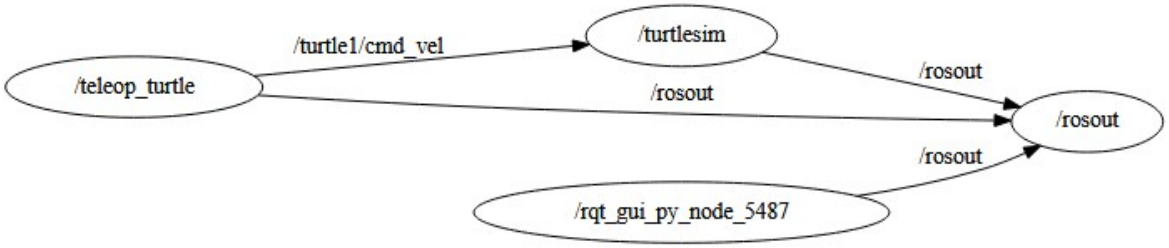
2. ROS core components

1. Communication infrastructure

ROS applications are composed of several processes loosely coupled, connected as nodes of a peer-to-peer network (the ROS runtime graph) using different types of strongly-typed communication.

A node is a process that performs computation.
- Gives additional fault tolerance as crashes are isolated to individual nodes.
- Expose a minimal API to the rest of the graph.
- Have a graph resource name that uniquely identify them to the rest.

A message is a simple data structure.
- Nodes communicate with each other using messages.
- Standard primitive types are supported (integer, floating point, boolean, …).



2. ROS core components

1. Communication infrastructure

At the lowest level, ROS offers a message passing interface that provides inter-process communication. The ROS middleware provides:

- Publish/subscribe anonymous message passing

  ROS's built-in and well-tested messaging system saves you time by managing the details of communication between distributed nodes via the anonymous publish/subscribe mechanism.

  Topics are named buses over which nodes exchange messages.

- Recording and playback of messages

  Because the publish/subscribe system is anonymous and asynchronous, the data can be easily captured and replayed without any changes to code.

- Request/response remote procedure calls

  Synchronous request/response interactions between processes

  Service is defined by a pair of messages: one for the request and one for the reply.

- Distributed parameter system

  A way for tasks to share configuration information through a global key-value store. It allows you to easily modify your task settings, and even allows tasks to change the configuration of other tasks.

2. ROS core components

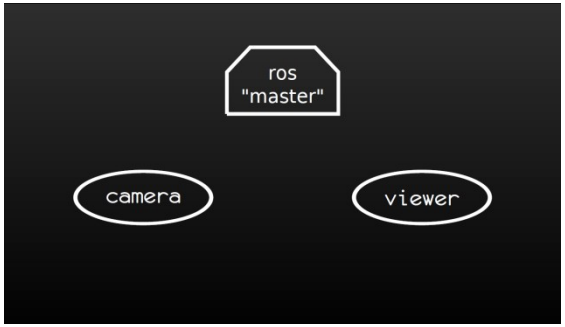
1. Communication infrastructure

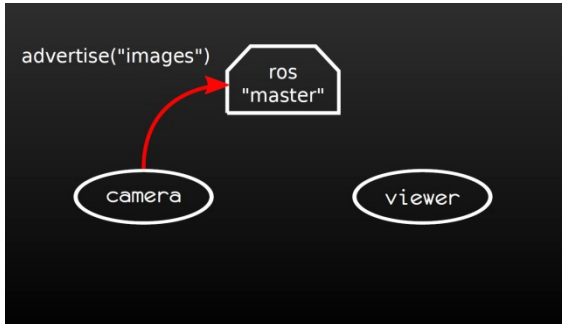
The ROS core is a set of the only three programs that are necessary for the ROS runtime:

