The Robot Operating System

A 5-day ROS crash course

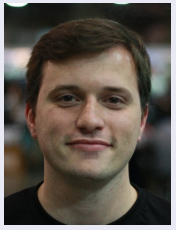
Murilo Fernandes Martins, PhD

Department of Electrical Engineering
FEI University Centre

28 January 2013

Part ∅: Meet the teachers

Meet the teachers



J. Angelo Gurzoni Jr., MSc
Ph.D. candidate in Robotics
Dept. of Mechanical Engineering, USP

Luiz A. Celiberto Jr., PhD
Research Fellow in Robotics
Dept. of Electrical Engineering, FEI

Murilo Fernandes Martins, PhD
Research Fellow in Robotics
Dept. of Electrical Engineering, FEI

ROS and the learning process

The task of learning ROS can be partitioned into...

- Learning HOW to use ROS and its tools
  - Core concepts of the framework
  - Command line tools
  - Graphical tools
- Learning HOW to make use of ROS to facilitate R & D
  - Object recognition
  - Mapping, localisation, navigation
  - Object manipulation

Part I: Course programme for the 5 days

1. Day 1 – An Introduction to ROS
2. Day 2 – Diving into ROS with practical examples
   - Tutorial I: Getting familiar with basic command line tools
   - Tutorial II: Programming ROS nodes and services in C++
3. Day 3: Case Study 1 – Mobile robots, SLAM and navigation
4. Day 4: Case Study 2 – Computer Vision using the Kinect sensor
5. Day 5: Hackathon! Group project

Day 1 – Presenter: Murilo F. M.

An Introduction to ROS

1. What is ROS?
2. Getting started
3. ROS filesystem
4. ROS graph concepts

Tutorial I – Presenter: Luiz A. Celiberto Jr.

Getting familiar with basic command line tools

- rosrun
- rostopic
- rxplot
- rxgraph
- rosnode
- using ROS with multiple machines

Tutorial II – Presenter: J. A. Gurzoni Jr.

Programming ROS nodes and services in C++

- Programming a publisher node
- Coding a subscriber node
- Writing a node which provides a service
- Calling a service from within a node
- Discussion: difference between Topics and Services

Day 3: Case Study 1 – Presenter: Murilo F. M.

Using ROS with mobile robots to perform SLAM and navigation

- Preparing the testbed: the Gazebo simulator and the PR2 robot
- Simultaneous Localisation and Mapping (SLAM) using GMAPPING
- Probabilistic robot localisation using the AMCL algorithm
- PR2 Navigation stack: path planning using the DWA approach

Also covering key concepts of ROS, featuring

- rviz: 3D visualisation environment for robots using ROS
- rosbag: tools for recording from and playing back to ROS Topics
- ROS launch files

Day 4: Case Study 2 – Presenters: J. A. Gurzoni Jr. and Luiz A. Celiberto Jr.

Computer Vision using the Kinect sensor

- Using depth information from RGB-D sensors with ROS
- Controlling a real robot
- Implementing a Kinect-powered person-follower robot
- ROS + OpenCV for face recognition
- Further exploring the power of OpenCV with ROS

Day 5: Hackathon!

Group project

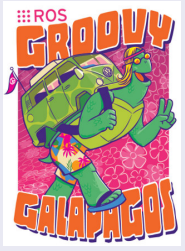
We need ideas. . . Have you got any?

Part II: Day 1 – An Introduction to ROS

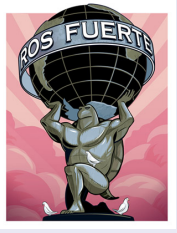
1. What is ROS?
2. Getting started
   - Supported operating systems
   - Supported hardware
   - Installation
3. Setting up VM
4. ROS filesystem
5. ROS graph concepts
   - Nodes
   - Master
   - Parameter server
   - Messages
   - Topics
   - Services
   - Messages, Topics and Services
   - Revisiting the filesystem

What is ROS?

ROS is an open-source meta-operating system



Groovy Galapagos
31 December 2012



Fuerte Turtle
23 April 2012

Electric Emys
30 August 2011

Diamondback
02 March 2011

C Turtle
02 August 2010

Box Turtle
02 March 2010

ROS key features

Features resembling a real operating system

- hardware abstraction and low-level device control
- programming language independence
- implementation of a wide range of commonly used algorithms
- message passing between processes (OS-independent)
- standardised package management
- useful set of shell commands and utilities with tab completion

ROS key features

ROS is inherently distributed

Structured as a peer-to-peer network of processes (Nodes) and loosely coupled at runtime, which may share messages using:

- synchronous RPC-style communication (over Services)
- asynchronous data streaming communication (over Topics)
- storage of data (on a Parameter Server)

ROS concepts and components

ROS client libraries

Main client libraries:

- Python
- C++
- Lisp

Experimental client libraries:

- Java (with Android support)
- Lua

Supported operating systems

Supported operating system

Ubuntu (12.04 LTS + ROS Fuerte)

