A Gentle Introduction to ROS

Chapter: Introduction

Jason M. O’Kane

Jason M. O'Kane
University of South Carolina
Department of Computer Science and Engineering
315 Main Street
Columbia, SC 29208

`http://www.cse.sc.edu/~jokane`

This is version 2.1.6 `(ab984b3)`, generated on April 24, 2018.

Typeset by the author using LaTeX and `memoir.cls`.

ISBN 978-14-92143-23-9

Chapter 1

Introduction

In which we introduce ROS, describe how it can be useful, and preview the remainder of the book.

1.1 Why ROS?

The robotics community has made impressive progress in recent years. Reliable and inexpensive robot hardware—from land-based mobile robots, to quadrotor helicopters, to humanoids—is more widely available than ever before. Perhaps even more impressively, the community has also developed algorithms that help those robots run with increasing levels of autonomy.

In spite of (or, some might argue, because of) this rapid progress, robots do still present some significant challenges for software developers. This book introduces a software platform called **Robot Operating System**, or **ROS**,[1] that is intended to ease some of these difficulties. The official description of ROS is:

> *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.*[1]

[1] When spoken aloud, the name "ROS" is nearly always pronounced as a single word that rhymes with "moss," and almost never spelled out "arrr-oh-ess."

[1] `http://wiki.ros.org/ROS/Introduction`

This description is accurate—and it correctly emphasizes that ROS does not replace, but instead works alongside a traditional operating system—but it may leave you wondering what the real advantages are for software that uses ROS. After all, learning to use a new framework, particularly one as complex and diverse as ROS, can take quite a lot of time and mental energy, so one should be certain that the investment will be worthwhile. Here are a few specific issues in the development of software for robots that ROS can help to resolve.

Distributed computation Many modern robot systems rely on software that spans many different processes and runs across several different computers. For example:

- ☞ Some robots carry multiple computers, each of which controls a subset of the robot's sensors or actuators.
- ☞ Even within a single computer, it's often a good idea to divide the robot's software into small, stand-alone parts that cooperate to achieve the overall goal. This approach is sometimes called "complexity via composition."
- ☞ When multiple robots attempt to cooperate on a shared task, they often need to communicate with one another to coordinate their efforts.
- ☞ Human users often send commands to a robot from a laptop, a desktop computer, or mobile device. We can think of this human interface as an extension of the robot's software.

The common thread through all of these cases is a need for *communication* between multiple processes that may or may not live on the same computer. ROS provides two relatively simple, seamless mechanisms for this kind of communication. We'll discuss the details in Chapters 3 and 8.

Software reuse The rapid progress of robotics research has resulted in a growing collection of good algorithms for common tasks such as navigation, motion planning, mapping, and many others. Of course, the existence of these algorithms is only truly useful if there is a way to apply them in new contexts, without the need to reimplement each algorithm for each new system. ROS can help to prevent this kind of pain in at least two important ways.

- ☞ ROS's standard packages provide stable, debugged implementations of many important robotics algorithms.

- ☞ ROS's message passing interface is becoming a *de facto* standard for robot software interoperability, which means that ROS interfaces to both the latest hardware and to implementations of cutting edge algorithms are quite often available. For example, the ROS website lists hundreds of publicly-available ROS packages.[2] This sort of uniform interface greatly reduces the need to write "glue" code to connect existing parts.

As a result, developers that use ROS can expect—after, of course, climbing ROS's initial learning curve—to focus more time on experimenting with new ideas, and less time reinventing wheels.

Rapid testing One of the reasons that software development for robots is often more challenging than other kinds of development is that testing can be time consuming and error-prone. Physical robots may not always be available to work with, and when they are, the process is sometimes slow and finicky. Working with ROS provides two effective workarounds to this problem.

- ☞ Well-designed ROS systems separate the low-level direct control of the hardware and high-level processing and decision making into separate programs. Because of this separation, we can temporarily replace those low-level programs (and their corresponding hardware) with a simulator, to test the behavior of the high-level part of the system.
- ☞ ROS also provides a simple way to record and play back sensor data and other kinds of messages. This facility means that we can obtain more leverage from the time we do spend operating a physical robot. By recording the robot's sensor data, we can replay it many times to test different ways of processing that same data. In ROS parlance, these recordings are called "bags" and a tool called `rosbag` is used to record and replay them. See Chapter 9.

A crucial point for both of these features is that the change is seamless. Because the real robot, the simulator, and the bag playback mechanism can all provide identical (or at least very similar) interfaces, your software does not need to be modified to operate in these distinct scenarios, and indeed need not even "know" whether it is talking to a real robot or to something else.

