The Robot Operating System

Day 2 Tutorial I – ROS tools

Dr. Murilo Fernandes Martins

Department of Electrical Engineering
Centro Universitário da FEI

28 January 2014

Outline

A Quick Overview of Graph Concepts

From last class...

- Nodes: executable that uses ROS to communicate with other nodes
- Messages: ROS data type used when subscribing or publishing to a topic
- Topics: nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages
- Master: name service for ROS (i.e., helps nodes find each other)

A Quick Overview of Graph Concepts

From last class...

- Nodes: executable that uses ROS to communicate with other nodes
- Messages: ROS data type used when subscribing or publishing to a topic
- Topics: nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages
- Master: name service for ROS (i.e., helps nodes find each other)

More on Graph Concepts

- rosout: ROS equivalent of stdout/stderr
- roscore: Master + rosout + parameter server

Running the ROS Master

ROS Master

- roscore is the command used to start the ROS Master
- ROS Master must be started before any other node

roscore http://jupiter:11311/

```
murilo@jupiter:~$ roscore
... logging to /home/murilo/.ros/log/33c038c6-8467-11e3-abf3-5cf9ddee07d1/roslaunch-jupiter-31607.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://jupiter:36392/
ros_comm version 1.9.53


SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion

NODES

auto-starting new master
process[master]: started with pid [31621]
ROS_MASTER_URI=http://jupiter:11311/

setting /run_id to 33c038c6-8467-11e3-abf3-5cf9ddee07d1
process[rosout-1]: started with pid [31634]
started core service [/rosout]
```

Navigating the ROS filesystem

Filesystem tools

- code and executables are spread across many ROS packages
- navigating with commands such as cd and ls can be very tedious
- ROS provides command line utilities to help us

rosbash – shell commands for using ROS with Bash

- roscd: change directory to package path
- rospd: pushd equivalent of roscd
- rosd: lists directories in the directory-stack
- rosls: list files of a ROS package
- rosed: edit a file in a package
- roscp: copy a file from a package
- rosrun: run executables of a ros package

Navigating the ROS filesystem

Using rospack

- rospack allows to get information about packages
- it is possible to:
  - list packages currently available
  - find the path of packages
  - lists dependencies of packages
  - and more...
- rospack find turtlesim

```
murilo@jupiter: ~
murilo@jupiter:~$ rospack find turtlesim
/opt/ros/hydro/share/turtlesim
murilo@jupiter:~$
```

Navigating the ROS filesystem

Using roscd

- simple usage: roscd [package_name[/subdir]]
- allow changing directories directly into a package
- packages must be within directories listed in ROS_PACKAGE_PATH
- executing roscd without specifying a package changes directory to ROS_WORKSPACE (if set)

```
murilo@jupiter: /opt/ros/hydro/share/turtlesim
murilo@jupiter:~$ pwd
/home/murilo
murilo@jupiter:~$ roscd turtlesim
murilo@jupiter:/opt/ros/hydro/share/turtlesim$ pwd
/opt/ros/hydro/share/turtlesim
murilo@jupiter:/opt/ros/hydro/share/turtlesim$
```

Navigating the ROS filesystem

Using rosls

- simple usage: rosls [package_name[/subdir]]
- allow listing the contents of a package by name rather than by absolute path
- packages must be within directories listed in ROS_PACKAGE_PATH

```
murilo@jupiter: ~
murilo@jupiter:~$ rosls turtlesim
cmake  images  msg  package.xml  srv
murilo@jupiter:~$
```

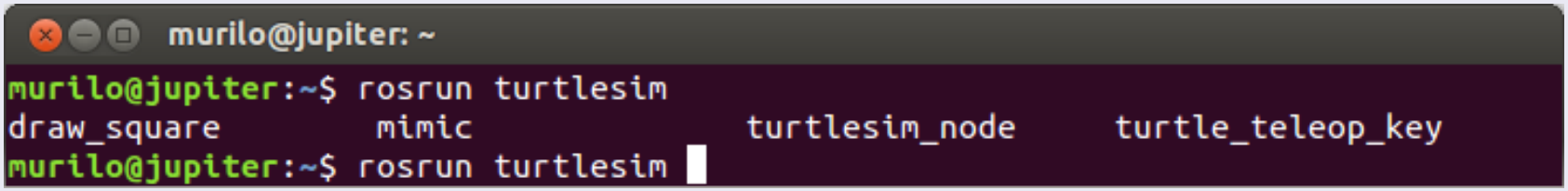
A note about command-line completion

Tab completion

- common feature of command line interpreters
- fills in partially typed commands
- ROS tools support Tab completion!

