Programming for Robotics
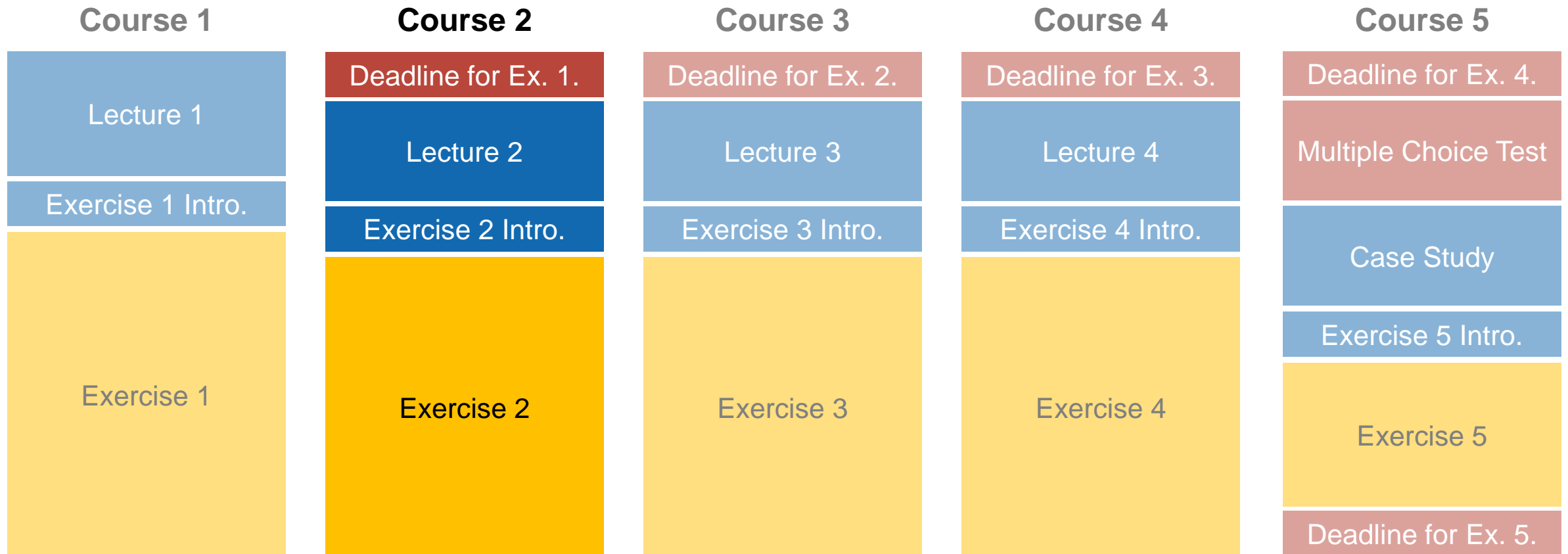## Introduction to ROS

Course 2

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

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.	Deadline for Ex. 4.
Exercise 1 Intro.	Lecture 2	Lecture 3	Lecture 4	Multiple Choice Test
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.