Of course, ROS is not the only platform that offers these capabilities. What is unique about ROS, at least in the author's judgment, is the level of widespread support for ROS

[2] `http://www.ros.org/browse`

across the robotics community. This “critical mass” of support makes it reasonable to predict that ROS will continue to evolve, expand, and improve in the future.

ROS is not . . . Finally, let’s take a moment to review a few things that are *not* true about ROS.

☞ *ROS is not a programming language.* In fact, ROS programs are routinely written in C++,[3] and this book has some explicit instructions on how to do that. Client libraries are also available for Python,[4] Java,[5] Lisp,[6] and a handful of other languages.[7]

☞ *ROS is not (only) a library.* Although ROS does include client libraries, it also includes (among other things), a central server, a set of command-line tools, a set of graphical tools, and a build system.

☞ *ROS is not an integrated development environment.* Although ROS does not prescribe any particular development environment, it can be used with most popular IDEs.[8] It is also quite reasonable (and, indeed, it is the author’s personal preference) to use ROS from a text editor and the command line, without any IDE.

1.2 What to expect

The goal of this book is to provide an integrated overview of the concepts and techniques you’ll need to know to write ROS software. This goal places a few important constraints on the content of the book.

☞ *This is not an introduction to programming.* We won’t discuss basic programming concepts in any great detail. This book assumes that you’ve studied C++ in sufficient depth to read, write, and understand code in that language.

[3] `http://wiki.ros.org/roscpp`

[4] `http://wiki.ros.org/rospy`

[5] `http://wiki.ros.org/rosjava`

[6] `http://wiki.ros.org/roslisp`

[7] `http://wiki.ros.org/Client Libraries`

[8] `http://wiki.ros.org/IDEs`

☞ *This is not a reference manual.* There is plenty of detailed information about ROS, including both tutorials[9] and exhaustive reference material,[10] available online. This book makes no attempt to replace those resources. Instead, we present a selected subset of ROS features that, in the author's view, represents a useful starting point for using ROS.

☞ *This is not a textbook on robotics algorithms.* The study of robots, especially the study of algorithms for controlling autonomous robots, can be quite fascinating. A dizzying variety of algorithms have been developed for various parts of this problem. This book will not teach you any of those algorithms.[2] Our focus is on a specific tool, namely ROS, that can ease the implementation and testing of those algorithms.

Chapters and dependencies Figure 1.1 shows the organization of the book. Chapters are shown as rectangles; arrows show the major dependencies between them. It should be fairly reasonable to read this book in any order that follows those constraints.

Intended audience This book should be useful for both students in robotics courses and for researchers or hobbyists that want to get a quick start with ROS. We'll assume that readers are comfortable with Linux (including tasks like using the command line, installing software, editing files, and setting environment variables), are familiar with C++, and want to write software to control robots. Generally, we'll assume that you are using Ubuntu Linux 14.04 (the newest version that is, at this writing, officially supported) and the `bash` shell. However, there are relatively few instances where these choices matter; other Linux distributions (especially those based on `deb` packages) and other shells will not usually be problematic.

[2] . . . but you should learn them anyway.

