TÉCNICO LISBOA

Institute for Systems and Robotics | LISBOA

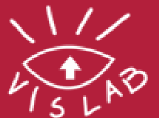
Short course on ROS programming 2020

Part 2

Portugal Chapter

João Avelino
Rodrigo Ventura

Instituto Superior Técnico – U. L.
23/11/2020



Outline 1/2

- Intro to Linux cli
- ROS workspace
- Creating ROS packages
- Python vs C++ nodes
- Launchers and parameters
- Messages, services, and actions

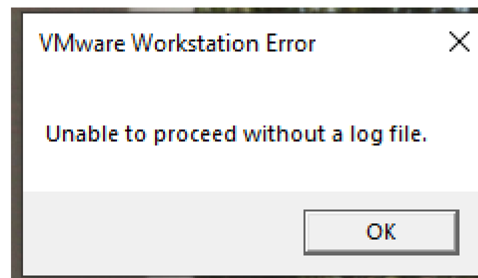
Outline 2/2

- RViz
- TF
- Command line and RQT ROS tools
- Simulation with Gazebo
- Common issues

The virtual machine

- Lightweight version of Ubuntu 18.04 (Bionic Beaver)
- ROS Melodic Morenia
- Credentials
  - Username: ros
  - Password: melodic
- **Launch your VM!**

And don't worry about this "error"



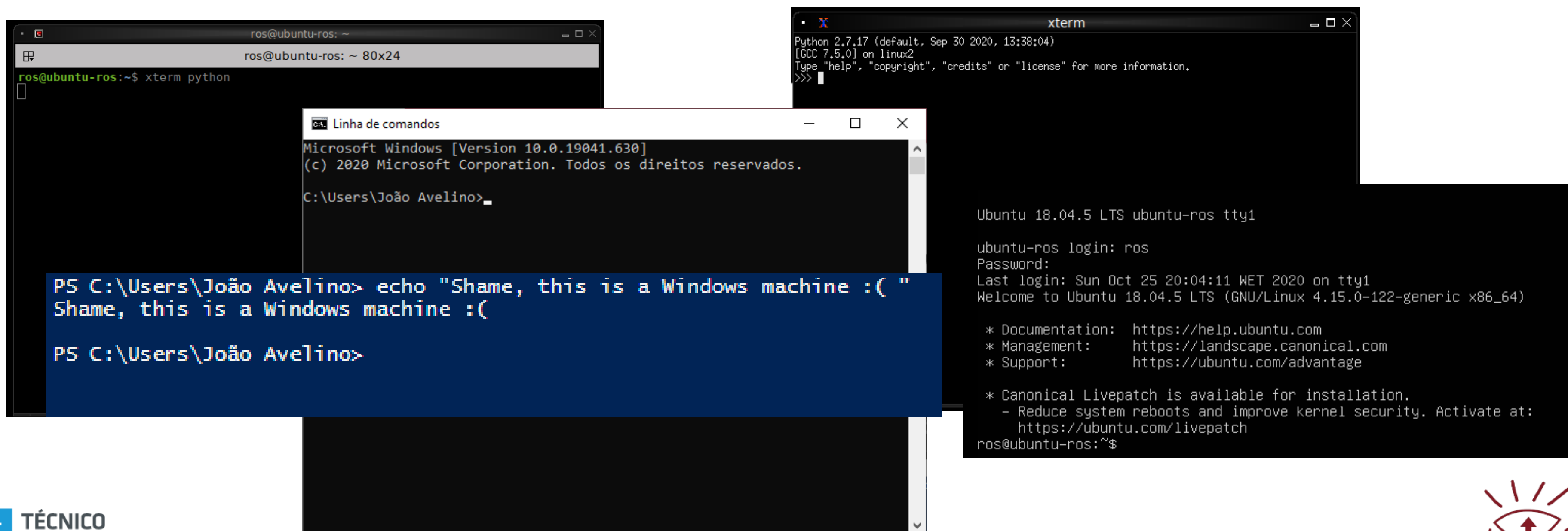
The virtual machine

- **Current state of the VM:** ROS installation and configuration completed (end of this page: http://wiki.ros.org/melodic/Installation/Ubuntu)