RSL
Robotic Systems Lab

Overview Course 2

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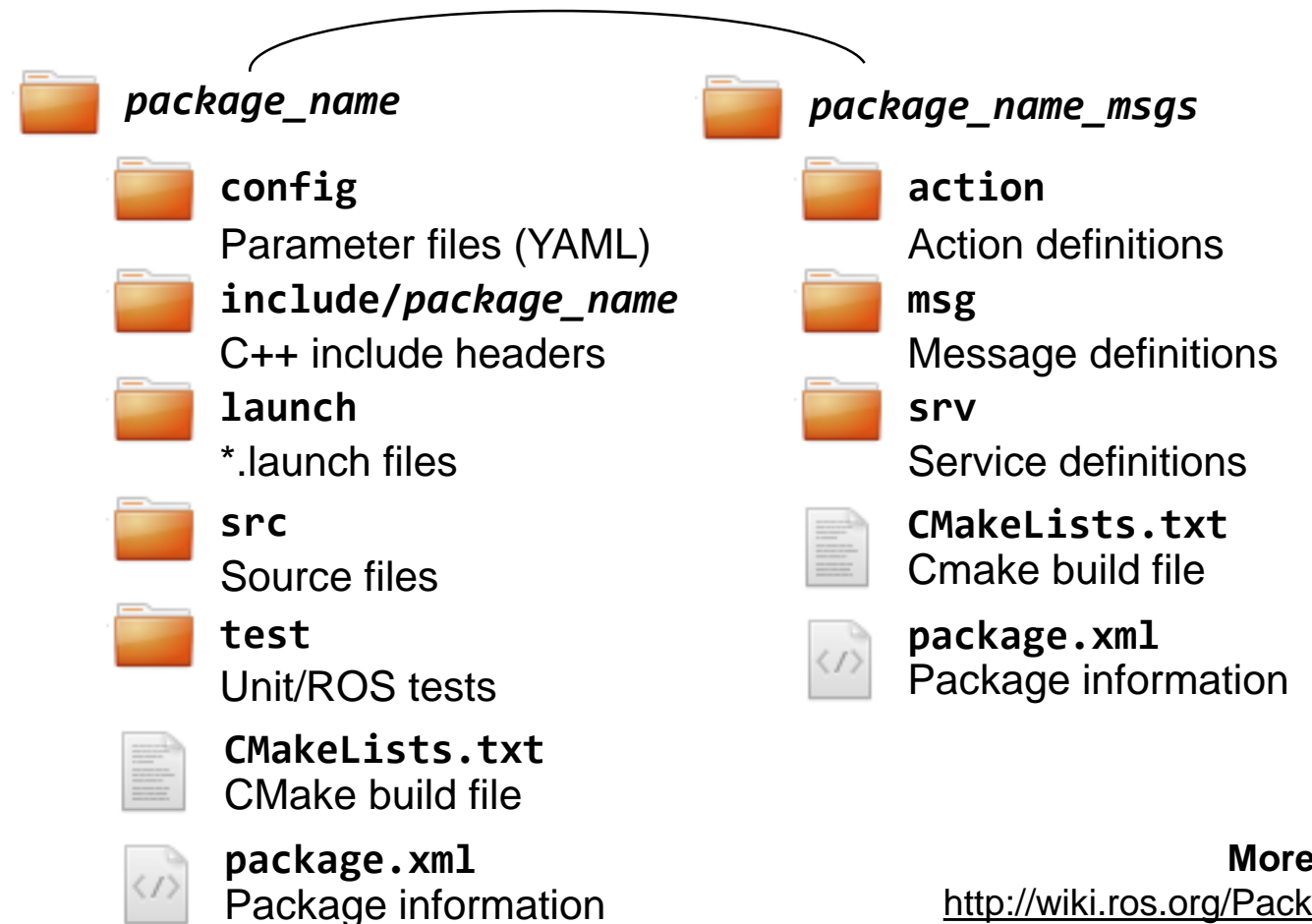- ROS parameter server
- RViz visualization

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!

package_name

config
Parameter files (YAML)

include/package_name
C++ include headers

launch
*.launch files

src
Source files

test
Unit/ROS tests

CMakeLists.txt
CMake build file

package.xml
Package information

package_name_msgs

action
Action definitions

msg
Message definitions

srv
Service definitions

CMakeLists.txt
Cmake build file

package.xml
Package information

More info
http://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**
  - …

package.xml

```xml
<?xml version="1.0"?>
<package format="2">
  <name>ros_package_template</name>
  <version>0.1.0</version>
  <description>A template for ROS packages.</description>
  <maintainer email="pfankhauser@any…">Peter Fankhauser</maintainer>
  <license>BSD</license>
  <url type="website">https://github.com/leggedrobotics/ros_…</url>
  <author email="pfankhauser@anybotics.com">Peter Fankhauser</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>sensor_msgs</depend>
</package>
```

More info
http://wiki.ros.org/catkin/package.xml

ROS Packages
CMakeLists.xml

The `CMakeLists.txt` is the input to the CMakebuild system

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

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_package_template)

## Use C++11
add_definitions(--std=c++11)

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

…
```

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

ROS Packages
CMakeLists.xml Example

```cmake
cmake_minimum_required(VERSION 2.8.3)
project(husky_highlevel_controller)
add_definitions(--std=c++11)

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/HuskyHighlevelController.cpp)