Experimental

Arch

Debian

Fedora

Gentoo

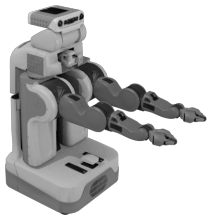
Mac OS X

OpenSuse

Windows

Supported robots


Nao


Willowgarage PR2


Baxter


Care-o-Bot


Toyota Helper


Gostai Jazz


Robonaut


Peoplebot

Kuka YouBot


Guardian


Husky A200


Summit


Turtlebot


Erratic


Qbo


AR.Drone


Miabot


AscTec


Lego NXT


Pioneer


SIA 10D

A lot more on http://www.ros.org/wiki/Robots

Videos – ROS Overview

Video: Celebrating 3 years of ROS[1]

[1] Video available at http://youtu.be/7cslPMzklVo

Videos – ROS Overview

Video: Celebrating 5 years of ROS[2]

[2]Video available at http://youtu.be/PGaXiLZD2KQ

Sensors

- 1D/2D/3D range finders
  - Sharp IR range finder
  - Hokuyo laser scanners
  - Sick lasers
  - Microsoft Kinect
  - Asus Xtion

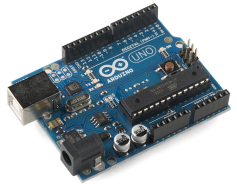
Sensors

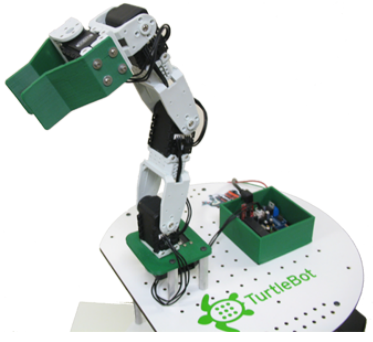
- 1D/2D/3D range finders
- Cameras
  - monocular and stereo
  - USB (uvc) and firewire
  - video streaming (gstreamer)




Sensors

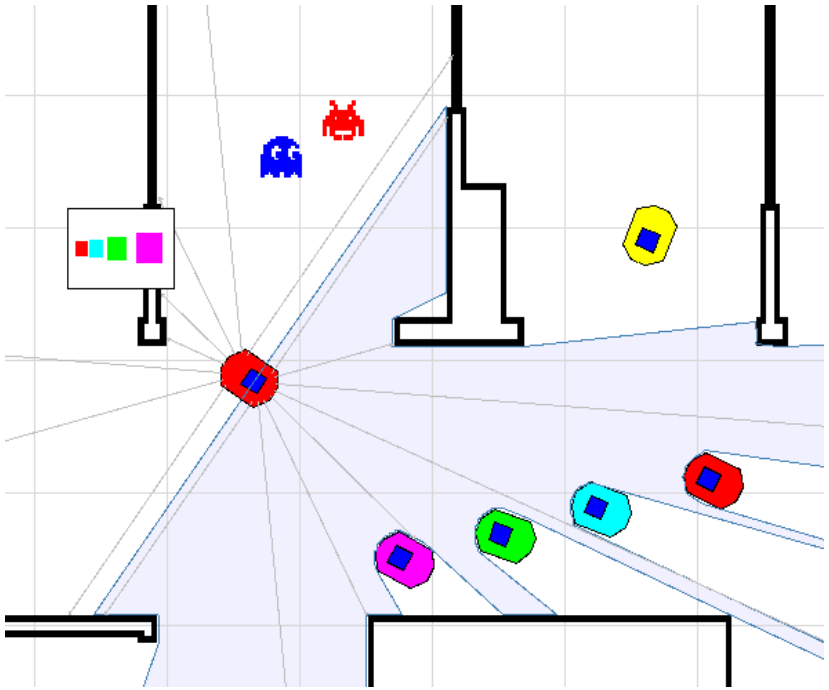
- 1D/2D/3D range finders
- Cameras
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces
  - Dynamixel
  - Phidgets
  - Arduino
  - Arbotix
  - Lego NXT
- And many more...