Hands on...

```
$ roscd tur<<<press TAB key once>>>
$ roscd turtle<<<press TAB key twice>>>
$ rospack find turtles<<<press TAB key once>>>
$ rosrun turtlesim <<<press TAB key twice>>>
```



Understanding ROS nodes

What happened when we ran *roscore*?

- rosnode: command-line tool for printing information about ROS Nodes which are currently running
- rosnode list: list active nodes

```
murilo@jupiter: ~
murilo@jupiter:~$ rosnode
rosnode is a command-line tool for printing information about ROS Nodes.

Commands:
        rosnode ping    test connectivity to node
        rosnode list    list active nodes
        rosnode info    print information about node
        rosnode machine list nodes running on a particular machine or list machines
        rosnode kill    kill a running node
        rosnode cleanup purge registration information of unreachable nodes

Type rosnode <command> -h for more detailed usage, e.g. 'rosnode ping -h'

murilo@jupiter:~$ rosnode list
/rosout
murilo@jupiter:~$
```

Understanding ROS nodes

What happened when we ran *roscore*?

- rosnode: command-line tool for printing information about ROS Nodes which are currently running
- rosnode info /rosout: print information about node */rosout*

```
murilo@jupiter: ~
murilo@jupiter:~$ rosnode info /rosout
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/set_logger_level
 * /rosout/get_loggers


contacting node http://jupiter:53593/ ...
Pid: 13087

murilo@jupiter:~$
```

Understanding ROS nodes

Bringing up ROS nodes

- rosrun: allow the use of package name to directly run a node within such package
- usage: rosrun [package_name] [node_name]
- example: rosrun turtlesim turtlesim_node

```
murilo@jupiter: ~
murilo@jupiter:~$ rosrun turtlesim turtlesim_node
[ INFO] [1390408364.307023409]: Starting turtlesim with node name /turtlesim
[ INFO] [1390408364.312821073]: Spawning turtle [turtle1] at x=[5.544445], y=[5.
544445], theta=[0.000000]
```

Hands on. . . nodes

Open a terminal and run *roscore*

```
$ roscore
```

Hands on. . . nodes

Open a terminal and run *roscore*

```
$ roscore
```

Open a new terminal and run *turtlesim*

```
$ rosrun turtlesim turtlesim_node
```

Hands on. . . nodes

Open a terminal and run *roscore*

```
$ roscore
```

Open a new terminal and run *turtlesim*

```
$ rosrun turtlesim turtlesim_node
```

In another terminal, analyse nodes currently running

```
$ rosnode list
$ rosnode info /turtlesim
$ rosnode ping /turtlesim
$ rosrun turtlesim turtle_teleop_key
```

Understanding ROS topics

Using rostopic

- rostopic: command-line tool for printing information about ROS Topics currently advertised

```
murilo@jupiter: ~
murilo@jupiter:~$ rostopic
rostopic is a command-line tool for printing information about ROS Topics.

Commands:
        rostopic bw     display bandwidth used by topic
        rostopic echo   print messages to screen
        rostopic find   find topics by type
        rostopic hz     display publishing rate of topic
        rostopic info   print information about active topic
        rostopic list   list active topics
        rostopic pub    publish data to topic
        rostopic type   print topic type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'

murilo@jupiter:~$
```

Understanding ROS topics

Using rostopic

- rostopic: command-line tool for printing information about ROS Topics currently advertised
- rostopic list: list advertised topics
- rostopic info /turtle1/cmd_vel: print information about the topic

```
murilo@jupiter: ~
murilo@jupiter:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
murilo@jupiter:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://jupiter:59211/)

Subscribers:
 * /turtlesim (http://jupiter:50849/)
```

Hands on. . . topics

Investigate the details of topics

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
$ rostopic list -h
$ rostopic info /turtle1/pose
$ rostopic hz   /turtle1/pose
$ rostopic bw   /turtle1/pose
$ rostopic echo /turtle1/pose
```

Understanding ROS messages

Using rosmsg

- rosmsg: command-line tool for displaying information about ROS Message types

```
murilo@jupiter: ~
murilo@jupiter:~$ rosmsg
rosmsg is a command-line tool for displaying information about ROS Message types.