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

Use the same name as in the `package.xml`

We use C++11 by default

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 a C++ executable

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 --cmake-args -G"Eclipse CDT4 - Unix Makefiles"
-D__cplusplus=201103L D__GXX_EXPERIMENTAL_CXX0X__=1
```

- To use flags by default in your catkin environment, use the `catkin config` command.
- The Eclipse project files will be generated in `~/catkin_ws/build`

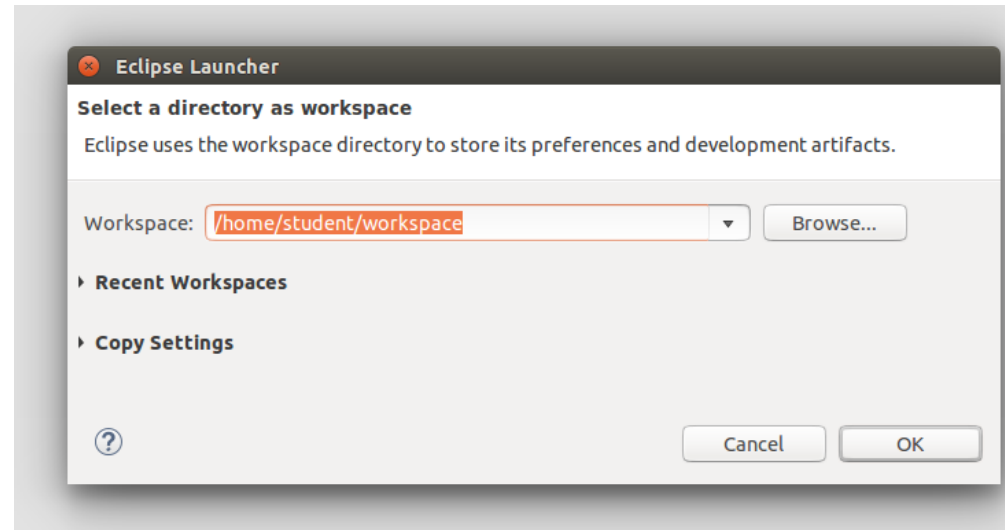The build flags are already setup in the provided installation.

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

RSL
Robotic Systems Lab

Setup a Project in Eclipse

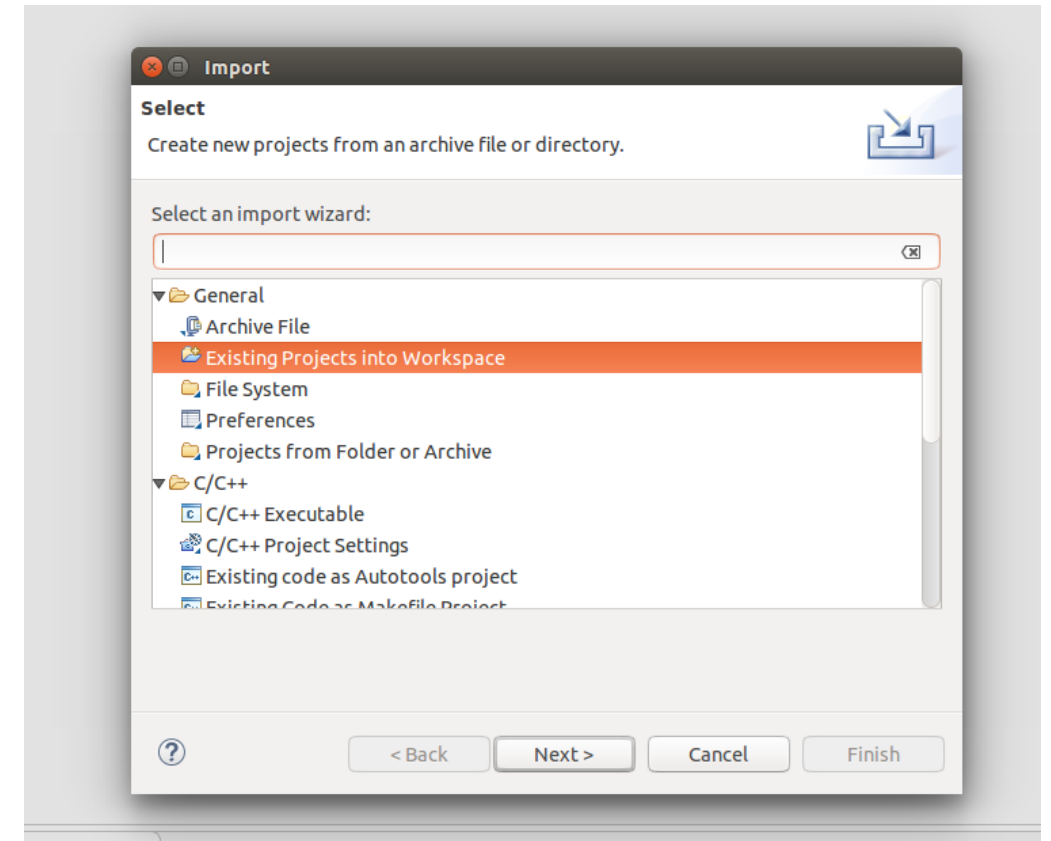▪ Start Eclipse and set the workspace folder

The Eclipse workspace is already set in the provided installation.
