Introduction to Robot Operating System

Michal Reinštein

Czech Technical University in Prague, Faculty of Electrical Engineering, Dept. Cybernetics, Center for Machine Perception

http://cmp.felk.cvut.cz/~reinsmic/

reinstein.michal@fel.cvut.cz

RECOMMENDED LITERARTURE & LINKS

- ROS http://wiki.ros.org/
- ROS Answers http://answers.ros.org/questions/
- Willow Garage https://www.willowgarage.com/
- Very good book http://www.cse.sc.edu/~jokane/agitr/

What is ROS?

*ROS is an **open-source**, **meta-operating system** for your robot. It provides the services you would expect from an operating system, including **hardware abstraction**, low-level **device control**, implementation of commonly-used functionality, **message-passing** between processes, and **package management**. It also provides tools and libraries for obtaining, **building**, **writing, and running** code across multiple computers.*
--- J. M. O' Kane, USC

>> ROS is a middleware ...

What is middleware?

- Middleware is software providing services to applications beyond those available from the operating system.
- Middleware makes it easier for software developers to perform communication and input/output.
- Most commonly used in the context of distributed applications.
- More specifically: dash in “client-server”.
- Also used in a sense of: a software driver, an abstraction layer hiding details of hardware and software from an application.

What is ROS?

- Stands for **Robot Operating System**, but runs on top of existing OS like (mostly) Linux (Ubuntu officially supported)
- Originates from SAIL/Willow Garage, now moved to OSRF
- Combines several features into a consistent project:
  - A software distribution with a dependency mechanism
  - An integrated build system
  - A communication middleware
  - Helper tools: data visualization, record/replay, etc.

Why ROS?

- Excellent documentation and tutorials
- Reliable software tools for data management
- Accepted standardization for releasing codes & datasets
- Reliable communication system
- Link with OpenCV (2D computer vision), PCL (3D computer vision), and many others …

What is ROS good for?

Distributed computation

- robot systems rely on software that **spans many different processes** and runs **across several different computers**
- need for **communication and data exchange** between multiple processes that may or may not live on the same computer

What is ROS good for?

Software reuse

progress of robotics research >> growing collection of good algorithms for common tasks: *perception, navigation, motion planning, mapping*, etc.

- ROS's standard packages provide stable, debugged implementations of many **important robotics algorithms**.
- ROS's message passing interface is becoming a **standard** for robot software interoperability

>> less time reinventing wheels

What is ROS good for?

Rapid testing

Testing in robotics >> slow, time consuming, error-prone, availability of the physical robots …

- Separation of the low-level direct control of the hardware and high-level processing and decision making
- Low-level programs (and their corresponding hardware) can easily be replaced with a simulator to test the behavior of the high-level part of the system
- Simple way to record and play back sensor data
- Less time spent operating a physical robot

What is ROS not?

- *ROS is not a programming language* - but works best with C++/ Python, interface to Java (Android)
- *ROS is not (only) a library* - but includes client libraries (OpenCV, PCL, ActionLib, SMACH, MoveIt, GAZEBO, etc.)
- *ROS is not an integrated development environment* - but can be used with most popular IDEs (e.g. *Eclipse + pydev*)
- *ROS is not a real-time architecture*

ROS basics …

- **Node:** A single process / computation unit (component of the system)
- **Message:** Data structure used by ROS nodes to communicate
- **Topic:** Broadcast communication between nodes (TCP/UDP)
- **Service:** Synchronous communication between nodes (RPC)
- **Roscore:** ROS (name) server for the whole system management

ROS node & ROS message

Node

- Single process that performs particular computation
- Communicate with each other by passing messages, through topic (**TCP**, **UDP**) or service (**RPC**).
- Connection between two nodes is accomplished using topics through the roscore, which acts as a name server.
- Each ROS node can be written in different language

Message

- Strictly typed data structure, support standard primitive types (*integer, float, Boolean*, etc.) and *arrays*.
- Messages can be composed of arrays of messages.
- Message type is defined in plain text using *Message Description Language* and code is generated for each supported language.