Commands:
        rosmsg show     Show message description
        rosmsg list     List all messages
        rosmsg md5      Display message md5sum
        rosmsg package  List messages in a package
        rosmsg packages List packages that contain messages

Type rosmsg <command> -h for more detailed usage

murilo@jupiter:~$
```

Hands on. . . messages

Investigate message types of topics currently advertised

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
$ rostopic info /turtle1/pose
$ rosmsg show geometry_msgs/Twist
```

murilo@jupiter: ~

```
murilo@jupiter:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z

murilo@jupiter:~$
```

Hands on. . . messages

What if we wanted to *publish* messages from the console? Easy!

- given the topic onto which message will be published. . .
- and the message type used by the topic
- use rostopic pub to publish a message

```
murilo@jupiter: ~
murilo@jupiter:~$ rostopic pub -h
Usage: rostopic pub /topic type [args...]

Options:
  -h, --help            show this help message and exit
  -v                    print verbose output
  -r RATE, --rate=RATE  publishing rate (hz).  For -f and stdin input, this
                        defaults to 10.  Otherwise it is not set.
  -1, --once            publish one message and exit
  -f FILE, --file=FILE  read args from YAML file (Bagy)
  -l, --latch           enable latching for -f, -r and piped input.  This
                        latches the first message.
murilo@jupiter:~$
```

Hands on. . . messages

Publishing messages using rostopic

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rostopic info /turtle1/cmd_vel
$ rosmsg show geometry_msgs/Twist
$ rostopic pub -r 10 /turtle1/cmd_vel
  geometry_msgs/Twist --
  '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

murilo@jupiter: ~

```
murilo@jupiter:~$ rostopic pub -r 10 /turtle1/cmd_vel geometry_msgs/Twist --
'[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

Understanding ROS services

Using rosmsg

- rossrv: command-line tool for displaying information about ROS Service types

```
murilo@jupiter: ~
murilo@jupiter:~$ rossrv
rossrv is a command-line tool for displaying information about ROS Service types.

Commands:
        rossrv show     Show service description
        rossrv list     List all services
        rossrv md5      Display service md5sum
        rossrv package  List services in a package
        rossrv packages List packages that contain services

Type rossrv <command> -h for more detailed usage

murilo@jupiter:~$
```

Hands on. . . services

Information about services provided by a package

```
$ roscore
$ rossrv package turtlesim
$ rossrv show turtlesim/Spawn
```

murilo@jupiter: ~

```
murilo@jupiter:~$ rossrv package turtlesim
turtlesim/Kill
turtlesim/SetPen
turtlesim/Spawn
turtlesim/TeleportAbsolute
turtlesim/TeleportRelative
murilo@jupiter:~$ rossrv show turtlesim/Spawn
float32 x
float32 y
float32 theta
string name
---
string name

murilo@jupiter:~$
```

Hands on. . . services

Information about services currently active

```
$ roscore; rosrun turtlesim turtlesim_node
$ rosservice list
$ rosservice info <service name>
```

murilo@jupiter: ~

```
murilo@jupiter:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
murilo@jupiter:~$
```

murilo@jupiter: ~

```
murilo@jupiter:~$ rosservice info /clear
Node: /turtlesim
URI: rosrpc://jupiter:40717
Type: std_srvs/Empty
Args:
murilo@jupiter:~$ rosservice info /reset
Node: /turtlesim
URI: rosrpc://jupiter:40717
Type: std_srvs/Empty
Args:
murilo@jupiter:~$ rosservice info /spawn
Node: /turtlesim
URI: rosrpc://jupiter:40717
Type: turtlesim/Spawn
Args: x y theta name
murilo@jupiter:~$
```

Hands on. . . services

Calling services from the console is also a piece of cake!

- given the service currently active. . .
- and the required arguments
- use rosservice call to call a service

