# Programming for Robotics
## Introduction to ROS

Course 1

Edo Jelavic, Tom Lankhorst, Prof. Dr. Marco Hutter

RSL
Robotic Systems Lab

# Overview

**Course 1**

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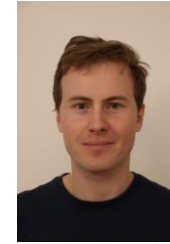- Gazebo simulator

*Edo Jelavic*

**Course 2**

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

*Tom Lankhorst*

**Course 3**

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

*Edo Jelavic*

**Course 4**

- ROS services
- ROS actions (actionlib)
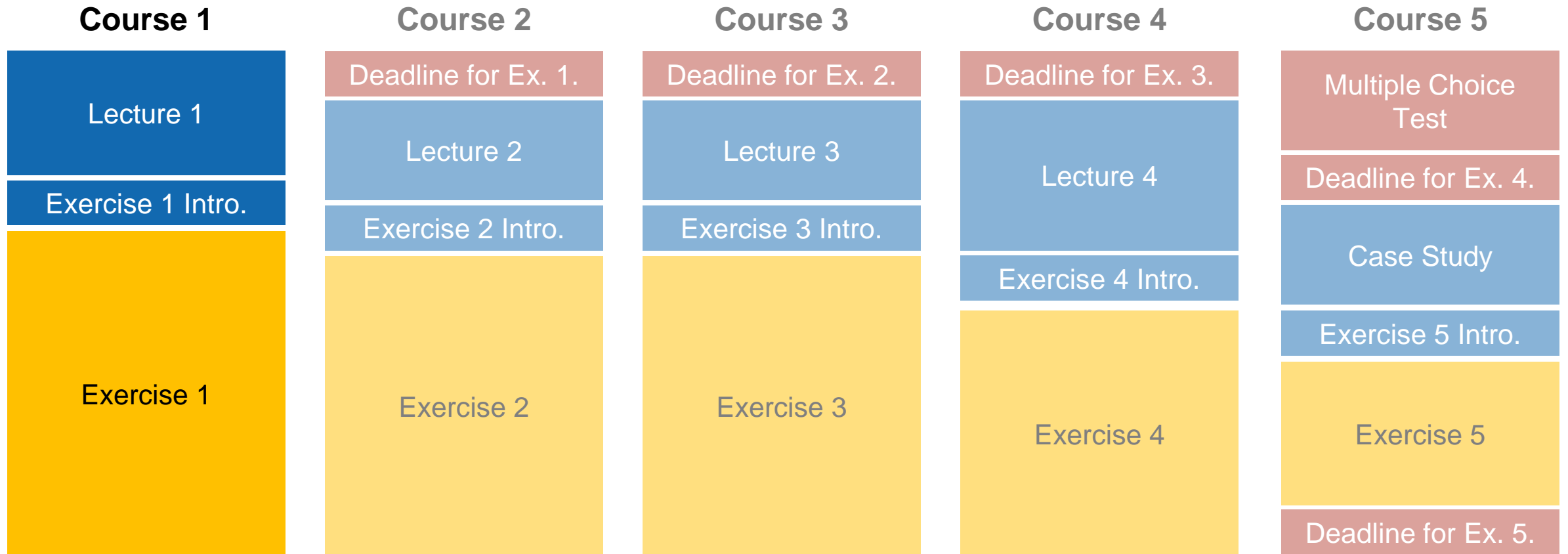- ROS time
- ROS bags
- Intro to ROS2

*Tom Lankhorst*

**Course 5**

- Case study

*Harmish & Max ANYbotics AG*

# Course Structure

| Course 1 | Course 2 | Course 3 | Course 4 | Course 5 |
|---|---|---|---|---|
| Lecture 1 | Deadline for Ex. 1. | Deadline for Ex. 2. | Deadline for Ex. 3. | Multiple Choice Test |
| Exercise 1 Intro. | Lecture 2 | Lecture 3 | Lecture 4 | Deadline for Ex. 4. |
| Exercise 1 | Exercise 2 Intro. | Exercise 3 Intro. | Exercise 4 Intro. | Case Study |
| | Exercise 2 | Exercise 3 | Exercise 4 | Exercise 5 Intro. |
| | | | | Exercise 5 |
| | | | | Deadline for Ex. 5. |

