

# Understanding ROS using Turtlesim

## **F1/10 Autonomous Racing**

CS 4501/SYS4582

Spring 2019

Madhur Behl

madhur.behl@virginia.edu

[Compiled using the official ROS Turtlesim tutorials.]



## Contents

<b>roscore.....</b>	<b>3</b>
<b>ROS Nodes, Topics, And Services using Turtlesim.....</b>	<b>3</b>
<b>Turtlesim Node .....</b>	<b>4</b>
<b>ROS Nodes with Turtlesim.....</b>	<b>4</b>
<b>roscd .....</b>	<b>4</b>
<b>ROS Services to move the Turtle .....</b>	<b>5</b>
<b>teleport_absolute.....</b>	<b>6</b>
<b>teleport_relative .....</b>	<b>6</b>
<b>Turtlesim Node Topic Pose.....</b>	<b>7</b>
<b>Make the Turtle move in a circle using rostopic pub .....</b>	<b>8</b>
<b>Type of message for cmd_vel .....</b>	<b>8</b>
<b>Move the Turtle once.....</b>	<b>9</b>
<b>Let's reset Turtlesim.....</b>	<b>11</b>
<b> racing@racing-vm~\$ rosservice call /reset.....</b>	<b>11</b>
<b>rostopic hz.....</b>	<b>11</b>
<b>rostopic hz /turtle1/pose.....</b>	<b>11</b>
<b>Using the rqt_plot tool with Turtlesim .....</b>	<b>12</b>
<b>rqt_plot.....</b>	<b>12</b>
<b>Keyboard control of the turtle using teleop_key.....</b>	<b>14</b>
<b>\$ rosrunc turtlesim turtle_teleop_key.....</b>	<b>14</b>
<b>New Node /teleop_turtle.....</b>	<b>14</b>
<b>Node /turtlesim info after /teleop_turtle.....</b>	<b>15</b>
<b>Determine data from Topic /turtle1/cmd_vel.....</b>	<b>17</b>
<b>To find turtle's position in window use /turtle1/pose.....</b>	<b>19</b>
<b>Clear the screen.....</b>	<b>20</b>
<b>Python and Turtlesim.....</b>	<b>20</b>

## Roscore

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

\$ **roscore** [1]

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.

## ROS Nodes, Topics, and Services using Turtlesim

 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](#)

## Turtlesim Node

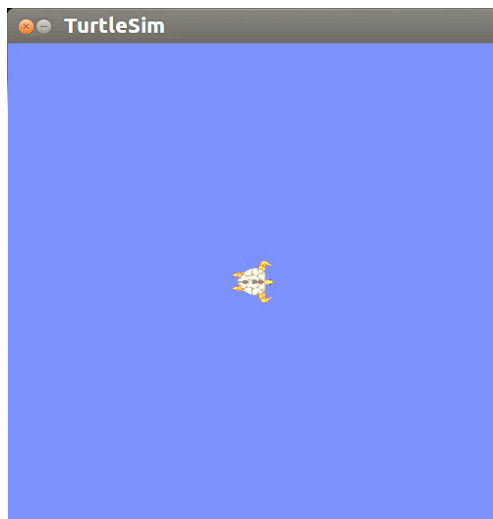
We will start the turtlesim node and explore its properties. Execute roscore and in a new terminal create the turtlesim node from the package turtlesim:

```
$ roscore
```

```
$ roslaunch turtlesim turtlesim_node [2]
```

```
[ INFO] [1516751529.792931813]: Starting turtlesim with node name /turtlesim  
[ INFO] [1516751529.797525686]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445],  
theta=[0.000000]
```

The roslaunch 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.



Before moving the turtle, let us study the properties of the nodes, topics, service and messages available within turtlesim package in another window. (remember to use terminator)

## ROS Nodes with Turtlesim

```
roslaunch turtlesim turtlesim_node [3]
```

```
$ roslaunch turtlesim turtlesim_node  
/rosout  
/turtlesim
```

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

racing@racing-vm:~\$ **rostopic info /turtlesim** [4]

-----  
Node [/turtlesim]  
Publications: **(This information is sent to nodes listening to /turtlesim)**  
\* /turtle1/color\_sensor [turtlesim/Color] **(Color message in turtlesim package)**  
  
\* /rosout [roscpp\_msgs/Log]  
\* /turtle1/pose [turtlesim/Pose] **(Pose message in turtlesim package for /turtle1)**  
  
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: **(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

contacting node http://D104-45931:42032/ ...

