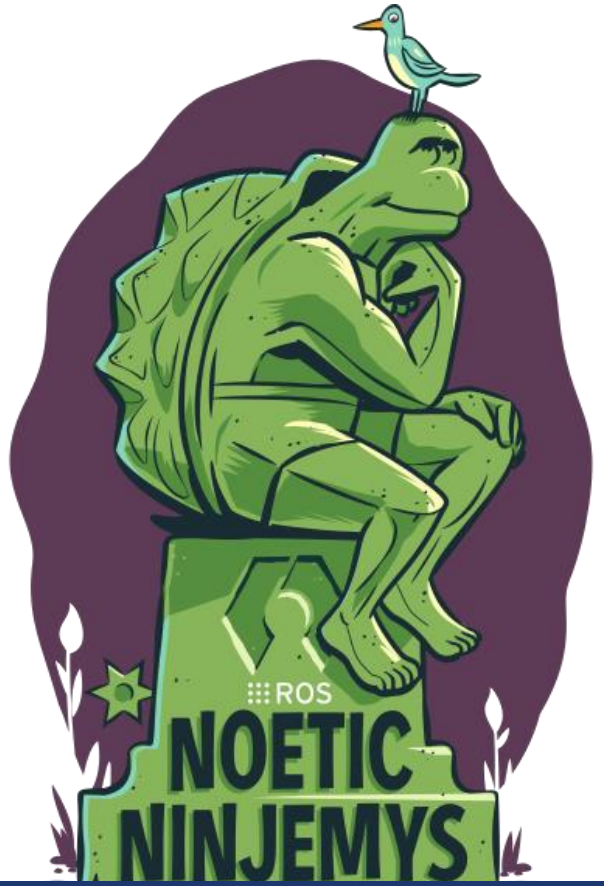


2021

INFO 802

Master Advanced Mechatronics

Luc Marechal

UNIVERSITÉ SAVOIE MONT BLANC

POLYTECH ANNECY-CHAMBÉRY

ROS

Environment Setup Tutorial 1

Exercice 1 - Bash

1– What is the bashrc file ?

2 – Where is located your bashrc file?

3 – Edit your bashrc file and add *your country name* in the following environment variable to your system:

MY_COUNTRY

>

4 – Source your environment

>

5 – Check that your variable exists with the command: ***echo***

>

Exercice 2 – Create your catkin workspace

1 – Create a catkin ROS Workspace named : *catkin_ws*

Explain each command

If you look in your current directory you should now have a 'build' and 'devel' folder.

Inside the 'devel' folder you can see that there are now several setup.*sh files.

Sourcing any of these files will overlay this workspace on top of your environment.