```
murilo@jupiter: ~
murilo@jupiter:~$ rosservice call /spawn 1.0 2.0 0.0 turtle2
name: turtle2
murilo@jupiter:~$ rosservice call /kill turtle1

murilo@jupiter:~$ rosservice call /reset

murilo@jupiter:~$
```

Understanding ROS parameters

Parameter Server

- shared, multi-variate dictionary accessible via network APIs
- nodes use this server to store and retrieve parameters at runtime
- not designed for high performance; better used for static data
- globally viewable, runs inside ROS Master (accessible via XMLRPC)

Understanding ROS parameters

Parameter Server

- shared, multi-variate dictionary accessible via network APIs
- nodes use this server to store and retrieve parameters at runtime
- not designed for high performance; better used for static data
- globally viewable, runs inside ROS Master (accessible via XMLRPC)

rosparam

- command-line tool for getting, setting, and deleting parameters from the ROS Parameter Server

```
murilo@jupiter: ~
Commands:
        rosparam set    set parameter
        rosparam get    get parameter
        rosparam load   load parameters from file
        rosparam dump   dump parameters to file
        rosparam delete delete parameter
        rosparam list   list parameter names
```

Understanding ROS parameters

Using rosparam

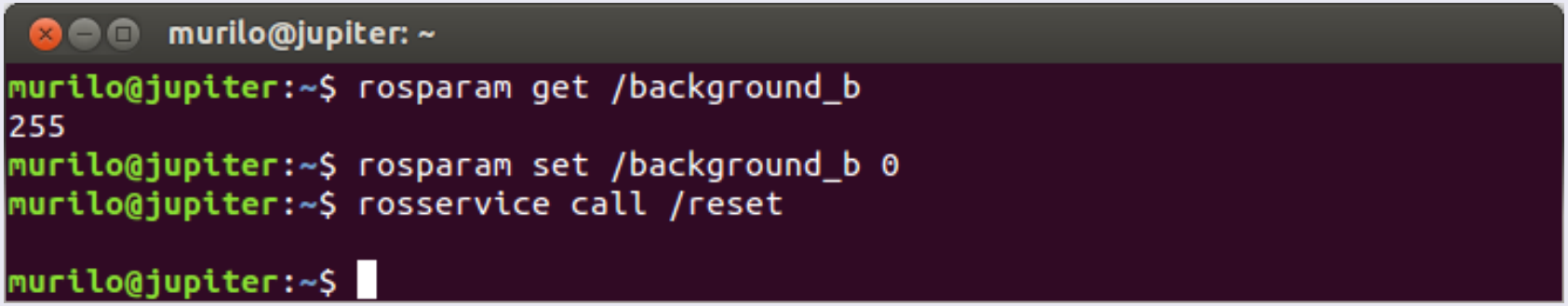
- rosparam list: list parameters currently stored
- rosparam get [param_name]: get parameter value
- rosparam set [param_name]: set parameter value

```
murilo@jupiter: ~
murilo@jupiter:~$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_jupiter__38073
/rosversion
/run_id
murilo@jupiter:~$
```

Hands on. . . parameters

Getting/setting parameters is rather simple!

