SUMMARY OF ROS BEGINNER-LEVEL TUTORIALS 1-10

ROBOT OPERATING SYSTEM LAB SESSION 3

20/03/2018

Installing and Configuring Your ROS Environment

- Check environment variables related to ROS:

```
$ printenv | grep ROS
```

- Source (read and execute) .*sh file:

```
$ source /opt/ros/<distro>/setup.bash
```

- Create a ROS Workspace:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Installing and Configuring Your ROS Environment

- Source (read and execute) new setup.*sh file:

```
$ source devel/setup.bash
```

- Display ROS_PACKAGE_PATH environment variable

```
$ echo $ROS_PACKAGE_PATH
```

Navigating the ROS Filesystem

- Installing ROS package on Linux:

```
$ sudo apt-get install ros-<distro>-ros-tutorials
$ sudo apt-get install ros-indigo-ros-tutorials
```

- To get information about packages:

```
$ rospack find [package_name]
$ rospack find roscpp
```

- To change directory (cd) directly to a package or a stack:

```
$ roscd [locationname[/subdir]]
$ roscd roscpp/cmake
```

Navigating the ROS Filesystem

- Print the working directory:

```
$ pwd
```

- Take you to the folder where ROS stores log files:

```
$ roscd log
```

- To ls directly in a package by name rather than by absolute path:

```
$ rosls [locationname[/subdir]]
$ rosls roscpp_tutorials
```

- Tab Completion

Creating a ROS Package

- The simplest possible package might have a structure which looks like this:

```
my_package/
  CMakeLists.txt
  package.xml
```

- The package must contain a catkin compliant package.xml file and a CMakeLists.txt which uses catkin.

Creating a ROS Package

- A trivial workspace might look like this:

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

Creating a ROS Package

- How to use the catkin_create_pkg script to create a new catkin package:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

- catkin_create_pkg requires that you give it a package_name and optionally a list of dependencies on which that package depends:

```
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

Creating a ROS Package

- To build the packages in the catkin workspace:

```
$ cd ~/catkin_ws
$ catkin_make
```

- After the workspace has been built it has created a similar structure in the devel subfolder as you usually find under /opt/ros/$ROSDISTRO_NAME.
- To add the workspace to your ROS environment you need to source the generated setup file:

```
$ . ~/catkin_ws/devel/setup.bash
```

Creating a ROS Package

- Typical procedures of creating a ROS package with nodes:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
$ catkin_make
$ source devel/setup.bash
$ roscd <package_name>
$ mkdir scripts
$ cd ~/scripts
$ vim XXX.py
$ chmod +x XXX.py
$ catkin_make
$ source devel/setup.bash
```

Creating a ROS Package

- First-order dependencies reviewed with the rospack tool:

```
$ rospack depends1 beginner_tutorials
```

- All nested dependencies reviewed with the rosapck tool:

```
$ rospack depends beginner_tutorials
```

- Elements of package.xml file:

description tag; maintainer tags; license tags; dependencies tags

Creating a ROS Package

```
<?xml version="1.0"?>
<package format="2">
  <name>beginner_tutorials</name>
  <version>0.1.0</version>
  <description>The beginner_tutorials package</description>

  <maintainer email="you@yourdomain.tld">Your Name</maintainer>
  <license>BSD</license>
  <url type="website">http://wiki.ros.org/beginner_tutorials</url>
  <author email="you@yourdomain.tld">Jane Doe</author>
```

Creating a ROS Package

```
 <buildtool_depend>catkin</buildtool_depend>

  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>

  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

</package>
```

Understanding ROS Nodes

- **Nodes**: A node is an 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)
- **rosout**: ROS equivalent of stdout/stderr
- **roscore**: Master + rosout + parameter server (parameter server will be introduced later)

Understanding ROS Nodes

- rosnode commands

rosnode info print information about node

rosnode kill kill a running node

rosnode list list active nodes

rosnode machine list nodes running on a particular machine or list machines

rosnode ping test connectivity to node

rosnode cleanup purge registration information of unreachable nodes

- Use the package name to directly run a node within a package:

```
$ rosrun [package_name] [node_name]
$ rosrun turtlesim turtlesim_node
```