# Course Logistics

- We have three sessions
  - Grading (08:00 – 08:45)
  - Lectures (08:45-10:00)
  - Exercises (10:00 – 12:00)
- **The lectures start at 08:45**

# 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 their own PC and is evaluated individually
- There are 7 TAs, each has a group of students assigned
- The TA will be in the zoom call with you (10:00 - 12:00)

# Evaluation – Exercises

- Exercises are checked by the teaching assistants when you are ready, but latest the following course day in the grading sessions
- Grading sessions (08:00 – 08:45, except for exercise 5)
- The TA will take you in a separate breakout room
- After that you leave the grading session
- Let the teaching assistant know once you are ready to present your results

RSL
Robotic Systems Lab

# Evaluation – Multiple Choice Test

- The test counts for 50 % of the final grade

- Duration: 45min

- The test will be in person

- The multiple choice test takes place at the last course day:

# 05.03.2021 at 08:00

# 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

# 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.org

# 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

ROS Master

Start a master with

```
> roscore
```

RSL
Robotic Systems Lab

# ROS Nodes

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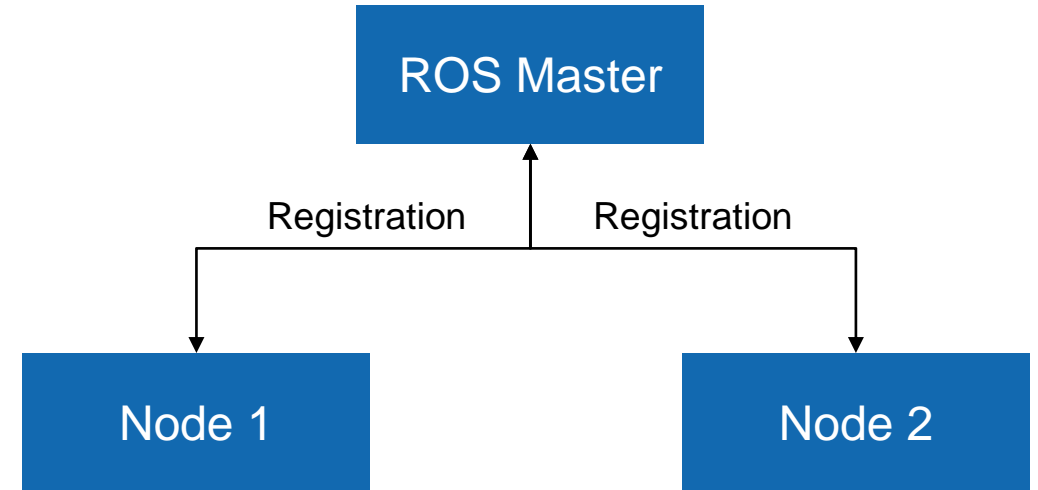- Organized in _packages_