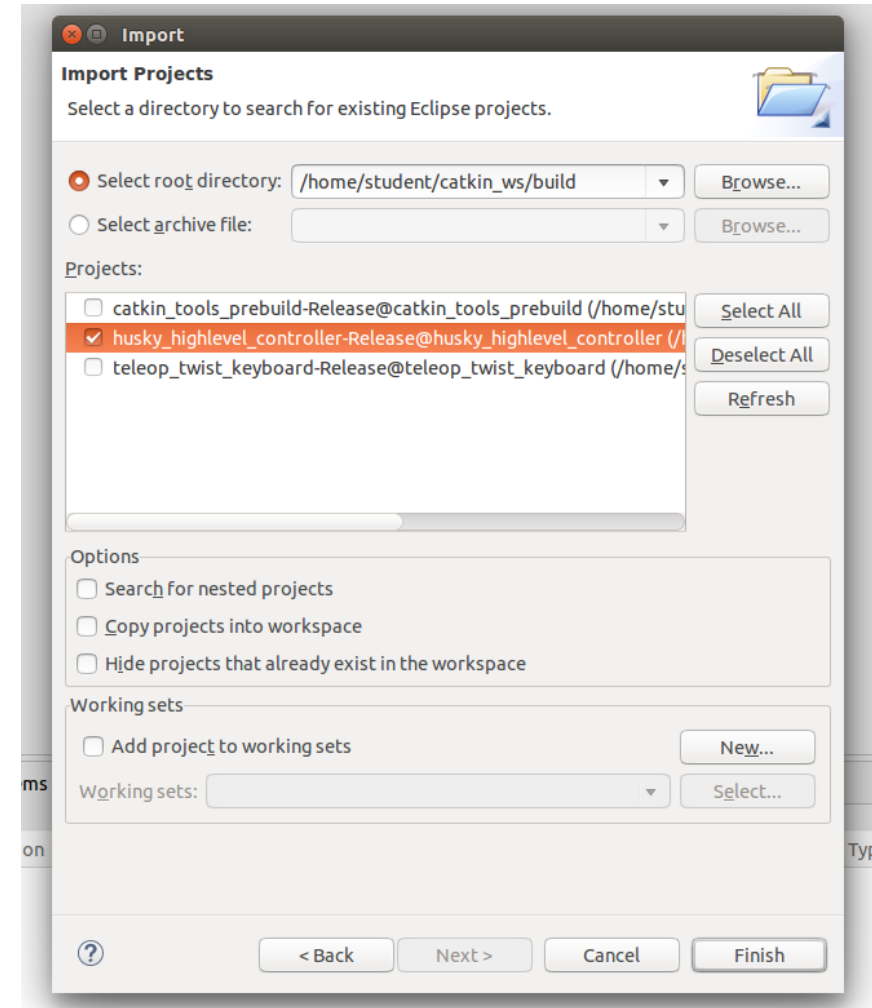
Setup a Project in Eclipse

- 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 `~/catkin_ws/build` 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
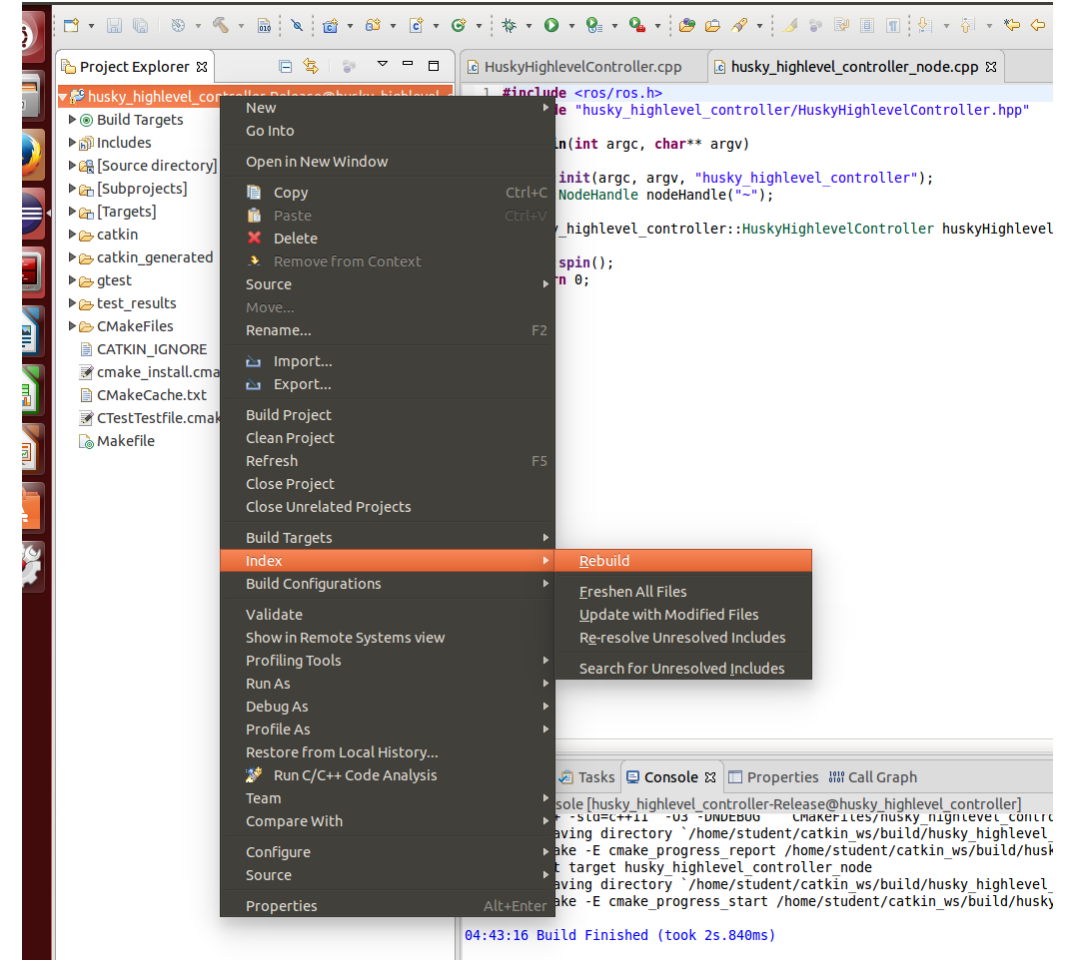
Setup a Project in Eclipse

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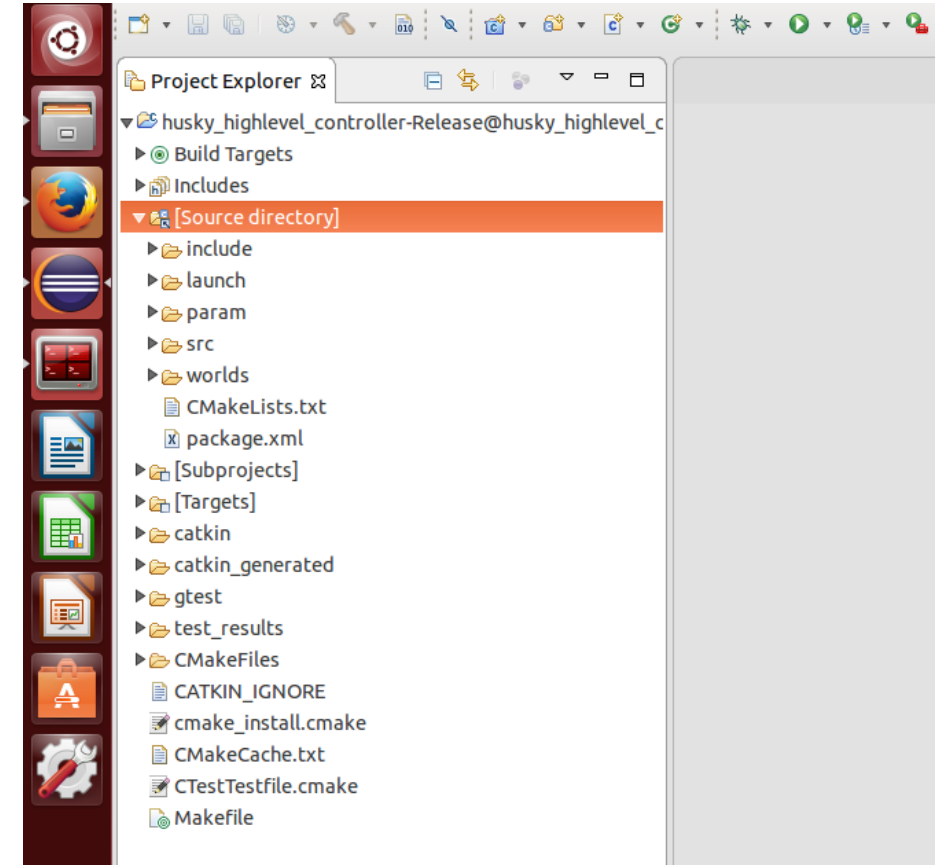
ROS C++ Client Library (*roscpp*)

hello_world.cpp

