ROS INTRODUCTION

ROBOTICS

POLITECNICO
MILANO 1863

ROS: ROBOT OPERATING SYSTEM

ROS main features:

- Distributed framework
- Reuse code
- Language independent
- Easy testing on Real Robot & Simulation
- Scaling

ROS Components

- Filesystem tools
- Building tools
- Packages
- Monitoring and GUIs
- Data Logging

ROS

OVERVIEW ON ROS ARCHITECTURE

Nodes: executables that uses ROS middleware to communicate with other nodes, they are processes and communication happens by publish/subscribe

Messages: data type for the Topics

Packages: main container of any element of the ROS architecture, may contain a collection of nodes and/or messages

Topics: nodes can publish messages to a topic or subscribe to a topic to receive messages; a topic is a typed communication channel

Master: Name service for ROS

rosout: standard output and standard error for ROS

roscore: Master + rosout + parameter server

FILESYSTEM TOOLS

Change directory in the ROS filesystem

```
roscd [locationname[/subdir]]
```

Examples:

```
roscd roscpp && pwd     /opt/ros/indigo/share/roscpp
roscd roscpp/srv        /opt/ros/indigo/share/roscpp/srv
roscd robby_roboto      ~/catkin_ws/src/robby_roboto
```

FILESYSTEM TOOLS

Getting information about installed packages

`rospack` `<subcommand> [options] [package]`

Allowed subcommands (among the others)

`help [subcommand]`	help menu
`depends [package]`	package dependencies
`find [package]`	find package directory
`list`	list available packages

Examples:

```
rospack find roscpp     /opt/ros/indigo/share/roscpp
rospack list            <several packages>
```

PACKAGE CREATION

Command to create a new package

```
catkin_create_pkg [package_name] [depend1] [depend2] [depend3]
```

Example

```
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

Important Notes

- Since Groovy catkin has become the default building tool
- roscpp and rospy are client libraries to use C++ and Python
- Before being able to do that you should have creates a ros_workspace

ROS CORE

The ROS core is a set of the only three programs that are necessary for the ROS runtime:

- ROS Master:
  - A centralized XML-RPC server
  - Negotiates communication connections
  - Registers and looks up names for ROS graph resources
- Parameter Server: stores persistent configuration parameters and other arbitrary data
- rosout: network-based stdout for human-readable messages

STARTING THE ROS MIDDLEWARE

To start the ROS middleware just type in a terminal

```
roscore
```

Now it is possible to display information about the nodes currently running

```
rosnode list
```

Retrieve information about a specific node

```
rosnode info /rosout
```

ROS NODES

The basic elements of a ROS architecture are nodes

- Nodes use a client library to communicate with other nodes
- Nodes can publish/subscribe to a Topic
- Nodes can use or host a Service
- Nodes are implemented using client libraries
  - rospy: Python library
  - roscpp: C++ library
  - rosjava: java library (for android)

The rosnode command can be used to get information about nodes

DEALING WITH NODES

Getting information about running nodes

`rosnode` `<command>`

Allowed commands (among the others)

`rosnode ping`	test connectivity to node
`rosnode info`	print information about node
`rosnode kill`	kill a running node
`rosnode cleanup`	purge registration information of unreachable nodes

Examples:

```
rosnode list
rosnode info /rosout
```

ROS "GRAPH" ABSTRACTION

- Nodes: represent processes distributed across the ROS network. A ROS node is a source and sink for data that is sent over ROS network.
- Parameters: Persistent (while the core is running) data such as configuration and initialization settings, stored on the parameter server.
- ROS Topics
  - Asynchronous "stream-like" communication
  - TCP/IP or UDP Transport
  - Strongly-typed (ROS .msg spec)
  - Can have one or more publishers / subscribers
- ROS Services
  - Synchronous "function-call-like" communication
  - TCP/IP or UDP Transport
  - Strongly-typed (ROS .srv spec)
  - Can have only one server, but several clients

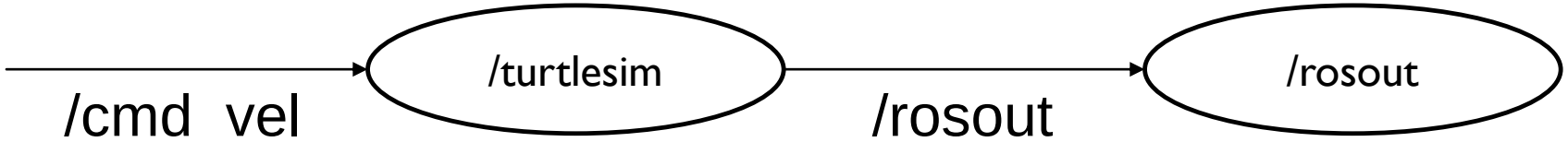
STARTING ROS NODES

To start a ROS node type in a terminal