More info

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

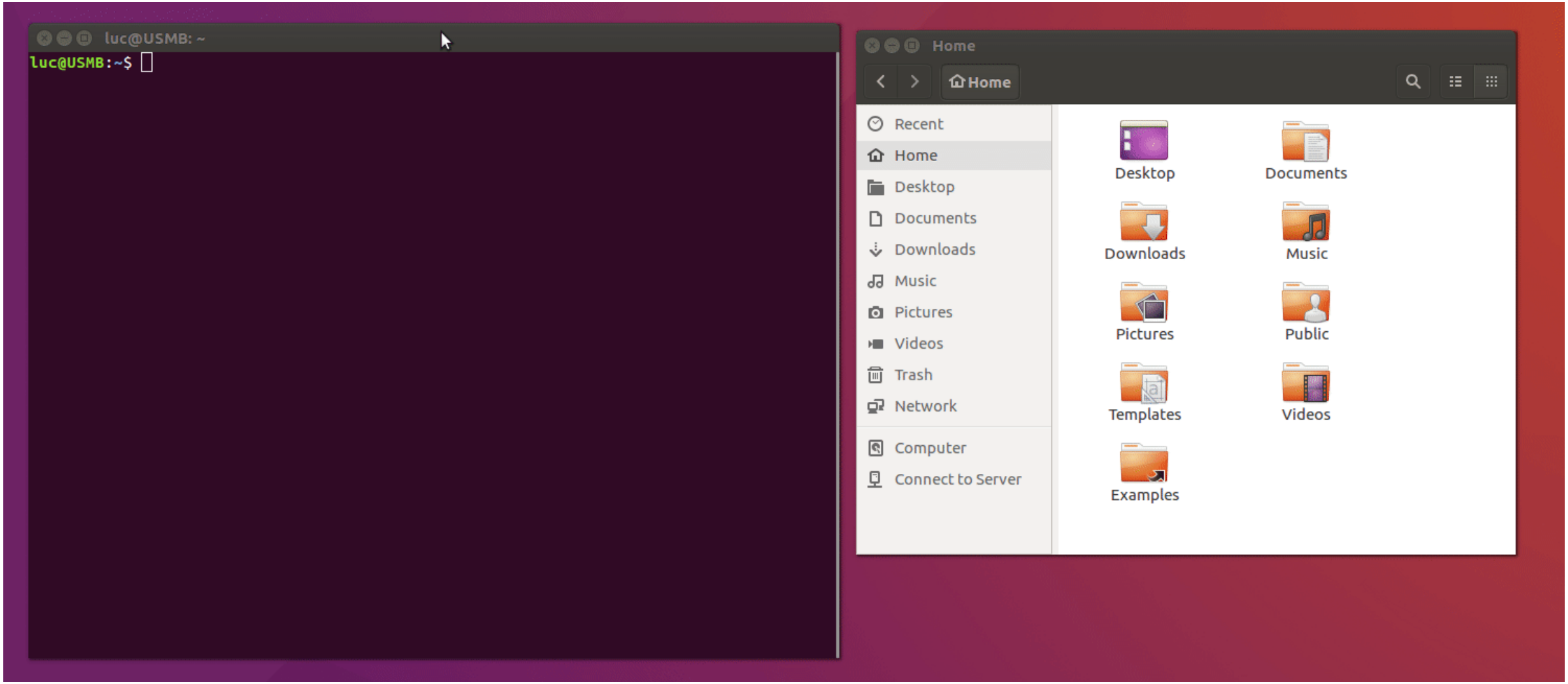
Exercice 2 – Create your catkin workspace

2 – Before continuing, source your new setup.*sh file

Explain what it does

3 – Check your workspace is properly overlayed by the setup script, make sure *ROS_PACKAGE_PATH* environment variable includes the directory you're in.

Exercice 2 – Create your catkin workspace: *Catkin_ws*



ROS System File

Commands

Get information on packages

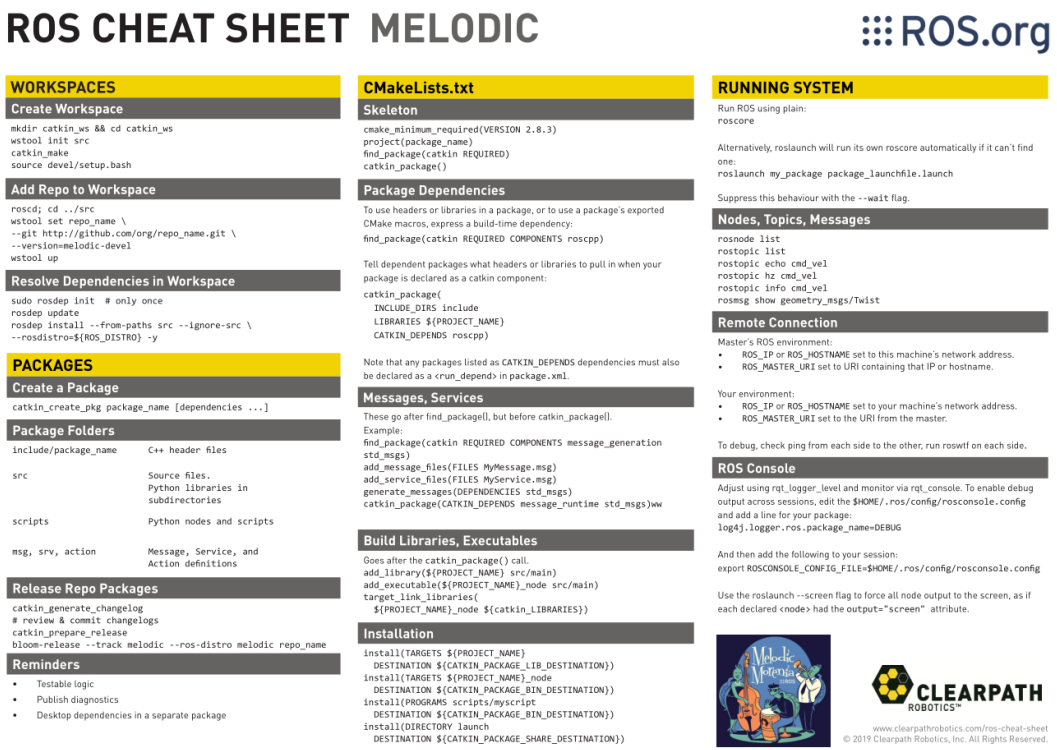
```
> rospack find [package_name]
```

Change directory (cd) directly to a package or a stack

```
> roscd [location_name[/subdir]]
```

