆和关节。其中，为了与ROS-Control共用，通常还包括用于设置关节和舵机之间的关系的`<transmission>`标签。接着，让我们仔细看看我们创建的testbot.urdf。

`material`标签描述连杆的颜色和纹理等信息。在下面的例子中，我们定义了两种材质，黑色和橙色，以区分每个连杆。颜色是利用`color`标签，可以在`rgba`选项后面输入对应于红色、绿色和蓝色的三个0.0到1.0之间的一个数字来分别设置。最后一个数字的透明度（alpha）值为0.0到1.0，值为1.0意味着没有透明度。

```
<material name=" black" >
<color rgba="0.0 0.0 0.0 1.0"/>
</material>
<material name="orange">
<color rgba="1.0 0.4 0.0 1.0"/>
</material>
```

机械手臂的第一种组件，基座在URDF中以连杆表示。基座通过关节连接到第一个连杆，这个关节是固定的，位于原点（0,0,0）。为了进行更多关于<link>标签的详细描述，我们先来看第一个连杆（link1）标签。

```
<link name=" base" />

<joint name="fixed" type="fixed">
<parent link="base"/>
<child link="link1"/>
</joint>

<link name="link1">
<collision>
<origin xyz="0 0 0.25" rpy="0 0 0"/>
<geometry>
<box size="0.1 0.1 0.5"/>
</geometry>
</collision>
<visual>
<origin xyz="0 0 0.25" rpy="0 0 0"/>
<geometry>
<box size="0.1 0.1 0.5"/>
</geometry>
<material name="black"/>
</visual>
<inertial>
<origin xyz="0 0 0.25" rpy="0 0 0"/>
<mass value="1"/>
<inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
</inertial>
</link>
```

如上面的link1例子，URDF <link>标签由碰撞（collision）、视觉（visual）和惯性（inertial）标签（见图13-7）组成。collision标签允许您输入表示连杆的外形范围的几何信息。origin写外形范围的中心坐标。geometry写以origin坐标为中心的外形范围的形状和大小。例如，长方体类型的外形范围是长、宽和高的值。除长方体形式以外，还有圆柱型和球型，他们的输入内容各不相同。在visual标签中写下实际的形状。origin和geometry与collision标签相同。而且也可以在这里输入CAD文件，如STL和DAE。collision标签可以在CAD模型中使用，但仅适用于ODE或Bullet等部分物理引擎，不支持DART和Simbody等。在inertial标签中，输入连杆的重量（质量单位为kg）和惯性矩（惯性矩单位为 $\text{kg}\cdot\text{m}^2$ ）。这种惯性信息可以通过设计软件、实际测量和计算得到，用于动力学仿真。

testbot.urdf示例中描述的link1、link2和link4的原点与各自的上部关节（分别为fixed、joint1和joint3）的原点相距0.25m，而这三个连杆的外形是以这个移动的原点为中心的长宽各0.1m，高0.5m（z轴正方向0.25m，z轴负方向0.25m）的长方体。link3的原点与上部关节（joint2）的原点相距0.5m，且link3的外形是以这个移动的原点为中心的长宽各0.1m，高1m的长方体。

在第一次遇到URDF时对于相对坐标变换的理解是相当困难的。一边直观地查看图形，一边理解各设定值会容易得多。为了更好地理解，可以通过图13-12观察各轴的相对坐标变换在RViz上是如何表现的，并通过修改设定值观察RViz上的变化。笔者建议读者务必尝试一下。

连杆（link）标签的属性	
<link ³³ >:	连杆的可视化、碰撞和惯性信息设置
<collision>:	设置连杆的碰撞计算的信息
<visual>:	设置连杆的可视化信息
<inertial>:	设置连杆的惯性信息
<mass>:	连杆重量（单位：kg）的设置
<inertia>:	惯性张量（Inertia tensor ³⁴ ）设置
<origin>:	设置相对于连杆相对坐标系的移动和旋转

³³ <http://wiki.ros.org/urdf/XML/link>

³⁴ https://en.wikipedia.org/wiki/Moment_of_inertia

<geometry>: 输入模型的形状。提供box、cylinder、sphere等形态，也可以导入COLLADA (.dae)、STL (.stl) 格式的设计文件。在<collision>标签中，可以指定为简单的形态来减少计算时间

<material>: 设置连杆的颜色和纹理

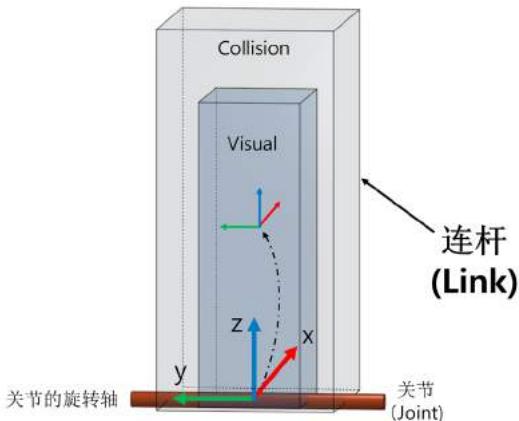


图 13-7 连杆的建模参数

接下来，我们来看看连接相邻连杆的关节（joint）标签。关节标签描述了关节的特征，如图13-8所示。具体来说它描述关节的名称和类型，如revolute（旋转运动型）、prismatic(平移运动型)、continuous（连续旋转的轮）、fixed（固定型）、floating（非固定）和planar（在与轴垂直的平面移动的形态）。它还描述连接的两个连杆的名称、关节的位置、旋转和平移运动的基准轴的动作限制。连接的连杆根据位置称为父连杆（parent link）和子连杆（child link），父连杆通常是靠近基座的连杆。以下示例显示了joint2关节的设置。

```
<joint name="joint2" type="revolute">
  <parent link="link2"/>
  <child link="link3"/>
  <origin xyz="0 0 0.5" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <limit effort="30" lower="-2.617" upper="2.617" velocity="1.571"/>
</joint>
```

关节 (joint) 标签的属性

- <joint³⁵>: 与连杆的关系和关节类型的设置
- <parent>: 关节的父连杆
- <child>: 关节的子连杆
- <origin>: 将父连杆坐标系转换为子连杆坐标系
- <axis>: 设置旋转轴
- <limit>: 设置关节的速度、力和半径（仅当关节是revolute或prismatic时）

为了更深入理解，让我们仔细了解一下。joint2的类型（type）设置为一种运动型关节revolute。父连杆（parent link）设置为link2，而子连杆设置为link3。另外，origin中以上级关节joint1的坐标系为原点，指定joint2的坐标系的相对位置姿态（位置+方向）。例如，joint2坐标系的原点是joint1坐标系朝z轴方向相距0.5m的位置。下面是轴设置。在轴设置axis中，如果是旋转型关节，则写入旋转轴的方向，如果是平移型关节，则写入运动方向。在joint2的情况下，设定为在y轴方向上旋转的关节。limit设定了关节运动的极限。属性包括给予关节的力（effort，单位N），最小、最大角度（下限，上限，以弧度为单位）和速度（以弧度/秒）等物理量的限制值。

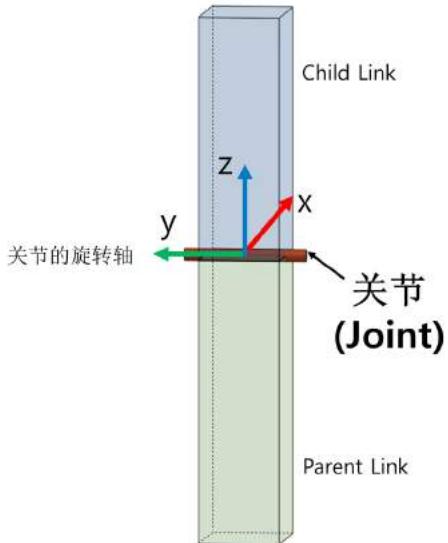


图 13-8 关节的建模参数

³⁵ <http://wiki.ros.org/urdf/XML/joint>

完成建模后，我们来检查每个连杆和关节，看它们是否逻辑正确。在ROS中，可以用check_urdf命令来检查已创建的URDF的语法错误以及每个连杆的连接关系，如下例所示。如果在语法和逻辑上没有问题，则可以看到连杆1、2、3和4正常连接，如下所示。

```
$ check_urdf testbot.urdf
robot name is: testbot
----- Successfully Parsed XML -----
root Link: base has 1 child(ren)
    child(1): link1
        child(1): link2
            child(1): link3
                child(1): link4
```

接下来，让我们来用关系图表示urdf_to_graphiz程序创建的模型吧。如果您运行urdf_to_graphiz，则会创建一个.gv文件和一个.pdf文件。如果您用PDF阅读器，可以一目了然地看到连杆与关节之间的关系，以及每个关节之间的相对坐标转换，如图13-9所示。

```
$ urdf_to_graphiz testbot.urdf
Created file testbot.gv
Created file testbot.pdf
```

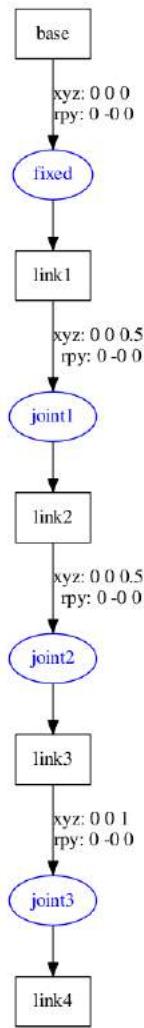


图 13-9 URDF连杆与关节的关系

用check_urdf和urdf_to_graphviz是检查模型的连杆关系的最快的方法。最后，让我们使用RViz检查机器人模型。为此，请转至testbot_description功能包目录并创建一个testbot.launch文件，如以下示例所示。

```

$ cd ~/catkin_ws/src/testbot_description
$ mkdir launch
$ cd launch
$ gedit testbot.launch

```

```
<launch>
<arg name="model" default="$(find testbot_description)/urdf/testbot.urdf" />
<arg name="gui" default="True" />
<param name="robot_description" textfile="$(arg model)" />
<param name="use_gui" value="$(arg gui)"/>
<node pkg="joint_state_publisher" type="joint_state_publisher" name="joint_state_publisher"/>
<node pkg="robot_state_publisher" type="state_publisher" name="robot_state_publisher"/>
</launch>
```

Launch文件由包含URDF的参数、joint_state_publisher³⁶ 节点和robot_state_publisher³⁷ 节点组成。joint_state_publisher节点通过sensor_msgs/JointState消息的形式发布URDF形式的机器人的关节状态，并提供一个GUI工具来给关节提供命令。robot_state_publisher节点以tf³⁸消息的形式发布forward kinematics的结果，这个结果是由URDF中设置的机器人信息和sensor_msgs/JointState话题信息来计算得出的。（参见图13-10）。

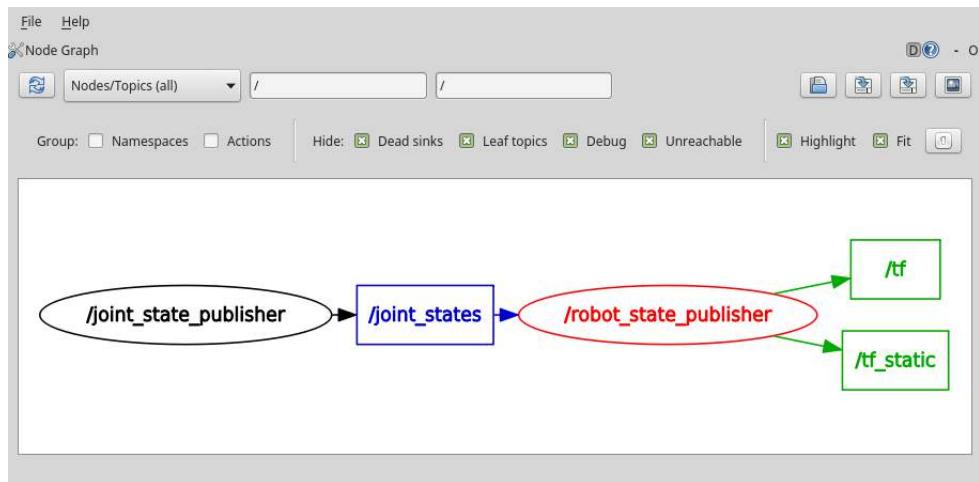


图 13-10 joint_state_publisher节点和robot_state_publisher节点中的话题

³⁶ http://wiki.ros.org/joint_state_publisher

³⁷ http://wiki.ros.org/robot_state_publisher

³⁸ <http://wiki.ros.org/tf>

一切准备就绪后，运行testbot.launch和RViz，如下所示：

```
$ roslaunch testbot_description testbot.launch  
$ rviz
```

运行launch文件时会运行joint_state_publisher节点的GUI，如图13-11所示。在这里您可以调整joint1、2和3的关节值。此外，运行RViz后将[Fixed Frame]选为“base”，点击左下方的Add按钮，添加“RobotModel”显示屏，就可以看到每个关节和连杆的形状，如图13-12的上图所示。如果添加“TF”显示屏，并将机器人模型的“Alpha”值修改为约0.3，则可以查看每个连杆的形状以及关节之间的关系，如图13-12的下图所示。

如果调整了joint_state_publisher节点的GUI滑动条，则可以看到RViz上的虚拟机器人的运动，如图13-13所示。相关的源代码可以在Github存储库中找到：