ROS topic & ROS service

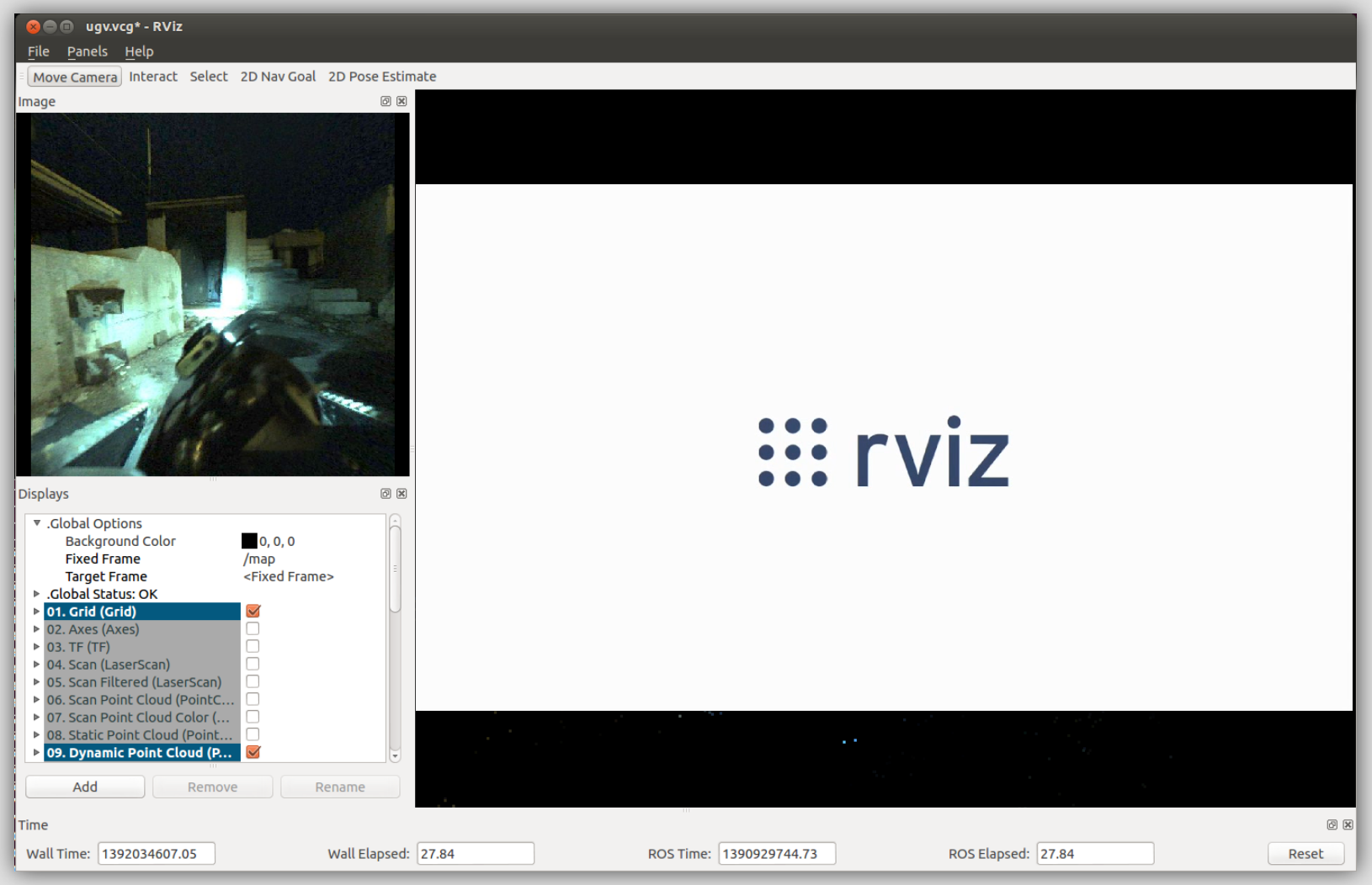
Topic

- A named broadcast stream of messages.
- Generally there can be more publishers of the same topic.
- Publishers are aware if someone is subscribed.
- Topic is defined by name and message type.

Service

- A named synchronous communication.
- There can be just one node providing a service of some name.
- Service is defined by name and two message types – *request* and *reply*.