```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 calling 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
http://wiki.ros.org/roscpp
http://wiki.ros.org/roscpp/Overview

RSL
Robotic Systems Lab

ROS C++ Client Library (*roscpp*)
Node Handle

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

- There are four main types of node handles

  1. Default (public) node handle:
     `nh_ = ros::NodeHandle();`

     /namespace/topic

  2. Private node handle:
     `nh_private_ = ros::NodeHandle("~");`

     Recommended

     /namespace/node/topic

  3. Namespaced node handle:
     `nh_eth_ = ros::NodeHandle("eth");`

     /namespace/eth/topic

  4. Global node handle:
     `nh_global_ = ros::NodeHandle("/");`

     Not recommended

     /topic

More info
http://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, Error etc.)

- Supports both printf- and stream-style formatting

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

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

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

! 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
http://wiki.ros.org/rosconsole
http://wiki.ros.org/roscpp/Overview/Logging

RSL Robotic Systems Lab

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

listener.cpp

- Start listening to a topic by calling the method `subscribe()` of the node handle

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

- When a message is received, callback function is called with the contents of the message as argument

- 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

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

RSL
Robotic Systems Lab

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);
```

More info
http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

talker.cpp

```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;
}
```

ROS C++ Client Library (*roscpp*)
Object Oriented Programming

my_package_node.cpp

```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

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

More info
http://wiki.ros.org/roscpp_tutorials/Tutorials/
UsingClassMethodsAsCallbacks

RSL
Robotic Systems Lab

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

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

package.launch

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

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

ROS Parameter Server
C++ API

- 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("~")`

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

More info
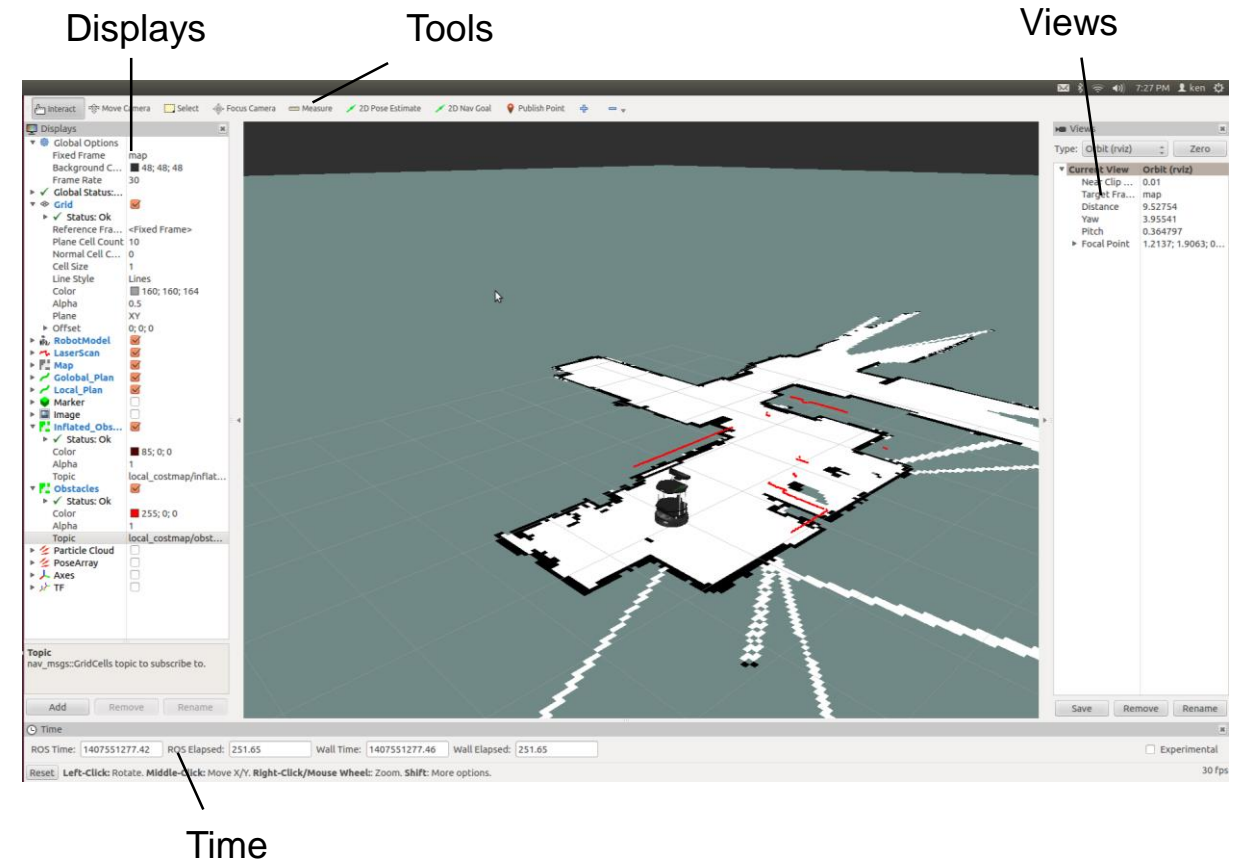http://wiki.ros.org/roscpp/Overview/Parameter%20Server

