Programming for Robotics
Introduction to ROS

Course 1

Dominic Jud, Martin Wermelinger, Marko Bjelonic, Péter Fankhauser
Prof. Dr. Marco Hutter
Overview

- **Course 1**
  - ROS architecture & philosophy
  - ROS master, nodes, and topics
  - Console commands
  - Catkin workspace and build system
  - Launch-files
  - Gazebo simulator

- **Course 2**
  - ROS package structure
  - Integration and programming with Eclipse
  - ROS C++ client library (roscpp)
  - ROS subscribers and publishers
  - ROS parameter server
  - RViz visualization

- **Course 3**
  - TF Transformation System
  - rqt User Interface
  - Robot models (URDF)
  - Simulation descriptions (SDF)

- **Course 4**
  - ROS services
  - ROS actions (actionlib)
  - ROS time
  - ROS bags

- **Course 5**
  - Case study

18.02.2019
Course Structure

**Course 1**
- Lecture 1
- Exercise 1 Intro.
- Exercise 1

**Course 2**
- Deadline for Ex. 1.
- Lecture 2
- Exercise 2

**Course 3**
- Deadline for Ex. 2.
- Lecture 3
- Exercise 3

**Course 4**
- Deadline for Ex. 3.
- Lecture 4
- Exercise 4

**Course 5**
- Deadline for Ex. 4.
- Multiple Choice Test
- Case Study
- Exercise 5 Intro.
- Exercise 5
- Deadline for Ex. 5.
Evaluation – Exercises

- Each exercise has several check questions
- Each exercise counts for 10% of the final grade (50 % in total)
- We encourage team work, but every student has to show the results on his own PC and is evaluated individually
- Exercises are checked by the teaching assistants when you are ready, but latest the following course day in the morning (08:15–08:45, except for exercise 5)
- Let the teaching assistant know once you are ready to present your results
- The lectures start at 08:45
Evaluation – Multiple Choice Test

- The test counts for 50 % of the final grade
- The multiple choice test takes place at the last course day:

01.03.2019 at 08:45, HG G1
Overview Course 1

- ROS architecture & philosophy
- ROS master, nodes, and topics
- Console commands
- Catkin workspace and build system
- Launch-files
- Gazebo simulator
What is ROS?

ROS = Robot Operating System

- Plumbing
  - Process management
  - Inter-process communication
  - Device drivers

- Tools
  - Simulation
  - Visualization
  - Graphical user interface
  - Data logging

- Capabilities
  - Control
  - Planning
  - Perception
  - Mapping
  - Manipulation

- Ecosystem
  - Package organization
  - Software distribution
  - Documentation
  - Tutorials

ros.org
History of ROS

- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory
- Since 2013 managed by OSRF
- Today used by many robots, universities and companies
- De facto standard for robot programming
ROS Philosophy

- **Peer to peer**
  Individual programs communicate over defined API (ROS messages, services, etc.).

- **Distributed**
  Programs can be run on multiple computers and communicate over the network.

- **Multi-lingual**
  ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).

- **Light-weight**
  Stand-alone libraries are wrapped around with a thin ROS layer.

- **Free and open-source**
  Most ROS software is open-source and free to use.
ROS Master

- Manages the communication between nodes (processes)
- Every node registers at startup with the master

Start a master with

> roscore

More info
http://wiki.ros.org/Master
### ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in *packages*

Run a node with

```
> rosrun package_name node_name
```

See active nodes with

```
> rosnodes list
```

Retrieve information about a node with

```
> rosnodes info node_name
```

More info

http://wiki.ros.org/rosnode
ROS Topics

- Nodes communicate over topics
  - Nodes can publish or subscribe to a topic
  - Typically, 1 publisher and $n$ subscribers
- Topic is a name for a stream of messages

List active topics with

```
> rostopic list
```

Subscribe and print the contents of a topic with

```
> rostopic echo /topic
```

Show information about a topic with

```
> rostopic info /topic
```

More info
http://wiki.ros.org/rostopic
ROS Messages

- Data structure defining the type of a topic
- Comprised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in *.msg files

See the type of a topic

> rostopic type /topic

Publish a message to a topic

> rostopic pub /topic type data

More info
http://wiki.ros.org/Messages
ROS Messages
Pose Stamped Example

**geometry_msgs/Point.msg**
- float64 x
- float64 y
- float64 z

**geometry_msgs/PoseStamped.msg**
- std_msgs/Header header
  - uint32 seq
  - time stamp
  - string frame_id
- geometry_msgs/Pose pose
  - geometry_msgs/Point position
    - float64 x
    - float64 y
    - float64 z
  - geometry_msgs/Quaternion orientation
    - float64 x
    - float64 y
    - float64 z
    - float64 w