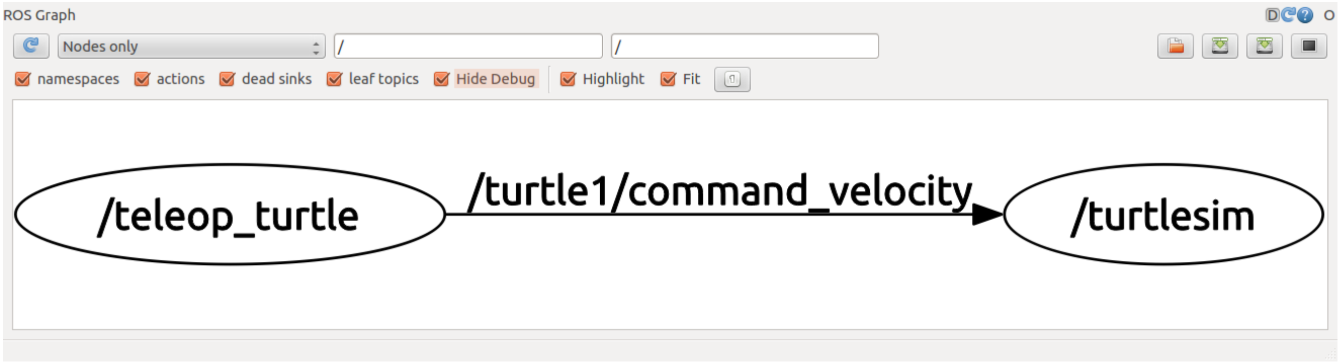
Understanding ROS Topics

- Testing with turtlesim:

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

- Using rqt_graph

```
$ rosrun rqt_graph rqt_graph
```



ROS Graph

Nodes only / /

☑ namespaces ☑ actions ☑ dead sinks ☑ leaf topics ☑ Hide Debug ☑ Highlight ☑ Fit

Understanding ROS Topics

rostopic bw   display bandwidth used by topic

rostopic echo   print messages to screen

```
$ rostopic echo /turtle1/cmd_vel
```

rostopic hz   display publishing rate of topic

rostopic list   print information about active topics

rostopic pub   publish data to topic

```
rostopic pub [topic] [msg_type] [args]
```

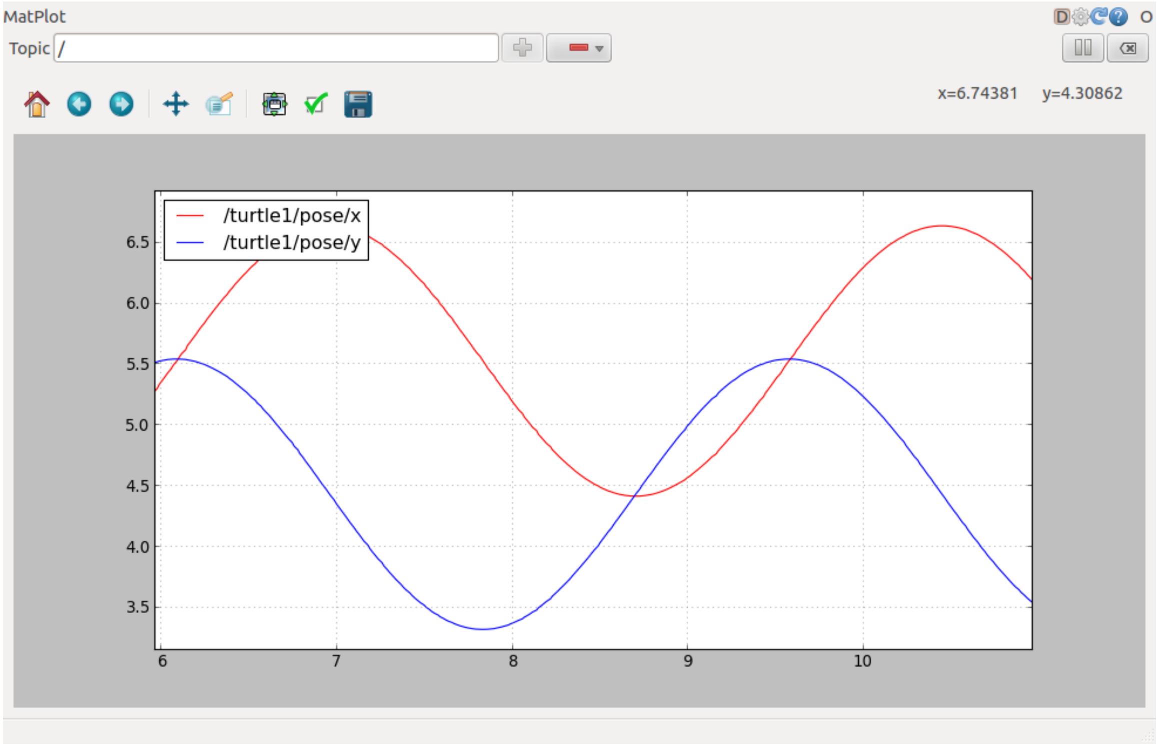
```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]'
'[0.0, 0.0, 1.8]'
```

