Introduction to ROS

Contacts:
Cristian Secchi (cristian.secchi@unimore.it)
Alessio Levratti (alessio.levratti@unimore.it)

Control Of Industrial Robots

November 2013

ROS
What is ROS?

Robotic Operating System

ROS is an open-source, meta-operating system for your robot.

ROS provides libraries and tools for robotics applications and software development.

ROS

What does ROS do?

ROS can be used for:

- Hardware abstraction
- Drivers management
- GUI
- Packages management
- Threads Message passing
- etc. . .

ROS
Objectives

The primary goal of ROS is to support code **reuse** in robotics research and development.
ROS is a distributed framework of processes (aka Nodes) that enables executables to be individually designed and loosely coupled at runtime.
These processes can be grouped into Packages and Stacks, which can be easily shared and distributed.

www.ros.org/wiki

ROS
Prerequisites

The ROS distributions are stable ONLY on Linux-based operative systems. In particular we will use a particular distribution of Linux which is called UBUNTU (version 12.04.03 LTS - Precise Pangolin).
It is required:

- Basic knowledge of Ubuntu
- Good C/C++ programming skills

Tutorial: ROS basics

Basic concepts: the filesystem

ROS filesystem level:

- **Packages**: Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (*nodes*), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most atomic build item and release item in ROS.
- **Package manifests**: Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages.
- **Metapackages (Stacks)**: Metapackages are specialized Packages which only serve to represent a group of related other packages.

Tutorial: ROS basics

Basic concepts: the filesystem

- **Message (msg)Types**: Message descriptions. Define the data structures for messages sent in ROS.
- **Service (srv) Types**: Service descriptions. Define the request and response data structures for services in ROS.

Tutorial: ROS basics

Basic concepts: the computation graph

- **Nodes**: Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system, and so on.
- **Master**: The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

Tutorial: ROS basics

Basic concepts: the computation graph

- **Parameter Server**: The Parameter Server allows data to be stored by key in a central location. It is currently part of the **Master**.
- **Messages**: Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types.
- **Topics**: Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics.

Tutorial: ROS basics

Basic concepts: the computation graph

- **Services**: The publish/subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request/reply interactions, which are often required in a distributed system. Request/reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply.
- **Bags**: Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

Tutorial: ROS basics
The ROS community

- **Distributions**: ROS Distributions are collections of versioned stacks that you can install. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent versions across a set of software.
- **Repositories**: ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.
- **The ROS wiki**: The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.

Tutorial: ROS basics

Groovy Galapagos

For the “Control of Industrial Robots” project you will use “ROS Groovy Galapagos” distribution

Tutorial: ROS basics

The noble art of "Doing it by yourself"

R.T.F.M.

If you don't get something you can:

1. Search it on www.ros.org or just Google it! If you're lucky someone has already done it!
2. Ask on the ROS forum
3. Inscribe to ROS mailing list
4. . . .
5. Ask for help to us.

Tutorial: ROS basics

Basic commands: navigating the filesystem

- `roscd`: navigates through the filesystem:
  `roscd [package[/subdir]]`
- `roscreate-pkg`: creates a new package:
  `roscreate-pkg [package_name] [dependences]`
- `rosmake`: compiles a ROS package:
  `rosmake [package_name]`

Tutorial: ROS basics

Basic commands

- `roscore`: starts the master node.
  `roscore`
- `rosrun`: starts a node execution.
  `rosrun [package] [exeFile_name]`
- `ctrl+C`: kills the node.

Tutorial: ROS basics

"*launch*" files and "`roslaunch`"

"`roslaunch`" is a tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the Parameter Server.

```
roslaunch [package] [file_name].launch
```

Tutorial: ROS basics

Useful tools: `rosdep`

`rosdep` is a command-line tool for installing system dependencies.

```
rosdep install package_name
```

Tutorial: ROS basics

Useful tools: `rostopic`

`rostopic` is a command-line tool for displaying debug information about ROS Topics, including publishers, subscribers, publishing rate, and ROS Messages.
Options:

- `rostopic echo [topic_name]`: Display messages published to a topic.
- `rostopic list -v`: Display a list of current topics verbosely.
- `rostopic hz [topic_name]`: Display the publishing rate of a topic. The rate reported is by default the average rate over the entire time rostopic has been running.
- `rostopic pub [topic_name] [msg_type] [data]`: Publish data to a topic.