Run a node with

```
> rosrun package_name node_name
```

See active nodes with

```
> rosnode list
```

Retrieve information about a node with

```
> rosnode info node_name
```



ROS Master

Registration      Registration

Node 1

Node 2

**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*
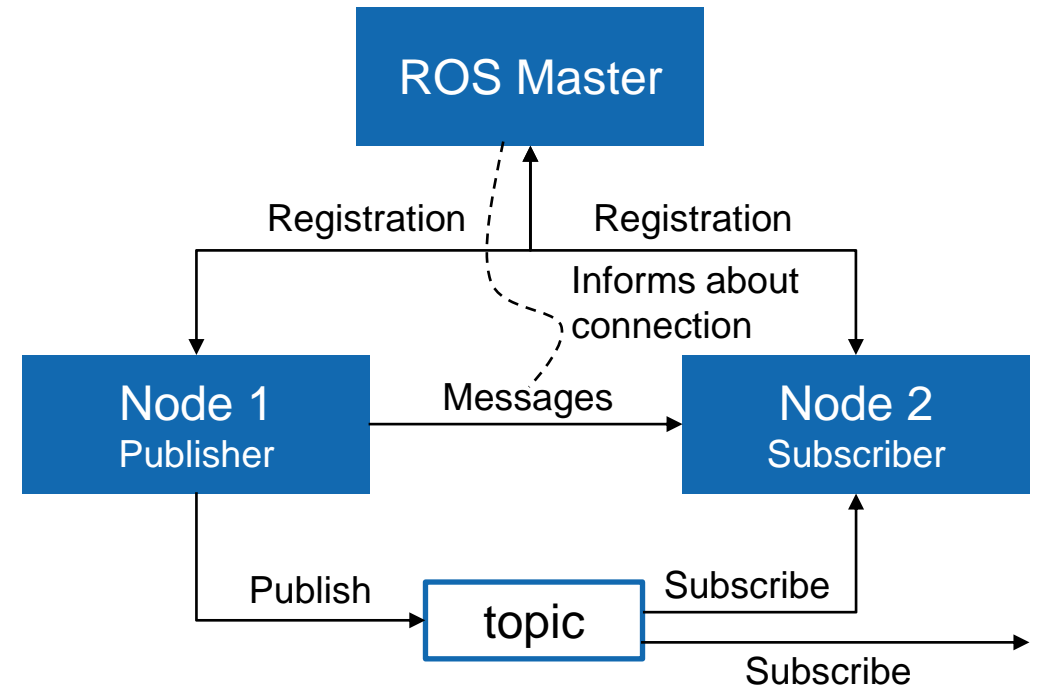
List active topics with

```
> 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**
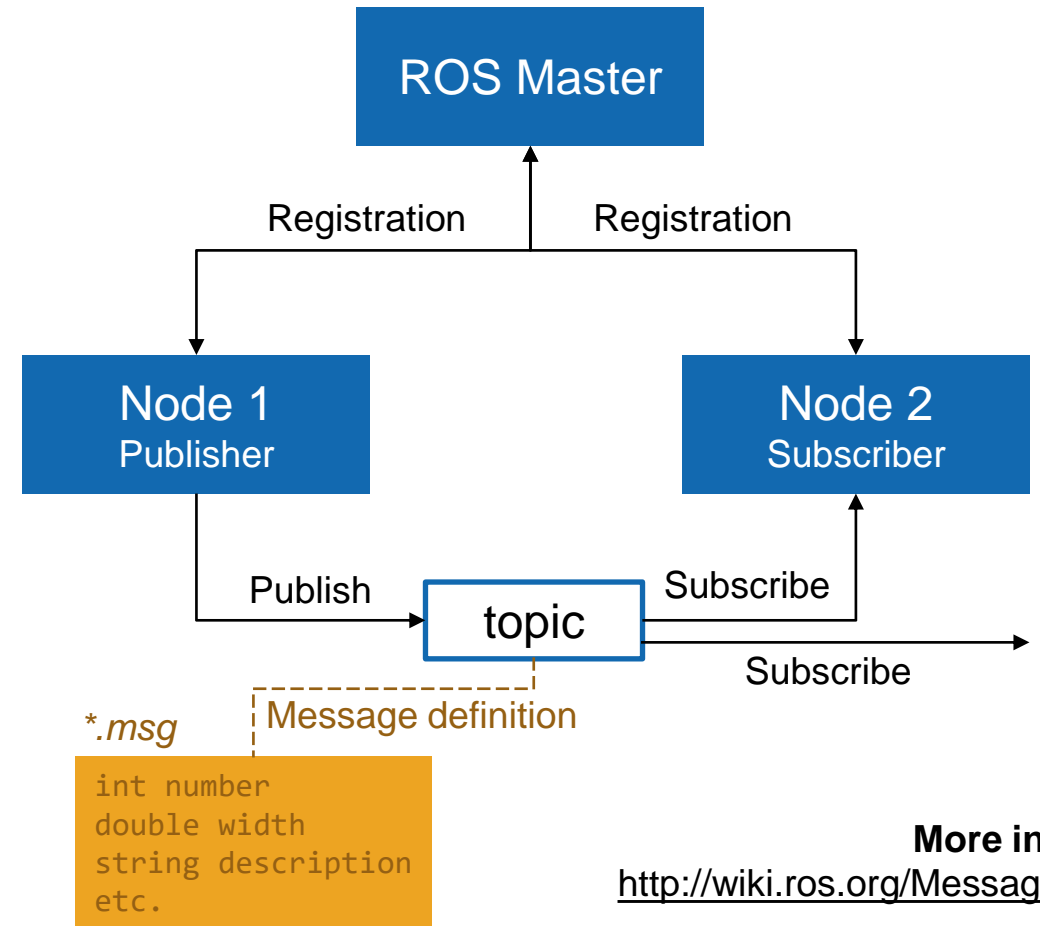http://wiki.ros.org/rostopic

# 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
```

