ROS-I Basic Training “Mobile Manipulation”

ROS-I Academy

M. Stüttgen, N. Limpert

Mobile Autonomous Systems and Cognitive Robotics Institute (MASCOR)

August 8, 2017

The Robot Operating System (ROS)

- Open Source framework
- Idea: “Meta-OS”
- Support for different architectures and operating systems (Linux/OSX/Windows)
- Large amount of functionality on several layers of a robot architecture
- Used by numerous research institutions
- De-facto standard

ROSIN and ROS-Industrial

ROSIN Project

ROS-Industrial Quality Assured Robot Software Components
More information: `http://rosin-project.eu`

ROS-I Academy

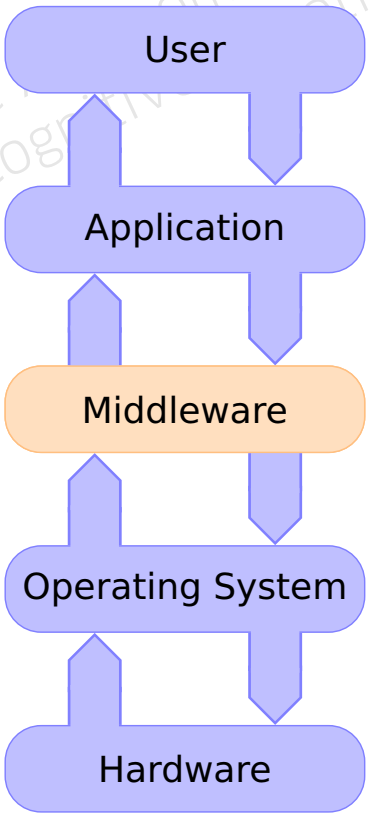
- ROSIN's effort in education for industry professionals
- provides a set of trainings on ROS and ROS-Industrial
- goal is to equip professionals with knowledge, skills, and competences to configure, use, and (eventually) develop (hq) ROS-based software solutions for industry applications

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732287.

Characteristics of Robot Middlewares

According to (Orebäck & Christensen, 2003) a middleware should enjoy the following properties:

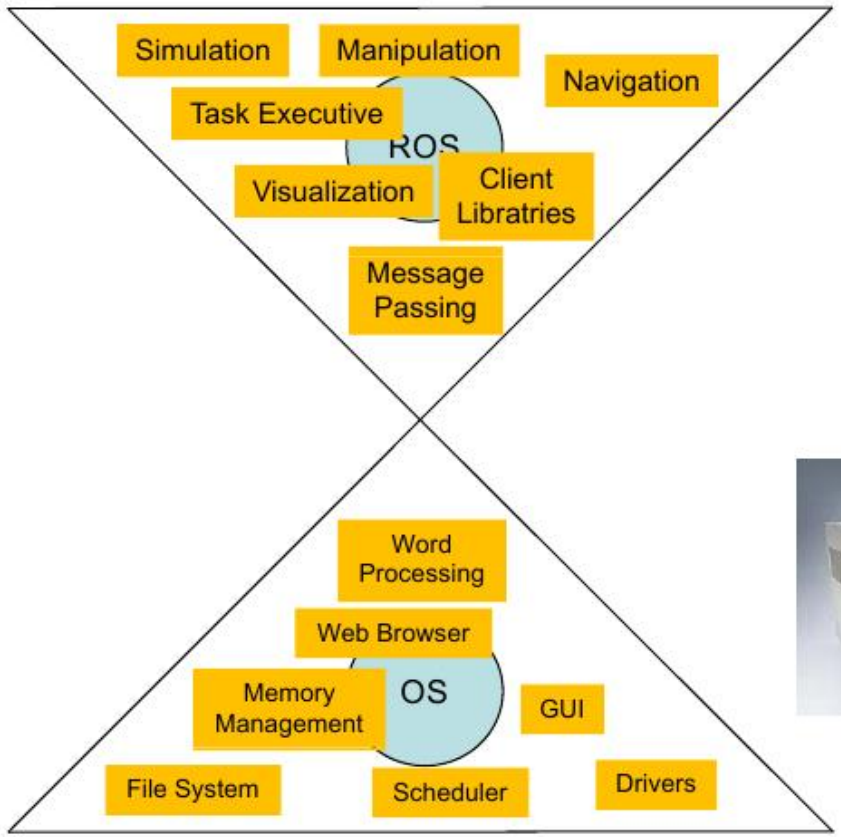
- Hardware abstraction
- Extendability and scalability
- Limited runtime overhead
- Actuator control model
- Common characteristics of good software
- Tools and methodes
- Documentation



Main Ideas of ROS

- **Peer-to-Peer**
  - Independent entities (processes) run on independent hosts
- **Multi-Lingual**
  - XML-RPC-based communication
  - Support for C++, Python, Octave, LISP, JAVA
  - Well-defined data and message types
- **Tool Boxes**
  - Along the line of Unix command line tools
- **Open Source**