rostopic type   print topic type

```
$ rostopic type /turtle1/cmd_vel
geometry_msgs/Twist
$ rosmsg show geometry_msgs/Twist
```

Understanding ROS Topics

- rqt_plot displays a scrolling time plot of the data published on topics:



Understanding ROS Services and Parameters

- Services are another way that nodes can communicate with each other. Services allow nodes to send a request and receive a response.

rosservice list	print information about active services
rosservice call	call the service with the provided args
rosservice type	print service type
rosservice find	find services by service type
rosservice uri	print service ROSRPC uri

Understanding ROS Services and Parameters

```
$ rosservice list
```

- The list command shows us that the turtlesim node provides nine services:

```
/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
```

Understanding ROS Services and Parameters

- rosservice type [service]
  - $ rosservice type /clear
  - std_srvs/Empty
- This service is empty, this means when the service call is made it takes no arguments (i.e. it sends no data when making a request and receives no data when receiving a response).
- rosservice call [service] [args]
  - $ rosservice call /clear
- This does what we expect, it clears the background of the turtlesim_node.

Understanding ROS Services and Parameters

- rosparam allows you to store and manipulate data on the ROS Parameter Server.

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 Services and Parameters

- rosparam set and rosparam get
- change the red channel of the background color:

```
$ rosparam set /background_r 150
```

- This changes the parameter value, now we have to call the clear service for the parameter change to take effect:

```
$ rosservice call /clear
```

Understanding ROS Services and Parameters

- rosparam dump [file_name] [namespace]
- rosparam load [file_name] [namespace]
- Here we write all the parameters to the file params.yaml

```
$ rosparam dump params.yaml
```

- You can even load these yaml files into new namespaces, e.g. copy:

```
$ rosparam load params.yaml copy
$ rosparam get /copy/background_b
```

Using rqt_console and roslaunch

- rqt_console attaches to ROS's logging framework to display output from nodes. rqt_logger_level allows us to change the verbosity level (DEBUG, WARN, INFO, and ERROR) of nodes as they run.

```
$ rosrun rqt_console rqt_console
$ rosrun rqt_logger_level rqt_logger_level
```

- Logging levels are prioritized in the following order:

Fatal

Error

Warn

Info

Debug

Using rqt_console and roslaunch

- Roslaunch starts nodes as defined in a launch file.
- $ roslaunch [package] [filename.launch]

