RQT Plugins

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
$ rqt # RQT graphical user interface
```

Available RQT plugins
http://wiki.ros.org/rqt/Plugins

- **Node Graph**
- **Topic Monitor**
- **Massage publisher**
- **Service caller**
- **rqt_plot**
- **Launch (experimental)**
- **Logger Level**
- **Console**
- **Tf tree (tf_echo, rviz)**

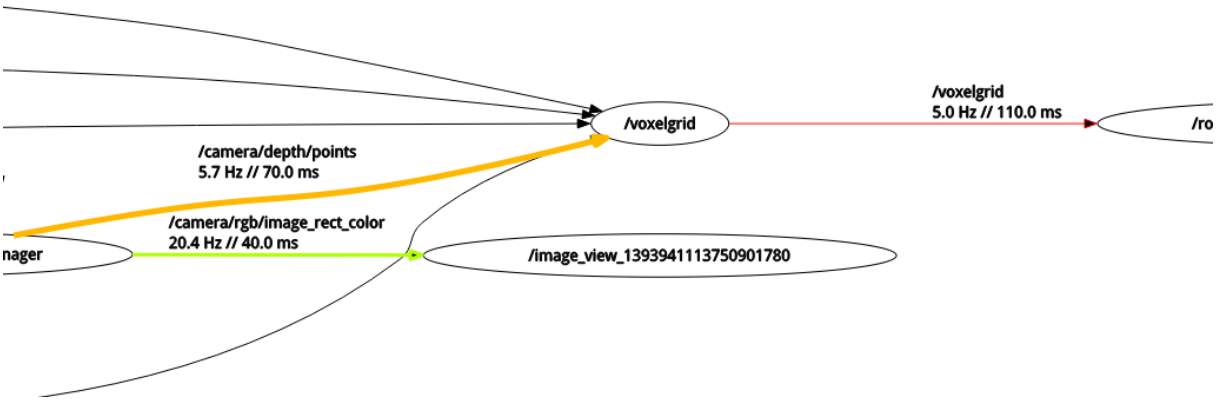
Dynamic Reconfigure
- Vortrag Rodion Marynych, 11.1.2016

Node Graph ($ rqt_graph)

- Choose Grape type
  - „Nodes Only“