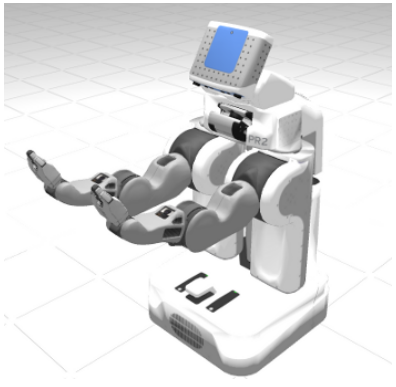
Simulators – Stage

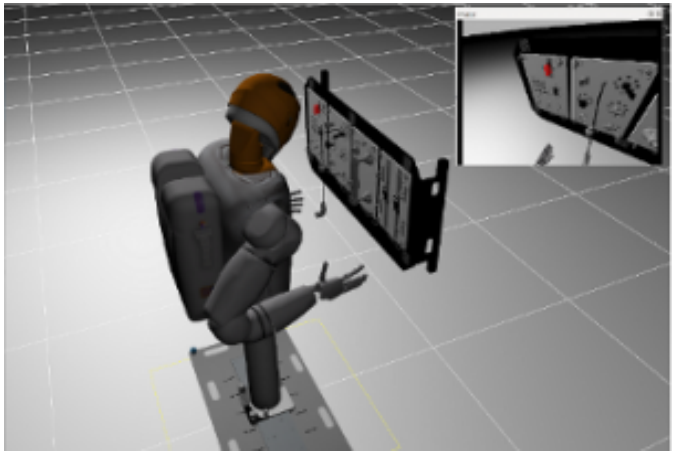
- Stage is a 2D simulator for multiple (large scale) mobile robots
- Models for sensors (e.g., laser, sonar) and actuators (e.g., gripper)
- Models of simple objects for (limited) manipulation
- No physics model at all (e.g., friction, collision, and so forth)
- Open source project

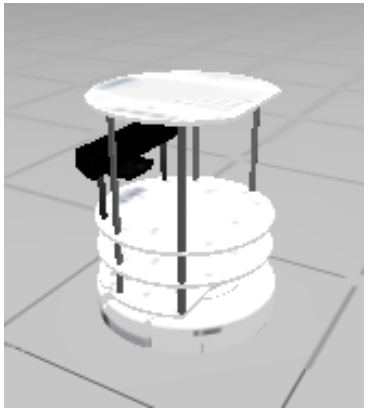


Simulators – Gazebo

- Gazebo is a 3D simulator of multiple robots in realistic environments
- Realistic simulation of rigid body physics/dynamics
- Models for complex robots, actuators and sensors (cameras, IMU)
- Support provided in part by Open Source Robotics Foundation
- Chosen as the simulator for DARPA's Robot Challenge
- Open source project

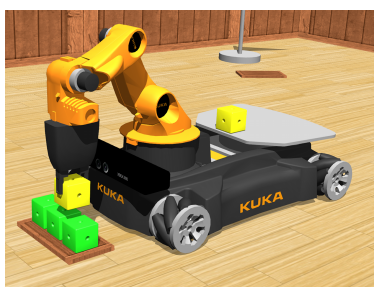


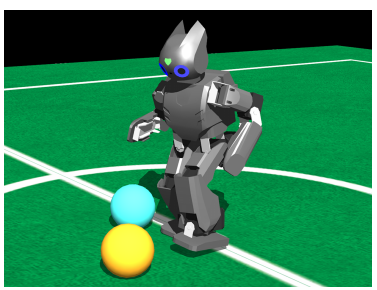




Simulators – Webots

- Development environment used to program and simulate robots
- Complex and realistic 3D simulation of physics/dynamics
- Large collection of robot, sensor and actuator models
- Library of indoor and outdoor objects
- Cross-platform, but proprietary





Installation – ROS (Fuerte) on Ubuntu 12.04 (Precise)

Configure Ubuntu repositories

Allow “restricted”, “universe” and “multiverse” to be used

Setup sources.list

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'
```

Setup keys

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Install ROS Desktop-Full, and standalone tools

```
sudo apt-get update
sudo apt-get install ros-fuerte-desktop-full
sudo apt-get install python-rosinstall python-rosdep
```

Setup environment (shell)

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

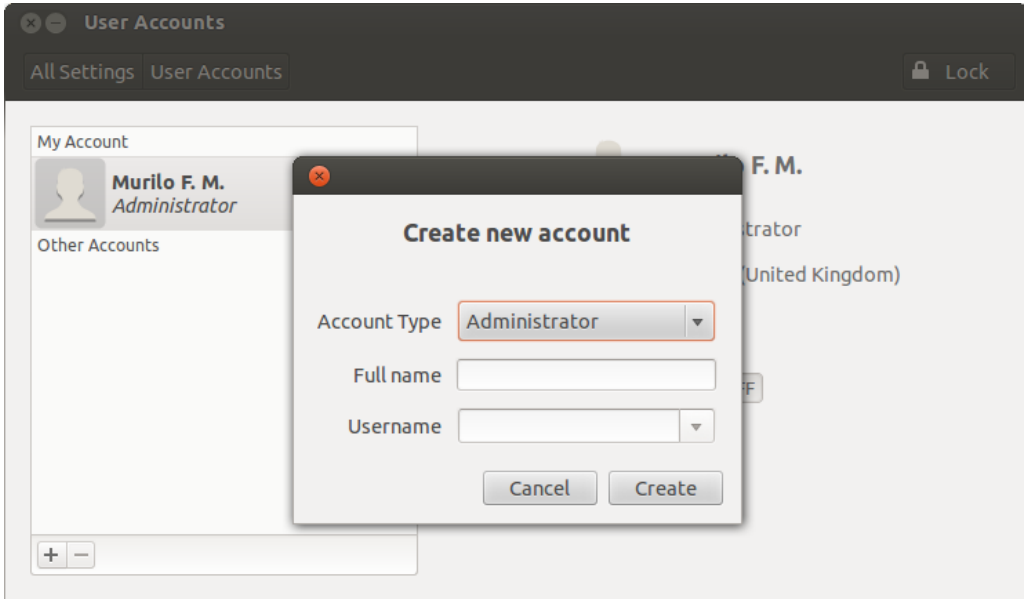
Configuring the Virtual Machine

Loading the VM and logging on to Ubuntu

1. Locate the VM (inside C:\temp folder)
2. Open the VM with VMPlayer (there should be a desktop shortcut)
3. Wait until it loads. . . and switch to full screen mode
4. Log in with username *roscourse* and password *123456*

Creating a new user within the VM

1. Open System Settings
2. Open User Accounts
3. Unlock dialog window
4. Add a new user
5. Set a password!



Setting up ROS environment for the new user

Open up a terminal

Press “windows” key, then type “terminal”, then press “Enter”

Setting up ROS environment for the new user

Type in the following commands

Remember that spaces are necessary, and linux is case sensitive!