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
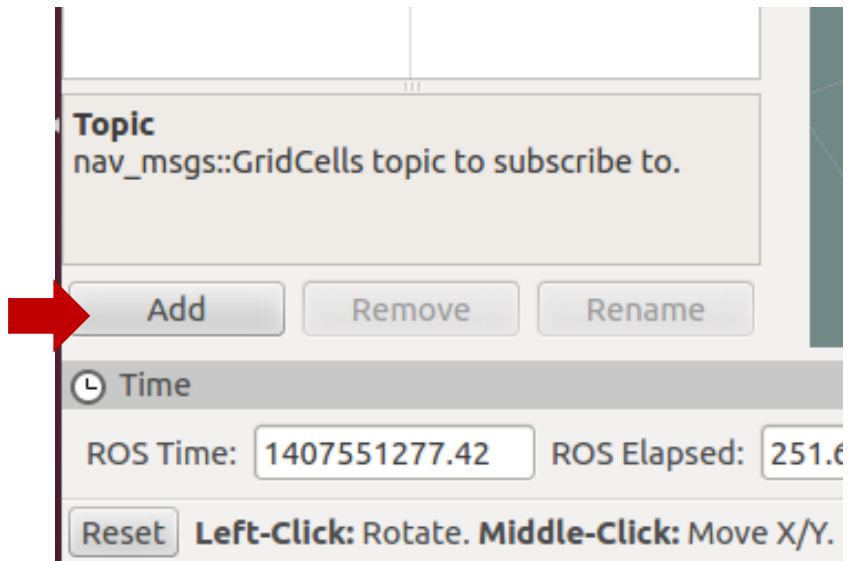- Extensible with plugins
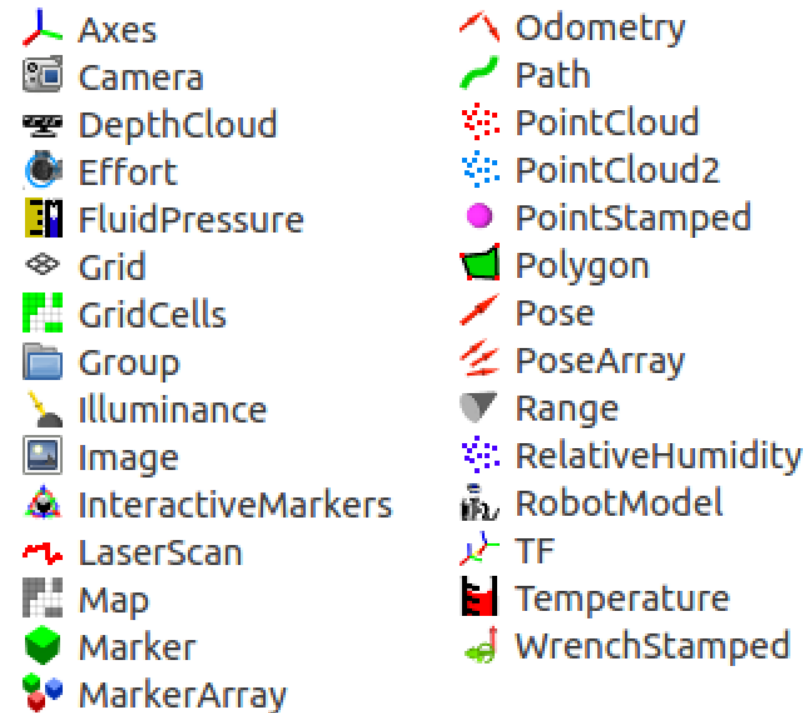
Run RViz with

```
> rosrun rviz rviz
```


Displays   Tools   Views

Time

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

RViz
Display Plugins

Topic
nav_msgs::GridCells topic to subscribe to.

