ROS 2 Concept by turtlesim
By TESR

Install turtlesim

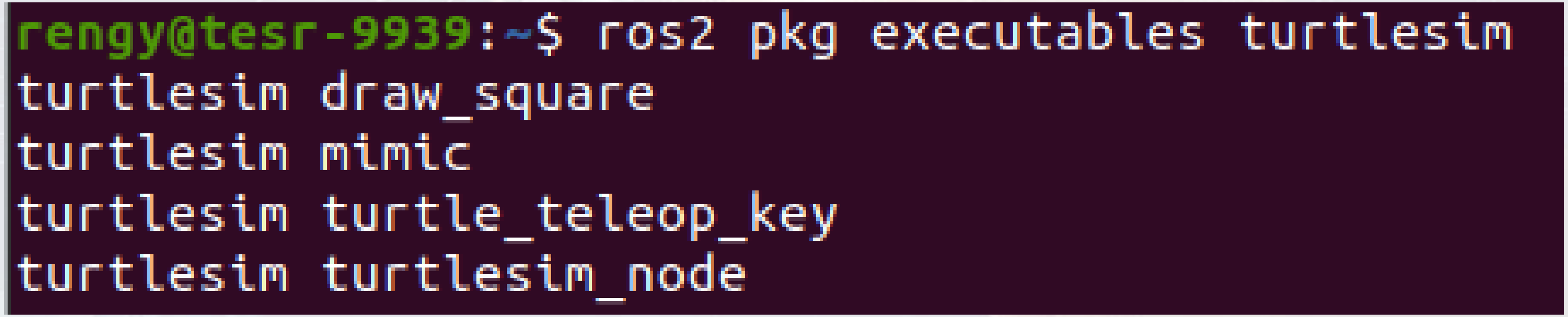
- Install the **turtlesim** package for your ROS 2 distro.

```
sudo apt update; sudo apt upgrade
sudo apt install ros-foxy-turtlesim
```

- Check that the package installed.

```
ros2 pkg executables turtlesim
```

- Output should be like this

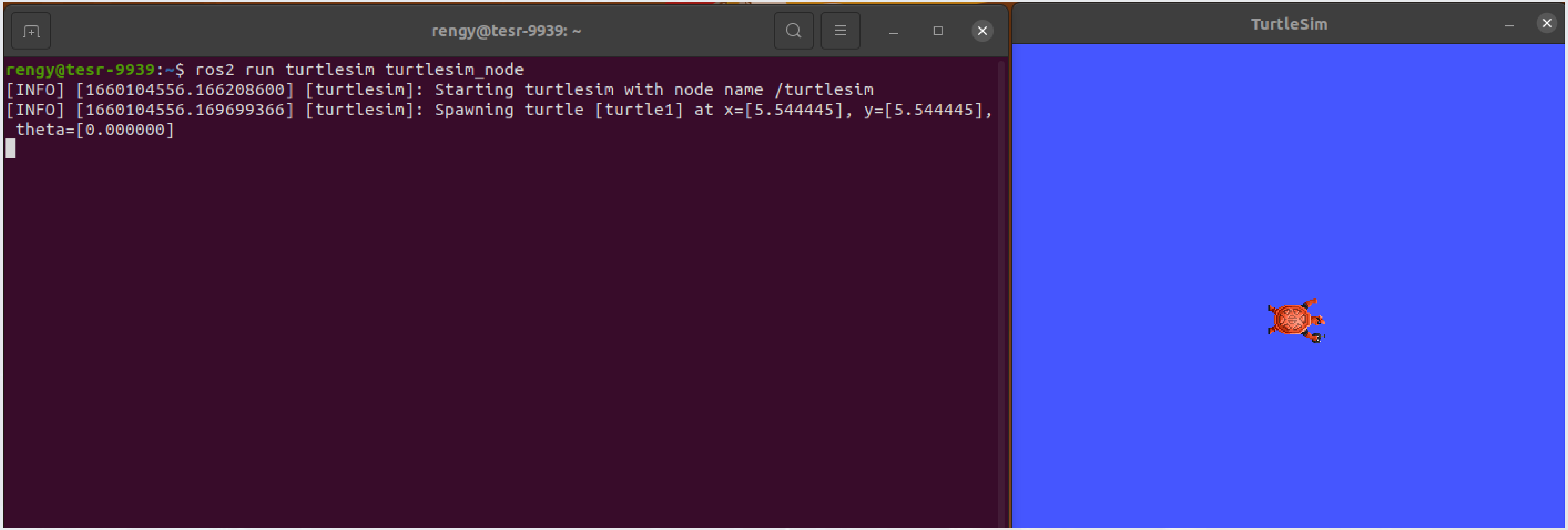


```
rengy@tesr-9939:~$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
```

Start turtlesim

- After **turtlesim** installed, start the turtlesim by type command:

```
ros2 run turtlesim turtlesim_node
```

