
TurtleSim

Contents

Table of Figures

ROS TERMS

Before a beginner even opens a web tutorial or book or sees a ROS video, it is helpful to learn a few terms that pertain to ROS. These terms describe the main components of a ROS system.

Table 1. ROS Useful Terms		
Item	Type	Comment
Repositories	A **software repository** is a storage location from which software packages may be retrieved and installed on a computer.	http://en.wikipedia.org/wiki/Software_repository GitHub is used to download the ROS packages used by the Baxter system: http://sdk.rethinkrobotics.com/wiki/Workstation_Setup
Packages	Contains files to allow execution of ROS programs	A package typically contains source files and executable scripts that can be BASH, Python, or other code.
Manifest Package.xml	Information about a package	The manifest defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other packages.
ROS Master	Registers the name and location of each node.	Allows nodes to communicate. Nodes can be in different computers.
Parameter Server	Data types that define certain information for nodes.	Certain nodes require parameters to define aspects of the node.
Nodes	Processes that execute commands.	Executable code written in Python or C++ usually. Python nodes use the client library rospy
Topic	Name of a message.	For example, Baxter's cameras "publish" the image they receive as a topic with a name that indicates it is a camera image.
Services	Allows communication between nodes.	Used by nodes to communicate with other nodes and request a response.
Messages	Data sent between nodes.	Messages are "published" by a node and "subscribed to" by another node.
Bags	Data storage for messages.	Used to save and playback data such as sensor data.

Table 1 ROS Terms

The following tables define the help for the various ROS commands.

TURTLESIM PACKAGE

Be sure that the turtlesim package is loaded on your system. Open a terminal window and try the following commands:

$ **rospack find turtlesim**
opt/ros/indigo/share/turtlesim

$ **rosls turtlesim** (List Files - Note no need to type full path)
cmake images msg package.xml srv
Read the package.xml, look at the images. Srv directory has services

$ pwd (Show working directory – not at turtlesim)

$ roscd turtlesim
/opt/ros/indigo/share/turtlesim

rospack List Packages

Two of the ROS packages we are using in this report are **std_msgs** and **turtlesim.** The command **rospack list** lists the packages and their directories on the workstation of which std_msgs and turtlesim are only two of many.

tlharmanphd@D125-43873:~$ **rospack list**
.
.
std_msgs /opt/ros/indigo/share/std_msgs

.
turtlesim /opt/ros/indigo/share/turtlesim
.
.

Note that the distribution of ROS is Indigo.

To clear the screen of the long list:
tlharmanphd@D125-43873:~$ **clear**

rospack help

tlharmanphd@D125-43873:/$ **rospack help**

```
USAGE: rospack <command> [options] [package]
 Allowed commands:
   help
   cflags-only-I     [--deps-only] [package]
   cflags-only-other [--deps-only] [package]
   depends           [package] (alias: deps)
   depends-indent    [package] (alias: deps-indent)
   depends-manifests [package] (alias: deps-manifests)
   depends-msgsrv    [package] (alias: deps-msgsrv)
   depends-on        [package]
   depends-on1       [package]
   depends-why --target=<target> [package] (alias: deps-why)
   depends1          [package] (alias: deps1)
   export [--deps-only] --lang=<lang> --attrib=<attrib> [package]
   find [package]
   langs
   libs-only-L     [--deps-only] [package]
   libs-only-l     [--deps-only] [package]
   libs-only-other [--deps-only] [package]
   list
   list-duplicates
   list-names
   plugins --attrib=<attrib> [--top=<toppkg>] [package]
   profile [--length=<length>] [--zombie-only]
   rosdep  [package] (alias: rosdeps)
   rosdep0 [package] (alias: rosdeps0)
   vcs  [package]
   vcs0 [package]
 Extra options:
   -q     Quiets error reports.

 If [package] is omitted, the current working directory
 is used (if it contains a manifest.xml).
```

tlharmanphd@D125-43873:~$ **rospack depends turtlesim**

```
cpp_common
rostime
roscpp_traits
roscpp_serialization
genmsg
genpy
message_runtime
rosconsole
std_msgs
rosgraph_msgs
xmlrpcpp
roscpp
catkin
rospack
roslib
std_srvs
```

ROSCORE

This starts ROS and creates the Master so that nodes can communicate.

tlharmanphd@D125-43873:~$ **roscore**

... logging to /home/tlharmanphd/.ros/log/2429b792-d23c-11e4-b9ee-3417ebbca982/roslaunch-D125-43873-21164.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://D125-43873:60512/
ros_comm version 1.11.10

SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.10 1.11.16 2/2016 HP 210 Laptop

NODES

auto-starting new master
process[master]: started with pid [21176]
ROS_MASTER_URI=http://D125-43873:11311/ (Lab Workstation)

setting /run_id to 2429b792-d23c-11e4-b9ee-3417ebbca982
process[rosout-1]: started with pid [21189]
started core service [/rosout]

Figure 1 ROSCORE Command

From the ROS tutorial http://wiki.ros.org/roscore

> roscore is 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. It is launched using the `roscore` command.
>
> NOTE: If you use roslaunch, it will automatically start `roscore` if it detects that it is not already running.
>
> `roscore` will start up:
>
> - a ROS Master
> - a ROS Parameter Server
> - a rosout logging node

Leave this window active but minimized so that the ROS Master is still available.

CREATE A TOPIC AND A MESSAGE

After ROSCORE, two topics are /rosout and /rosout_agg. Let's publish a topic to see how this works from Clearpath Robotics ROS-101 Practical Example.

In another terminal list the topics:

tlharmanphd@D125-43873:~$ **rostopic list**
/rosout
/rosout_agg

`rosout` is the name of the console log reporting mechanism in ROS. It can be thought as comprising several components:

For a little explanation from the ROS wiki:

http://wiki.ros.org/rosout

- The `rosout` node for subscribing, logging, and republishing the messages.
- The /rosout topic
- The /rosout_agg topic for subscribing to an aggregated feed
- rosgraph_msgs/Log message type, which defines standard fields as well as verbosity levels.

The `rosout` *package* only provides the `rosout` node.

Create a Topic /hello and A String Message with rostopic pub
tlharmanphd@D125-43873:~$ **rostopic pub /hello std_msgs/String "Hello User"**
publishing and latching message. Press ctrl-C to terminate

New Topic /hello from rostopic list

In a new window, note that the topic **/hello** is present using **rostopic list**.

tlharmanphd@D125-43873:~$ **rostopic list**
/hello
/rosout
/rosout_agg

Data for Topic /hello- message
tlharmanphd@D125-43873:~$ **rostopic echo /hello**
data: Hello User

New Node caused by rostopic pub
tlharmanphd@D125-43873:~$ **rosnode list**
/rosout
/rostopic_4375_1424715269456

Thus far, we have a new topic **/hello** and a new node **/rostopic_4375_1424715269456.** The topic number will change with each run since it indicates time as well as other information.
.

Node Information

tlharmanphd@D125-43873:~$ **rosnode info /rostopic_4375_1424715269456**

```
--------------------------------------------------------------------------------
Node [/rostopic_4375_1424715269456]
Publications:
 * /hello [std_msgs/String]

Subscriptions: None
Services:
 * /rostopic_4375_1424715269456/set_logger_level
 * /rostopic_4375_1424715269456/get_loggers

contacting node http://D125-43873:36024/ ...
Pid: 4375
Connections:
 * topic: /hello
   * to: /rostopic_4504_1424715395163
   * direction: outbound
⓾       transport: TCPROS
```

This created the **topic** /hello with **topic type** std_msgs/String. **Hello User** is the data.

We now have a new topic **/hello** with data that can be sent to the screen by the **echo** option of the **rostopic** command. There are now three topics as shown by the **rostopic list** command. The publications, subscriptions and services for the new node are shown by the **rosnode info** command.

This also created the node **/rostopic_4375_1424715269456** which is publishing the data of topic **/hello.**

From O'Kane Chapter 3 "By the way, the numbers in this output line represent the time—measured in seconds since January 1, 1970—when our ROS_INFO_STREAMline was executed."

In our case, take the 10 digits that represent seconds, **1424715269** and show that this represents about 45 years (2015-1970).

Kill A Node

You can close the window with the node /hello defined or kill the node with **rosnode kill <node>** command**.** The <node> here is **/rostopic_4375_1424715269456.**

tlharmanphd@D125-43873:~$ rosnode kill -h

```
Usage: rosnode kill [node]...

Options:
  -h, --help  show this help message and exit
  -a, --all   kill all nodes
```

To check running process use **$ps –ef** to see all the processes running.

ROS NODES, TOPICS, AND SERVICES USING TURTLESIM

Before going through this section and especially the tutorials on ros.org, you should have read and understood the material about ROS covered in the *Introduction to Baxter* report by T.L. Harman and Carol Fairchild.

A READ OF THE FIRST FEW CHAPTERS IN BOOKS SUCH AS THE FOLLOWING WILL BE HELPFUL:

The textbook *Learning ROS for Robotics Programming* by Aaron Martinez is useful. The examples are in C++.

A Gentle Introduction to ROS by Jason M. O'Kane is very readable and can be downloaded from the site: http://www.cse.sc.edu/~jokane/agitr/agitr-letter.pdf
The author's website is http://www.cse.sc.edu/~jokane/agitr/

These other ROS books might be helpful as referenced by O'Kane:

ROS by Example by R. Patrick Goebel

Learning ROS for Robotics Programming - Second Edition

https://www.packtpub.com/hardware-and-creative/learning-ros-robotics-programming-second-edition

by Enrique Fernandez and others. The examples are in C++.

Always be sure to check of any changes in the Ubuntu or ROS distribution. This *Turtlesim Guide* is written using Ubuntu 14.04 and ROS Indigo.

 If you are new to ROS - don't be impatient. There is a great deal to learn but the Turtlesim example shown here should make things easier.

The ROS official tutorials are at these WEB sites: **http://wiki.ros.org/turtlesim/Tutorials**

ROS Tutorials Helpful for the Examples to Follow:

- ROS/Tutorials/UnderstandingNodes
- ROS/Tutorials/UnderstandingTopics
- ROS/Tutorials/UnderstandingServicesParams

Other useful references are Listed in Appendix I

rosnode, rostopic help

tlharmanphd@D125-43873:~$ **rosnode help**

rosnode is a command-line tool for printing information about ROS Nodes.

Commands: **Example**

rosnode ping	test connectivity to node	**($ rosnode ping <node>)**
rosnode list	list active nodes	
rosnode info	print information about node	**($ rosnode info <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'

tlharmanphd@D125-43873:~$ **rosnode list -h**

Usage: rosnode list

Options:

-h, --help show this help message and exit
-u list XML-RPC URIs
-a, --all list all information rosnode kill kill a running node
rosnode cleanup purge registration information of unreachable nodes

tlharmanphd@D125-43873:/$ **rostopic help**

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'

TURTLESIM NODE

We will start the turtlesim node and explore its properties. In a new terminal create the turtlesim node from the package turtlesim:

tlharmanphd@D125-43873:~$ **rosrun turtlesim turtlesim_node**

[INFO] [1427212356.117628994]: Starting turtlesim with node name /turtlesim
[INFO] [1427212356.121407419]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]

The rosrun command takes the arguments [package name] [node name]. The node creates the screen image and the turtle. Here the turtle is in the center in x=5.5, y=5.5 with no rotation.