- https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/testbot_description

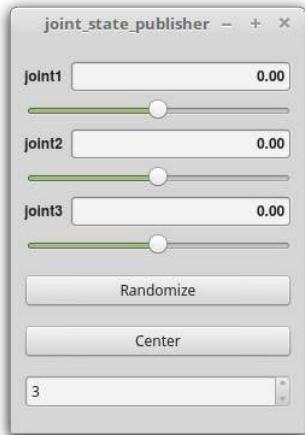


图 13-11 Joint State Publisher的GUI

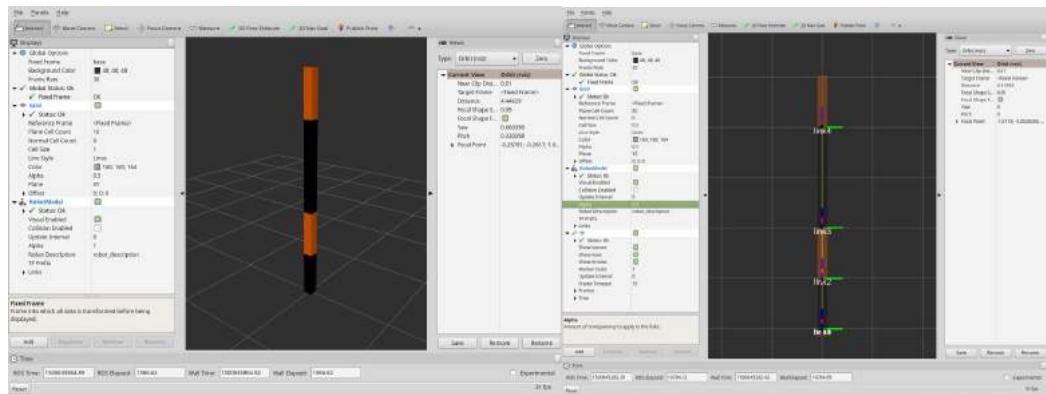


图 13-12 RViz中各个关节和连杆的形象

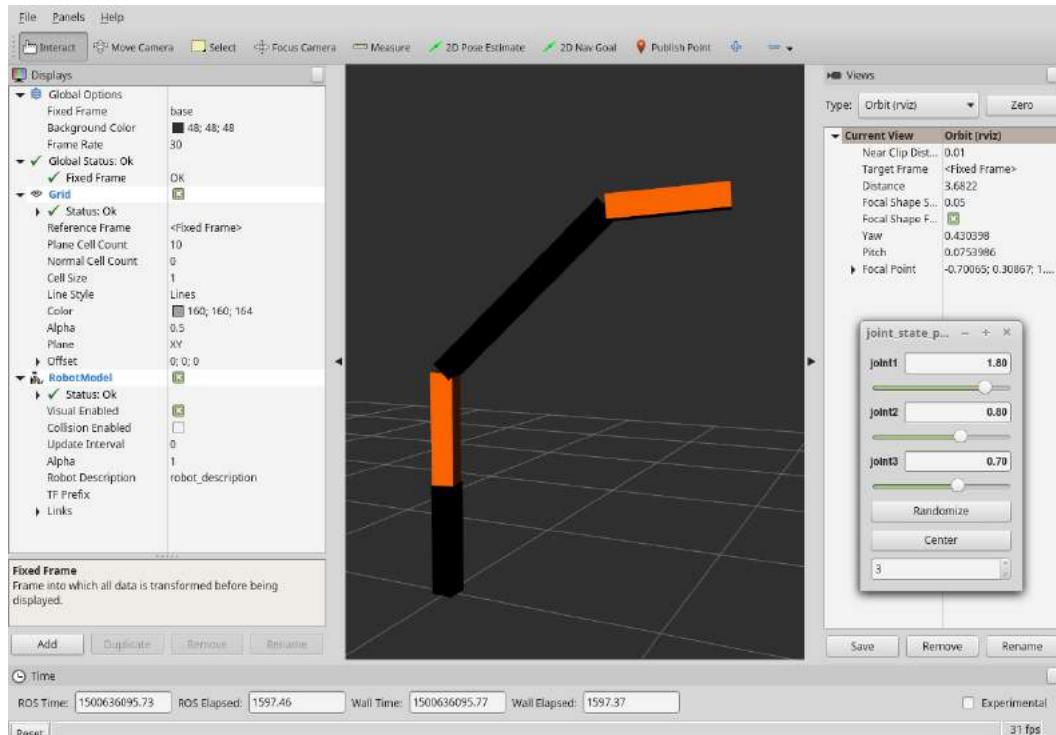


图 13-13 在Joint State Publisher GUI工具中操纵各关节的结果

前面已经按URDF格式对三轴机械手臂进行了建模，并使用RViz对其进行确认。基于此，我们来看一下OpenManipulator Chain的URDF，它由4轴关节和线性抓手组成。首先，下载OpenManipulator和TurtleBot3的GitHub上的源代码。

```
$ cd ~/catkin_ws/src  
$ git clone https://github.com/ROBOTIS-GIT/open_manipulator.git  
$ cd ~/catkin_ws && catkin_make
```

```
$ cd ~/catkin_ws/src/  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git  
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git  
$ cd ~/catkin_ws && catkin_make
```

以下是从Github复制的OpenManipulator目录中的内容。

```
$ cd ~/catkin_ws/src/open_manipulator  
$ ls  
Arduino                               → Arduino库  
open_manipulator_description          → 建模功能包  
open_manipulator_dynamixel_ctrl      → Dynamixel控制功能包  
open_manipulator_gazebo              → Gazebo package  
open_manipulator_moveit              → MoveIt!功能包  
open_manipulator_msgs                → 信息功能包  
open_manipulator_position_ctrl      → 位置控制功能包  
open_manipulator_with_tb3            → OpenManipulator和TurtleBot3功能包
```

建模功能包（open_manipulator_description）由包含可执行文件的launch目录、包含设计文件的meshes目录、包含发布者节点的src目录以及urdf目录组成。打开urdf目录和launch目录并查看内容。

```
$ roscd open_manipulator_description/urdf  
$ ls  
materials.xacro                      → 材质信息  
open_manipulator_chain.xacro          → 机械手臂建模  
open_manipulator_chain.gazebo.xacro   → 机械手臂Gazebo建模
```

```
$ roscd open_manipulator_description/launch  
$ ls  
open_manipulator.rviz                 → RViz配置文件  
open_manipulator_chain_ctrl.launch    → 机械手臂状态信息发布者节点运行文件  
open_manipulator_chain_rviz.launch    → 机器人建模信息可视化节点运行文件
```

如果确认完毕，请打开materials.xacro文件。

```
$ roscd open_manipulator_description/urdf  
$ gedit materials.xacro
```

```
open_manipulator_description/urdf/materials.xacro

<?xml version="1.0"?>
<robot>

<material name="black">
<color rgba="0.0 0.0 0.0 1.0"/>
</material>

<material name="white">
<color rgba="1.0 1.0 1.0 1.0"/>
</material>

<material name="red">
<color rgba="0.8 0.0 0.0 1.0"/>
</material>

<material name="blue">
<color rgba="0.0 0.0 0.8 1.0"/>
</material>

<material name="green">
<color rgba="0.0 0.8 0.0 1.0"/>
</material>

<material name="grey">
<color rgba="0.5 0.5 0.5 1.0"/>
</material>

<material name="orange">
<color rgba="${255/255} ${108/255} ${10/255} 1.0"/>
</material>

<material name="brown">
<color rgba="${222/255} ${207/255} ${195/255} 1.0"/>
```

```
</material>  
  
</robot>
```

.xacro文件是XML Macro³⁹的缩略语，是一种可以调用反复使用的代码的宏语言，建议将重复使用的代码做一个宏。material.xacro文件指定了接下来要制作的机械手臂的可视化过程中需要的颜色。

接下来，我们来查看OpenManipulator Chain的建模和可视化所需的URDF文件。

```
$ roscd open_manipulator_description/urdf  
$ gedit open_manipulator_chain.xacro
```

open_manipulator_description/urdf/open_manipulator_chain.xacro

```
<!-- some parameters -->  
<xacro:property name="pi" value="3.141592654" />  
  
<!-- Import all Gazebo-customization elements, including Gazebo colors -->  
<xacro:include filename="$(find open_manipulator_description)/urdf/open_manipulator_chain.gazebo.xa  
cro" />  
<!-- Import RViz colors -->  
<xacro:include filename= "$(find open_manipulator_description)/urdf/materials.xacro" />
```

URDF有一个缺点是，在表达连杆和关节的连续结构时需要用到许多重复的语句，因此需要较多时间花费在修改代码上。但是，使用xacro可以大大减少这些任务。例如，如上面的代码，可以设置圆周的变量，或分别管理前面创建的物质信息文件和Gazebo配置文件并将他们包含在实际使用的文件当中，以此有效地管理代码。

我们已经在为三轴机械手臂创建URDF时讨论了<link>和<joint>标签。除此之外，OpenManipulator Chain为了和ROS-CONTROL一起运行，还使用<transmission>标签。我们来看看这个。

³⁹ <http://wiki.ros.org/xacro>

open_manipulator_description/urdf/open_manipulator_chain.xacro

```
<!-- Transmission 1 -->
<transmission name="tran1">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint1">
    <hardwareInterface>PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor1">
    <hardwareInterface>PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

<transmission>是与ROS-CONTROL一起运行所必须的标签，它输入关节与舵机之间的命令接口。命令接口有力（effort）、速度（velocity）和位置（position），用户可以根据需要选择要输入的控制量。



<transmission>标签

<transmission>:	设置关节和舵机之间的变量
<type>:	设置力的传递方式的形状
<joint>:	设置关节信息设置
<hardwareInterface>:	设置硬件接口
<actuator>:	设置舵机信息
<mechanicalReduction>:	设置舵机与关节之间的齿轮比

OpenManipulator Chain由四个关节（电机）和五个连杆组成，其中两个连杆和一个关节（电机）组成一个线性抓手。线性抓手部分的关节形态是prismatic的，其余描述和前面的内容重复，所以请查看URDF文件。

为了利用已完成的URDF文件在RViz中显示机械手臂，运行launch文件，并使用joint_state_publisher GUI移动关节，如图13-14所示。

```
$ roslaunch open_manipulator_description open_manipulator_chain_rviz.launch
```

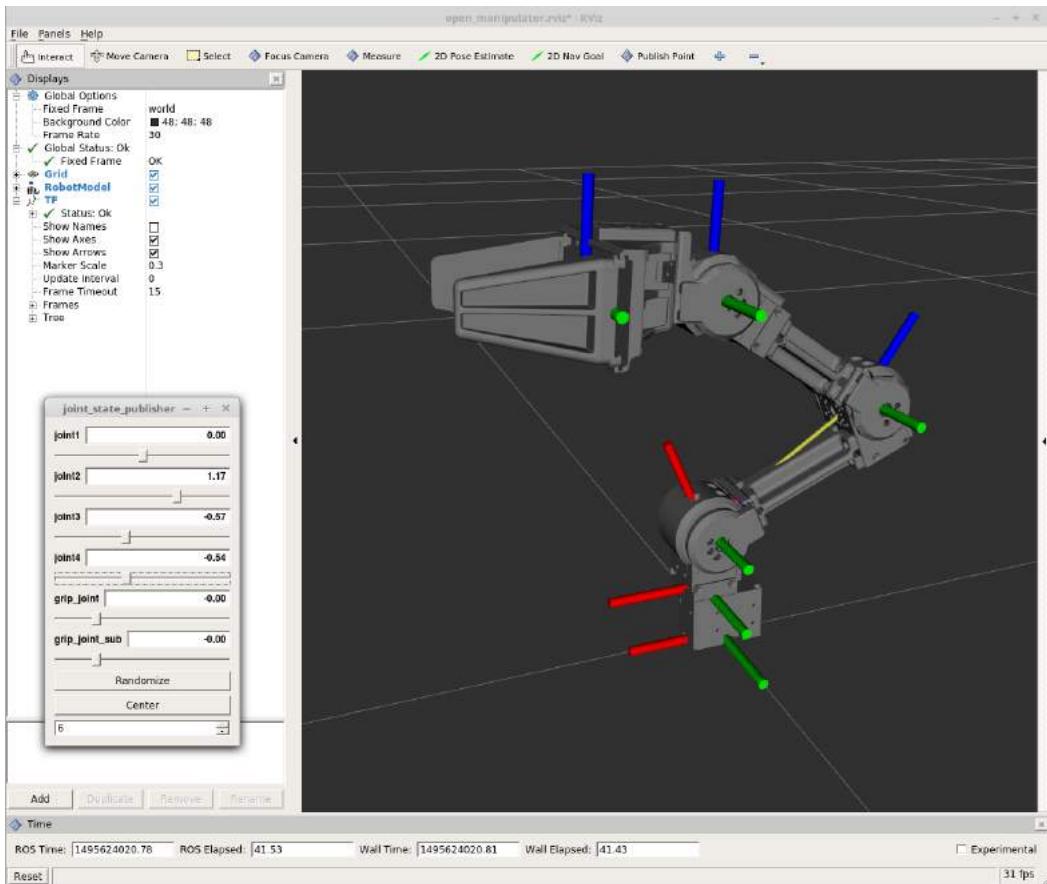


图 13-14 通过GUI更改关节值的OpenManipulator Chain

13.2.3. Gazebo设置

Gazebo是一个三维机器人仿真器，它是独立的软件，支持ROS。可以用它进行机器人设计、算法测试、回归分析和人工智能训练，并且它支持多种机器人，因此许多ROS用户将它用于机器人仿真。由于RViz是一种可视化工具，因此无法实时地获得工作中的机器人或周围环境的物理变化（惯性、扭矩和碰撞等），而Gazebo具有可以对这些数据进行实时监控的优点。通过这种实时监控功能，可以防止机器人在实验过程中发生的故障和人为事故。

上一节中创建的URDF是为使用RViz实现可视化而设计的。让我们在这个文件中添加一些可以用于Gazebo仿真环境的几种标签。用于Gazebo仿真的标签存储在open_manipulator_chain.gazebo.xacro文件中。我们来看看这个。

```
$ roscd open_manipulator_description/urdf  
$ gedit open_manipulator_chain.gazebo.xacro
```

```
open_manipulator_description/urdf/open_manipulator_chain.gazebo.xacro
```

```
<!-- Link1 -->  
<gazebo reference="link1">  
  <mu1>0.2</mu1>  
  <mu2>0.2</mu2>  
  <material>Gazebo/Grey</material>  
</gazebo>
```

对于在Gazebo中使用的连杆的设置必不可少的是色彩和惯性信息。由于惯性信息包含在之前创建的URDF文件中，因此只需设置颜色。另外，可以给Gazebo支持的开放动力学引擎（ODE， Open Dynamics Engine）⁴⁰ ⁴¹设置重力、阻尼和摩擦力。在上述文件中，仅以摩擦系数为例。虽然没有指出，但还有关节信息的参数，望读者自己确认。



<gazebo>标签

```
<gazebo>:      设置Gazebo仿真的参数  
<mu1>, <mu2>: 设置摩擦系数  
<material>:    设置连杆颜色
```

```
open_manipulator_description/urdf/open_manipulator_chain.gazebo.xacro
```

```
<!-- ros_control plugin -->  
<gazebo>  
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">  
    <robotNamespace>/open_manipulator_chain</robotNamespace>  
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
```

⁴⁰ http://gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros

⁴¹ <http://www.ode.org/>