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roscd beginner_tutorials
$ mkdir launch
$ cd launch
```

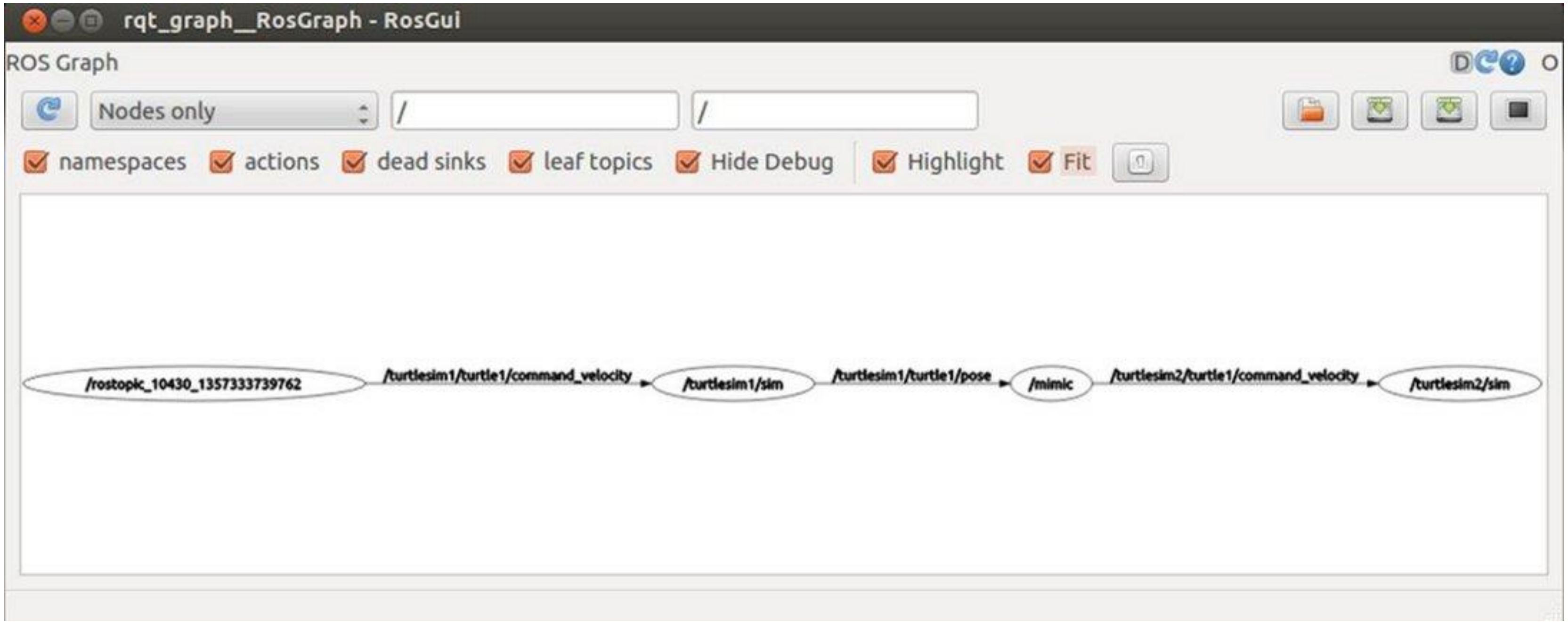
Using rqt_console and roslaunch

```
<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>
```

Using rqt_console and roslaunch


rqt_graph__RosGraph - RosGui
ROS Graph
/rostopic_10430_1357333739762
/turtlesim1/turtle1/command_velocity
/turtlesim1/sim
/turtlesim1/turtle1/pose
/mimic
/turtlesim2/turtle1/command_velocity
/turtlesim2/sim

Nodes only

/

/

☑ namespaces ☑ actions ☑ dead sinks ☑ leaf topics ☑ Hide Debug ☑ Highlight ☑ Fit

Using rosed to edit files in ROS

- rosed is part of the rosbash suite. It allows you to directly edit a file within a package by using the package name rather than having to type the entire path to the package:

```
$ rosed [package_name] [filename]

$ rosed roscpp Logger.msg
```

Creating a ROS msg and srv

- msg: msg files are simple text files that describe the fields of a ROS message. They are used to generate source code for messages in different languages
- srv: an srv file describes a service. It is composed of two parts: a request and a response.
- msg files are stored in the msg directory of a package, and srv files are stored in the srv directory.
- There is also a special type in ROS: Header, the header contains a timestamp and coordinate frame information that are commonly used in ROS. You will frequently see the first line in a msg file have Header header.

Creating a ROS msg and srv

- An example of a msg that uses a Header:

```
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

- An example of a srv file:

```
int64 A
int64 B
---
int64 Sum
```

Creating a ROS msg and srv

●Creating a msg

1 Define a new msg in the package that was created in the previous tutorial:

```
$ roscd beginner_tutorials
$ mkdir msg
$ echo "int64 num" > msg/Num.msg
```