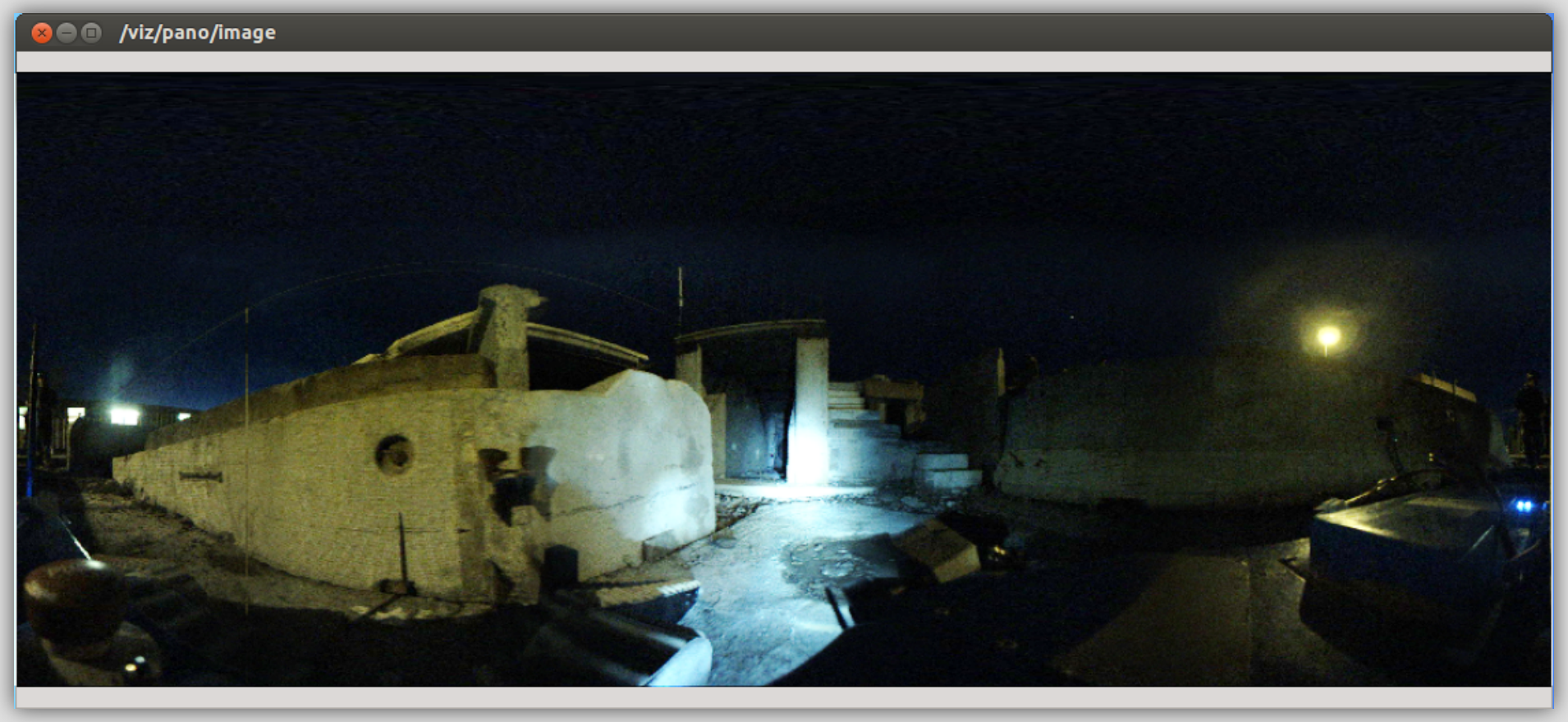
ROS Tools: RVIZ



>> rosrun rviz rviz

http://wiki.ros.org/rviz/UserGuide

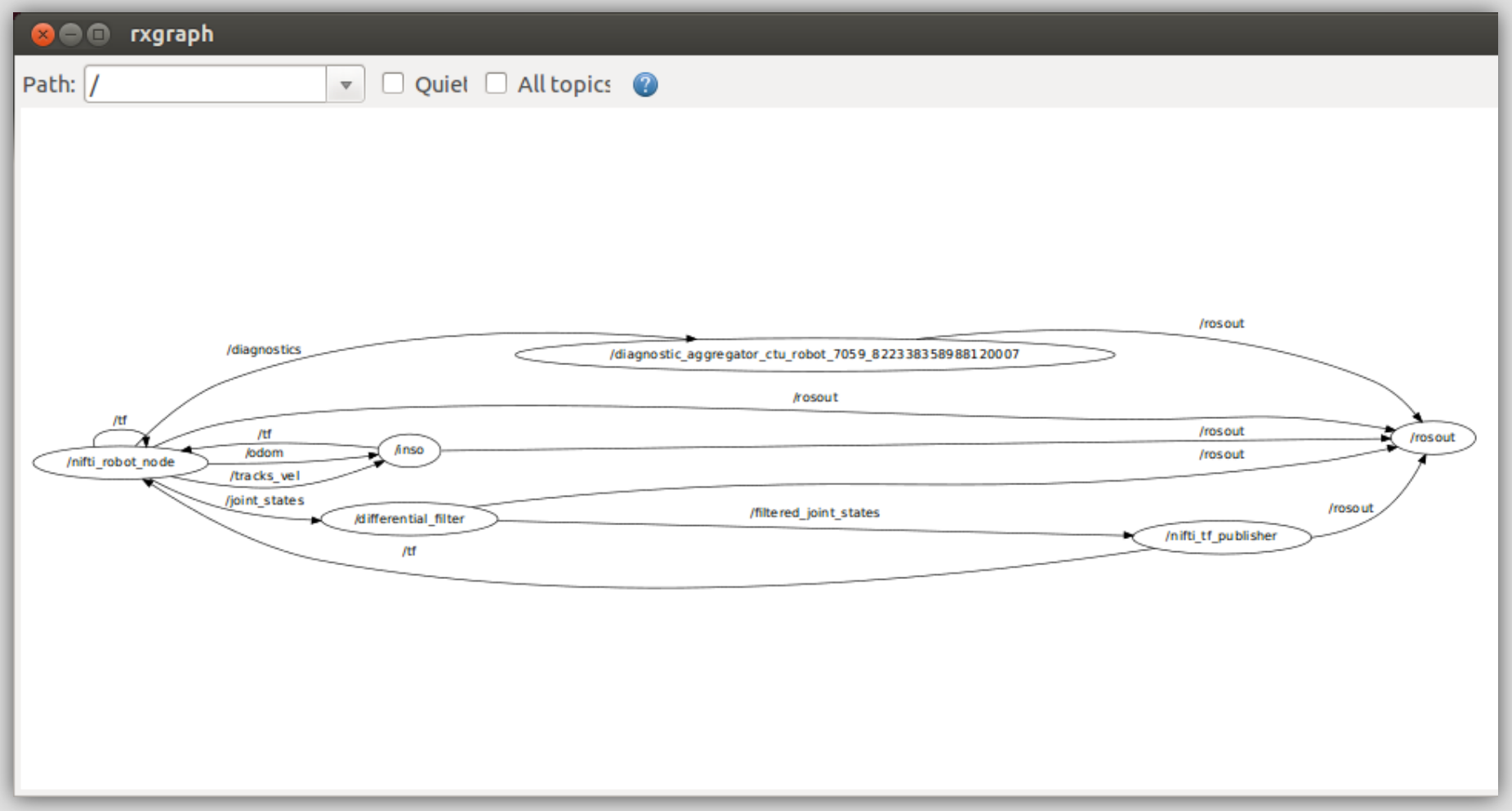
ROS Tools: IMAGEVIEW



>> rosrun image_view image_view image:=<image topic> [image transport type]