```
1  echo ''source /opt/ros/fuerte/setup.bash'' >> ~/.bashrc
2  source ~/.bashrc
3
4  mkdir ~/fuerte_workspace
5  rosws init ~/fuerte_workspace /opt/ros/fuerte
6
7  echo ''source ~/fuerte_workspace/setup.bash'' >> ~/.bashrc
8  source ~/fuerte_workspace/setup.bash
9
10 mkdir ~/fuerte_workspace/sandbox
11 rosws set ~/fuerte_workspace/sandbox
12 source ~/fuerte_workspace/setup.bash
13
14 echo $ROS_PACKAGE_PATH
```

Setting up ROS environment for the new user

Understanding the commands (line by line)

1. Add the command (in double quotes) to the end of the file ∼/.bashrc
2. Re-set the settings in ∼/.bashrc for the current terminal
3.
4. Create a new directory called ∼/.bashrc
5. Initialise a ROS overlay pointing at ∼/fuerte_workspace
6.
7. Same effect as line 1
8. Same effect as line 2
9.
10. Create a new directory called *sandbox* inside fuerte_workspace
11. Set *sandbox* as the workspace
12. Same effect as lines 2 and 8

ROS filesystem – Overview

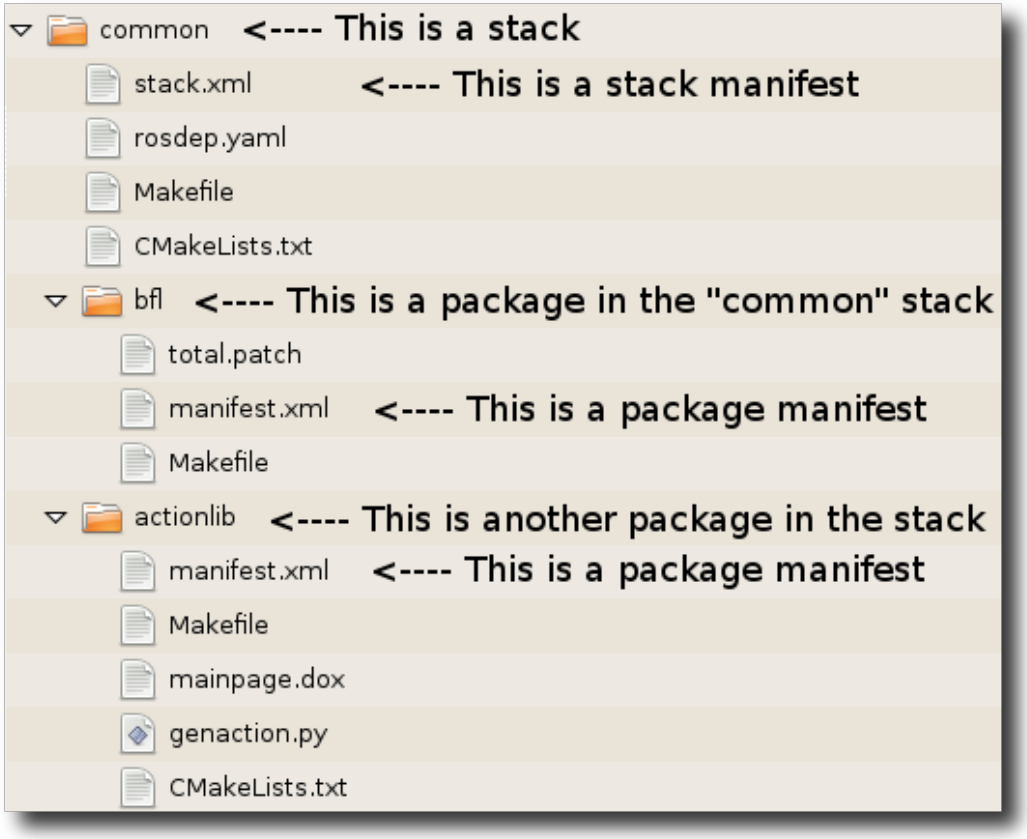
Package

- Lowest level of ROS software organisation
- Dedicated to a **single** functionality (e.g., data acquisition from a laser scanner)
- Manifest: description (metadata) of a *package*, which main role is to define dependencies between *packages* (manifest.xml)

Stack

- Collection of *packages* forming a higher level library
- Stack Manifest: same as in *packages*, but for *stacks* (stack.xml)
- Stacks do not exist in Groovy anymore (fear not, now there are "meta-packages"!)

ROS filesystem – Example



ROS filesystem – Package structure

Hypothetical package myPkg/

- CmakeLists.txt: CMake build settings for package myPkg
- manifest.xml: metadata and dependencies required by package
- mainpage.dox: doc information of package myPkg
- include/myPkg: C++ header files
- src/: source code directory
- bin/: compiled binaries directory
- launch/: where launch files are stored (if needed)
- msg/: message (.msg) types
- srv/: service (.srv) types
- scripts/: executable scripts

rosbash – ROS command line tools

Open up a terminal

Press “windows” key, then type “terminal”, then press “Enter”

rosbash – ROS command line tools

rospack: ROS package management tool

