



Programming for Robotics

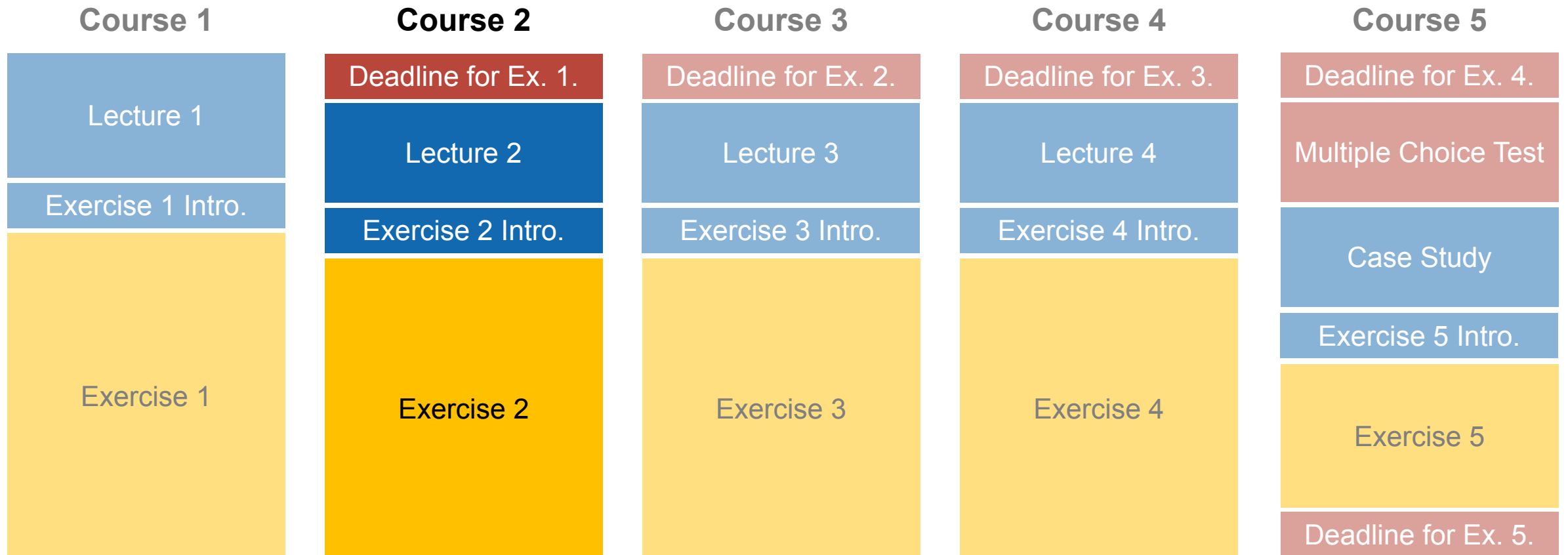
Introduction to ROS

Course 2

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



Course Structure



Overview Course 2

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

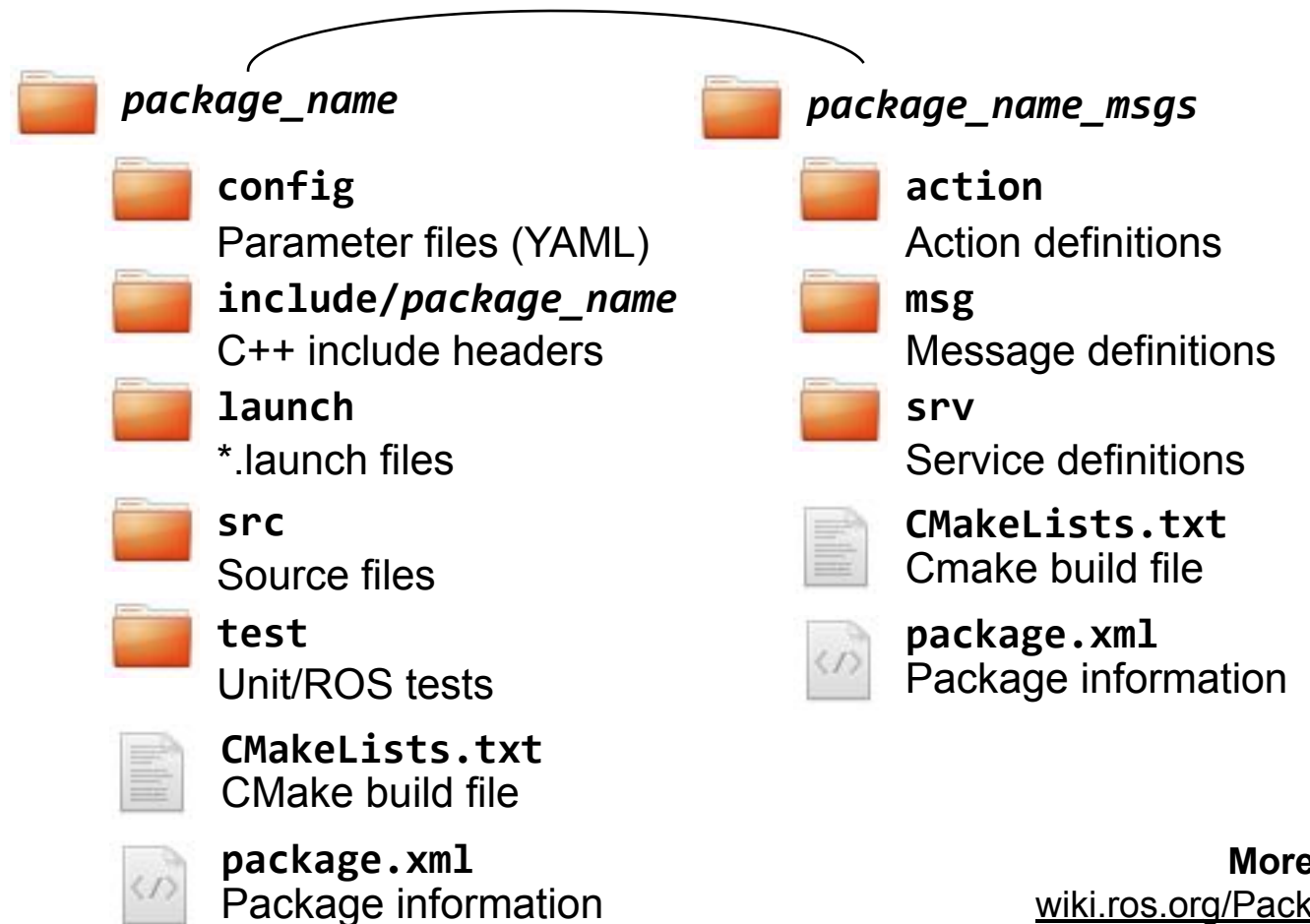
ROS Packages

- 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*

To create a new package, use

```
> catkin_create_pkg package_name  
{dependencies}
```

Separate message definition
packages from other packages!



More info
wiki.ros.org/Packages

ROS Packages

package.xml

- The package.xml file defines the properties of the package
 - Package name
 - Version number
 - Authors
 - **Dependencies on other packages**
 - ...

More info

wiki.ros.org/catkin/package.xml

package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>ros_package_template</name>
  <version>0.1.0</version>
  <description>A ROS package that...</description>
  <maintainer email="tlankhorst@ethz.ch">Tom Lankhorst</maintainer>
  <license>BSD</license>
  <url type="website">https://github.com/leggedrobotics/ros_...</url>
  <author email="tlankhorst@ethz.ch">Tom Lankhorst</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>std_msgs</depend>

  <build_depend>message_generation</build_depend>
</package>
```

ROS Packages

CMakeLists.xml

The CMakeLists.txt is input to the CMake build system

