
TurtleSim

Contents

ROSCORE

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

$ **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.
>
> 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

Other useful references are Listed in Appendix I

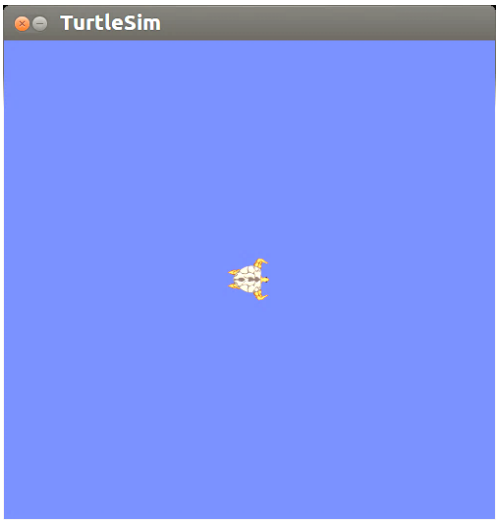
TURTLESIM NODE

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

$ roscore

$ **rosrun turtlesim turtlesim_node**

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.



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

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 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: (t**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.

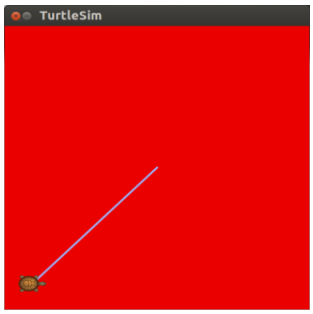
ROS SERVICES TO MOVE TURTLE

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

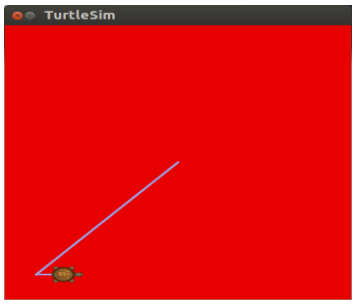
The turtle can be moved using the rosservice teleport option. The format of the position is [x y theta].