ls directly in a package by name rather than by absolute path

```
> rosls [location_name[/subdir]]
```



ROS CHEAT SHEET MELODIC

ROS.org

WORKSPACES

Create Workspace

```
mkdir catkin_ws && cd catkin_ws
wstool init src
catkin_make
source devel/setup.bash
```

Add Repo to Workspace

```
roscd; cd ../src
wstool set repo_name \
--git http://github.com/org/repo_name.git \
--version=melodic-devel
wstool up
```

Resolve Dependencies in Workspace

```
sudo rosdep init  # only once
rosdep update
rosdep install --from-paths src --ignore-src \
--rosdistro=${ROS_DISTRO} -y
```

PACKAGES

Create a Package

```
catkin_create_pkg package_name [dependencies ...]
```

Package Folders

include/package_name	C++ header files
src	Source files. Python libraries in subdirectories
scripts	Python nodes and scripts
msg, srv, action	Message, Service, and Action definitions

Release Repo Packages

```
catkin_generate_changelog
# review & commit changelogs
catkin_prepare_release
bloom-release --track melodic --ros-distro melodic repo_name
```

Reminders

- Testable logic
- Publish diagnostics
- Desktop dependencies in a separate package

CMakeLists.txt

Skeleton

```
cmake_minimum_required(VERSION 2.8.3)
project(package_name)
find_package(catkin REQUIRED)
catkin_package()
```

Package Dependencies

To use headers or libraries in a package, or to use a package's exported CMake macros, express a build-time dependency:

```
find_package(catkin REQUIRED COMPONENTS roscpp)
```

Tell dependent packages what headers or libraries to pull in when your package is declared as a catkin component:

```
catkin_package(
  INCLUDE_DIRS include
  LIBRARIES ${PROJECT_NAME}
  CATKIN_DEPENDS roscpp)
```

Note that any packages listed as CATKIN_DEPENDS dependencies must also be declared as a <run_depend> in package.xml.

Messages, Services

These go after find_package(), but before catkin_package().
Example:

```
find_package(catkin REQUIRED COMPONENTS message_generation
std_msgs)
add_message_files(FILES MyMessage.msg)
add_service_files(FILES MyService.msg)
generate_messages(DEPENDENCIES std_msgs)
catkin_package(CATKIN_DEPENDS message_runtime std_msgs)ww
```

Build Libraries, Executables

Goes after the catkin_package() call.

```
add_library(${PROJECT_NAME} src/main)
add_executable(${PROJECT_NAME}_node src/main)
target_link_libraries(
  ${PROJECT_NAME}_node ${catkin_LIBRARIES})
```

Installation

```
install(TARGETS ${PROJECT_NAME}
  DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION})
install(TARGETS ${PROJECT_NAME}_node
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
install(PROGRAMS scripts/myscript
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
install(DIRECTORY launch
  DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
```

RUNNING SYSTEM

Run ROS using plain:

```
roscore
```

Alternatively, roslaunch will run its own roscore automatically if it can't find one:

```
roslaunch my_package package_launchfile.launch
```

Suppress this behaviour with the --wait flag.

Nodes, Topics, Messages

```
rosnode list
rostopic list
rostopic echo cmd_vel
rostopic hz cmd_vel
rostopic info cmd_vel
rosmsg show geometry_msgs/Twist
```

Remote Connection

Master's ROS environment:

- ROS_IP or ROS_HOSTNAME set to this machine's network address.
- ROS_MASTER_URI set to URI containing that IP or hostname.

Your environment:

- ROS_IP or ROS_HOSTNAME set to your machine's network address.
- ROS_MASTER_URI set to the URI from the master.

To debug, check ping from each side to the other, run roswtf on each side.