http://wiki.ros.org/image_view

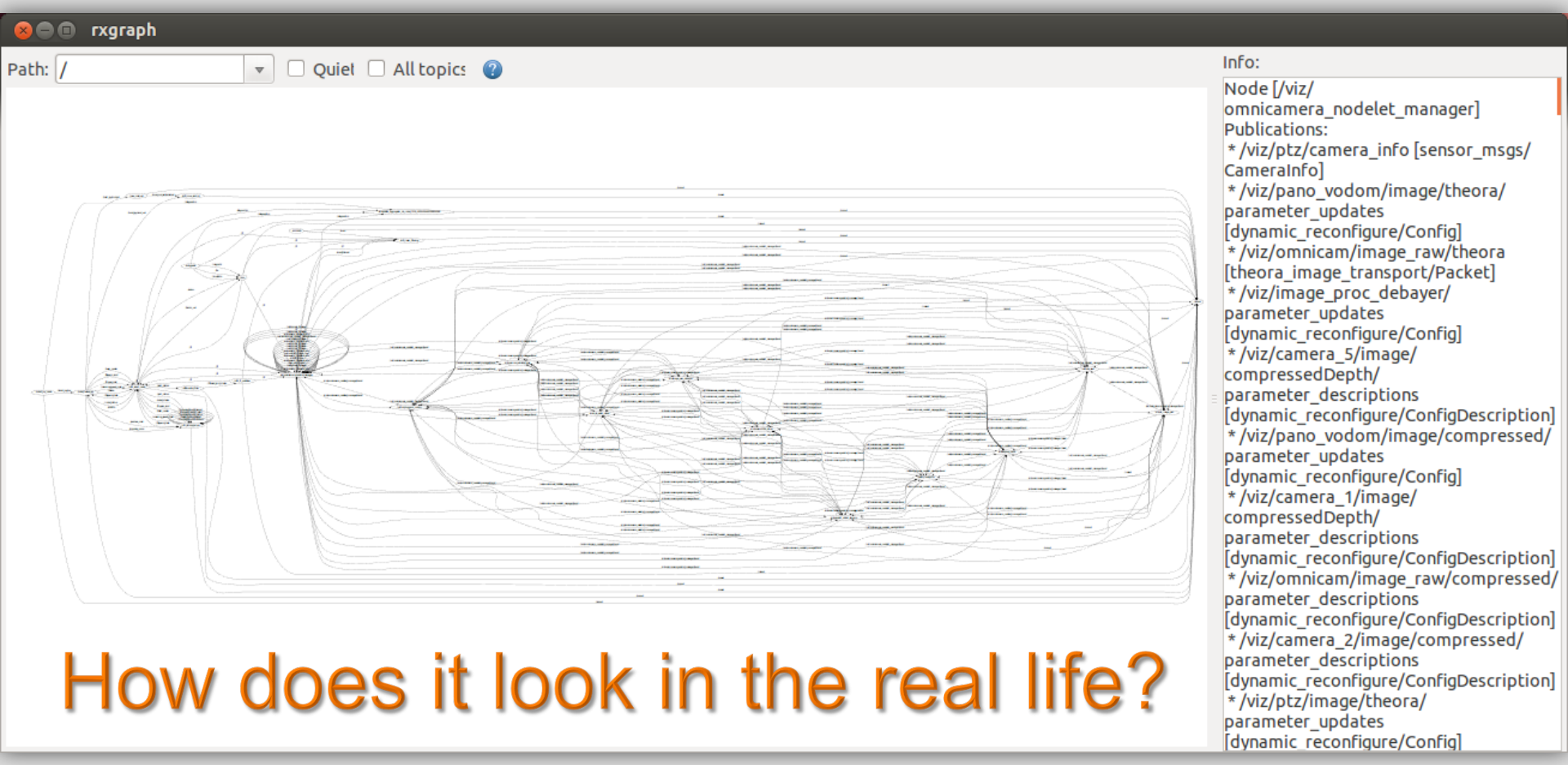
ROS Tools: RXGRAPH



>> rxgraph [options] // in newer version of ROS groovy+ is replaced by rqt_graph

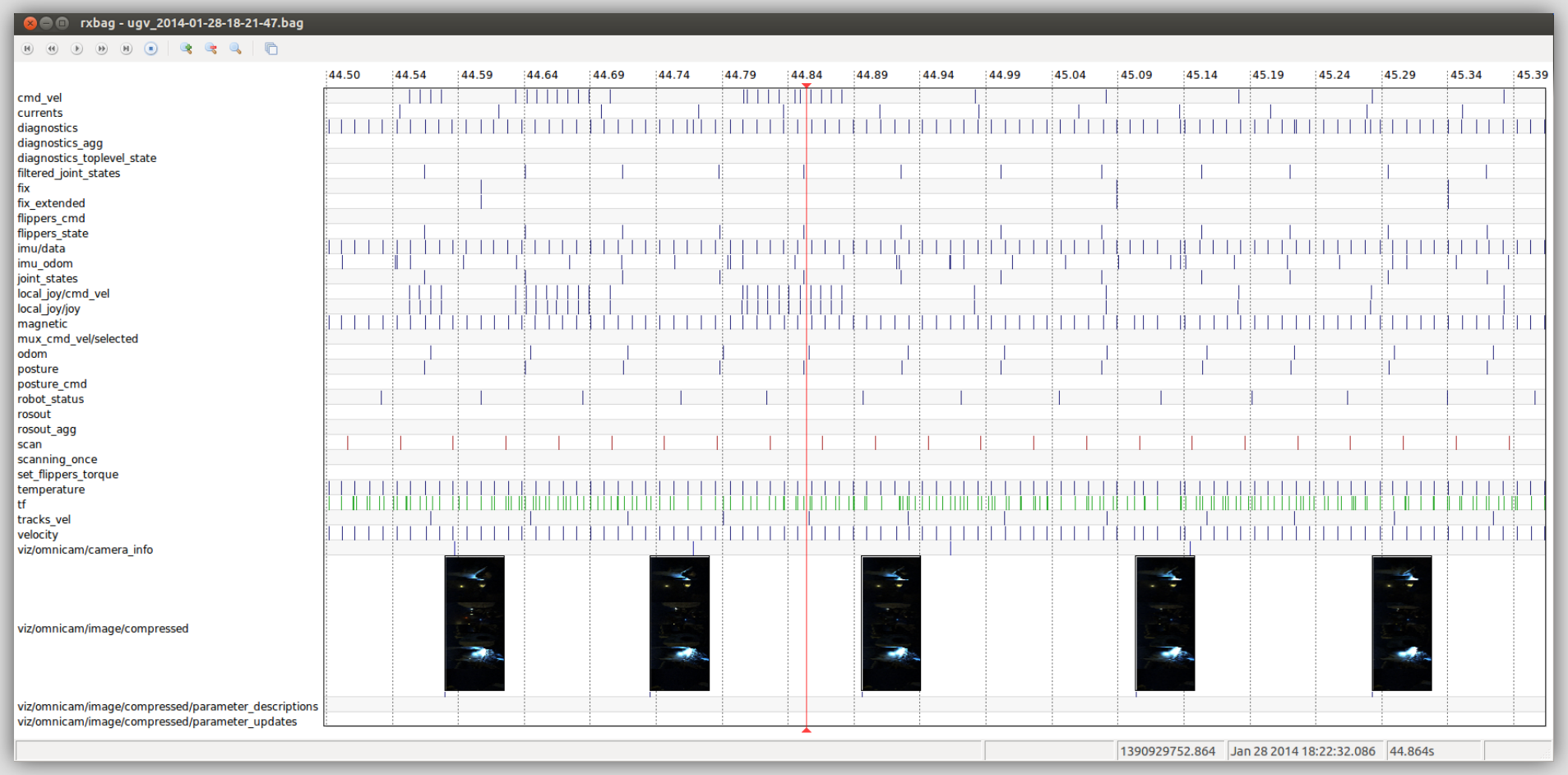
http://wiki.ros.org/rxgraph

ROS Tools: RXGRAPH



How does it look in the real life?