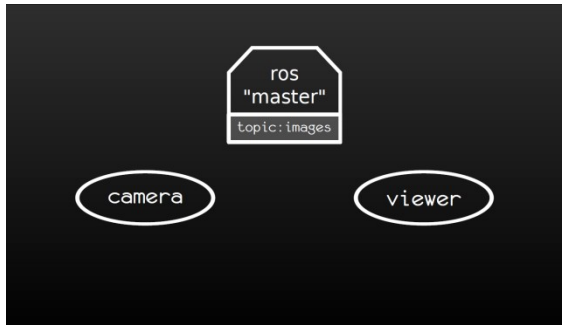
1. **ROS Master**
   A centralized XML-RPC server
   Negotiates communication connections
   Registers and looks up names for ROS graph resources
2. **Parameter Server**
   Stores persistent configuration parameters and other arbitrary data
3. **rosout**
   Essentially a network-based stdout for human-readable messages

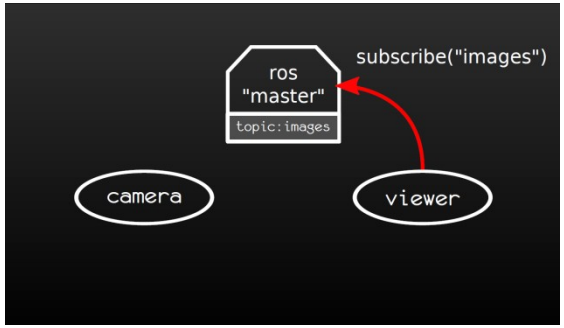
2. ROS core components

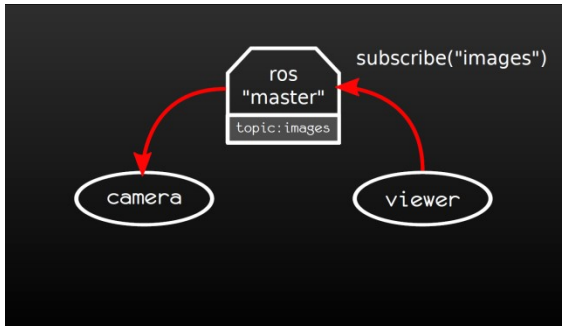
1. Communication infrastructure

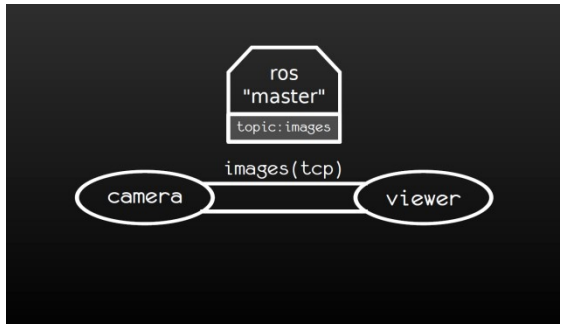


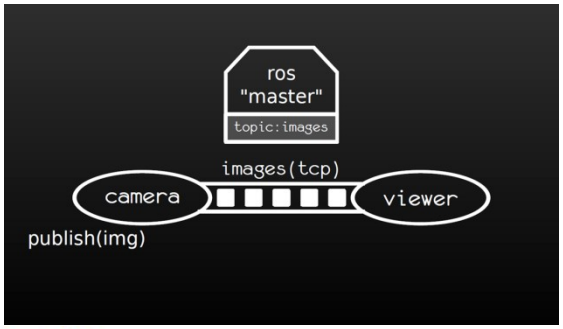


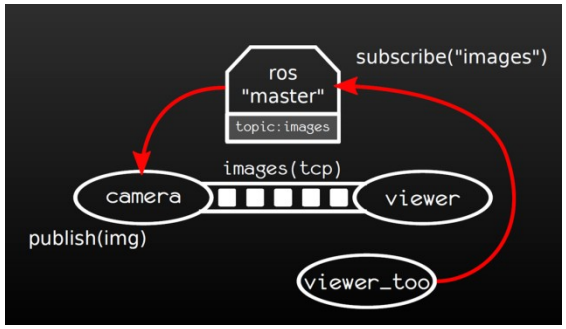


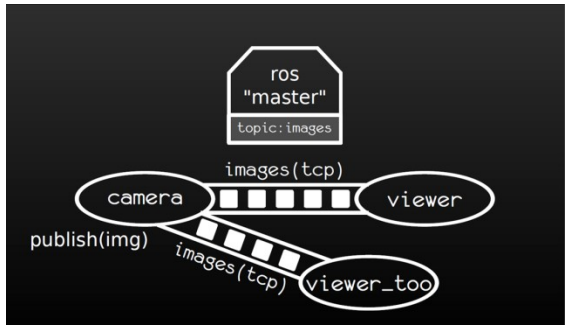












2. ROS core components

2. Robot specific features

Some of the robot-specific capabilities that ROS provides are:

- Standard Message Definitions for Robots
  There are message definitions for geometric concepts like poses, transforms, and vectors; for sensors like cameras, IMUs and lasers; and for navigation data like odometry, paths, and maps; among many others.
- Robot Geometry Library
  The **tf** library manages coordinate transform data for robots (both static transforms, such as a camera that is fixed to a mobile base, and dynamic transforms, such as a joint in a robot arm).
- Robot Description Language