Tutorial: ROS basics

Useful tools: `rostopic`

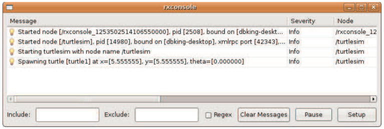
NB

Topics must be mapped correctly in order for nodes to work!

Tutorial: ROS basics

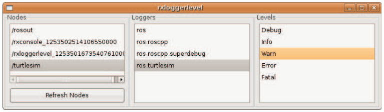
Useful tools: `rqt_logger_level` and `rqt_console`

`rqt_console` (a) provides a GUI plugin for displaying and filtering ROS messages.
`rqt_logger_level` (b) provides a GUI plugin for configuring the logger level of ROS nodes.



Message	Severity	Node
Started node [/rxconsole_1253502514106550000], pid [2508], bound on [dbking-deskt...	Info	/rxconsole_12
Started node [/turtlesim], pid [14980], bound on [dbking-desktop], xmlrpc port [42343],...	Info	/turtlesim
Starting turtlesim with node name /turtlesim	Info	/turtlesim
Spawning turtle [turtle1] at x=[5.555555], y=[5.555555], theta=[0.000000]	Info	/turtlesim

(a)



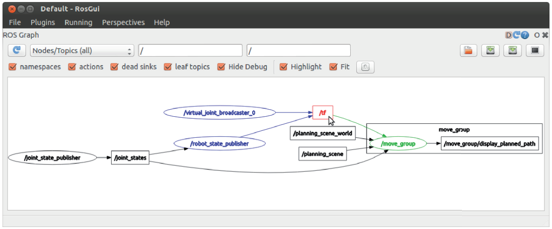
Nodes	Loggers	Levels
/rosout	ros	Debug
/rxconsole_1253502514106550000	ros.roscpp	Info
/rxloggerlevel_125350167354076100(	ros.roscpp.superdebug	Warn
/turtlesim	ros.turtlesim	Error
		Fatal

(b)

Tutorial: ROS basics

Useful tools: `rqt_graph`

`rqt_graph` provides a GUI plugin for visualizing the ROS computation graph.



Tutorial: ROS basics

How to create an executable

Once the C++ program is written then you have to compile it. To compile just type:

```
rosmake [package_name]
```

...but first you have edit the *Manifest.xml* file in order to add the dependences (if you haven't done it yet) and edit the *CMakeList.txt* file.

Tutorial: ROS basics

The *CMakeList.txt* file

To add an executable:

```
rosbuild_add_executable(binFile_name src1 src2...)
```

Tutorial: higher-level concepts

Message types

ROS provides some standard structures for standard messages:

- actionlib_msgs messages for representing actions.
- diagnostic_msgs messages for sending diagnostic data.
- **geometry_msgs** messages for representing common geometric primitives.
- **nav_msgs messages** for navigation.
- **sensor_msgs** messages for representing sensor measurements.

Tutorial: higher-level concepts

Message type: an example

Let's see how the odometry message provided by a mobile robot can be represented in ROS:
Message type: `nav_msgs::Odometry` Structure:

- std_msgs/Header header;
- string child_frame_id;
- geometry_msgs/PoseWithCovariance pose:
  - geometry_msgs/Pose pose:
    - geometry_msgs/Point position: x,y,z;
    - geometry_msgs/Quaternion orientation: x,y,z,w;
  - float64[36] covariance
- geometry_msgs/TwistWithCovariance twist:
  - geometry_msgs/Twist twist:
    - geometry_msgs/Vector3 linear: x,y,z;
    - geometry_msgs/Vector3 angular: x,y,z;
  - float64[36] covariance

Tutorial: higher-level concepts

Quaternions

Quaternion

Def: "*Unit quaternions, also known as versors, provide a convenient mathematical notation for representing orientations and rotations of objects in three dimensions. Compared to Euler angles they are simpler to compose and avoid the problem of gimbal lock. Compared to rotation matrices they are more numerically stable and may be more efficient.*"

A quaternion derives from the "axis-angle" parameterization.

$$\vec{q} = \cos\left(\frac{1}{2}\vartheta\right) + \left(a_x\hat{i} + a_y\hat{j} + a_z\hat{k}\right)\sin\left(\frac{1}{2}\vartheta\right)$$

Tutorial: higher-level concepts
Quaternions

Briefly

ROS provides tools for quaternions convertion and management inside the "`tf`" package.

i.e.: `tf::CreateQuaternionFromYaw(`ϑ`);`

`tf::CreateQuaternionFromRPY(Roll,Pitch,Yaw);`

Tutorial: higher-level concepts

The “tf” package

The tf (which stands for “Transform Frame”) package provides a distributed, ROS-based framework for calculating the positions of multiple coordinate frames over time.

