Introduction to ROS

(Robot Operating System)

HMW-Alexander

Nagoya University

2016-03-29

Contents

The Goal of This Course

- Know why we need ROS.
- Know how ROS works.
- Know basic operations to play with ROS.

[1] No real practice, we will use bag data for instead.

Contents

The Goal of This Course

- Know why we need ROS.
- Know how ROS works.
- Know basic operations to play with ROS.

[1] No real practice, we will use bag data for instead.

Contents

The Goal of This Course

- Know why we need ROS.
- Know how ROS works.
- Know basic operations to play with ROS.

[1] No real practice, we will use bag data for instead.

Contents

- **Introduction**
  - What is ROS?
  - Why do we need ROS?
  - How does ROS Work?
- **Getting Started with ROS**
  - Turtle Example
  - Basic Operations
- **Practice**
  - 2D LiDAR and SLAM
  - Data Collection[1], Visualization and Processing

The Goal of This Course

- Know why we need ROS.
- Know how ROS works.
- Know basic operations to play with ROS.

[1] No real practice, we will use bag data for instead.

Reference

- **ROS Documentation**: `http://wiki.ros.org`
- **ROS Introduction Lecture**: Real-world data circulation science leader human resource development program
- **"A Gentle Introduction to ROS"** – Jason M. O'Kane.
- **"Learning ROS for Robotics Programming"** – Aaron Martinez & Enrique Fernndez

Introduction

ROS: Five Years (Video)

Five Years.mp4

What is ROS (Robot Operating System)?

A meta-operating system for robot software development

- **Partly based on a ”real” operating system**
  - processes management system
  - file system
  - user interface
  - programming utilities (compiler, threading model, etc.)
- **A meta-operating system**
  - built on the top of an operating system
  - works alongside an operating system
  - allows different processes to communicate with each other at runtime
- **A framework for robot software development**
  - easily share a piece of software with other robots [philosophy of ROS]

What is ROS (Robot Operating System)?

A meta-operating system for robot software development

- **Partly based on a ”real” operating system**
  - processes management system
  - file system
  - user interface
  - programming utilities (compiler, threading model, etc.)
- **A meta-operating system**
  - built on the top of an operating system
  - works alongside an operating system
  - allows different processes to communicate with each other at runtime
- **A framework for robot software development**
  - easily share a piece of software with other robots [philosophy of ROS]

What is ROS (Robot Operating System)?

A meta-operating system for robot software development

- **Partly based on a ”real” operating system**
  - processes management system
  - file system
  - user interface
  - programming utilities (compiler, threading model, etc.)
- **A meta-operating system**
  - built on the top of an operating system
  - works alongside an operating system
  - allows different processes to communicate with each other at runtime
- **A framework for robot software development**
  - easily share a piece of software with other robots [philosophy of ROS]

What is ROS (Robot Operating System)?

A meta-operating system for robot software development

- **Partly based on a ”real” operating system**
  - processes management system
  - file system
  - user interface
  - programming utilities (compiler, threading model, etc.)
- **A meta-operating system**
  - built on the top of an operating system
  - works alongside an operating system
  - allows different processes to communicate with each other at runtime
- **A framework for robot software development**
  - easily share a piece of software with other robots [philosophy of ROS]

What is ROS (Robot Operating System)?

ROS is not...

- **is not a programming language**
  - ROS programs are routinely written in C++.
  - client libraries are also available for Python, Java, Lisp, etc.
- **is not only a library**
  - ROS also includes a central server, a set of command-line tools, a set of graphical tools, and a build system.
- **is not an integrated development environment**
  - ROS can be used with most popular IDEs.

What is ROS (Robot Operating System)?

ROS is not...

- **is not a programming language**
  - ROS programs are routinely written in C++.
  - client libraries are also available for Python, Java, Lisp, etc.
- **is not only a library**
  - ROS also includes a central server, a set of command-line tools, a set of graphical tools, and a build system.
- **is not an integrated development environment**
  - ROS can be used with most popular IDEs.