  - “Nodes/Topics (active)”
  - “Nodes/Topics (all)”
- Namespace Filter
- Topic Filter
- Group Namespaces (/turtle1/...)
- Highlight
  - ROS nodes (here blue and green)
  - Topics (here red)

Topic Statistics

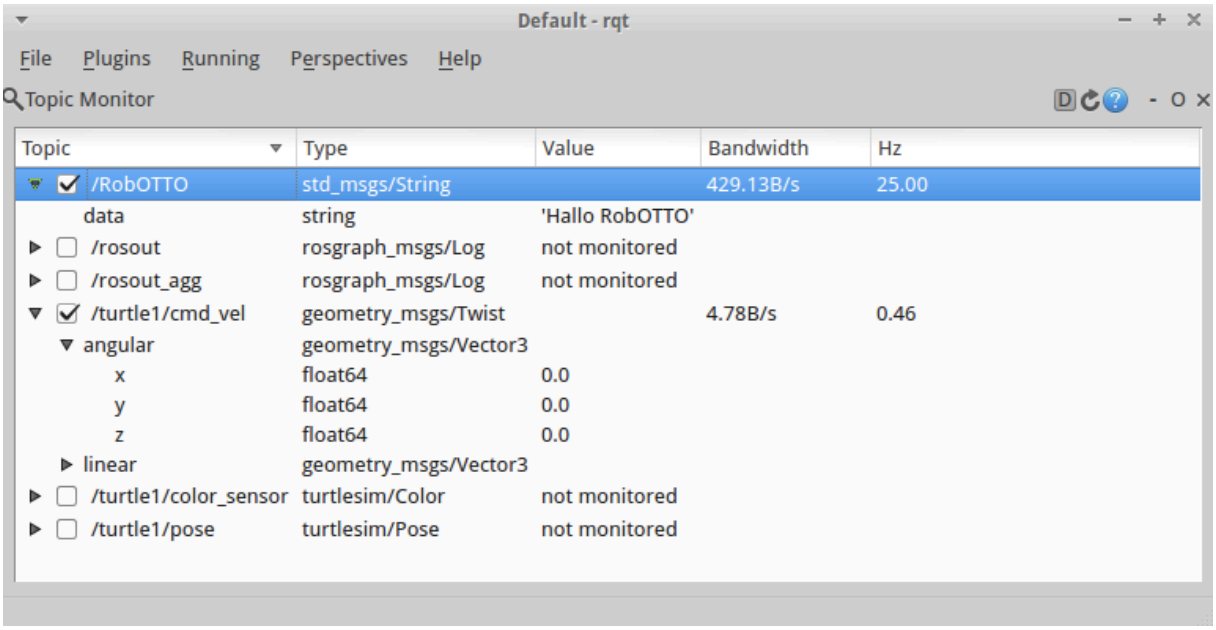
```
$ rosparam set enable_statistics true

$ rostopic pub /RobOTTO std_msgs/String "data: 'Hallo
  RobOTTO' " -r 25

$ rostopic bw /RobOTTO
```



Topic Monitor ($ rosrun rqt_topic rqt_topic)

```
$ rostopic pub /RobOTTO std_msgs/String "data: 'Hallo
  RobOTTO' " -r 25