```
rosrun [package_name] [node_name]
```

Examples:

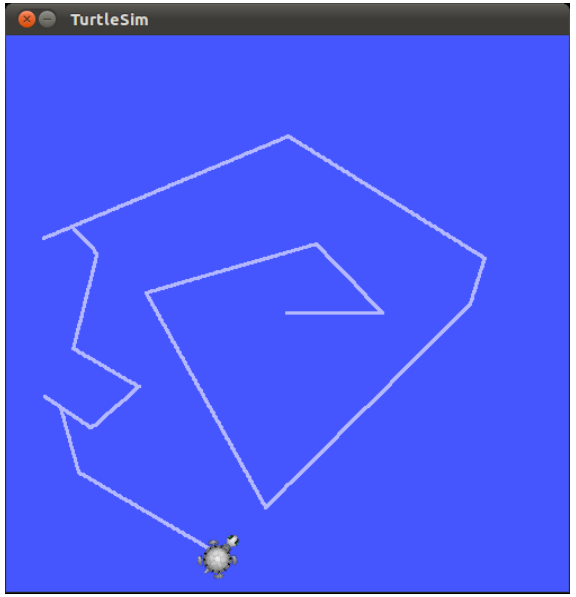
```
rosrun turtlesim turtlesim_node
rosnode ping turtlesim
rosnode info turtlesim
```



STARTING ROS NODES

In a new terminal

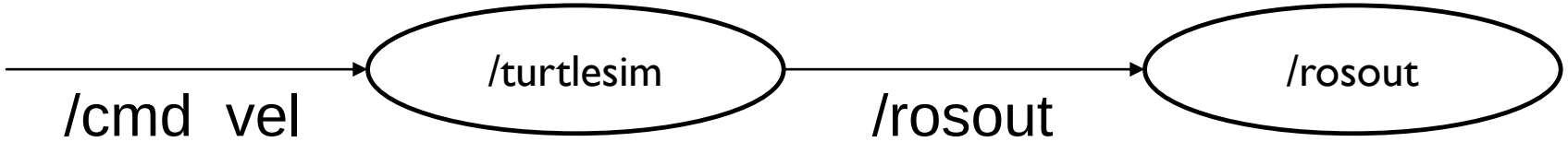
```
rosrun turtlesim turtle_teleop_key
```



Notes:

`turtle_teleop_key` is publishing the key strokes on a topic

`turtlesim` subscribes to the same topic to receive the key strokes



DEALING WITH TOPICS

To show the running node type in a terminal

```
rosrun rqt_graph rqt_graph
```

To plot published data on a topic

```
rosrun rqt_plot rqt_plot /turtle1/pose/x /turtle1/pose/y
rosrun rqt_plot rqt_plot /turtle1/pose/x:y
```

To monitor a topic on a terminal type

```
rostopic echo /turtle1/cmd_vel
```

DEALING WITH TOPICS CONT.

Getting information about ROS topics

`rostopic` `<command> [options]`

Allowed commands (among the others)

`rostopic bw`	display bandwidth used by topic
`rostopic echo`	print messages to screen
`rostopic find`	find topics by type
`rostopic hz`	display publishing rate of topic
`rostopic info`	print information about active topic
`rostopic list`	list active topics
`rostopic pub`	publish data to topic
`rostopic type`	print topic type

DEALING WITH TOPICS CONT.

Getting information about ROS topics

```
rostopic type [message]
```

Examples:

```
rostopic type /turtle1/cmd_vel
rosmsg show turtlesim/Pose
```

Publishing ROS topics

```
rostopic pub [topic] [msg type] [args]
```

Example:

```
rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist '{linear:  {x: 0.1, y:
0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'
```