Integrating and Engineering Intelligent Systems

– Robot Operating System –

O. Boissier

Univ. Lyon, IMT Mines Saint-Etienne, LaHC UMR CNRS 5516, France

UP9 DEFI IA – Winter 2019

Robot Operating System (ROS)

Material issued partially from Rodrigo Ventura, João Reis, Institute for Systems and Robotics, Instituto Superior Tecnico, Lisboa, 2013

Introduction to ROS Framework

ROS:

- is an open-source, software framework for robot software development
- provides operating system-like functionality on heterogenous computer cluster
  - OS services: hardware abstraction, low-level device control, commonly used functionality, message-passing between processes, package management
- is based on graph architecture where processing takes place in a distributed framework of processes (aka nodes)
- enables executables to be individually designed and loosely coupled at runtime
- is appropriate for large runtime systems and for large development processes
- has two basic sides: operating system side, suite of user contributed packages or stacks

ROS Concepts at Filesystem level

Introduction to ROS Framework

- Packages: main unit for organizing software in ROS, e.g. ROS runtime processes (nodes), ROS-dependent library, datasets, configuration files.
  It is a directory with a manifest.xml file.
  A package manifest is a set of metadata about a package (e.g. dependencies, compiler flags)
- Stacks: collections of packages that provide aggregate functionality, such as a navigation stack.
  It is a directory with a stack.xml file.
  A stack manifest is a set of metadata about a stack (e.g. dependencies on other stacks).
  A package inside a stack's directory is part of that stack.

ROS Concepts at Computation Graph level

Peer-to-peer network of ROS processes that are processing data together based on:

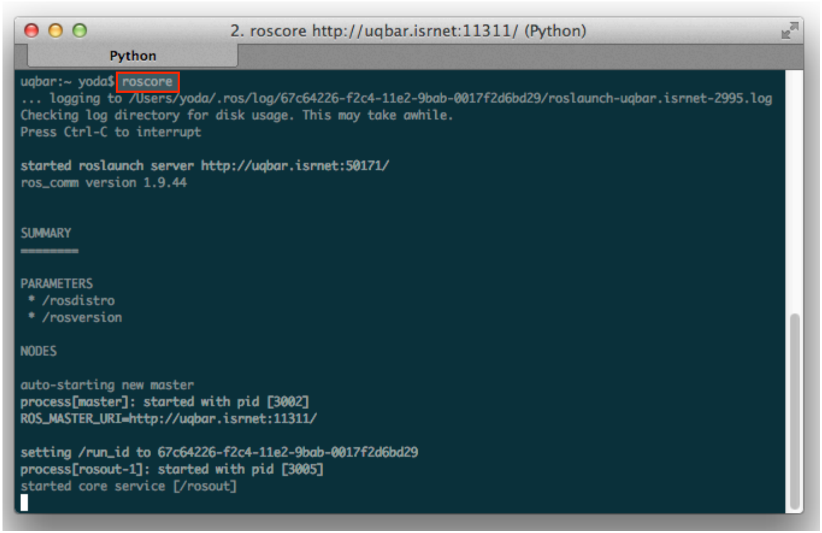
- Name and Parameter server: roscore; singleton (i.e. only one instance running)
    - name registration and lookup to the rest of the computation graph
    - stores topics and services, registration of information for ROS nodes

  By default, used roscore is the one running in localhost by default. It is overriden by the env. var. `ROS_MASTER_URI`
- Nodes: a process performing computation and communicating with other nodes via roscore using topics or services
    - Services: request/response pattern via typed messages
    - Topics: publish/subscribe pattern via typed messages

  For example, one node controls a camera processing, another node performs object recognition.
  A ROS node is written with the use of a ROS client library (e.g. roscpp, rospy)

ROS Concepts at Computation Graph level

Introduction to ROS Framework



2. roscore http://uqbar.isrnet:11311/ (Python)

Python

```
uqbar:~ yoda$ roscore
... logging to /Users/yoda/.ros/log/67c64226-f2c4-11e2-9bab-0017f2d6bd29/roslaunch-uqbar.isrnet-2995.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt

started roslaunch server http://uqbar.isrnet:50171/
ros_comm version 1.9.44


SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion

NODES

auto-starting new master
process[master]: started with pid [3002]
ROS_MASTER_URI=http://uqbar.isrnet:11311/

setting /run_id to 67c64226-f2c4-11e2-9bab-0017f2d6bd29
process[rosout-1]: started with pid [3005]
started core service [/rosout]
```

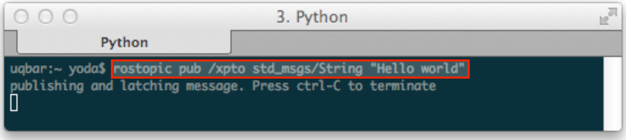
ROS Concepts at Computation Graph level (contd)

