ETH zürich

Programming for Robotics
Introduction to ROS

Course 4

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

ROS

RSL Robotic Systems Lab

Course Structure

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

Overview Course 4

- ROS services
- ROS actions (actionlib)
- ROS time
- ROS bags
- Debugging strategies

ROS Services

- Request/response communication between nodes is realized with *services*
  - The *service server* advertises the service
  - The *service client* accesses this service
- Similar in structure to messages, services are defined in **.srv* files

List available services with

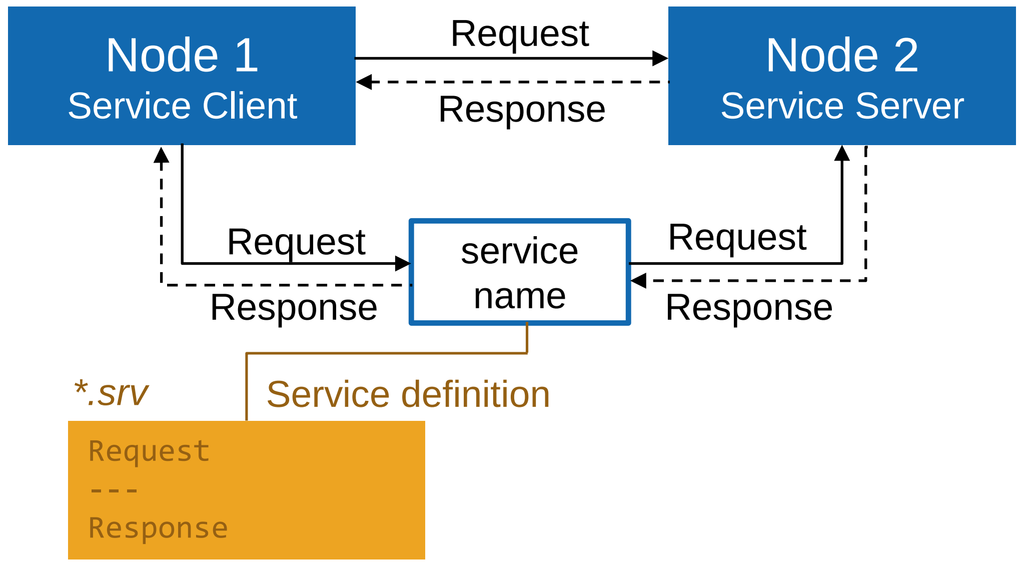
```
> rosservice list
```

Show the type of a service

```
> rosservice type /service_name
```

Call a service with the request contents

```
> rosservice call /service_name args
```



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

ROS Services
Examples

std_srvs/Trigger.srv

```
---
bool success
string message
```

Request

Response

nav_msgs/GetPlan.srv

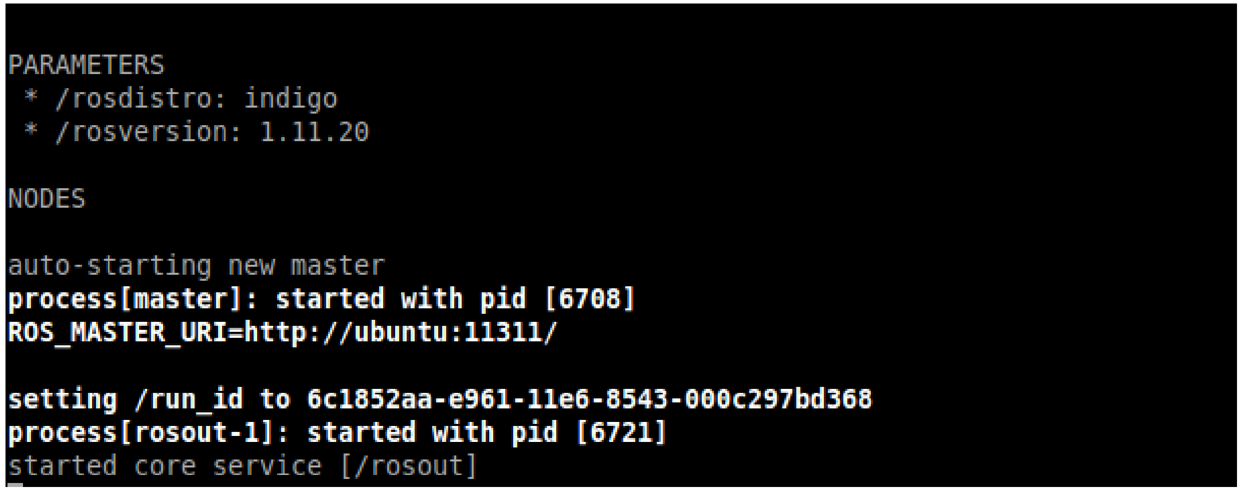
```
geometry_msgs/PoseStamped start
geometry_msgs/PoseStamped goal
float32 tolerance
---
nav_msgs/Path plan
```