What is ROS (Robot Operating System)?

ROS is not...

- **is not a programming language**
  - ROS programs are routinely written in C++.
  - client libraries are also available for Python, Java, Lisp, etc.
- **is not only a library**
  - ROS also includes a central server, a set of command-line tools, a set of graphical tools, and a build system.
- **is not an integrated development environment**
  - ROS can be used with most popular IDEs.

What is ROS (Robot Operating System)?

ROS is not...

- **is not a programming language**
  - ROS programs are routinely written in C++.
  - client libraries are also available for Python, Java, Lisp, etc.
- **is not only a library**
  - ROS also includes a central server, a set of command-line tools, a set of graphical tools, and a build system.
- **is not an integrated development environment**
  - ROS can be used with most popular IDEs.

Why do we need ROS?

Programming in Your CS courses

- Seldom run two or more programs simultaneously.
- Seldom make programs that talk to each other.
- Seldom consider letting others take an advantage of your programs.

Before ROS

- It is very hard to realize *inter-communication* and *extensibility* by raw APIs from operate system directly

After ROS

- Hierarchical abstraction and management of running programs
- Universal communication between programs
- A collection of powerful programs and code libraries as extension.

Why do we need ROS?

Programming in Your CS courses

- Seldom run two or more programs simultaneously.
- Seldom make programs that talk to each other.
- Seldom consider letting others take an advantage of your programs.

Before ROS

- It is very hard to realize *inter-communication* and *extensibility* by raw APIs from operate system directly

After ROS

- Hierarchical abstraction and management of running programs
- Universal communication between programs
- A collection of powerful programs and code libraries as extension.

Why do we need ROS?

Programming in Your CS courses

- Seldom run two or more programs simultaneously.
- Seldom make programs that talk to each other.
- Seldom consider letting others take an advantage of your programs.

Before ROS

- It is very hard to realize *inter-communication* and *extensibility* by raw APIs from operate system directly

After ROS

- Hierarchical abstraction and management of running programs
- Universal communication between programs
- A collection of powerful programs and code libraries as extension.

How does ROS work? (Three levels of ROS)

File System Level

- package, manifest, message types
- An organization of the ROS related files (application, manifest, definition)

Computation Graph Level

- node, master, parameter server, message, topic, bag
- The graph model and architecture of ROS

Community Level

- distribution, repository, The ROS Wiki, etc..
- The resources from Internet

How does ROS work? (Basic Concepts) I

Package

- all ROS software is organized into packages
- a coherent collection of executable and supporting files

Manifest

- a ”package.xml” file
- defines some details about the package:
  - name, version, maintainer, dependencies

Message Type

- define the type of message in the communication within ROS

How does ROS work? (Basic Concepts) II

Node

- a running instance of a ROS program

Master

- enable the communications of a collection of independent nodes

Parameter Server

- a central location to store parameters for nodes

How does ROS work? (Basic Concepts) III

Message

- nodes communicate with each other through messages
- a message contains data
- pre-defined and user-defined message types

Topics

- each massage must have a name to be routed by the ROS network
- node is sending data = node is publishing a topic
- node is receiving data = node is subscribing a topic