**sensor_msgs/Image.msg**
- std_msgs/Header header
  - uint32 seq
  - time stamp
  - string frame_id
- uint32 height
- uint32 width
- string encoding
- uint8 is_bigendian
- uint32 step
- uint8[] data
Example
Console Tab Nr. 1 – Starting a roscore

Start a roscore with

```bash
> roscore
```

```
student@ubuntu:-/catkin ws$ roscore
... logging to /home/student/.ros/log/6c1852aa-e961-11e6-8543-000c297bd368/ros_launch-ubuntu-6696.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:34089/
ros_comm version 1.11.20

SUMMARY
=======

PARAMETERS
* /rosdistro: indigo
* /rosversion: 1.11.20

NODES
auto-starting new master
process[master]: started with pid [6708]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 6c1852aa-e961-11e6-8543-000c297bd368
process[rosout-1]: started with pid [6721]
started core service [/rosout]
```
Example
Console Tab Nr. 2 – Starting a *talker* node

Run a talker demo node with

```bash
> rosrun roscpp_tutorials talker
```

```
student@ubuntu:/catkin_ws$ rosrun roscpp_tutorials talker
[ INFO] [1486051788.424601519]: hello world 0
[ INFO] [1486051788.525227845]: hello world 1
[ INFO] [1486051788.624747612]: hello world 2
[ INFO] [1486051788.724826782]: hello world 3
[ INFO] [1486051788.825928577]: hello world 4
[ INFO] [1486051788.925379775]: hello world 5
[ INFO] [1486051789.024971132]: hello world 6
[ INFO] [1486051789.125450968]: hello world 7
[ INFO] [1486051789.225272747]: hello world 8
[ INFO] [1486051789.325389218]: hello world 9
```
Example
Console Tab Nr. 3 – Analyze talker node

See the list of active nodes

> rosnodes list

Show information about the talker node

> rosnodes info /talker

See information about the chatter topic

> rostopic info /chatter
Example
Console Tab Nr. 3 – Analyze *chatter* topic

Check the type of the *chatter* topic

```bash
> rostopic type /chatter
```

Show the message contents of the topic

```bash
> rostopic echo /chatter
```

Analyze the frequency

```bash
> rostopic hz /chatter
```

---

18.02.2019

---
Example
Console Tab Nr. 4 – Starting a *listener* node

Run a listener demo node with

```
> rosrun roscpp_tutorials listener
```

```
student@ubuntu:~/.catkin_ws$ rosrun roscpp_tutorials listener
INFO] [1486053802.204104598]: I heard: [hello world 19548]
INFO] [1486053802.304538827]: I heard: [hello world 19549]
INFO] [1486053802.403853395]: I heard: [hello world 19550]
INFO] [1486053802.504438133]: I heard: [hello world 19551]
INFO] [1486053802.604297608]: I heard: [hello world 19552]
```
Example
Console Tab Nr. 3 – Analyze

See the new *listener* node with

```
> rosnode list
```

Show the connection of the nodes over the chatter topic with

```
> rostopic info /chatter
```
Example

Console Tab Nr. 3 – Publish Message from Console

Close the *talker* node in console nr. 2 with Ctrl + C

Publish your own message with

```
> rostopic pub /chatter std_msgs/String "data: 'ETH Zurich ROS Course'"
```

Check the output of the *listener* in console nr. 4
ROS Workspace Environment

- Defines context for the current workspace
- Default workspace loaded with

```bash
> source /opt/ros/kinetic/setup.bash
```

Overlay your `catkin workspace` with

```bash
> cd ~/catkin_ws
> source devel/setup.bash
```

Check your workspace with

```bash
> echo $ROS_PACKAGE_PATH
```

More info

- [http://wiki.ros.org/catkin/workspaces](http://wiki.ros.org/catkin/workspaces)
catkin Build System

- *catkin* is the ROS build system to generate executables, libraries, and interfaces
- We suggest to use the *Catkin Command Line Tools*

  → Use `catkin build` instead of `catkin_make`

Navigate to your catkin workspace with

```
> cd ~/catkin_ws
```

Build a package with

```
> catkin build package_name
```

Whenever you build a **new** package, update your environment

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

More info

- [https://catkin-tools.readthedocs.io/](https://catkin-tools.readthedocs.io/)
catkin Build System

The catkin workspace contains the following spaces

**Work here**

- **src**

  The *source space* contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

**Don't touch**

- **build**

  The *build space* is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

- **devel**

  The *development (devel) space* is where built targets are placed (prior to being installed).

