Introduction to Ros

What is Ros ?

The Robot Operating System (ROS) is a flexible framework for writing robot software.
ie: getting robots to do things.

It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

This common platform lets people share code and ideas more readily and, perhaps more importantly, means that you do not have to spend years writing software infrastructure before your robots start moving.

Whats Ros is made of ?

Plumbing: ROS provides publish-subscribe messaging infrastructure designed to support the quick and easy construction of distributed computing systems.

Tools: ROS provides an extensive set of tools for configuring, starting, introspecting, debugging, visualizing, logging, testing, and stopping distributed computing systems.

Capabilities: ROS provides a broad collection of libraries that implement useful robot functionality, with a focus on mobility, manipulation, and perception.

Ecosystem: ROS is supported and improved by a large community, with a strong focus on integration and documentation.

Ros Philosophy

Broadly speaking, ROS follows the Unix philosophy of software development in several key aspects. This tends to make ROS feel “natural” for developers coming from a Unix background.

Peer to Peer: ROS systems consist of numerous small computer programs that connect to one another and continuously exchange messages.

Scaling: Unlike many other robotics software frameworks, ROS does not have an integrated development and runtime environment. Instead it creates complex software systems from many small programs

Multilingual: Ros supports C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, Haskell, R, Julia, and others.

Thin: ROS is designed to be as thin as possible so that code written for ROS can be used with other robot software frameworks.

Free and open source: The core of ROS is released under the permissive BSD license, which allows commercial and noncommercial use.

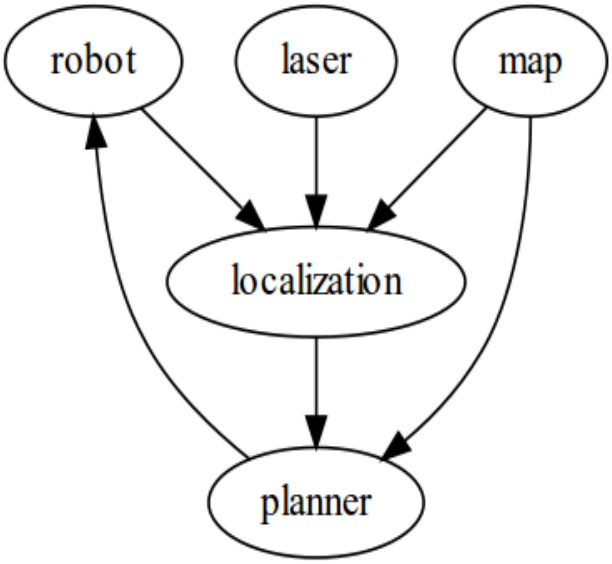
Key Concepts

Ros Graph

Robots consist of many independent subsystems and programs, such as navigation, computer vision, grasping, and so on, that all talk to each other.

Ros attempts to streamline all of these programs.
A ROS system maps all of these programs to one another in a ROS graph.

Within a ROS graph, a Node represents a program that is sending or receiving data called messages.



Roscore

Roscore is how Nodes communicate with each other, it acts like web-based client/server system.

Nodes can send information (Publish) and revive information (Subscribe), Roscore transfers this information called Messages from Publisher to Subscriber.

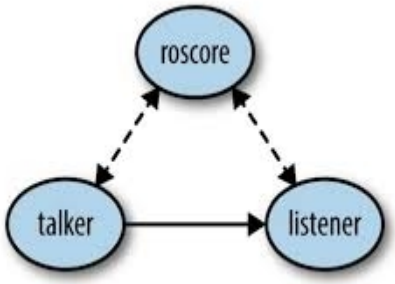


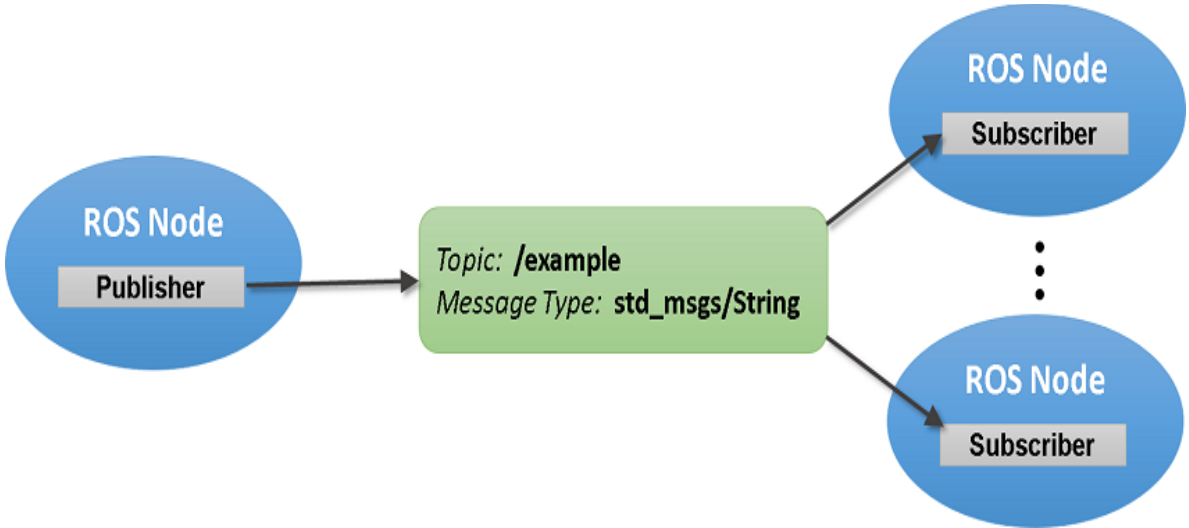
Figure 2-2. roscore connects only ephemerally to the other nodes in the system

Topic

A Topic is a continual stream of Data or Messages with a defined Type. eg: information from a camera is sent over a Topic called *Whats in front* with Type *Image*

Before Nodes can send (Publish) Topics they must first advertise both Topic name and Message Type.