Robotic Systems Lab

ROS Service Example
Starting a *roscore* and a *add_two_ints_server* node

In console nr. 1:
Start a roscore with

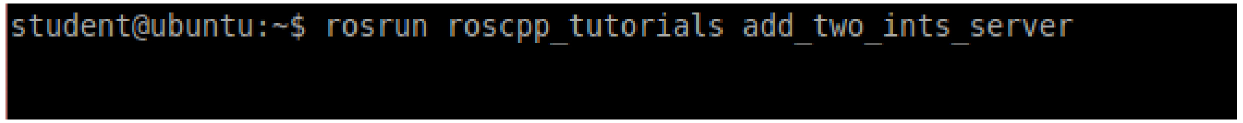
```
> roscore
```



```
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]
```

In console nr. 2:
Run a service demo node with

```
> rosrun roscpp_tutorials add_two_ints_server
```

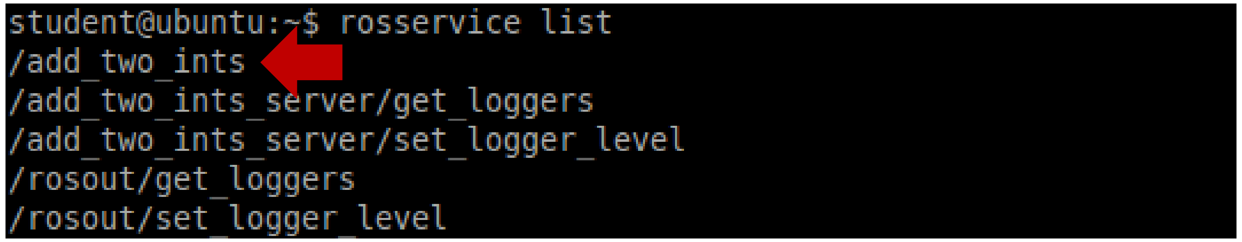


```
student@ubuntu:~$ rosrun roscpp_tutorials add_two_ints_server
```

ROS Service Example
Console Nr. 3 – Analyze and call service

See the available services with

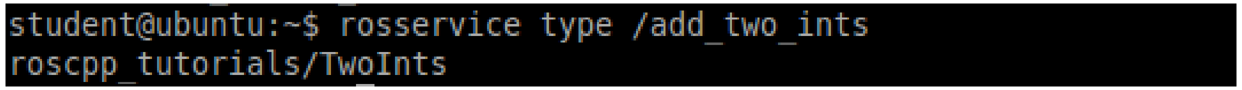
```
> rosservice list
```



```
student@ubuntu:~$ rosservice list
/add_two_ints
/add_two_ints_server/get_loggers
/add_two_ints_server/set_logger_level
/rosout/get_loggers
/rosout/set_logger_level
```

See the type of the service with

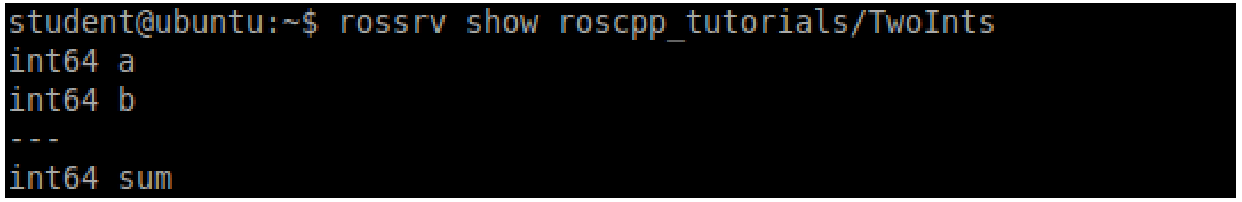
```
> rosservice type /add_two_ints
```



```
student@ubuntu:~$ rosservice type /add_two_ints
roscpp_tutorials/TwoInts
```