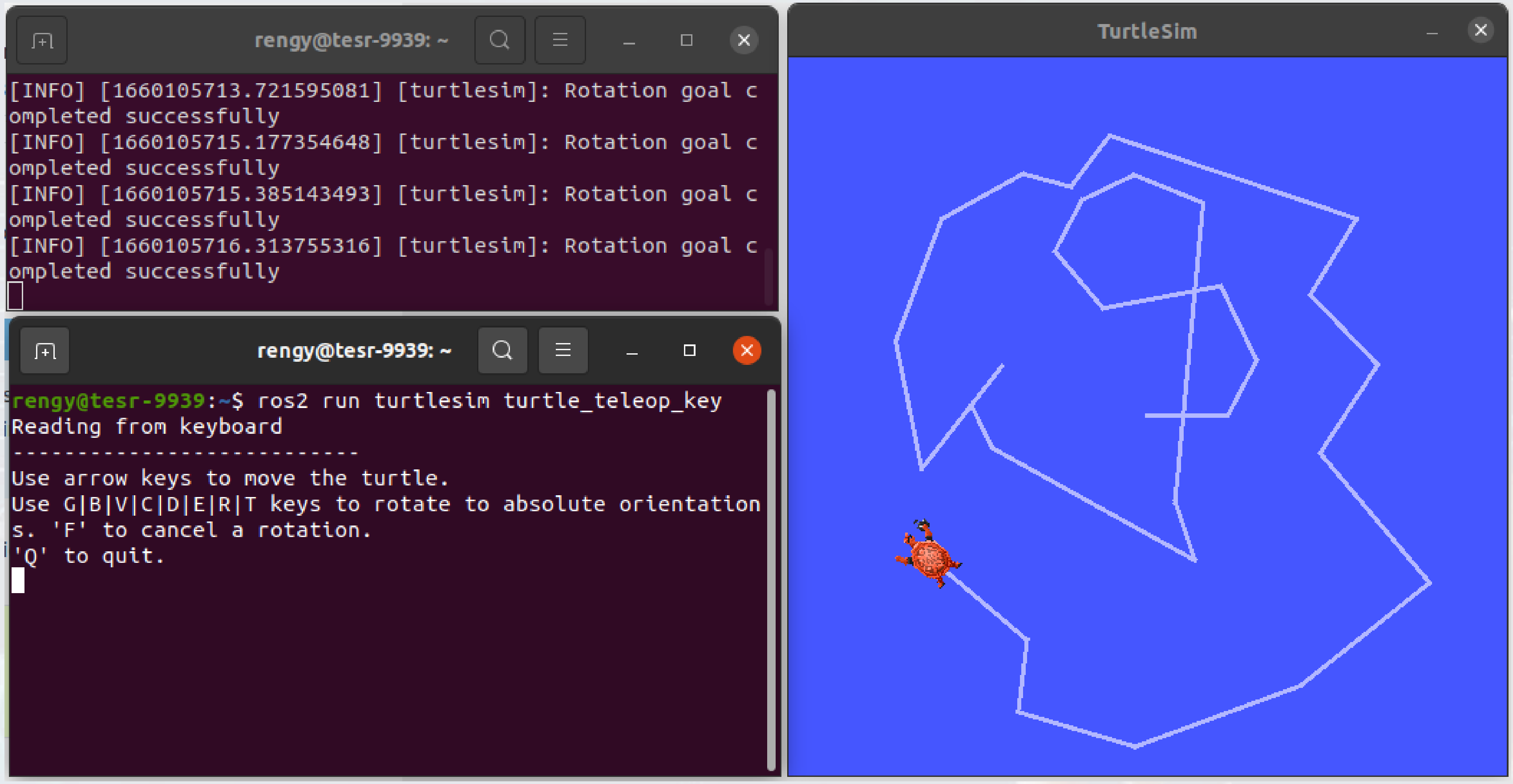


```
rengy@tesr-9939:~$ ros2 run turtlesim turtlesim_node
[INFO] [1660104556.166208600] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1660104556.169699366] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445],
 theta=[0.000000]
```

Use turtlesim

- Let's control turtle using keyboard by run command below:

```
ros2 run turtlesim turtle_teleop_key
```



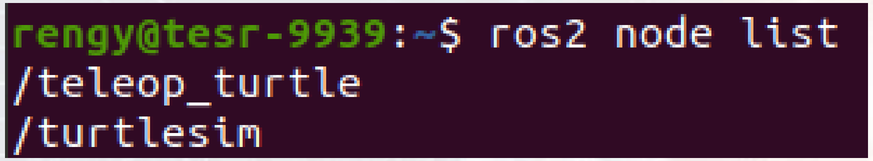
```
[INFO] [1660105713.721595081] [turtlesim]: Rotation goal c
ompleted successfully
[INFO] [1660105715.177354648] [turtlesim]: Rotation goal c
ompleted successfully
[INFO] [1660105715.385143493] [turtlesim]: Rotation goal c
ompleted successfully
[INFO] [1660105716.313755316] [turtlesim]: Rotation goal c
ompleted successfully
```

```
rengy@tesr-9939:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientation
s. 'F' to cancel a rotation.
'Q' to quit.
```

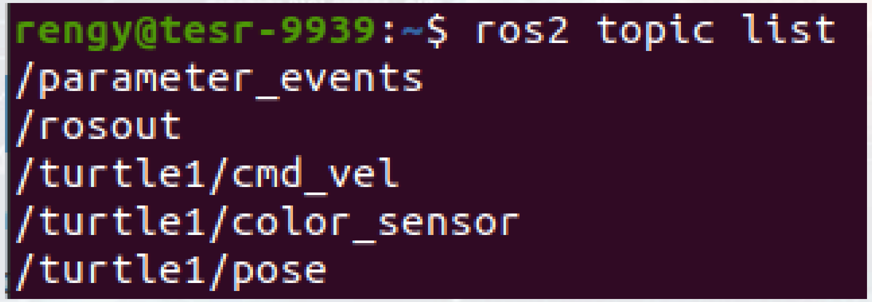
Use turtlesim

- Now, you can use **"list"** command to see the **nodes** and their associated **services**, **topics** and **actions.**

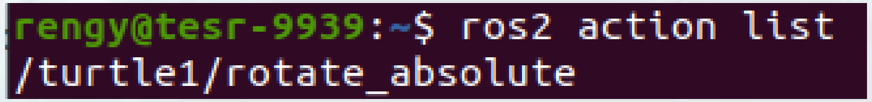
```
ros2 node list
ros2 topic list
ros2 service list
ros2 action list
```