```

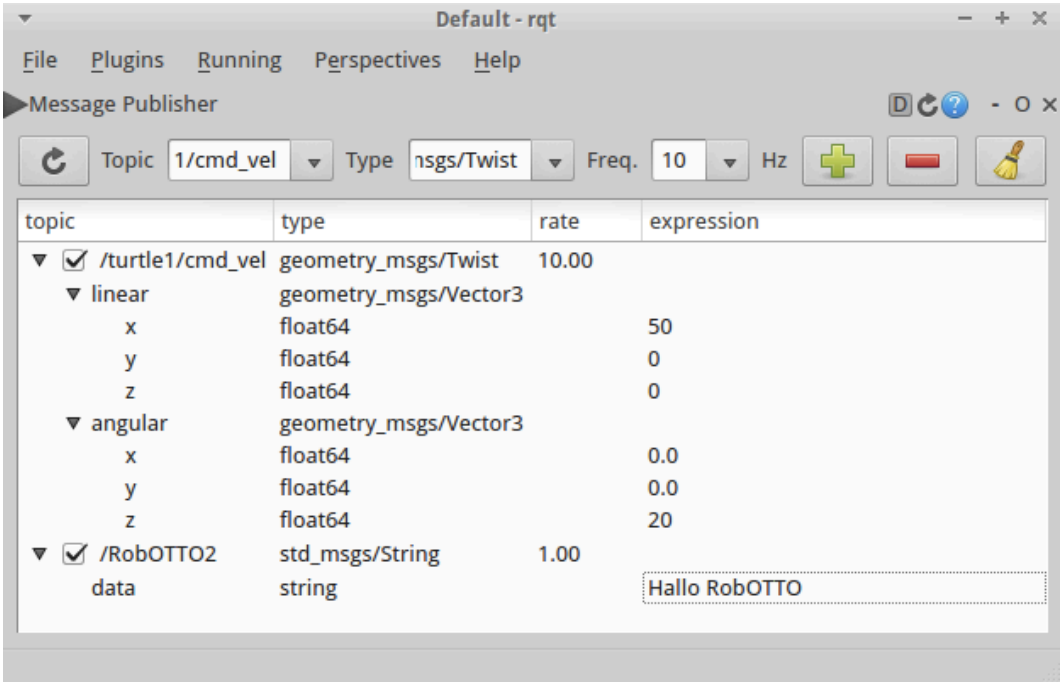
- topic monitor list
  - Topic
  - Type
  - Value
  - Bandwidth (Bytes per second)
  - Frequency



Massage publisher ($ rosrun rqt_publisher rqt_publisher)

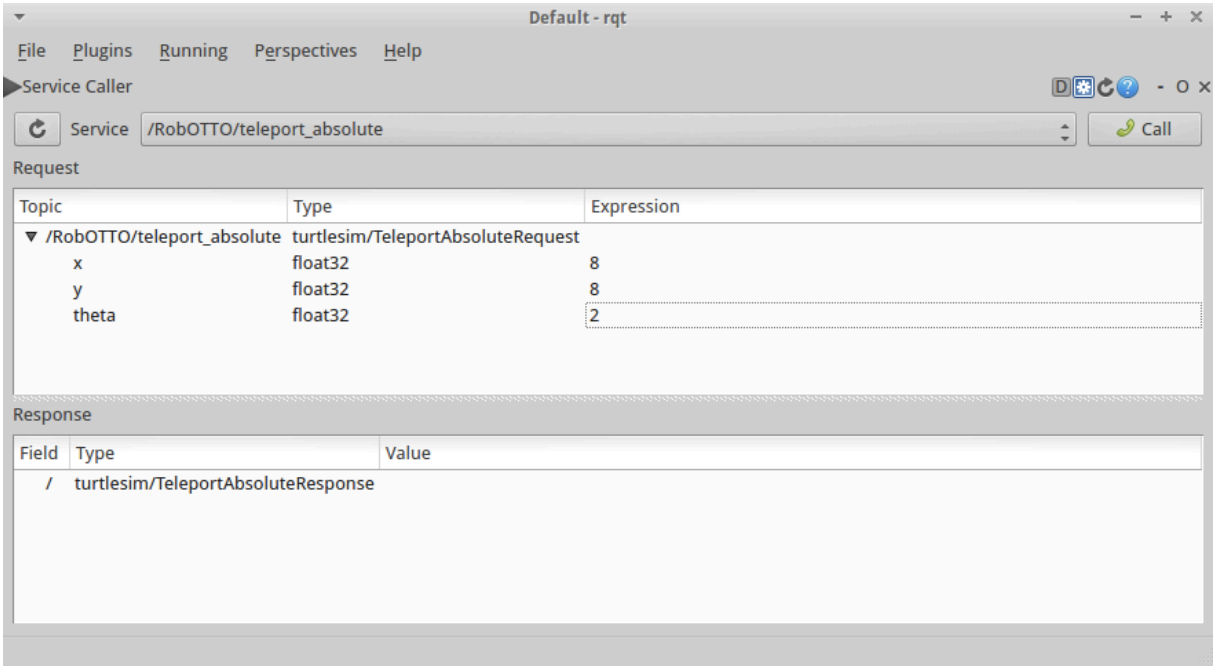
```
$ rostopic echo /RobOTTO2
```

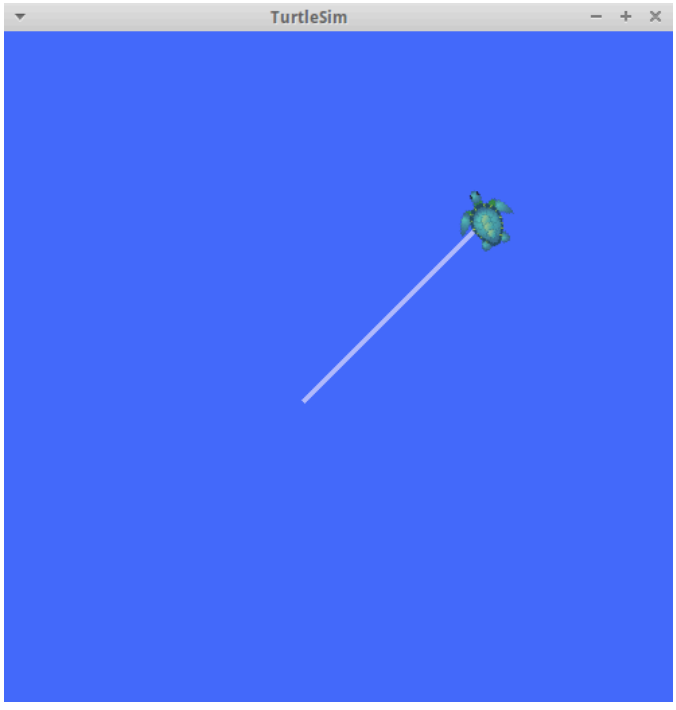
- Create a Publisher
  - Topic name to publish on
  - Message Type to publish
  - Frequency for periodic publishers in Hz (set to 0 for manual publishing)
  - Add new publisher
  - Remove selected publisher
