Robot Operating System (ROS)
A quick overview

Dr. Ramviyas Nattanmai Parasuraman, UGA

CSCI/ATRI 4530/6530 Introduction to Robotics

What ROS is NOT?

Not an Operating System

Not an IDE (Integrated Development Environment)

Not just a collection of libraries

Not a programming language

What ROS is then?

A flexible **framework** for writing and sharing robot software!

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 computer**

- **http://wiki.ros.org/ROS/Introduction**

Why ROS?

Distributed computation

Software Reuse

Language independence (C++, Python, Lisp at present. Java and Lua in future)

Rapid implementation and testing

Contributing and exploiting amazing works in the community

What ROS does

Message Passing

Debugging tools

Visualization tools

Software Management (organization, compiling, packaging)

Libraries

Hardware Abstraction for all of these items

ROS Concepts

ROS Filesystem: Packages, workspaces, messages, services descriptions

ROS Computation: Master, Nodes, Topics, Services, Parameters

ROS Community: Distributions, Repositories, etc.

ROS command-line tools

Packages are a very central concept to how files in ROS are organized, so there are quite a few tools in ROS that help you manage them. This includes:

rospack: find and retrieve information about packages

roscd: navigate to packages or folders within ROS system

catkin_create_pkg: create a new package

catkin_make: build a workspace of packages

rosdep: install system dependencies of a package

rqt: In rqt there is a plugin called "Introspection/Package Graph", which visualizes package dependencies as a graph

ROS Packages

Software in ROS is organized in *packages*. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module.

ROS packages tend to follow a common structure.

ROS Packages

Here are some of the directories and files you may notice.

include/package_name: C++ include headers (make sure to export in the CMakeLists.txt)

msg/: Folder containing Message (msg) types

src/package_name/: Source files, especially Python source that are exported to other packages.

srv/: Folder containing Service (srv) types

scripts/: executable scripts

CMakeLists.txt: CMake build file (see catkin/CMakeLists.txt)

package.xml: Package catkin/package.xml

CHANGELOG.rst: Many packages will define a changelog which can be automatically injected into binary packaging and into the wiki page for the package

ROS Master

The Master has *registration* APIs, which allow nodes to **register** as publishers, subscribers, and service providers.

The Master has a *URI* and is stored in the ROS_MASTER_URI environment variable. This URI corresponds to the host:port of the XML-RPC server it is running. By default, the Master will bind to port 11311.

Nodes

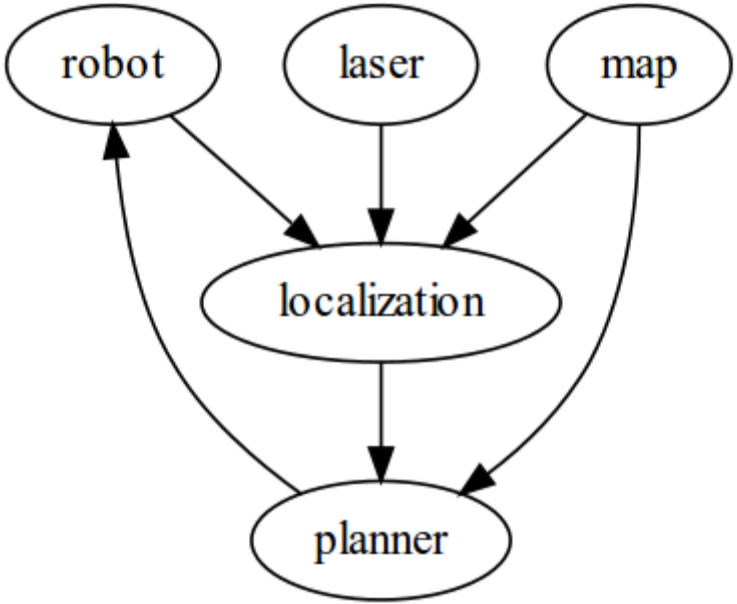
A ROS node has several APIs:

1. A *slave* API. The slave API is an XMLRPC API that has two roles: receiving callbacks from the Master, and negotiating connections with other nodes.

2. A topic transport protocol implementation (see TCPROS and UDPROS). Nodes establish topic connections with each other using an agreed protocol. The most general protocol is TCPROS, which uses persistent, stateful TCP/IP socket connections.

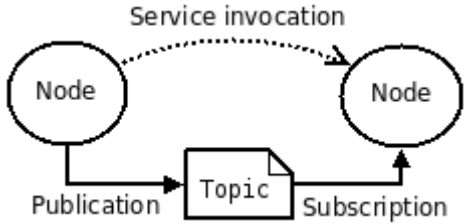
3. A command-line API. Every node should support command-line remapping arguments, which enable names within a node to be configured at runtime.

Nodes



Messages and Topics : publish/subscribe model

Nodes communicate with each other by publishing **messages** to topics



Topics are named buses over which nodes exchange messages.

Topics have anonymous publish/subscribe semantics,

Message types use standard ROS naming conventions: e.g. std_msgs/String

***rosmsg* - tool to deal with ros messages : definitions, type, etc.**

***rostopic* - tool to list, see and details on ros topics currently available**

Services

Remote procedure call.

A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply.

rossrv: displays information about .srv data structures. See rossrv for documentation on how to use this tool.

rosservice: lists and queries ROS Services

Topics

Topics are named buses over which nodes exchange messages

Topics have anonymous publish/subscribe semantics

Parameter Server

Part of ROS Master

Used to set/get/reset parameters passed across the ROS system

The Parameter Server uses a dictionary-of-dictionary representation for namespaces. E.g. /ns1/foo = 1

ROS Cheat Sheet

https://github.com/ros/cheatsheet/releases/download/0.0.1/ROScheatsheet_catkin.pdf

References

www.ros.org

A Gentle Introduction to ROS: https://cse.sc.edu/~jokane/agitr/

ROS: RSS Technical Lecture 6:
http://courses.csail.mit.edu/6.141/spring2012/pub/lectures/Lec06-ROS.pdf