Pid: 4911

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.

## **ROS Services to Move the Turtle**

Services: **(We can use ROS services to manipulate the turtle and perform other operations**

**- the format is \$rosservice call <service> <arguments>)**

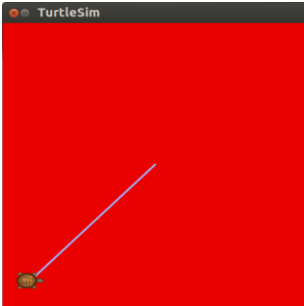
- \* **/turtle1/teleport\_absolute**
- \* /turtlesim/get\_loggers
- \* /turtlesim/set\_logger\_level
- \* /reset
- \* /spawn
- \* /clear

```
* /turtle1/set_pen
* /turtle1/teleport_relative
* /kill
```

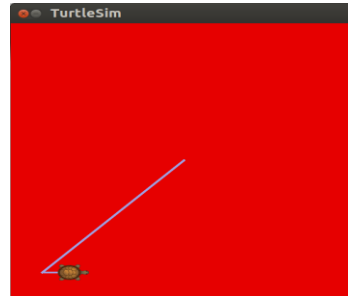
The turtle can be moved using the rosservice teleport option. The format of the position is [x y theta].

### **teleport\_absolute**

```
racing@racing-vm:/$ rosservice call /turtle1/teleport_absolute 1 1 0 [5]
```



Turtle After Absolute Move



Turtle After Relative Move

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

### **teleport\_relative**

```
rosservice call /turtle1/teleport_relative 1 0 [6]
```

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.

```
$ rostopic info /turtle1/pose [7]
Type: turtlesim/Pose
Publishers:
  * /turtlesim (http://D104-45931:42032/)
Subscribers: None
This displays the message type.
```

```
racing@racing-vm:~$ rostopic type /turtle1/pose [8]
turtlesim/Pose
Confirming the message type
```

```
tlharmanphd@D125-43873:~$ rostopic show turtlesim/Pose [9]
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
Show/display the message fields.
```

```
tlharmanphd@D125-43873:/$ rostopic echo /turtle1/pose [10]
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
.
.
.
Echo the message to the terminal, i.e. display the message values.
```

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

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

## Make the Turtle move in a circle rostopic pub <command>

```
racing@racing-vm:~$ roscat info /turtlesim [11]
```

```
-----  
Node [/turtlesim]  
Publications:  
* /turtle1/color_sensor [turtlesim/Color]  
* /rosout [roscpp_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://D104-45931:42032/ ...
```

```
Pid: 4911
```

```
Connections:
```

```
* topic: /rosout  
* to: /rosout  
* direction: outbound  
* transport: TCPROS
```

## Type of message for cmd\_vel

```
racing@racing-vm:~$ rostopic type /turtle1/cmd_vel [12]
```

```
geometry_msgs/Twist
```

Once again, rostopic type <topic name> displays the type of message in the topic

```
racing@racing-vm:~$ rostopic show geometry_msgs/Twist [13]
```

```
geometry_msgs/Vector3 linear
```

```
float64 x
```

```
float64 y
```

```
float64 z
```

```
geometry_msgs/Vector3 angular
```

```
float64 x
```

```
float64 y
```

```
float64 z
```

Let us take a look at the geometry\_msgs/Twist message type



Using the Linux shell we can combine two commands. The | operator is used to pipe commands into the shell. A pipe is a form of redirection that is used in Linux systems to send the output of one program to another program. The general syntax for pipes is: `command_1 | command_2 | command_3 . . .`

```
$ rostopic type /turtle1/cmd_vel | rosmmsg show [14]
  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](http://wiki.ros.org/geometry_msgs)

This displays a verbose list of topics to publish to and subscribe to and their type:

```
$ rostopic list -v [15]
  Published topics:
    * /turtle1/color_sensor [turtlesim/Color] 1 publisher
    * /rosout [rograph_msgs/Log] 1 publisher
    * /rosout_agg [rograph_msgs/Log] 1 publisher
    * /turtle1/pose [turtlesim/Pose] 1 publisher