A lazy introduction to the Linux command line interface

Command line interface

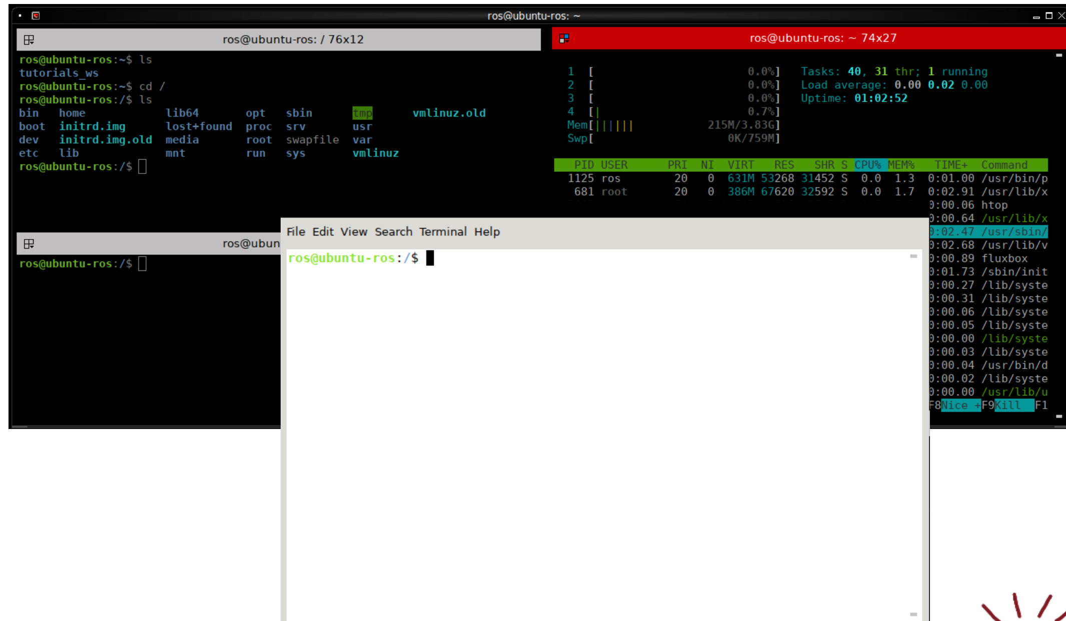
An all-text display in a terminal window provided by a shell



Console / Terminal emulator

The window that shows you the cli, takes what you type, shows you the text

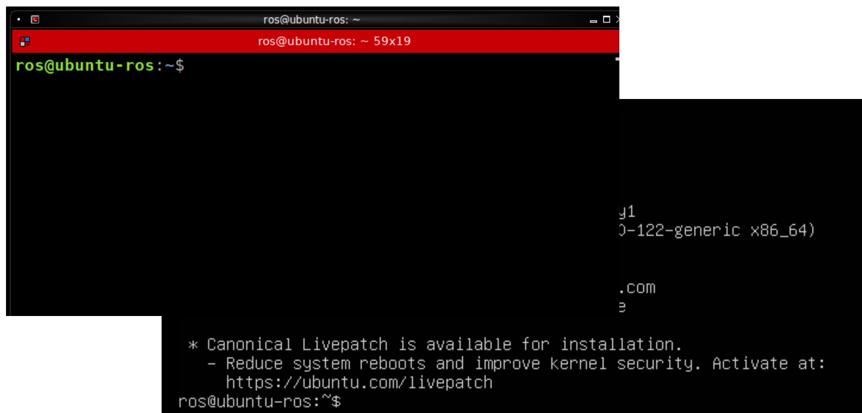
- Examples
  - Terminator
  - Gnome terminal
  - Konsole



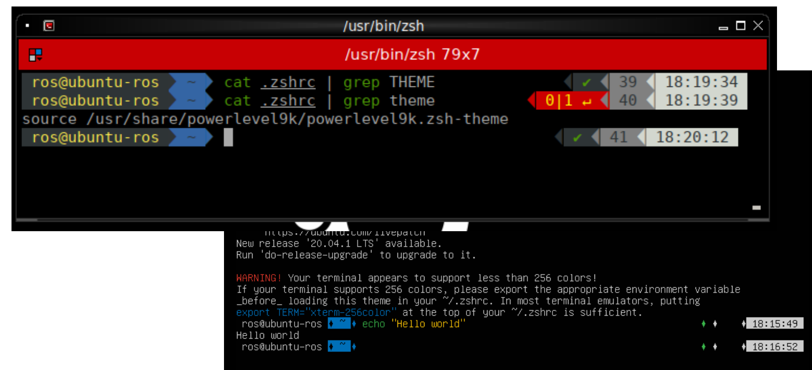
Shell

"…a program that turns the 'text' that you type into commands/orders for your computer to perform. "

@linux.com



BASH is the default interactive shell on Ubuntu



But there are others like zsh

BASH command line interface

A command line.
The **prompt** + your command

The output

Prompt: A short informative text at the start of the command line. Default for Bash: **<username>@<hostname>:<current directory><$** or **#>**