ROS Master

Registration          Registration

Node 1
Publisher

Node 2
Subscriber

Publish          topic          Subscribe

Subscribe

*.msg          Message definition

```
int number
double width
string description
etc.
```

**More info**
http://wiki.ros.org/Messages

# ROS Messages
## Pose Stamped Example

*geometry_msgs/Point.msg*

```
float64 x
float64 y
float64 z
```
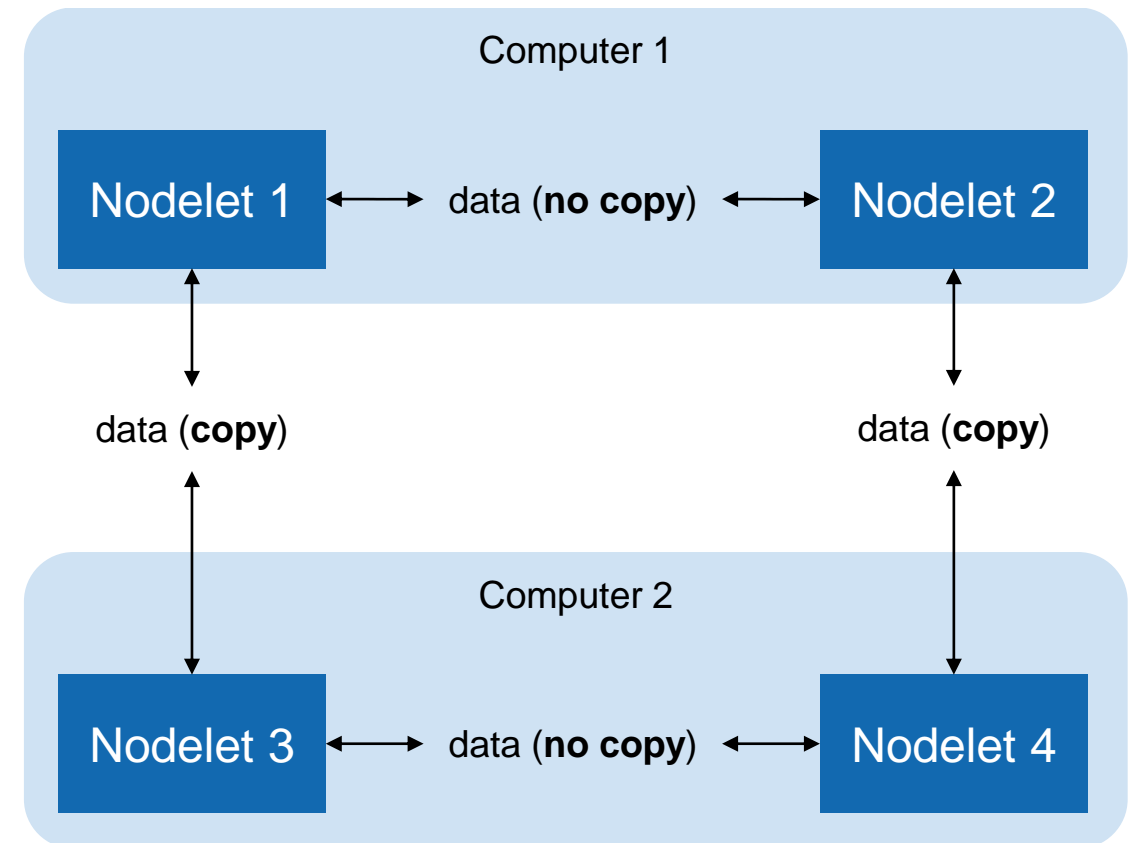
*sensor_msgs/Image.msg*

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

*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
```

# ROS Nodelets

- Same concept as ROS nodes
- Reduce communication overhead when running on same machine
- Try to use ROS nodes first
- Nodeletes are more complicated to implement

Computer 1

| Nodelet 1 | ← data (**no copy**) → | Nodelet 2 |

data (**copy**)                        data (**copy**)

Computer 2

| Nodelet 3 | ← data (**no copy**) → | Nodelet 4 |

**More info**
http://wiki.ros.org/nodelet

Robotic Systems Lab

Edo Jelavic  |  22.02.2021  |  17

# Example
## Console Tab Nr. 1 – Starting a *roscore*

Start a roscore with