ROS Console

Adjust using rqt_logger_level and monitor via rqt_console. To enable debug output across sessions, edit the $HOME/.ros/config/rosconsole.config and add a line for your package:

```
log4j.logger.ros.package_name=DEBUG
```

And then add the following to your session:

```
export ROSCONSOLE_CONFIG_FILE=$HOME/.ros/config/rosconsole.config
```

Use the roslaunch --screen flag to force all node output to the screen, as if each declared <node> had the output="screen" attribute.

CLEARPATH ROBOTICS™

www.clearpathrobotics.com/ros-cheat-sheet

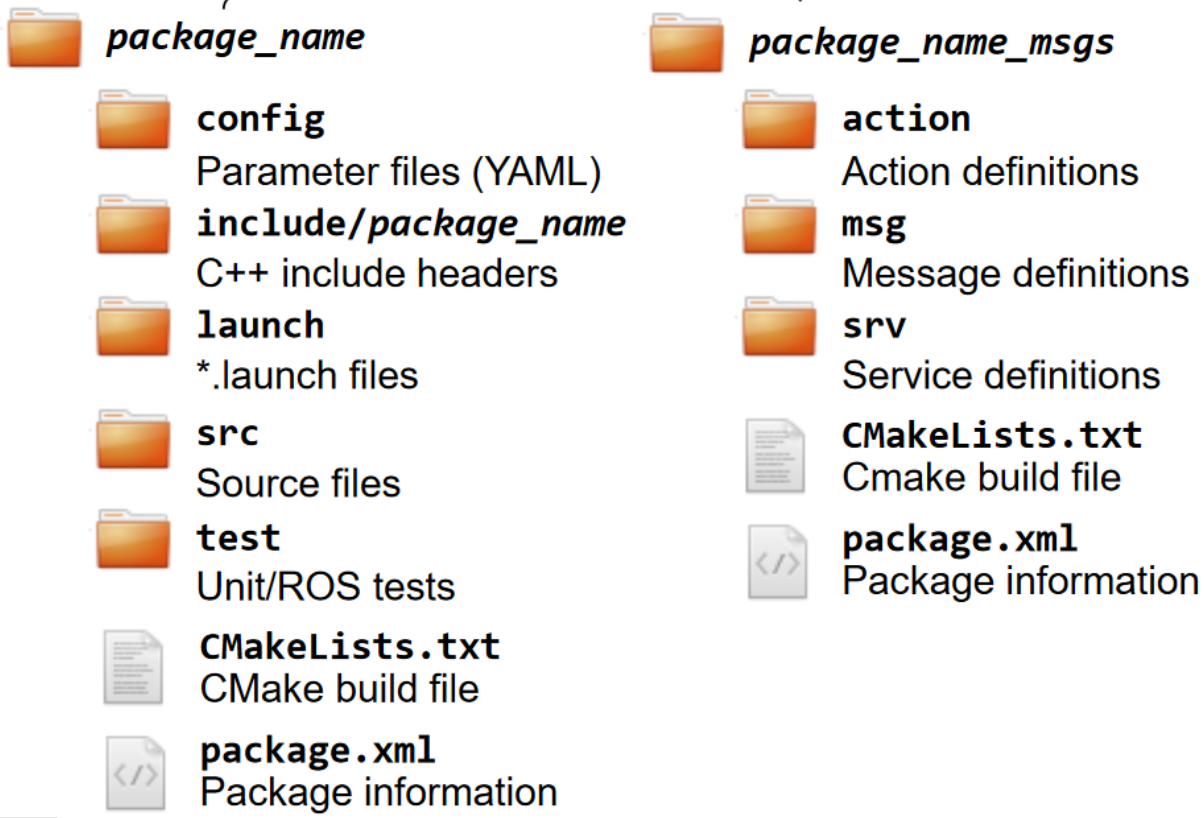
More info

http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

Exercice 3 – Create a ROS Package

- ROS software is organized into packages, which can contain source code, launch files, configuration files, message definitions, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as dependencies

Separate message definition packages from other packages!