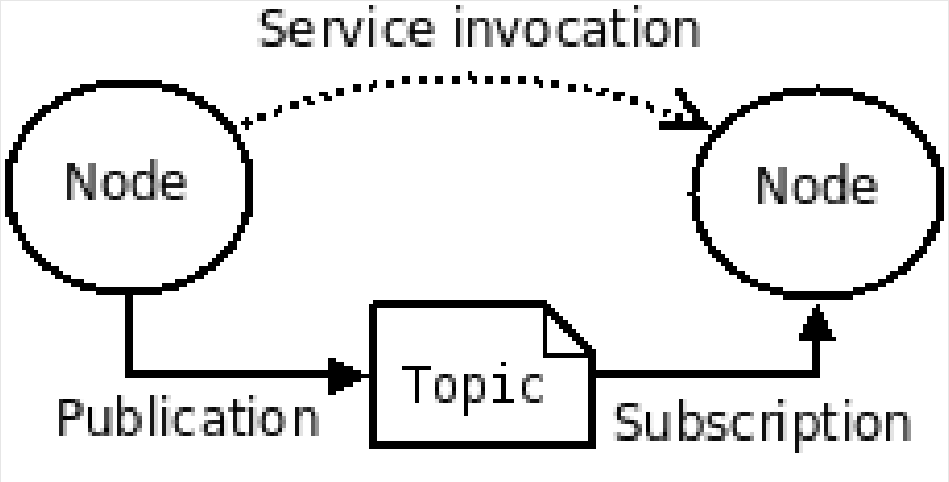
Bags

- a universal format to save and play back ROS message data

How does ROS work?

How does ROS work

- ROS architecture is based on the graph-model
- the node of the graph is an instance of software
- the edge of the graph is the message communication route
- the master of ROS maintains the many-many communication