```
> roscore
```

# Example
## Console Tab Nr. 2 – Starting a *talker* node

Run a talker demo node with

```
> rosrun roscpp_tutorials talker
```

```
student@ubuntu:~/catkin_ws$ rosrun roscpp_tutorials talker
[ INFO] [1486051708.424661519]: hello world 0
[ INFO] [1486051708.525227845]: hello world 1
[ INFO] [1486051708.624747612]: hello world 2
[ INFO] [1486051708.724826782]: hello world 3
[ INFO] [1486051708.825928577]: hello world 4
[ INFO] [1486051708.925379775]: hello world 5
[ INFO] [1486051709.024971132]: hello world 6
[ INFO] [1486051709.125450960]: hello world 7
[ INFO] [1486051709.225272747]: hello world 8
[ INFO] [1486051709.325389210]: hello world 9
```

# Example
## Console Tab Nr. 3 – Analyze *talker* node

See the list of active nodes

```
> rosnode list
```

```
student@ubuntu:~/catkin_ws$ rosnode list
/rosout
/talker
```

Show information about the *talker* node

```
> rosnode info /talker
```

```
student@ubuntu:~/catkin_ws$ rosnode info /talker
--------------------------------------------------
--
Node [/talker]
Publications:
 * /chatter [std_msgs/String]
 * /rosout [rosgraph_msgs/Log]

Subscriptions: None

Services:
 * /talker/get_loggers
 * /talker/set_logger_level
```

See information about the *chatter* topic

```
> rostopic info /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic info /chatter
Type: std_msgs/String

Publishers:
 * /talker (http://ubuntu:39173/)

Subscribers: None
```

# Example
## Console Tab Nr. 3 – Analyze *chatter* topic

Check the type of the *chatter* topic

```
> rostopic type /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic type /chatter
std_msgs/String
```

Show the message contents of the topic

```
> rostopic echo /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic echo /chatter
data: hello world 11874
---
data: hello world 11875
---
data: hello world 11876
```

Analyze the frequency

```
> rostopic hz /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic hz /chatter
subscribed to [/chatter]
average rate: 9.991
        min: 0.099s max: 0.101s std dev: 0.00076s window: 10
average rate: 9.996
        min: 0.099s max: 0.101s std dev: 0.00069s window: 20
```

# 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
```

```
student@ubuntu:~/catkin_ws$ rosnode list
/listener     ◄
/rosout
/talker
```

```
student@ubuntu:~/catkin_ws$ rostopic info /chatter
Type: std_msgs/String

Publishers:
 * /talker (http://ubuntu:39173/)     ◄

Subscribers:
 * /listener (http://ubuntu:34664/)     ◄
```

# 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'"
```

```
student@ubuntu:~/catkin_ws$ rostopic pub /chatter std_msgs/String "data: 'ETH
Zurich ROS Course'"
publishing and latching message. Press ctrl-C to terminate
```

Check the output of the *listener* in console nr. 4

```
[ INFO] [1486054667.604322265]: I heard: [hello world 28202]
[ INFO] [1486054667.704264199]: I heard: [hello world 28203]
[ INFO] [1486054667.804389058]: I heard: [hello world 28204]
[ INFO] [1486054707.646404558]: I heard: [ETH Zurich ROS Course]
```

# ROS Workspace Environment

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

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

Overlay your *catkin workspace* with

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

Check your workspace with

```
> echo $ROS_PACKAGE_PATH
```

This is already setup in the provided installation.

See setup with

```
> cat ~/.bashrc
```

**More info**
http://wiki.ros.org/kinetic/Installation/Ubuntu
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
```

The catkin command line tools are pre-installed in the provided installation.

**More info**
http://wiki.ros.org/catkin/Tutorials
https://catkin-tools.readthedocs.io/

# catkin Build System

The catkin workspace contains the following spaces

<div style="text-align:center">

**Work here**                    **Don't touch**                    **Don't touch**

</div>

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.

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.