2 Make sure two lines are uncommented in package.xml:

```
 <build_depend>message_generation</build_depend>
  <exec_depend>message_runtime</exec_depend>
```

Creating a ROS msg and srv

3 Add the message_generation dependency to the find_package call which already exists in your CMakeLists.txt:

```
find_package(catkin REQUIRED COMPONENTS
   roscpp
   rospy
   std_msgs
   message_generation
)
```

4 Export the message runtime dependency:

```
catkin_package(
  ...
  CATKIN_DEPENDS message_runtime ...
  ...)
```

Creating a ROS msg and srv

5 Find the following block of code:

```
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )
```

Uncomment it by removing the # symbols and then replace the stand in Message*.msg files with your .msg file, such that it looks like this:

```
add_message_files(
  FILES
  Num.msg
)
```

Creating a ROS msg and srv

6 Uncomment these lines:

```
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )
```

So it looks like:

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

Creating a ROS msg and srv

Procedures of Creating a msg:

1 Create msg/Num.msg

----------------------package.xml----------------------------

2 Uncommt two lines of dependencies

(message_generation&message_runtime)

----------------------CmakeList.txt----------------------------

3 Add the message_generation dependency to the find_package call

4 make sure you export the message runtime dependency (catkin_package)

5 Uncomment add_message_files by removing the # symbols and then replace the stand in Message*.msg files with your .msg file

6 Ensure the generate_messages() function uncommented.

Creating a ROS msg and srv

- Use rosmsg:

```
$ rosmsg show [message type]
$ rosmsg show beginner_tutorials/Num
int64 num
```

beginner_tutorials -- the package where the message is defined

Num -- The name of the msg Num.

```
$ rosmsg show Num
[beginner_tutorials/Num]:
int64 num
```

Creating a ROS msg and srv

- Creating a srv:

1 create a srv folder:

```
$ roscd beginner_tutorials
$ mkdir srv
```

2 roscp is a useful commandline tool for copying files from one package to another:

```
$ roscp [package_name] [file_to_copy_path] [copy_path]
```

```
$ roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv
```

Creating a ROS msg and srv

3 Open package.xml, and make sure these two lines are in it and uncommented:

```
 <build_depend>message_generation</build_depend>
  <exec_depend>message_runtime</exec_depend>
```

4 Add the message_generation dependency to generate messages in CMakeLists.txt:

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
```

Creating a ROS msg and srv

5 Remove # to uncomment the following lines:

```
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )
```

replace the placeholder Service*.srv files for your service files:

```
add_service_files(
  FILES
  AddTwoInts.srv
)
```

Creating a ROS msg and srv

●Procedures of Creating a srv:

1 Create a .srv file in srv folder.

----------------------package.xml---------------------------

2 Uncommt two lines of dependencies

(message_generation&message_runtime)

----------------------CmakeList.txt----------------------------

3 Add the message_generation dependency to generate messages in CMakeLists.txt.(find_package)

4 uncomment add_service_files function and replace

placeholder Service*.srv files for your service files。

Creating a ROS msg and srv

```
$ rossrv show <service type>
$ rossrv show beginner_tutorials/AddTwoInts ：
int64 a
int64 b
---
int64 sum
```

Similar to rosmsg, you can find service files like this without specifying package name:

```
$ rossrv show AddTwoInts
[beginner_tutorials/AddTwoInts]:
int64 a
int64 b
---
int64 sum
[rospy_tutorials/AddTwoInts]:
int64 a
int64 b
---
int64 sum
```

Creating a ROS msg and srv

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

Tasks

- Understand the basic knowledge of ROS

- Finish beginner-level ROS tutorials.

- Build ROS package including a publisher and a subscriber. (deadline: 27/03/2018)

Useful links

- JetBrain Pycharm
  https://www.jetbrains.com/pycharm/download/#section=linux

- ROS Environment Setup (Pycharm)

https://www.ncnynl.com/archives/201611/1056.html

- Writing a Simple Publisher and Subscriber (Python)

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29