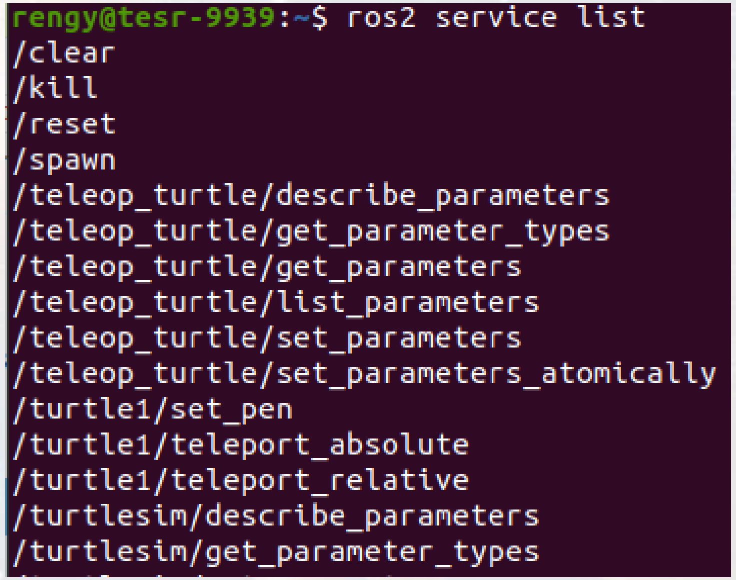
```
rengy@tesr-9939:~$ ros2 node list
/teleop_turtle
/turtlesim
```



```
rengy@tesr-9939:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```



```
rengy@tesr-9939:~$ ros2 action list
/turtle1/rotate_absolute
```

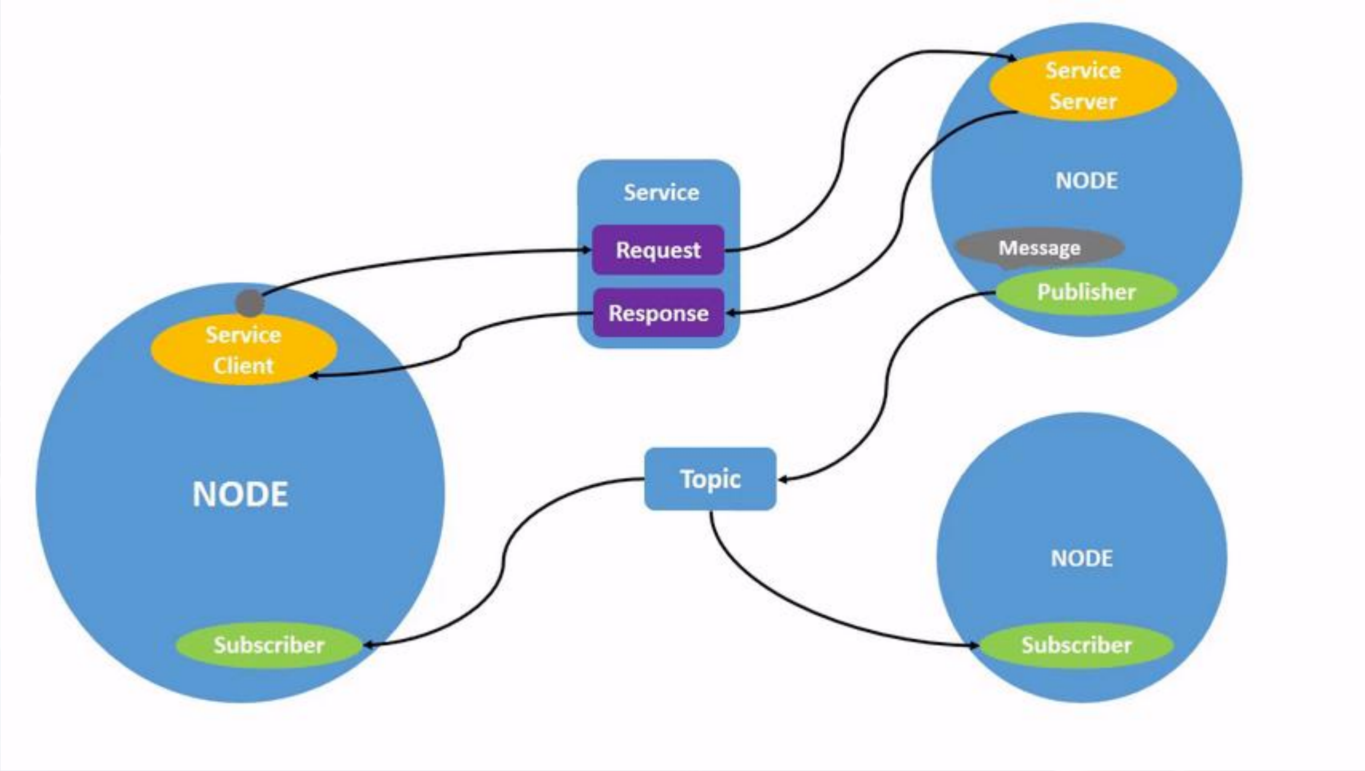


```
rengy@tesr-9939:~$ ros2 service list
/clear
/kill
/reset
/spawn
/teleop_turtle/describe_parameters
/teleop_turtle/get_parameter_types
/teleop_turtle/get_parameters
/teleop_turtle/list_parameters
/teleop_turtle/set_parameters
/teleop_turtle/set_parameters_atomically
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
```

Understanding nodes

NODE can define as element within ROS network that should response for a single, module purpose.

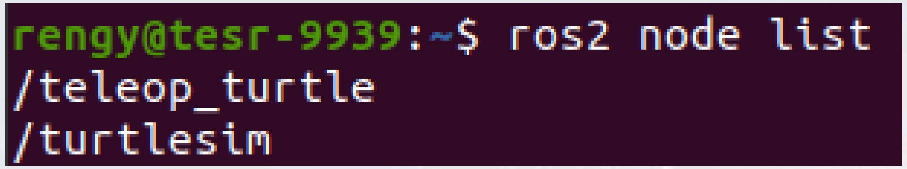
- This **ROS graph** show the network of **NODE,** Each **NODE** can send and receive data to other **NODES** via **topics, services, actions, or parameters**.



Understanding nodes

- **ROS 2 node list**

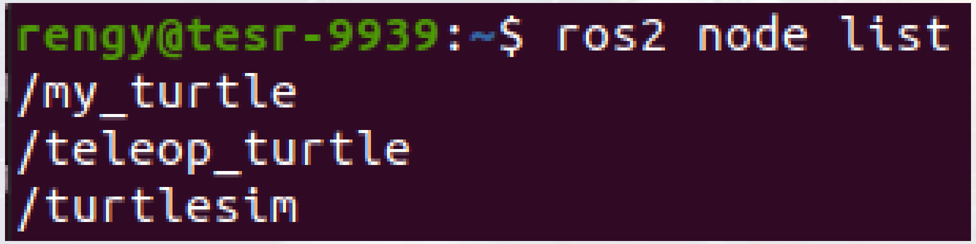
```
ros2 node list
```