```
rospack list
rospack find turtlesim
rospack depends turtlesim
rospack profile
```

roscd: "change directory" command for ROS

```
roscd
roscd turtlesim
ls (standard linux shell command)
```

rosls: allows you to list the contents of a ROS package

```
roscd (return to workspace directory)
rosls turtlesim
```

Creating a new ROS package

rosed: allows you to quickly see/edit a file from a given package

```
1  echo $EDITOR (if blank, default is vi)
2  export EDITOR=gedit
3  rosed turtlesim manifest.xml
```

Change directory to your sandbox and create a new ROS package

```
1  roscd
2  cd sandbox
3  roscreate-pkg myPkg std_msgs rospy roscpp
```

Creating a new ROS package

Locating and examining the newly created package

```
1  rospack find myPkg
2  roscd myPkg
3  rospack depends1 myPkg
4  rosed myPkg manifest.xml
```

Building the package using rosmake

```
1  rosmake myPkg
```

Now examine the output!

Nodes

Nodes are processes which perform specific computations:

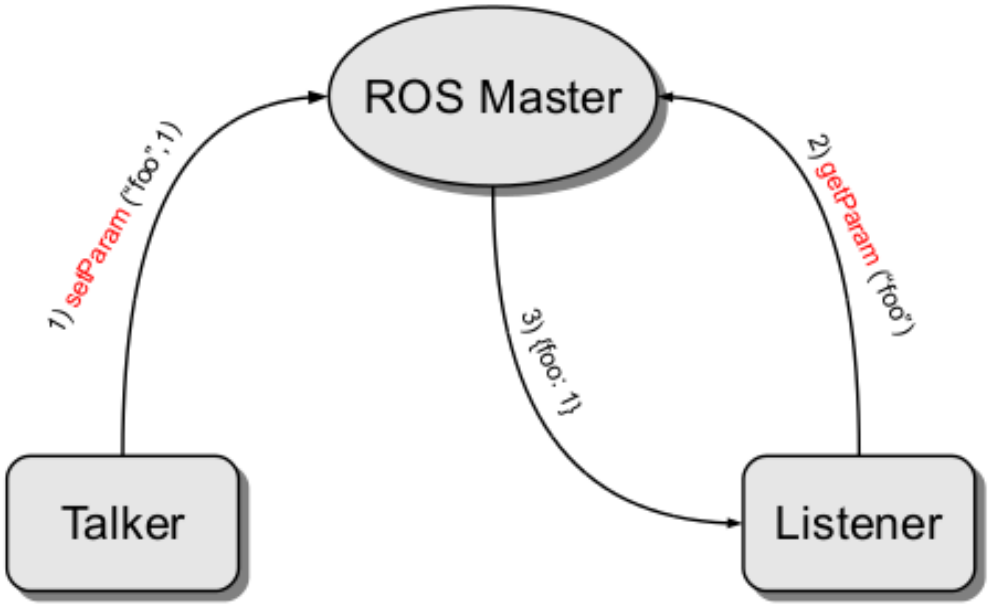
- control robot wheel motors
- acquire data from laser scanner
- acquire images from camera
- perform localisation
- perform path planning
- provide graphical visualisation of the system

Master

- Master is the core *node* of ROS, called *roscore*
- Acts as a nameservice for the Computation Graph (DNS-like server)
- Stores *topics* and *services* registration information for ROS *nodes*
- *Nodes* then establish connections as appropriate
- Also makes callbacks to *nodes* when registration information changes
- Allows *nodes* to dynamically create connections as new *nodes* are run

Parameter server

- Shared, multi-variate dictionary which is acessible via network APIs
- Currently runs inside ROS Master
- *Nodes* use this server to store and retrieve parameters at runtime
- Not designed for high performance, and hence...
- Better suited for configuration parameters
- Follows ROS naming convention, having a hierarchy meant to protect parameters from conflicting (namespaces)



Parameter server

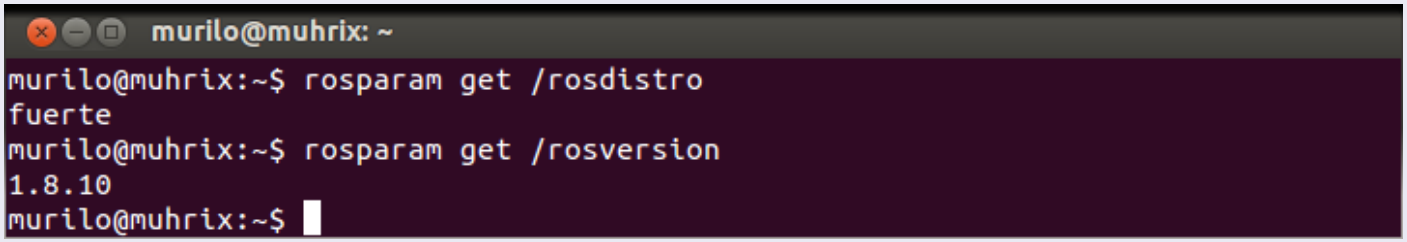
Open up a terminal, then run ROS Master node