Add   Remove   Rename

🕐 Time

ROS Time: 1407551277.42   ROS Elapsed: 251.6

Reset | **Left-Click:** Rotate. **Middle-Click:** Move X/Y.

Save configuration with `Ctrl + S`

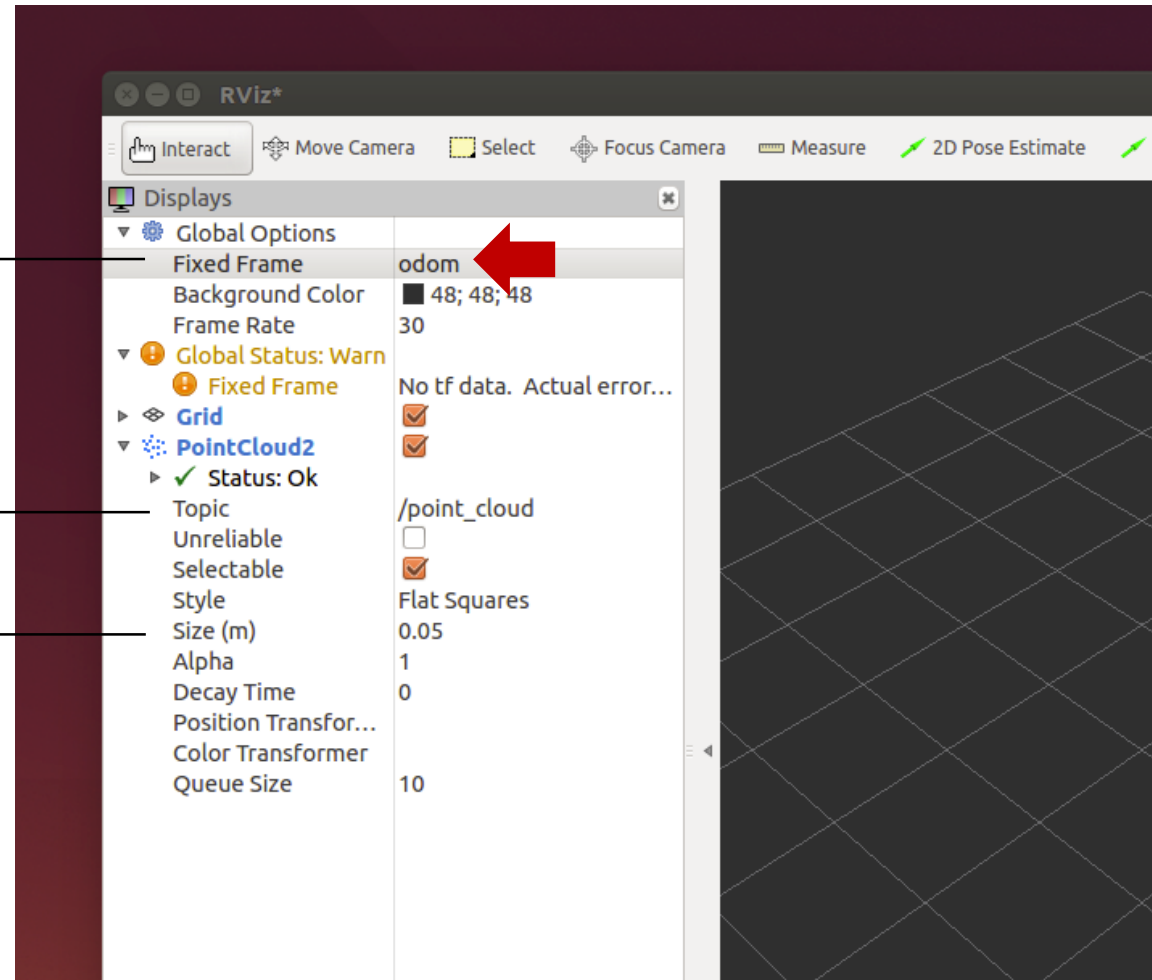|	
Axes	Odometry
Camera	Path
DepthCloud	PointCloud
Effort	PointCloud2
FluidPressure	PointStamped
Grid	Polygon
GridCells	Pose
Group	PoseArray
Illuminance	Range
Image	RelativeHumidity
InteractiveMarkers	RobotModel
LaserScan	TF
Map	Temperature
Marker	WrenchStamped
MarkerArray	

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**
  - 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