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
```

**More info**
http://wiki.ros.org/catkin/workspaces

# catkin Build System

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**
http://catkin-tools.readthedocs.io/en/latest/verbs/catkin_config.html
http://catkin-tools.readthedocs.io/en/latest/cheat_sheet.html

```
student@ubuntu:~/catkin_ws$ catkin config

Profile:                    default
Extending:          [env] /opt/ros/indigo:/home/student/catkin_ws/devel
Workspace:                 /home/student/catkin_ws

Source Space:      [exists] /home/student/catkin_ws/src
Log Space:         [exists] /home/student/catkin_ws/logs
Build Space:       [exists] /home/student/catkin_ws/build
Devel Space:       [exists] /home/student/catkin_ws/devel
Install Space:     [unused] /home/student/catkin_ws/install
DESTDIR:           [unused] None

Devel Space Layout:        linked
Install Space Layout:      None

Additional CMake Args:        -GEclipse CDT4 - Unix Makefiles -DCMAKE_CXX_COMP
ILER_ARG1=-std=c++11 -DCMAKE_BUILD_TYPE=Release
Additional Make Args:         None
Additional catkin Make Args:  None
Internal Make Job Server:     True
Cache Job Environments:       False

Whitelisted Packages:         None
Blacklisted Packages:         None

Workspace configuration appears valid.
```

Already setup in the provided installation.

# Example

Open a terminal and browse to your `git` folder

```
> cd ~/git
```

Clone the Git repository with

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

Symlink the new package to your catkin workspace

```
> 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.*

*https://github.com/leggedrobotics/ros_best_practices*
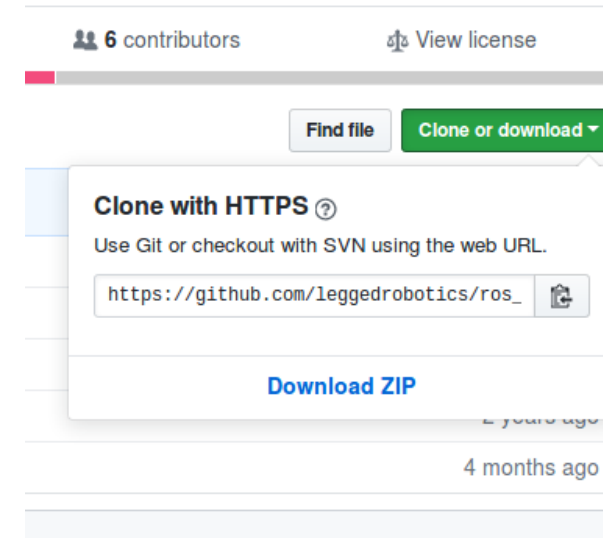
# Example

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

Example console output for
`roslaunch roscpp_tutorials talker_listener.launch`

```
student@ubuntu:~/catkin_ws$ roslaunch roscpp_tutorials talker_listener.launch
... logging to /home/student/.ros/log/794321aa-e950-11e6-95db-000c297bd368/rosl
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:37592/

SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES
  /
    listener (roscpp_tutorials/listener)
    talker (roscpp_tutorials/talker)

auto-starting new master
process[master]: started with pid [5772]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 794321aa-e950-11e6-95db-000c297bd368
process[rosout-1]: started with pid [5785]
started core service [/rosout]
process[listener-2]: started with pid [5788]
process[talker-3]: started with pid [5795]
[ INFO] [1486044252.537801350]: hello world 0
[ INFO] [1486044252.638886504]: hello world 1
[ INFO] [1486044252.738279674]: hello world 2
[ INFO] [1486044252.838357245]: hello world 3
```

# ROS Launch
## File Structure

**!** Attention when copy & pasting code from the internet

*talker_listener.launch*

```
<launch>
    <node name="listener" pkg="roscpp_tutorials" type="listener" output="screen"/>
    <node name="talker" pkg="roscpp_tutorials" type="talker" output="screen"/>
</launch>
```

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

- **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)

**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)

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

- Use arguments in launch file with

  `$(arg arg_name)`

- When launching, arguments can be set with

  `> roslaunch launch_file.launch arg_name:=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/arg

**RSL** Robotic Systems Lab

# ROS Launch
## Including Other Launch Files

*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>
```

- 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"/>
  ```

**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

```
> rosrun gazebo_ros gazebo
```

Object tree          Toolbar (to navigate and new objects)