```
rengy@tesr-9939:~$ ros2 node list
/teleop_turtle
/turtlesim
```

- **Remapping**

```
ros2 run turtlesim turtlesim_node --ros-args --remap __node:=my_turtle
```

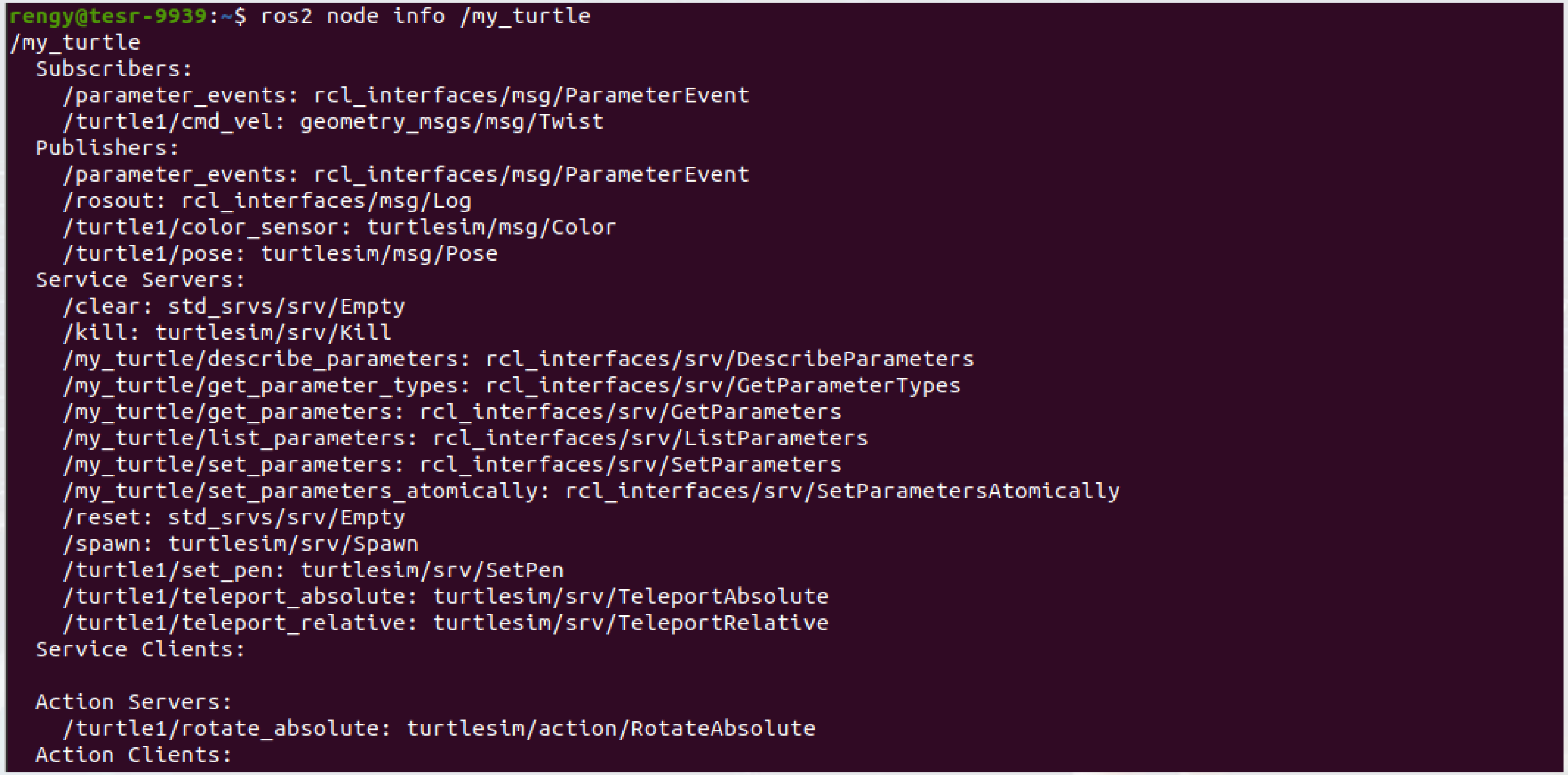


```
rengy@tesr-9939:~$ ros2 node list
/my_turtle
/teleop_turtle
/turtlesim
```

Understanding nodes

- **ROS 2 node info**

```
ros2 node info /my_turtle
```



```
rengy@tesr-9939:~$ ros2 node info /my_turtle
/my_turtle
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /turtle1/cmd_vel: geometry_msgs/msg/Twist
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /turtle1/color_sensor: turtlesim/msg/Color
    /turtle1/pose: turtlesim/msg/Pose
  Service Servers:
    /clear: std_srvs/srv/Empty
    /kill: turtlesim/srv/Kill
    /my_turtle/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /my_turtle/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /my_turtle/get_parameters: rcl_interfaces/srv/GetParameters
    /my_turtle/list_parameters: rcl_interfaces/srv/ListParameters
    /my_turtle/set_parameters: rcl_interfaces/srv/SetParameters
    /my_turtle/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
    /reset: std_srvs/srv/Empty
    /spawn: turtlesim/srv/Spawn
    /turtle1/set_pen: turtlesim/srv/SetPen
    /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
    /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
  Service Clients:

  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

Install rqt

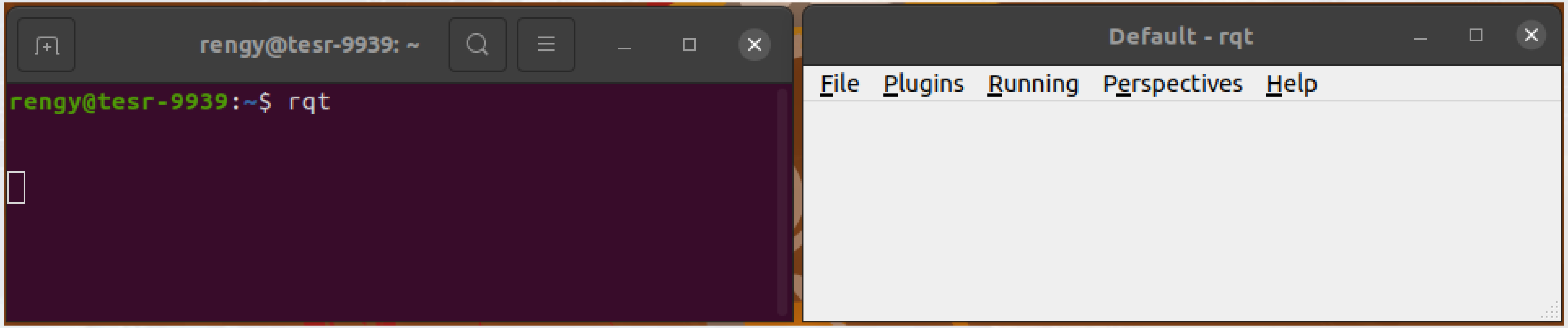
- **Open a new terminal to install rqt and it plugin.**

```
sudo apt update; sudo apt upgrade
sudo apt install ~nros-foxy-rqt*
```

And, to run rqt

```
rqt
```

Output should be like this

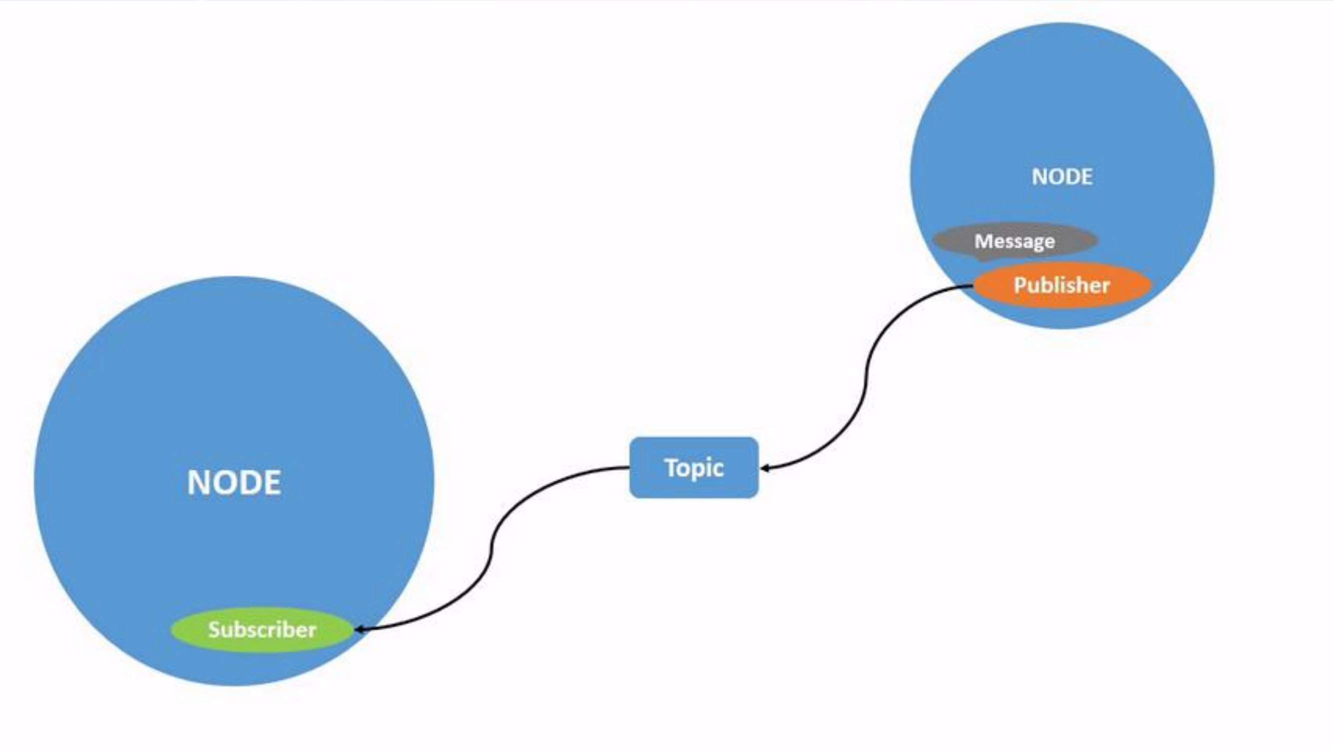


```
rengy@tesr-9939:~$ rqt
```

Understanding topics

Topics are one of the main ways in which data is moved between nodes and therefore between different parts of the system.

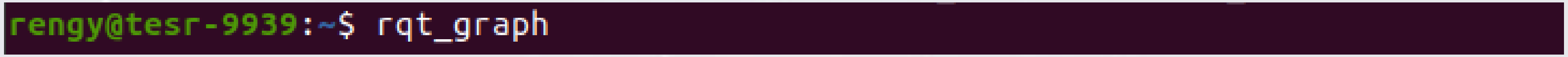
- This **ROS graph** show the send and receive of message from **NODE** to **NODE** pass through the topics by each NODE's **publisher** to NODE's **subscriber**.