Regular user — User with root

```
ros@ubuntu-ros: ~/tutorials_ws/src
ros@ubuntu-ros: ~/tutorials_ws/src 59x19
ros@ubuntu-ros:~$ ls -a
.                 .gtkrc-2.0          .sudo_as_admin_successful
..                .icons              tutorials_ws
.bash_history     .ignition           .viminfo
.bash_logout      .local              .vscode
.bashrc           .pki                .wget-hsts
.cache            .profile            .Xauthority
.config           .pylint.d           .xsession-errors
.fehbg            .ros                .zcompdump
.fluxbox          .rviz               .zsh_history
.gazebo           .sdformat           .zshrc
.gnupg            .selected_editor
ros@ubuntu-ros:~$ cd tutorials_ws/
ros@ubuntu-ros:~/tutorials_ws$ ls
build  devel  src
ros@ubuntu-ros:~/tutorials_ws$ cd src/
ros@ubuntu-ros:~/tutorials_ws/src$ pwd
/home/ros/tutorials_ws/src
ros@ubuntu-ros:~/tutorials_ws/src$
```

Summary: terminal emulator =/= shell =/= prompt

Quiz time!

Bash basics

man <command>

cd

mkdir

rmdir

rm

ls

<command>&

chmod

TAB: autocompletes

Ctrl+C: sends SIGINT

Ctrl+Z: sends TSTP

echo

ps -aux

source

Useful command line tools

top

kill

grep

nano / vim

cat

less

ssh

apt

stdout, stdin, stderr

Output streams: stdout (1), stderr (2)

Standard output of a command / program

Error messages

Handled as files. You can read from a file and you can write to a file.

Input streams: stdin (0)

Accepts text as input

With streams we can combine multiple commands to achieve complex tasks!

Piping and redirecting output

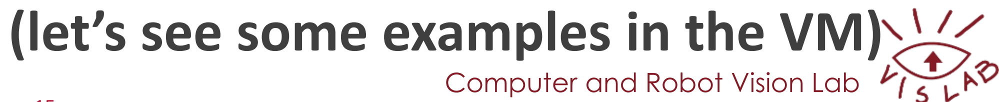
Pipe: connects the STDOUT of one command to the STDIN of another

<command1> | <command2>

Redirects: redirects to/from output/input streams:

<command> <redirect> <file>

<, >, <<, >>, 0>, 1>, 2>, 2>&1



Environment variables

Can be global, user specific, or shell specific

Only available for current shell

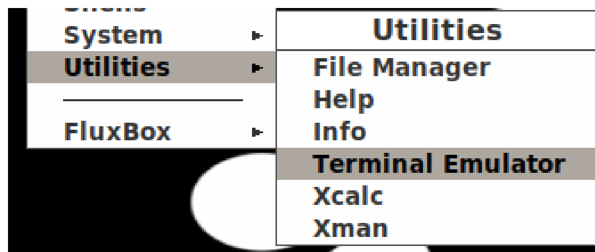
Can be inherited by other programs / shells

Write: VARNAME=value; export VARNAME=value

Read: $VARNAME

```
ros@ubuntu-ros:~$ STUFF="test value"
ros@ubuntu-ros:~$ echo $STUFF
test value
ros@ubuntu-ros:~$ export OTHERSTUFF="other value"
ros@ubuntu-ros:~$ echo $OTHERSTUFF
other value
```

Environment variables - inheritance



Loads system vars and vars on ~/.profile and **~/.bashrc**

```
ros@ubuntu-ros:~$ export ROS_MASTER_URI=http://10.10.3.91:11311
ros@ubuntu-ros:~$ ROS_IP=10.10.3.55
ros@ubuntu-ros:~$ echo $ROS_MASTER_URI
http://10.10.3.91:11311
ros@ubuntu-ros:~$ echo $ROS_IP
10.10.3.55
ros@ubuntu-ros:~$ xterm&
```

```
ros@ubuntu-ros:~$ echo $ROS_MASTER_URI
http://localhost:11311
ros@ubuntu-ros:~$ echo $ROS_IP

ros@ubuntu-ros:~$
```

```
ros@ubuntu-r
ros@ubuntu-ros:~$ echo $ROS_MASTER_URI
http://10.10.3.91:11311
ros@ubuntu-ros:~$ echo $ROS_IP

ros@ubuntu-ros:~$
```

You can load vars from a file with the **source <filename>** command

Add them to ~/.bashrc (or ~/.zshrc if you ush zsh) if you want to load them when the shell opens

