Control of Mobile Robots

An introduction to ROS

Prof. Luca Bascetta (luca.bascetta@polimi.it)

Politecnico di Milano – Dipartimento di Elettronica, Informazione e Bioingegneria

What is ROS?

- ROS is an open-source robot operating system
- A set of software libraries and tools (middleware) that helps you build robot applications that work across a wide variety of robotic platforms
- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory, development continued at Willow Garage
- Since 2013 managed by OSRF (Open Source Robotics Foundation)

ROS main features

ROS has two parts:

- The operating system side, which provides standard operating system services such as:
  - hardware abstraction
  - low-level device control
  - implementation of commonly used functionality
  - message-passing between processes
  - package management
- A suite of user contributed packages that implement common robot functionality such as SLAM, planning, perception, vision, manipulation, etc.

ROS installation

- Detailed information can be found on the ROS Wiki (wiki.ros.org)
- We consider the installation of ROS Melodic on Ubuntu 18.0.4...
- ... assuming that an installation of Ubuntu 18.04 is already available as
    - a partition of the internal hard drive
    - an external USB drive/stick
    - a virtual machine
- ... and an internet connection is available

- Open a terminal to start the installation...

Installing ROS Melodic on UBUNTU 18.04

- Setup your computer to accept software from packages.ros.org

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

- Set up your keys

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

- Desktop-Full Install

sudo apt-get update

sudo apt install ros-melodic-desktop-full

- Initialize rosdep

sudo apt install python-rosdep

sudo rosdep init

rosdep update

Installing ROS Melodic on UBUNTU 18.04

- Environment setup

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

source ~/.bashrc

- Dependencies for building packages

sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential

- Create a ROS Workspace

mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/

catkin_make

- Before continuing, and every time you start working in a new terminal

source devel/setup.bash

Installing ROS Melodic on UBUNTU 18.04

- To ease working with many terminal install terminator

sudo add-apt repository ppa:gnome-terminator

sudo apt-get update

sudo apt-get install terminator

ROS core concepts

- Nodes
- Messages and Topics
- Services
- ROS Master
- Parameters
- Stacks and packages

ROS nodes

- Single-purpose executable programs
  - e.g. sensor driver(s), actuator driver(s), mapper, planner, UI, etc.
- Individually compiled, executed, and managed
- Nodes are written using a **ROS client library**
  - roscpp, C++ client library
  - rospy, python client library
- Nodes can publish or subscribe to a Topic
- Nodes can also provide or use a Service

ROS topics

- A topic is a name for a stream of messages with a defined type
  - e.g., data from a laser range-finder might be sent on a topic called scan, with a message of type LaserScan
- Nodes communicate with each other by publishing messages to topics
- Publish/Subscribe model: 1-to-N broadcasting

Nodes and messages: an example

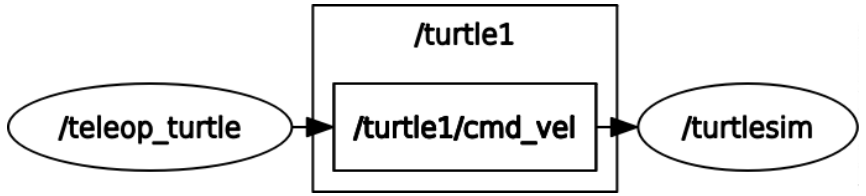
- Download in your *catkin_ws/src* the ROS tutorials package

git clone https://github.com/ros/ros_tutorials.git

- Compile (run *catkin_make* from your workspace folder) and source
- Open four terminals (sourcing for each one) and run
  - *roscore*
  - *rosrun turtlesim turtlesim_node*
  - *rosrun turtlesim turtle_teleop_key*

rosrun package node
runs a node

- You can play with the turtle... then in the last terminal run *rqt_graph*



Nodes and messages: an example

- Using *rostopic* and *rosnode* you can see all the information on the nodes and topics available on the system, for example
  - *rosnode list*, lists the active nodes
  - *rostopic list*, lists the active topics
  - *rostopic echo /topic*, shows all the messages published on topic */topic*
  - *rostopic info /topic*, prints information about topic */topic*
  - *rostopic type /topic*, prints topic type
  - *rostopic pub /topic type arguments*, publish a message on topic */topic*
- To see all the available commands you can use *rostopic –h* or *rosnode –h*