Introduction to ROS Framework

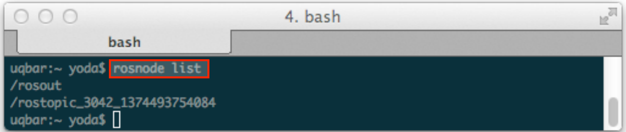
- Messages: data structure of types fields.
  - Standard primitive types (integer, floating point, boolean, string, etc), arrays of primitive types
  - Can include arbitrarily nested structures and arrays
- Topics: 1:n non blocking communication, name used to identify the content of a message
  - a node interested in a certain kind of data will subscribe to the appropriate topic
  - corresponds to a strongly typed message bus: each bus has a name and anyone can connect to the bus to send or receive messages as long as they are the right type
- Services: 1:1 blocking communication, pair of message structures: one for the request, one for the reply
  - a providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply

Publishing String to topic

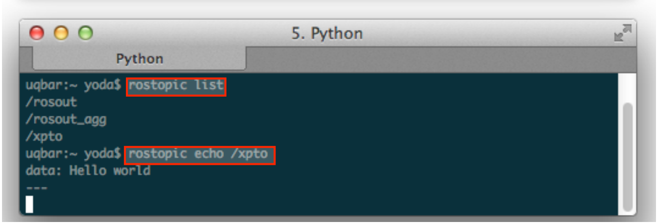
Introduction to ROS Framework



```
uqbar:~ yoda$ rostopic pub /xpto std_msgs/String "Hello world"
publishing and latching message. Press ctrl-C to terminate
```



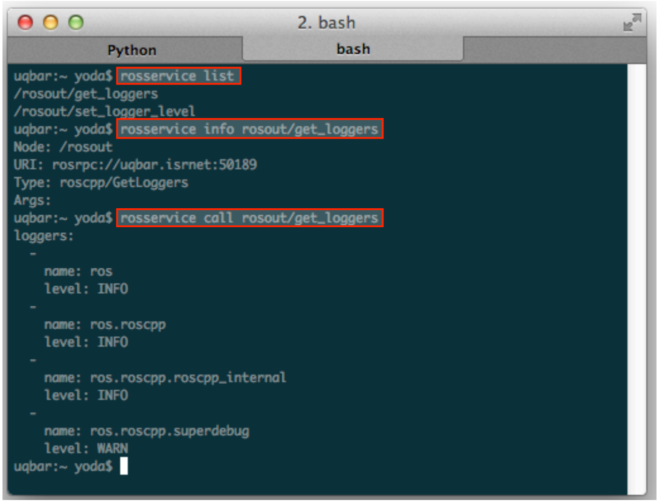
```
uqbar:~ yoda$ rosnode list
/rosout
/rostopic_3042_1374493754084
uqbar:~ yoda$
```



```
uqbar:~ yoda$ rostopic list
/rosout
/rosout_agg
/xpto
uqbar:~ yoda$ rostopic echo /xpto
data: Hello world
---
```

Querying and calling a service

Introduction to ROS Framework



```
uqbar:~ yoda$ rosservice list
/rosout/get_loggers
/rosout/set_logger_level
uqbar:~ yoda$ rosservice info rosout/get_loggers
Node: /rosout
URI: rosrpc://uqbar.isrnet:50189
Type: roscpp/GetLoggers
Args:
uqbar:~ yoda$ rosservice call rosout/get_loggers
loggers:
  -
    name: ros
    level: INFO
  -
    name: ros.roscpp
    level: INFO
  -
    name: ros.roscpp.roscpp_internal
    level: INFO
  -
    name: ros.roscpp.superdebug
    level: WARN
uqbar:~ yoda$
```

Message types

Introduction to ROS Framework

- All messages (including service requests/responses) are defined in text files in a folder `msg`

```
--- sensor_msgs/msg/LaserScan.msg ---

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Introduction to ROS Framework

cf. file: master-ros-framework-intro.pdf

Developing Packages on ROS

cf. file: master-ros-packages.pdf

rosbridge

- JSON protocol to bridge to non-ROS systems
  - for example, connect web browsers to ROS
  - more broadly, connect sockets to ROS
- Much more at `http://rosbridge.org` and `http://www.ros.org/wiki/rosbridge_suite`

References

- Entry point to ROS: http://wiki.ros.org/
- ROS users forum: http://answers.ros.org
- ROS cheat sheet: `https://github.com/ros/cheatsheet/releases/download/0.0.1/ROScheatsheet_catkin.pdf`
- `https://www.youtube.com/playlist?list=PLDC89965A56E6A8D6`
- Turtlebot: https://www.turtlebot.com/
- Learn Turtlebot and ROS: http://learn.turtlebot.com/