[9] `http://wiki.ros.org/ROS/Tutorials`
[10] `http://wiki.ros.org/APIs`

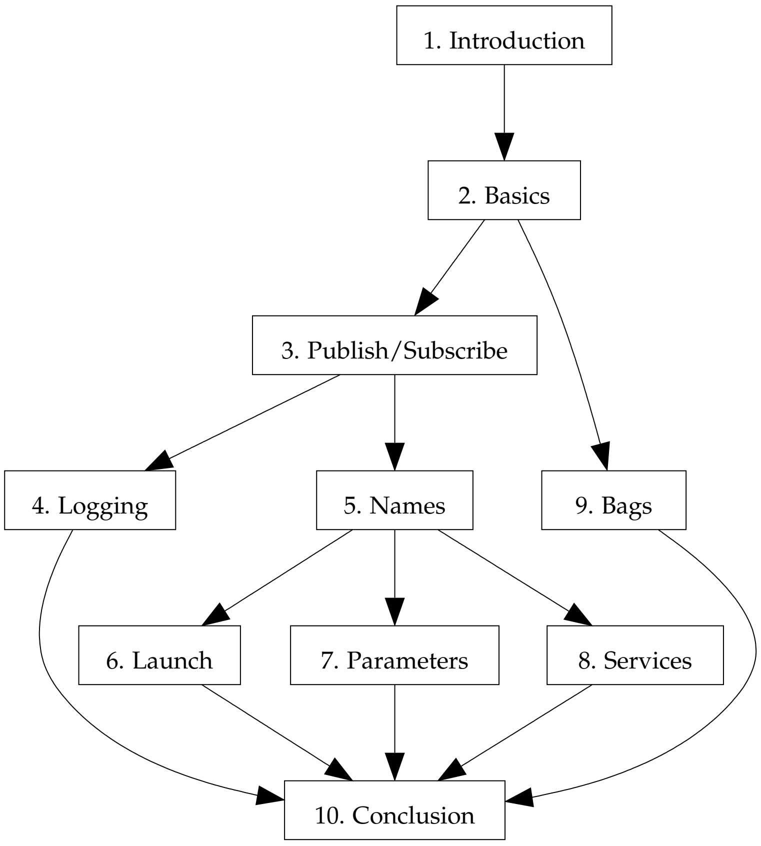

Figure 1.1: Dependencies between chapters.

1.3 Conventions

Throughout the book, we'll attempt to anticipate some of the most common sources of problems. These kinds of warnings, which are worthy of your attention, especially if things are not working as expected, are marked like this:

This "dangerous bend" sign indicates a common source of problems.

In addition, some sections include explanations that will be of interest to some readers, but are not crucial for understanding the concepts at hand. These comments are marked like this:

⏭ *This "fast forward" symbol indicates information that can be safely skipped, especially on a first reading.*

1.4 Getting more information

As alluded to above, this book makes no attempt to be a comprehensive reference for ROS. It's all but certain that you will need additional details to do anything interesting. Fortunately, online information about ROS is abundant.

☞ Most importantly, the developers of ROS maintain extensive documentation,[11] including a set of tutorials. This book includes links, each marked with a ⌕, to many of the corresponding pages in this documentation. If you are reading an electronic version of the book in a reasonably modern PDF viewer, you should be able to click these links directly to open them in your browser.

☞ When unexpected things happen—and chances are quite good that they will—there is a question and answer site (in the style of Stack Exchange) devoted to ROS.[12]

[11] `http://wiki.ros.org`
[12] `http://answers.ros.org`

☞ It may also be valuable to subscribe to the ros-users mailing list,[13] on which announcements sometimes appear.

Here are two important details that will help you make sense of some of the documentation, but are not always fully explained in context there.

Distributions Major versions of ROS are called **distributions**, and are named using adjectives that start with with successive letters of the alphabet.[14] (This is, for comparison, very similar to the naming schemes used for other large software projects, including Ubuntu and Android.) At the time of this writing, the current distribution is indigo. The next distribution, named jade, is due in May 2015.[15] Older distributions include hydro, groovy, fuerte, electric, diamondback, C Turtle, and box turtle. These distribution names appear in many places throughout the documentation.

To keep things as simple and up-to-date as possible, this book assumes that you are using indigo.

⏭ *If, for some reason, you need to use* hydro *instead of* indigo, *nearly all of the book's content still applies without modification.*

The same is true for groovy *as well, with one important exception: In distributions newer than* groovy *(and, therefore, in this book), velocity commands for the* turtlesim *simulator have been changed to use a standard message type and topic name that happen to be shared with many real mobile robots.*

Distribution	***Topic name***	***Message type***
groovy	/turtle1/command_velocity	turtlesim/Velocity
indigo, hydro	/turtle1/cmd_vel	geometry_msgs/Twist

This change has a few practical implications:

☞ *When adding dependencies to your package (see page 44), you'll need a dependency on* turtlesim, *instead of on* geometry_msgs.

[13] http://lists.ros.org/mailman/listinfo/ros-users

[14] http://wiki.ros.org/Distributions

[15] http://wiki.ros.org/jade

☞ *The relevant header file (see page 49) is*

    turtlesim/Velocity.h

rather than

    geometry_msgs/Twist.h

☞ *The* `turtlesim/Velocity` *message type has only two fields, called* `linear` *and* `angular`*. These fields play the same roles as the* `linear.x` *and* `angular.z` *fields of* `geometry_msgs/Twist`*. This change applies both on the command line (see page 31) and in C++ code (see page 51).*

Build systems Starting with the `groovy` distribution, ROS made some major changes to the way software is compiled. Older, pre-`groovy` distributions used a build system called `rosbuild`, but more recent versions have begun to replace `rosbuild` with a new build system called `catkin`. It is important to know about this change because a few of the tutorials have separate versions, depending on whether you're using `rosbuild` or `catkin`. These separate versions are selected using a pair of buttons near the top of the tutorial. This book describes `catkin`, but there may be some cases in which `rosbuild` is a better choice.[16]

1.5 Looking forward

In the next chapter, we'll get started working with ROS, learning some basic concepts and tools.

[16] `http://wiki.ros.org/catkin_or_rosbuild`