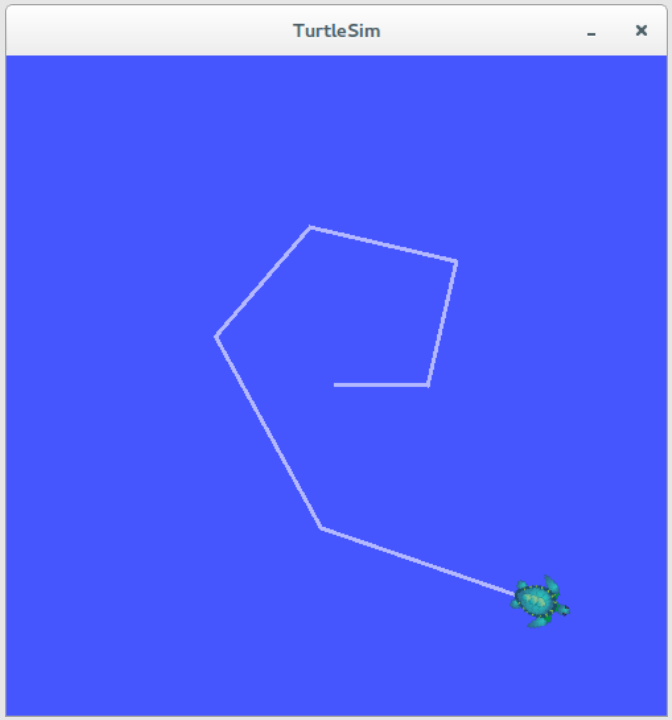


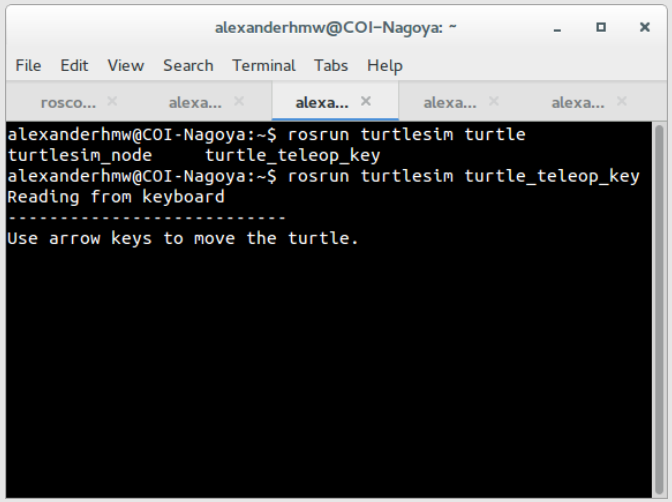
Getting Started with ROS

Architecture with Turtle Example

Starting turtlesim

- roscore
- rosrun turtlesim turtlesim_node
- rosrun turtlesim turtle_teleop_key

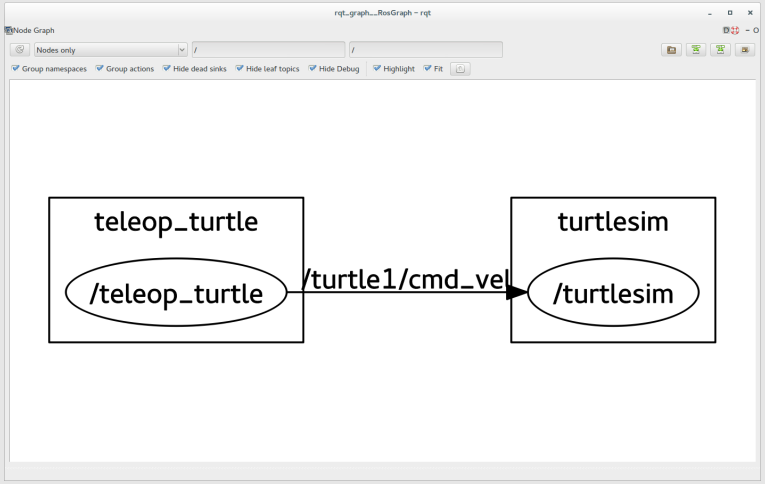


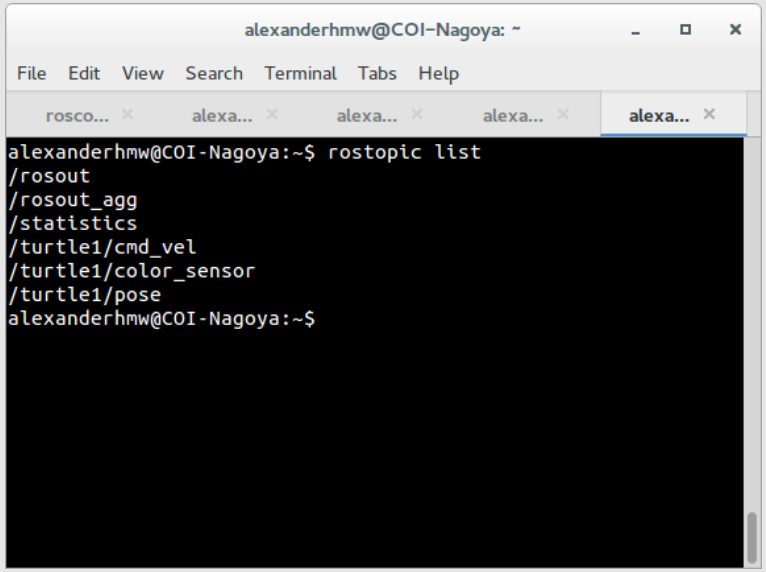


Architecture with Turtle Example

Check ROS Graph-model, Topics and Messages

- rqt_graph
- rostopic list





Basic Operation I

Navigating through the ROS filesystem

- Find package's location: rospack find turtlesim
- List the files inside a package: rosls turtlesim
- Go to the package's folder: roscd turtlesim

Creating an ROS package

- catkin_create_pkg hokuyo_test std_msgs rospy roscpp
- catkin_create_pkg [package_name] [depend1] [depend2] [depend3]

Playing with ROS nodes

- start master: roscore
- rosnode command: rosnode [param] -h
- launch node: rosrun [package] [executable] [ARGS]
  rosrun turtlesim turtlesim_node

Basic Operation II

Interact with Topics and Messages

- rostopic list: list the active topics
- rostopic echo: print messages to the screen
- rostopic info: print information about active topics
- rostopic type: print the topic message type
- rosmsg show: print the message fields
- rostopic pub: publish message to the topic
  rostopic pub -r 1 /turtle1/cmd_vel geometry_msgs/Twist '[1,0,0]' '[0,0,1]'