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

CMakeLists.txt

```
cmake_minimum_required (VERSION 3.10.2 )
project (ros_package_template )

## Use C++14, or 11...
set (CMAKE_CXX_STANDARD 14 )
set (CMAKE_CXX_STANDARD_REQUIRED TRUE )

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

...
```

More info

wiki.ros.org/catkin/CMakeLists.txt

ROS Packages

CMakeLists.xml Example

```
cmake_minimum_required(VERSION 3.10.2)
project(smb_highlevel_controller)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED TRUE)

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/SmbHighlevelController.cpp
)

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

Use the same name as in the package.xml

Use C++11 by default (or 14)

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 an executable, the node, with two src files

Specify libraries to link the executable against

Setup a Project in Eclipse

- Build the Eclipse project files with additional build flags

```
> catkin build package_name -G"Eclipse CDT4 - Unix Makefiles"
```

- To use flags by default in your catkin environment, use the `catkin config` command. E.g., to build in release mode:

```
> catkin config -G"Eclipse CDT4 - Unix Makefiles"
```

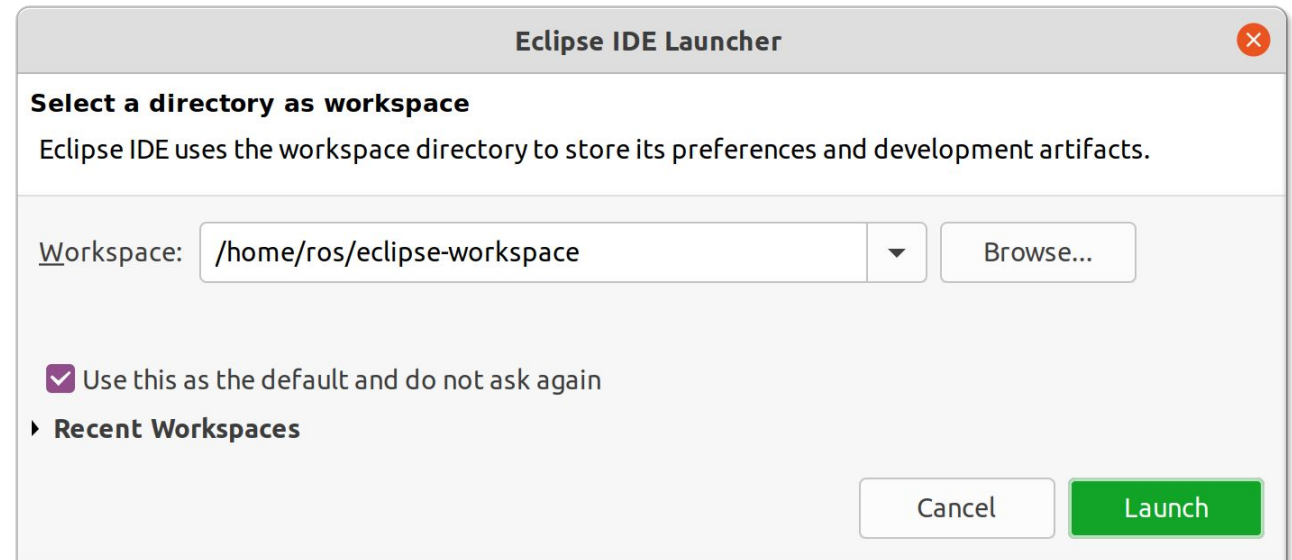
- The Eclipse project files will be generated in the build folder, in e.g.:
~/Workspaces/smb_ws/build