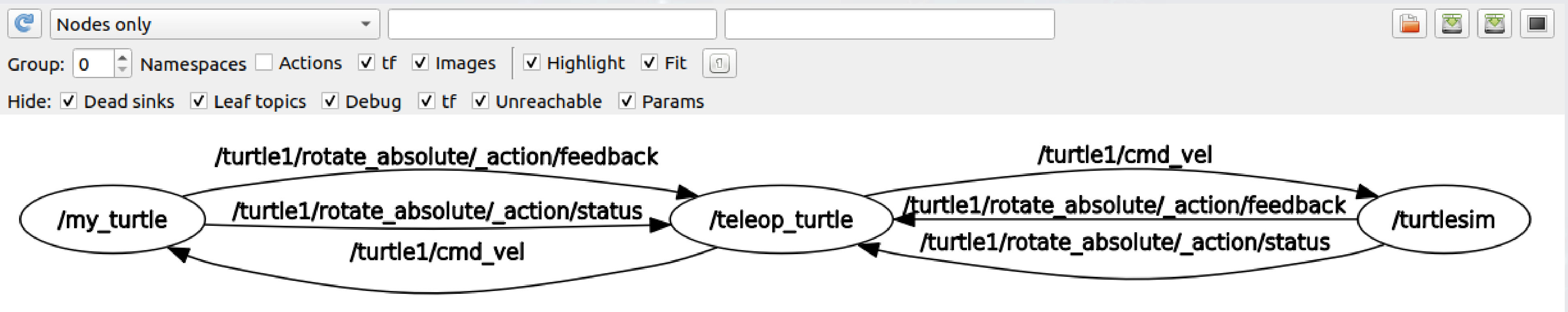
Understanding topics

- **rqt_graph**

```
rqt_graph
```



```
rengy@tesr-9939:~$ rqt_graph
```



Nodes only

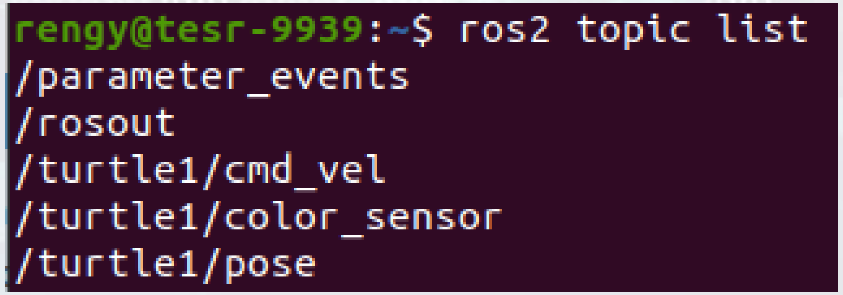
Group: 0 Namespaces ☐ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug ☑ tf ☑ Unreachable ☑ Params

/turtle1/rotate_absolute/_action/feedback
/turtle1/cmd_vel
/my_turtle
/turtle1/rotate_absolute/_action/status
/teleop_turtle
/turtle1/rotate_absolute/_action/feedback
/turtlesim
/turtle1/cmd_vel
/turtle1/rotate_absolute/_action/status

- **ROS 2 topic list**

```
ros2 topic list
```

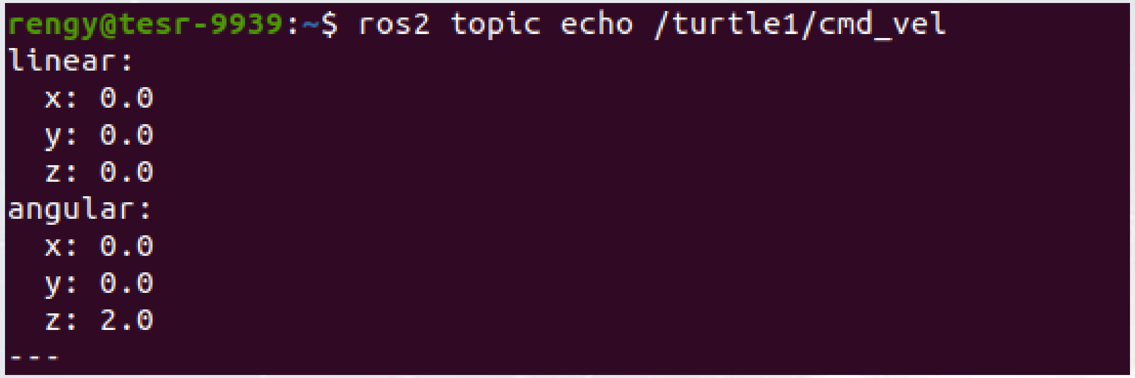


```
rengy@tesr-9939:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

Understanding topics

- **topics echo**

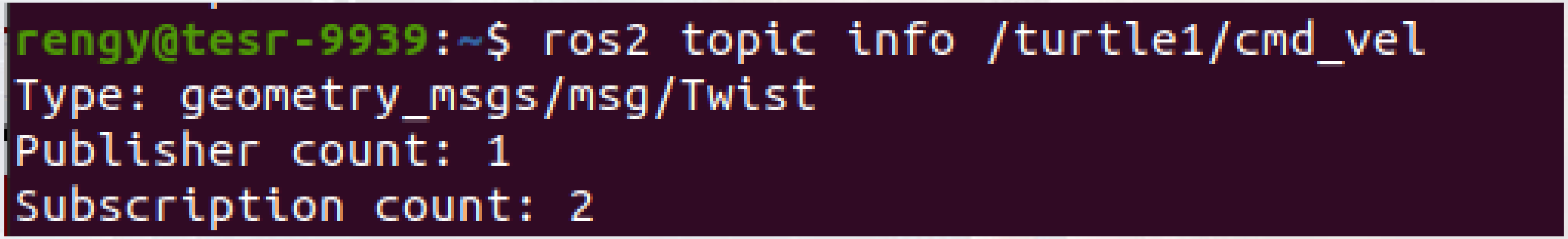
```
ros2 topic echo /turtle1/cmd_vel
```