Main Ideas of ROS



Resources

- Distributionen
  - Diamondback, Electric, Fuerte, Groovey, Hydro, Indigo, Jade, Kinetic
- ROS Wiki
  - `http://www.ros.org/wiki`
- Bug System
  - `http://www.ros.org/wiki/Tickets`
- ROS FAQ
  - `http://answers.ros.org`
- Other sources: Willow Garage Blogs, Repos, Mailing-Listen, ...
- Annual ROSCon

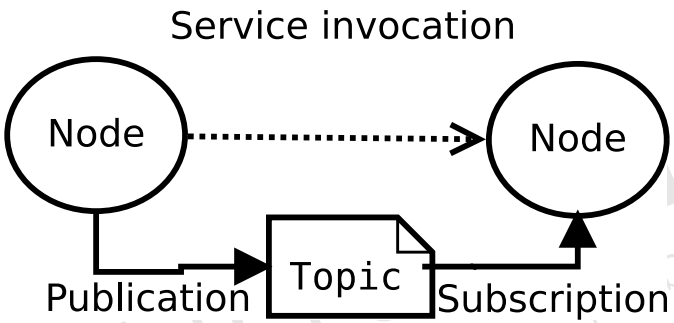
ROS Terminology: File Layer

- **Packages**
  - functional entity (component)
- **Manifesto**
  - contains package information
  - contains dependencies and compiler flags (cmake)
- **Stack**
  - collection of packages aggregated to some functionality
- **Stack Manifesto**
  - contains stack infos (analogous zu manifestos)
- **Message Types**
  - define data structures and message types
- **Service Types**
  - defines simple protocol structures

ROS Terminology: Computation Graph Layer (1)

- **Nodes**
  - Computation entity
  - Makes use of ROS client library
  - Functionality is commonly encapsulated as a node
- **Master**
  - Provides name service
  - Stores topic and service infos
  - Organises node graph structure such that nodes can communicate with each other
- **Parameter Server**
  - Stores data centrally under specified key

ROS Terminology: Computation Graph Layer (2)

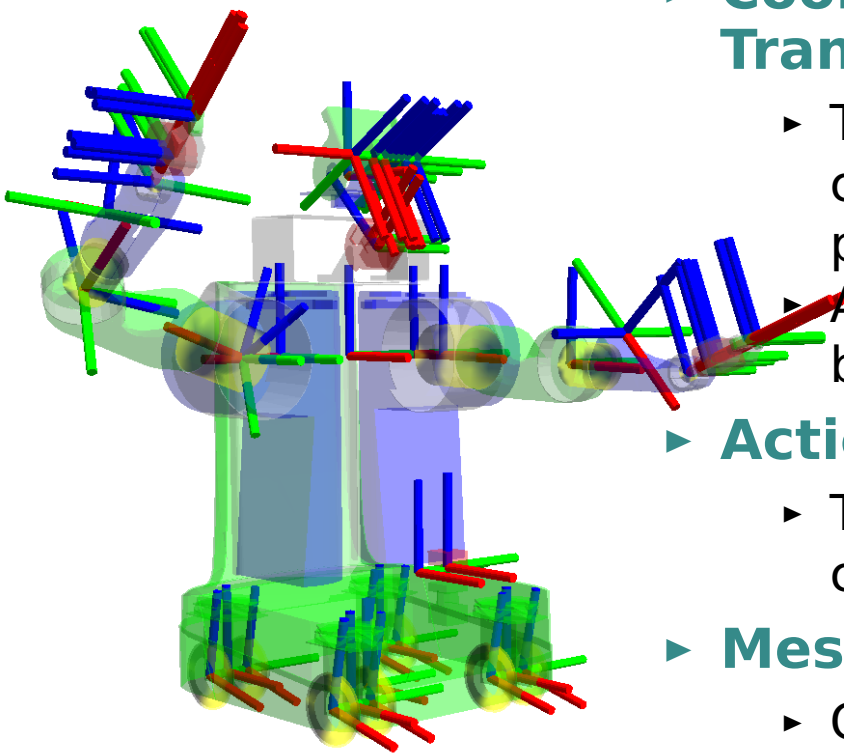


- **Messages**
  - Data structures (and data) exchanged between nodes
- **Topic**
  - Gives a name to the contents of messages
  - Publisher publishes a topic
  - Subscriber subscribes to a topic
- **Services**
  - Protocol structure (Request/Reply)
- **Bags**
  - Data container to store run-time data which can be replayed at a later time

ROS Namespaces

- **Naming Resources in Computation Graph**
  - Nodes, parameter, services, and topics are defined within namespaces
  - a namespace can either be global or local
- **Package Naming**
  - references the file and data type naming
  - message types and service types can also be found by the package name
  - renaming is possible

Higher Level Concepts (1)



Quelle: ros.org

- **Coordinate Systems/ Transforms**
  - Tool for dealing with numerous coordinate systems on a robot platform
  - Allows to transform coordinates between coordinate systems
- **Actions/Tasks**
  - Topic-based interface for more complex tasks
- **Message Ontology**
  - General messages
  - Navigation messages
  - Sensor messages
  - ...