  - Clear all publishers



Service caller ($ rosrun rqt_service_caller rqt_service_caller)

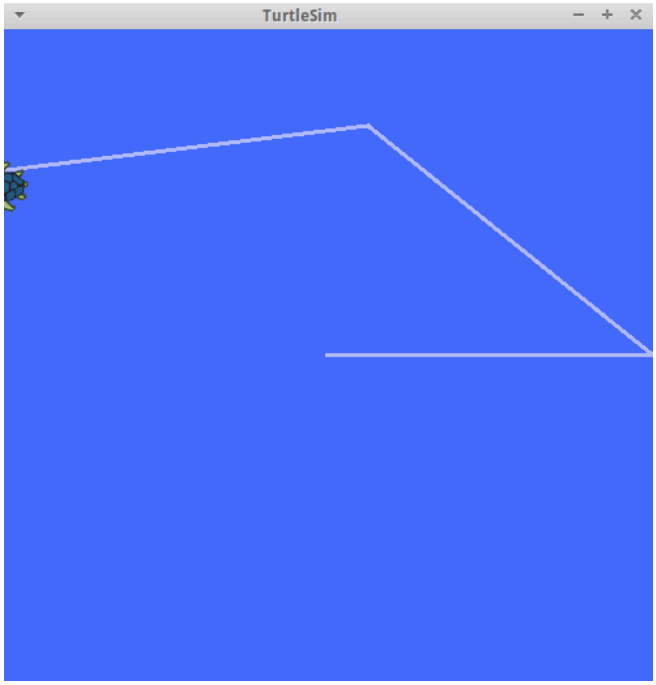
- Call Service
  - Select service from drop down menu
  - Change the expressions in the request
  - Call selected service
  - Get the response in the lower window





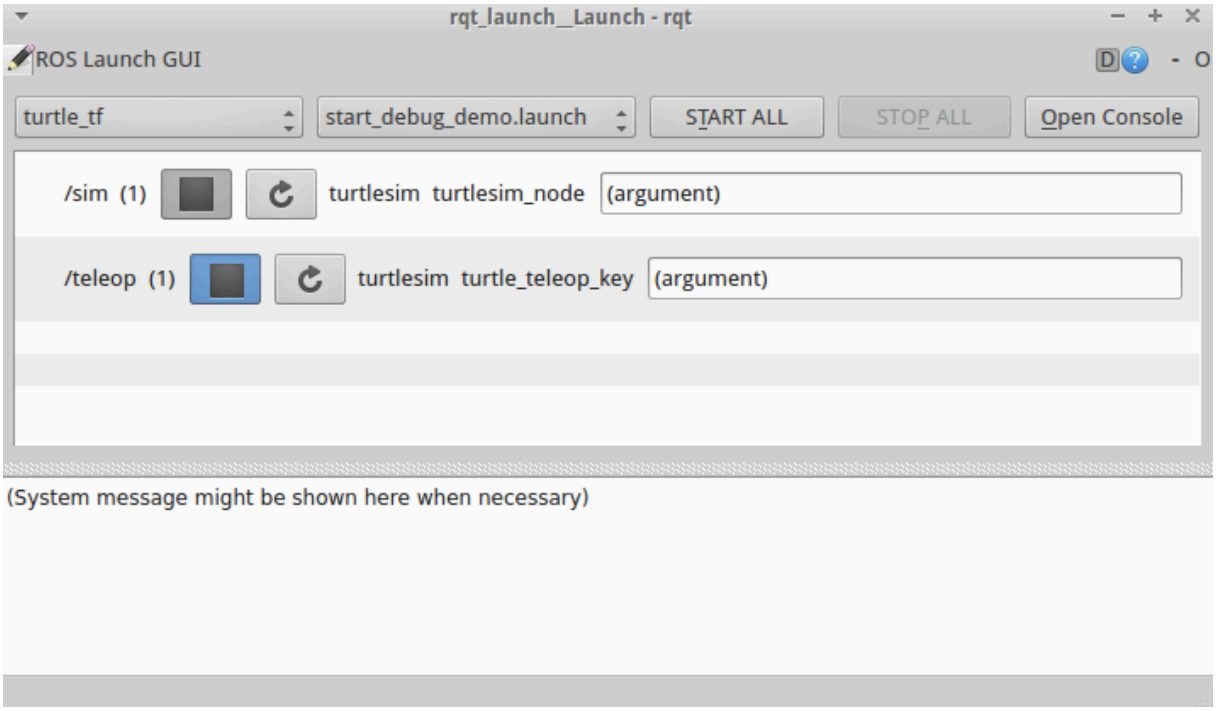
rqt_plot ($ rosrun rqt_plot rqt_plot)

- select the topic that is to be plotted
- autoscroll
- pause plot
- clear plot
- Pan axes with left mouse, zoom with right