BASH is a language.

```
#!/usr/bin/env bash
# Plinio Moreno @ Vislab
ROS_VERSION=melodic
read -p "[Vizzy]: What is your ROS VERSION? Default: $ROS_VERSION   " NEW_ROS_VERSION
if [ ! -z "$NEW_ROS_VERSION" ]; then
  ROS_VERSION=$NEW_ROS_VERSION
  echo "[Vizzy]: I'm going to install ROS $ROS_VERSION"
fi
sudo apt install -y lsb-release
sudo apt install -y add-apt-key

if [ $ROS_VERSION = "melodic" ]; then
  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
  sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
elif [ $ROS_VERSION = "kinetic" ];then
  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
  sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
fi

sudo apt update
sudo apt-get install -y ros-$ROS_VERSION-desktop-full
sudo apt install -y python-rosdep
sudo rosdep init
rosdep update
echo "source /opt/ros/$ROS_VERSION/setup.bash" >> $HOME/.bashrc
printf "\n [Vizzy]: After this script finishes, verify that you execute: source ~/.bashrc in your terminal\n"
```

shebang: choose interpreter should read this script. BASH in this case

This script automatically installs ROS Melodic ☺

End of section!

Quiz time!

Understanding the ROS workspace

ROS packages: where are they?

- System-wide packages installed through the system repositories (via apt, synaptic, others): /opt/ros/melodic
  - Need to load needed variables to use ROS tools

    source /opt/ros/melodic/setup.bash ← It was added to your .bashrc at the end of ROS installation
- Local packages:
  - Download the source code into a workspace in your home
  - Compile, load variables related to workspace

    (source ~/my_ws/devel/setup.bash)

Important variables for ROS

PATH ← Shell looks for executables in this list of paths

PYTHONPATH ← Python looks for packages in this list of paths

ROS_PACKAGE_PATH ← ROS tools look for packages here

ROS_MASTER_URI ← Where is ROS master running?

ROS_IP/ROS_HOSTNAME ← My IP in the ROS network

Each “setup.bash” configures these ones

You might need to set up / change these for your network. Put them on ~/.bashrc

Setup the vars you need to change AFTER sourcing “setup.bash”!

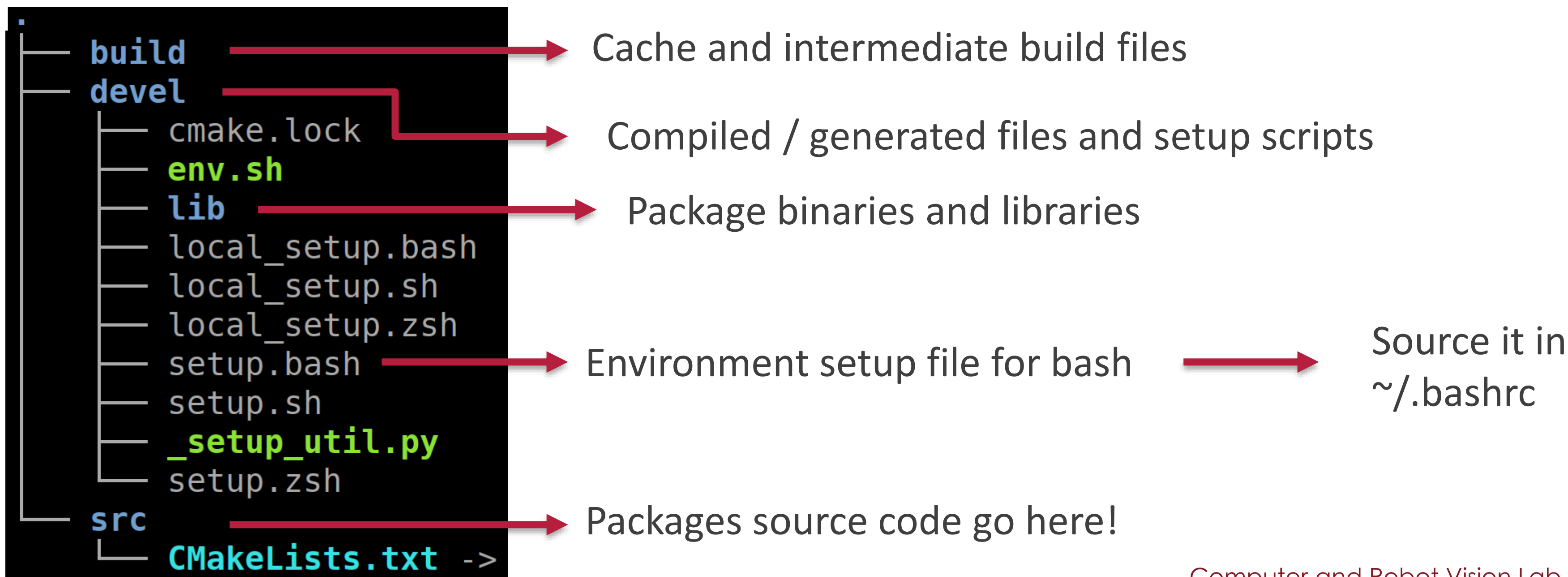
Catkin

- Build system to make roboticists' life easier
- Compiles and generates files for your: **catkin_make**
- Let's create a catkin workspace