If necessary, clean the entire build and devel space with

```
catkin clean
```
The catkin workspace setup can be checked with

> catkin config

For example, to set the **CMake build type** to Release (or Debug etc.), use

> catkin build --cmake-args -DCMAKE_BUILD_TYPE=Release

More info

Already setup in the provided installation.
Example

Open a terminal and browse to your git folder

```bash
> cd ~/git
```

Clone the Git repository with

```bash
> git clone https://github.com/leggedrobotics/ros_best_practices.git
```

Symlink the new package to your catkin workspace

```bash
> ln -s ~/git/ros_best_practices/ ~/catkin_ws/src/
```

Note: You could also directly clone to your catkin workspace, but using a common git folder is convenient if you have multiple catkin workspaces.
Example

Go to your catkin workspace

> cd ~/catkin_ws

Build the package with

> catkin build ros_package_template

Re-source your workspace setup

> source devel/setup.bash

Launch the node with

> roslaunch ros_package_template ros_package_template.launch
**ROS Launch**

- *launch* is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as *.launch* files
- If not yet running, launch automatically starts a roscore

Browse to the folder and start a launch file with

```
> roslaunch file_name.launch
```

Start a launch file from a package with

```
> roslaunch package_name file_name.launch
```

More info

http://wiki.ros.org/roslaunch
ROS Launch
File Structure

**talker_listener.launch**

- `<launch>`: Root element of the launch file
- `<node>`: Each `<node>` tag specifies a node to be launched
- `<name>`: Name of the node (free to choose)
- `<pkg>`: Package containing the node
- `<type>`: Type of the node, there must be a corresponding executable with the same name
- `<output>`: Specifies where to output log messages (screen: console, log: log file)

Attention when copy & pasting code from the internet

Notice the syntax difference for self-closing tags: `<tag></tag>` and `<tag/>`

More info
http://wiki.ros.org/roslaunch/XML
http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects
ROS Launch Arguments

- Create re-usable launch files with `<arg>` tag, which works like a parameter (default optional)

  ```xml
  <arg name="arg_name" default="default_value"/>
  ```

- Use arguments in launch file with

  ```
  $(arg arg_name)
  ```

- When launching, arguments can be set with

  ```
  $($(arg arg_name))
  ```

```xml
<launch>
  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <include file="$($(find gazebo_ros) /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```
ROS Launch
Including Other Launch Files

- Include other launch files with <include> tag to organize large projects
  <include file="package_name"/>

- Find the system path to other packages with
  $(find package_name)

- Pass arguments to the included file
  <arg name="arg_name" value="value"/>

range_world.launch (simplified)

```xml
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>
  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true"/>
  </group>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

More info
http://wiki.ros.org/roslaunch/XML/include
Gazebo Simulator

- Simulate 3d rigid-body dynamics
- Simulate a variety of sensors including noise
- 3d visualization and user interaction
- Includes a database of many robots and environments (*Gazebo worlds*)
- Provides a ROS interface
- Extensible with plugins

Run Gazebo with

```bash
> rosrun gazebo_ros gazebo
```

More info

http://gazebosim.org/
http://gazebosim.org/tutorials
Further References

- **ROS Wiki**

- **Installation**

- **Tutorials**

- **Available packages**
  - [http://www.ros.org/browse/](http://www.ros.org/browse/)

- **ROS Cheat Sheet**
  - [https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index](https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index)

- **ROS Best Practices**
  - [https://github.com/leggedrobotics/ros_best_practices/wiki](https://github.com/leggedrobotics/ros_best_practices/wiki)

- **ROS Package Template**
Contact Information

ETH Zurich
Robotic Systems Lab
Prof. Dr. Marco Hutter
LEE H 303
Leonhardstrasse 21
8092 Zurich
Switzerland

http://www.rsl.ethz.ch

Lecturers
Dominic Jud (dominic.jud@mavt.ethz.ch)
Martin Wermelinger (martin.wermelinger@mavt.ethz.ch)
Marko Bjelonic (marko.bjelonic@mavt.ethz.ch)
Péter Fankhauser (pfankhauser@anybotics.com)

Course website: http://www.rsl.ethz.ch/education-students/lectures/ros.html