Show the service definition with

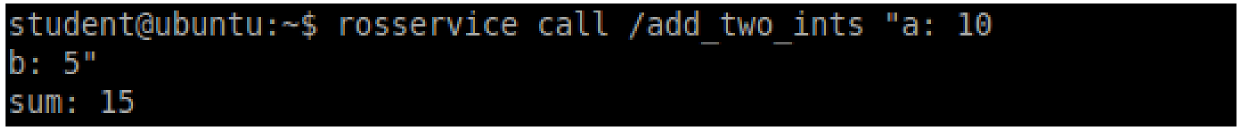
```
> rossrv show roscpp_tutorials/TwoInts
```



```
student@ubuntu:~$ rossrv show roscpp_tutorials/TwoInts
int64 a
int64 b
---
int64 sum
```

Call the service (use `Tab` for auto-complete)

```
> rosservice call /add_two_ints "a: 10
  b: 5"
```



```
student@ubuntu:~$ rosservice call /add_two_ints "a: 10
b: 5"
sum: 15
```

ETH zürich

ROS C++ Client Library (*roscpp*)

Service Server

- Create a service server with

```
ros::ServiceServer service =
 nodeHandle.advertiseService(service_name,
                      callback_function);
```

- When a service request is received, the callback function is called with the request as argument
- Fill in the response to the response argument
- Return to function with true to indicate that it has been executed properly

More info
http://wiki.ros.org/roscpp/Overview/Services

add_two_ints_server.cpp

```
#include <ros/ros.h>
#include <roscpp_tutorials/TwoInts.h>

bool add(roscpp_tutorials::TwoInts::Request &request,
         roscpp_tutorials::TwoInts::Response &response)
{
  response.sum = request.a + request.b;
  ROS_INFO("request: x=%ld, y=%ld", (long int)request.a,
                                    (long int)request.b);
  ROS_INFO("  sending back response: [%ld]",
           (long int)response.sum);
  return true;
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_server");
  ros::NodeHandle nh;
  ros::ServiceServer service =
          nh.advertiseService("add_two_ints", add);
  ros::spin();
  return 0;
}
```

Robotic Systems Lab

ROS C++ Client Library (*roscpp*)
Service Client

- Create a service client with

```
ros::ServiceClient client =
    nodeHandle.serviceClient<service_type>
                              (service_name);
```

- Create service request contents `service.request`
- Call service with

```
client.call(service);
```

- Response is stored in `service.response`

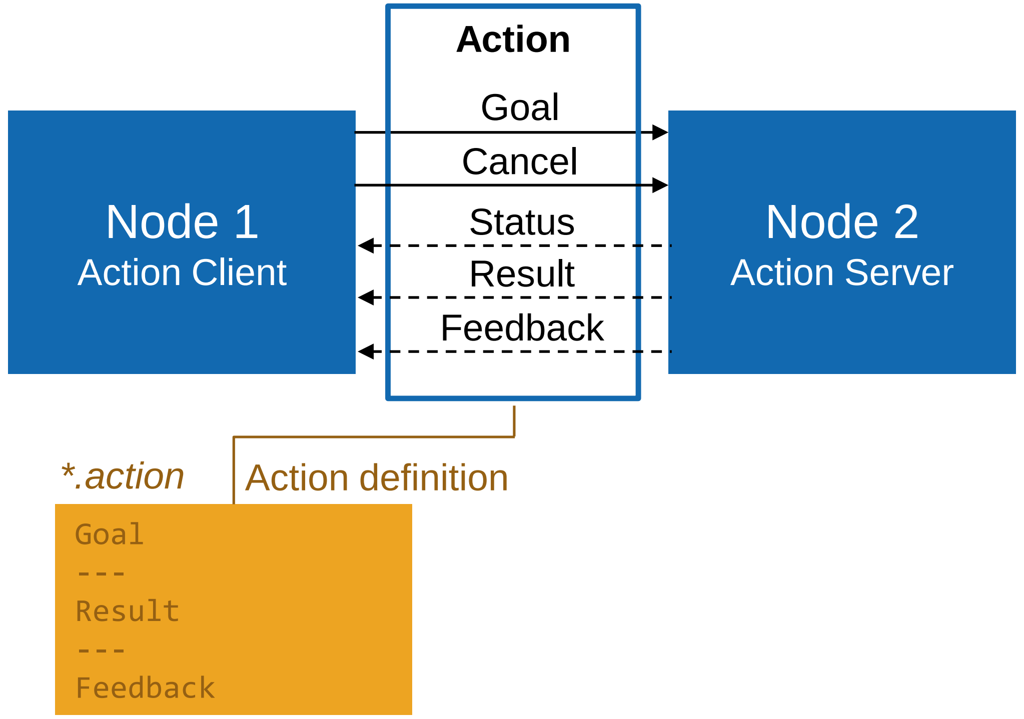
add_two_ints_client.cpp

```
#include <ros/ros.h>
#include <roscpp_tutorials/TwoInts.h>
#include <cstdlib>

int main(int argc, char **argv) {
  ros::init(argc, argv, "add_two_ints_client");
  if (argc != 3) {
    ROS_INFO("usage: add_two_ints_client X Y");
    return 1;
  }

  ros::NodeHandle nh;
  ros::ServiceClient client =
  nh.serviceClient<roscpp_tutorials::TwoInts>("add_two_ints");
  roscpp_tutorials::TwoInts service;
  service.request.a = atoi(argv[1]);
  service.request.b = atoi(argv[2]);
  if (client.call(service)) {
    ROS_INFO("Sum: %ld", (long int)service.response.sum);
  } else {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }
  return 0;
}
```