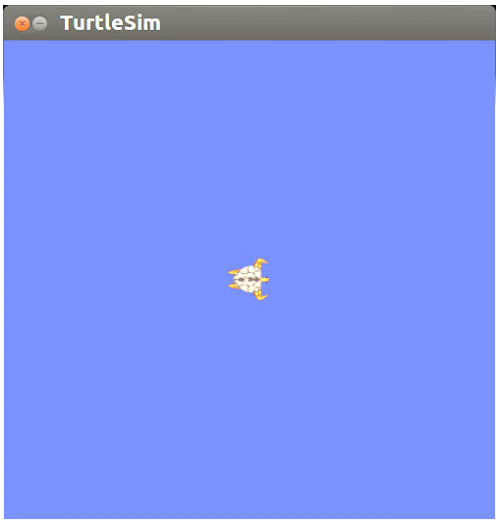


Figure 2 Turtlesim Window

Before moving the turtle, let's study the properties of the nodes, topics, service and messages available with turtlesim package in another window.

ROS Nodes with Turtlesim

Open a new window. Most ROS commands have help screens that are usually helpful.

Using the **rosnode** command we can determine information about any node. First list the two active nodes:

rosnode list

tlharmanphd@D125-43873:~$ **rosnode list**

/rosout
/turtlesim

Note the difference in notation between the node **/turtlesim** and the *package* **turtlesim**.

tlharmanphd@D125-43873:~$ **rosnode info /turtlesim**

```
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:                (This information is sent to nodes listening to /turtlesim)
 * /turtle1/color_sensor [turtlesim/Color]   (Color message in turtlesim package)

 * /rosout [rosgraph_msgs/Log]
 * /turtle1/pose [turtlesim/Pose]       (Pose message in turtlesim package)


Subscriptions:
 * /turtle1/cmd_vel [unknown type]  (This node will listen for command velocities)

(We can use ROS services to manipulate the turtle and perform other operations.)

Services: (he format is $rosservice call <service> <arguments>)
* /turtle1/teleport_absolute
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level
 * /reset
 * /spawn
 * /clear
 * /turtle1/set_pen
 * /turtle1/teleport_relative
 * /kill

contacting node http://D125-43873:47701/ ...
Pid: 21255
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
```

The node **/turtlesim** publishes three topics and subscribes to the **/turtle1/cmd_vel** topic**.** The services for the node are listed also.

TURTLESIM NODE TOPICS AND MESSAGES

For these exercises, keep the roscore and turtlesim windows active. Open other terminal windows as needed. We will concentrate on the node **turtle1/color_sensor** first.

tlharmanphd@D125-43873:/$ **rosservice help**
Commands:

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

Type rosservice <command> -h for more detailed usage, e.g. 'rosservice call -h'

Use the $**rosservice list** command to see the services for the active node.

Messages

tlharmanphd@D125-43873:/$ **rosmsg help**
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

If a topic publishes a message, we can determine the message type and read the message. This is shown in the example to determine the color of the background for the turtle.

tlharmanphd@D125-43873:~$ **rostopic type /turtle1/color_sensor**
turtlesim/Color

The message type in the case of the topic /turtle1/color_sensor is
turtlesim/Color.

The word “type” in this context is referring to the concept of a
data type . It’s important to understand message types because they determine the content of the messages. That is, the message type of a topic tells you what information is included in each message on that topic, and how that information is organized.

From the message *type* we can fine the format of the message. Be sure to note that Color in the message type starts with a capital letter. http://wiki.ros.org/rostopic

rosmsg show

tlharmanphd@D125-43873:~$ **rosmsg show turtlesim/Color**

```
uint8 r
uint8 g
uint8 b
```

http://wiki.ros.org/msg

To understand the format of the message it is necessary to find the message type. The types include integers of 8, 16, 32, or 64 bits, floating point numbers, strings and other formats. The structure of the message type is:

<field> <constant>

where the field defines the type of data and the constant is the name. For example, the red color in the background is defined by **uint8 r** as shown below. This indicates that if we wish to modify the red value, an 8-bit unsigned integer is needed. jThe amount of red in the background is thus in the range of 0-255.

Example 1: Determine the color mixture of red, green, and blue in the background of our turtle.

tlharmanphd@D125-43873:~$ **rostopic echo /turtle1/color_sensor**

```
r: 69
g: 86
b: 255
---
r: 69
g: 86
b: 255
 .
 .
```

The WEB site ColorChart explains the color chart. A later example will show how to change the background color for the turtle. The color values are **parameters** that can be changed.

PARAMETER SERVER

rosparam help

tlharmanphd@D125-43873:/$ **rosparam help**
rosparam is a command-line tool for getting, setting, and deleting parameters from the ROS Parameter Server.

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

tlharmanphd@D125-43873:~$ **roscore**
(Start ROS Master)

tlharmanphd@D125-43873:~$ **rosparam list**
/rosdistro
/roslaunch/uris/host_d125_43873__39549
/rosversion
/run_id

tlharmanphd@D125-43873:~$ **rosparam get rosversion**
'1.11.10 (1.11.16 2/08/2016)

tlharmanphd@D125-43873:~$ **rosparam get rosdistro**
'indigo

As the node publishes, the color of the background for example, it is possible to change the parameters. The command format is

rosparam list for turtlesim node
To list the parameters for the turtlesim node:

tlharmanphd@D125-43873:/$ **rosparam list**
/background_r
/background_g
/background_b
/rosdistro
/roslaunch/uris/host_d125_43873__51759
/rosversion

/run_id

change parameters for color of turtle background

Let's turn the turtle's background red. To do this, make the blue and green parameters equal to zero and saturate red = 255 by using the **rosparam set** command. Note the **clear** option from **rosservice** must be executed before the screen changes color.

rosparam get
tlharmanphd@D125-43873:~$ **rosparam get /**

```
background_b: 255
background_g: 86
background_r: 69
rosdistro: 'indigo
roslaunch:
  uris: {host_d125_43873__60512: 'http://D125-43873:60512/'}
rosversion: '1.11.10
run_id: 2429b792-d23c-11e4-b9ee-3417ebbca982
```

rosparam set

Change the colors:
tlharmanphd@D125-43873:/$ **rosparam set background_b 0**
tlharmanphd@D125-43873:/$ **rosparam set background_g 0**
tlharmanphd@D125-43873:/$ **rosparam set background_r 255**
tlharmanphd@D125-43873:/$ **rosservice call /clear**

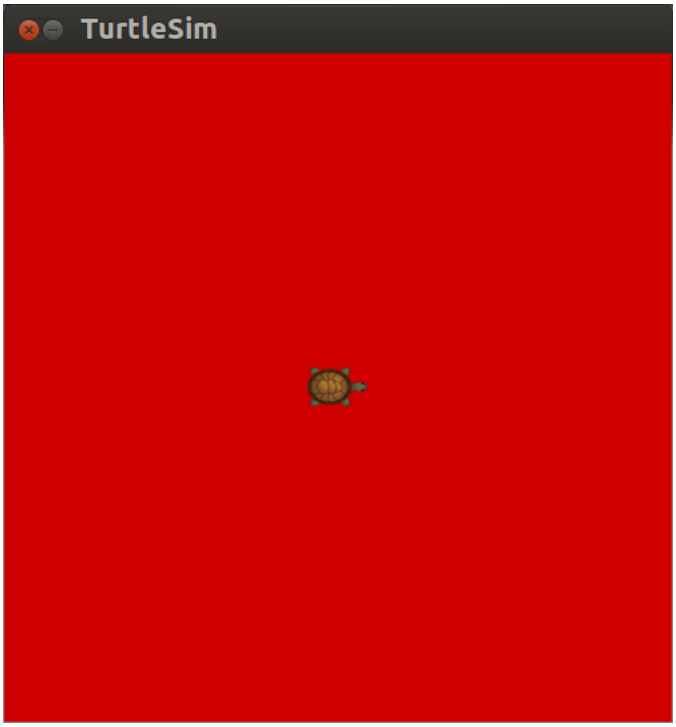


Figure 3 Turtlesim Window Color Change

rosparam get turtlesim parameters
To check the results, use the **rosparam get** command.

```
tlharmanphd@D125-43873:~$ rosparam get /
        background_b: 0
        background_g: 0
        background_r: 255
        rosdistro: 'indigo
        roslaunch:
          uris: {host_d125_43873__60512: 'http://D125-43873:60512/'}
        rosversion: '1.11.10
        run_id: 2429b792-d23c-11e4-b9ee-3417ebbca982
```

ROS SERVICES TO MOVE TURTLE

Services: **(We can use ROS services to manipulate the turtle and perform other operations - the format is $rosservice call <service> <arguments>)**

* /turtle1/teleport_absolute
* /turtlesim/get_loggers
* /turtlesim/set_logger_level
* /reset
* /spawn
* /clear
* /turtle1/set_pen
* /turtle1/teleport_relative
* /kill

The turtle can be moved using the rosservice teleport option. The format of the position is [x y theta].

teleport_absolute

tlharmanphd@D125-43873:/$ **rosservice call /turtle1/teleport_absolute 1 1 0**

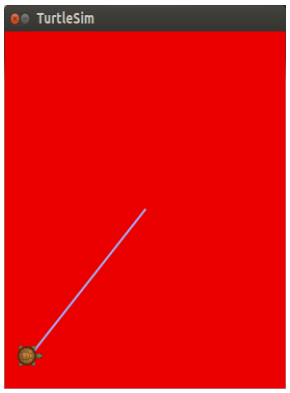


Figure 4 Turtle After Absolute Move

The relative teleport option moves the turtle with respect to its present position. The arguments are [linear, angle]

teleport_relative

tlharmanphd@D125-43873:/$ **rosservice call /turtle1/teleport_relative 1 0**

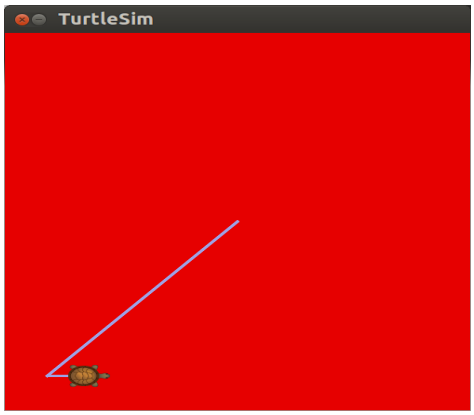


Figure 5 Turtle After Relative Move

Turtle now at x=2, y=1.

TURTLESIM NODE TOPIC POSE

Another topic for turtlesim node is the turtle's **pose.** This is the x, y position, angular direction, and the linear and angular velocity. In this example, the turtle is not moving as shown in Figure 5.

tlharmanphd@D125-43873:~$ **rostopic info /turtle1/pose**

Type: turtlesim/Pose

Publishers:
* /turtlesim (http://D125-43873:47701/)

Subscribers: None

tlharmanphd@D125-43873:~$ **rostopic type /turtle1/pose**

turtlesim/Pose

tlharmanphd@D125-43873:~$ **rosmsg show turtlesim/Pose**

float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity

tlharmanphd@D125-43873:/$ **rostopic echo /turtle1/pose**

x: 2.0
y: 1.0
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 2.0
y: 1.0
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
.
.
.