More info

catkin-tools.readthedocs.io/en/latest/verbs/catkin_config.html
github.com/leggedrobotics/ros_best_practices/wiki#catkin-build-flags

Setup a Project in Eclipse

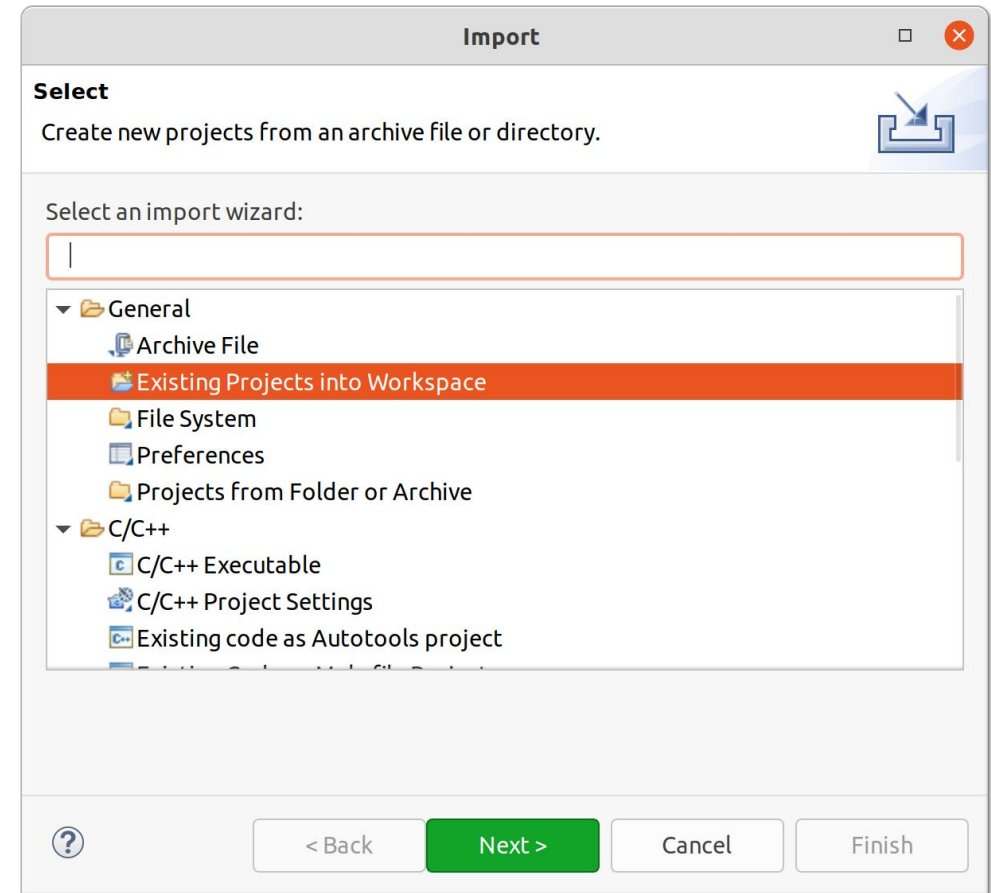
- Start Eclipse and set the workspace folder