More info
http://wiki.ros.org/roscpp/Overview/Services

ROS Actions (actionlib)

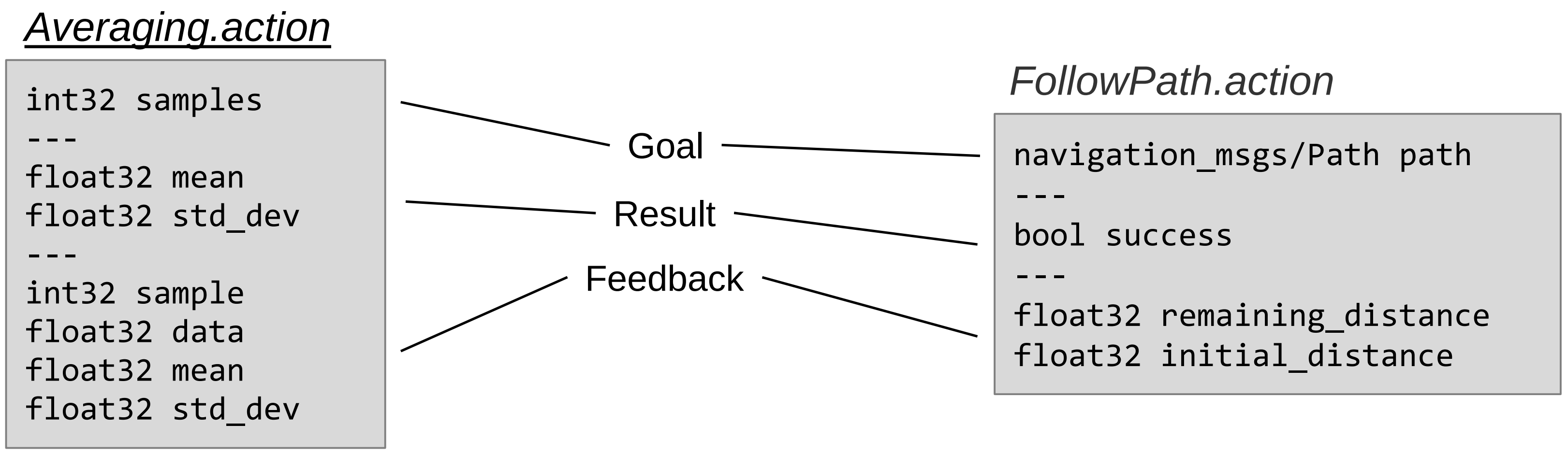
- Similar to service calls, but provide possibility to
  - Cancel the task (preempt)
  - Receive feedback on the progress
- Best way to implement interfaces to time-extended, goal-oriented behaviors
- Similar in structure to services, action are defined in **.action* files
- Internally, actions are implemented with a set of topics



```
Goal
---
Result
---
Feedback
```

More info
http://wiki.ros.org/actionlib
http://wiki.ros.org/actionlib/DetailedDescription

ROS Actions (actionlib)