```
</plugin>  
</gazebo>
```

Gazebo plugin⁴²是使用ROS消息和服务通信支持通过URDF或SDF创建的机器人模型的传感器和电机的状态和控制的工具。Plugin支持各种传感器，如相机、激光和惯性导航传感器等多种传感器，还支持差速器、滑移转向、平行移动等移动平台控制和ROS-CONTROL。OpenManipulator Chain使用关节的位置控制界面，并启用默认的插件库。



<gazebo>标签

- <gazebo>： 设置Gazebo仿真的参数
- <plugin>： 传感器和机器人状态控制工具
- <robotNamespace>： 设置在Gazebo中使用的机器人名称
- <robotSimType>： 设置机器人仿真界面的插件名称

其余的是重复的内容，所以请自行查看下面的源代码。

open_manipulator_description/urdf/open_manipulator_chain.gazebo.xacro

```
<?xml version="1.0"?>  
<robot>  
  
    <!-- World -->  
    <gazebo reference="world">  
        </gazebo>  
  
    <!-- Link1 -->  
    <gazebo reference="link1">  
        <mu1>0.2</mu1>  
        <mu2>0.2</mu2>  
        <material>Gazebo/Grey</material>  
    </gazebo>  
  
    <!-- Link2 -->
```

⁴² http://gazebosim.org/tutorials?tut=ros_gzplugins&cat=connect_ros

```
<gazebo reference="link2">
<mu1>0.2</mu1>
<mu2>0.2</mu2>
<material>Gazebo/Grey</material>
</gazebo>

<!-- Link3 -->
<gazebo reference="link3">
<mu1>0.2</mu1>
<mu2>0.2</mu2>
<material>Gazebo/Grey</material>
</gazebo>

<!-- Link4 -->
<gazebo reference="link4">
<mu1>0.2</mu1>
<mu2>0.2</mu2>
<material>Gazebo/Grey</material>
</gazebo>

<!-- Link5 -->
<gazebo reference="link5">
<mu1>0.2</mu1>
<mu2>0.2</mu2>
<material>Gazebo/Grey</material>
</gazebo>

<!-- grip_link -->
<gazebo reference="grip_link">
<mu1>0.2</mu1>
<mu2>0.2</mu2>
<material>Gazebo/Grey</material>
</gazebo>

<!-- grip_link_sub -->
<gazebo reference="grip_link_sub">
<mu1>0.2</mu1>
<mu2>0.2</mu2>
<material>Gazebo/Grey</material>
```

```
</gazebo>

<!-- ros_control plugin-->
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/open_manipulator_chain</robotNamespace>
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
  </plugin>
</gazebo>

</robot>
```

open_manipulator_chain.gazebo.xacro是为设置Gazebo仿真参数单独创建的文件，包含在open_manipulator_chain.xacro文件中。现在使用已完成的URDF将OpenManipulator Chain输出到Gazebo环境中。

```
$ roscd open_manipulator_gazebo/launch
$ ls
open_manipulator_gazebo.launch          → Gazebo运行文件
position_controller.launch             → ROS-CONTROL运行文件
```

launch目录是open_manipulator_gazebo目录的子目录，包含用于运行Gazebo并执行ROS-CONTROL的launch文件。我们来打开Gazebo运行文件，查看都包含哪些节点。

```
$ roscd open_manipulator_gazebo/launch
$ gedit open_manipulator_gazebo.launch
```

```
open_manipulator_gazebo/launch/open_manipulator_gazebo.launch

<?xml version="1.0" ?>
<launch>
  <!-- These are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
```

```

<include file="$(find gazebo_ros)/launch/empty_world.launch">
<arg name="world_name" value="$(find open_manipulator_gazebo)/world/empty.world"/>
<arg name="debug" value="$(arg debug)" />
<arg name="gui" value="$(arg gui)" />
<arg name="paused" value="$(arg paused)" />
<arg name="use_sim_time" value="$(arg use_sim_time)" />
<arg name="headless" value="$(arg headless)" />
</include>

<!-- Load the URDF into the ROS Parameter Server -->
<param name="robot_description" command="$(find xacro)/xacro.py '$(find open_manipulator_description)/urdf/open_manipulator_chain.xacro'" />

<!-- Run a python script to the send a service call to gazebo_ros to spawn a URDF robot -->
<node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen"
args="-urdf -model open_manipulator_chain -z 0.0 -param robot_description" />

<!-- ros_control robotis manipulator launch file -->
<include file="$(find open_manipulator_gazebo)/launch/position_controller.launch" />
</launch>
```

前面的可执行文件包含empty_world.launch文件、spawn_model节点和position_controller.launch文件。empty_world.launch文件包含运行Gazebo的节点，因此您可以设置仿真环境、GUI和时间。Gazebo仿真环境支持SDF⁴³格式的文件。spawn_model节点根据上面创建的URDF调用机器人，position_controller.launch负责设置和运行ROS-CONTROL。

现在，通过在终端中输入以下可执行代码来运行Gazebo，则可以在Gazebo仿真空间中看到OpenManipulator Chain，如图13-15所示。

```
$ roslaunch open_manipulator_gazebo open_manipulator_gazebo.launch
```

⁴³ <http://sdformat.org/>

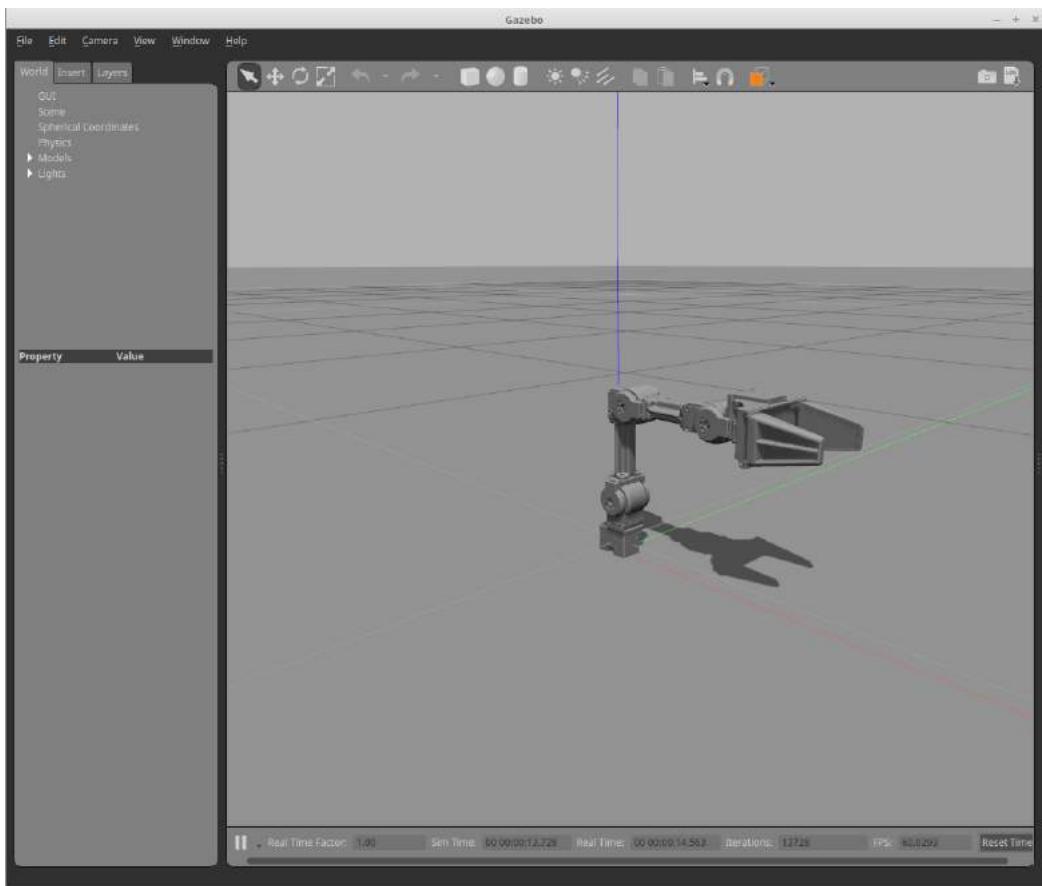


图 13-15 仿真实空间中的OpenManipulator Chain

```
$ rostopic list
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/joint_states
/open_manipulator_chain/grip_joint_position/command
/open_manipulator_chain/grip_joint_sub_position/command
/open_manipulator_chain/joint1_position/command
/open_manipulator_chain/joint2_position/command
```

```
/open_manipulator_chain/joint3_position/command  
/open_manipulator_chain/joint4_position/command  
/open_manipulator_chain/joint_states  
/rosout  
/rosout_agg
```

查看话题列表，可以看到话题分为具有/gazebo命名空间的话题和具有/open_manipulator_chain命名空间的话题。我们可以通过ROS-CONTROL使用/open_manipulator_chain命名空间的话题来检查和控制Gazebo上机器人的状态。让我们通过以下命令移动机器人。

```
$ rostopic pub /open_manipulator_chain/joint2_position/command std_msgs/Float64 "data: 1.0" --once
```

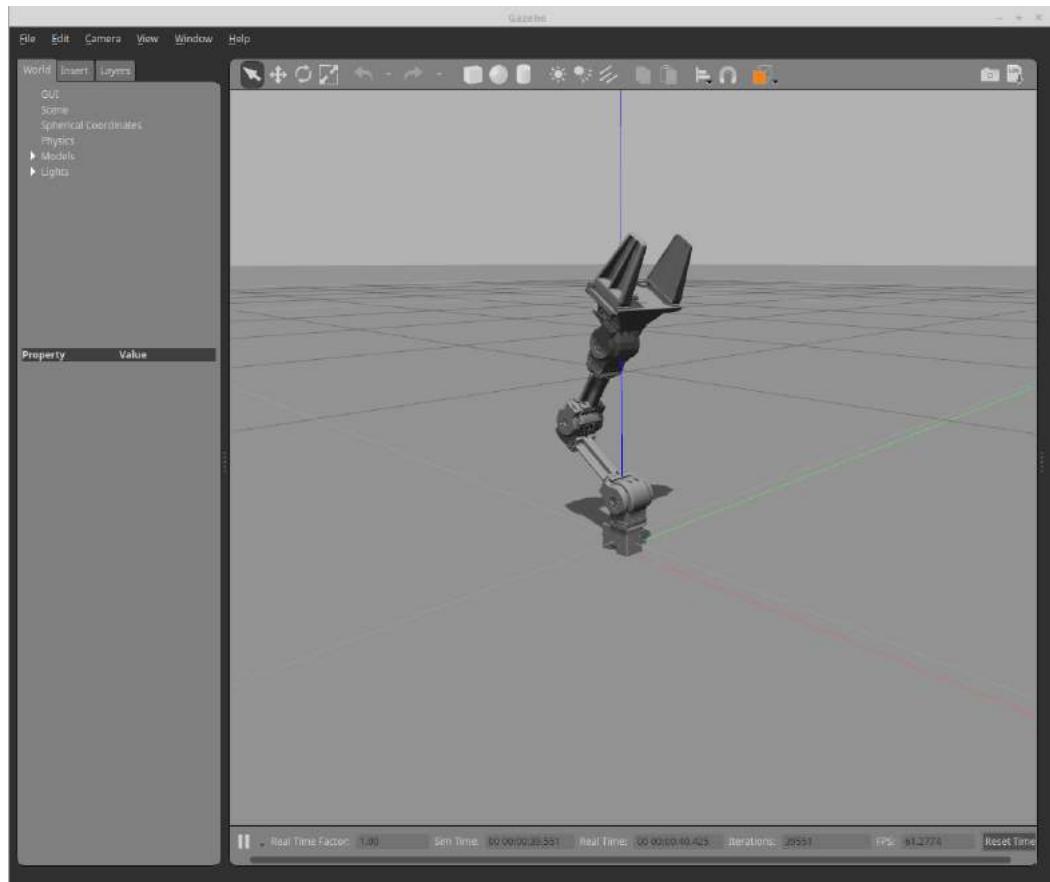


图 13-16 通过与ROS-CONTROL的通信来控制OpenManipulator Chain

通过简单的消息通信，可以看到OpenManipulator Chain的第二个关节的运动，如图13-16所示。

13.3. MoveIt!

MoveIt!⁴⁴是一个集成的机械手臂库，提供多种功能，包括用于运动规划的快速逆运动学分析、用于操纵的高级算法、机械手控制、动力学、控制器和运动规划。通过提供一个GUI来协助MoveIt!所需的各种设置，它还具有的一个优点是没有对机械手臂的高级知识也能容易使用。这是许多ROS用户喜欢的工具，因为它允许使用RViz进行视觉反馈。下面我们先简单了解MoveIt!的结构，然后创建一个用于控制OpenManipulator Chain的MoveIt!功能包。

13.3.1. move_group

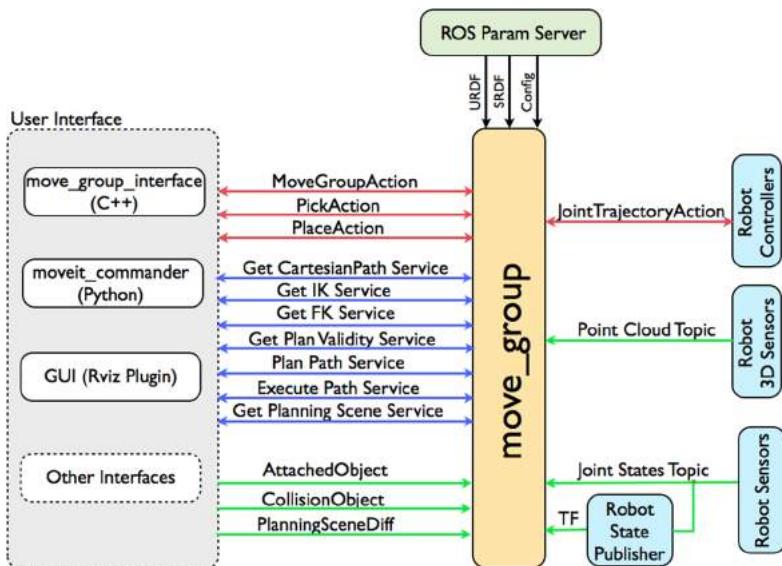


图 13-17 与 move_group 节点的通信方案

⁴⁴ <http://moveit.ros.org/>

如图13-17所示，move_group节点可以使用ROS动作和服务与用户交换命令。提供各种用户界面的MoveIt!为C++语言提供move_group interface，为Python语言提供move_commander，而且构建了允许很多用户利用“Motion Planning plugin to RViz”选择自己熟悉的界面与move_group节点通信的服务。

move_group节点从URDF、Semantic Robot Description Format (SRDF)⁴⁵和MoveIt! Configuration接收关于机器人的信息。URDF使用先前创建的文件，而SRDF和MoveIt! Configuration将通过MoveIt!提供的Setup Assistant⁴⁶创建。

move_group节点通过ROS话题和动作提供机器人的状态与控制，还提供周围环境。关节状态是通过sensor_msgs/JointStates消息，变换信息是通过tf库，控制器是通过FollowTrajectoryAction接口向用户发送关于机器人的信息。另外，通过planning scene向用户提供关于机器人工作的环境信息和机器人的状态。

move_group为其可扩展性提供了一个plugin功能，并提供了一个通过开源库将各种功能（控制、路径生成、动力学等）应用到用户的机器人的机会。MoveIt!内置的插件是已经被许多人验证过的优秀的库，并且还有许多最近开发的开源库也待发布。典型的例子有The Open Motion Planning Library (OMPL)⁴⁷、Kinematic and Dynamic Library (KDL)⁴⁸和A Flexible Collision Library (FCL)⁴⁹。

13.3.2. MoveIt! Setup Assistant

为了创建用于机械手臂的MoveIt!功能包，需要URDF、SRDF以及用于MoveIt! Configuration的文件。MoveIt!提供的Setup Assistant（设置助手）基于URDF生成用于创建MoveIt!功能包的SRDF和MoveIt! Configuration文件。接下来了解一下利用MoveIt! Setup Assistant来创建用于OpenManipulator Chain的MoveIt!功能包的方法。

在终端中输入以下命令运行MoveIt! Setup Assistant。