(Streaming VM screen)

Catkin workspace

Automagically populated with "catkin_make"



Create a ROS package

catkin_create_pkg <package_name> <dependencies>

(Streaming VM screen)

Structure of a minimal ROS package

package.xml

Package manifest

CMakeLists.txt

CMake build file

include

C/C++ headers

src

Source files

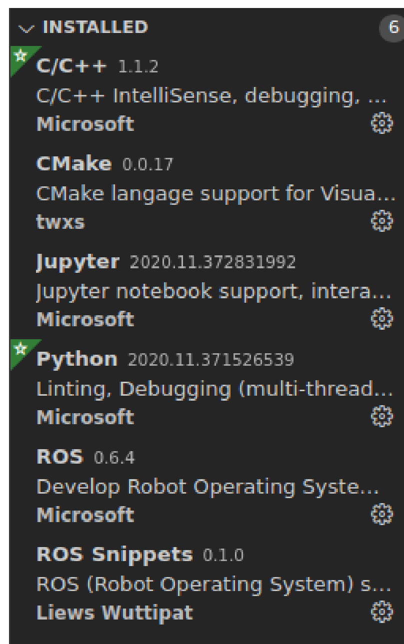
Let’s code

Launch vscode, install some useful extensions for ROS, and open the workspace

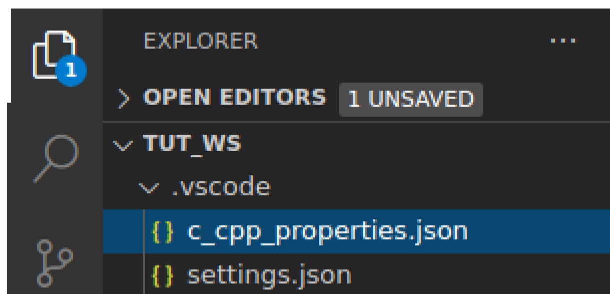
(Streaming VM screen)

Visual Studio Code tips

Useful modules:



Add include directories for linting and code completion:



Add include directories as needed

```
},
"includePath": [
    "/opt/ros/melodic/include/**",
    "/home/ros/tut_ws/src/short_course/include/**",
    "/usr/include/**"
],
```

Command a robot with Python (Publisher)

Let’s create a naïve remote controller for the husky robot

(Streaming VM screen)

Receiving messages in Python (Subscriber)

Create a python node that subscribes two topics

(Streaming VM screen)

Receiving messages in C++ (Subscriber)

Create a C++ node that subscribes two topics

(Streaming VM screen)

How to compile a C++ node: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.2)
project(short_course)
…
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
  ${<otherlibrary_INCLUDE_DIRS},
  …
)
…
```

Your package name

Where to look for headers (*.h, *.hpp)

CMake variable with the location of headers of catkin packages

CMake variable with the location "otherlibrary" headers

We need a way to define these… (we learn how in 2 slides ☺)

How to compile a C++ node: CMakeLists.txt (2)