Continuous output of the position, orientation, and velocities. Compare to the position on the turtle window. CNTL+c to stop output.

MAKE TURTLE RUN IN A CIRCLE WITH TURTLESIM

tlharmanphd@D125-43873:~$ **rosnode info /turtlesim**

```
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:
 * /turtle1/color_sensor [turtlesim/Color]
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [unknown type]

Services:
 * /turtle1/teleport_absolute
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level
 * /reset
 * /spawn
 * /clear
 * /turtle1/set_pen
 * /turtle1/teleport_relative
 * /kill


contacting node http://D125-43873:47701/ ...
Pid: 21255
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
```

type of message for cmd_vel

tlharmanphd@D125-43873:~$ **rostopic type /turtle1/cmd_vel**

```
geometry_msgs/Twist
```

tlharmanphd@D125-43873:~$ **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
```

COMBINE TWO COMMANDS

tlharmanphd@D125-43873:~$ **rostopic type /turtle1/cmd_vel | rosmsg show**

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

The requirement is for two vectors with 3 elements each. The message type is geometry_msgs/Twist .

To get a list of messages for ROS of **geometry_msgs**
http://wiki.ros.org/geometry_msgs

This displays a verbose list of topics to publish to and subscribe to and their type:
tlharmanphd@D125-43873:~$ **rostopic list -v**

```
Published topics:
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher
 * /rosout [rosgraph_msgs/Log] 1 publisher
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher

Subscribed topics:
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
* /rosout [rosgraph_msgs/Log] 1 subscriber
```

MOVE TURTLE ONCE
The following command will send a single message to turtlesim telling it to move with a linear velocity of 2.0, and an angular velocity of 1.8. It will move from its starting position along a circular trajectory for a distance and then stop.

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

Where is the turtle?

tlharmanphd@D125-43873:~$ **rostopic echo /turtle1/pose**

```
x: 3.0583717823
y: 2.39454507828
theta: 1.81439995766
linear_velocity: 0.0
angular_velocity: 0.0
```

Use CNTL+c to stop the output of position, orientation and velocity.

From the ROS tutorial, a geometry_msgs/Twist msg has two vectors of three floating point elements each: `linear` and `angular`. In this case, '`[2.0, 0.0, 0.0]`' becomes the linear value with `x=2.0`, `y=0.0`, and `z=0.0`, and '`[0.0, 0.0, 1.8]`' is the `angular` value with `x=0.0`, `y=0.0`, and `z=1.8`. These arguments are actually in YAML syntax, which is described more in the YAML command line documentation.

'[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

You will have noticed that the turtle has stopped moving; this is because the turtle requires a steady stream of commands at 1 Hz to keep moving. We can publish a steady stream of commands using `rostopic pub -r` command:

Here we publish the topic **/turtle1/command_velocity** with the message to repeat the message at 1 second intervals with linear velocity 2 and angular velocity 1.8. The node turtlesim subscribes to the message as shown by the command $ **rosnode info /turtlesim** shown before with the subscription:

Subscribed topics:
* /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber **rostopic pub**

To make the turtle move in a circle

harman@Laptop-M1210:~$ **rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'**

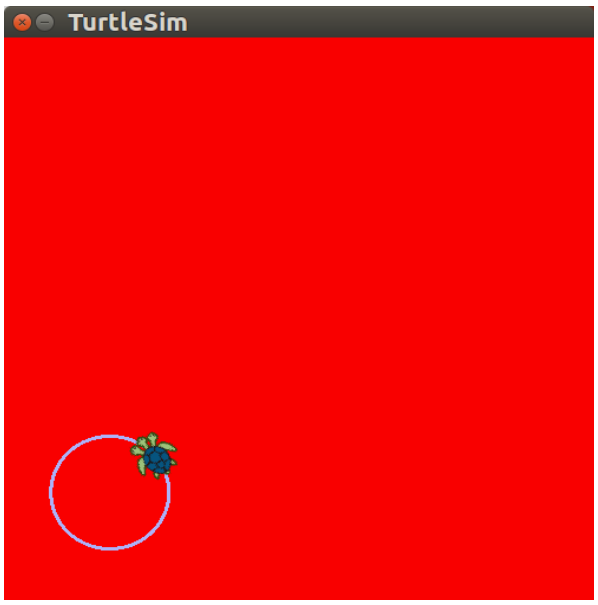


Figure 6 Turtle Running in A Circle

rostopic hz

Show the rate in Hz for publication (Cntl-C to stop data):

rostopic hz /turtle1/pose
tlharmanphd@D125-43873:/$ **rostopic hz /turtle1/pose**

subscribed to [/turtle1/pose]
average rate: 62.501
min: 0.016s max: 0.016s std dev: 0.00014s window: 62
average rate: 62.501
min: 0.016s max: 0.016s std dev: 0.00014s window: 124
average rate: 62.504
min: 0.016s max: 0.016s std dev: 0.00014s window: 187
average rate: 62.500
min: 0.016s max: 0.016s std dev: 0.00014s window: 249
average rate: 62.496
min: 0.015s max: 0.017s std dev: 0.00014s window: 300

Output at about a 60 Hz rate. Updated every 16 ms.

rqt_plot
We can plot information about the nodes and topics.

tlharmanphd@D125-43873:~$ **rqt_plot**

Select plotting type:

Figure 7 Selection of Plotting for rqt_plot

Experiment with different plot types and controls allowed for the plot such as changing the scales, etc.

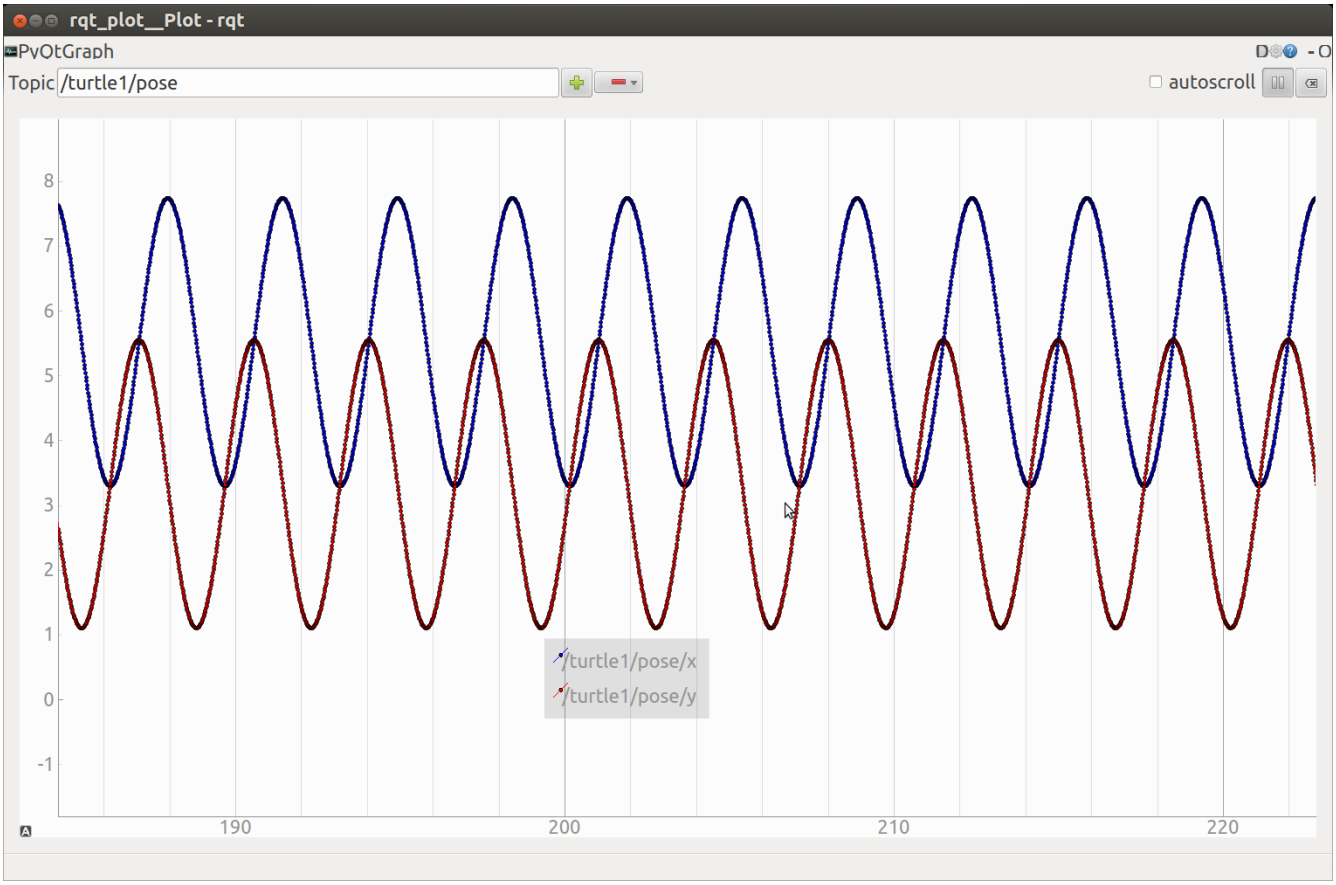


Figure 8 Plot of /turtle1/pose/x and /pose/y

Period of just over 3 seconds for 360 degree rotation. Note the periodic motion in x and y. Right click to change values for axes, etc.

With this plot, right click to set the axes ranges and other aspects of the plot. The pose has five values as shown before, but we have chosen to only plot the x and y variations as the turtle moves in a circle.

Choosing only x and y positions and experimenting with scales and autoscroll. See the tutorial for further help.

http://wiki.ros.org/rqt_plot

To plot from the command line, both of the following lines plot the same topics according to the wiki.