```
1    roscore
```

In another terminal, explore the parameter server

```
1    rosparam list
2    rosparam get /rosdistro
3    rosparam get /rosversion
```

It should look like this



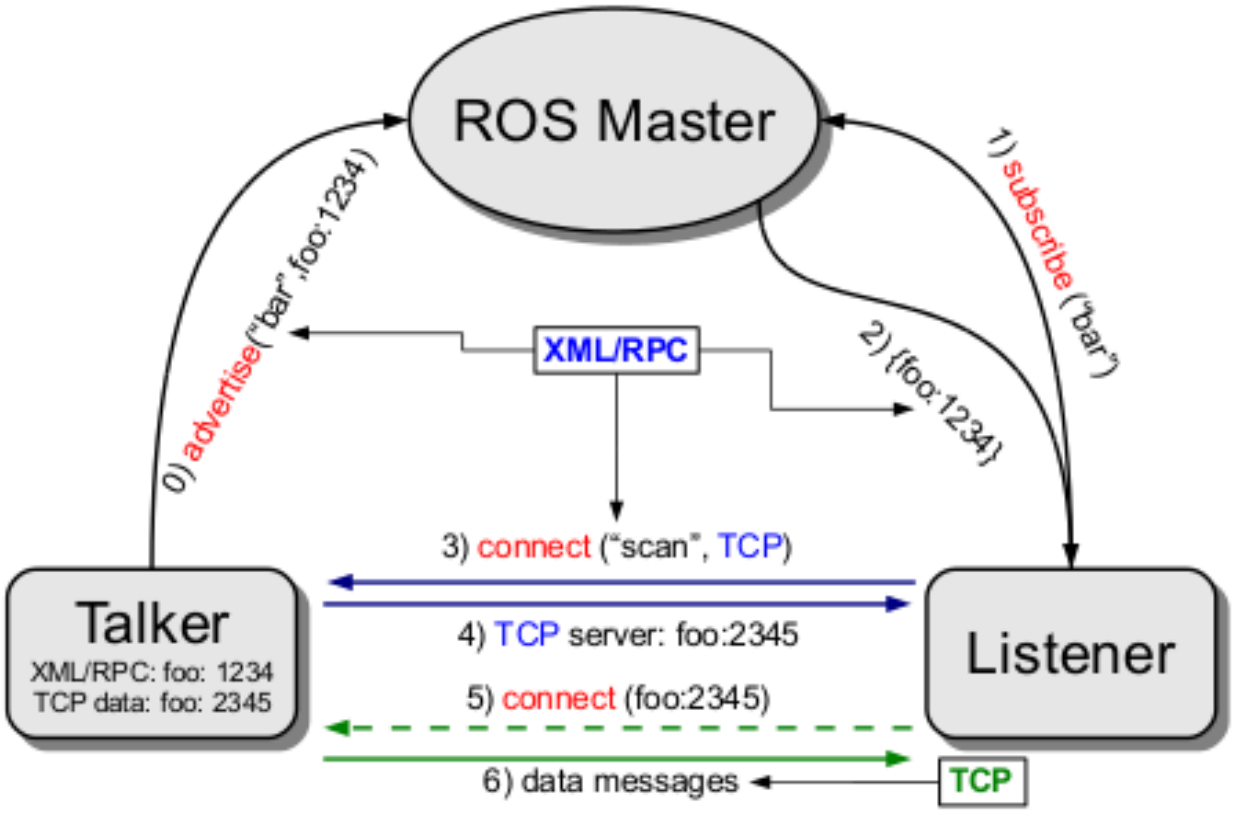
Messages

- Messages are simply a data structure, consisting of typed fields
- Standard primitive types (and nested arrays) are supported:
  - int{8, 16, 32, 64}
  - float{32, 64}
  - string
  - time
  - duration
  - array[]
- *Nodes* communicate with each other by passing messages
- Routed via a transport system with publish/subscribe semantics
- When used with topics: *.msg (n:n non-blocking)
- When used with services: *.srv (1:1 blocking – request + response)

Topics

- A *node* sends out a message by publishing it to a given Topic
- The Topic type is defined by the message type publish on it
- A *node* requiring a certain type of data must subscribe to the appropriate Topic