- save the figure as .png
- edit curves and axes parameters



Launch (experimental) ($ rosrun rqt_launch rqt_launch)

- select a .launch files on the local file system
- See all nodes defined in the .launch file
- Run and stop nodes individually
- Start and stop all nodes at once
- Add an argument value before the node gets started



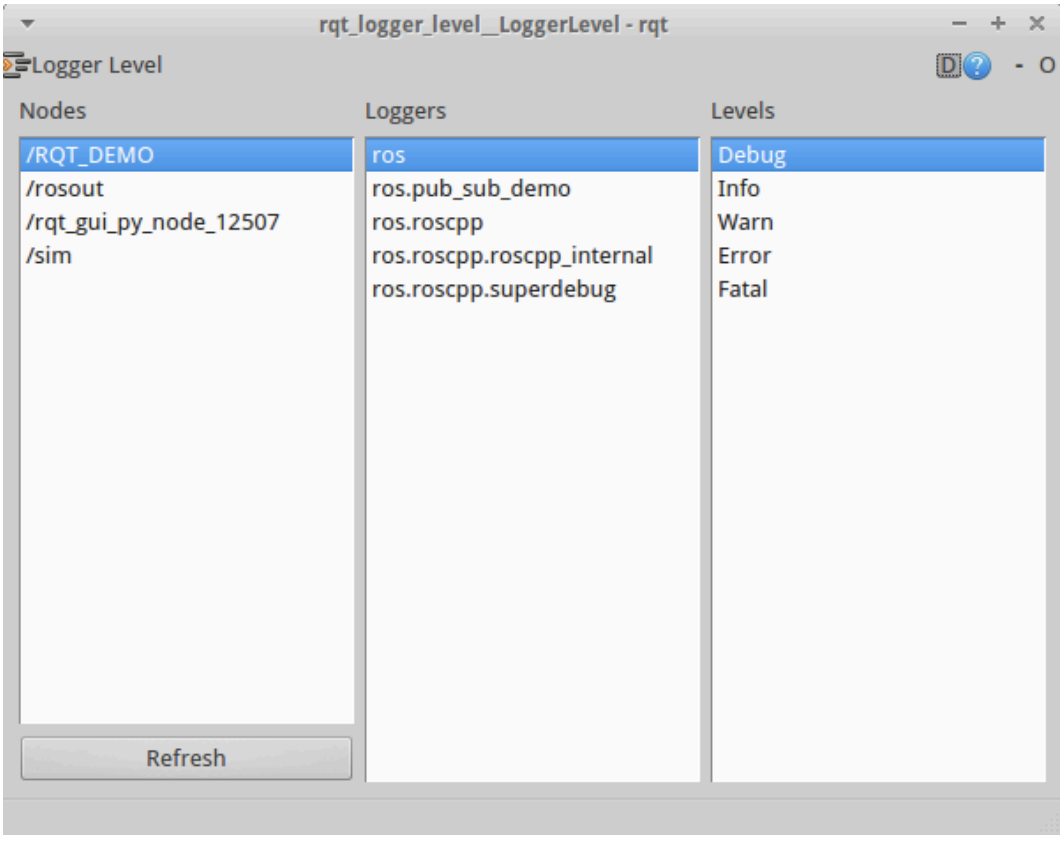
Logger Level (`$ rosrun rqt_logger_level rqt_logger_level`)

- `rqt_logger_level` is an application for adjusting the logger level of ros nodes.