Setup a Project in Eclipse

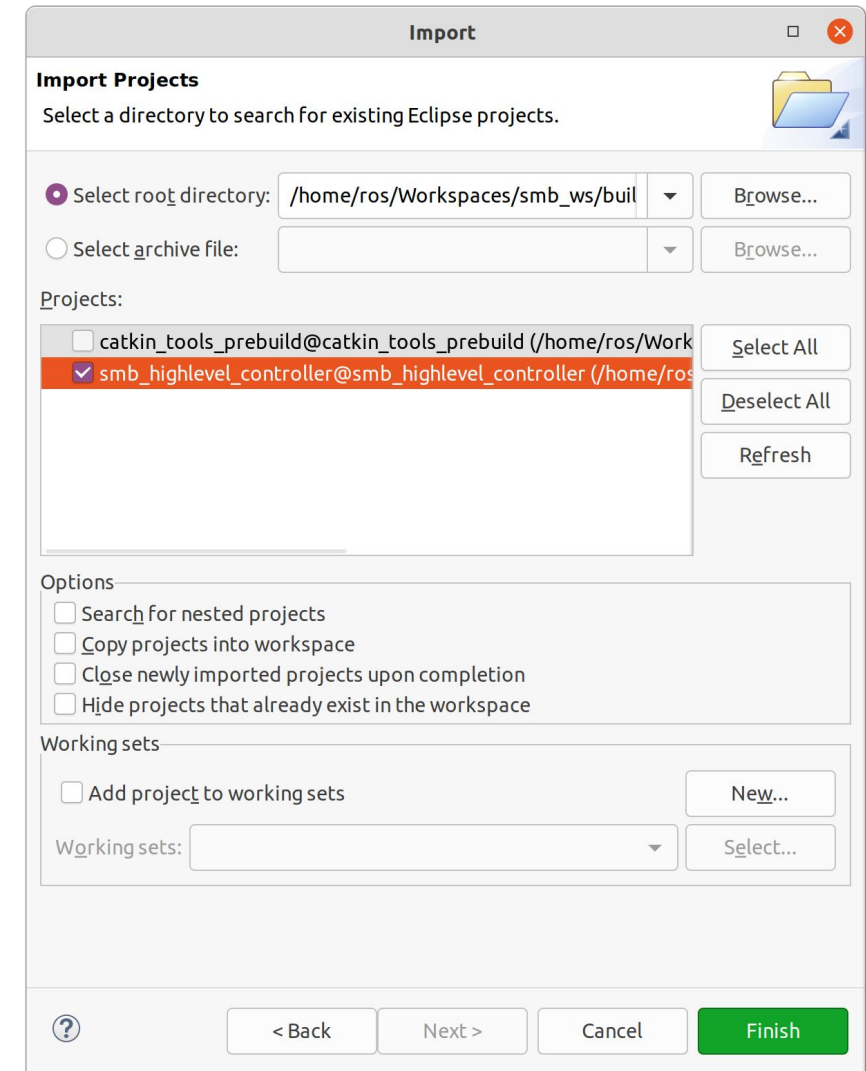
- Import your project to Eclipse

File > Import > General
> Existing Projects into Workspace



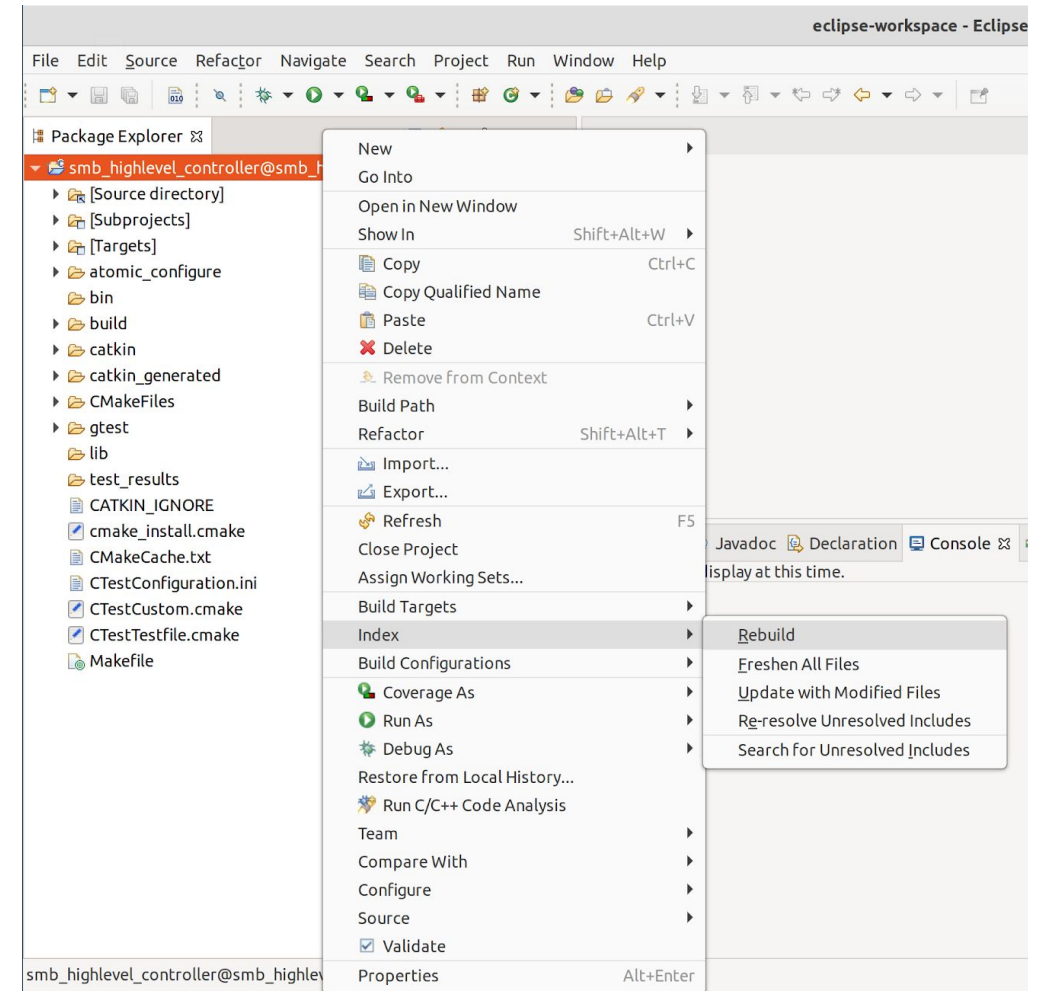
Setup a Project in Eclipse

- The project files can be imported from the ROS workspace folder



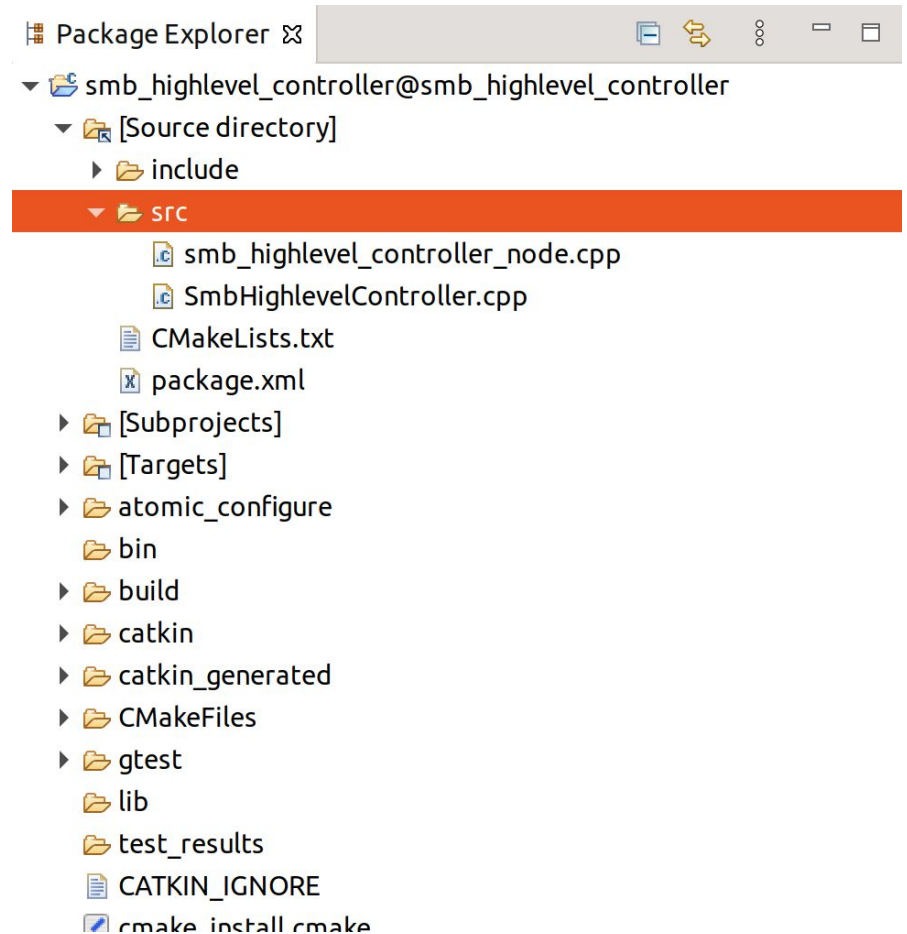
Setup a Project in Eclipse

- Rebuild the C/C++ index of your project by Right click on Project □ Index □ Rebuild
- Resolving the includes enables
 - Fast navigation through links (Ctrl + click)
 - Auto-completion (Ctrl + Space)
 - Building (Ctrl + B) and debugging your code in Eclipse



Setup a Project in Eclipse

- Within the project a link [Source directory] is provided such that you can edit your project
- Useful Eclipse shortcuts
 - Ctrl + Space: Auto-complete
 - Ctrl + /: Comment / uncomment line or section
 - Ctrl + Shift + F: Auto-format code using code formatter
 - Alt + Arrow Up / Arrow Down: Move line or selection up or down
 - Ctrl + D: Delete line



Other IDEs

- Underlying CMake build-systems provides flexibility

wiki.ros.org/IDEs

- E.g.:
 - CLion
 - Vim
 - VSCode

Not supported during the course

ROS C++ Client Library (roscpp)

Essential components of the client library

- Initialization and spinning
- Node handle
- Logging
- Subscriber / Publisher
- Parameters

Discussed in lecture 4

- Services
- Actions
- Time

ROS C++ Client Library (*roscpp*)

Initialization and spinning

hello_world.cpp

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

int main(int argc, char* argv[])
{
    ros::init(argc, argv, "hello_world");
    ros::NodeHandle nodeHandle;
    ros::Rate loopRate(10);

    unsigned int count = 0;
    while (ros::ok()) {
        ROS_INFO_STREAM("Hello World " << count);
        ros::spinOnce();
        loopRate.sleep();
        count++;
    }

    return 0;
}
```

ROS main header file include

`ros::init(...)` has to be called before other ROS functions

The node handle is the access point for communications with the ROS system (topics, services, parameters)

`ros::Rate` is a helper class to run loops at a desired frequency

`ros::ok()` checks if a node should continue running