- Multiple publishers/subscribers to the same Topic are allowed
- A single *node* may publish and/or subscribe to multiple Topics
- Publishers and subscribers are generally unaware of each other's existence
- Publish/subscribe model is a flexible paradigm (many-to-many, one-way transport)
- There is no order of execution required

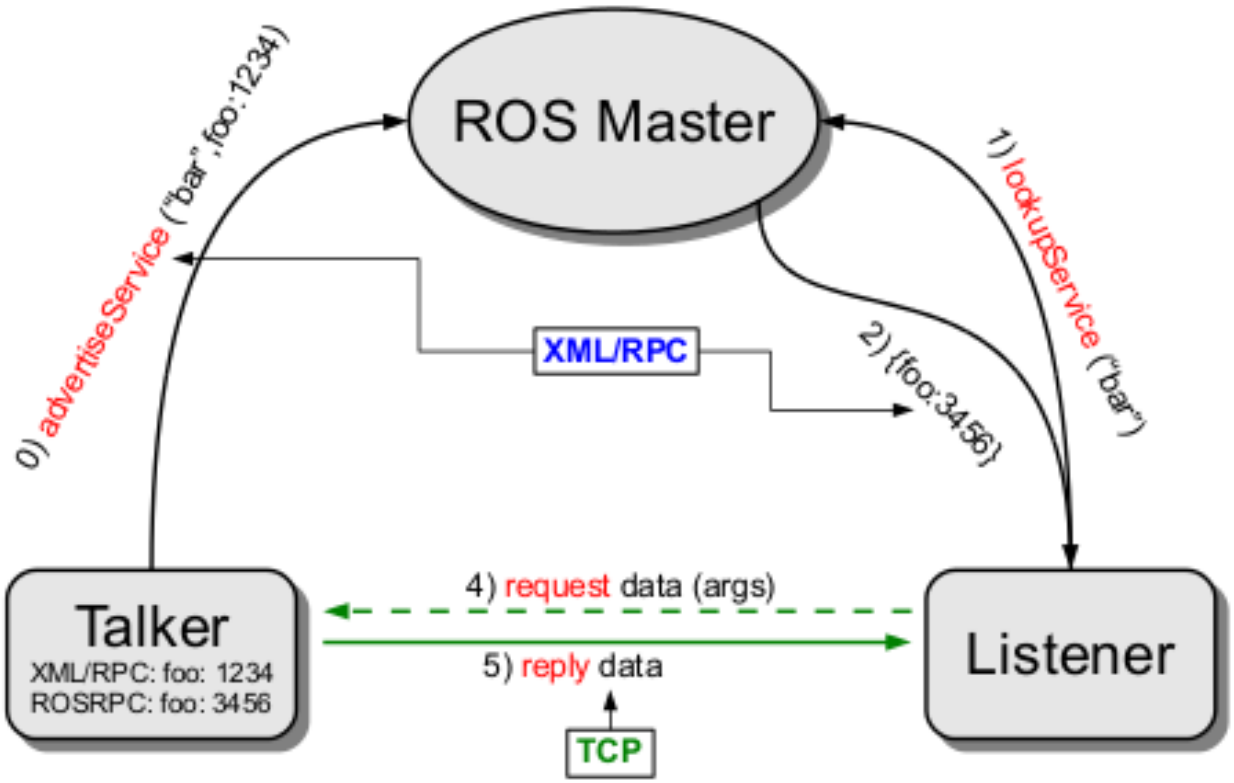
Topics – diagrammatic representation



Services

- Publish/subscribe paradigm not appropriate for services
- Services implement the request/reply functionality
- Pair of message structures: one for request and one for reply
- A *node* provider offers a service under a specific name
- A client *node* uses the service by sending the request message and awaits for the reply
- From the programmer perspective, works as a remote procedure call

Services – diagrammatic representation

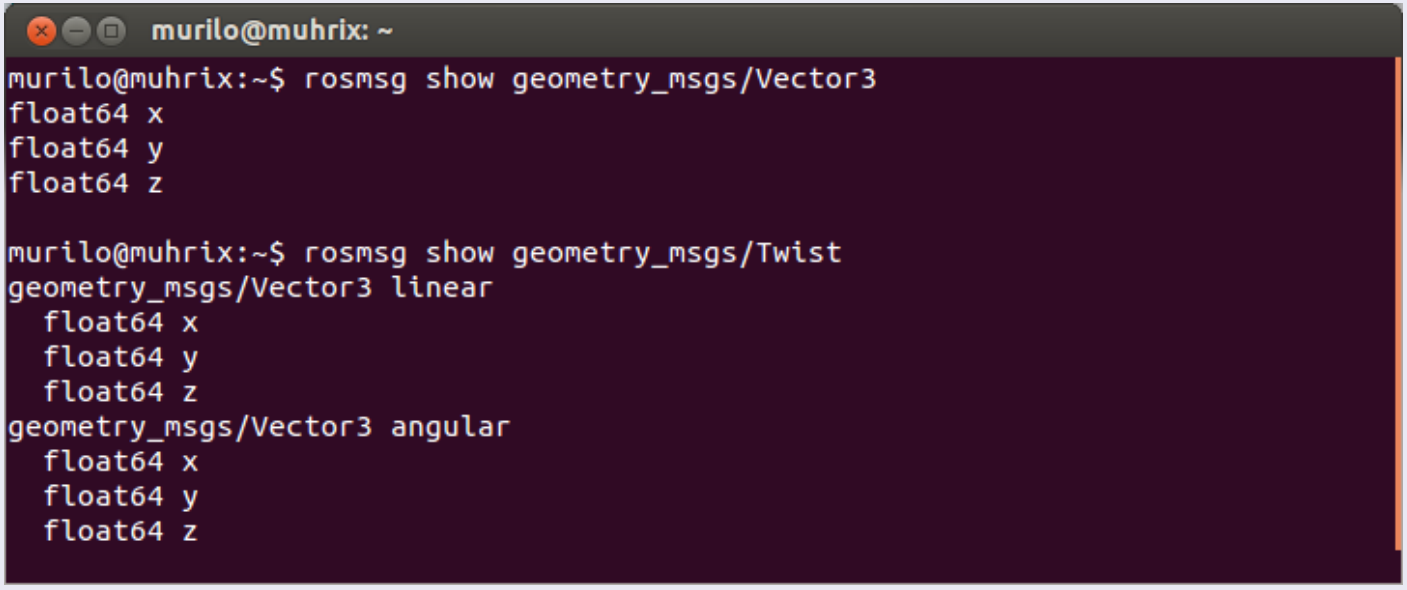


Messages – more ROS command line goodies

Messages over Topics