```
$ rqt_plot /turtle1/pose/x:y:z

$ rqt_plot /turtle1/pose/x /turtle1/pose/y /turtle1/pose/z
```

Obviously, if you want to change the topics to plot, you need to restart the program and give the new topic names.

ENABLE KEYBOARD CONTROL OF TURTLE

In a third window, we execute a node that allows keyboard control of the turtle. Roscore is running in one window and turtlesim_node in another.

rosrun turtlesim turtle_teleop_key

tlharmanphd@D125-43873:~$ **rosrun turtlesim turtle_teleop_key**

```
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Up arrow       Turtle up
Down arrow     Turtle down
Right arrow    Rotate CW
Left  arrow    Rotate CCW
```

tlharmanphd@D125-43873:~$ **rosnode list**

```
/rosout
/rqt_gui_py_node_22321
/teleop_turtle
/turtlesim
```

New Node /teleop_turtle

tlharmanphd@D125-43873:~$ **rosnode info /teleop_turtle**

```
--------------------------------------------------------------------------------
Node [/teleop_turtle]
Publications:
 * /turtle1/cmd_vel [geometry_msgs/Twist]
 * /rosout [rosgraph_msgs/Log]

Subscriptions: None

Services:
 * /teleop_turtle/get_loggers
 * /teleop_turtle/set_logger_level

contacting node http://D125-43873:44984/ ...
Pid: 22585
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
 * topic: /turtle1/cmd_vel
    * to: /turtlesim
    * direction: outbound
* transport: TCPROS
```

Notice publication of /turtle1/cmd_vel [geometry_msgs/Twist]

Node /turtlesim after /teleop_turtle

tlharmanphd@D125-43873:~$ **rosnode info /turtlesim**

```
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:
 * /turtle1/color_sensor [turtlesim/Color]
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
 * /turtle1/teleport_absolute
 * /reset
 * /clear
 * /turtle1/teleport_relative
 * /kill
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level
 * /spawn
 * /turtle1/set_pen

contacting node http://D125-43873:36624/ ...
Pid: 22605
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
 * topic: /turtle1/pose
    * to: /rqt_gui_py_node_22321
    * direction: outbound
    * transport: TCPROS
 * topic: /turtle1/cmd_vel
    * to: /teleop_turtle (http://D125-43873:44984/)
    * direction: inbound
* transport: TCPROS
```

Note: New topic /turtle1/cmd_vel to /teleop_turtle

To move turtle with arrow keys, be sure the focus is on the window that started turtle_teleop_key.

Figure 9 Turtlesim After Moving

We start a fourth terminal window to view the information that is available through ROS for the Turtlesim. The commands in that window elicit data while the other windows keep the turtle active. To move the turtle, use window three.

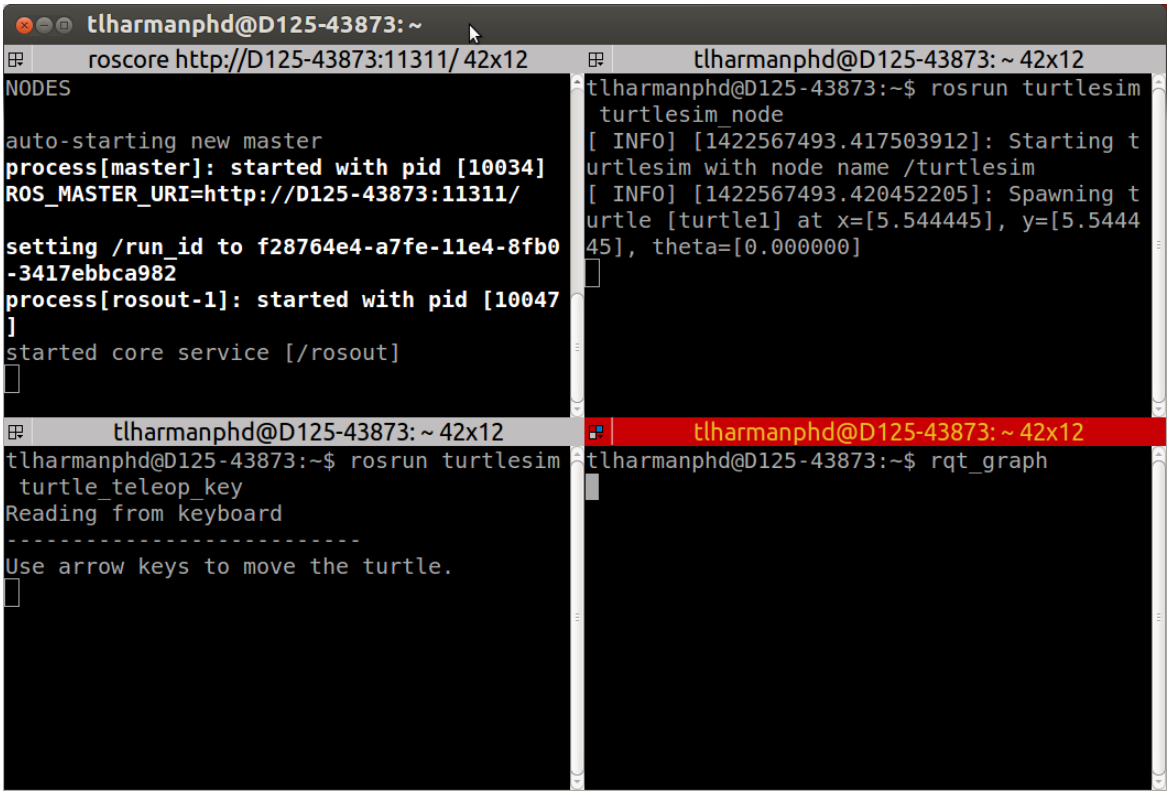


Figure 10 Four Turtlesim Windows using Terminator

The screen with four windows was created using Terminator. It is downloaded in Ubuntu from the Software Center Icon on the launcher: http://en.wikipedia.org/wiki/Ubuntu_Software_Center

The terminator is described at this site: https://apps.ubuntu.com/cat/applications/terminator/

1. List the ROS parameters to get information about the ROS nodes. The nodes are generally the executable scripts in ROS.
2. Determine what information you can get for the node turtlesim.

(Publications and Subscriptions)
tlharmanphd@D125-43873:~$ **rostopic list**

/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose

One important topic is `/turtle1/cmd_vel` which will be **published** using the keyboard or by publishing the topic with the `rostopic pub` command.

Determine data from Topic /turtle1/cmd_vel in Indigo

The **rostopic echo** command shows the data sent by the node to control the turtle. As you move the turtle, the data are updated. As you press the arrow keys the displayed values will change: x velocity if linear motion, z velocity if rotation.

tlharmanphd@D125-43873:~$ **rostopic echo /turtle1/cmd_vel**

```
linear:
  x: 2.0            (Velocity ahead)
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: -2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 2.0    (Counter Clockwise Rotational velocity about z axis – out of window)
---
.
.
```

These show the parameters for **cmd_vel** which are linear velocity and angular velocity. In this result, the turtle was moved linearly until the last output which shows a rotation.

To find turtle's position in window use /turtle1/pose

tlharmanphd@D125-43873:~$ **rostopic echo /turtle1/pose**

```
x: 5.544444561
y: 5.544444561
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
---
.
.
```

CNTL+c to stop output. Here the turtle is at rest in the center of the window.

If you return to the teleop_key window and move the turtle with the arrow keys you can see the output of the pose message (turtlesim/Pose) change. Remember the format:

tlharmanphd@D125-43873:~$ **rosmsg show turtlesim/Pose**

```
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

We can make the turtle turn in a circle by **publishing** the topic **/turtle1/command_velocity** as shown before using the node **/turtlesim**.

$**rostopic pub -r 1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'** for Indigo

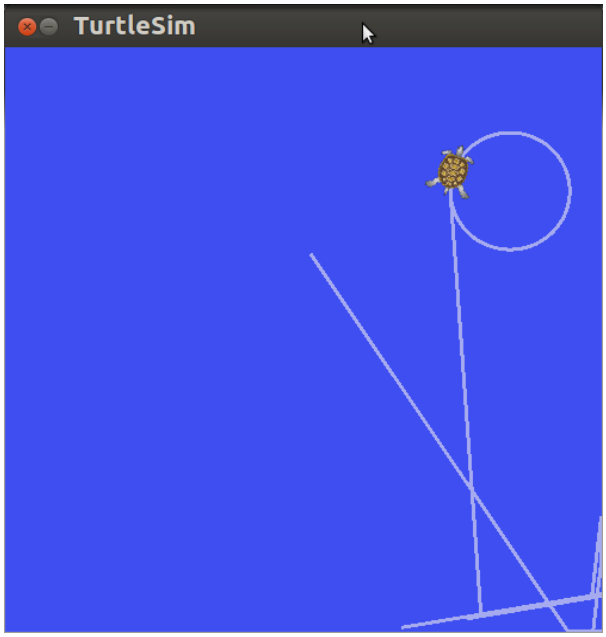


Figure 11 Turtle responds to published topic

The command will publish at a rate (`-r`) of once a second (1 Hz). The topic `/turtle1/command_velocity` is followed by the message type `turtlesim/Velocity` that commands the turtle to turn with linear velocity 2.0 and angular velocity 1.8 according to the ROS tutorial:

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

As noted before, a turtlesim/Velocity message has two floating point elements : `linear` and `angular`. In this case, `2.0` becomes the linear value, and `1.8` is the `angular` value. These arguments are actually in YAML syntax, which is described more in the YAML command line documentation.

Clear the screen
When you want to CLEAR THE SCREEN
tlharmanphd@D125-43873:~$ **rosservice call /clear**

rqt_graph

There is another feature of ROS that is useful for those who wish to see a graphical view of the communication between nodes. We know that `/teleop_turtle` node **publishes** a message on the topic called `/turtle1/command_velocity` and the node `/Turtlesim` **subscribes** to those messages.

This can be shown in a graphical form with the command:

tlharmanphd@D125-43873:~$ **rqt_graph**

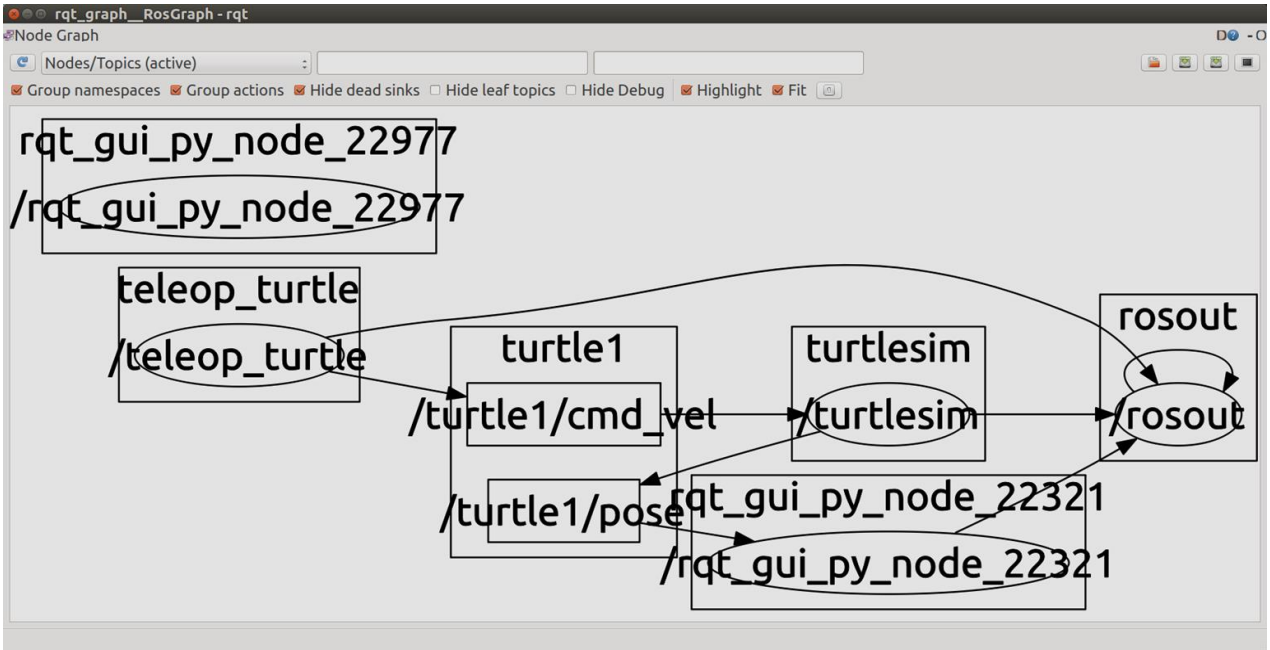


Figure 12 Turtlesim graph showing communication

The advantage of Turtlesim is as follows:

1. Easy to Learn and Use
2. Shows basic ROS capability
3. Can be downloaded with ROS for use on a laptop

To restrict the graph to the /turtlesim node:

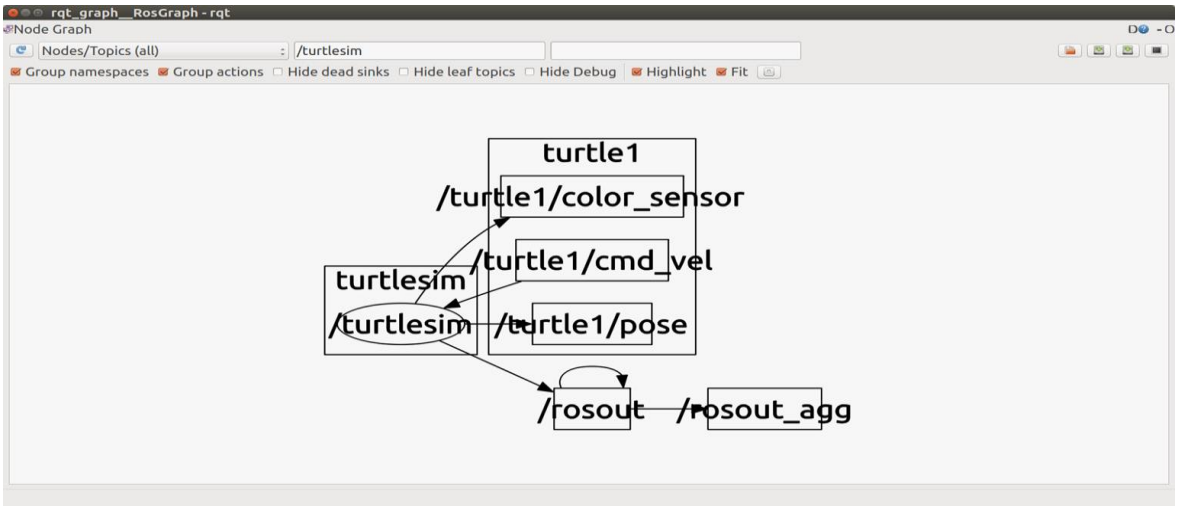


Figure 13 RQT_GRAPH for /turtlesim

PYTHON and TURTLESIM

LETS CONTROL THE TURTLE- Publish to /turtle1/cmd_vel:
(roscore and turtlesim_node running)

Create a Python script turtlesim1.py and make executable ($ chmod +x turtlesim1.py).

$ python turtlesim1.py (Make Executable $chmod +x turtlesim1.py)

```
[INFO] [WallTime: 1455313387.186692]  Press CTRL+c to stop TurtleBot
[INFO] [WallTime: 1455313387.188315]  Set rate 10Hz
```

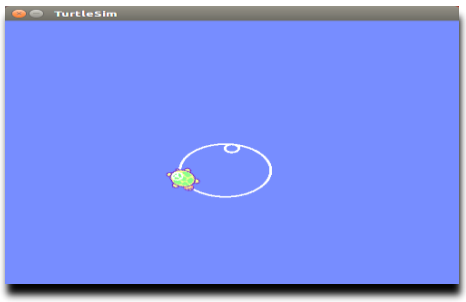
```
#!/usr/bin/env python   turtlesim1.py
# Execute as a python script
# Set linear and angular values of Turtlesim's speed and turning.
import rospy                                # Needed to create a ROS node
from geometry_msgs.msg import Twist    # Message that moves base

class ControlTurtlesim():
    def __init__(self):
        # ControlTurtlesim is the name of the node sent to the master
        rospy.init_node('ControlTurtlesim', anonymous=False)

                  # Message to screen
        rospy.loginfo(" Press CTRL+c to stop TurtleBot")

        # Keys CNTL + c will stop script
        rospy.on_shutdown(self.shutdown)

         # Publisher will send Twist message on topic
        # /turtle1/cmd_vel

        self.cmd_vel = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

              # Turtlesim will receive the message 10 times per second.
        rate = rospy.Rate(10);
              # 10 Hz is fine as long as the processing does not exceed
        #   1/10 second.
        rospy.loginfo(" Set rate 10Hz")
        # Twist is geometry_msgs for linear and angular velocity
        move_cmd = Twist()
              # Linear speed in x in meters/second is + (forward) or
        # - (backwards)
        move_cmd.linear.x = 0.3    # Modify this value to change speed
              # Turn at 0 radians/s
        move_cmd.angular.z = 0
        # Modify this value to cause rotation rad/s

              # Loop and TurtleBot will move until you type CNTL+c
        while not rospy.is_shutdown():
                  # publish Twist values to the Turtlesim node /cmd_vel
            self.cmd_vel.publish(move_cmd)
                  # wait for 0.1 seconds (10 HZ) and publish again
            rate.sleep()


    def shutdown(self):
        # You can stop turtlebot by publishing an empty Twist message
        rospy.loginfo("Stopping Turtlesim")
```

```
        self.cmd_vel.publish(Twist())
             # Give TurtleBot time to stop
        rospy.sleep(1)

if __name__ == '__main__':
    try:
        ControlTurtlesim()
    except:
        rospy.loginfo("End of the trip for Turtlesim")

$ rosnode list
         /ControlTurtlesim
         /rosout
         /teleop_turtle
         /turtlesim
```

Change Python script turtlesim1.py to run turtle in a circle (Make executable)

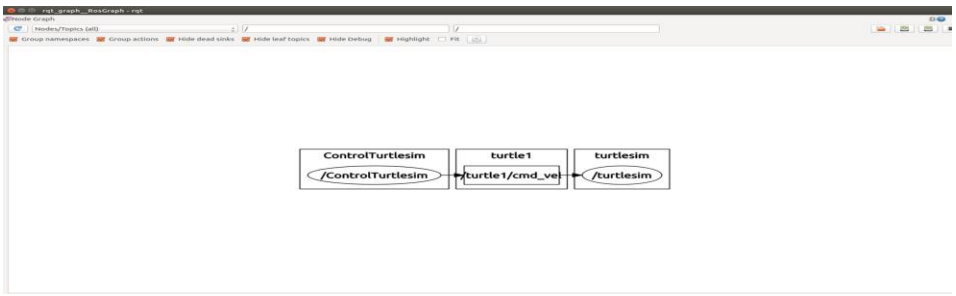
tlharmanphd@D125-43873:~/ros_ws$ **python turtlesim2.py**

[INFO] [WallTime: 1455383932.566053] Press CTRL+c to stop TurtleBot
[INFO] [WallTime: 1455383932.567306] Set rate 10Hz
^C[INFO] [WallTime: 1455383937.538077] Stopping Turtlesim



In turtlesim2.py Script

```
move_cmd = Twist()
        # Linear speed in x in meters/second is + (forward) or
    # - (backwards)
    move_cmd.linear.x = 2.0        # Modify this value to change speed
        # Turn at 1.8 radians/s
    move_cmd.angular.z = 1.8
    # Modify this value to cause rotation rad/s
```



ROSBAG

rosbag help

tlharmanphd@D125-43873:/$ **rosbag help**
Usage: rosbag <subcommand> [options] [args]

A bag is a file format in ROS for storing ROS message data. The rosbag command can record, replay and manipulate bags.

Available subcommands:

check	Determine whether a bag is playable in the current system, or if it can be migrated.
compress	Compress one or more bag files.
decompress	Decompress one or more bag files.
filter	Filter the contents of the bag.
fix	Repair the messages in a bag file so that it can be played in the current system.
help	
info	Summarize the contents of one or more bag files.
play	Play back the contents of one or more bag files in a time-synchronized fashion.
record	Record a bag file with the contents of specified topics.
reindex	Reindexes one or more bag files.

For additional information, see http://wiki.ros.org/rosbag

Table 2 ROS Help Information

A bag is a file format in ROS for storing ROS message data. The rosbag command can record, replay and manipulate bags.

rosbag help

Usage: rosbag <subcommand> [options] [args]

Available subcommands:

check	Determine whether a bag is playable in the current system, or if it can be migrated.
compress	Compress one or more bag files.
decompress	Decompress one or more bag files.
filter	Filter the contents of the bag.
fix	Repair the messages in a bag file so that it can be played in the current system.
help	
info	Summarize the contents of one or more bag files.
play	Play back the contents of one or more bag files in a time-synchronized fashion.
record	Record a bag file with the contents of specified topics.
reindex	Reindexes one or more bag files.

We will use the record and play option to learn how to save and replay messages.

For additional information, see http://wiki.ros.org/rosbag

References that describe the rosbag commands in more detail:

http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data

http://wiki.ros.org/rosbag/Commandline

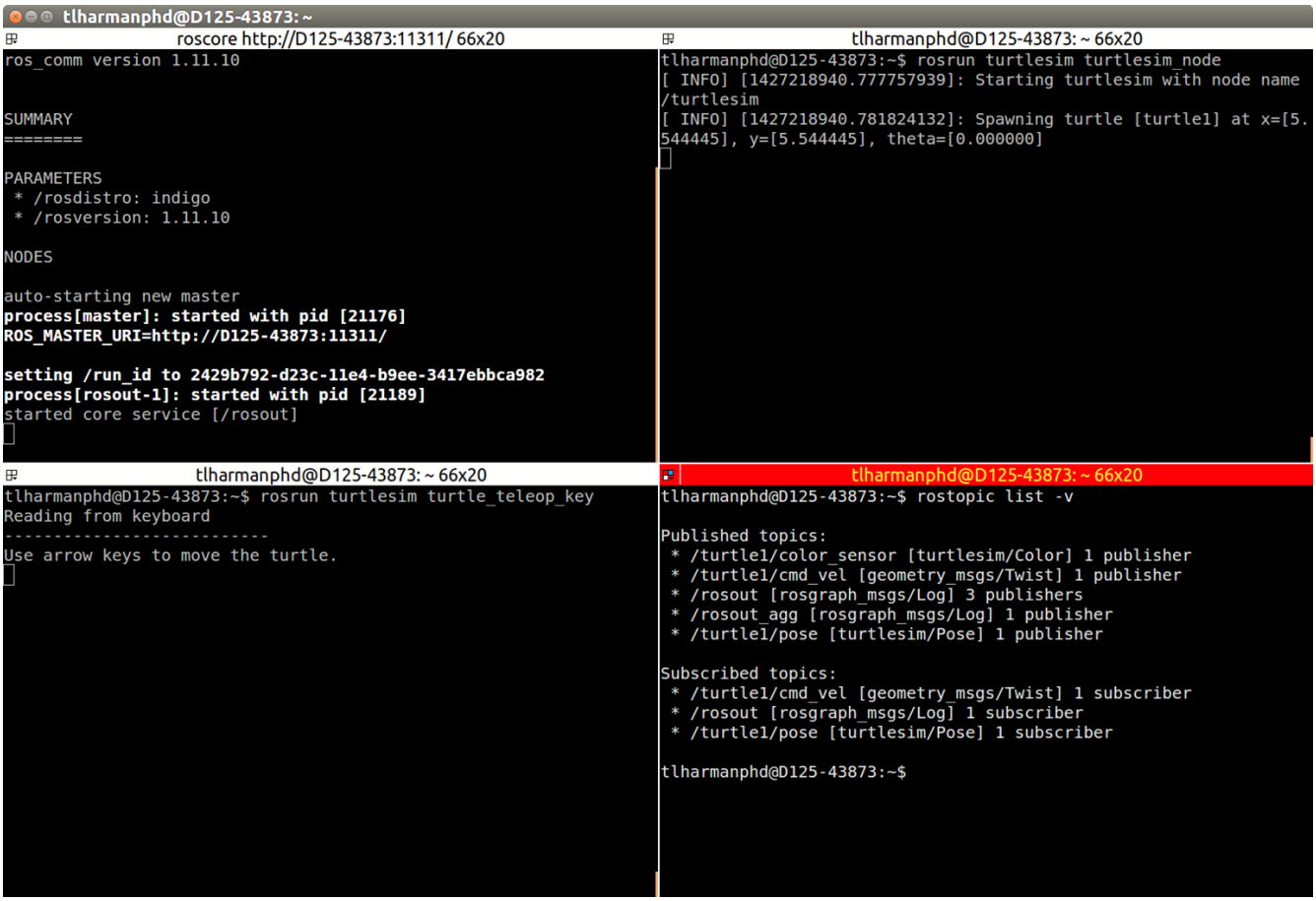


Figure 14 Windows for turtlesim

tlharmanphd@D125-43873:~$ **pwd**
/home/tlharmanphd
tlharmanphd@D125-43873:~$ **mkdir bagfilesturtle**
tlharmanphd@D125-43873:~$ **ls -d b***
backup bagfilesturtle baxter.sh~