Properties          Start and pause simulation

**More info**
http://gazebosim.org/
http://gazebosim.org/tutorials
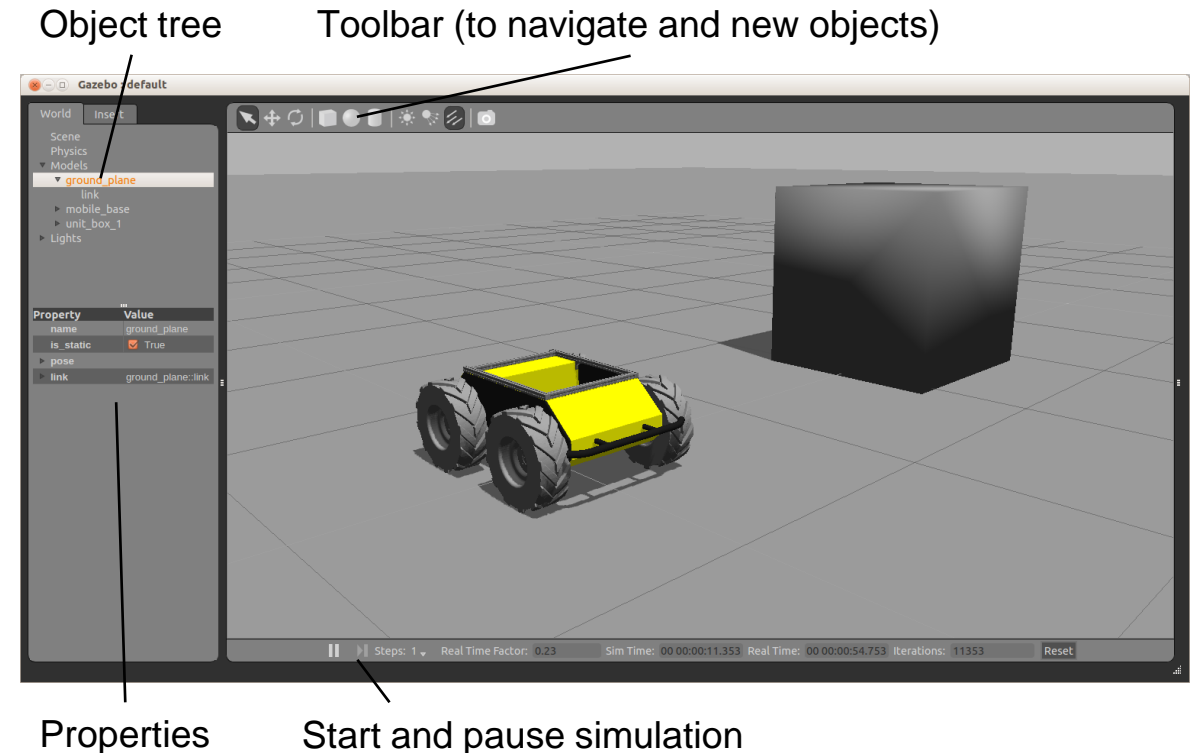
# Further References

- **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/Contents/Resources/Documents/index
- **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

# ETH Robotics Summer School

- A full week (July 5-9) with lectures, tutorials and hands-on focus, ending with a robotic challenge.

- Students are invited to attend the ETH Robotic Symposium (tentative date: July 2).

- Takes place in Wangen an der Aare

- Application window Feb. 15 to Mar. 26.

  - Acceptance notification April 15.