Nodes and messages: an example

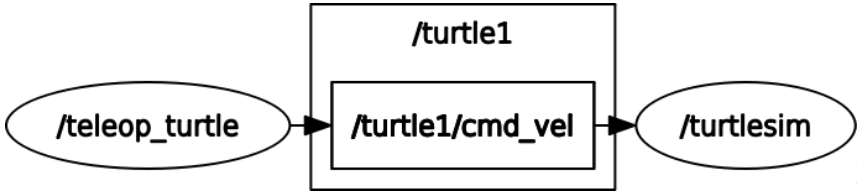
- Play with this example to become familiar with topics, nodes, and messages...
- Examples of possible exercises
  - use *rostopic list* to see all the available topics
  - use *rostopic type* to understand the type associated to each topic
  - use *rostopic hz* to check the frequency at which messages are published
  - use *rostopic echo* to show the messages exchanged between the two nodes
  - use *rostopic pub* to command the turtle

ROS messages

- Strictly-typed data structures for inter-node communication (on the same or on different machines connected through a network)
- For example, *geometry_msgs/Twist* is used to express velocity commands:
  - *Vector3 linear*
  - *Vector3 angular*
- *Vector3* is another message type composed of:
  - *float64 x*
  - *float64 y*
  - *float64 z*

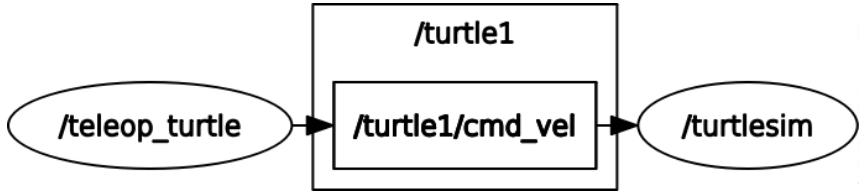
ROS master (roscore)

- Provides connection information to nodes so that they can transmit messages to each other
  - Every node connects to a master at start-up to register details of the message streams it publishes, and the streams to which it subscribes
  - When a new node appears, the master provides it with the information that it needs to form a direct peer-to-peer connection with other nodes publishing and subscribing to the same message topics
- Let’s say we have two nodes, as in the previous example



ROS master (roscore)

- If you start */turtlesim* first it notifies the master that it wants to subscribe to the topic /turtle1/cmd_vel to receive velocity commands
- Then you start */teleop_turtle* and it notifies the master that it wants to publish velocity commands on */turtle1/cmd_vel*
- Now that the topic */turtle1/cmd_vel* has both a publisher and a subscriber, the master node notifies */teleop_turtle* and */turtlesim* about each others existence, so that they can start transferring data to one another



ROS services

- Synchronous inter-node transactions / RPC
- Service/Client model: 1-to-1 request-response
- Service roles:
  - carry out remote computation
  - trigger functionality / behaviour
- Example:
  - map_server/static_map – retrieves the current grid map used by the robot for navigation

ROS services: an example

- Run again the *turtlesim* node
- Use *rosservice list* to print information about the active services
- Use *rosservice type* to print the service argument types
- Call the service */spawn* to create a new turtle

rosservice call /spawn <tab> and then modify the template

Parameter server

- A shared, multi-variate dictionary that is accessible via network APIs
- Best used for static, non-binary data such as configuration parameters
- Runs inside the ROS master
- Parameter values can be stored in a *yaml* file
- Let’s use again the *turtlesim* node to explore the parameter server
- Use *rosparam list* to see the parameters in the parameter server
- The value of a parameter in the server can be get/set using *rosparam get/set*
- Use *rosparam dump* to write the parameters in the server on a file

rosparam dump parameters.yaml

- Check the file to understand how a yaml file can be created

ROS packages

- Software in ROS is organized in packages
- A package contains one or more nodes and provides a ROS interface
- Most of ROS packages are hosted in GitHub

Creating a ROS package

- Change to the source space directory of the catkin workspace

cd ~/catkin_ws/src

- Create a new package called *my_first_package* which depends on *std_msgs*, *roscpp*, and *rospy*

catkin_create_pkg my_first_package std_msgs rospy roscpp

- This will create a *my_first_package* folder which contains a *package.xml* and a *CMakeLists.txt*, which have been partially filled out with the information you gave *catkin_create_pkg*

How does a node work?

Initialization

- Variable initialization
- Registration with the Master
- Publisher/subscriber initialization
- Service initialization
- Getting parameters from parameter server

Infinite loop

- Execution of node code
- During idle time all callbacks are executed

Shutdown

- CTRL+C stops a node
- Deregistration from the master