ROS-I Basic Training “Mobile Manipulation”

ROS Filesystem

M.Stuettgen, N. Limpert

Mobile Autonomous Systems and Cognitive Robotics Institute (MASCOR)

Aug 8, 2017

Learning Objectives

You will learn

- about the structure of catkin workspaces and catkin packages

Learning Objectives

You will learn

- about the structure of catkin workspaces and catkin packages
- how to create a catkin workspace and a catkin package

Learning Objectives

You will learn

- about the structure of catkin workspaces and catkin packages
- how to create a catkin workspace and a catkin package
- how to include repositories into your ROS workspace

Learning Objectives

You will learn

- about the structure of catkin workspaces and catkin packages
- how to create a catkin workspace and a catkin package
- how to include repositories into your ROS workspace
- how to write your own custom ROS message

Learning Objectives

You will learn

- about the structure of catkin workspaces and catkin packages
- how to create a catkin workspace and a catkin package
- how to include repositories into your ROS workspace
- how to write your own custom ROS message
- how to use the ROS-Cmake-Toolchain to compile C++ ROS nodes

Structure of a catkin workspace

A catkin workspace has the following structure:

- the workspace root (e.g. /home/user/catkin_ws)

Structure of a catkin workspace

A catkin workspace has the following structure:

- the workspace root (e.g. /home/user/catkin_ws)
- three subfolders:
  - src : contains ROS packages and their source code
  - devel : compiled binaries and libraries are deployed here
  - build : build folder of the CMake-Toolchain

http://wiki.ros.org/catkin/Tutorials/create_a_workspace

Setting up a catkin workspace - I

A catkin workspace can be setup as following:

1. create a workspace folder and the subfolder src
2. invoke the caktin_make command (nothing will be compiled, but all folders will be initialized)
3. source the devel folder of the workspace by invoking the source command

Setting up a catkin workspace - I

A catkin workspace can be setup as following:

1. create a workspace folder and the subfolder src
2. invoke the caktin_make command (nothing will be compiled, but all folders will be initialized)
3. source the devel folder of the workspace by invoking the source command

Example usage:

```
$ cd ~
$ mkdir -p catkin_ws/src
$ cd catkin_ws
$ catkin_make
$ source ~/catkin_ws/devel/setup.bash
```

http://wiki.ros.org/catkin/Tutorials/create_a_workspace

Setting up a catkin workspace - II

It is recommended to automate the sourcing of the desired workspace or ROS version by configuring the .bashrc file in your home directory. Some benefits are

- fast switching between different ROS versions

Setting up a catkin workspace - II

It is recommended to automate the sourcing of the desired workspace or ROS version by configuring the .bashrc file in your home directory. Some benefits are

- fast switching between different ROS versions
- fast switching between workspaces

Setting up a catkin workspace - II

It is recommended to automate the sourcing of the desired workspace or ROS version by configuring the .bashrc file in your home directory. Some benefits are

- fast switching between different ROS versions
- fast switching between workspaces
- multiple workspaces can be chained into each other depending on the order they were sourced (advanced)

Setting up a catkin workspace - II

It is recommended to automate the sourcing of the desired workspace or ROS version by configuring the .bashrc file in your home directory. Some benefits are

- fast switching between different ROS versions
- fast switching between workspaces
- multiple workspaces can be chained into each other depending on the order they were sourced (advanced)

The standard use-case is using one workspace at a time, e.g. for a specific robot

Setting up a catkin workspace - III

Example .bashrc content:

```
...
  ## ROS VERSION ##
  #source /opt/ros/indigo/setup.bash
  source /opt/ros/kinetic/setup.bash

  ## WORKSPACES ##
  source ~/catkin_ws/devel/setup.bash
  #source ~/robotino_ws/devel/setup.bash
  #source ~/husky_ws/devel/setup.bash
...
```

Structure of a catkin package - I

A catkin package has the following structure:

- the package root (e.g. ~/catkin_ws/src/mypackage)

Structure of a catkin package - I

A catkin package has the following structure:

- the package root (e.g. ~/catkin_ws/src/mypackage)
- multiple subfolders, depending on what the package contains:
  - src : c++ source code for ROS nodes or libraries

Structure of a catkin package - I

A catkin package has the following structure:

- the package root (e.g. ~/catkin_ws/src/mypackage)
- multiple subfolders, depending on what the package contains:
  - src : c++ source code for ROS nodes or libraries
  - nodes : python code for ROS nodes

Structure of a catkin package - I

A catkin package has the following structure:

- the package root (e.g. ~/catkin_ws/src/mypackage)
- multiple subfolders, depending on what the package contains:
  - src : c++ source code for ROS nodes or libraries
  - nodes : python code for ROS nodes
  - launch : roslaunch files

Structure of a catkin package - I

A catkin package has the following structure:

- the package root (e.g. ~/catkin_ws/src/mypackage)
- multiple subfolders, depending on what the package contains:
  - src : c++ source code for ROS nodes or libraries
  - nodes : python code for ROS nodes
  - launch : roslaunch files
  - msg : definition of custom ROS messages

Structure of a catkin package - I

A catkin package has the following structure:

- the package root (e.g. ~/catkin_ws/src/mypackage)
- multiple subfolders, depending on what the package contains:
  - src : c++ source code for ROS nodes or libraries
  - nodes : python code for ROS nodes
  - launch : roslaunch files
  - msg : definition of custom ROS messages
  - srv : definition of custom ROS services

Structure of a catkin package - I

A catkin package has the following structure:

- the package root (e.g. ~/catkin_ws/src/mypackage)
- multiple subfolders, depending on what the package contains:
  - src : c++ source code for ROS nodes or libraries
  - nodes : python code for ROS nodes
  - launch : roslaunch files
  - msg : definition of custom ROS messages
  - srv : definition of custom ROS services
  - action : definition of custom ROS actions

Structure of a catkin package - I

A catkin package has the following structure:

- the package root (e.g. ~/catkin_ws/src/mypackage)
- multiple subfolders, depending on what the package contains:
  - src : c++ source code for ROS nodes or libraries
  - nodes : python code for ROS nodes
  - launch : roslaunch files
  - msg : definition of custom ROS messages
  - srv : definition of custom ROS services
  - action : definition of custom ROS actions
  - urdf : robot description in **u**niversal **r**obot **d**escription **f**ormat

http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage

Structure of a catkin package - II

- two main files:
  - package.xml: maintainer info and list of ROS dependencies
  - CMakeLists.txt: configuration for the CMake-Toolchain (e.g. compiler settings)

http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage

Creating a catkin package

A catkin package can be created by the following steps:

1. change to the src-folder of your currently used workspace
2. invoke the catkin_create_pkg command

Creating a catkin package

A catkin package can be created by the following steps:

1. change to the src-folder of your currently used workspace
2. invoke the catkin_create_pkg command

Example usage:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg mypackage roscpp rospy
```

Creating a catkin package

A catkin package can be created by the following steps:

1. change to the src-folder of your currently used workspace
2. invoke the catkin_create_pkg command

Example usage:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg mypackage roscpp rospy
```

The example will create a package named *mypackage* and automatically include the ROS C++ and Python dependencies

http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage

Compiling C++ ROS Nodes - I

The following steps are necessary to compile C++ ROS nodes:

1. placing the source code in the src-folder of the package
2. configuring the CMakeLists.txt of the package (compiler/linker)
3. invoking the catkin_make command in the root of the workspace (!)

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

Compiling C++ ROS Nodes - II

Example: to compile *publisher.cpp* and *subscriber.cpp*, the following lines need to be added to CMakeLists.txt:

Compiling C++ ROS Nodes - II

Example: to compile *publisher.cpp* and *subscriber.cpp*, the following lines need to be added to CMakeLists.txt:

```
   ...
#compiler
add_executable(publisher src/publisher.cpp)
add_executable(subscriber src/subscriber.cpp)
#linker
target_link_libraries(publisher ${catkin_LIBRARIES})
target_link_libraries(subscriber ${catkin_LIBRARIES})
   ...
```

Compiling C++ ROS Nodes - II

Example: to compile *publisher.cpp* and *subscriber.cpp*, the following lines need to be added to CMakeLists.txt:

```
  ...
#compiler
add_executable(publisher src/publisher.cpp)
add_executable(subscriber src/subscriber.cpp)
#linker
target_link_libraries(publisher ${catkin_LIBRARIES})
target_link_libraries(subscriber ${catkin_LIBRARIES})
  ...
```

then

```
$ cd ~/catkin_ws
$ catkin_make
```

Running/Launching ROS Nodes

ROS offers two ways of launching ROS Nodes:

1. single node: rosrun <package> <node>
2. multiple nodes: roslaunch <package> <launchfile>

Running/Launching ROS Nodes

ROS offers two ways of launching ROS Nodes:

1. single node: rosrun <package> <node>
2. multiple nodes: roslaunch <package> <launchfile>

Example usage:

```
$ rosrun mypackage publisher
$ rosrun mypackage subscriber
$ roslaunch mypackage example.launch
```

http://wiki.ros.org/rosbash#rosrun

http://wiki.ros.org/roslaunch/XML

Launchfiles - I

- launching multiple nodes or a whole system at once
- better configuration management (param server, yaml-files)
- XML syntax

Launchfiles - I

- launching multiple nodes or a whole system at once
- better configuration management (param server, yaml-files)
- XML syntax

Example Launch file:

```
<?xml version="1.0"?>
<launch>
  <node pkg="mypackage" type="publisher" name="publisher"/>
  <node pkg="mypackage" type="subscriber" name="subscriber"/>
</launch>
```

http://wiki.ros.org/roslaunch/XML

Launchfiles - II

Easy remapping of topics:

```
<?xml version="1.0"?>
<launch>
  <node pkg="gmapping" type="slam_gmapping" name="gmapping">
    <remap from="scan" to="youbot/scan_front" />
    <remap from="base_link" to="youbot/base_link" />
    <remap from="odom" to="youbot/odom" />
  </node>
</launch>
```

http://wiki.ros.org/roslaunch/XML

Launchfiles - III

Launch files can include other launch files:

```
<?xml version="1.0"?>
<launch>
    <include file="$(find some_package)/launch/launchfile.launch"
</launch>
```

http://wiki.ros.org/roslaunch/XML

Messages

Lots of build-in message types, e.g.:

- std_msgs
- sensor_msgs
- nav_msgs
- geometry_msgs
- ...

API for creating own messages → Tutorial

http://wiki.ros.org/msg

Services

Services

- provide a request / reply mechanism
- are defined by two messages
  - request message
  - reply message
- are provided by a ROS node
- are called by a client node, sending a request message

Services

Services

- provide a request / reply mechanism
- are defined by two messages
  - request message
  - reply message
- are provided by a ROS node
- are called by a client node, sending a request message

Best use for instantaneous tasks, e.g.:

- retrieve a single datum (map, single image)
- camera tilt

API for creating own services

http://wiki.ros.org/Services

Actions

Actions work the same way as services, but

- are defined by three messages:
  - Goal
  - Feedback
  - Result
- are called by a client node, sending a goal
- provide feeback about the current state of the action
- can be canceled
- can be preempted

Actions

Actions work the same way as services, but

- are defined by three messages:
  - Goal
  - Feedback
  - Result
- are called by a client node, sending a goal
- provide feeback about the current state of the action
- can be canceled
- can be preempted

Best use for longterm, complex tasks, e.g.:

- Navigation
- Gripper Action

API for creating own actions

http://wiki.ros.org/actionlib

Repository handling

Easy handling, if repository is a ROS package or a meta package:

1. go to src folder of your workspace
2. clone/checkout your repository
3. go to root of workspace and invoke catkin_make

Repository handling

Easy handling, if repository is a ROS package or a meta package:

1. go to src folder of your workspace
2. clone/checkout your repository
3. go to root of workspace and invoke catkin_make

Example usage (git):

```
$ cd ~/catkin_ws/src
$ git clone git@maskor.fh-aachen.de:maskor_allegro
$ cd ~/catkin_ws/
$ catkin_make
```

Metapackages

- specialized package in ROS/catkin
- does not contain any tests, code, files, or other items
- references one or more related packages which are loosely grouped together
- requires special boiler-plate CMakeLists.txt
- requires <metapackage/> in the exports of package.xml
- can only have run dependencies on packages of which they group

http://wiki.ros.org/Metapackages http://wiki.ros.org/catkin/package.xml#Metapackages