Returns false if SIGINT is received (Ctrl + C) or `ros::shutdown()` has been called

`ROS_INFO()` logs messages to the filesystem

`ros::spinOnce()` processes incoming messages via callbacks

More info

wiki.ros.org/roscpp

wiki.ros.org/roscpp/Overview

ROS C++ Client Library (*roscpp*)

Node Handle

- There are four main types of node handles

1. Default (public) node handle:
`nh_ = ros::NodeHandle();`
2. Private node handle:
`nh_private_ = ros::NodeHandle("~");`
3. Namespaced node handle:
`nh_eth_ = ros::NodeHandle("eth");`
4. Global node handle:
`nh_global_ = ros::NodeHandle("/");`

Recommended

Not recommended

For a *node* in *namespace* looking up *topic*, these will resolve to:

`/namespace/topic`

`/namespace/node/topic`

`/namespace/eth/topic`

`/topic`

More info

wiki.ros.org/roscpp/Overview/NodeHandles

ROS C++ Client Library (*roscpp*)

Logging

- Mechanism for logging human readable text from nodes in the console and to log files
- Instead of `std::cout`, use e.g. `ROS_INFO`
- Automatic logging to **console**, log **file**, and `/rosout` **topic**
- Different severity levels (INFO, WARN, etc.)
- Supports both printf- and stream-style formatting

```
ROS_INFO("Result: %d", result); // printf
ROS_INFO_STREAM("Result: " << result);
```

- Further features such as conditional, throttled, delayed logging etc.

	Debug	Info	Warn	Error	Fatal
stdout	✓	✓			
stderr			✓	✓	✓
Log file	✓	✓	✓	✓	✓
/rosout	✓	✓	✓	✓	✓

! To see the output in the console, set the output configuration to screen in the launch file

```
<launch>
  <node name="listener" ... output="screen"/>
</launch>
```

More info

wiki.ros.org/rosconsole

wiki.ros.org/roscpp/Overview/Logging

ROS C++ Client Library (*roscpp*)

Subscriber

- When a message is received, callback function is called with the contents of the message as argument
- Start listening to a topic by calling the method `subscribe()` of the node handle

```
ros::Subscriber subscriber =  
nodeHandle.subscribe(topic, queue_size,  
callback_function);
```

- Hold on to the subscriber object until you want to unsubscribe

`ros::spin()` processes callbacks and will not return until the node has been shutdown

listener.cpp