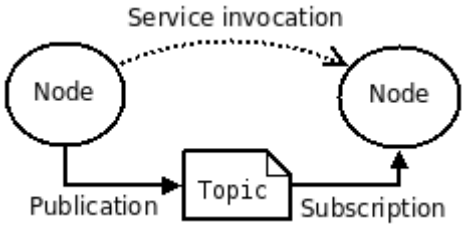
Nodes can both Publish and Subscribe eg: node might subscribe to a topic containing camera images, identify faces in those images, and publish the positions of those faces in another topic, like in Snapchat.



Services

Services are another way to pass data between nodes ROS.
They allow one node to call a function that executes in another node.

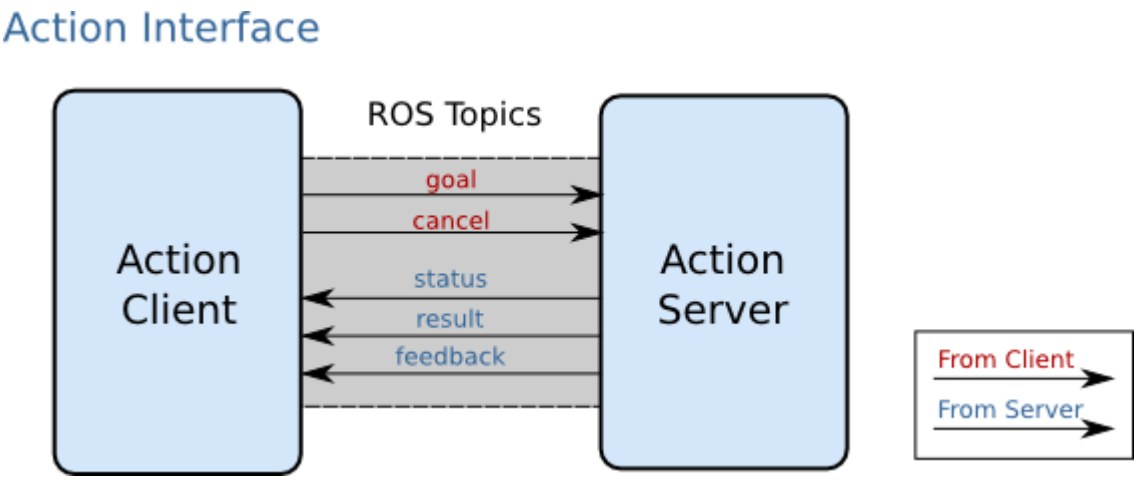
They are ideal for discrete procedure calls that terminate quickly such as such as turning on a sensor or taking a high-resolution picture with a camera.



Actions

Actions are the most powerful communication tool commonly used in Ros.
Similar to a service, an action uses a goal to initiate a behavior and sends a result when the behavior is complete.

Actions can send periodic progression updates as it runs eg: distance traveled, current speed.
Actions can be canceled at anytime.



ROS

Which is best ?

Topic is used for: One-way communication, especially if there might be multiple nodes listening (eg: streams of sensor data)

Service is used for: Simple request/response interactions, such as asking a question about a node's current state .

Action is used for: Most request/response interactions, especially when servicing the request is not instantaneous (eg: navigating to a goal location)


Installing ROS

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > /etc/apt/sources.list.d/ros-latest.list'

$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

$ sudo apt-get update

$ sudo apt-get install ros-indigo-desktop-full python rosinstall

$ sudo rosdep init

$ rosdep update

$ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc

$ source ~/.bashrc
```


Installing Turtlebot

Ubuntu Package Installation

$ sudo apt-get install ros-indigo-turtlebot ros-indigo-turtlebot-apps ros-indigo-turtlebot-interactions ros-indigo-turtlebot-simulator ros-indigo-kobuki-ftdi ros-indigo-rocon-remocon ros-indigo-rocon-qt-library ros-indigo-ar-track-alvar-msgs

Preperation

$ sudo apt-get install python-rosdep python-wstool ros-indigo-ros

$ sudo rosdep init

$ rosdep update

Workspaces Rocon

$ mkdir ~/rocon

$ wstool init -j5 src https://raw.github.com/robotics-in-concert/rocon/release/indigo/rocon.rosinstall

$ source /opt/ros/indigo/setup.bash

$ rosdep install --from-paths src -i -y

$ catkin_make

$ cd

Workspace kobuki

$ mkdir ~/kobuki

$ cd ~/kobuki

$ wstool init src -j5
https://raw.github.com/yujinrobot/yujin_tools/master/rosinstalls/indigo/kobuki.rosinstall

$ source ~/rocon/devel/setup.bash

$ rosdep install --from-paths src -i -y

$ catkin_make

$ cd

Workspace Turtlebot

$ mkdir ~/turtlebot

$ cd ~/turtlebot

$ wstool init src -j5
https://raw.github.com/yujinrobot/yujin_tools/master/rosinstalls/indigo/turtlebot.rosinstall

$ source ~/kobuki/devel/setup.bash

$ rosdep install --from-paths src -i -y

$ catkin_make

$ cd