ROS Overview
Prepared by Nat Tuck

What? Why?
ROS is a set of conventions and tools for building complex robot control systems.
In a system built on ROS, the primary building block of the system is the “node”. A bunch of nodes working together concurrently make up a complete ROS-based robot control system.

A node is a Linux process that performs some logically separable task. Nodes communicate by message passing using a well defined protocol. This is primarily intended for isolation and concurrency, but it has the neat side effect of providing programming language interoperability—a node written in COBOL will work fine (and you'll never even know) as long as it does the message passing correctly.

In order for it to be possible to build systems out of nodes, ROS provides some services and conventions:

• A file system structure.
  ○ Nodes live in “packages”, which optionally live in “stacks”.
  ○ A tool is provided, called “rosrun”, that will run a node based on its package and executable name.
  ○ A tool called “roslaunch” can be used to start many nodes at once, making it possible to start a complete system with one command.

• A message passing system.
  ○ Messages are passed via named “topics”.
  ○ Topics are many-to-many mechanism.
  ○ Example:
    ▪ A mapping node periodically publishes a PNG image to “/map”.
    ▪ A pathfinding node listens to “/map” and computes a path whenever a new map is published.
    ▪ A visualization node also listens to “/map” and displays the lastest image in a GUI window.
  ○ If messages get dropped because nobody is listening or the listeners are too busy, that's fine.

• A services system.
  ○ As an alternative to topics, nodes can provide services.
  ○ This is a one to one mechanism where one node makes a request of
another and then expects a response.

- A parameter storage mechanism.
  - This provides a place to put parameters that may need to be tweaked.
  - Can be used for dynamic reconfiguration at runtime.

- A message data type definition system.
  - This allows both messages on topics and services to be efficient, strongly typed.
  - Enables programming language interoperability.

- Network transparency.
  - Can put nodes on separate machines; ROS will happily pass messages across the network.
  - Uncompressed video wants Gigabit ethernet.

- Full system recording and playback.
  - Everything that happens in a running system can be recorded and played back later.
  - This lets you see what went wrong.
  - You can have some nodes be simulated by the playback, and others be running based on those messages. Consider playing back the output of a vision processing system during a live robot test run to test changes to control code.

- A build system with dependency management.
  - Lets you link to or depend on libraries provided by ROS packages.

- Existing code.
  - ROS packages a bunch of existing nodes and libraries.

- Tools to query and visualize the running system.

- Simulators: Stage for 2D and Gazebo for 3D environments

What code exists?

- Drivers for a bunch of robot-related hardware.
- Time dependent coordinate tracking and position transforms.
- Libraries and nodes for data processing.
  - Computer vision, point cloud manipulation
- A large set of existing message types appropriate for robots. Using these messages allows for interoperability of separately written code.
- All kinds of other stuff, of varying quality and generality.