```
#include <ros/ros.h>  
#include <std_msgs/String.h>  
  
void chatterCallback(const std_msgs::String& msg)  
{  
    ROS_INFO("I heard: [%s]", msg.data.c_str());  
}  
  
int main(int argc, char* argv[])  
{  
    ros::init(argc, argv, "listener");  
    ros::NodeHandle nodeHandle;  
  
    ros::Subscriber subscriber =  
        nodeHandle.subscribe("chatter",10,chatterCallback);  
  
    ros::spin();  
    return 0;  
}
```

More info

wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

ROS C++ Client Library (*roscpp*)

Publisher

- Create a publisher with help of the node handle

```
ros::Publisher publisher =  
nodeHandle.advertise<message_type>(topic,  
queue_size);
```

- Create the message contents
- Publish the contents with

```
publisher.publish(message);
```

talker.cpp

```
#include <ros/ros.h>  
#include <std_msgs/String.h>  
  
int main(int argc, char* argv[]) {  
    ros::init(argc, argv, "talker");  
    ros::NodeHandle nh;  
    ros::Publisher chatterPublisher =  
        nh.advertise<std_msgs::String>("chatter", 1);  
    ros::Rate loopRate(10);  
  
    unsigned int count = 0;  
    while (ros::ok()) {  
        std_msgs::String message;  
        message.data = "hello world " + std::to_string(count);  
        ROS_INFO_STREAM(message.data);  
        chatterPublisher.publish(message);  
        ros::spinOnce();  
        loopRate.sleep();  
        count++;  
    }  
    return 0;  
}
```

More info

wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

ROS C++ Client Library (*roscpp*)

Object Oriented Programming



my_package_node.cpp

```
#include <ros/ros.h>
#include "my_package/MyPackage.hpp"
int main(int argc, char* argv[])
{
    ros::init(argc, argv, "my_package");
    ros::NodeHandle nodeHandle("~");

    my_package::MyPackage myPackage(nodeHandle);

    ros::spin();
    return 0;
}
```



MyPackage.hpp



MyPackage.cpp

class MyPackage

Main node class
providing ROS interface
(subscribers, parameters,
timers etc.)



Algorithm.hpp



Algorithm.cpp

class Algorithm

Class implementing the
algorithmic part of the
node

*Note: The algorithmic part of the
code could be separated in a
(ROS-independent) library*

! Specify a function handler to a method from within the class as

```
subscriber_ = nodeHandle_.subscribe(topic, queue_size,
    &ClassName::methodName, this);
```

More info

[wiki.ros.org/roscpp_tutorials/Tutorials/
UsingClassMethodsAsCallbacks](http://wiki.ros.org/roscpp_tutorials/Tutorials/UsingClassMethodsAsCallbacks)

ROS Parameter Server

- Nodes use the *parameter server* to store and retrieve parameters at runtime
- Best used for static data such as configuration parameters
- Parameters can be defined in launch files or separate *YAML* files

List all parameters with

```
> rosparam list
```

Get the value of a parameter with

```
> rosparam get parameter_name
```

Set the value of a parameter with

```
> rosparam set parameter_name value
```

config.yaml

```
camera:
  left:
    name: left_camera
    exposure: 1
  right:
    name: right_camera
    exposure: 1.1
```

package.launch

```
<launch>
  <node name="name" pkg="package" type="node_type">
    <rosparam command="load"
      file="$(find package)/config/config.yaml" />
  </node>
</launch>
```

More info
wiki.ros.org/rosparam

ROS C++ Client Library (*roscpp*)

Parameters

- Get a parameter in C++ with

```
nodeHandle.getParam(parameter_name, variable)
```

- Method returns true if parameter was found, false otherwise
- Global and relative parameter access:
 - Global parameter name with preceding /

```
nodeHandle.getParam("/package/camera/left/exposure", variable)
```

- Relative parameter name (relative to the node handle)

```
nodeHandle.getParam("camera/left/exposure", variable)
```

- For parameters, typically use the private node handle `ros::NodeHandle("~")`