Definition (from ROS)

tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

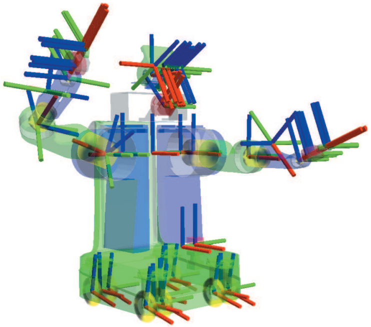
Tutorial: higher-level concepts

The “tf” package

Child and Parent frame

WARNING! Remember that each frame can have many children frames but it can have **ONLY ONE** parent frame!

Tutorial: higher-level concepts
The “tf” package



Tutorial: higher-level concepts

"tf" tools

`view_frames` creates a diagram of the frames being broadcast by tf over ROS. `view_frames`

`evince frames.pdf`

`tf_echo` reports the transform between any two frames broadcast over ROS.

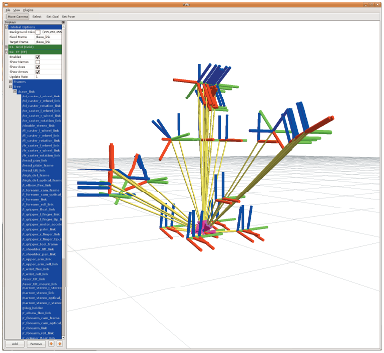
`rosrun tf tf_echo [reference_frame] [target_frame]`

Tutorial: higher-level concepts

rviz

"rviz" is a visualization tool that is useful for examining tf frames.

```
rosrun rviz rviz
```



.Global Options
Background Colo (255,255,255
Fixed Frame /base_link
Target Frame /base_link
01. Grid (Grid)
02. TF (TF)
Enabled ✓
Show Names
Show Axes ✓
Show Arrows ✓
Update Rate 1
Frames
Tree
/base_link
/bl_caster_l_wheel_link
/bl_caster_r_wheel_link
/bl_caster_rotation_link
/br_caster_l_wheel_link
/br_caster_r_wheel_link
/br_caster_rotation_link
/double_stereo_link
/fl_caster_l_wheel_link
/fl_caster_r_wheel_link
/fl_caster_rotation_link
/fr_caster_l_wheel_link
/fr_caster_r_wheel_link
/fr_caster_rotation_link
/head_pan_link
/head_plate_frame
/head_tilt_link
/high_def_frame
/high_def_optical_frame
/l_elbow_flex_link
/l_forearm_cam_frame
/l_forearm_cam_optical_
/l_forearm_link
/l_forearm_roll_link
/l_gripper_float_link
/l_gripper_l_finger_link
/l_gripper_l_finger_tip_li
/l_gripper_motor_accele
/l_gripper_palm_link
/l_gripper_r_finger_link
/l_gripper_r_finger_tip_l
/l_gripper_tool_frame
/l_shoulder_lift_link
/l_shoulder_pan_link
/l_upper_arm_link
/l_upper_arm_roll_link
/l_wrist_flex_link
/l_wrist_roll_link
/laser_tilt_link
/laser_tilt_mount_link
/narrow_stereo_l_stereo
/narrow_stereo_link
/narrow_stereo_optical_
/narrow_stereo_r_stereo
/plug_holder
/r_elbow_flex_link
/r_forearm_cam_frame
/r_forearm_cam_optical_
/r_forearm_link
/r_forearm_roll_link
/r_gripper_float_link

Homeworks

In order to understand properly the ROS environment, it is **STRONGLY RECOMMENDED** to:

- Install the proposed distribution of Ubuntu (12.04.03 - Precise Pangolin) on your personal laptop. You can find it **here**.
- Download and install ROS groovy by following **these instructions**.
- Do the tutorial proposed **here**.

If you are new to Ubuntu/Linux environment you should also check **this one**. If you are new to C++ and Object Oriented programming, you should check **this**.

N.B.

As said above, the course will refer to the GROOVY distribution of ROS. So, be certain to install the right version! Also be certain to install the FULL set of packages. We will refer to ROSBUILD package manager.