```
$ rosrun moveit_setup_assistant setup_assistant.launch
```

⁴⁵ <http://wiki.ros.org/srdf>

⁴⁶ http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html

⁴⁷ <http://ompl.kavrakilab.org/>

⁴⁸ <http://www.orocos.org/kdl>

⁴⁹ http://gamma.cs.unc.edu/FCL/fcl_docs/webpage/generated/index.html



图13-18 MoveIt! Setup Assistant的开始页面

图13-18是运行MoveIt! Setup Assistant后看到的第一个开始页面。在此页面中，您可以在右侧看到ROS的代表性的乌龟图标，并且可以在左侧窗口中选择创建新的功能包还是修改现有的功能包。现在我们需要创建一个新的功能包，所以让我们点击Create New MoveIt Configuration Package按钮。

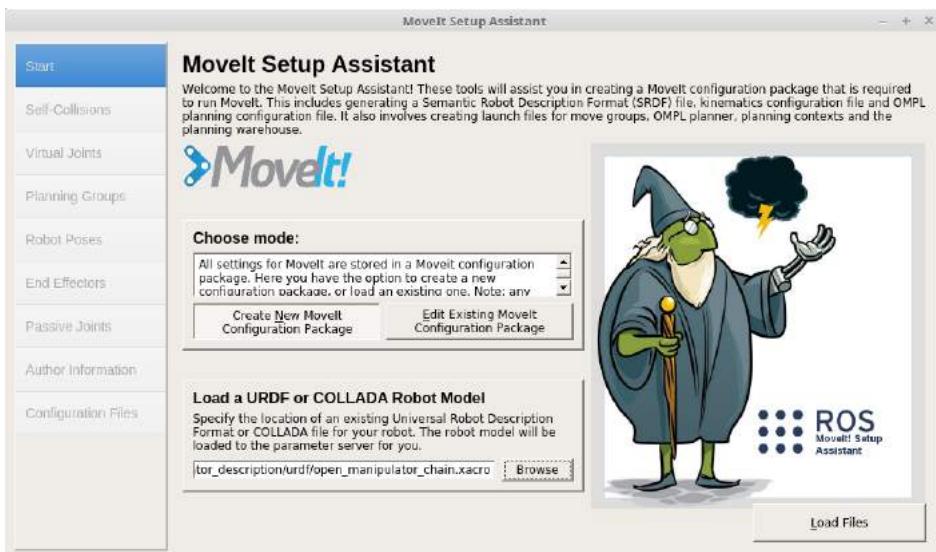


图13-19 MoveIt! Setup Assistant的开始页面

MoveIt! Setup Assistant会根据存储在URDF文件或COLLADA文件中的机器人模型，通过附加的设置生成SRDF文件。点击图13-19中所示的Browse按钮，打开前面创建的open_manipulator_chain.xacro文件，然后点击Load Files按钮。

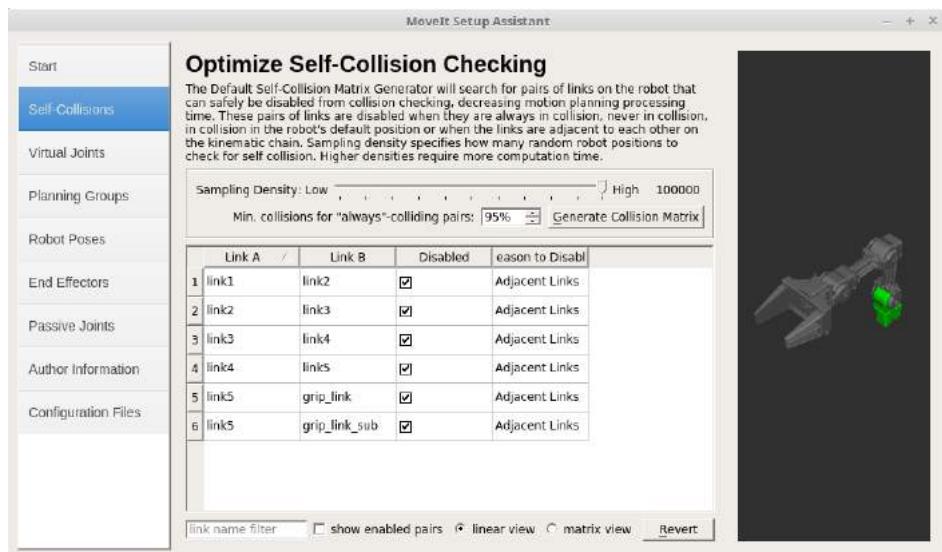


图13-20 MoveIt! Setup Assistant中的“Self-Collisions”页面

如果您已成功加载文件，请转到“Self-Collision”页面。通过此页面，可以设置构建Self-Collision Matrix（自碰撞矩阵）所需的Sampling density（采样密度），并可以根据需要确定组成机器人的连杆之间的碰撞范围，如图13-20所示。Sampling density越高，需要越多的计算来防止机器人在各种姿态中的连杆之间的碰撞，因此需要用户的工作环境进行适当的设置。设置需要的Sampling density，然后单击Generate Collision Matrix按钮。默认值设置为10,000。



图13-21 MoveIt! Setup Assistant的Virtual Joints页面

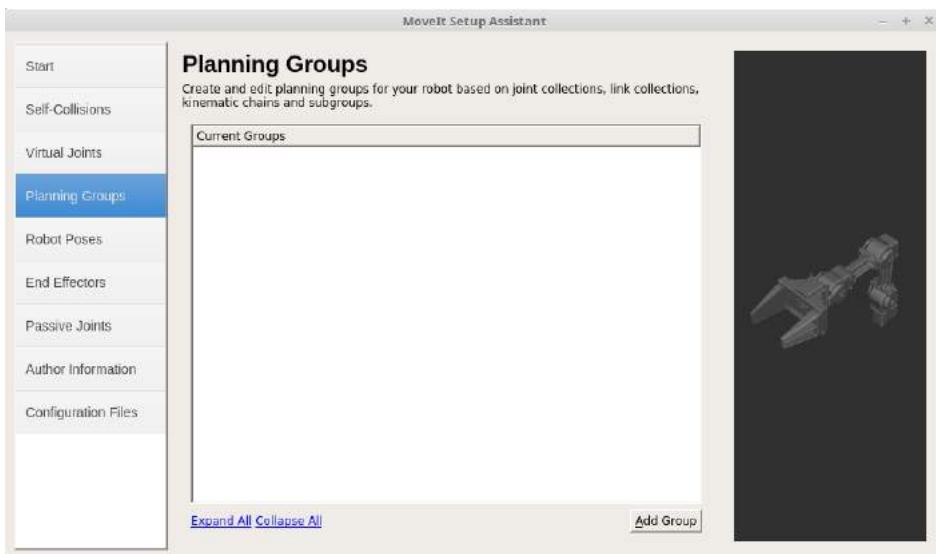


图13-22 MoveIt! Setup Assistant的Planning Groups页面

MoveIt!将机械手臂分为几个组，为各组分别提供运动规划，用户可以在Planning Groups页面上进行分组。OpenManipulator Chain由四个关节和一个抓手组成。点击图13-22右下方的add group按钮后窗口画面将会改变，如图13-23所示。

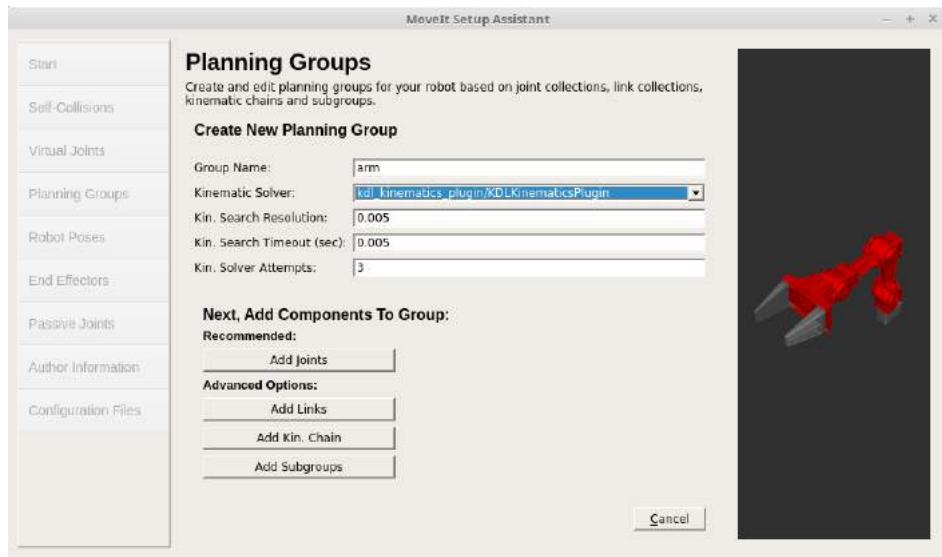


图13-23 在Planning Groups页面上新建组的页面

在上一个窗口中，可以指定组名称并选择所需的运动分析插件（Kinematic Slover项目）。组名设为arm，并选择需要的机械学解析插件。因为将要以关节为单元分组，因此点击Add joint按钮。如果窗口如图13-24所示变化，则选择第一个关节到第四个关节，然后单击Save按钮，则可以看到已创建的组，如图13-25所示。

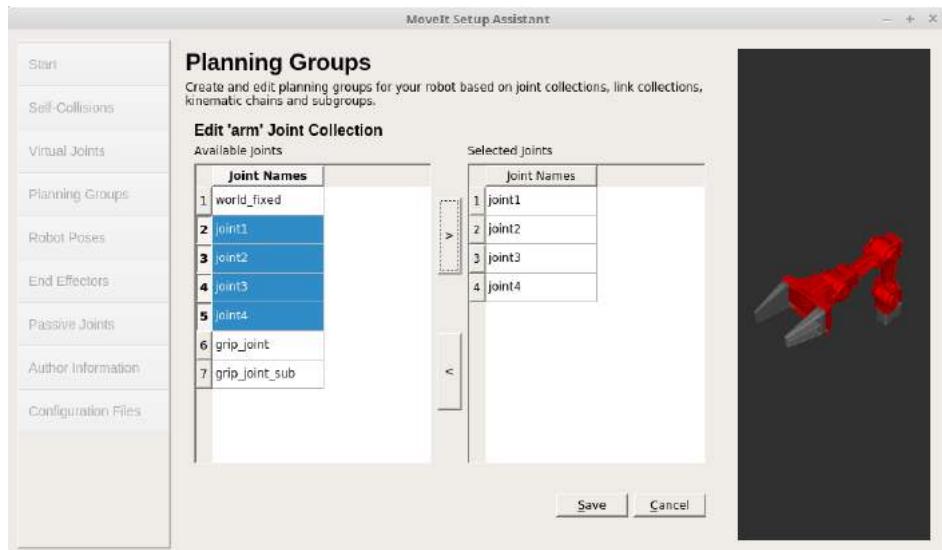