```
$ roscore; rosrun turtlesim turtlesim_node
$ rosparam list
$ rosparam get /background_b
$ rosparam set /background_b 0
$ rosservice call /reset
```



Quick review

So far...

rospack	=	ros + package
roscd	=	ros + cd (change directory)
rosls	=	ros + ls (list directory)
roscore	=	ros master + rosout + parameter server
rosnode	=	ros + node
rosrun	=	ros + run
rostopic	=	ros + topic
rosmsg	=	ros + message
rossrv	=	ros + service
rosservice	=	ros + service
rosparam	=	ros + parameter

Rqt – Qt-based framework GUI for ROS

rqt_gui widget

- allow multiple rqt widgets to be docked in a single window
- various GUI tools implemented as plugins:
  - Introspection (e.g., Node Graph)
  - Logging (e.g., Bag, Console)
  - Services (e.g., Service Caller)
  - Topics (e.g., Message Publisher)
  - Visualisation (e.g., Plot)

Rqt – Qt-based framework GUI for ROS

rqt_gui widget

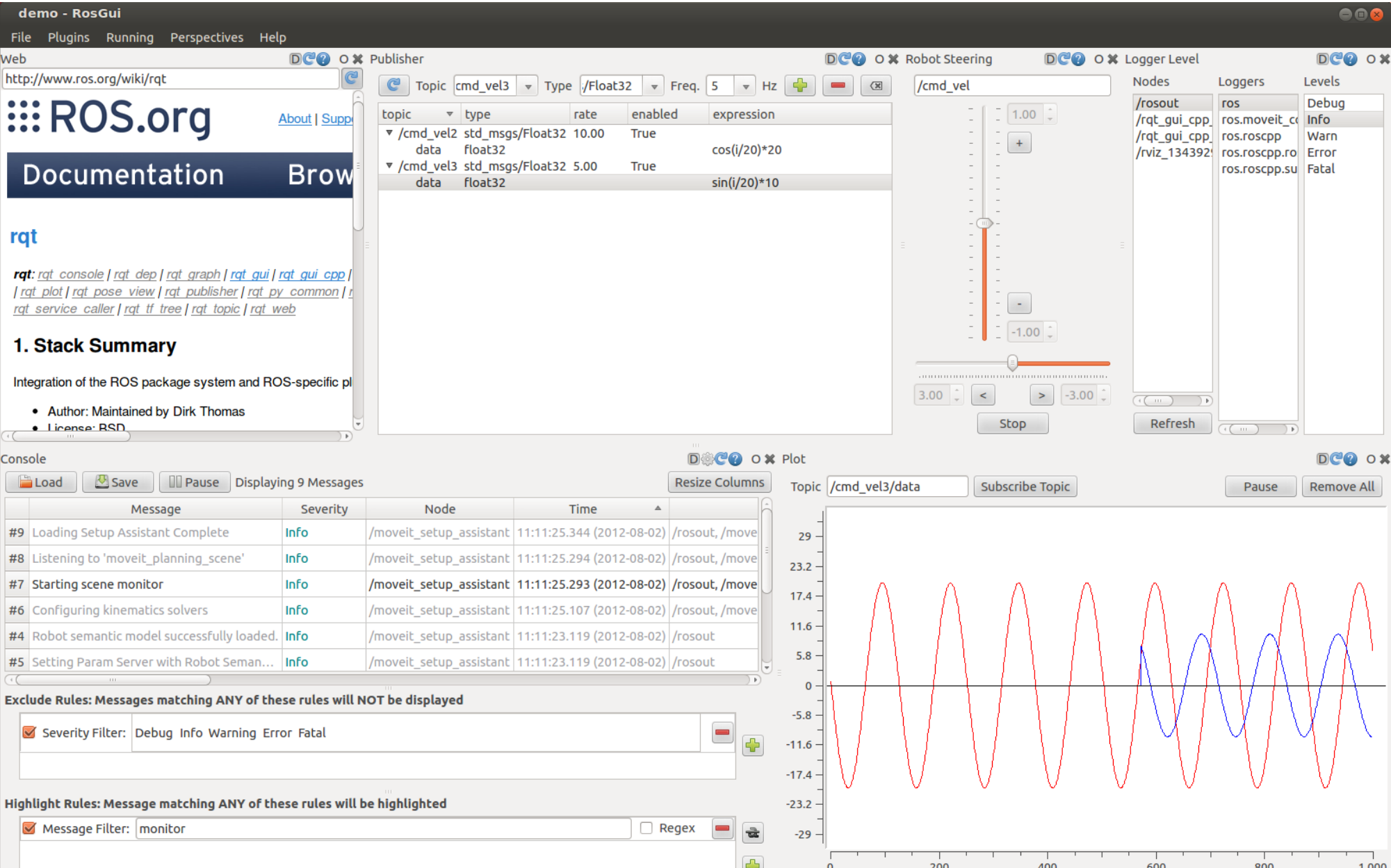