To create a new package, use

```
> cd ~/catkin_ws/src
> catkin_create_pkg package_name {dependencies}
```

More info
http://wiki.ros.org/Packages

Exercice 3 – Create a ROS Package

1 – Create a ROS package named : *beginner_tutorials_pkg*

```
> cd ~/catkin_ws/src
> catkin_create_pkg beginner_tutorials_pkg rospy
```

2 - Whenever you build a new package, update your environment

```
> source devel/setup.bash
```

More info
http://wiki.ros.org/Packages

ROS Package

package.xml

- The package.xml defines properties of the package
  - <name> Package name
  - <version> Version numbers
  - <description> Description of the content
  - <maintainer> Person maintaning the package
  - <licence> Type of licence (usualy BSD)
  - **Dependencies on other catkin packages**

    The dependencies are split into :
    build_depend, buildtool_depend, exec_depend, test_depend

  - and more

```
<?xml version="1.0"?>
<package format="2">
<name>ros_package_template</name>
<version>0.1.0</version>
<description>A template for ROS packages.</description>
<maintainer email="luc.marechal@univ-smb.fr">Luc</maintainer>
<license>BSD</license>

<url type="website">https://github.com/my_project/ros_…</url>
<author email="luc.marechal@univ-smb.fr">Luc Mare</author>

<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>

<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>

<run_depend>roscpp</run_depend>
<run_depend>rospy</run_depend>
<run_depend>std_msgs</run_depend>
</package>
```

ROS Package

CMakeLists.txt

The `CMakeLists.txt` is the input to the CMakebuild system

1. Required CMake Version (`cmake_minimum_required`)
2. Package Name (`project()`)
3. Find other CMake/Catkin packages needed for build (`find_package()`)
4. Message/Service/Action Generators (`add_message_files()`, `add_service_files()`, `add_action_files()`)
5. Invoke message/service/action generation (`generate_messages()`)
6. Specify package build info export (`catkin_package()`)
7. Libraries/Executables to build (`add_library()/add_executable()/target_link_libraries()`)
8. Tests to build (`catkin_add_gtest()`)
9. Install rules (`install()`)

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_package_template)

## Use C++11
add_definitions(--std=c++11)

## Find catkin macros and libraries
find_package(catkin REQUIRED
  COMPONENTS
    roscpp
    sensor_msgs
)

…
```

More info
http://wiki.ros.org/catkin/CMakeLists.txt

ROS Package

CMakeLists.txt Example

```
cmake_minimum_required(VERSION 2.8.3)
project(husky_highlevel_controller)
add_definitions(--std=c++11)

find_package(catkin REQUIRED
  COMPONENTS roscpp sensor_msgs
)

catkin_package(
  INCLUDE_DIRS include
  # LIBRARIES
  CATKIN_DEPENDS roscpp sensor_msgs
  # DEPENDS
)

include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(${PROJECT_NAME} src/${PROJECT_NAME}_node.cpp
src/HuskyHighlevelController.cpp)