Here we are making a temporary directory to record data.

Then running rosbag record command with the option –a indicates that all published topics will be accumulated in a bag file.

Start to record the topics with the **rosbag record -a** command:

tlharmanphd@D125-43873:~/bagfilesturtle$ **rosbag record -a**

```
[ INFO] [1427220792.012510086]: Recording to 2015-03-24-13-13-12.bag.
[ INFO] [1427220792.012714289]: Subscribing to /turtle1/color_sensor
[ INFO] [1427220792.015024218]: Subscribing to /turtle1/cmd_vel
[ INFO] [1427220792.017232168]: Subscribing to /rosout
[ INFO] [1427220792.019675036]: Subscribing to /rosout_agg
[ INFO] [1427220792.021687650]: Subscribing to /turtle1/pose
```

Now change the focus to the teleop_key window move turtle with arrow keys for 10 or so seconds.

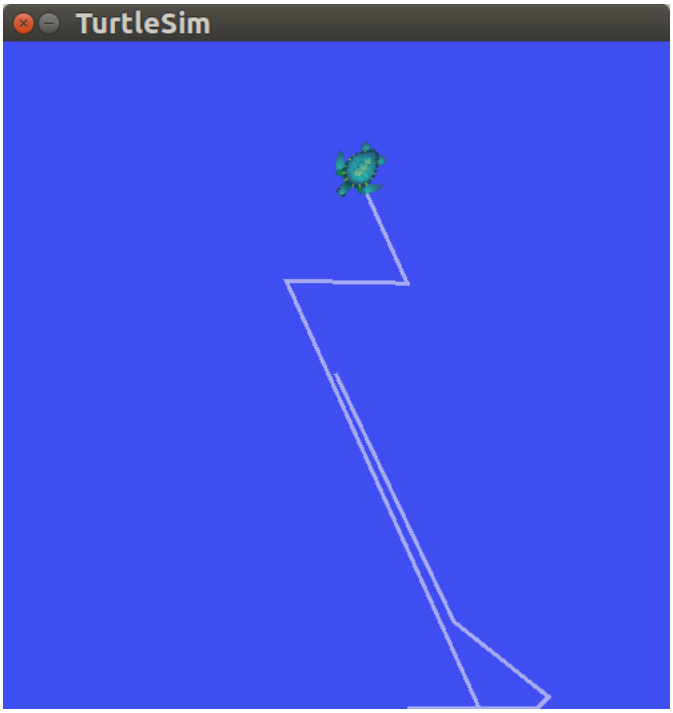


Figure 15 Turtle moved with keyboard keys with rosbag recording

In the window running rosbag record, exit with a Ctrl-C when you have finished moving the turtle. Now examine the contents of the directory **bagfilesturtle**. You should see a file with a name that begins with the year, data, and time and the suffix .bag. This is the bag file that contains all topics published by any node in the time that rosbag record was running.

Now that we've recorded a bag file using rosbag record option we can examine it and play it back using the commands rosbag info and rosbag play. First we are going to see what's recorded in the bag file.

rosbag info

tlharmanphd@D125-43873:~/bagfilesturtle$ **ls**
2015-03-24-13-13-12.bag

Here the name is the date and time.

tlharmanphd@D125-43873:~/bagfilesturtle$ **rosbag info 2015-03-24-13-13-12.bag**

```
path:        2015-03-24-13-13-12.bag
version:     2.0
duration:    1:22s (82s)
start:       Mar 24 2015 13:13:12.02 (1427220792.02)
end:         Mar 24 2015 13:14:34.58 (1427220874.58)
size:        823.2 KB
messages:    10736
compression: none [1/1 chunks]
types:    geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
          rosgraph_msgs/Log   [acffd30cd6b6de30f120938c17c593fb]
          turtlesim/Color     [353891e354491c51aabe32df673fb446]
          turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:   /rosout               160 msgs    : rosgraph_msgs/Log   (2 connections)
          /rosout_agg           156 msgs    : rosgraph_msgs/Log
          /turtle1/cmd_vel      130 msgs    : geometry_msgs/Twist
          /turtle1/color_sensor 5145 msgs   : turtlesim/Color
          /turtle1/pose         5145 msgs   : turtlesim/Pose
```

This tells us topic names and types as well as the number (count) of each message topic contained in the bag file. We can see that of the topics being advertised that we saw in the rostopic output, four of the five were actually published over our recording interval. As we ran rosbag record with the -a flag it recorded all messages published by all nodes.

The next step in this tutorial is to replay the bag file to reproduce behavior in the running system. First kill the teleop program that may be still running from the previous section - Ctrl-c in the terminal where you started turtle_teleop_key.

rosbag play

Leave turtlesim running or restart with a “fresh” turtle.

tlharmanphd@D125-43873:~$ **rosrun turtlesim turtlesim_node**

```
[ INFO] [1427221332.211909961]: Starting turtlesim with node name /turtlesim
[ INFO] [1427221332.225487283]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445],
theta=[0.000000]
```

In a terminal window run the following command in the directory where you took the original bag file:

tlharmanphd@D125-43873:~/bagfilesturtle$ **rosbag play 2015-03-24-13-13-12.bag**

```
[ INFO] [1427221486.993700128]: Opening 2015-03-24-13-13-12.bag
```

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
 [RUNNING] Bag Time: 1427220874.545656   Duration: 82.521750 / 82.553575
Done.

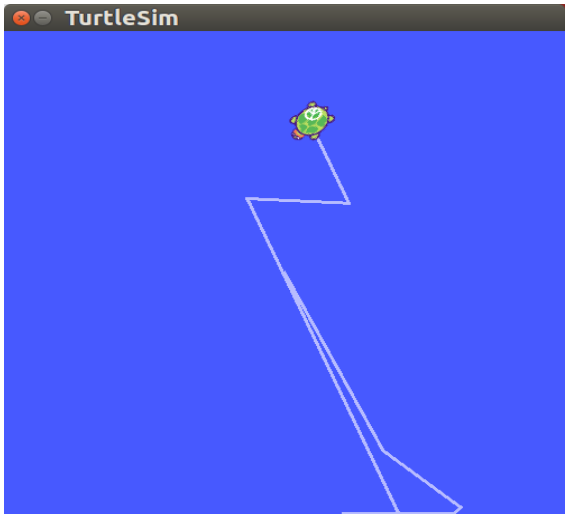


Figure 16 Turtle Replay of rosbag data

Turtle begins executing messages from its last location.

In its default mode rosbag play will wait for a certain period (.2 seconds) after advertising each message before it actually begins publishing the contents of the bag file. Waiting for some duration allows any subscriber of a message to be alerted that the message has been advertised and that messages may follow. If rosbag play publishes messages immediately upon advertising, subscribers may not receive the first several published messages. The waiting period can be specified with the -d option.

Eventually the topic /turtle1/command_velocity will be published and the turtle should start moving in turtlesim in a pattern similar to the one you executed from the teleop program. The duration between running rosbag play and the turtle moving should be approximately equal to the time between the original rosbag record execution and issuing the commands from the keyboard in the beginning part of the tutorial. You can have rosbag play not start at the beginning of the bag file but instead start some duration past the beginning using the -s argument. A final option that may be of interest is the -r option, which allows you to change the rate of publishing by a specified factor. If you execute:

```
rosbag play -r 2 <your bagfile>
```

You should see the turtle execute a slightly different trajectory - this is the trajectory that would have resulted had you issued your keyboard commands twice as fast.

After - the motion will start on playback from the current position of the turtle.

tlharmanphd@D125-43873:~/bagfilesturtle$ **rosbag play -r2 2015-03-24-13-13-12.bag**
[INFO] [1427221716.127268792]: Opening 2015-03-24-13-13-12.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
[RUNNING] Bag Time: 1427220874.545836 Duration: 82.521930 / 82.553575
Done.

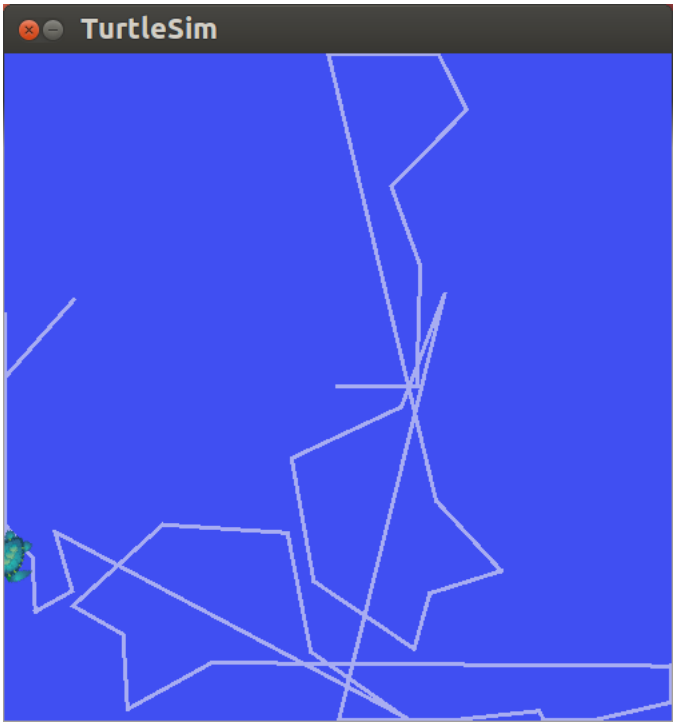


Figure 17 Turtle rosbag replay at 2x speed

Recording a subset of the data
When running a complicated system, such as the pr2 software suite, there may be hundreds of topics being published, with some topics, like camera image streams, potentially publishing huge amounts of data. In such a system it is often impractical to write log files consisting of all topics to disk in a single bag file. The rosbag record command supports logging only particular topics to a bag file, allowing a user to only record the topics of interest to them.