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

### Moving the 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]' [16]

-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
```

NOTE: Here is a place to use TAB completion to find data formats for this command:

Lets try it:

**\$ rostopic pub -1 /turtle1/cmd\_vel geometry\_msgs/Twist "linear: [17] With result:**

**racing@racing-vm:~\$ rostopic pub -1 /turtle1/cmd\_vel geometry\_msgs/Twist "linear:**

**x: 0.0**

**y: 0.0**

**z: 0.0**

**angular:**

**x: 0.0**

**y: 0.0**

**z: 0.0"**

**Now back space to fill in the values z= 1.8 and x=0.0. (Not executed)**

**Where is the turtle? (After the Initial Command)**

**\$ rostopic echo /turtle1/pose [18]**

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.

**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 **\$ rostopic info /turtlesim** shown before with the subscription:

Subscribed topics:

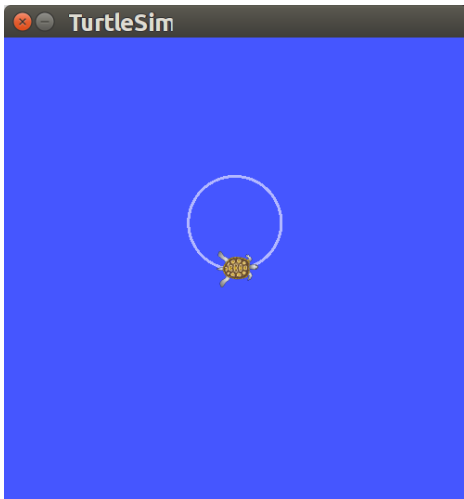
\* /turtle1/cmd\_vel [geometry\_msgs/Twist] 1 subscriber **rostopic pub**

## Make the turtle move in a circle

Let's reset Turtlesim

```
 racing@racing-vm:~$ rosservice call /reset [19]
```

```
 racing@racing-vm:~$ rostopic pub -r 1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' [20]
```



Turtle moving in a circle

**rostopic hz**

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

```
 rostopic hz /turtle1/pose [21]
```

```
 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.

## Using rqt plot with Turtlesim

[http://wiki.ros.org/rqt\\_plot](http://wiki.ros.org/rqt_plot)

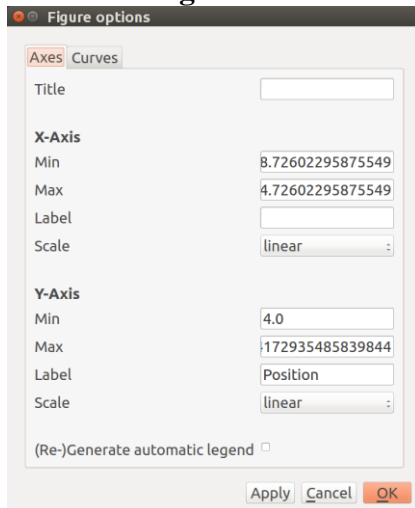
### rqt\_plot

We can plot information about the nodes and topics.

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

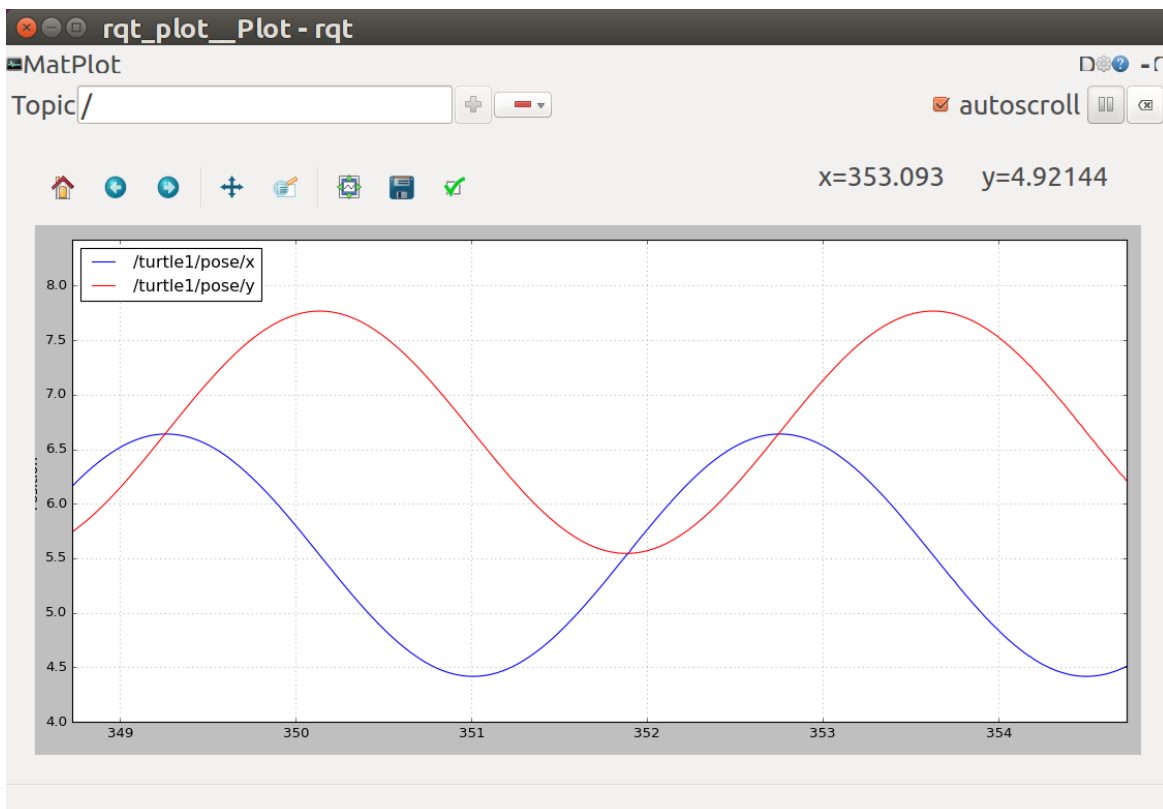
[22]