Basic Operation III

Using the Parameter Server

- rosparam set parameter value: set the parameter
- rosparam get parameter: get the parameter
- rosparam load file: load parameters from the file
- rosparam dump file: dump parameters to the file
- rosparam delete parameter: delete the parameter
- rosparam list: list the parameter names

Some Useful Tools

- rqt_graph: display graph-model
- rviz: visualization tool

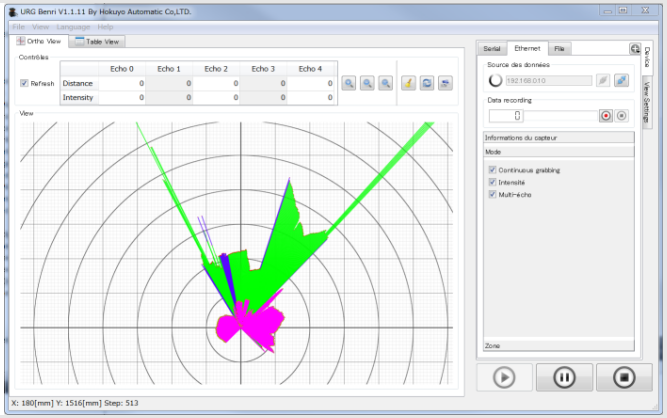
Practice

2D LiDAR and SLAM (Video)

2D LiDAR and SLAM I

2D LiDAR

A set of horizontal laser beams are emitted for range detection.