To name the bag file and selectively record

(This option is the letter O)

tlharmanphd@D125-43873:~/bagfilesturtle$ **rosbag record -O cmdvel /turtle1/cmd_vel /turtle1/pose**

```
[ INFO] [1427222327.911823890]: Subscribing to /turtle1/cmd_vel
[ INFO] [1427222327.914523800]: Subscribing to /turtle1/pose
[ INFO] [1427222327.917503556]: Recording to cmdvel.bag.
```

tlharmanphd@D125-43873:~/bagfilesturtle$ **ls**

```
2015-03-24-13-13-12.bag  cmdvel.bag
```

Move the turtle with the keys with focus on the teleop window. The -O argument tells rosbag record to log to a file named subset.bag, and the topic arguments cause rosbag record to only subscribe to these two topics. Move the turtle around for several seconds using the keyboard arrow commands, and then Ctrl-c in the rosbag window to stop the rosbag record.

tlharmanphd@D125-43873:~/bagfilesturtle$ **rosbag info cmdvel.bag**

```
path:        cmdvel.bag
version:     2.0
duration:    1:01s (61s)
start:       Mar 24 2015 13:38:48.20 (1427222328.20)
end:         Mar 24 2015 13:39:49.94 (1427222389.94)
size:        311.4 KB
messages:    3972
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /turtle1/cmd_vel    112 msgs    : geometry_msgs/Twist
             /turtle1/pose       3860 msgs    : turtlesim/Pose
```

tlharmanphd@D125-43873:~/bagfilesturtle$ **rosbag play cmdvel.bag**

```
[ INFO] [1427222827.531968073]: Opening cmdvel.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
 [RUNNING]  Bag Time: 1427222389.908203   Duration: 61.712115 / 61.743916
Done.
```

WATCH THE TURTLE MOVE!

Figure 18 Turtle moving with subset of rosbag data

The limitations of rosbag record/play

In the previous section you may have noted that the turtle's path may not have exactly mapped to the original keyboard input - the rough shape should have been the same, but the turtle may not have exactly tracked the same path. The reason for this is that the path tracked by turtlesim is very sensitive to small changes in timing in the system, and rosbag is limited in its ability to exactly duplicate the behavior of a running system in terms of when messages are recorded and processed by rosrecord, and when messages are produced and processed when using rosplay. For nodes like turtlesim, where minor timing changes in when command messages are processed can subtly alter behavior, the user should not expect perfectly mimicked behavior.

APPENDIX I. REFERENCES

GETTING STARTED WITH TURTLESIM

http://wiki.ros.org/turtlesim

GENTLE INTRODUCTION O'KANE CHAPTER 2

http://www.cse.sc.edu/~jokane/agitr/agitr-letter-start.pdf

TUTORIALS USING TURTLESIM – A LIST

http://wiki.ros.org/turtlesim/Tutorials

ROS CONCEPTS

ROS has three levels of concepts: the Filesystem level, the Computation Graph level, and the Community level. These levels and concepts are summarized below and later sections go into each of these in greater detail.

The filesystem level concepts mainly cover ROS resources that you encounter on disk, such as packages, metapackages, manifests, repositories, messages, and services

The *Computation Graph* is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are *nodes*, *Master*, *Parameter Server*, *messages*, *services*, *topics*, and *bags*, all of which provide data to the Graph in different ways.

The ROS Community Level concepts are ROS resources that enable separate communities to exchange software and knowledge. These resources include distributions, repositories, ROS wiki, ROS answers, and a Blog.

In addition to the three levels of concepts, ROS also defines two types of names -- Package Resource Names and Graph Resource Names -- which are discussed below.

http://wiki.ros.org/ROS/Concepts

ROSCORE

From the ROS tutorial http://wiki.ros.org/roscore

roscore is 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. It is launched using the `roscore` command.

ROS MASTER

The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.
http://wiki.ros.org/Master

Clearpath diagram of Master

http://www.clearpathrobotics.com/blog/how-to-guide-ros-101/

ROS NODES AND TURTLESIM

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

ROS TOPICS AND TURTLESIM

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROSSERVICE
rosservice contains the rosservice command-line tool for listing and querying ROS Services
http://wiki.ros.org/rosservice

ROSSERVICE AND ROS SERVICE PARAMETERS
This tutorial introduces ROS services, and parameters as well as using the rosservice and rosparam commandline tools.
http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

http://wiki.ros.org/Parameter%20Server

http://wiki.ros.org/rosparam

http://www.cse.sc.edu/~jokane/agitr/agitr-small-param.pdf (Chapter 7 of O'Kane)

ROSSERVICE AND ROS TELEPORT PARAMETER

Let's bring the turtle to a known starting point using absolute teleportation. Its inputs are [x y theta]. The origin [0 0 0] is offscreen so we will start with [1 1 0]. The turtle should be facing to the right (0*).

```
rosservice call /turtle1/teleport_absolute 1 1 0
```
https://sites.google.com/site/ubrobotics/ros-documentation

USING RQT_PLOT, RQT_CONSOLE AND ROSLAUNCH WITH TURTLESIM
http://wiki.ros.org/rqt_plot

This tutorial introduces ROS using rqt_console and rqt_logger_level for debugging and roslaunch for starting many nodes at once.
http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch

ROSBAG TURTLESIM EXAMPLE
This tutorial will teach you how to record data from a running ROS system into a .bag file, and then to play back the data to produce similar behavior in a running system.

Keywords: data, rosbag, record, play, info, bag

TURTLESIM EXAMPLE
http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data/

DATA LOGGING USING ROSBAG
http://www.fer.unizg.hr/_download/repository/p08-rosbag.pdf

INTRODUCTION TO TF AND TURTLESIM
This tutorial will give you a good idea of what tf can do for you. It shows off some of the tf power in a multi-robot example using turtlesim. This also introduces using tf_echo, view_frames, rqt_tf_tree, and rviz.
http://wiki.ros.org/tf/Tutorials/Introduction%20to%20tf/

YAML Command LINE

Several ROS tools (rostopic, rosservice) use the YAML markup language on the command line. YAML was chosen as, in most cases, it offers a very simple, nearly markup-less solution to typing in typed parameters.

For a quick overview of YAML, please see YAML Overview.

http://wiki.ros.org/ROS/YAMLCommandLine

APPENDIX II. TURTLESIM MANIFEST (PACKAGE.XML)

tlharmanphd@D125-43873:~$ **gedit /opt/ros/indigo/share/turtlesim/package.xml**

```
<?xml version="1.0"?>
<package>
  <name>turtlesim</name>
  <version>0.5.2</version>
  <description>
    turtlesim is a tool made for teaching ROS and ROS packages.
  </description>
  <maintainer email="dthomas@osrfoundation.org">Dirk Thomas</maintainer>
  <license>BSD</license>

  <url type="website">http://www.ros.org/wiki/turtlesim</url>
  <url type="bugtracker">https://github.com/ros/ros_tutorials/issues</url>
  <url type="repository">https://github.com/ros/ros_tutorials</url>
  <author>Josh Faust</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>geometry_msgs</build_depend>
  <build_depend>libqt4-dev</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>qt4-qmake</build_depend>
  <build_depend>rosconsole</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>roscpp_serialization</build_depend>
  <build_depend>roslib</build_depend>
  <build_depend>rostime</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>std_srvs</build_depend>

  <run_depend>geometry_msgs</run_depend>
  <run_depend>libqt4</run_depend>
  <run_depend>message_runtime</run_depend>
  <run_depend>rosconsole</run_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>roscpp_serialization</run_depend>
  <run_depend>roslib</run_depend>
  <run_depend>rostime</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>std_srvs</run_depend>
</package>
```

APPENDIX III. TURTLESIM DIRECTORIES AND FILES

tlharmanphd@D125-43873:~$ **locate turtlesim**
/home/ceng5931/Documents/simple turtlesim~
/home/ceng5931/Documents/how to/How to run turtlesim
/home/fairchildc/Desktop/Turtlesim/turtlesim .odt
/home/fairchildc/Desktop/baxter 2_12_2015/work on turtlesim 2_12_2015.odt
/home/louiseli/Desktop/How To/How to run turtlesim Indigo
/home/louiseli/Desktop/How To/How to run turtlesim Indigo~
/home/louiseli/Desktop/How To/How to run turtlesimGroovy
/home/louiseli/Desktop/How To/How to run turtlesim~
/home/tlharmanphd/Desktop/0_BaxterFrom Office/Copied/UsersGuide/turtlesimFiles1_23_2015A.docx
/home/tlharmanphd/Desktop/Baxter Guides/turtlesimUpdatesIndigo.odt
/home/tlharmanphd/Guides_data/Turtlesim/turtlesimFiles.odt
/home/tlharmanphd/Guides_data/Turtlesim/turtlesimFiles1_23_2015.odt
/home/tlharmanphd/Guides_data/Turtlesim/turtlesimFiles1_23_2015A.docx
/home/tlharmanphd/Guides_data/Turtlesim/turtlesimwindows2015-01-29 14:18:27.png
/home/tlharmanphd/Videos/turtlesimNode 2015-02-21 16:01:00.png
/opt/ros/indigo/include/turtlesim
/opt/ros/indigo/include/turtlesim/Color.h
/opt/ros/indigo/include/turtlesim/Kill.h
/opt/ros/indigo/include/turtlesim/KillRequest.h
/opt/ros/indigo/include/turtlesim/KillResponse.h
/opt/ros/indigo/include/turtlesim/Pose.h
/opt/ros/indigo/include/turtlesim/SetPen.h
/opt/ros/indigo/include/turtlesim/SetPenRequest.h
/opt/ros/indigo/include/turtlesim/SetPenResponse.h
/opt/ros/indigo/include/turtlesim/Spawn.h
/opt/ros/indigo/include/turtlesim/SpawnRequest.h
/opt/ros/indigo/include/turtlesim/SpawnResponse.h
/opt/ros/indigo/include/turtlesim/TeleportAbsolute.h
/opt/ros/indigo/include/turtlesim/TeleportAbsoluteRequest.h
/opt/ros/indigo/include/turtlesim/TeleportAbsoluteResponse.h
/opt/ros/indigo/include/turtlesim/TeleportRelative.h
/opt/ros/indigo/include/turtlesim/TeleportRelativeRequest.h
/opt/ros/indigo/include/turtlesim/TeleportRelativeResponse.h
/opt/ros/indigo/lib/turtlesim
/opt/ros/indigo/lib/pkgconfig/turtlesim.pc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/__init__.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/__init__.pyc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/msg
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/msg/_Color.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/msg/_Color.pyc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/msg/_Pose.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/msg/_Pose.pyc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/msg/__init__.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/msg/__init__.pyc