target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES})
```

Use the same name as in the `package.xml`

We use C++11 by default

List the packages that your package requires to build (have to be listed in `package.xml`)

Specify build export information

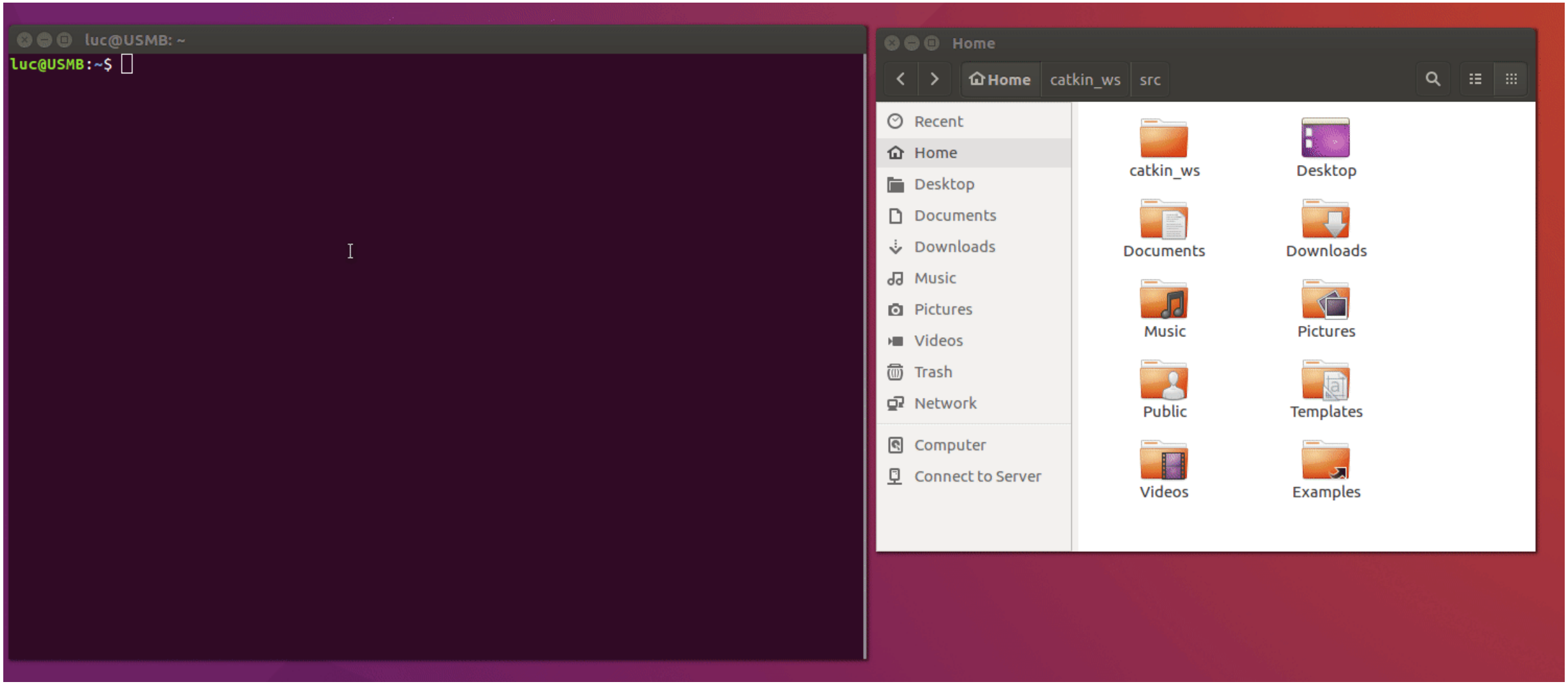
- `INCLUDE_DIRS`: Directories with header files
- `LIBRARIES`: Libraries created in this project
- `CATKIN_DEPENDS`: Packages dependent projects also need
- `DEPENDS`: System dependencies dependent projects also need (have to be listed in `package.xml`)

Specify locations of header files

Declare a C++ executable

Specify libraries to link the executable against

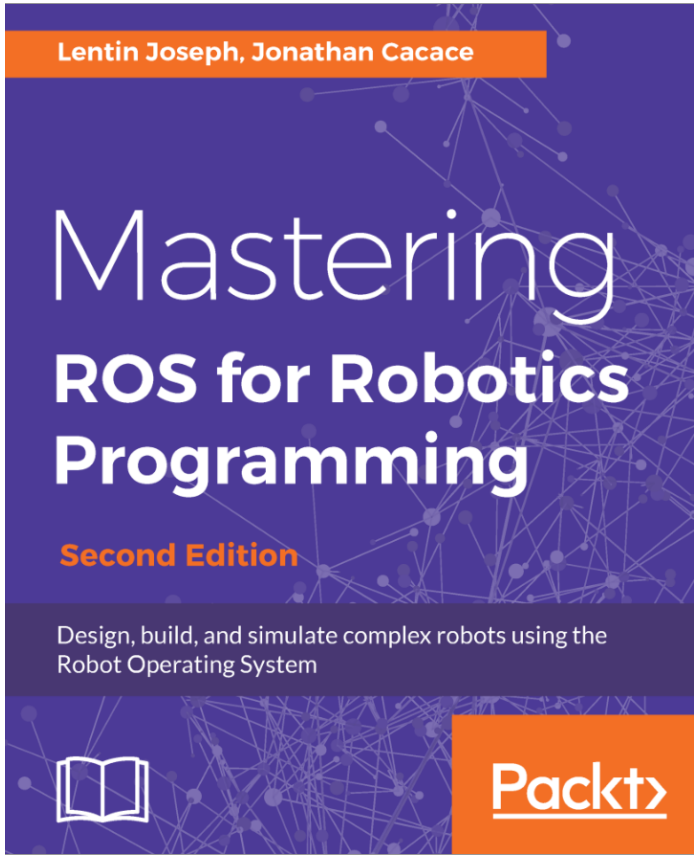
Creating a ROS package : *beginner_tutorials*

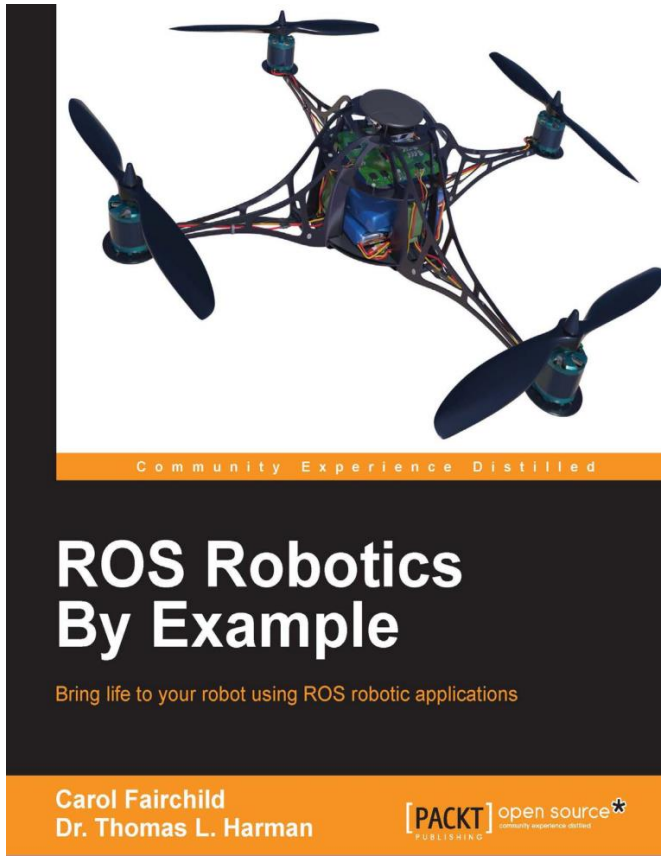


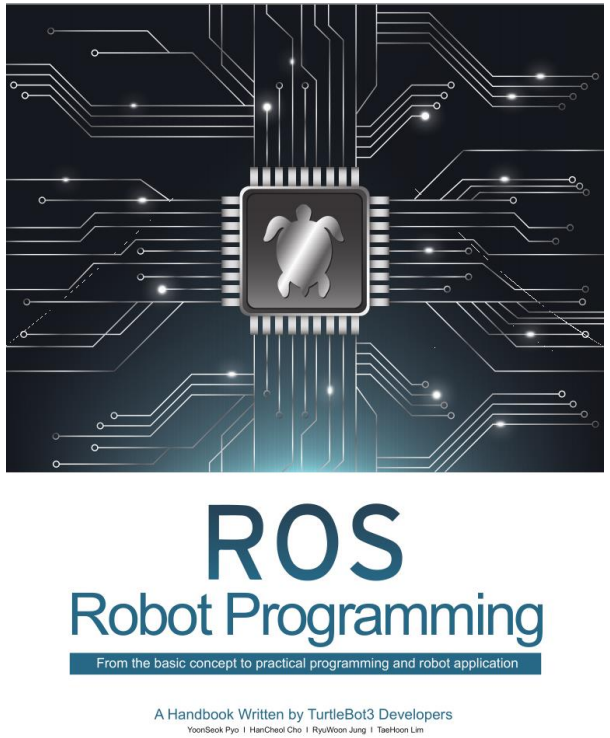
Further References

- ROS Wiki
  - http://wiki.ros.org/
- Installation
  - http://wiki.ros.org/ROS/Installation
- Tutorials
  - http://wiki.ros.org/ROS/Tutorials
- Available packages
  - http://www.ros.org/browse/
- ROS Cheat Sheet
  - https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
  - https://kapeli.com/cheat_sheets/ROS.docset/
- ROS Best Practices
  - https://github.com/leggedrobotics/ros_best_practices/wiki
- ROS Package Template
  - https://github.com/leggedrobotics/ros_best_practices/tree/master/ros_package_template

Relevant books







Contact Information

Université Savoie Mont Blanc

Polytech' Annecy Chambery
Chemin de Bellevue
74940 Annecy
France

https://www.polytech.univ-savoie.fr

Lecturer

Luc Marechal (luc.marechal@univ-smb.fr)
SYMME Lab (Systems and Materials for Mechatronics)

SYMME

UNIVERSITÉ SAVOIE MONT BLANC
POLYTECH ANNECY-CHAMBÉRY