  The Unified Robot Description Format (**URDF**) allows you to describe, using an XML document, the physical properties of your robot, from the lengths of limbs and sizes of wheels to the locations of sensors and the visual appearance of each part of the robot.
- Preemptable Remote Procedure Calls
  Used for communications where you need to initiate a goal-seeking behavior, monitor its progress, be able to preempt it along the way, and receive notification when it is complete.

Action is defined by three messages: one for the request and one for the reply and one for the feedback.

2. ROS core components

3. Tools

One of the strongest features of ROS is the powerful development toolset. These tools support introspecting, debugging, plotting, and visualizing the state of the system being developed.

Command-Line Tools

`roscore`: Run ROS Core Stack (Master, Parameter Server, etc...).
`roscd`: `cd` directly to a location.
`rosrun`: Allows to run an executable (node).
`roslaunch`: Launch a series of nodes from an XML file (`*.launch`).
`rosbag`: To work with `*.bag` files (ROS data recordings).
`rosdep`: To install package dependencies.
`roscreate_pkg`: Create several common files for a new package.
`rosmsg`: Message data structure.
`rosnode`: Runtime node information.
`rospack`: Packages information.
`rosparam`: To get and set values in the parameters server (YAML).
`rossrv`: Service data structure.
`rosservice`: Runtime information about services.
`rostopic`: Runtime information about topics.
`rosversion`: Report the ROS version.

