A Gentle Introduction to ROS

Chapter: Conclusion

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

©2014, Jason Matthew O'Kane. All rights reserved.

This is version 2.1.3 (5776342), generated on April 20, 2016.

Typeset by the author using LaTeX and memoir.cls.

ISBN 978-14-92143-23-9

Chapter 10

Conclusion

In which we preview some additional topics.

In the preceding chapters, we've looked into the basic workings of ROS in some detail. We saw examples of most of those concepts using the `turtlesim` simulator. Of course, it is quite unlikely that your original interest in ROS was based on a desire to drive imaginary turtles around. If this book has done its job, you should be ready to start using ROS—certainly with the help of the documentation, and quite likely with the help of existing packages—to solve the real robotics problems that you care about.

10.1 What next?

This section contains short previews (with links to the relevant documentation) of a few additional topics that, although we have not covered them here, do occur quite commonly in real ROS systems.

Running ROS over a network You might remember from Chapter 1 that one of the advantages of ROS is that it facilitates distributed operation of robots, in which many different programs running on multiple computers can interact with each other. However, throughout this book, the entire ROS system has been contained on a single computer.

To use ROS across a network of multiple computers requires configuration both at the network level (to ensure that all of the computers can talk to each other) and at the ROS

level (to ensure that all of nodes can communicate with the master).[1][2][3] The good news is that, once you have configured things correctly, ROS will take care of the details of network communication. Nodes on different machines can communicate seamlessly, using precisely the same methods that we've been using on a single machine.

Writing cleaner programs The source code for the example programs in this book is primarily optimized for brevity and clarity, rather than for extensibility and maintainability. In fact, several guidelines are often suggested to write "cleaner" programs that we have not obeyed in this book. For example, some developers suggest the use of `ros::Timer` callbacks instead of `ros::Rate` objects.[4] Some developers also prefer to reduce the number of global variables and functions by encapsulating all or part of a node's data in a class, using methods of that class as callbacks.[5] The payoff from these kinds of techniques tends to increase as the size and complexity of the program grows.

Visualizing data with `rviz` Working with `turtlesim`, nearly all of the data in our messages dealt with relatively simple information such as two-dimensional positions, orientations, and velocities. In contrast, real robots are often substantially more complicated, and none of the techniques we've learned in this book are really suitable for viewing the complex and noisy data that they typically produce. To fill this gap, ROS provides a graphical tool called `rviz` that can display a wide variety of information—naturally, by subscribing to appropriately-typed topics that a user selects—about how the robot itself is operating.[6]

Creating message and service types The examples in this book have all relied exclusively on existing data types for messages and services. However, it is also straightforward to create new data types that belong to our own packages.[7][8]

Managing coordinate frames with `tf` Because robots operate in the physical world, it is very natural to use coordinates to describe the positions of various parts of the robot, along with objects the robot would like to avoid or interact with. Therefore, it becomes

[1] http://wiki.ros.org/ROS/NetworkSetup
[2] http://wiki.ros.org/ROS/Tutorials/MultipleMachines
[3] http://wiki.ros.org/ROS/EnvironmentVariables
[4] http://wiki.ros.org/roscpp/Overview/Timers
[5] http://wiki.ros.org/roscpp_tutorials/Tutorials/UsingClassMethodsAsCallbacks
[6] http://wiki.ros.org/rviz
[7] http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv
[8] http://wiki.ros.org/ROS/Tutorials/DefiningCustomMessages

crucial to keep careful track of the **coordinate frames** in which those coordinates are expressed. Many message types include a `frame_id` field that identifies the coordinate frame in which that message's data are expressed.

To make sense of those different coordinate frames, we need to know they are related to one another. Specifically, we would often like to know the **transformation** that can convert coordinates from one frame to another. ROS provides a standard package called `tf`—short for "transformation"—whose job is to enable nodes to utilize information about those kinds of transforms.[9][10][11][12] The `tf` package is designed to work robustly, even when the transformation data is distributed across many nodes, and even when the transformations are changing over time.

Simulating with Gazebo One of the biggest advantages of the modular software design that ROS encourages is that we can easily "swap out" various components of a working system, in order to reduce development time and make testing easier. Chapter 9 described one example of this capability, in which we can temporarily replace one or more nodes with a recorded bag of the messages those nodes published. Another more powerful option is to use **Gazebo**, which is a high-fidelity robot simulator.[13] Using Gazebo, we can define the characteristics of both our robot (or robots) and the world, and interact with that robot, via ROS, in the same way that we would interact with the real thing.

10.2 Looking forward

That's the end of our gentle introduction to ROS. The author sincerely hopes, however, this is only the very beginning of your journey using ROS to create a new generation of smarter, more capable robots.

[9] `http://wiki.ros.org/tf/Tutorials/Introduction to tf`
[10] `http://wiki.ros.org/tf/Tutorials/Writing a tf listener (C++)`
[11] `http://wiki.ros.org/tf/Overview/Data Types`
[12] `http://ros.org/doc/indigo/api/tf/html/c++/`
[13] `http://gazebosim.org/wiki/Tutorials/1.9/Overview_of_new_ROS_integration`