**Turtle is turning in a circle about 5.5 Ymin x goes from about 4.5 to 6.5.**



*Selection of Axis for rqt\_plot (Click on the check mark)*

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



*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.

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

[http://wiki.ros.org/rqt\\_plot](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.

## Keyboard Control

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.

```
$ roslaunch turtlesim turtlesim_teleop_key [23]
```

```
 racing@racing-vm:~$ roslaunch turtlesim turtlesim_teleop_key
```

```
  Reading from keyboard
```

```
-----
```

```
  Use arrow keys to move the turtle.
```

```
  Up arrow   Turtle In Turtle's x direction
```

```
  Down arrow Turtle In Turtles's -x direction
```

```
  Right arrow Rotate CW
```

```
  Left arrow  Rotate CCW
```

```
 racing@racing-vm:~$ rostopic list [24]
```

```
  /rosout
```

```
  /teleop_turtle
```

```
  /turtlesim
```

**Notice that now we have a new node in the list called /teleop\_turtle**

```
 racing@racing-vm:~$ rostopic info /teleop_turtle [25]
```

```
-----  
  Node [/teleop_turtle]
```

```
  Publications:
```

```
  * /turtle1/cmd_vel [geometry_msgs/Twist]
```

```
  * /rosout [roscpp_msgs/Log]
```

The /teleop\_turtle node is publishing on topic /turtle1/cmd\_vel  
Can you tell the message type for this topic ?

```
  Subscriptions: None
```

The /teleop\_turtle node does not subscribe to any topic.

```
  Services:
```

```
  * /teleop_turtle/get_loggers
```

```
  * /teleop_turtle/set_logger_level
```

```
  contacting node http://D104-45931:43692/ ...
```

```
  Pid: 8381
```

```
  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]**

Let us look again at the node /turtlesim after we have started running the /teleop\_turtle node

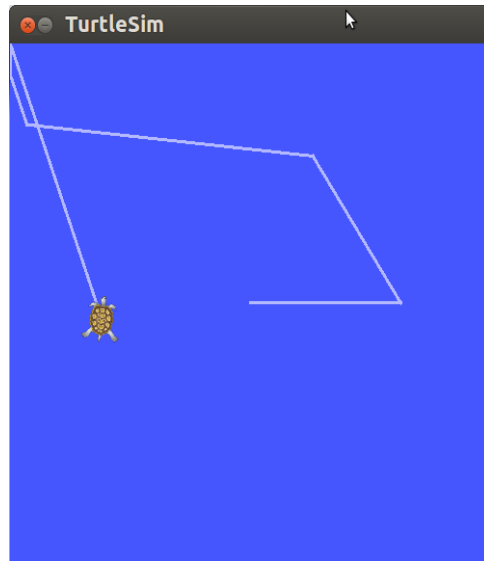
racimg@racimg-vm:~\$ **rostopic info /turtlesim**

[26]