- allow multiple rqt widgets to be docked in a single window
- various GUI tools implemented as plugins:
  - Introspection (e.g., Node Graph)
  - Logging (e.g., Bag, Console)
  - Services (e.g., Service Caller)
  - Topics (e.g., Message Publisher)
  - Visualisation (e.g., Plot)

Standalone rqt GUI widgets

rqt widgets can also be executed as standalone tools (separate window)
For now, let us focus on the following rqt widgets:

- Node Graph
- Plot

Rqt – Qt-based framework GUI for ROS



Plotting data published on topics

Using rqt_plot

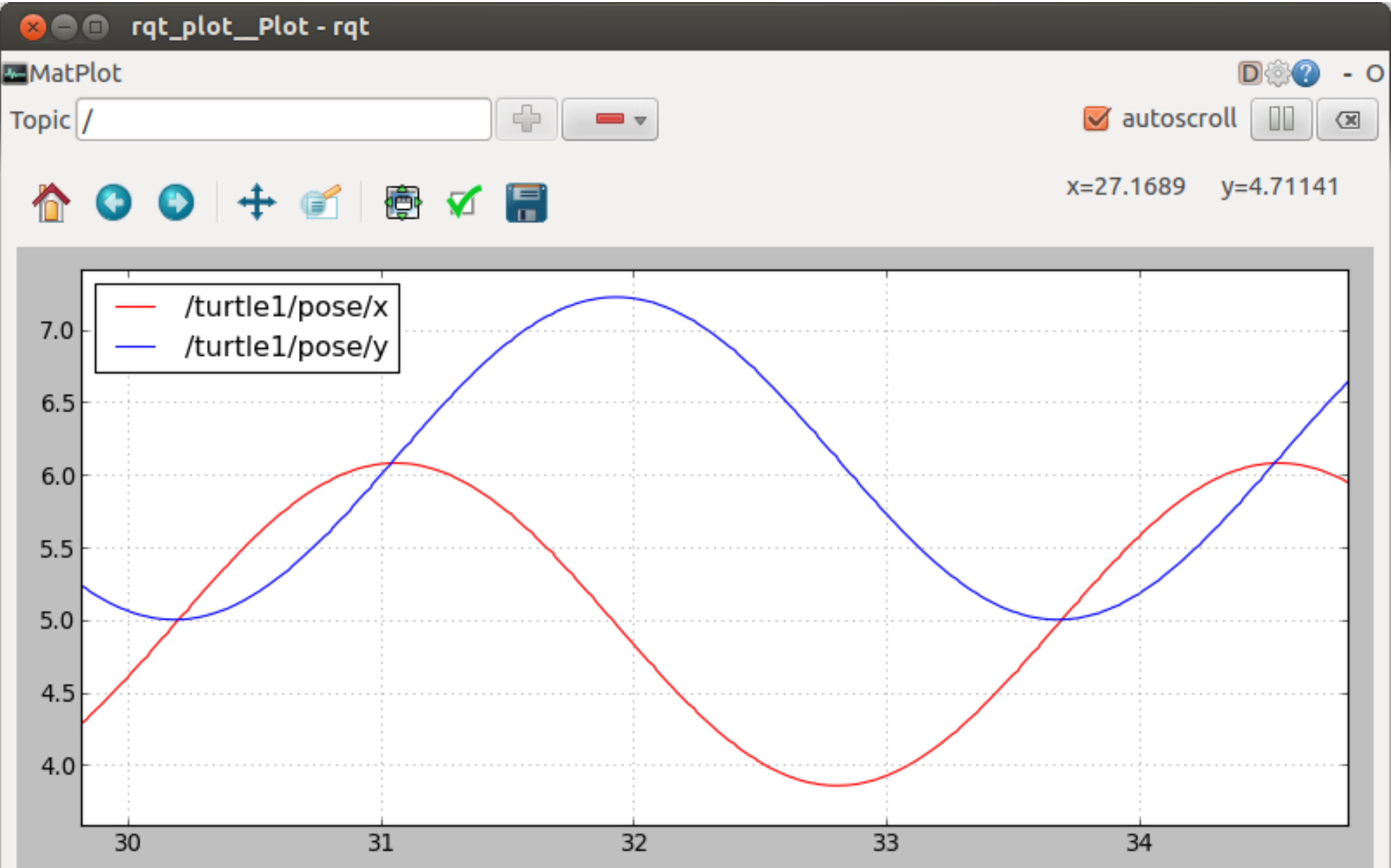
- display a scrolling time plot of the data published on topics
- rosrun rqt_plot rqt_plot
- As an example, let us plot the pose of *turtle1*:
  - we know the topic (/turtle1/pose) – recall rostopic list
  - we can find out the data types of ROS message being published: rostopic type /turtle1/pose | rosmsg show

murilo@jupiter: ~