图13-24 Planning Groups页面中创建新组的窗口

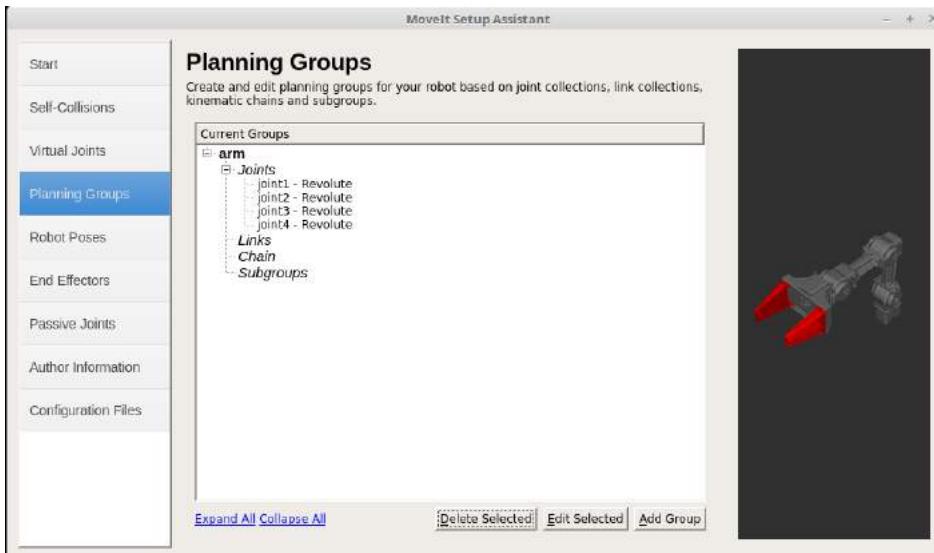


图13-25 在Planning Groups页面上创建的arm组

如果您创建了一个arm组，让我们创建一个类似的gripper组。只由一个舵机控制的线性抓手无法在机械分析插件中运行。因此，在创建gripper组时，请将运动学分析插件设置为None。并且像arm组一样，在Add joints中，选择相关的关节grip_joint和grip_joint_sub。gripper组完成后，您可以看到arm组和gripper组，如图13-26所示。

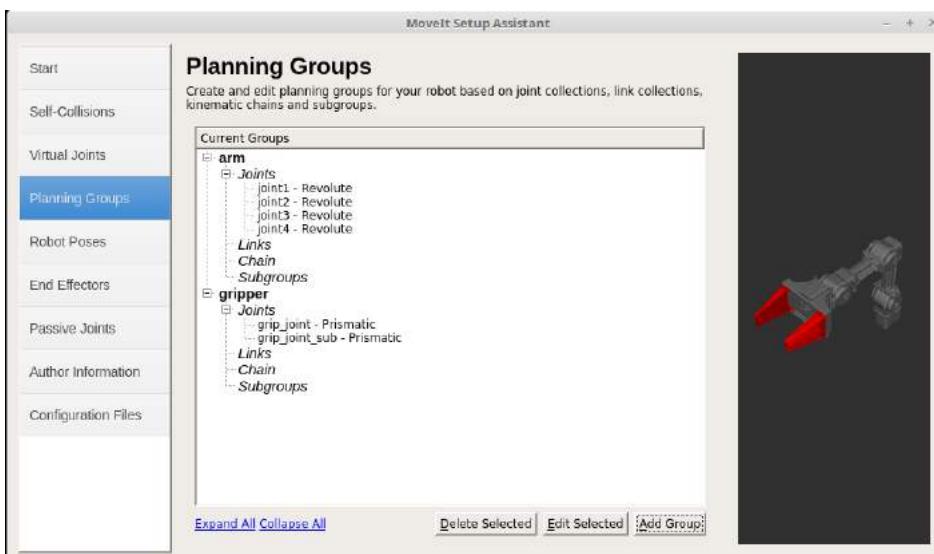


图13-26 Planning Groups页面上创建的arm组和gripper组

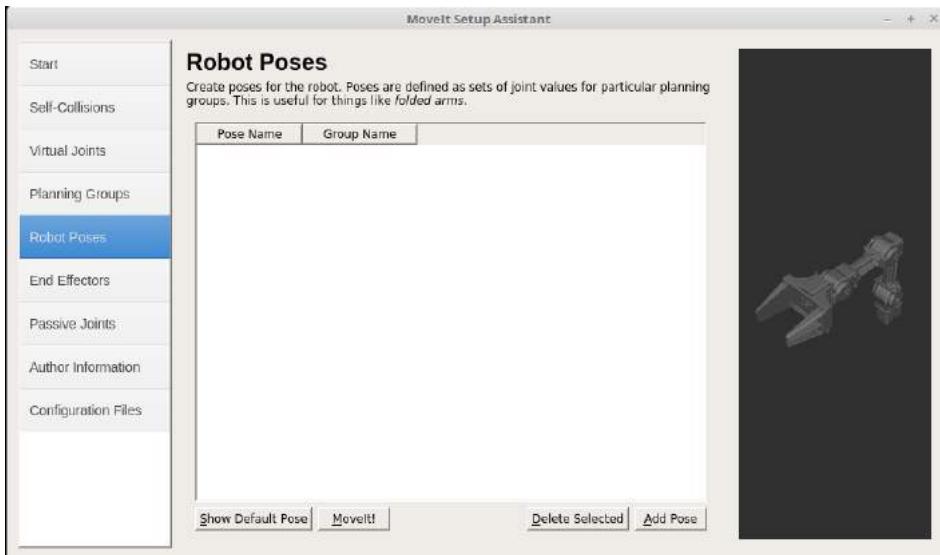


图13-27 MoveIt! Setup Assistant的Robot Poses页面

Robot Poses页面允许您创建和注册机器人的特殊姿态。点击图13-27右下方的Add Pose按钮，注册所有舵机的角度为0的姿态。点击按钮会进入姿态创建窗口，如图13-28所示，在这里您可以将所有关节值设置为0.0，然后填入姿态名称。

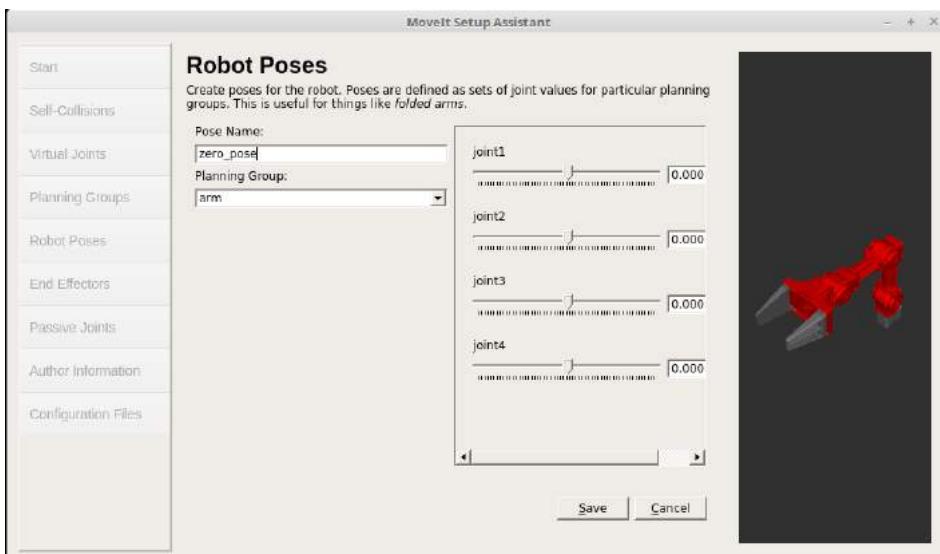


图13-28 在Robot Poses页面窗口上创建姿态



图13-29 MoveIt! Setup Assistant的End Effectors页面

End Effectors页面可以注册机械手臂的end-effector（末端执行器）。点击图13-29右下角的Add End Effector按钮来注册OpenManipulator Chain的线性抓手。

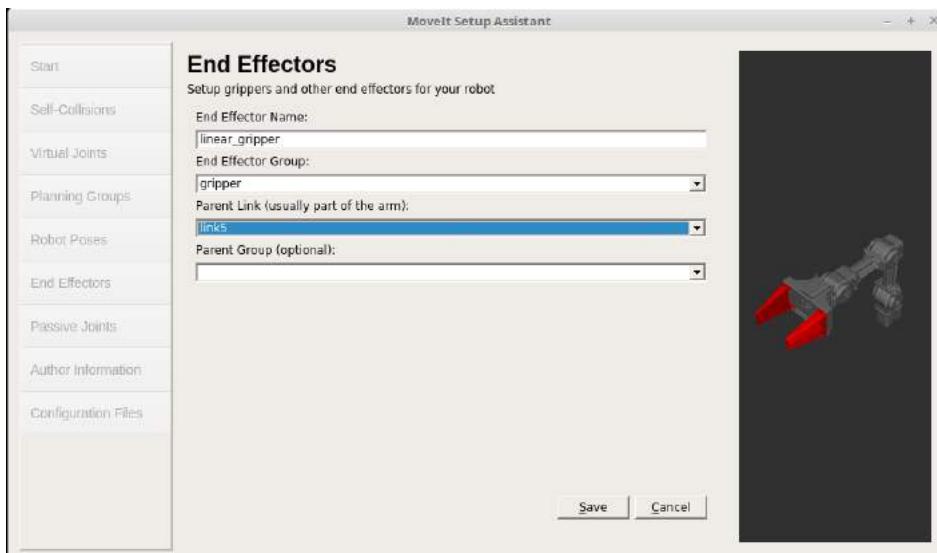


图13-30 End Effectors页面的End Effectors设置窗口

如图13-30所示，命名End Effectors，并在Planning groups页面中创建的组中选择gripper。检查URDF可以发现，抓手的第五个连杆作为其父连杆。

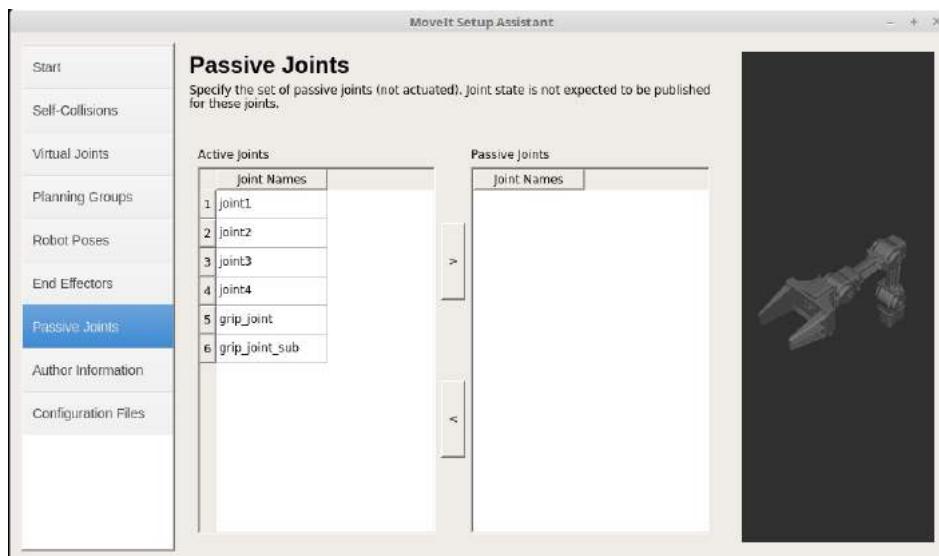


图13-31 MoveIt! Setup Assistant的Passive Joints页面

passive Joints页面允许您指定不属于运动规划的关节。在OpenManipulator Chain中，没有passive joint，因此不作任何改变进入下一阶段，如图13-31所示。

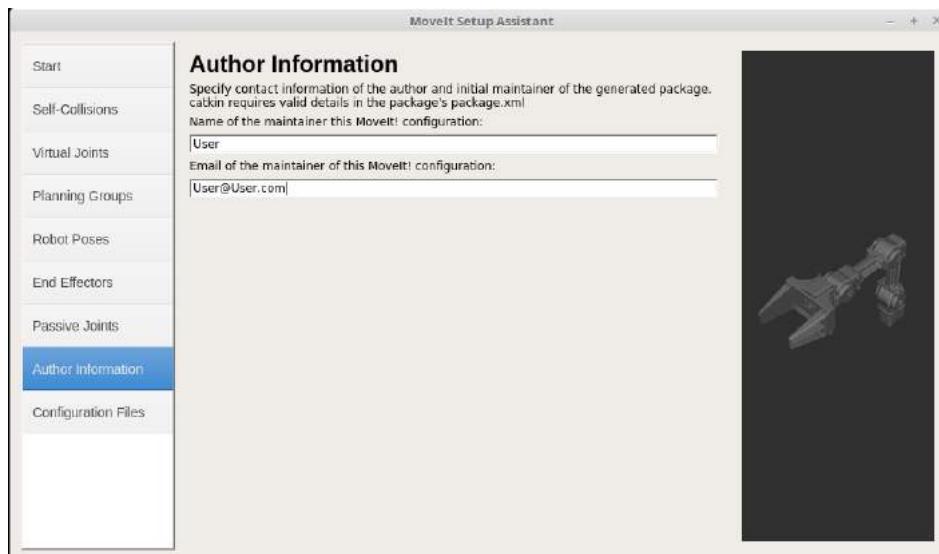


图13-32 MoveIt! Setup Assistant的Author Information页面

在Author information页面上，要输入创建功能包的用户的姓名和电子邮件，如图13-32所示。

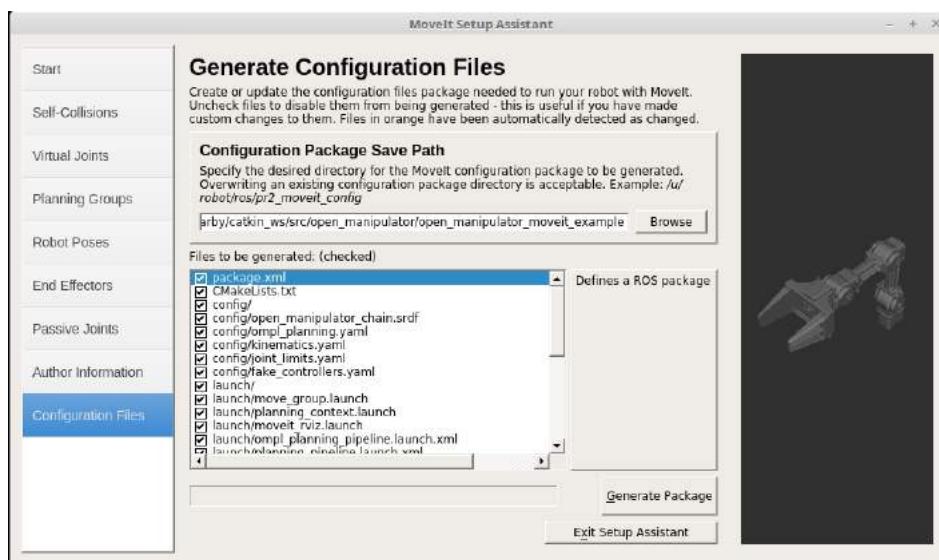


图13-33 MoveIt! Setup Assistant的Configuration Files页面

完成所有设置后，您可以在Configuration Files页面上完成配置。单击图13-33顶部的Browse按钮，在open_manipulator目录中创建open_manipulator_moveit_example目录，选中它，然后单击右下角的Generate Package按钮，则会生成MoveIt! Configuration所需的config目录和包含可执行文件的launch目录。

查看生成的功能包并运行演示。