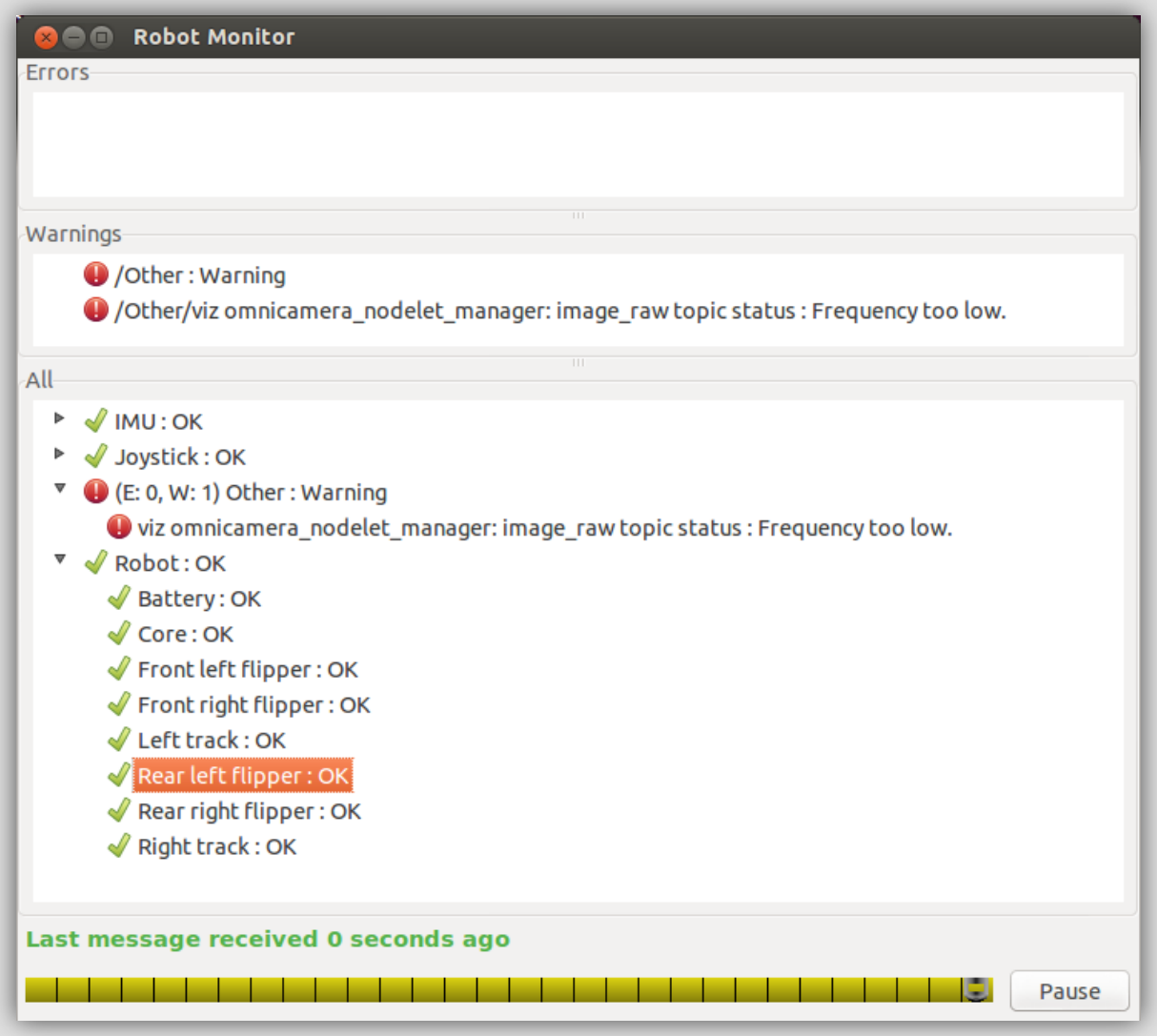
ROS Tools: RXBAG



>> rxbag name_of_bag.bag // in newer version of ROS groovy+ replaced by rqt_bag

http://wiki.ros.org/rxbag

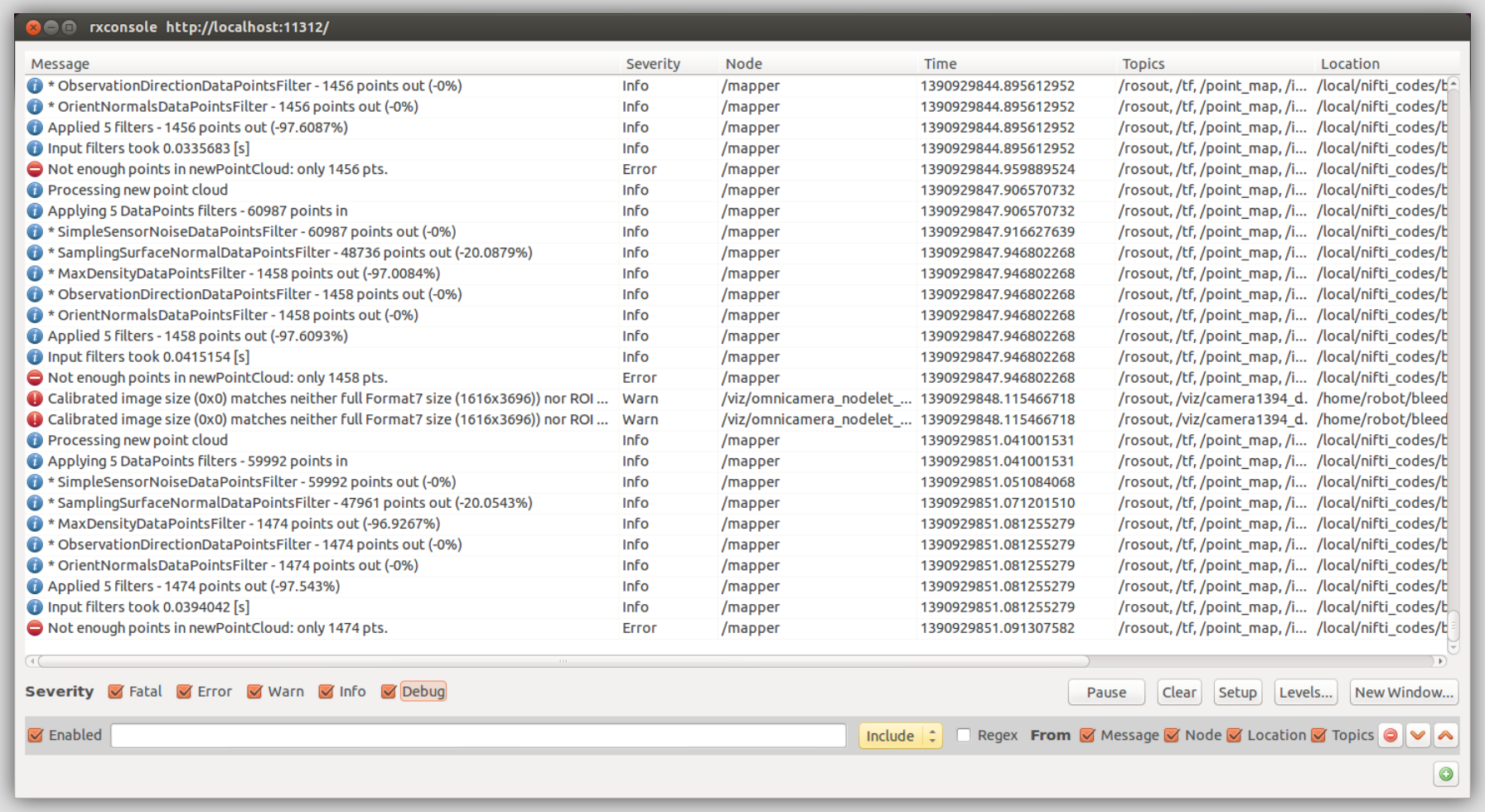
ROS Tools: ROBOT MONITOR



>> rosrun robot_monitor robot_monitor

http://wiki.ros.org/robot_monitor

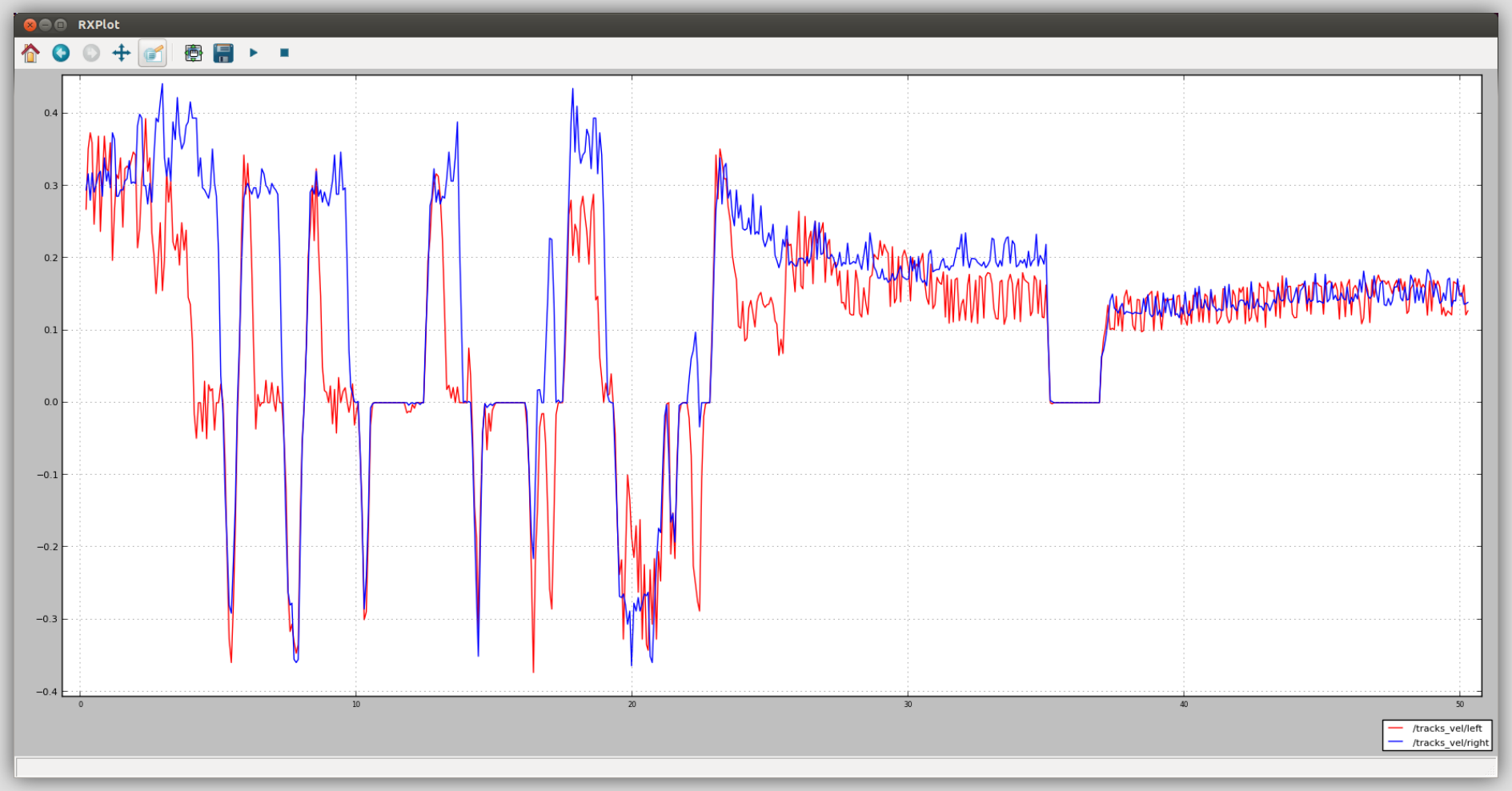
ROS Tools: RXCONSOLE



>> rxconsole // in newer version of ROS groovy+ is replaced by rqt_console

http://wiki.ros.org/rxconsole

ROS Tools: RXPLOT



>> rxplot /topic1/field1:field2:field3

http://wiki.ros.org/rxplot

ROS Tools: CHEATSHEET

ROS Cheat Sheet

Filesystem Command-line Tools

rospack/rosstack	A tool inspecting packages/stacks.
roscd	Changes directories to a package or stack.
rosls	Lists package or stack information.
roscreate-pkg	Creates a new ROS package.
roscreate-stack	Creates a new ROS stack.
rosdep	Installs ROS package system dependencies.
rosmake	Builds a ROS package.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
rxdeps	Displays package structure and dependencies.

Usage:

```
$ rospack find [package]
$ roscd [package[/subdir]]
$ rosls [package[/subdir]]
$ roscreate-pkg [package_name]
$ rosmake [package]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ rxdeps [options]
```

Common Command-line Tools

roscore

A collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate.

roscore is currently defined as:

- master
- parameter server
- rosout

Usage:

```
$ roscore
```

rosmsg/rossrv

rosmsg/rossrv displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg.
rosmsg users	Search for code using the msg.
rosmsg md5	Display the msg md5 sum.
rosmsg package	List all the messages in a package.
rosnode packages	List all the packages with messages.