**More info:**
https://robotics-summerschool.ethz.ch/

RSL
Robotic Systems Lab

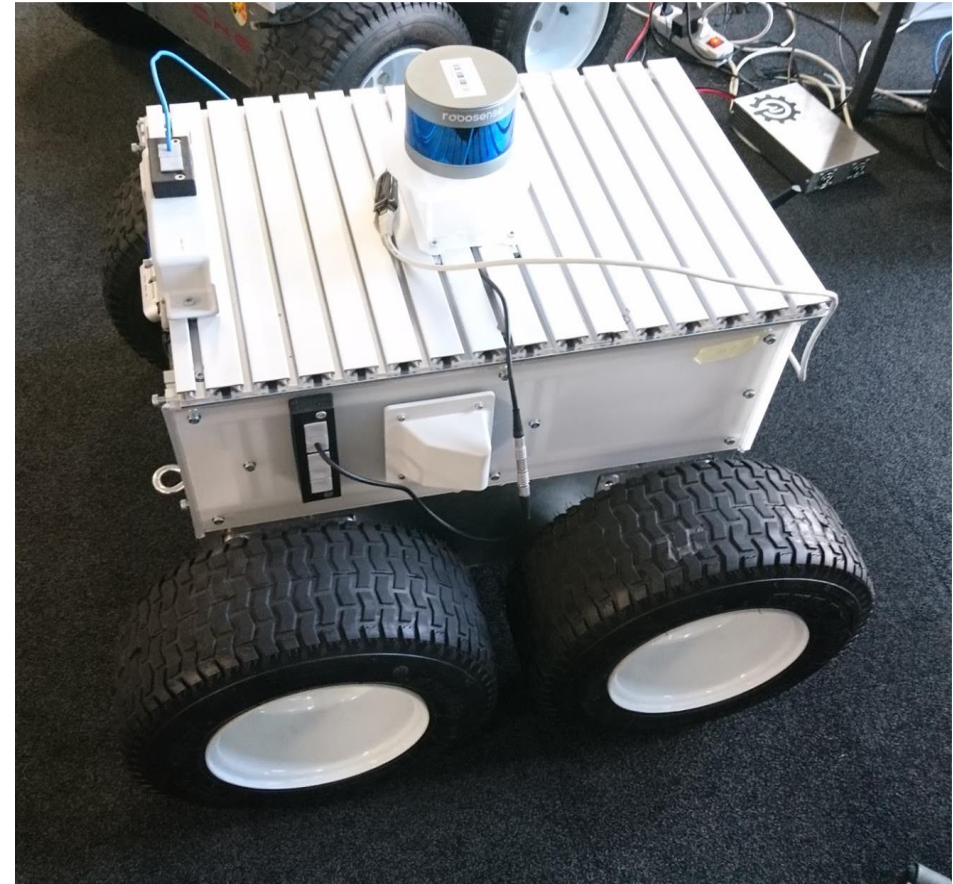# RobotX - Robotic Platforms

**4 all terrain robots "Super Mega Bot" (SMB)**

- Base (skid steer drive, driving battery, E-Stop, remote control)

- Payload
  - Intel NUC (option for GPU and 2nd NUC)
  - Connectivity (WiFi, 4G Modem)
  - Sensors:
    - RGB-D - RealSense D435
    - 3D LiDAR - RoboSense RS-16
    - Tracking Cam - RealSense T265
    - VIO-System - VersaVis / RGB / IMU

RSL
Robotic Systems Lab

# Tentative Program



| | Su 04 | Mo 05 | Tu 06 | We 07 | Th 08 | Fr 09 |
|---|---|---|---|---|---|---|
| 9:00 | | Intro to RSS | **Localization & Mapping** | **Visual Recognition** | Robot time | Robot time |
| 10:00 | | Intro to Robots | Tutorial | Tutorial | Robot time | Robot time |
| 11:00 | | Intro to Robots | Tutorial | Tutorial | Robot time | Robot time |
| 12:00 | Transfer to WadA | Lunch | Robot time | Robot time | Lunch | Lunch |
| 13:00 | Transfer to WadA | **State Estimation** | Lunch | Lunch | Robot time | Challenge |
| 14:00 | Transfer to WadA | Tutorial | **Path Planning** | Robot time | Robot time | Challenge |
| 15:00 | Transfer to WadA | Tutorial | Tutorial | Robot time | Robot time | Challenge |
| | Transfer to WadA | Break | Tutorial | Robot time | Break | Challenge |
| 16:00 | Social Activity | **Trajectory Optimization** | Break | Social Activity | Robot time | Awards |
| 17:00 | Social Activity | Tutorial | Robot time | Social Activity | Robot time | Transfer to ZH |
| 18:00 | Social Activity | Tutorial | Robot time | Social Activity | Robot time | Transfer to ZH |
| 19:00 | Social Activity | Robot time | Robot time | Social Activity | Robot time | Transfer to ZH |

# Preview

# 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**
Edo Jelavic (edo.jelavic@mavt.ethz.ch)
Tom Lankhorst (tom.lankhorst@mavt.ethz.ch)

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