```
add_executable(executable1_name
src/main_exec1.cpp
src/file_1.cpp
...)
...
target_link_libraries(executable1_name
  ${catkin_LIBRARIES}
  ${anotherlib_LIBRARIES}
 ...)
```

Define your executable and list the required *.cpp files

Link with these libraries

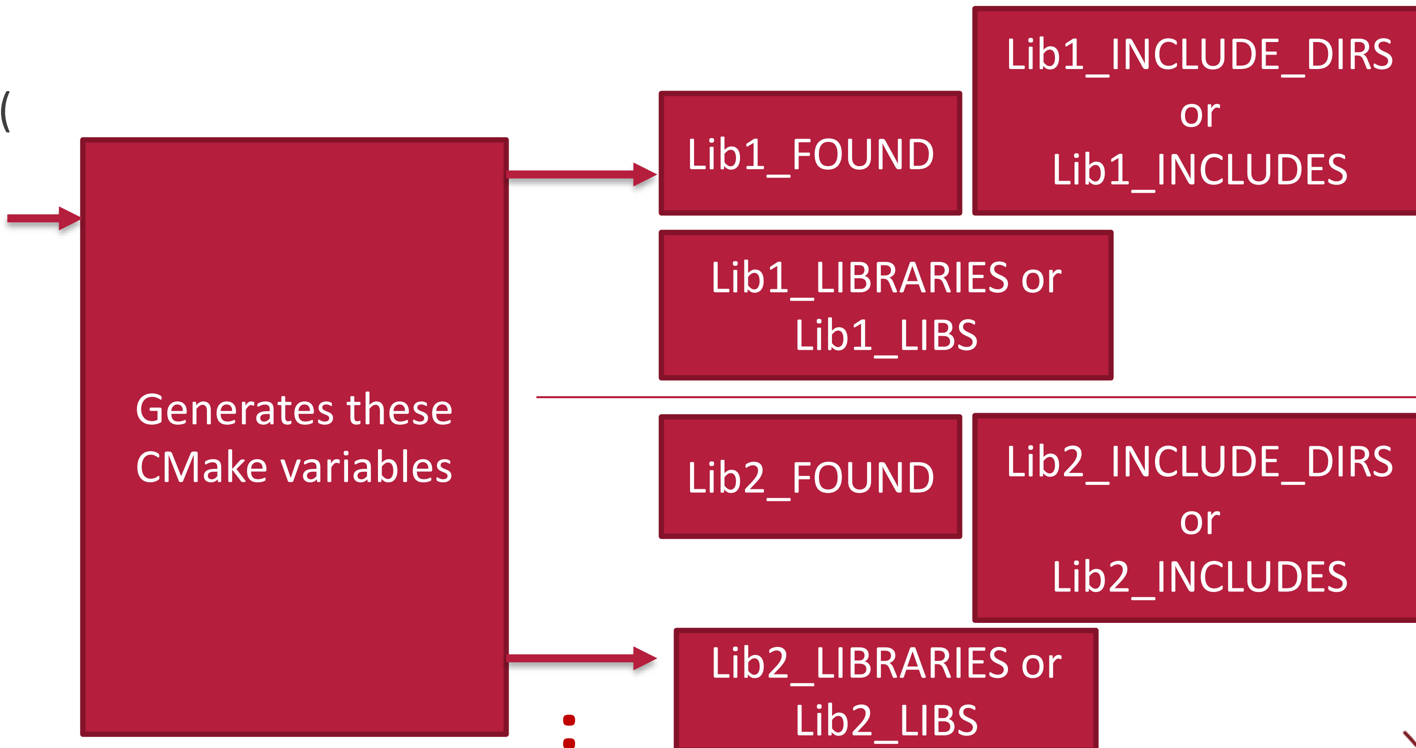
Var with the locations of libraries from catkin packages

Var with the location of another library...

Need more ROS nodes? Then replicate these commands!

Once again, we need to define these!

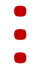
Finding dependencies with CMake –find_package()



CMakeLists.txt – find_package(catkin REQUIRED COMPONENTS) – packages as components of catkin

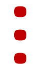
```
find_package(catkin REQUIRED
COMPONENTS
  geometry_msgs
  roscpp
  rospy
  std_msgs
  …
)
```

Generates these CMake variables



catkin_ INCLUDE_DIRS

catkin_LIBRARIES



Vars contain info about all packages in the list!

This way we don't need to add a variable per package ☺

CMakeLists.txt – Detailed description

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

https://gitlab.kitware.com/cmake/community/-/wikis/home

(It can be a course by itself…)

Receiving messages in C++ (Subscriber) - continued

Let’s compile, run, and compare C++ and Python subscribers

(Streaming VM screen)

What we may miss from the tutorials…

C++ and Python subscribers actually behave differently by default!

Subscribers in C++ run in the main thread, while each subscriber in Python has its own thread!

Do you really need one thread per subscriber in C++?

In general it is not advisable to abuse the number of threads. Create them only as necessary…

But if you really need, replace ros::spin() by:

```
ros::MultiThreadedSpinner spinner(2);
spinner.spin();
```

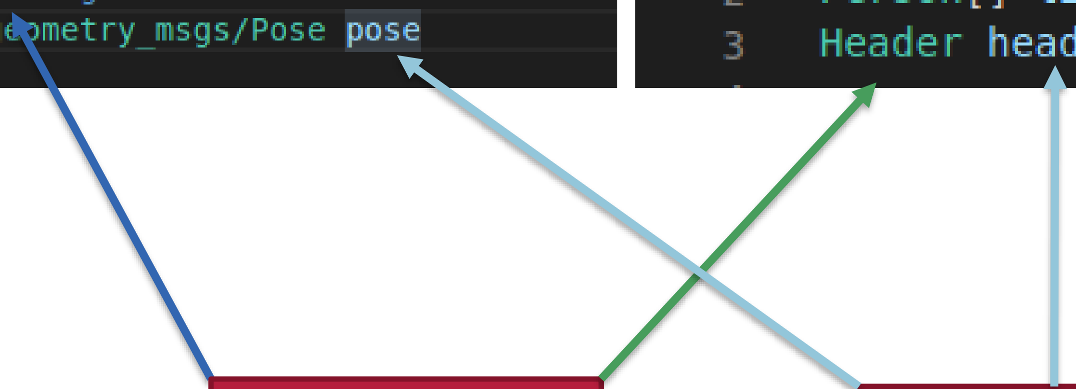
http://wiki.ros.org/roscpp/Overview/Callbacks%20and%20Spinning

Creating new message types

Let’s create new messages in a new package

(Streaming VM screen)

Defining message of Person and People



Generating them (for future reference, text is small)

Add message generation dependencies to package.xml