```
$ cd ~/catkin_ws/src/open_manipulator/open_manipulator_moveit_example
$ ls
Config           → 用于MoveIt! Configuration的yaml文件和SRDF文件
launch          → 运行文件
.setup_assistant → 由setup assistant创建的功能包的信息
CMakeLists.txt   → CMake构建系统输入文件
package.xml      → 定义功能包的属性
```

```
$ cd ~/catkin_ws && catkin_make
$ roslaunch open_manipulator_moveit_example demo.launch
```

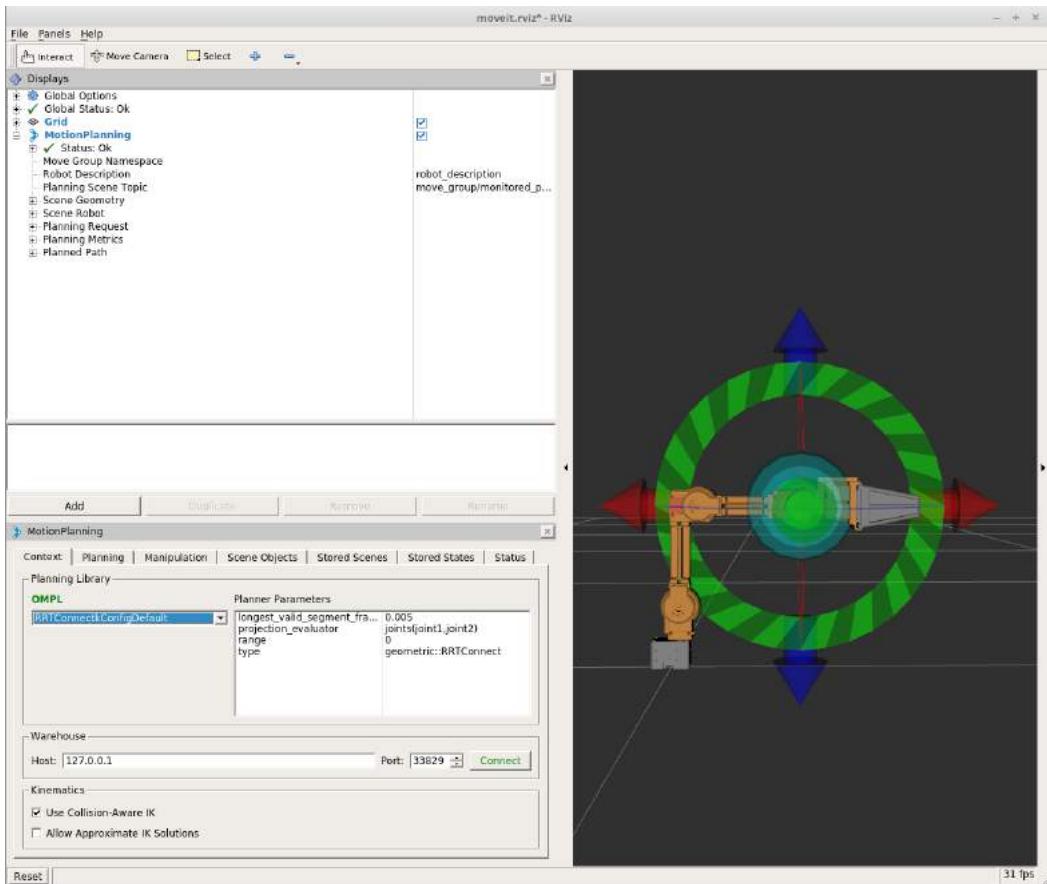


图13-34 MoveIt! RViz demo

运行Demo时，可以通过RViz窗口看到OpenManipulator Chain，如图13-34所示。在位于左下角的MotionPlanning命令窗口的Context页面中，可以选择OMPL支持的路径规划库⁵⁰。选择RRTConnectkConfigDefault并转到Planning页面。

本来，机械手臂终点的坐标由表示运动的X、Y、Z和表示旋转的 Θ （Roll）、 Φ （Pitch）和 Ψ （Yaw）组成的姿态值来表示。但是，由于OpenManipulator Chain只有四个关节，因此终点将只有X、Z和Pitch轴的自由度（其余的一个自由度是一号关节的Yaw轴旋转）。记住这一点，并将终点移动到所需的坐标位置。

⁵⁰ <http://ompl.kavrakilab.org/planners.html>

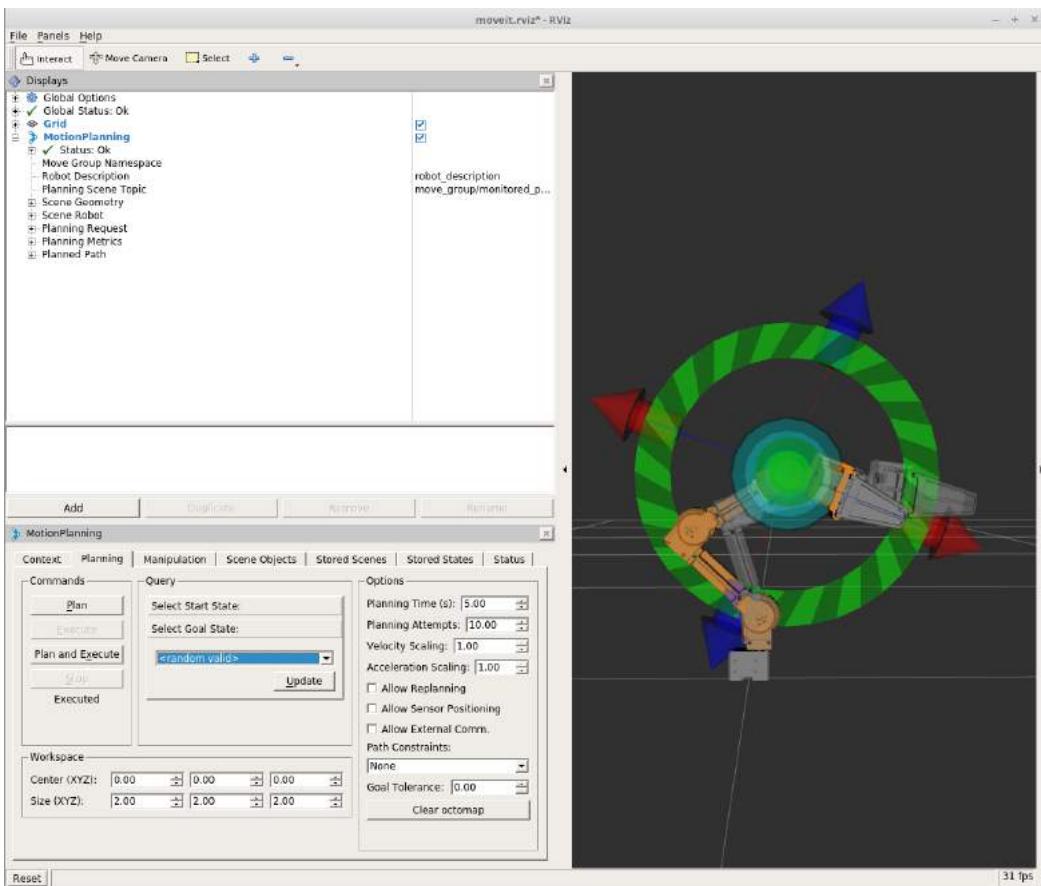


图13-35 使用MoveIt! RViz demo的机械手臂运动规划

如果已将终点移至所需位置，请单击Planning页面上的[Plan & Execute]按钮以检查移动。



指定机械手臂的目标姿态（位置+方向）的方式

在这个例子中，您可以通过在RViz上输入位置和方向来指定机械手臂的目标姿态。除此之外，还可以采用通过绿色球状物（Interactive marker）来指定位置的方法。为此，将position_only_ik: true记录在config目录下的kinematics.yaml文件的底部。

```
$ roscd open_manipulator_moveit/config/
$ gedit kinematics.yaml
```

```
arm:  
kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin  
kinematics_solver_search_resolution: 0.005  
kinematics_solver_timeout: 0.005  
kinematics_solver_attempts: 3  
position_only_ik: true
```

OpenManipulator提供MoveIt!示例包。检查open_manipulator目录中的open_manipulator_moveit功能包目录。

```
$ roscd open_manipulator_moveit  
$ ls  
Config           → 用于move_group设置的yaml文件和SRDF文件  
include          → trajectory filter头文件  
launch           → 运行文件  
src              → trajectory filter源代码文件  
.setup_assistant → 由setup assistant创建的功能包的信息  
CMakeLists.txt   → CMake构建系统输入文件  
package.xml      → 定义功能包的属性  
planning_request_adapters_plugin_description.xml → plugin安装文件
```

可以看到比起用setup assistant生成的文件更多的文件。路径生成算法中被人们熟知的Rapidly-exploring Random Tree (RRT)⁵¹随机采样路径以找到从当前位置移动到目标位置的路径，然后返回搜索得出的结果。返回的每个关节值表示移动到目标位置所需的姿态。但是，为了从第n个姿态向第n+1个姿态移动，需要以较短的采样时间获得的关节值，因此使用了ROS-INDUSTRIAL支持的industrial_trajectory_filters⁵²。



industrial_trajectory_filters应用方法

1. 下载ROS-INDUSTRIAL CORE功能包。

```
$ git clone https://github.com/ros-industrial/industrial_core.git
```

⁵¹ https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

⁵² http://wiki.ros.org/industrial_trajectory_filters

2. 查看下载的ros-industrial功能包中的industrial_trajectory_filters。

```
$ cd ~/catkin_ws/src/industrial_core/industrial_trajectory_filters/
```

3. 将industrial_trajectory_filters目录中的planning_request_adapters_plugin_description.xml目录和src目录复制到之前创建的moveit功能包中。

```
$ cp -r planning_request_adapters_plugin_description.xml src include ~/catkin_ws/src/open_manipulator/open_manipulator_moveit_example/
```

4. 在config目录中创建smoothing_filter_params.yaml并指定系数。

```
$ cd ~/catkin_ws/src/open_manipulator/open_manipulator_moveit_example/config  
$ gedit smoothing_filter_params.yaml  
smoothing_filter_name: /move_group/smoothing_5_coef  
smoothing_5_coef:  
    - 0.25  
    - 0.50  
    - 1.00  
    - 0.50  
    - 0.25
```

5. 打开launch目录中的ompl_planning_pipeline.launch.xml，并在planning_adapters添加以下过滤器。

```
$ cd ~/catkin_ws/src/open_manipulator/open_manipulator_moveit_example/launch  
$ gedit ompl_planning_pipeline.launch.xml  
industrial_trajectory_filters/UniformSampleFilter  
industrial_trajectory_filters/AddSmoothingFilter
```

6. 将以下参数添加到同一个文件中。

```
<param name="sample_duration" value="0.04" />  
<rosparam command="load" file="$(find open_manipulator_moveit)/config/smoothing_filter_params.yaml" />
```

7. 下面的示例显示了插入步骤5和6时启动文件的内容。

```
<launch>  
    <!-- OMPL Plugin for MoveIt! -->  
    <arg name="planning_plugin" value="ompl_interface/OMPLPlanner" />
```

```

<!-- The request adapters (plugins) used when planning with OMPL.
     ORDER MATTERS -->

<arg name="planning_adapters" value="
    industrial_trajectory_filters/UniformSampleFilter
    industrial_trajectory_filters/AddSmoothingFilter
    default_planner_request_adapters/AddTimeParameterization
    default_planner_request_adapters/FixWorkspaceBounds
    default_planner_request_adapters/FixStartStateBounds
    default_planner_request_adapters/FixStartStateCollision
    default_planner_request_adapters/FixStartStatePathConstraints" />

<arg name="start_state_max_bounds_error" value="0.1" />

<param name="planning_plugin" value="$(arg planning_plugin)" />
<param name="request_adapters" value="$(arg planning_adapters)" />
<param name="start_state_max_bounds_error" value="$(arg start_state_max_bounds_error)" />

<param name="sample_duration" value="0.04" />

<rosparam command="load" file="$(find open_manipulator_moveit)/config/
ompl_planning.yaml" />
<rosparam command="load" file="$(find open_manipulator_moveit)/config/
smoothing_filter_params.yaml" />

</launch>
```

8. 运行命令

```
$ rosrun open_manipulator_moveit_example demo.launch
```

请运行以下命令并观察与以前的演示有什么不同。

```
$ rosrun open_manipulator_moveit open_manipulator_demo.launch
```

13.3.3. Gazebo仿真

此前讲到，Gazebo仿真器能够测试机器人在真实环境中的运动。在本节中，我们尝试通过Gazebo仿真器中的move_group和OpenManipulator Chain之间的消息通信来检查机器人的运动。我们先运行Gazebo。

```
$ rosrun open_manipulator_gazebo open_manipulator_gazebo.launch
```

在上一节中，我们尝试通过消息通信来控制Gazebo仿真器的机器人。我们来看一下将这个功能用源代码表现出来的open_manipulator_position_ctrl功能包。

```
$ roscd open_manipulator_position_ctrl/src  
$ gedit position_controller.cpp
```

open_manipulator_position_ctrl/src/position_controller.cpp

```
bool PositionController::initStatePublisher(bool using_gazebo)  
{  
    // ROS Publisher  
    if (using_gazebo)  
    {  
        ROS_WARN("SET Gazebo Simulation Mode");  
        for (std::map<std::string, uint8_t>::iterator state_iter = joint_id_.begin();  
             state_iter != joint_id_.end(); state_iter++)  
        {  
            std::string joint_name = state_iter->first;  
            gazebo_goal_joint_position_pub_[joint_id_[joint_name]-1]  
                = nh_.advertise<std_msgs::Float64>("/" + robot_name_ + "/" + joint_name + "_position/command", 10);  
        }  
  
        gazebo_gripper_position_pub_[LEFT_GRIP] = nh_.advertise<std_msgs::Float64>("/" + robot_name_ +  
"/grip_joint_position/command", 10);  
        gazebo_gripper_position_pub_[RIGHT_GRIP] = nh_.advertise<std_msgs::Float64>("/" + robot_name_ +  
"/grip_joint_sub_position/command", 10);  
    }  
    else  
    {  
        goal_joint_position_pub_ = nh_.advertise<sensor_msgs::JointState>("/robotis/open_manipulator/  
goal_joint_states", 10);  
    }  
}
```