```
ros::NodeHandle nodeHandle("~");
std::string topic;
if (!nodeHandle.getParam("topic", topic)) {
    ROS_ERROR("Could not find topic
parameter!");
}
ROS_INFO_STREAM("Read topic: " << topic);
```

More info

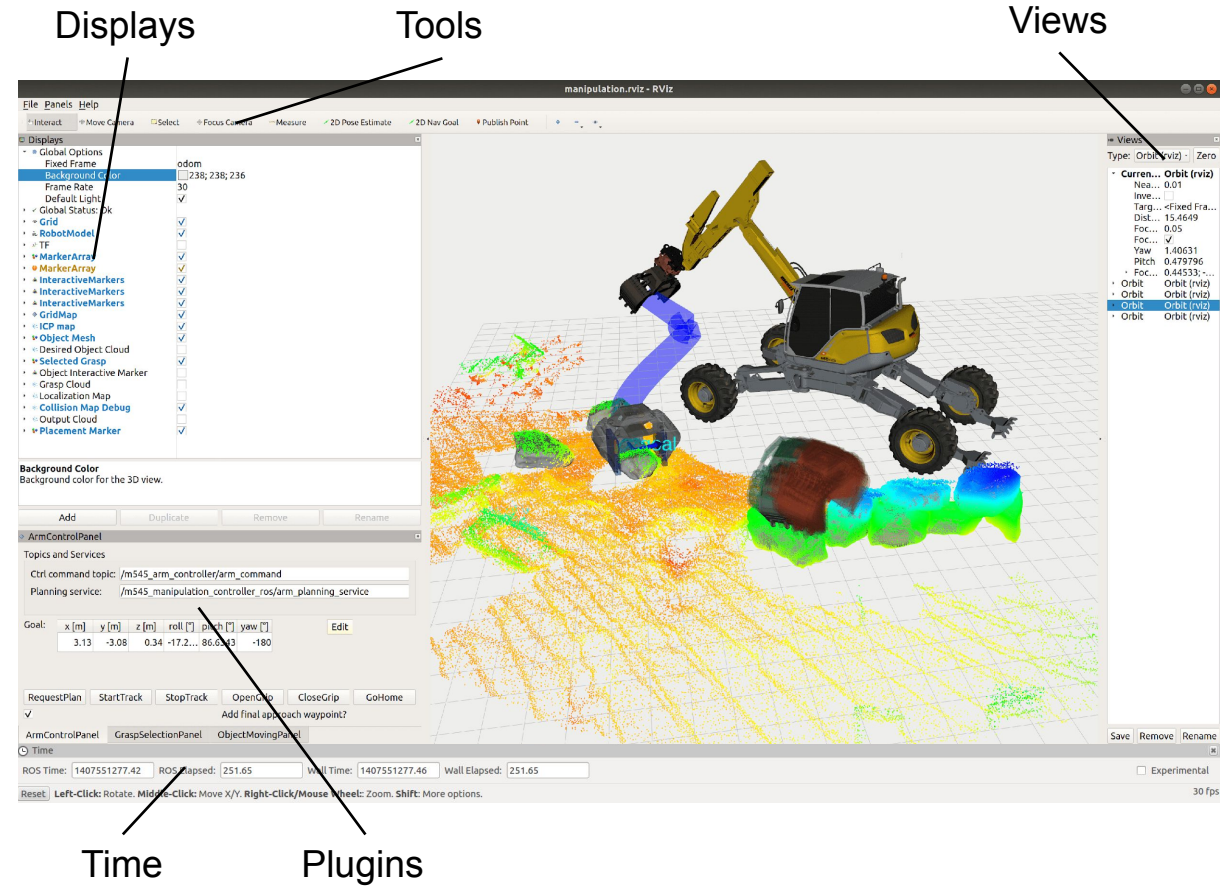
wiki.ros.org/roscpp/Overview/Parameter%20Server

RViz

- 3D visualization tool for ROS
- Subscribes to topics and visualizes the message contents
- Different camera views (orthographic, top-down, etc.)
- Interactive tools to publish user information
- Save and load setup as RViz configuration
- Extensible with plugins

Run RViz with

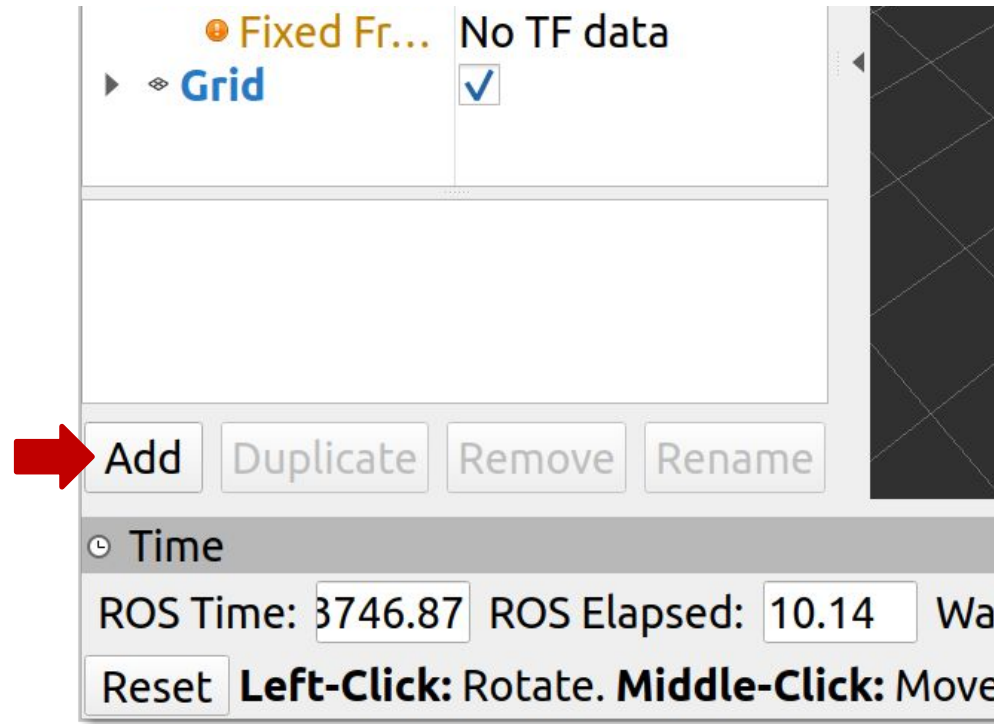
```
> rviz
```



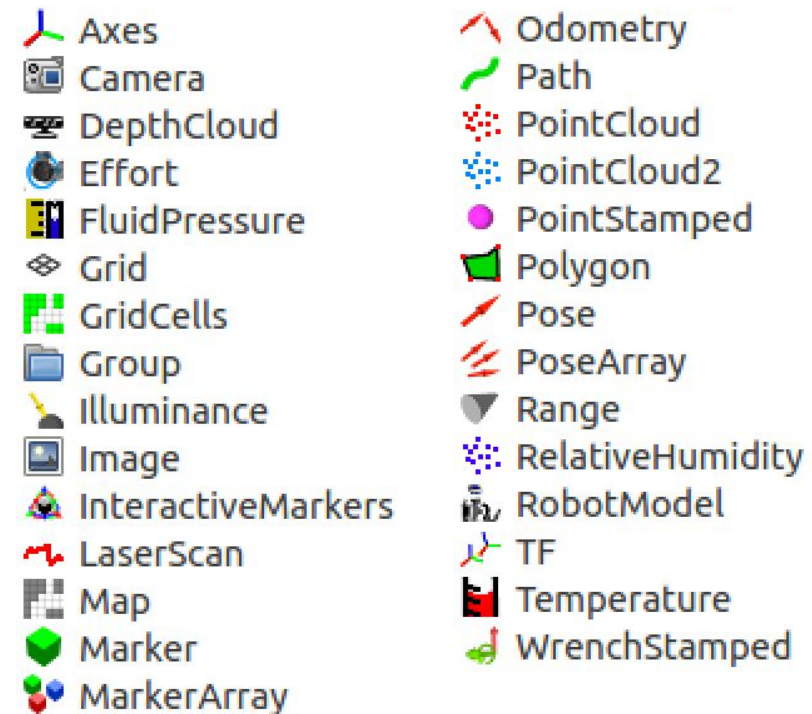
More info
wiki.ros.org/rviz

RViz

Display Plugins



Save configuration with Ctrl + S



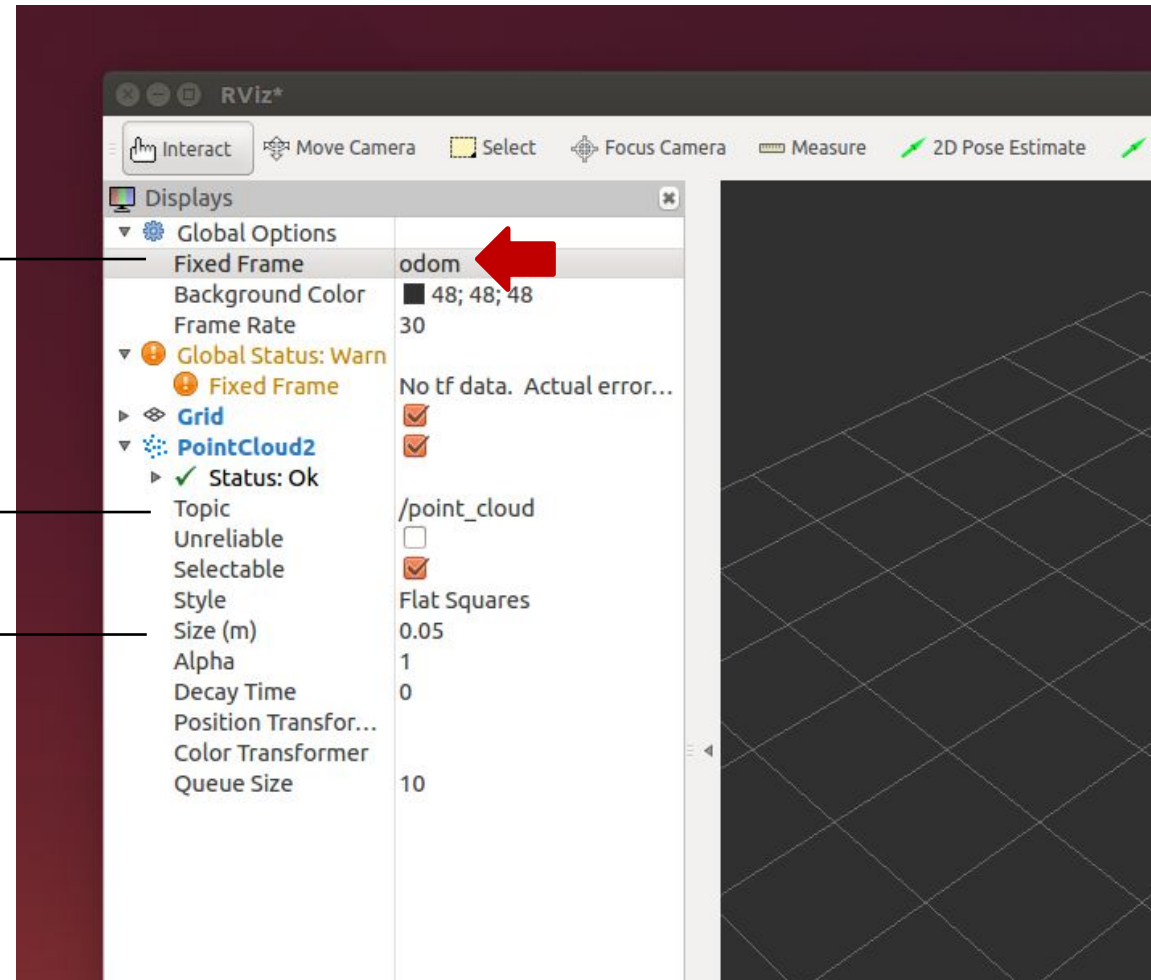
RViz

Visualizing Point Clouds Example

! Frame in which the data is displayed (has to exist!)

Choose the topic for the display

Change the display options (e.g. size)



Further References

- **ROS Wiki**
 - <https://wiki.ros.org/>
- **Installation**
 - <https://wiki.ros.org/ROS/Installation>
- **Tutorials**
 - <https://wiki.ros.org/ROS/Tutorials>
- **Available packages**
 - <https://www.ros.org/browse/>
- **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
- **ROS Cheat Sheet**
 - https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index

Contact Information

ETH Zurich

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

rsl.ethz.ch

Lecturers

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

Course website:

rsl.ethz.ch/education-students/lectures/ros.html