Averaging.action

```
int32 samples
---
float32 mean
float32 std_dev
---
int32 sample
float32 data
float32 mean
float32 std_dev
```

FollowPath.action

```
navigation_msgs/Path path
---
bool success
---
float32 remaining_distance
float32 initial_distance
```

ROS Parameters, Dynamic Reconfigure, Topics, Services, and Actions Comparison

	Parameters	Dynamic Reconfigure	Topics	Services	Actions
Description	Global constant parameters	Local, changeable parameters	Continuous data streams	Blocking call for processing a request	Non-blocking, preemptable goal oriented tasks
Application	Constant settings	Tuning parameters	One-way continuous data flow	Short triggers or calculations	Task executions and robot actions
Examples	Topic names, camera settings, calibration data, robot setup	Controller parameters	Sensor data, robot state	Trigger change, request state, compute quantity	Navigation, grasping, motion execution

ROS Time

- Normally, ROS uses the PC's system clock as time source (*wall time*)
- For simulations or playback of logged data, it is convenient to work with a simulated time (pause, slow-down etc.)
- To work with a simulated clock:
  - Set the `/use_sim_time` parameter

    ```
    > rosparam set use_sim_time true
    ```

  - Publish the time on the topic `/clock` from
    - Gazebo (enabled by default)
    - ROS bag (use option `--clock`)

- To take advantage of the simulated time, you should always use the ROS Time APIs:
  - **`ros::Time`**

    ```
    ros::Time begin = ros::Time::now();
    double secs = begin.toSec();
    ```

  - **`ros::Duration`**

    ```
    ros::Duration duration(0.5); // 0.5s
    ```

  - **`ros::Rate`**

    ```
    ros::Rate rate(10); // 10Hz
    ```

- If wall time is required, use `ros::WallTime`, `ros::WallDuration`, and `ros::WallRate`

More info
http://wiki.ros.org/Clock
http://wiki.ros.org/roscpp/Overview/Time

ROS Bags

- A *bag* is a format for storing message data
- Binary format with file extension *.bag
- Suited for logging and recording datasets for later visualization and analysis

Record all topics in a bag

```
> rosbag record --all
```

Record given topics

```
> rosbag record topic_1 topic_2 topic_3
```

Stop recording with `Ctrl` + `C`
Bags are saved with start date and time as file name in the current folder (e.g. `2019-02-07-01-27-13.bag`)

Show information about a bag

```
> rosbag info bag_name.bag
```

Read a bag and publish its contents

```
> rosbag play bag_name.bag
```

Playback options can be defined e.g.

```
> rosbag play --rate=0.5 bag_name.bag
```

`--rate=`*`factor`*	Publish rate factor
`--clock`	Publish the clock time (set param `use_sim_time` to `true`)
`--loop`	Loop playback
	etc.

More info
http://wiki.ros.org/rosbag/Commandline

Debugging Strategies

Debug with the tools you have learned

- Compile and run code often to catch bugs early
- Understand compilation and runtime error messages
- Use analysis tools to check data flow (`rosnode info`, `rostopic echo`, `roswtf`, `rqt_graph` etc.)
- Visualize and plot data (RViz, RQT Multiplot etc.)
- Divide program into smaller steps and check intermediate results (`ROS_INFO`, `ROS_DEBUG` etc.)
- Make your code robust with argument and return value checks and catch exceptions
- If things don't make sense, clean your workspace

```
> catkin clean --all
```