```
rengy@tesr-9939:~$ ros2 topic echo /turtle1/cmd_vel
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 2.0
---
```

- **topic info**

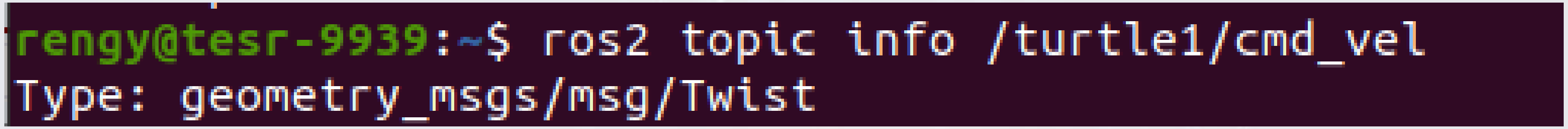
```
ros2 topic info /turtle1/cmd_vel
```



```
rengy@tesr-9939:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 2
```

Understanding topics

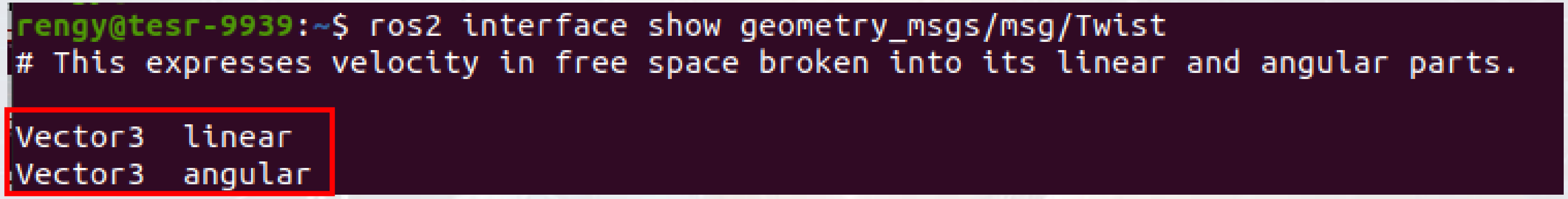
- **Interface show**
  - From info show type of messages



```
rengy@tesr-9939:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
```

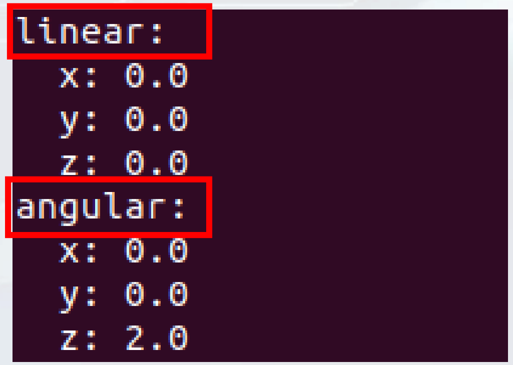
```
ros2 interface show geometry_msgs/msg/Twist
```

  - Interface show output



```
rengy@tesr-9939:~$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its linear and angular parts.

Vector3  linear
Vector3  angular
```

* This show that /turtlesim expecting message with two vectors, linear and angular.



```
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 2.0
```

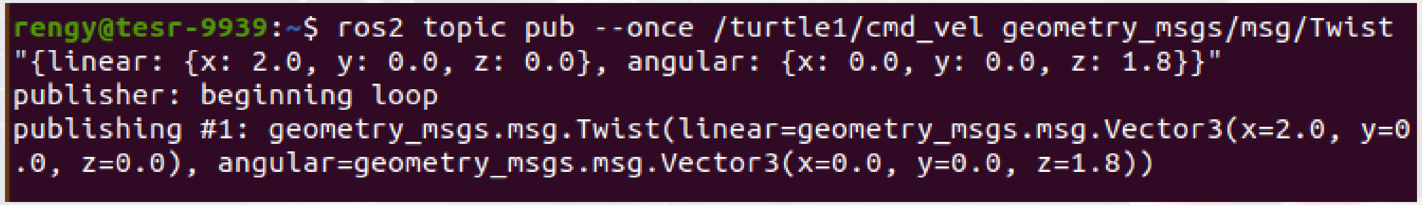
Understanding topics

- **topic pub**
  - Now that you have the message structure, you can publish data onto a topic directly from the command line using:

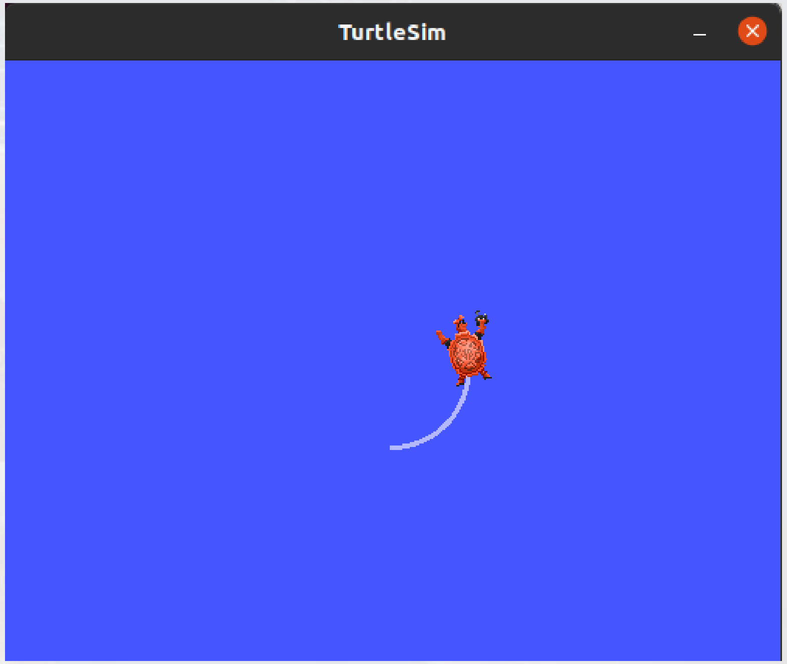
```
ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

