# Welcome to ROS Topics

*Steve Cousins*

Robot operating system (ROS) is a free and open-source system that has grown out of a novel collaboration between industry and academia. This column is designed to introduce you and help track this important community effort. The latest information about ROS will always be available on the Web (http://ros.org). My goal is to help you decide whether or not to download and try the system. In future columns, I'll write about the latest developments in ROS and its progress in the ROS community.

## Why ROS?

ROS is an open source-software platform designed to support a new generation of personal robots. Personal robots move around in a human environment and interact with the same objects people use. These robots are sometimes called *service robots* or *mobile manipulators*. Think of Rosie from *The Jetsons*, but without the attitude. Robots in this class today include the PR2 from Willow Garage, the HRP2 from Kawada Industries, and a number of academic prototypes such as the STAIR robots from Stanford University, HERB from Intel/CMU, and El-E from Georgia Tech.

Personal robotics research builds on many other subfields of robotics. Because these robots move around in the world, they make use of the research results from mobile robotics for the past 20 years, including SLAM and many navigation algorithms. Robot perception (including much of computer vision) is required to make sense of the world around the robot. Motion planning is required to compute safe trajectories for the arms and end effectors. To manipulate objects, grasp analysis and planning are required, as well as reasoning about object properties (e.g., keep the cup upright or don't squeeze the egg too hard).

This class of robots poses interesting software design challenges. The breadth of expertise necessary to program a personal robot is beyond the capacity of a single researcher, and it is therefore necessary to simplify the integration of different software libraries from different institutions. Perception and planning are computationally expensive, so supporting many processors is a requirement. Robotics is a challenging systems integration problem and requires a rich set of tools to successfully manage the complexity. Personal robotics software is necessarily complex, so the software system requires modern software engineering techniques such as continuous testing and integration to be successful. Finally, since robots operate in the real world, the system must support efficient communication between its components and be able to support real-time components. ROS provides an approach for each of these challenges.

*Digital Object Identifier 10.1109/MRA.2010.935808*

ROS is available for commercial as well as noncommercial use. The software is licensed under BSD or Apache 2.0 licenses, and the system is designed to be able to include components written under various GNU licenses as well. Every package in ROS is clearly labeled with its licensing terms so that researchers and developers can immediately know which components can be incorporated into their work.

## What Is ROS?

ROS is a thin, message-based, tool-based system designed for mobile manipulators. The system is composed of reusable libraries that are designed to work independently. The libraries are wrapped with a thin message-passing layer that enables them to be used by and make use of other ROS nodes. Messages are passed peer to peer and are not based on a specific programming language; nodes can be written in C++, Python, C, LISP, Octave, or any other language for which someone has written a ROS wrapper. ROS is based on a Unix-like philosophy of building many small tools that are designed to work together (more on that in a bit). ROS grows out of a collaboration between industry and academia and is a novel blend of professional software development practices and the latest research results.

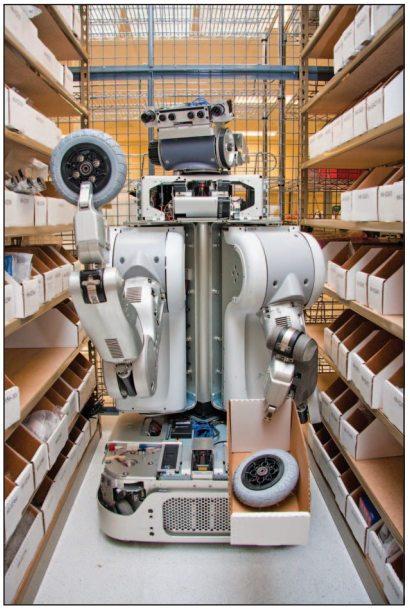
Software libraries and ROS nodes are organized into packages, stacks, and ultimately Apps. Packages can contain anything: libraries, nodes, message definitions, or tools. Each package should have enough functionality to be useful but not so much as to make it heavyweight. Stacks collect sets of packages that together provide useful functionality. Examples of stacks are ros_core, which contains the basic infrastructure of ROS, and navigation, which was used to make the PR2 autonomously travel around an office building for 26.2 mi (a marathon). The navigation stack has also been ported to other platforms such as the HRP2 at the JSK laboratory in Tokyo. Applications are similar to stacks but package up an executable robot program instead of just a library of reusable functionality.

A ROS system is a computation graph consisting of a set of nodes communicating with one another over edges. The communication consists of messages that are organized by topics. ROS contains tools for inspecting the graph and monitoring what is being said by node or by topic. One of the basic tools of ROS, rostopic, allows a command-line user to see what is being said about a topic, how frequently messages are being published, etc. That's where this column gets its name.

In addition to rostopic, ROS contains many useful tools. There is a set of tools for finding or creating packages, resolving dependencies, and compiling them. There are tools for visualizing the running system and graphing the output of nodes in

the system. ROS contains a very powerful visualizer, based on Ogre, that can display a three-dimensional (3-D) rendering of what the robot is perceiving in its environment. All of the messages in the system, including output from cameras and other sensors and all motor-control commands, can be logged and played back later using record and playback tools provided.

ROS is designed to help researchers leverage one another's work, so it has tools to support multisite collaborative development. The ROS-built system supports a federated development model, where organizations can make code available in repositories, and other organizations can easily incorporate components from those repositories into their own work. The basic ROS tools make it easy to locate, download, compile, and integrate code from other sites in the community.



Large distributed software projects require discipline to avoid exponential complexity, but software engineering process can be at odds with flexible, open-community development. ROS uses two key software engineering techniques to manage the complexity: testing and releases. Automated software tests are run every time a change is checked in, and notifications are automatically sent when the tests fail. When improvements in one part of the system break another, tests can help warn developers that their changes may have unintended consequences. Releases help ensure that the stability of the code base by locking down application programming interfaces (APIs) and verifying that documentation, tutorials, tests, and sample code are in place. Researchers can choose to live on the bleeding edge with the latest code or use a recent stable release of ROS stacks.

## Getting Started with ROS

To get started using ROS, just go to http://ros.org and follow the installation instructions and getting started the links. As of this writing, there are around 175 tutorials documenting the released stacks. The getting started pages will guide you through the initial tutorials. If you begin with an Ubuntu or other supported Linux system, you should be up and running in well under an hour.

The initial release, ROS 1.0, contains around 50 stacks and hundreds of packages, so spending some time understanding what's there could help you to avoid reinventing the wheel. As of this writing, there are 15 repositories from around the world in the federation. These packages are not part of the formal release but have a wide range of useful functionality. ROS has an active mailing list called ROS users that new users should subscribe to. The community is very supportive of new users.

## ROS Community

The ROS community began with a core group of developers at Stanford and Willow Garage. Morgan Quigley, Brian Gerkey, Ken Conley, and Eric Berger had all worked previously on software systems for distributed systems or robotics (or both). The original ROS core team included Jeremy Leibs, Tully Foote, Josh Faust, and Rob Wheeler. By now, the ROS community includes almost 40 software developers at Willow Garage, numerous members of the academic robotics community at dozens of institutions, and researchers at other companies, notably, Intel and Bosch.

***Address for Correspondence:*** Steve Cousins, Willow Garage, Inc., 68 Willow Road, Menlo Park, CA 94025 USA. E-mail: cousins@willowgarage.com.