teleport_absolute

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



Turtle After Relative 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

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**

```
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 rostopic pub COMMAND

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]'**

NOTE: HERE IS A PLACE TO USE TAB COMPLETION TO FIND THE DATA FORMATS FOR THIS COMMAND

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]'**

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.

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

rqt_plot
We can plot information about the nodes and topics.

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

Select plotting type:

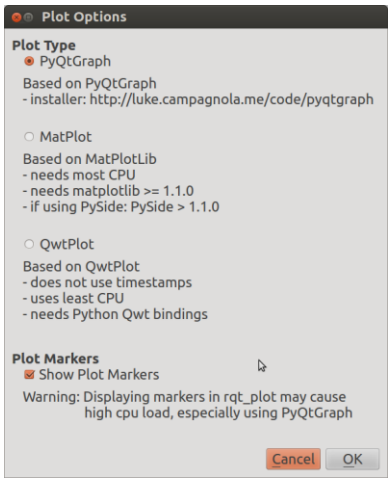


Figure 6 Selection of Plotting for rqt_plot

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

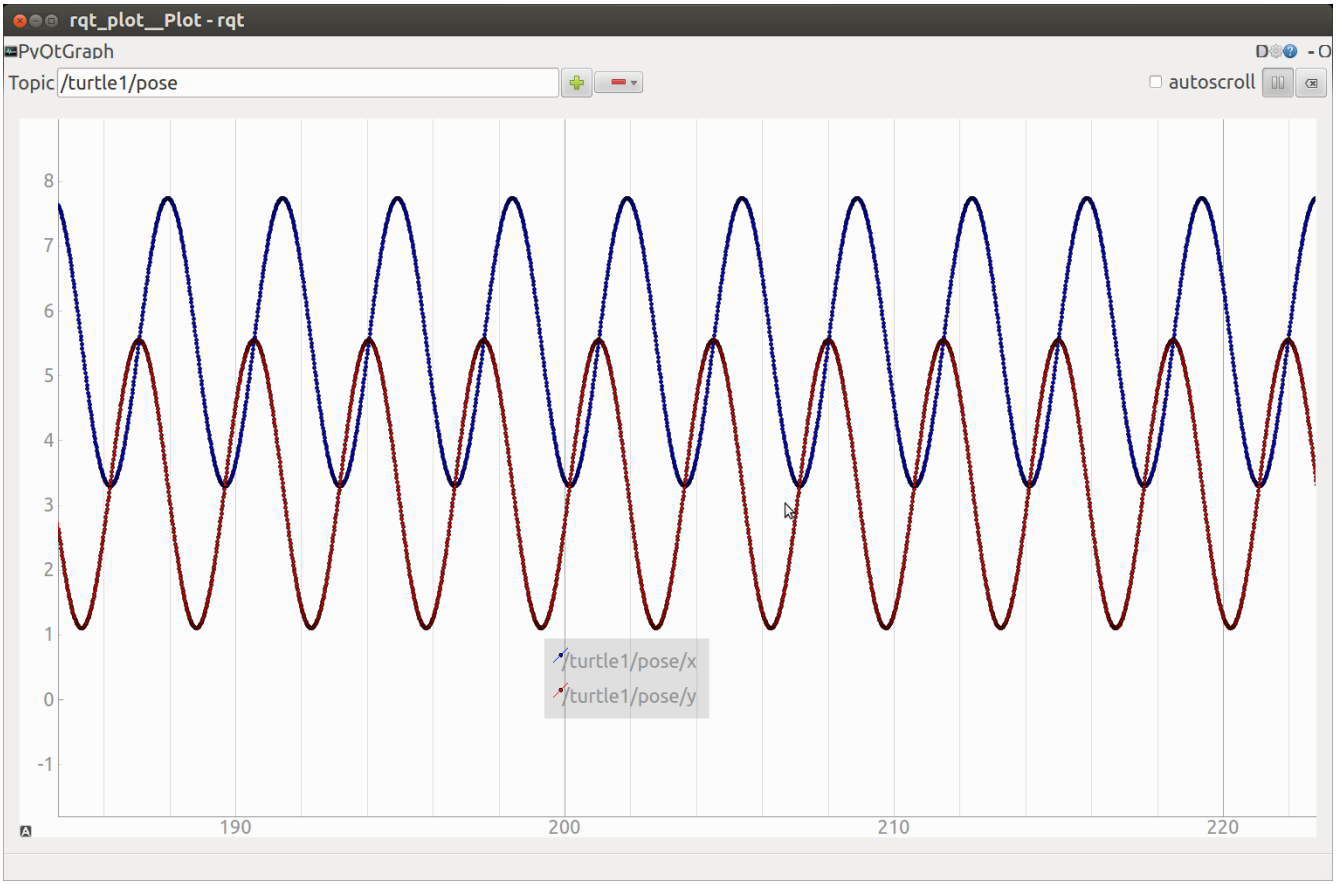


Figure 7 Plot of /turtle1/pose/x and /pose/y

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.

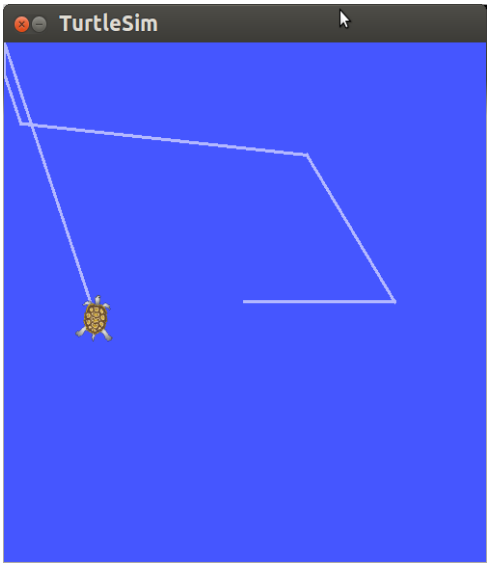


Figure 7 Turtlesim After Moving

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.

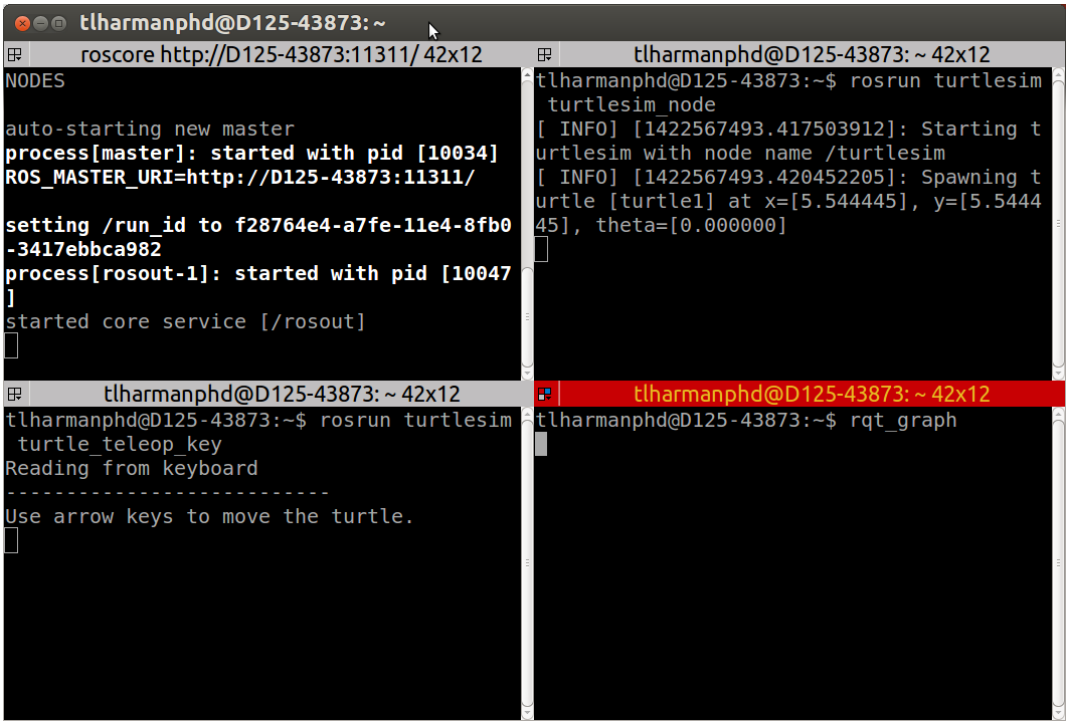


Figure 8 Four Turtlesim Windows using Terminator

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

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

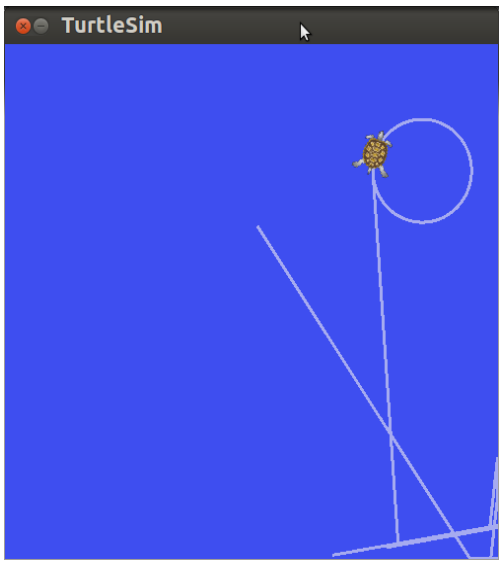


Figure 9 Turtle responds to published topic

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**

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

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
```