Learn new tools

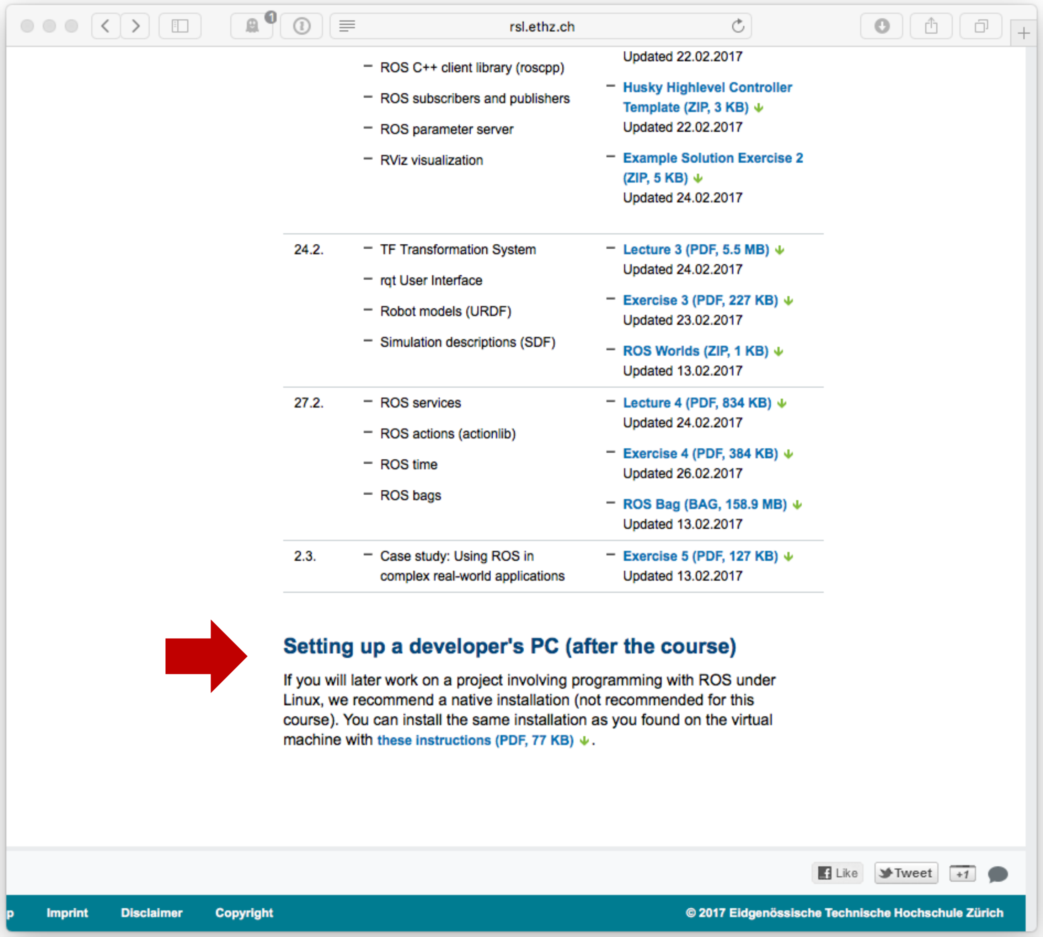
- Build in *debug* mode and use GDB or Valgrind

```
> catkin config --cmake-args
                -DCMAKE_BUILD_TYPE=Debug
```

- Use Eclipse breakpoints
- Maintain code with unit tests and integration tests

More info
http://wiki.ros.org/UnitTesting
http://wiki.ros.org/gtest
http://wiki.ros.org/rostest
http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20Nodes%20in%20Valgrind%20or%20GDB

Setting Up up a Developer’s PC



rsl.ethz.ch

	– ROS C++ client library (roscpp) – ROS subscribers and publishers – ROS parameter server – RViz visualization	Updated 22.02.2017 – Husky Highlevel Controller Template (ZIP, 3 KB) Updated 22.02.2017 – Example Solution Exercise 2 (ZIP, 5 KB) Updated 24.02.2017
24.2.	– TF Transformation System – rqt User Interface – Robot models (URDF) – Simulation descriptions (SDF)	– Lecture 3 (PDF, 5.5 MB) Updated 24.02.2017 – Exercise 3 (PDF, 227 KB) Updated 23.02.2017 – ROS Worlds (ZIP, 1 KB) Updated 13.02.2017
27.2.	– ROS services – ROS actions (actionlib) – ROS time – ROS bags	– Lecture 4 (PDF, 834 KB) Updated 24.02.2017 – Exercise 4 (PDF, 384 KB) Updated 26.02.2017 – ROS Bag (BAG, 158.9 MB) Updated 13.02.2017
2.3.	– Case study: Using ROS in complex real-world applications	– Exercise 5 (PDF, 127 KB) Updated 13.02.2017

Setting up a developer's PC (after the course)

If you will later work on a project involving programming with ROS under Linux, we recommend a native installation (not recommended for this course). You can install the same installation as you found on the virtual machine with these instructions (PDF, 77 KB).

Like Tweet +1

Imprint Disclaimer Copyright © 2017 Eidgenössische Technische Hochschule Zürich

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

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
Martin Wermelinger (martin.wermelinger@mavt.ethz.ch)
Dominic Jud (dominic.jud@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