```
<build_depend>message_generation</build_depend>
<build_export_depend>message_generation</build_export_depend>
<exec_depend>message_runtime</exec_depend>
```

Add messages that we use to build ours as a new dependency

```
<build_depend>geometry_msgs</build_depend>
<build_export_depend>geometry_msgs</build_export_depend>
<exec_depend>geometry_msgs</exec_depend>
```

Configure CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
  geometry_msgs
)
```

```
add_message_files(
  FILES
  Person.msg
  People.msg
)
```

```
catkin_package(
  CATKIN_DEPENDS roscpp rospy std_msgs geometry_msgs message_runtime
)
```

How to use them with other packages

Update the other package's manifest

```
<build_depend>short_course_msgs</build_depend>
<build_export_depend>short_course_msgs</build_export_depend>
<exec_depend>short_course_msgs</exec_depend>
```

Update the other package's CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
  ...
  short_course_msgs
)
```

Crucial for correct package build order!!

```
add_dependencies(subscriber_cpp
  short_course_msgs_generate_messages_cpp
  ${${PROJECT_NAME}_EXPORTED_TARGETS}
  ${catkin_EXPORTED_TARGETS} )
```

Custom services and actions: similar steps

```
request_var_1
request_var_2
request_var_n
---
response_var_1
response_var_2
response_var_n
```

```
add_service_files(
  FILES
  Service1.srv
  Service2.srv
)
```

```
request_var_1
request_var_n
---
feedback_var_1
feedback_var_n
---
response_var_1
response_var_n
```

```
add_action_files(
  FILES
  Action1.action
  Action2.action
)
```

```
find_package(catkin
  REQUIRED COMPONENTS
  ...
  actionlib
  actionlib_msgs
)
```

Services

Explaining with ROS wiki

Services

Actions

Explaining with ROS wiki

Action client

Action server

Final note on subscribers, publishers, service clients, service servers, action clients and action servers

A single ROS node can implement all of them simultaneously.

Example: the move_base node for navigation

Action server: receives new navigation goals

Services: receive requests to clear costmaps and more…

Publisher: Robot velocity controls (cmd_vel)

Subscribers: Sensor information

And more..

A ROS system has many nodes…

Robot: very complex system, many nodes

Do we execute each node separately? NO! We use launchers and the roslaunch command

roslaunch <package_name> <launcher_name.launch>

A simple launcher

Let's create a launcher to run Husky simulation and

(Streaming VM screen)

Say no to hardcoded values

Parameters, roslaunch arguments, and topic remaps

(Streaming VM screen)

RViz

Let’s see some sensor information

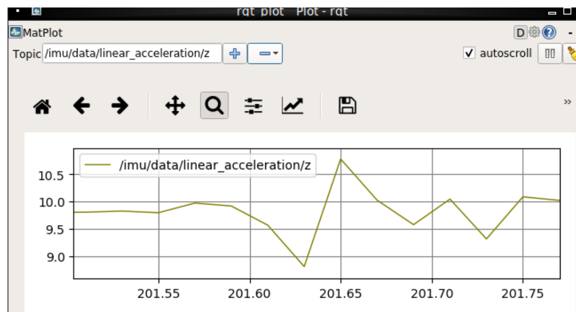
(Streaming VM screen)

Command line tools

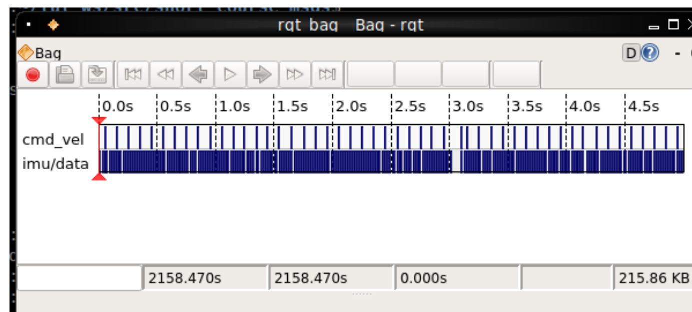
Remember the morning session commands

(Streaming VM screen)

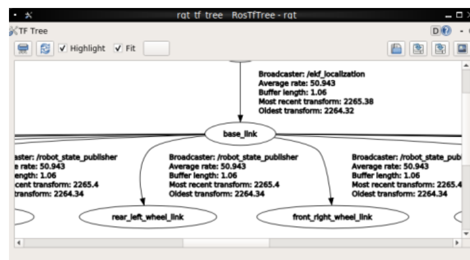
RQT Tools (some examples)



rqt_plot



rqt_bag (heavier than rosbag)



rqt_tf_tree

rqt_console

TF – Static publish

Static:

- Define on robot URDF to use robot_state_publisher
- If not possible: use static_transform_publisher node