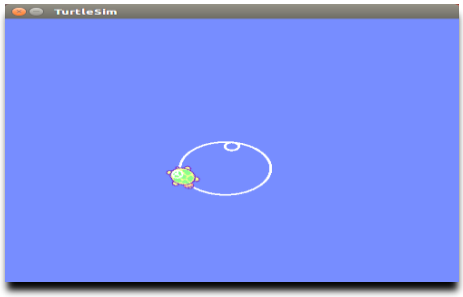
```
#!/usr/bin/env python   turtlesim1.py
# Execute as a python script
# Set linear and angular values of Turtlesim's speed and turning.
import rospy                                # Needed to create a ROS node
from geometry_msgs.msg import Twist    # Message that moves base

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
```

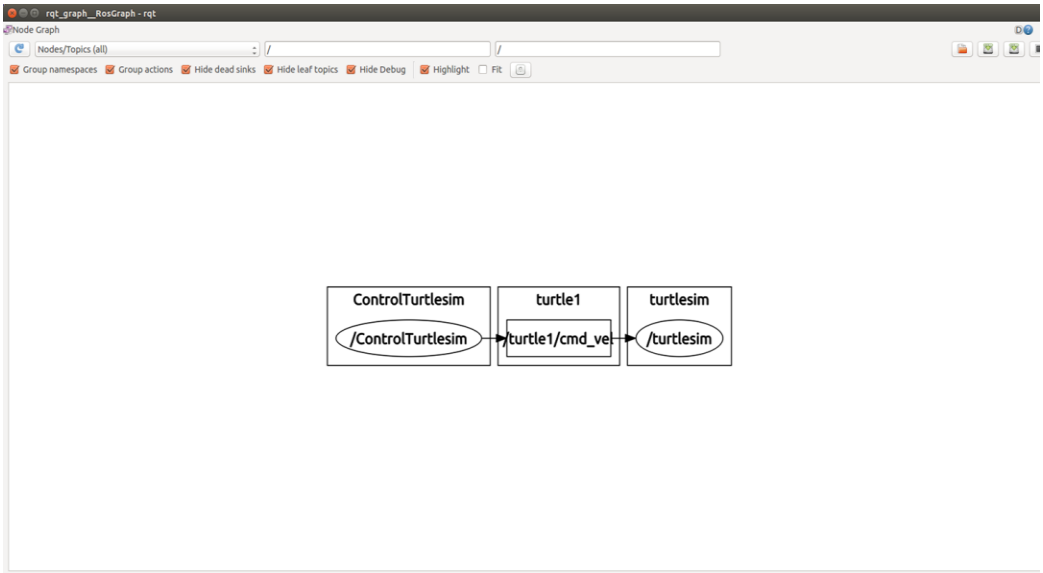
Change Python script turtlesim1.py to run turtle in a circle (Make executable)

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

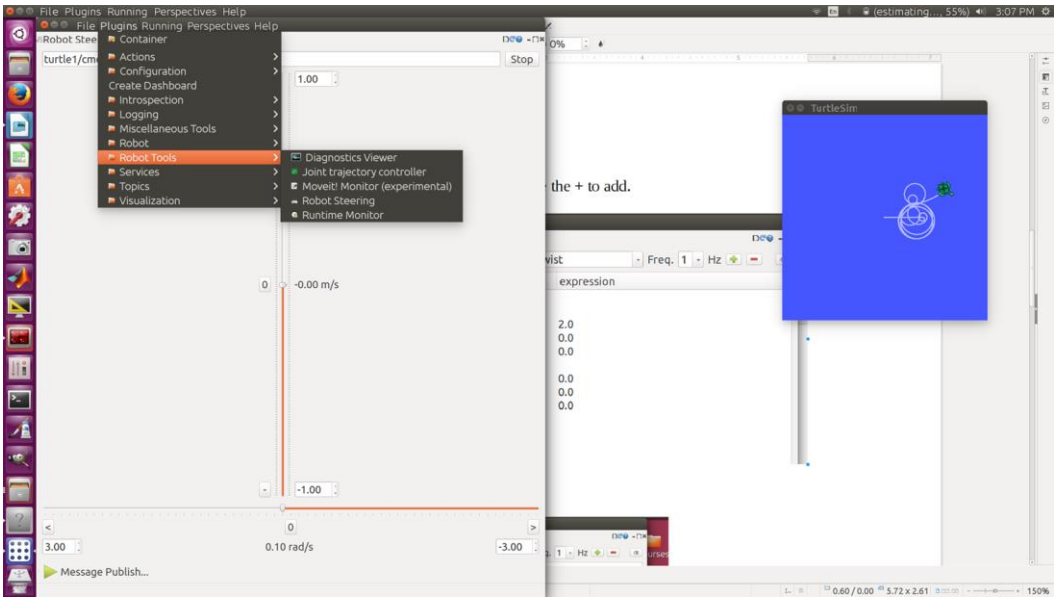
[INFO] [WallTime: 1455383932.566053] Press CTRL+c to stop TurtleBot
[INFO] [WallTime: 1455383932.567306] Set rate 10Hz
^C[INFO] [WallTime: 1455383937.538077] Stopping Turtlesim



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



Steering Plugin rqt Robot Tools/ Robot


File Plugins Running Perspectives Help
Container
Actions
Configuration
Create Dashboard
Introspection
Logging
Miscellaneous Tools
Robot
Robot Tools
Services
Topics
Visualization
Diagnostics Viewer
Joint trajectory controller
MoveIt! Monitor (experimental)
Robot Steering
Runtime Monitor
Stop
1.00
-0.00 m/s
-1.00
0.10 rad/s
Message Publish...
TurtleSim
the + to add.
Freq.
Hz
expression
2.0
0.0
0.0
0.0
0.0
0.0

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

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