```
-----  
Node [/turtlesim]  
Publications:  
* /turtle1/color_sensor [turtlesim/Color]  
* /rosout [roscpp_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://D104-45931:42252/ ...  
Pid: 7956  
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
```

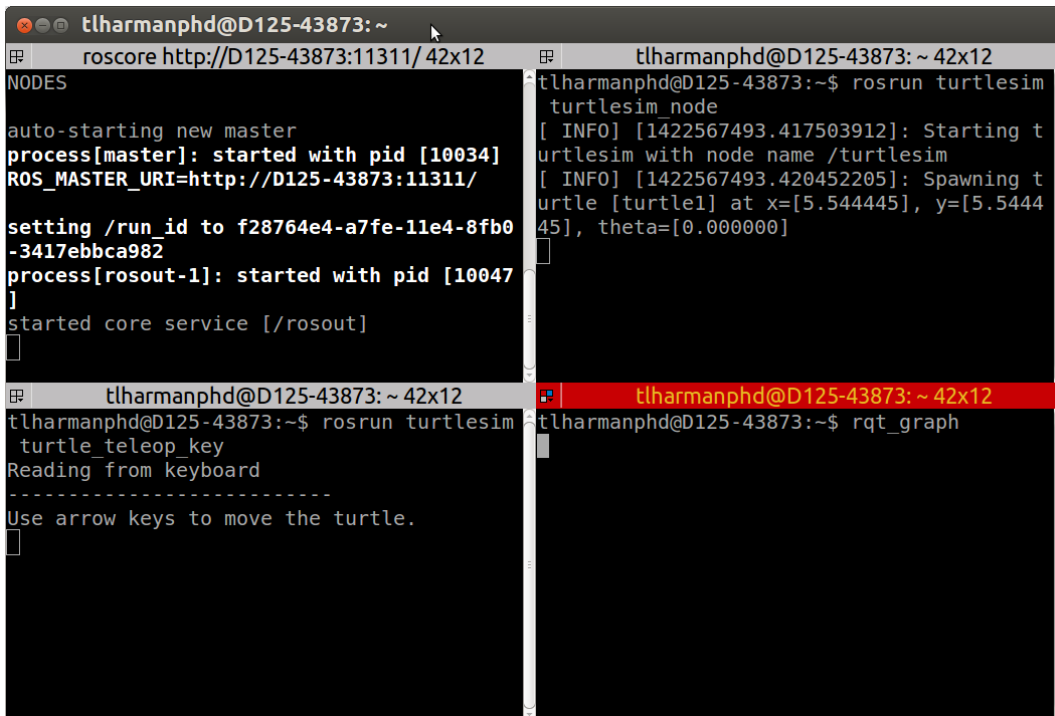
**Note:** New topic /turtle1/cmd\_vel to /teleop\_turtle

To move turtle with arrow keys, be sure the focus is on the terminal that is running `turtle_teleop_key`.



Turtlesim keyboard control

Now 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.





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)

racing@racing-vm:~\$ rostopic list

[27]

```
/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

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

[28]

```
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 the turtle's position in the ocean, use /turtle1/pose

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

[29]

```
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.

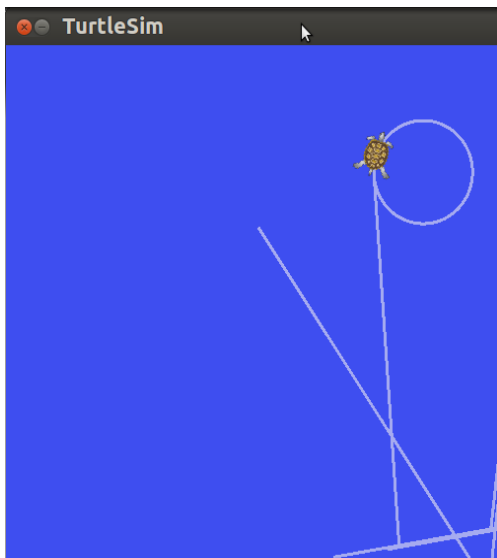
If you return to the teleop\_key window and move the turtle with the arrow keys you can see the output of the pose message (turtlesim/Pose) change. Remember the format:

```
tlharmanphd@D125-43873:~$ rosmmsg 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/cmd\_velocity

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



**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>

Try changing the rate to 0.5 or some value less than 1 to see the turtle stall in the circle.

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](#).

**To clear the turtlesim screen use:**

```
racing@racing-vm:~$ rosservice call /clear
```