```
$ rosrun pub_sub_demo node1
```

inside the node

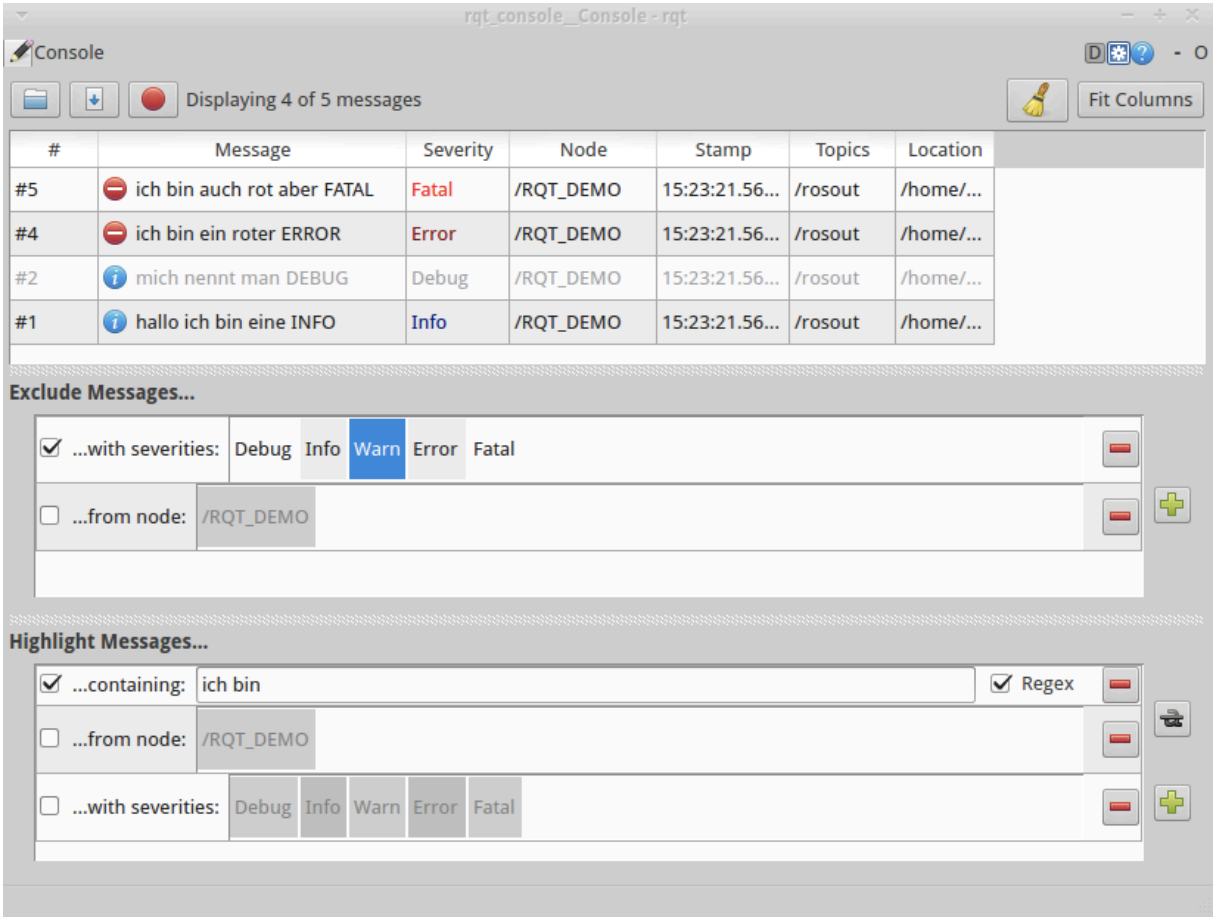
```
ROS_INFO("hallo ich bin eine INFO");
ROS_DEBUG("mich nennt man DEBUG");
ROS_WARN("ich WARN dich!");
ROS_ERROR("ich bin ein roter ERROR");
ROS_FATAL("ich bin auch rot aber FATAL");
```



Console ($ rqt_console)

Logging levels are prioritized in the following order:
- Fatal
- Error
- Warn
- Info
- Debug

- Pause/Resume the message stream
- Configuration
  - Logger Level
- Exclude Messages
  - ...containing
  - ...from time range
  - ...from topic
  - Custom
- Highlight Messages
  - ...containing
  - ...from time range
  - ...from topic
  - Custom

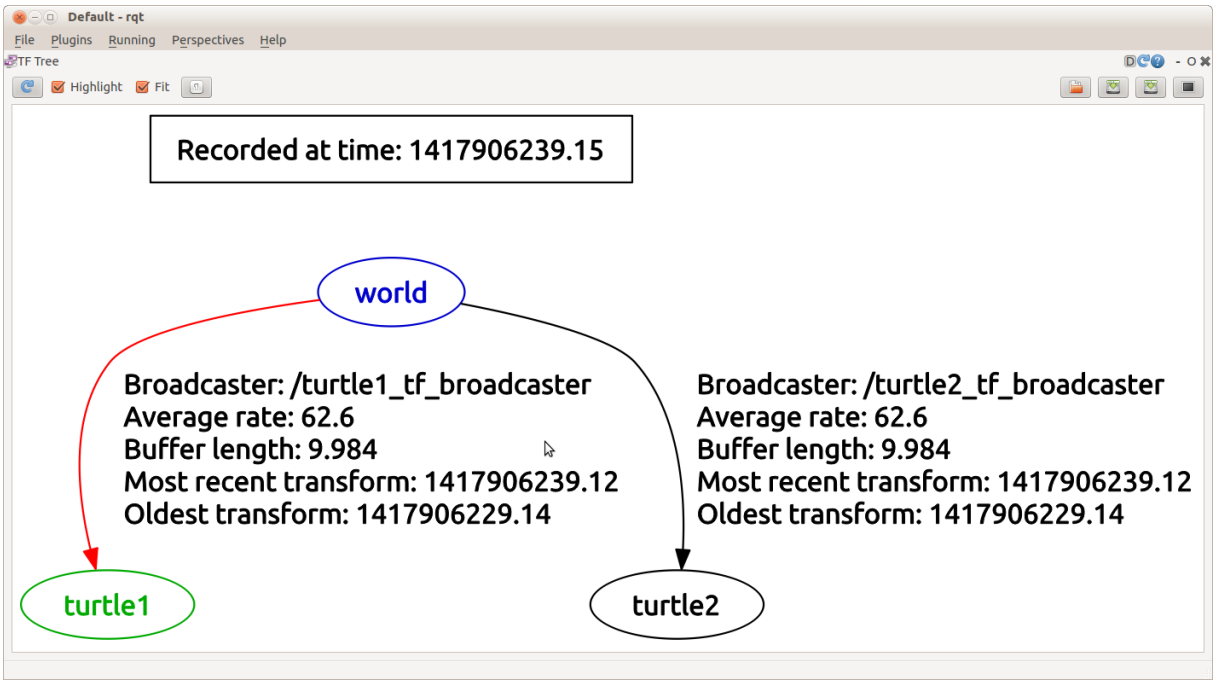


Tf tree ($ rosrun rqt_tf_tree rqt_tf_tree)

```
$ roslaunch turtle_tf turtle_tf_demo.launch
```

Ros Wiki:

> This demo is using the tf library to create three coordinate frames: a world frame, a turtle1 frame, and a turtle2 frame. This tutorial uses a tf broadcaster to publish the turtle coordinate frames and a tf listener to compute the difference in the turtle frames and move one turtle to follow the other.



We can see that "world" is the parent of the "turtle1" and "turtle2" frames and some more informations for debugging purposes.

tf_echo (not a rqt plugin)

```
$ rosrun tf tf_echo world turtle1

$ rosrun tf tf_echo turtle1 world
```

rviz (rosrun rviz rviz)

```
$ rosrun rviz rviz -d `rospack find
turtle_tf`/rviz/turtle_rviz.rviz
```