/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_Kill.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_Kill.pyc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_SetPen.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_SetPen.pyc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_Spawn.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_Spawn.pyc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_TeleportAbsolute.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_TeleportAbsolute.pyc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_TeleportRelative.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/_TeleportRelative.pyc
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/__init__.py
/opt/ros/indigo/lib/python2.7/dist-packages/turtlesim/srv/__init__.pyc
/opt/ros/indigo/lib/turtlesim/draw_square
/opt/ros/indigo/lib/turtlesim/mimic
/opt/ros/indigo/lib/turtlesim/turtle_teleop_key
/opt/ros/indigo/lib/turtlesim/turtlesim_node
/opt/ros/indigo/share/turtlesim
/opt/ros/indigo/share/common-lisp/ros/turtlesim
/opt/ros/indigo/share/common-lisp/ros/turtlesim/msg
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv
/opt/ros/indigo/share/common-lisp/ros/turtlesim/msg/Color.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/msg/Pose.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/msg/_package.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/msg/_package_Color.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/msg/_package_Pose.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/msg/turtlesim-msg.asd
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/Kill.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/SetPen.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/Spawn.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/TeleportAbsolute.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/TeleportRelative.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/_package.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/_package_Kill.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/_package_SetPen.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/_package_Spawn.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/_package_TeleportAbsolute.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/_package_TeleportRelative.lisp
/opt/ros/indigo/share/common-lisp/ros/turtlesim/srv/turtlesim-srv.asd
/opt/ros/indigo/share/turtlesim/cmake
/opt/ros/indigo/share/turtlesim/images
/opt/ros/indigo/share/turtlesim/msg
/opt/ros/indigo/share/turtlesim/package.xml
/opt/ros/indigo/share/turtlesim/srv
/opt/ros/indigo/share/turtlesim/cmake/turtlesim-msg-extras.cmake
/opt/ros/indigo/share/turtlesim/cmake/turtlesim-msg-paths.cmake
/opt/ros/indigo/share/turtlesim/cmake/turtlesimConfig-version.cmake
/opt/ros/indigo/share/turtlesim/cmake/turtlesimConfig.cmake
/opt/ros/indigo/share/turtlesim/images/box-turtle.png
/opt/ros/indigo/share/turtlesim/images/diamondback.png

/opt/ros/indigo/share/turtlesim/images/electric.png
/opt/ros/indigo/share/turtlesim/images/fuerte.png
/opt/ros/indigo/share/turtlesim/images/groovy.png
/opt/ros/indigo/share/turtlesim/images/hydro.png
/opt/ros/indigo/share/turtlesim/images/hydro.svg
/opt/ros/indigo/share/turtlesim/images/indigo.png
/opt/ros/indigo/share/turtlesim/images/indigo.svg
/opt/ros/indigo/share/turtlesim/images/palette.png
/opt/ros/indigo/share/turtlesim/images/robot-turtle.png
/opt/ros/indigo/share/turtlesim/images/sea-turtle.png
/opt/ros/indigo/share/turtlesim/images/turtle.png
/opt/ros/indigo/share/turtlesim/msg/Color.msg
/opt/ros/indigo/share/turtlesim/msg/Pose.msg
/opt/ros/indigo/share/turtlesim/srv/Kill.srv
/opt/ros/indigo/share/turtlesim/srv/SetPen.srv
/opt/ros/indigo/share/turtlesim/srv/Spawn.srv
/opt/ros/indigo/share/turtlesim/srv/TeleportAbsolute.srv
/opt/ros/indigo/share/turtlesim/srv/TeleportRelative.srv
/usr/share/doc/ros-indigo-turtlesim
/usr/share/doc/ros-indigo-turtlesim/changelog.Debian.gz
/var/lib/dpkg/info/ros-indigo-turtlesim.list
/var/lib/dpkg/info/ros-indigo-turtlesim.md5sums
tlharmanphd@D125-43873:~$ l

03/25/15

tlharmanphd@D125-43873:~$ **cd /opt/ros/indigo/lib/turtlesim**
tlharmanphd@D125-43873:/opt/ros/indigo/lib/turtlesim$ **ls -la**
total 400
drwxr-xr-x   2 root root   4096 Mar 16 20:56 .
drwxr-xr-x 105 root root 20480 Mar 18 15:47 ..
-rwxr-xr-x   1 root root 72248 Feb 20 13:18 draw_square
-rwxr-xr-x   1 root root 59832 Feb 20 13:18 mimic
-rwxr-xr-x   1 root root 220920 Feb 20 13:18 turtlesim_node
-rwxr-xr-x   1 root root 27112 Feb 20 13:18 turtle_teleop_key
tlharmanphd@D125-43873:/opt/ros/indigo/lib/turtlesim$

APPENDIX 1V. INDIGO VS GROOVY

SOME CHANGES FOR TURTLESIM UPDATE INDIGO (3/2015)

Go through all examples and also update References in the Appendix.

Command the turtle with cmd_vel (Not command_velocity)

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

($rostopic pub /turtle1/command_velocity turtlesim/Velocity -r 1 -- 4.0 -1.8 for groovy)

rostopic type for /turtle1/cmd_vel

$rostopic type /turtle1/cmd_vel //indigo
geometry_msgs/Twist

$rostopic type /turtle1/command_velocity //groovy
turtlesim/Velocity

$rostopic type /turtle1/pose
turtlesim/Pose

Echo cmd_vel to show pose

$rostopic echo /turtle1/cmd_vel //indigo
linear:
  x: 4.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.8

$rostopic echo /turtle1/command_velocity // groovy
linear: 2.0
angular: -1.79999995232

APPENDIX V. TURTLESIM CHEATSHEET

02/13/16 Turtlesim Cheat Sheet

1. $ roscore (leave running but minimize)

2. 2nd Terminal

3. $ **rosrun turtlesim turtlesim_node**
(See the turtle with Blue Background – leave terminal window running and view turtle)

4. 3rd Terminal
$ rosnode info turtlesim (Determine node information)

```
tlharmanphd@D125-43873:~$ rosnode info turtlesim
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:
 * /turtle1/color_sensor [turtlesim/Color]
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [unknown type]

Services:
 * /turtle1/teleport_absolute
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level
 * /reset
 * /spawn
 * /clear
 * /turtle1/set_pen
 * /turtle1/teleport_relative
 * /kill


contacting node http://D125-43873:41890/ ...
Pid: 7420
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
```

NOW YOU HAVE THE INFORMATION TO CONTROL TURTLESIM!

- - - - - -
Look at Colors
$ rostopic echo /turtle1/color_sensor (r: b: g: Cntl+C to stop)
- - - -
rosparam get parameters and change color of background to Red
tlharmanphd@D125-43873:~$ **rosparam get /**

```
background_b: 255
background_g: 86
background_r: 69
rosdistro: 'indigo
roslaunch:
  uris: {host_d125_43873__60512: 'http://D125-43873:60512/'}
rosversion: '1.11.10
run_id: 2429b792-d23c-11e4-b9ee-3417ebbca982
```

rosparam set
Change the colors:
tlharmanphd@D125-43873:/$ **rosparam set background_b 0**
tlharmanphd@D125-43873:/$ **rosparam set background_g 0**
tlharmanphd@D125-43873:/$ **rosparam set background_r 255**
tlharmanphd@D125-43873:/$ **rosservice call /clear (See Red!)**

- - - -
Services Absolute and Relative Move
$ rosservice call /turtle1/teleport_absolute 1 1 0 (Move to 1,1)

$ rosservice call /turtle1/teleport_relative 1 0

- - - -
Check Turtle1's pose
$ rostopic echo /turtle1/pose

```
x: 1.0
y: 1.0
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
```

LETS CONTROL THE TURTLE- Publish to /turtle1/cmd_vel:
(roscore and turtlesim_node running)

1. Command line
2. Keyboard
3. Joystick
4. Python

Subscriptions: /turtle1/cmd_vel [unknown type]

$ rostopic type /turtle1/cmd_vel

geometry_msgs/Twist

1a. Move a bit 1 command

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

1b. Move in a circle repeat at frequency $ **rostopic hz /turtle1/pose**

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

2. $ rosrun turtlesim turtle_teleop_key (Cntl+c to exit)

Reading from keyboard

Use arrow keys to move the turtle.

```
Up arrow       Turtle up
Down arrow     Turtle down
Right arrow    Rotate CW
Left  arrow    Rotate CCW
```

$ rqt_graph (See the namespaces, nodes and topics)

3. Joystick

4. Python Creates node /ControlTurtlesim ; Publishes t0 /turtle1/cmd_vel with Twist msg

$ python turtlesim1.py (Make Executable $chmod +x turtlesim1.py

```
[INFO] [WallTime: 1455313387.186692]  Press CTRL+c to stop TurtleBot
[INFO] [WallTime: 1455313387.188315]  Set rate 10Hz
```

```
#!/usr/bin/env python   turtlesim1.py
# Execute as a python script
# Set linear and angular values of Turtlesim's speed and turning.
import rospy                                # Needed to create a ROS node
from geometry_msgs.msg import Twist    # Message that moves base

class ControlTurtlesim():
    def __init__(self):
        # ControlTurtlesim is the name of the node sent to the master
        rospy.init_node('ControlTurtlesim', anonymous=False)

                # Message to screen
        rospy.loginfo(" Press CTRL+c to stop TurtleBot")

        # Keys CNTL + c will stop script
        rospy.on_shutdown(self.shutdown)

         # Publisher will send Twist message on topic
        # /turtle1/cmd_vel

        self.cmd_vel = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

             # Turtlesim will receive the message 10 times per second.
        rate = rospy.Rate(10);
             # 10 Hz is fine as long as the processing does not exceed
        #   1/10 second.
        rospy.loginfo(" Set rate 10Hz")
        # Twist is geometry_msgs for linear and angular velocity
        move_cmd = Twist()
             # Linear speed in x in meters/second is + (forward) or
        # - (backwards)
        move_cmd.linear.x = 0.3    # Modify this value to change speed
             # Turn at 0 radians/s
        move_cmd.angular.z = 0
        # Modify this value to cause rotation rad/s

             # Loop and TurtleBot will move until you type CNTL+c
        while not rospy.is_shutdown():
                # publish Twist values to the Turtlesim node /cmd_vel
            self.cmd_vel.publish(move_cmd)
                # wait for 0.1 seconds (10 HZ) and publish again
            rate.sleep()


    def shutdown(self):
        # You can stop turtlebot by publishing an empty Twist message
        rospy.loginfo("Stopping Turtlesim")

        self.cmd_vel.publish(Twist())
             # Give TurtleBot time to stop
        rospy.sleep(1)

if __name__ == '__main__':
```

```
    try:
        ControlTurtlesim()
    except:
        rospy.loginfo("End of the trip for Turtlesim")
```

```
$ rosnode list
/ControlTurtlesim
/rosout
/teleop_turtle
/turtlesim
```

Change Python script turtlesim1.py to run turtle in a circle (Make executable)
In ros_ws

tlharmanphd@D125-43873:~/ros_ws$ **python turtlesim2.py**
[INFO] [WallTime: 1455383932.566053] Press CTRL+c to stop TurtleBot
[INFO] [WallTime: 1455383932.567306] Set rate 10Hz
^C[INFO] [WallTime: 1455383937.538077] Stopping Turtlesim

In turtlesim2.py Script
move_cmd = Twist()
	# Linear speed in x in meters/second is + (forward) or
	# - (backwards)
	move_cmd.linear.x = 2.0	# Modify this value to change speed
	# Turn at 1.8 radians/s
	move_cmd.angular.z = 1.8
	# Modify this value to cause rotation rad/s