```
<node pkg="tf" type="static_transform_publisher"
name="link1_broadcaster" args="1 0 0 0 0 0 1
link1_parent link1 100" />
```

- If you don't want to use static_transform_publisher you can do it in your own code (wiki): Python C++

TF – Dynamic publish

- Use robot URDF + robot_state_publisher + joint_states information
- Use a localization package like AMCL (adds a TF from the robot base to map)
- Write your own code if you really need it:

C++ broadcast Python broadcast

Add a frame (C++) Add a frame (Python)

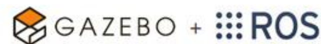
Simulation with Gazebo

TL; DR:
ROS interacts with Gazebo with the gazebo_ros package

rosrun gazebo_ros gazebo

or

rosrun gazebo_ros gzserver
+
rosrun gazebo_ros gzclient



Meta Package: **gazebo_ros_pkgs**

gazebo
Stand Alone Core
urdfdom

gazebo_msgs
Msg and Srv data structures for interacting with Gazebo from ROS.

gazebo_tests
Merged to gazebo_plugins
Contains a variety of unit tests for gazebo, tools and plugins.

gazebo_ros
Formally simulator_gazebo/gazebo

This package wraps gzserver and gzclient by using two Gazebo plugins that provide the necessary ROS interface for messages, services and dynamic reconfigure

ROS node name:
gazebo

Plugins:
gazebo_ros_api_plugin
gazebo_ros_paths_plugin

Usage:
rosrun gazebo_ros gazebo
rosrun gazebo_ros gzserver
rosrun gazebo_ros gzclient
rosrun gazebo_ros spawn_model
rosrun gazebo_ros perf
rosrun gazebo_ros debug

gazebo_plugins
Robot-independent Gazebo plugins.

Sensory
gazebo_ros_projector
gazebo_ros_p3d
gazebo_ros_imu
gazebo_ros_laser
gazebo_ros_f3d
gazebo_ros_camera_utils
gazebo_ros_depth_camera
gazebo_ros_openni_kinect
gazebo_ros_camera
gazebo_ros_bumper
gazebo_ros_block_laser
gazebo_ros_gpu_laser

Motory
gazebo_ros_joint_trajectory
gazebo_ros_diffdrive
gazebo_ros_force
gazebo_ros_template

Dynamic Reconfigure
vision_reconfigure
hokuyo_node
camera_synchronizer

gazebo_worlds
Merged to gazebo_ros
Contains a variety of unit tests for gazebo, tools and plugins.

wg
simple_erratic
simple_office
wg_collada_throttled - delete
wg_collada
grasp
empty_throttled
3stacks
elevator
simple_office_table
scan
empty
simple
balcony
camera
test_friction
simple_office2
empty_listener

gazebo_tools
Removed

gazebo_ros_api_plugin

Gazebo Subscribed Topics
~/set_link_state
~/set_model_state

Gazebo Published Parameters
/use_sim_time

Gazebo Published Topics
/clock
~/link_states
~/model_states

Gazebo Services
~/spawn_urdf_model
~/spawn_sdf_model
~/delete_model

State and properties getters
...

State and properties setters
...

Simulation control
~/pause_physics
~/unpause_physics
~/reset_simulation
~/reset_world

Force control
~/apply_body_wrench
~/apply_joint_effort
~/clear_joint_forces
~/clear_body_wrenches

gazebo_ros_paths_plugin
Provides ROS package paths to Gazebo

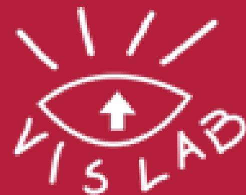
ROS packages | Gazebo Plugin | Depreciated from simulator_gazebo

Exploring a simulation launcher

(Streaming VM screen)

Issues and doubts

Discord server: https://discord.gg/beXDDpat



Thank you

javelino[at]isr.tecnico.ulisboa.pt
rodrigo.ventura[at]isr.tecnico.ulisboa.pt