Examples:

Display the Pose msg:

```
$ rosmsg show Pose
```

List the messages in nav_msgs:

```
$ rosmsg package nav_msgs
```

List the files using sensor_msgs/CameraInfo:

```
$ rosmsg users sensor_msgs/CameraInfo
```

rosrun

rosrun allows you to run an executable in an arbitrary package without having to cd (or roscd) there first.

Usage:

```
$ rosrun package executable
```

Example:

Run turtlesim:

```
$ rosrun turtlesim turtlesim_node
```

rosnode

Displays debugging information about ROS nodes, including publications, subscriptions and connections.

Commands:

rosnode ping	Test connectivity to node.
rosnode list	List active nodes.
rosnode info	Print information about a node.
rosnode machine	List nodes running on a particular machine.
rosnode kill	Kills a running node.

Examples:

Kill all nodes:

```
$ rosnode kill -a
```

List nodes on a machine:

```
$ rosnode machine aqy.local
```

Ping all nodes:

```
$ rosnode ping --all
```

roslaunch

Starts ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server.

Examples:

Launch on a different port:

```
$ roslaunch -p 1234 package filename.launch
```

Launch a file in a package:

```
$ roslaunch package filename.launch
```

Launch on the local nodes:

```
$ roslaunch --local package filename.launch
```

rostopic

A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Commands:

rostopic bw	Display bandwidth used by topic.
rostopic echo	Print messages to screen.
rostopic hz	Display publishing rate of topic.
rostopic list	Print information about active topics.
rostopic pub	Publish data to topic.
rostopic type	Print topic type.
rostopic find	Find topics by type.

Examples:

Publish hello at 10 Hz:

```
$ rostopic pub -r 10 /topic_name std_msgs/String hello