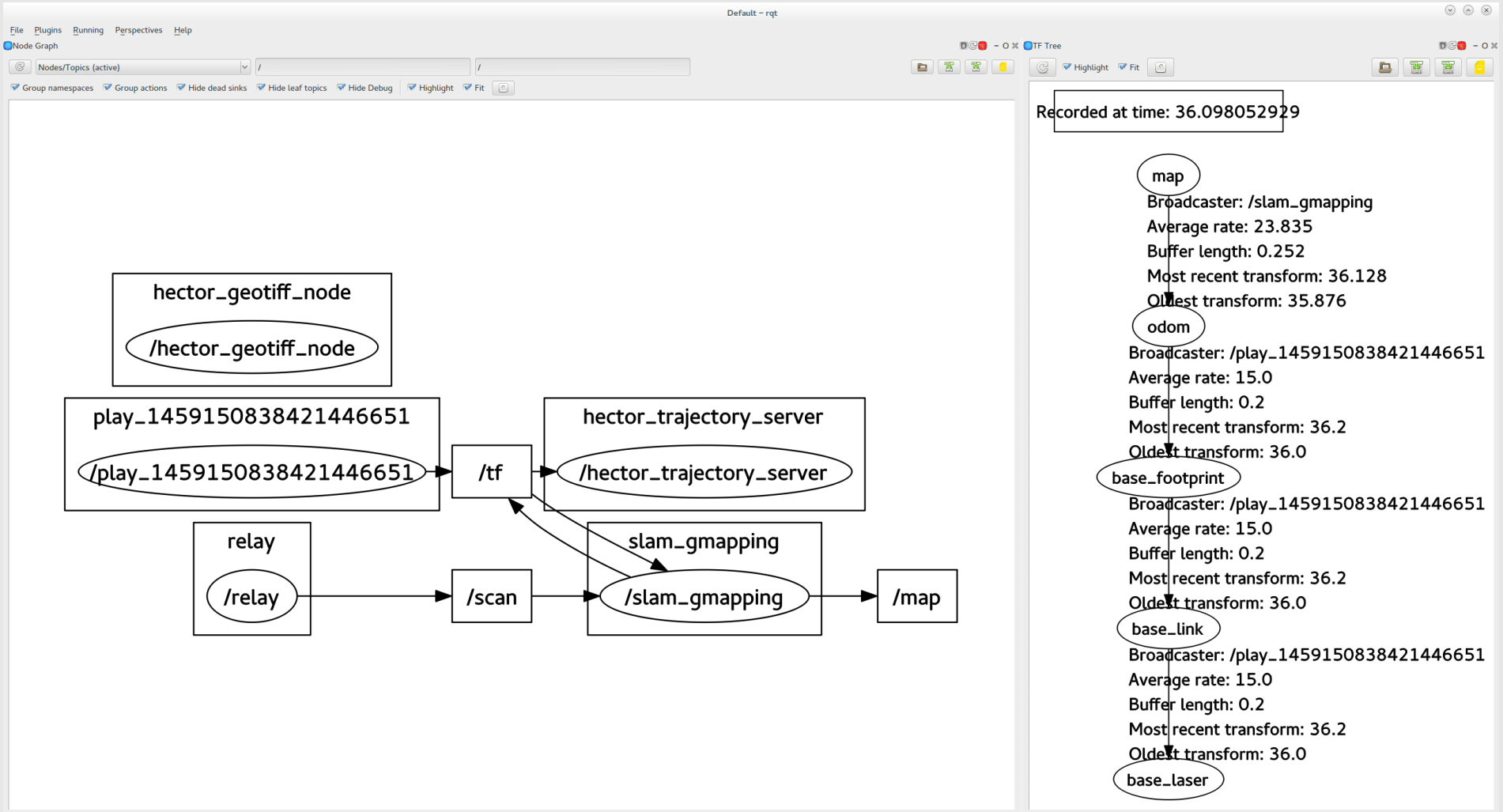


SLAM

Simultaneous Localization and Mapping

2D LiDAR and SLAM II

Graph-Model



Data Collection, Visualization and Processingg I

Preparation

- Install external libraries:
  - sudo apt-get install ros-hydro-gmapping [ros-hydro-hokuyo-node]
- Build a workspace:
  - mkdir ∼/catkin_ws/src
  - cd ∼/catkin_ws/src
  - catkin_init_workspace
- Create a package:
  - catkin_create_pkg hokuyo_test std_msgs rospy roscpp
  - copy the ”launch” folder to path ∼/catkin_ws/src/hokuyo_test/
  - cd ∼/catkin_ws
  - catkin_make

Data Collection, Visualization and Processingg II

Test Package and create a ”maps” folder

- cd ∼/catkin_ws
- source devel/setup.bash[a]
- roscd hokuyo_test
- mkdir maps

[a] Register the package to the system. In the following, you should execute this order when you open a new terminal.

Data Collection, Visualization and Processingg III

Data Collection (No practice)

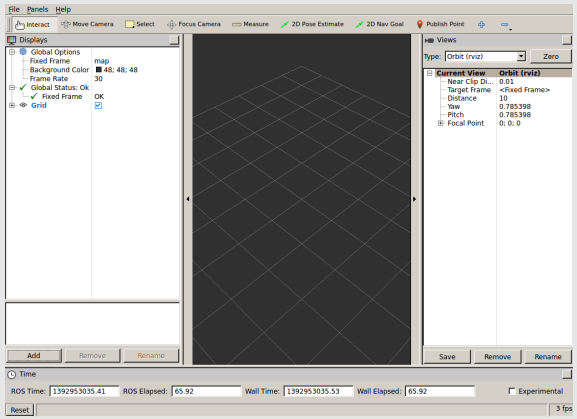
You can play with a Hokuyo laser scanner using the following orders.