```
murilo@jupiter:~$ rostopic type /turtle1/pose | rosmsg show
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity

murilo@jupiter:~$
```

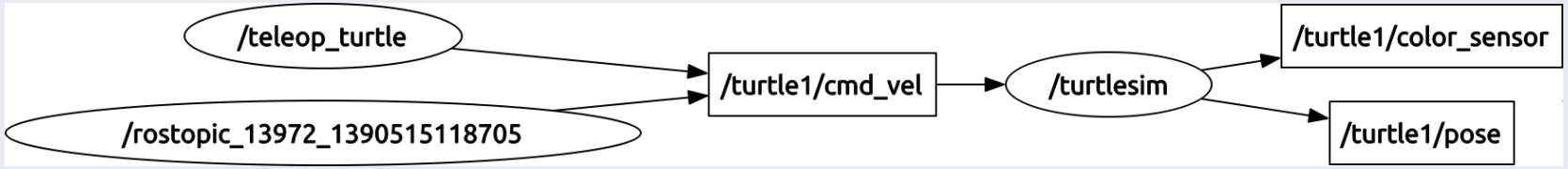
Plotting data published on topics



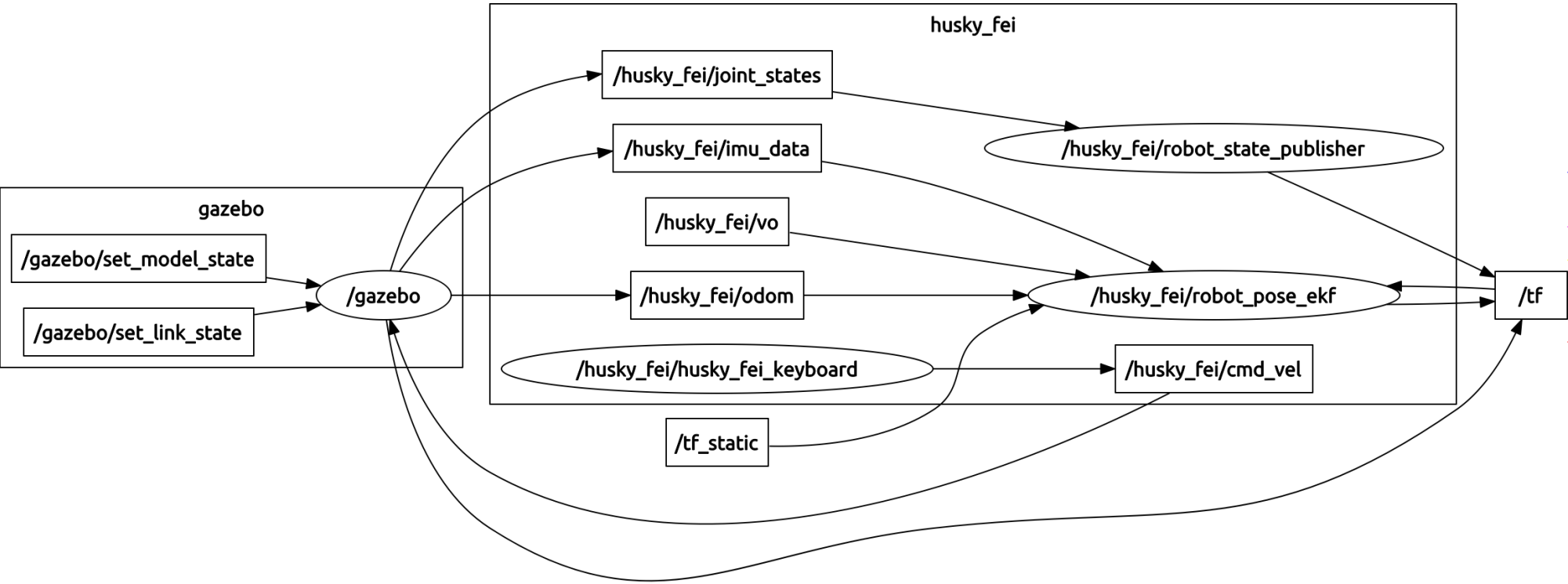
ROS Computation Graph (live)

Using rqt_graph

- the ROS Computation Graph can get extremely complex
- it is difficult to keep track of all nodes publishing/subscribing...
- and many distinct topics currently advertised
- rqt_graph creates a dynamic graph of what is going on in the system
- rosrun rqt_graph rqt_graph loads the GUI
- for instance, this is the Computation Graph of roscore + turtlesim_node + turtle_teleop_key + rostopic pub:



Another example of the ROS Computation Graph



Launch files

Overview

- we have progressively increased the number of open terminals:
  - roscore and rostopic pub
  - turtlesim_node and turtle_teleop_key
  - rqt_plot and rqt_graph
- at this pace, things will get out of hand very quickly
- and therefore we must start using launch files

roslaunch

- tool for easily launching multiple ROS nodes...
- and for setting parameters on the Parameter Server
- includes options to automatically respawn processes which died
- takes in one or more XML configuration files (.launch):
  - specifying paramaters to set and nodes to launch
  - machines on which the nodes should be loaded

Launch files

Using roslaunch

- simple usage: roslaunch [package] [filename.launch]
- we do not know how to create ROS packages yet...
- hence we will exceptionally create a separate launch file
- open gedit (press "windows" key and type "gedit")
- now type in the following...

turtle_mimic.launch