```
1  rosmsg list
2  rosmsg show geometry_msgs/Vector3
3  rosmsg show geometry_msgs/Twist
```

Vector3.msg and Twist.msg, from package geometry_msgs

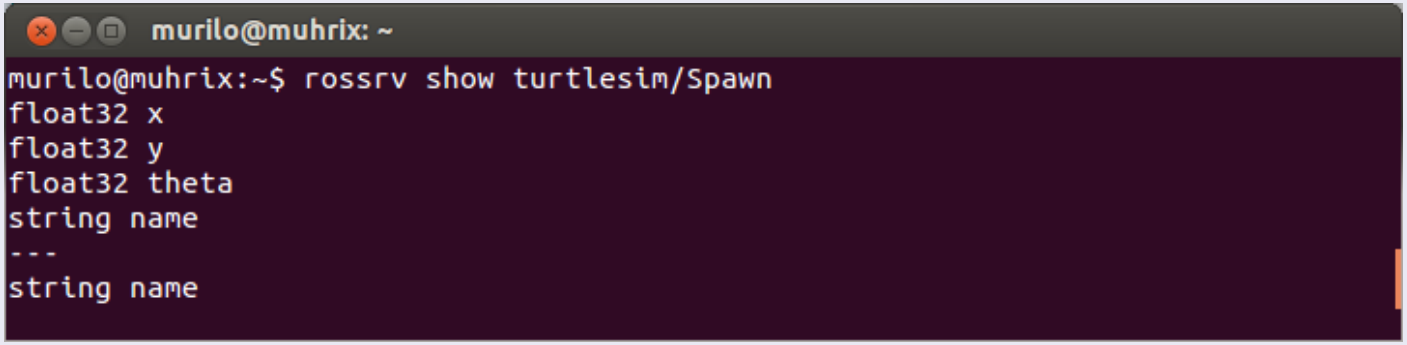


Messages – more ROS command line goodies

Messages over Services

```
1  rossrv list
2  rossrv show turtlesim/Spawn
```

Spawn.srv, from package turtlesim



ROS filesystem – Package structure revisited

Hypothetical package myPkg/

- CmakeLists.txt: CMake build settings for package myPkg
- manifest.xml: metadata and dependencies required by package
- mainpage.dox: doc information of package myPkg
- include/myPkg: C++ header files
- src/: source code directory
- bin/: compiled binaries directory
- launch/: where launch files are stored (if needed)
- msg/: message (.msg) types
- srv/: service (.srv) types
- scripts/: executable scripts

That's all for today, folks!

Today we have learnt

- How awesome (and complex) ROS is
- Why we should surrender to Ubuntu
- How easy it is to install (on Ubuntu)
- The power of rosbash
- The ROS filesystem concepts
- The ROS Graph concepts (most important lesson of today!)

Question time

Thank you very much!
Questions?