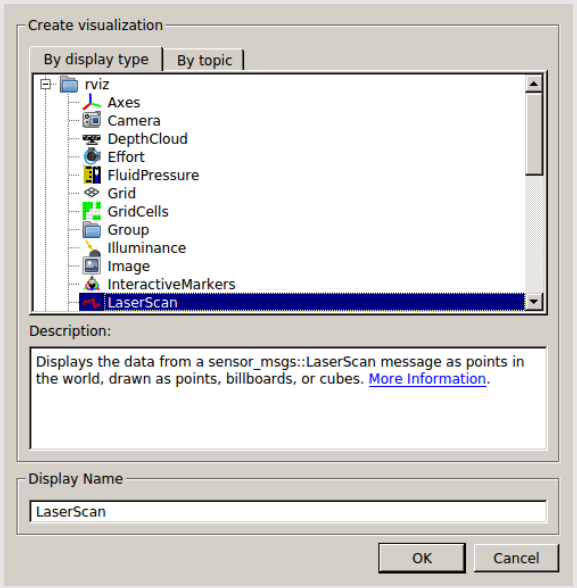
- rosparam set hokuyo_node/calibrate_time false
- rosparam set hokuyo_node/port /dev/ttyACM0
- sudo chmod a+rw /dev/ttyACM0
- rosrun hokuyo_node hokuyo_node
- rosbag record -o hokuyo /scan

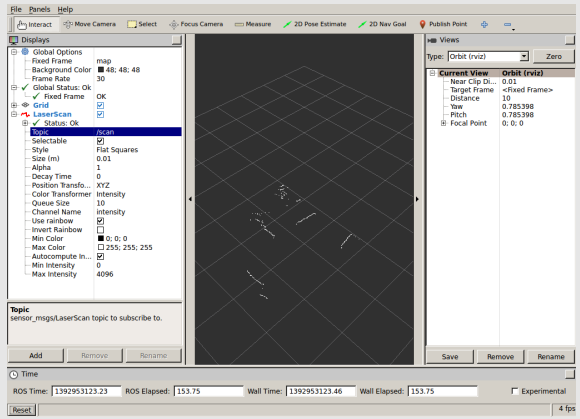
Data Collection, Visualization and Processingg IV

Visualization of Hokuyo Data

- roscore (if you don't start a master)
- rviz
- cd (the path holding hokuyo.bag)
- rosbag play hokuyo.bag −−clock



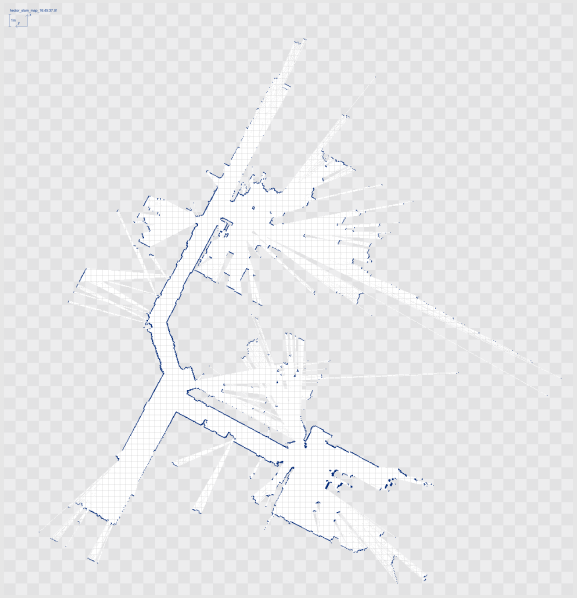




Data Collection, Visualization and Processingg V

Processing (SLAM)

- roslaunch hokuyo_test slam.launch
- rostopic pub syscommand std_msgs/String ”savegeotiff”
- roscd hokuyo_test
- eog maps/hector_slam_map_xx:xx:xx.tif



Conclusion

About ROS

- ROS is a meta-operation system for the robot software development
- Easily realize inter-communication and extensibility of softwares
- ROS is based on graph-model
  - Node is a software instance
  - Edge is a message communication route
  - Master maintains the many-many communication

This Course

- Play with Turtle Example
- Basic Operations
- Practice with 2D LiDAR Data and SLAM
- How to develop a node is not included in this course (see Reference)

Thank you very much!