```

    }

}

bool PositionController::initStateSubscriber(bool using_gazebo)
{
    // ROS Subscriber
    if (using_gazebo)
    {
        gazebo_present_joint_position_sub_ = nh_.subscribe("/" + robot_name_ + "/joint_states", 10,
            &PositionController::gazeboPresentJointPositionMsgCallback, this);
    }
    else
    {
        present_joint_position_sub_ = nh_.subscribe("/robotis/open_manipulator/present_joint_states", 10,
            &PositionController::presentJointPositionMsgCallback, this);
    }

    move_group_feedback_sub_ = nh_.subscribe("/move_group/feedback", 10,
        &PositionController::moveGroupActionFeedbackMsgCallback, this);

    display_planned_path_sub_ = nh_.subscribe("/move_group/display_planned_path", 10,
        &PositionController::displayPlannedPathMsgCallback, this);

    gripper_position_sub_ = nh_.subscribe("/robotis/open_manipulator/gripper", 10,
        &PositionController::gripperPositionMsgCallback, this);
}

```

open_manipulator_position_ctrl功能包的position_controller节点通过与move_group节点的消息通信获得运动规划的结果，然后通过逆运动学计算（用于追随生成的路径）来创建可以控制机械手臂的最终输入值。在上面的代码中，先检查是否使用了Gazebo，如果是在Gazebo环境中控制机械手臂，则会注册与它相对应的话题发布者，并订阅当前机械手臂的每个关节值的状态。

position_controller节点包含在open_manipulator_demo.launch文件中。在运行此演示的命令后面输入与Gazebo进行通信的参数值。

```
$ roslaunch open_manipulator_moveit open_manipulator_demo.launch use_gazebo:=true
```

如果在RViz窗口中选择了所需的运动规划库并定义了终点坐标，并单击Plan和Execute按钮（如图13-35所示），则可以看到，Gazebo仿真器环境中的机械手臂会与RViz窗口中的机械手臂一起运动，如图13-36和图13-37所示。

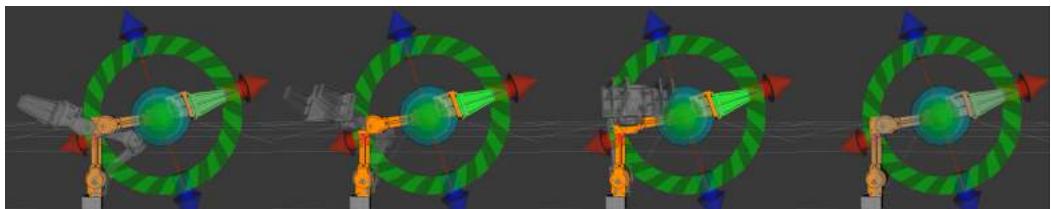


图13-36 使用MoveIt! RViz Demo的机械手臂运动规划



图13-37 使用了MoveIt!的Gazebo环境中的机械手臂的运动画面

position_controller节点负责与move_group的消息通信和线性抓手的控制。

```
open_manipulator_position_ctrl/src/position_controller.cpp

void PositionController::gripperPositionMsgCallback(const std_msgs::String::ConstPtr &msg)
{
    if (msg->data == "grip_on")
    {
        gripOn();
    }
    else if (msg->data == "grip_off")
    {
        gripOff();
    }
    else
    {
        ROS_ERROR("If you want to grip or release something, publish 'grip_on' or 'grip_off'");
    }
}
```

移动线性抓手的方法是发布如下话题。按照如下命令运行，就可以看到抓手是如何工作的，如图13-38所示。要向反方向运行，可以在data输入grip_off。

```
$ rostopic pub /robotis/open_manipulator/gripper std_msgs/String "data: 'grip_on'" --once
```

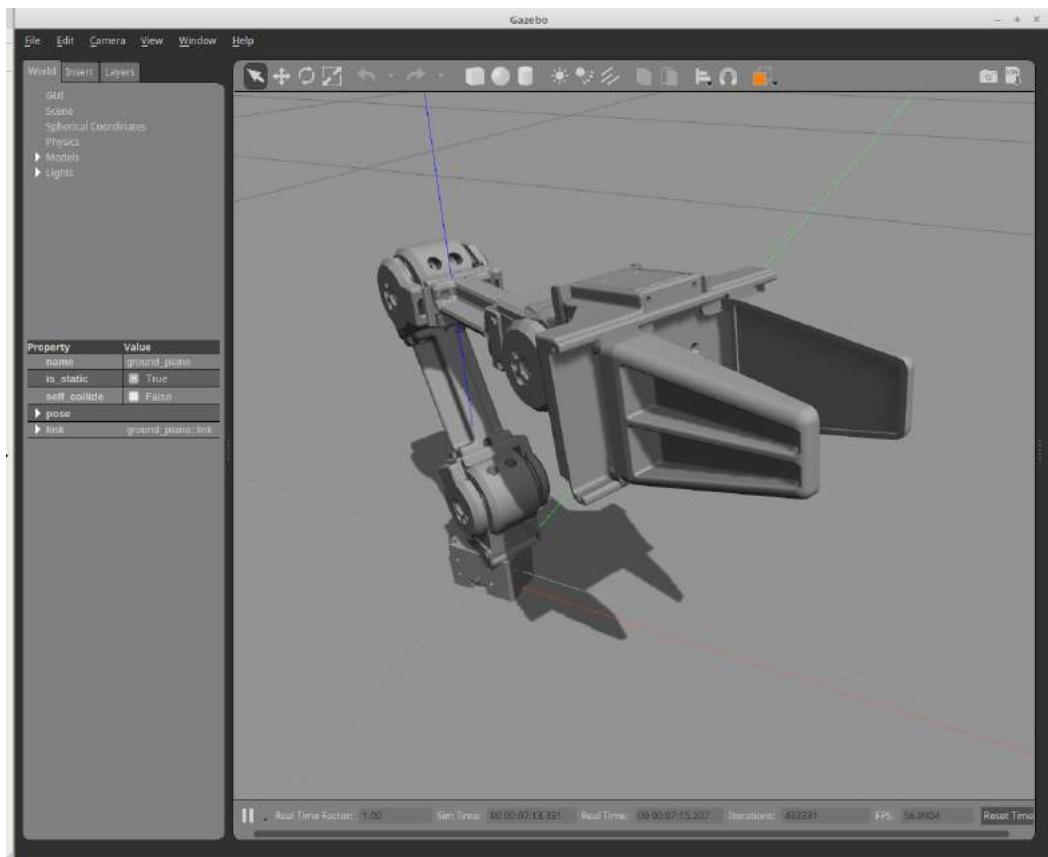


图13-38 在使用MoveIt!的Gazebo环境中进行抓取操作

13.4. 应用于实际平台

到目前为止，我已经使用MoveIt!来控制了Gazebo仿真器上的机械手臂。在本节中，我们将学习如何控制实际的OpenManipulator Chain，并进行实际操作。

13.4.1. 准备和控制OpenManipulator

OpenManipulator Chain由总共5个DYNAMIXEL X系列舵机和与其兼容的连接件和3D打印部件组成。用户可以自行购买DYNAMIXEL X和连接件，并从onshape⁵³下载3D打印设计文件。

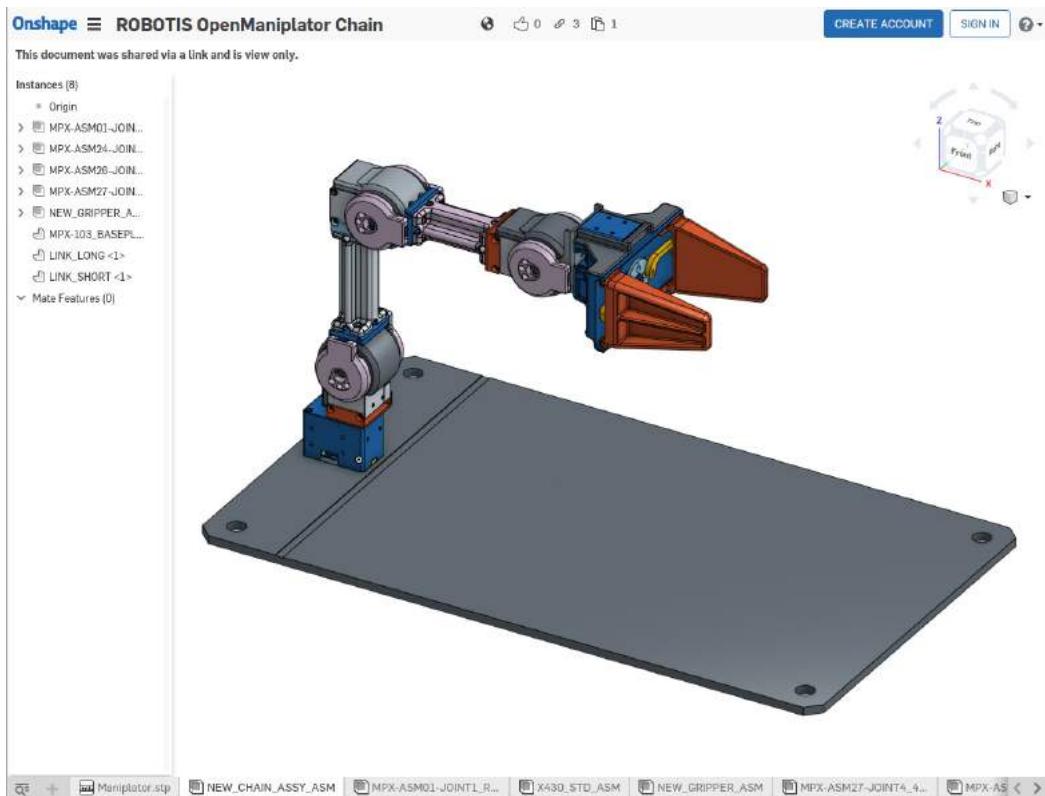


图13-39 上传到Onshape的OpenManipulator Chain装配文件

要在ROS上运行DYNAMIXEL，需要使用dynamixel_sdk功能包⁵⁴。dynamixel_sdk是ROBOTIS提供的DYNAMIXEL SDK中包含的一个ROS功能包，它提供了用于功能包通信的函数，因此有助于更轻松地控制DYNAMIXEL。

⁵³ <https://goo.gl/NsqJMu>

⁵⁴ http://wiki.ros.org/dynamixel_sdk

DYNAMIXEL SDK



图13-40 由ROBOTIS提供的DYNAMIXEL SDK

ROBOTIS提供的dynamixel_workbench⁵⁵元功能包通过single_manager功能包更容易地改变Dynamixel控制表的参数，同时也提供了一个合适的GUI。它还提供了一个通过toolbox库和controller功能包来控制ROS中的DYNAMIXEL的例子。

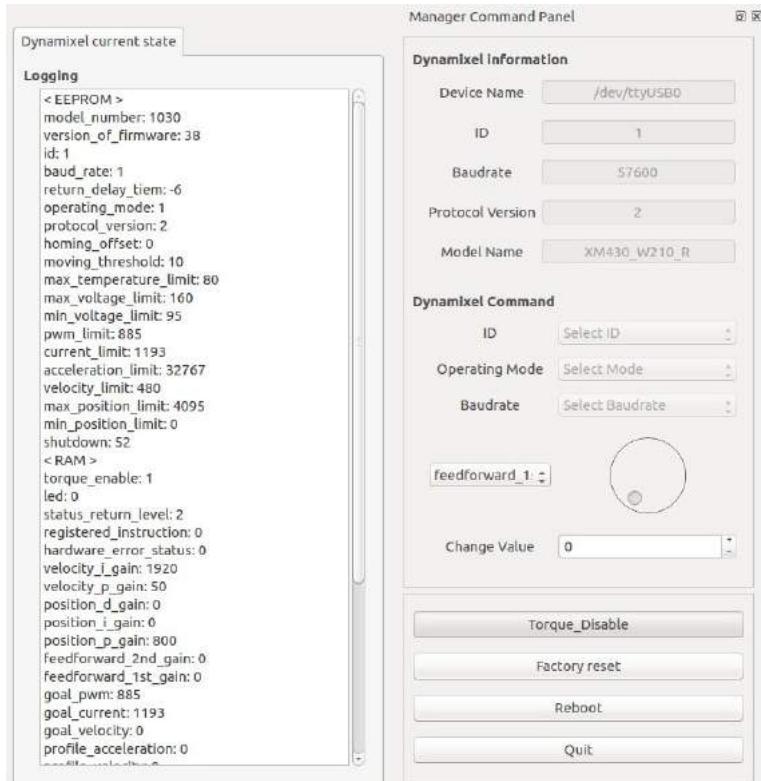


图13-41 ROSOTIS提供的ROS官方功能包之一，dynamixel-workbench

⁵⁵ http://wiki.ros.org/dynamixel_workbench

将准备好的5个Dynamixel的通信速度均设为1M（1000000）bps，并将工作模式都设置为位置控制模式，将ID分别指定为1至5，并参考OpenManipulator wiki和已发布的硬件（Onshape）信息开始组装。组装完成后，为了与OpenManipulator Chain进行通信，使用U2D2将通信方式转换为USB，并将U2D2连接至主计算机。电源采用12V5A输出的SMPS，可以通过SMPS2DYNAMIXEL给Dynamixel供电。

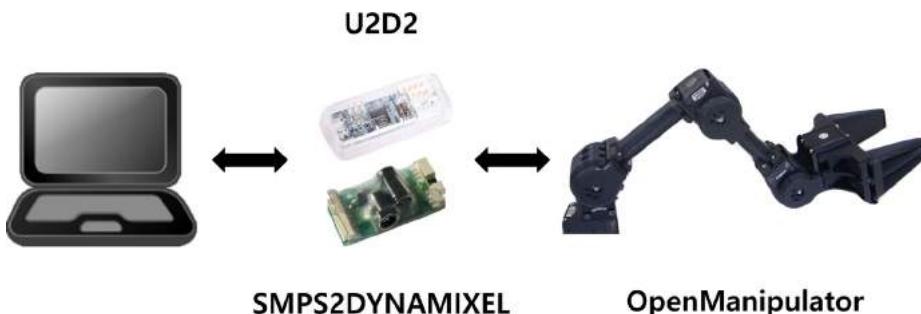


图13-42 运行OpenManipulator所需的配置和连接方法

连接完成后，在终端窗口中输入以下命令。在这里，chmod命令是为了使用设备而设置权限的命令。下面的例子是U2D2被识别为ttyUSB0的一个例子。

```
$ sudo chmod a+rw /dev/ttyUSB0  
$ roslaunch open_manipulator_dynamixel_ctrl dynamixel_controller.launch
```

当运行完成后，每个DYNAMIXEL都会输出扭力。我们来看看话题列表。

```
$ rostopic list  
/robotis/dynamixel/goal_states  
/robotis/dynamixel/present_states  
/rosout  
/rosout_agg
```

如上所述，通过话题消息通信可以监视每个Dynamixel的当前位置，并且可以以任意的角度值控制Dynamixel。

现在让我们通过MoveIt!来控制实际的机械手臂吧。