Higher-Level Concepts (2)

- **Plugins**
  - Possibility to load plug-ins at run-time via plugin library
- **Filters**
  - Data filters, e.g. plug-ins for mean filter, median filter, increment filter, . . .
- **Robot Models**
  - 3D models of a robot will be defined in URDF

ROS Cheat Sheet

Filesystem Command-line Tools

rospack/rosstack	A tool inspecting packages/stacks.
roscd	Changes directories to a package or stack.
rosls	Lists package or stack information.
roscreate-pkg	Creates a new ROS package.
roscreate-stack	Creates a new ROS stack.
rosdep	Installs ROS package system dependencies.
rosmake	Builds a ROS package.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
rxdeps	Displays package structure and dependencies.

Usage:

```
$ rospack find [package]
$ roscd [package[/subdir]]
$ rosls [package[/subdir]]
$ roscreate-pkg [package_name]
$ rosmake [package]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ rxdeps [options]
```

Common Command-line Tools

roscore

A collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate.

roscore is currently defined as:

- master
- parameter server
- rosout

Usage:

```
$ roscore
```

rosmsg/rossrv

rosmsg/rossrv displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg.
rosmsg users	Search for code using the msg.
rosmsg md5	Display the msg md5 sum.
rosmsg package	List all the messages in a package.
rosnode packages	List all the packages with messages.

Examples:

Display the Pose msg:

```
$ rosmsg show Pose
```

List the messages in nav_msgs:

```
$ rosmsg package nav_msgs
```

List the files using sensor_msgs/CameraInfo:

```
$ rosmsg users sensor_msgs/CameraInfo
```

rosrun

rosrun allows you to run an executable in an arbitrary package without having to cd (or roscd) there first.

Usage:

```
$ rosrun package executable
```

Example:

Run turtlesim:

```
$ rosrun turtlesim turtlesim_node
```

rosnode

Displays debugging information about ROS nodes, including publications, subscriptions and connections.

Commands:

rosnode ping	Test connectivity to node.
rosnode list	List active nodes.
rosnode info	Print information about a node.
rosnode machine	List nodes running on a particular machine.
rosnode kill	Kills a running node.

Examples:

Kill all nodes:

```
$ rosnode kill -a
```

List nodes on a machine:

```
$ rosnode machine aqy.local
```

Ping all nodes:

```
$ rosnode ping --all
```

roslaunch

Starts ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server.

Examples:

Launch on a different port:

```
$ roslaunch -p 1234 package filename.launch
```

Launch a file in a package:

```
$ roslaunch package filename.launch
```

Launch on the local nodes:

```
$ roslaunch --local package filename.launch
```

rostopic

A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Commands:

rostopic bw	Display bandwidth used by topic.
rostopic echo	Print messages to screen.
rostopic hz	Display publishing rate of topic.
rostopic list	Print information about active topics.
rostopic pub	Publish data to topic.
rostopic type	Print topic type.
rostopic find	Find topics by type.

Examples:

Publish hello at 10 Hz:

```
$ rostopic pub -r 10 /topic_name std_msgs/String hello
```

Clear the screen after each message is published:

```
$ rostopic echo -c /topic_name
```

Display messages that match a given Python expression:

```
$ rostopic echo --filter "m.data=='foo'" /topic_name
```

Pipe the output of rostopic to rosmsg to view the msg type:

```
$ rostopic type /topic_name | rosmsg show
```

rosparam

A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.

Commands:

rosparam set	Set a parameter.
rosparam get	Get a parameter.
rosparam load	Load parameters from a file.
rosparam dump	Dump parameters to a file.
rosparam delete	Delete a parameter.
rosparam list	List parameter names.

Examples:

List all the parameters in a namespace:

```
$ rosparam list /namespace
```

Setting a list with one as a string, integer, and float:

```
$ rosparam set /foo "['1', 1, 1.0]"
```

Dump only the parameters in a specific namespace to file:

```
$ rosparam dump dump.yaml /namespace
```

rosservice

A tool for listing and querying ROS services.

Commands:

rosservice list	Print information about active services.
rosservice node	Print the name of the node providing a service.
rosservice call	Call the service with the given args.
rosservice args	List the arguments of a service.
rosservice type	Print the service type.
rosservice uri	Print the service ROSRPC uri.
rosservice find	Find services by service type.

Examples:

Call a service from the command-line:

```
$ rosservice call /add_two_ints 1 2
```

Pipe the output of rosservice to rossrv to view the srv type:

```
$ rosservice type add_two_ints | rossrv show
```

Display all services of a particular type:

```
$ rosservice find rospy_tutorials/AddTwoInts
```

Some ROS Commands

- File system commands:
  `roscd, rosls, rosdep, catkin_make, catkin_create_package`
- Starting ROS: `roscore`
  launches Master, Parameter Server and StdOut (rosout)
- Message-based commands:
  `rosmsg show, rosmsg package, ...`
- Lauchning packages: `rosrun package executable`
- Debug info about a node:
  `rosnode info, rosnode kill, rosnode list, ...`
- Executing launch files `roslaunch package launchfile`
- Displaying topics
  `rostopic list, rostopic echo, rostopic pub, rostopic info`