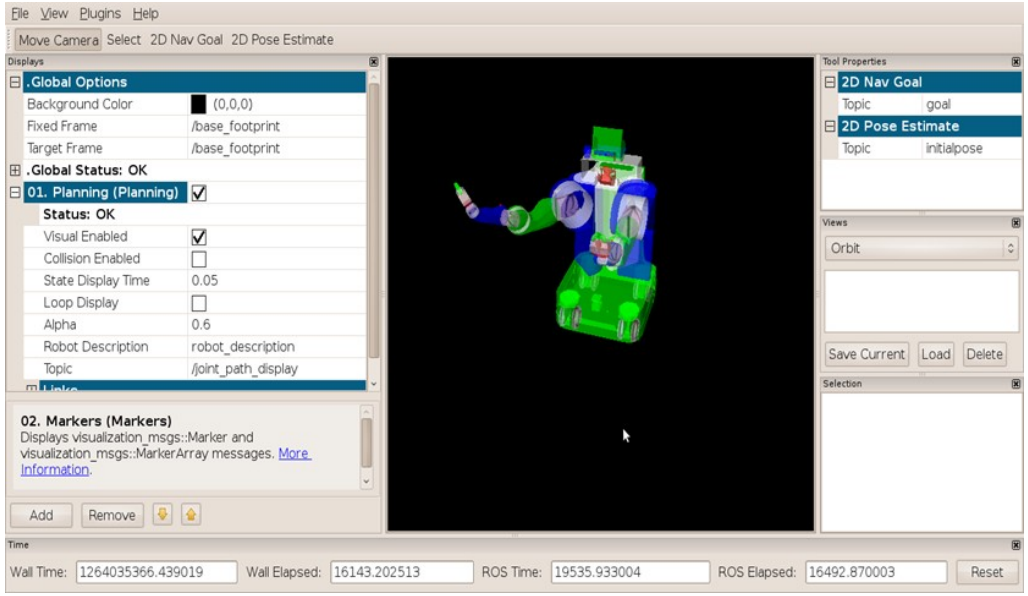
2. ROS core components

3. Tools

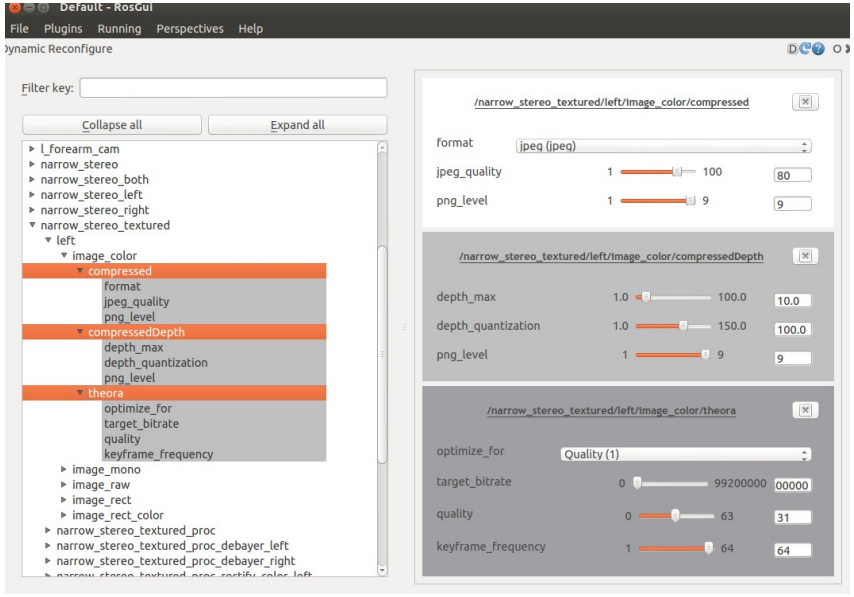
One of the strongest features of ROS is the powerful development toolset. These tools support introspecting, debugging, plotting, and visualizing the state of the system being developed.

Graphical Tools

rviz



rqt_reconfigure



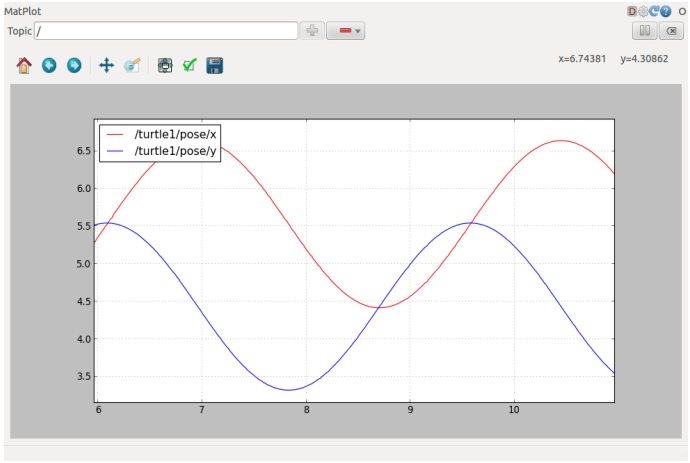
2. ROS core components

3. Tools

One of the strongest features of ROS is the powerful development toolset. These tools support introspecting, debugging, plotting, and visualizing the state of the system being developed.

Graphical Tools

rqt_plot



rqt_bag