```
$ rosrun open_manipulator_moveit open_manipulator_demo.launch
```

在RViz窗口中，选择所需的运动规划库，输入终点坐标，然后单击Plan & Execute按钮就可以看到实际机器人正在移动。

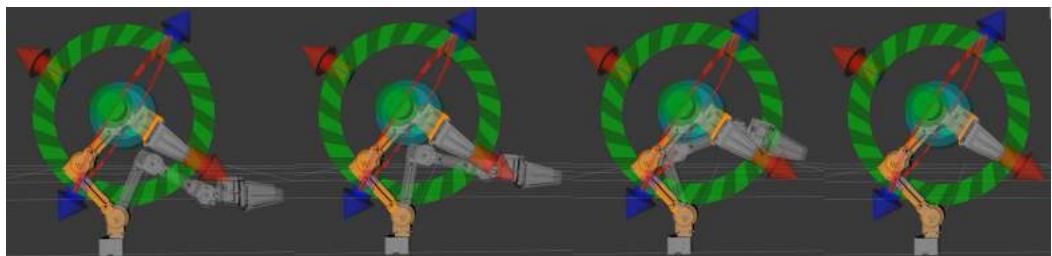


图13-43 使用MoveIt!的OpenManipulator Chain条运动规划。



图13-44 OpenManipulator Chain的运动

以下示例显示了13.3.2. MoveIt! Setup Assistant中说明的参考内容“指定机械手臂的目标姿态（位置+方向）的方式”中将position_only_ik项目设为true，并指定目标姿态的位置的例子。详细说明请参阅该参考内容。

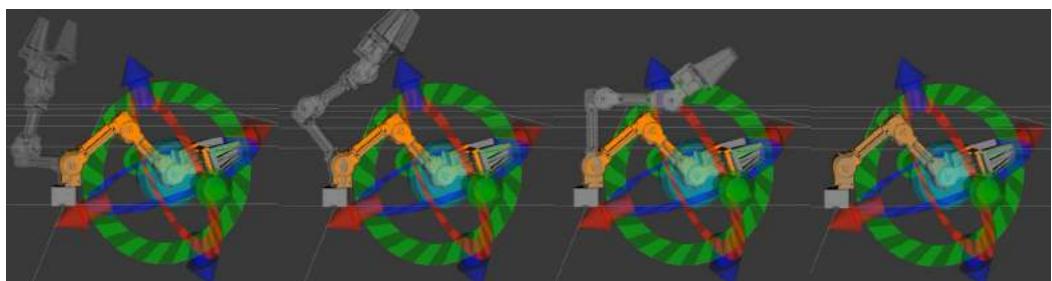


图13-45 利用MoveIt!的OpenManipulator Chain运动规划（Position IK Only）



图13-46 OpenManipulator Chain的运行

13.4.2. OpenManipulator与TurtleBot3 Waffle及Waffle Pi

OpenManipulator Chain具有与Turtlebot 3 Waffle和Waffle Pi兼容的优点。这可以弥补自由度不足，而且还能利用TurtleBot3 Waffle和Waffle Pi具有的SLAM和导航功能，因此可以提高完成度。

在本节中，我们将TurtleBot3 Waffle URDF添加到上面创建的open_manipulator_chain.xacro文件，并在RViz中观察。

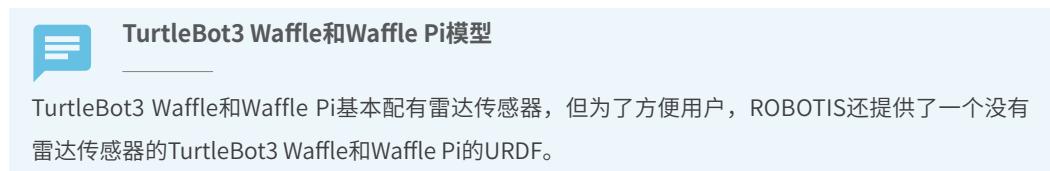
可以看到在turtlebot3_description目录中有一个保存有URDF文件的URDF目录。当创建OpenManipulator Chain的URDF文件时，笔者提到如果将其保存为xacro文件格式，则以后可以方便地重用。我们来看看open_manipulator_with_tb3功能包。

```
$ roscd open_manipulator_with_tb3/urdf  
$ gedit open_manipulator_chain_with_tb3.xacro
```

open_manipulator_with_tb3/urdf/open_manipulator_chain_with_tb3.xacro

```
<!-- Include TurtleBot3 Waffle URDF -->  
<xacro:include filename="$(find turtlebot3_description)/urdf/turtlebot3_waffle_naked.urdf.xacro" />  
  
<!-- Base fixed joint -->  
<joint name="base_fixed" type="fixed">  
  <origin xyz="-0.005 0.0 0.091" rpy="0 0 0"/>  
  <parent link="base_link"/>  
  <child link="link1"/>  
</joint>
```

如果您检查open_manipulator_with_tb3功能包的URDF文件，则可以看到在代码顶部有一段包含turtlebot3_waffle_naked.urdf.xacro文件的代码。通过这段代码，简单地加载了TurtleBot3的 Waffle后，在想要放置机械手臂的关节创建了固定关节并连接了机械手臂。



现在用下面的命令运行RViz。

```
$ roslaunch open_manipulator_with_tb3 open_manipulator_chain_with_tb3_rviz.launch
```

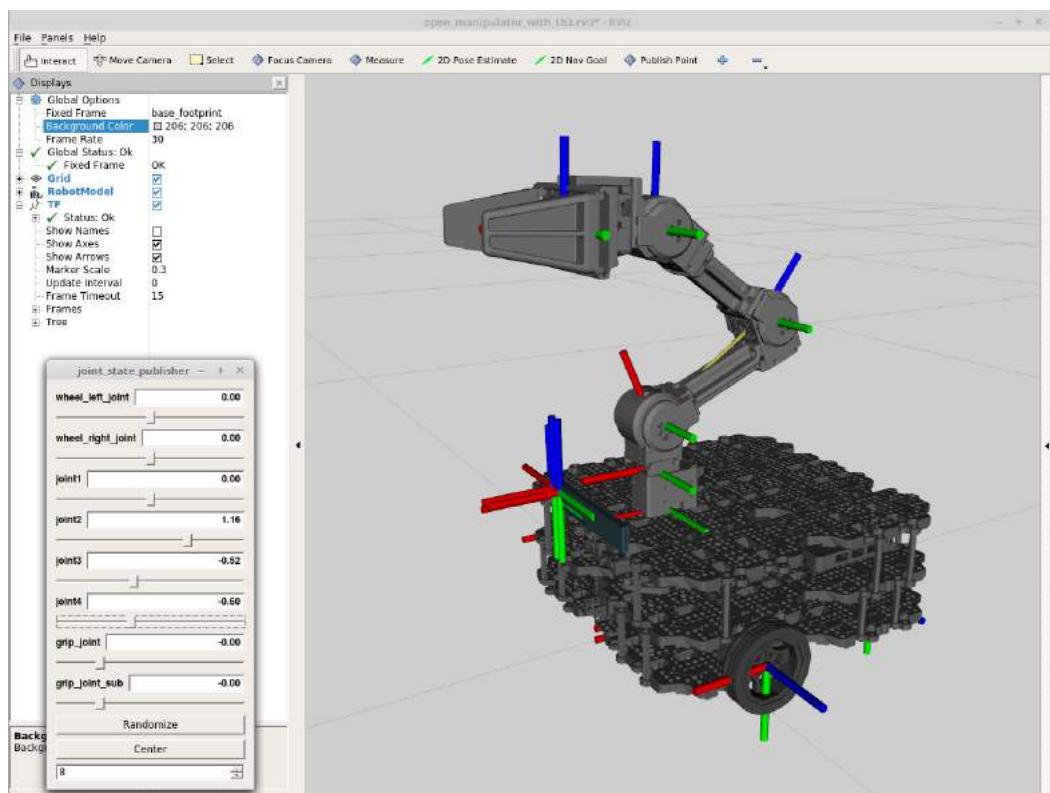


图13-47 上部结合了OpenManipulator Chain的TurtleBot3 Waffle

OpenManipulator Chain采用了模块化舵机Dynamixel，因此可以与各种机器人组合使用。建议利用xacro的可重用的优势，把OpenManipulator Chain安装在您自己制作的机器人上。

符号	
~ /	71
~ (波浪号)	65
__ (两个下划线)	66
--- (三个连字符)	64, 167, 177
--screen	292
/ (斜杠)	65
_ (一个下划线)	66
catkin_python_setup	85
CMake	75
CMakeLists.txt	49, 75, 79, 154, 165, 175
collision	416
costmap	357, 361
参数	45, 54, 185
参数服务器	45
测位	347
传感器	199
存储库	47
A	
ACML	346
action文件	64
AMCL	353, 354, 363
Android	396
APT	26
Arduino	244
ARM	238
安装	24
B	
bag	47, 327
bashrc	29
Button	267
Buzzer	255
C	
catkin	45, 75, 122
catkin_create_pkg	153
catkin_make	123
D	
DAE	416
DWA	346, 359, 365
Dynamixel	226, 257, 278, 456
dynamixel_sdk	456
dynamixel_workbench	457
单位	150
导航	312, 317, 342, 345, 361
导航推测	318
地图	318, 326, 328
地图服务器	352
电机	226
电压	269
订阅	43
订阅者	43, 57, 158
动作	44, 52
动作服务器	45, 177
动作客户端	45, 180
动作文件	176

F	I
feedback 59	IDE 33
发布 43	Ifconfig 31
发布者 43, 56, 157	IMU 240, 256, 271
仿真 301, 315	
仿真器 307	
服务 44, 52	
服务从机 369	
服务从节点 390	
服务服务器 44, 59, 167	
服务核心 368, 373	
服务客户端 44, 59, 168	
服务文件 166	
服务主机 369	
服务主节点 383	
G	J
Gazebo 307, 428, 452	Java 396
Gmapping 341	机器人 196
goal 59	机械手臂 405
GUI 130, 137, 422	建模 410
工具 5, 130, 137	节点 42
工作目录 27, 73	
功能包 42, 227	
关节 406, 418	
关节空间控制 407	
惯性 416	
H	K
话题 43, 51	Kinetic Kame 18, 25
	可重用性 5
	客户端库 13, 48, 68
I	L
	LDS 220
	LED 254, 265
	link 415
	Log 98
	LTS 19
	历史 15
	粒子滤波器 339, 341
	连杆 406

M	P
map 360	package.xml 49, 76, 153, 164, 174
map_saver 333	pgm 326, 347
map_server 313, 324, 326, 328	平台 2
MD5 48	
move_base 353, 356	
move_group 436	
MoveIt! 436	
MoveIt! Setup Assistant 437	
msg文件 63	
名称 48, 64	
命名 152	
命名空间 65, 66, 152, 192	
模型 352	
N	Q
namespace 372	Qt 33, 138
NTP 24	棋盘 210
ntpdate 24	卡尔曼滤波器 338
逆运动学 408	嵌入式系统 236
O	R
Odometry 303, 322	REP 151
OGM 329	result 59
Open Robotics 17	ROS 41
OpenCR 238	ROS_HOSTNAME 31, 207, 291
OpenManipulator 410, 460	ROS_MASTER_URI 31, 207, 291
OpenSLAM 341	rosbag 118, 147
OSRF 16	rosbuild 46
	roscd 95
	rosclean 100
	roscore 28, 36, 46, 55, 97
	roscpp 69
	rosdep 26, 127
	rosed 96
	rosinstall 27, 127
	roslaunch 46, 55, 99, 190
	roslocate 128
	rosls 96
	rosmg 114
	rosnode 102

S			
SDF	409		
Service Calle	172		
SLAM	312, 321, 328, 337		
SRDF	437		
srv文件	63		
SSH	294		
STL	416		
Switchyard	15		
社区	6		
T			
rospack	125	深度相机	214
rosparam	111	生态系统	6, 14
rospy	69	四元数	68, 319
rosrun	46, 55, 99		
rosserial	258, 262, 263, 265		
rosserial client	259		
rosserial server	259		
rosserial_python	259	TCP/IP	49
rosserial协议	260	TCPROS	49, 58
rosservice	108	teleoperation	291
rossrv	116	TF	67, 303, 347, 352
rostopic	104	Turtlebot3	
RPC	48	TurtleBot3	276, 283, 287, 288, 291, 294, 301, 307
rqt	138	turtlesim	37
rqt_bag	147	通信	5
rqt_graph	38, 47, 143		
rqt_image_view	142		
rqt_plot	145		
RViz	130, 218, 223, 275, 301, 302		
RViz显示屏	136	U	
任务空间控制	407	U2D2	458
日志	118, 147	URDF	409, 410, 437
		urdf_to_graphviz	419
		URI	48
V			
visual	416		
W			
Wiki	47		
维基	93		

X

XML	49, 414
XMLRPC	49
相机	201
相机校准	208
消息	43, 60, 156
消息通信	50
消息文件	156

Y

yaml	112, 209, 326, 347, 446
元操作系统	10
元功能包	43

Z

占用网格地图	329
正向运动学	407
重新映射	65
主节点	41, 54
姿态	318, 319
自由度	408, 447
坐标表现方式	151

ROS 机器人编程

从基本概念到机器人应用程序编程实战！

- ROS Kinetic Kame: 基本概念、讲解和工具
- 在ROS中使用传感器功能包和电机功能包
- 专为ROS而设计的嵌入式控制板: OpenCR1.0
- 利用TurtleBot3实现SLAM与导航
- 用ROS Java编写配送机器人程序
- 使用MoveIt!和Gazebo来仿真OpenManipulator机械手臂

本手册的目标读者

- 希望学习基于ROS (Robot Operating System) 的机器人编程的大学生和研究生
- 想采用ROS的专业研究者和工程师

我们尽最大努力提供了我们在开发和应用TurtleBot3和OpenManipulator的过程中所掌握到的详细的信息。我们希望这本书能成为ROS初学者和将来为开源机器人领域做出贡献的众多研究者和开发者的完全手册。



ROS Official Robot Platform
TURTLEBOT3 Series

ROBOTIS CO., LTD.

<http://www.robotis.com>

<http://www.turtlebot.com>

<http://turtlebot3.robotis.com>

ROBOTIS

ISBN 979-11-962307-2-2