```

Clear the screen after each message is published:

```
$ rostopic echo -c /topic_name
```

Display messages that match a given Python expression:

```
$ rostopic echo --filter "m.data=='foo'" /topic_name
```

Pipe the output of rostopic to rosmsg to view the msg type:

```
$ rostopic type /topic_name | rosmsg show
```

rosparam

A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.

Commands:

rosparam set	Set a parameter.
rosparam get	Get a parameter.
rosparam load	Load parameters from a file.
rosparam dump	Dump parameters to a file.
rosparam delete	Delete a parameter.
rosparam list	List parameter names.

Examples:

List all the parameters in a namespace:

```
$ rosparam list /namespace
```

Setting a list with one as a string, integer, and float:

```
$ rosparam set /foo "['1', 1, 1.0]"
```

Dump only the parameters in a specific namespace to file:

```
$ rosparam dump dump.yaml /namespace
```

rosservice

A tool for listing and querying ROS services.

Commands:

rosservice list	Print information about active services.
rosservice node	Print the name of the node providing a service.
rosservice call	Call the service with the given args.
rosservice args	List the arguments of a service.
rosservice type	Print the service type.
rosservice uri	Print the service ROSRPC uri.
rosservice find	Find services by service type.

Examples:

Call a service from the command-line:

```
$ rosservice call /add_two_ints 1 2
```

Pipe the output of rosservice to rossrv to view the srv type:

```
$ rosservice type add_two_ints | rossrv show
```

Display all services of a particular type:

```
$ rosservice find rospy_tutorials/AddTwoInts
```

>> rxplot /topic1/field1:field2:field3

http://wiki.ros.org/rxplot