```xml
<?xml version="1.0"?>
<launch>

  <group ns="turtlesim1">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <group ns="turtlesim2">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <node pkg="turtlesim" name="mimic" type="mimic">
    <remap from="input" to="turtlesim1/turtle1"/>
    <remap from="output" to="turtlesim2/turtle1"/>
  </node>

</launch>
```

Launch files

Using roslaunch

- now save the file inside your catkin workspace
- close all terminals (shutdown all nodes and roscore)
- open a new terminal
- change directory to your catkin workspace...
- and type roslaunch turtle_mimic.launch

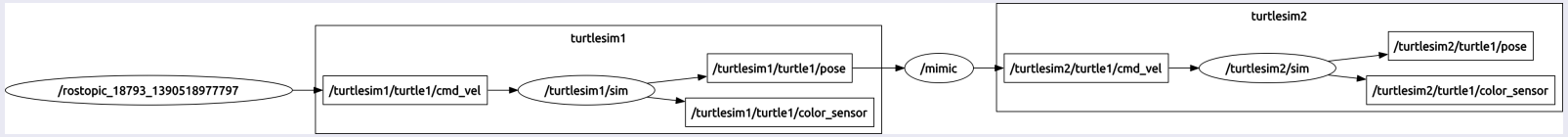
Thinking through what just happened...

- roscore was not running prior to roslaunch
- this is because roscore is a specialisation of roslaunch for bringing up the core ROS system

Using roslaunch

Testing whether topic remapping is working

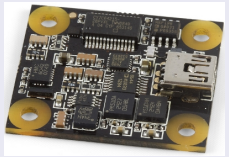
- check which topics are advertised
- notice the turtles have the same name...
- although they are within different namespaces
- publish messages over /turtlesim1/turtle1/cmd_vel
- verify the Computation Graph; it should look like this:



Live demo: rqt_gui interface with Phidgets IMU 3/3/3

Phidgets IMU 3/3/3 1056_0

- 3-axis compass: magnetic filed up to ± 4 Gauss
- 3-axis gyroscope: angular rotation up to $\pm 400^\circ/s$
- 3-axis accelerometer: up to 5G



ROS drivers/tools

- phidgets_drivers ROS package for Phidgets IMU interfaces
- imu_tools ROS package for IMU data fusion/filtering
- rqt_gui GUI for visualising (in real time):
  - ROS Computation Graph
  - live IMU data

Live demo: rqt_gui interface with Phidgets IMU 3/3/3