  - Output should be like this:

```
publisher: beginning loop

publishing#1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vecto
r3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0,
z=1.8))
```



```
rengy@tesr-9939:~$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist
"{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0
.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))
```

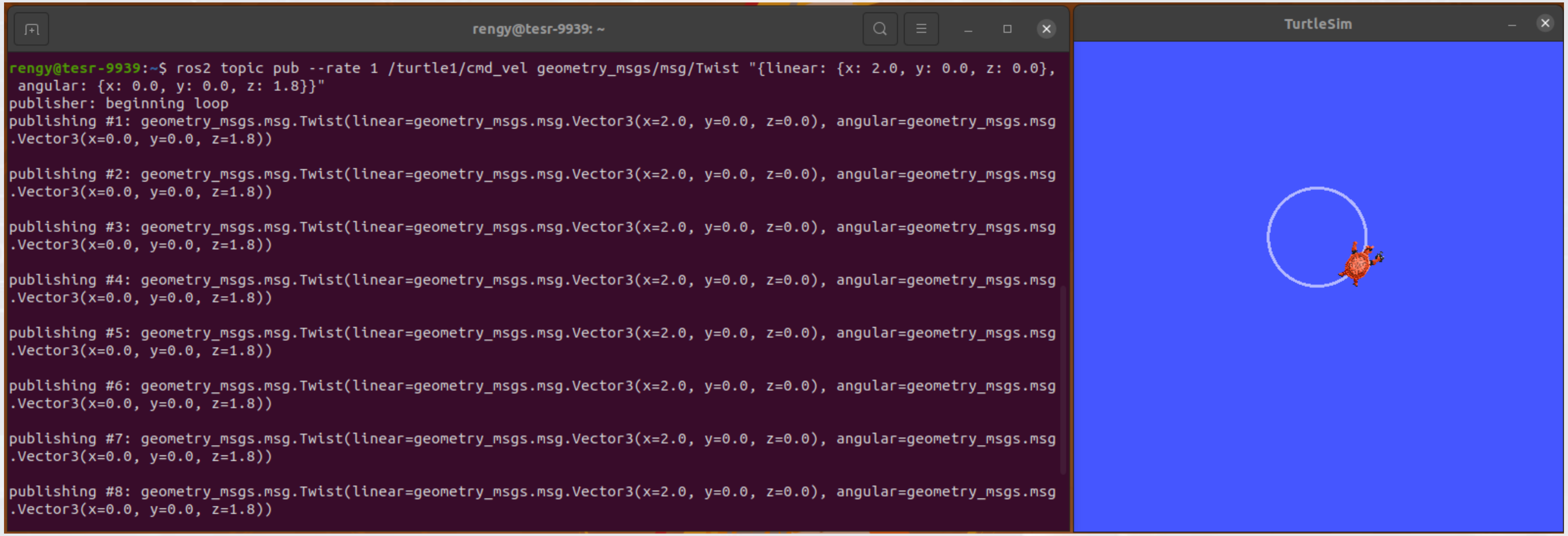


Understanding topics

- **topic pub**
  - You can command the turtle to keep moving you can change **"--once"** in command line to **"--rate 1"** which tell **"ros2 topic pub"** to publish the command in steady stream at 1 Hz.

```
ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

  - Output should be like this:



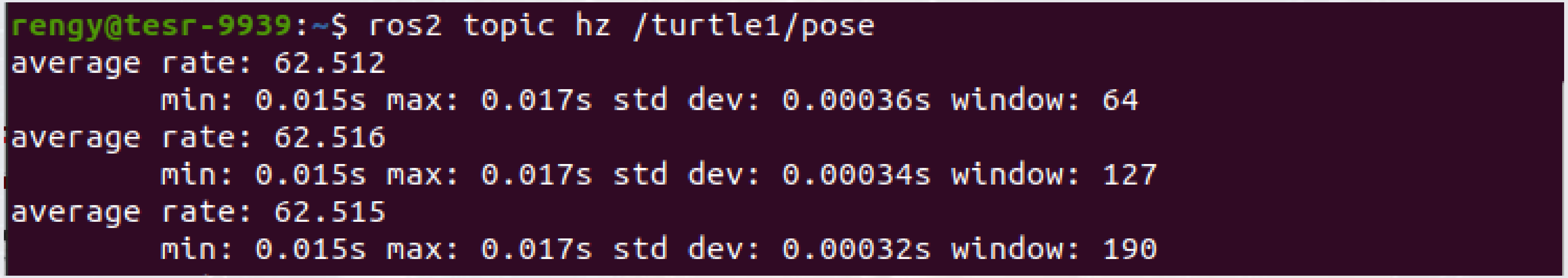
```
rengy@tesr-9939:~$ ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0},
 angular: {x: 0.0, y: 0.0, z: 1.8}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg
.Vector3(x=0.0, y=0.0, z=1.8))

publishing #2: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg
.Vector3(x=0.0, y=0.0, z=1.8))

publishing #3: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg
.Vector3(x=0.0, y=0.0, z=1.8))

publishing #4: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg
.Vector3(x=0.0, y=0.0, z=1.8))

publishing #5: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg
.Vector3(x=0.0, y=0.0, z=1.8))

publishing #6: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg
.Vector3(x=0.0, y=0.0, z=1.8))

publishing #7: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg
.Vector3(x=0.0, y=0.0, z=1.8))

publishing #8: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg
.Vector3(x=0.0, y=0.0, z=1.8))
```

Understanding topics

- **topic hz**
  - you can view the rate at which data is published using:

```
ros2 topic hz /turtle1/pose
```

  - Output should be like this:

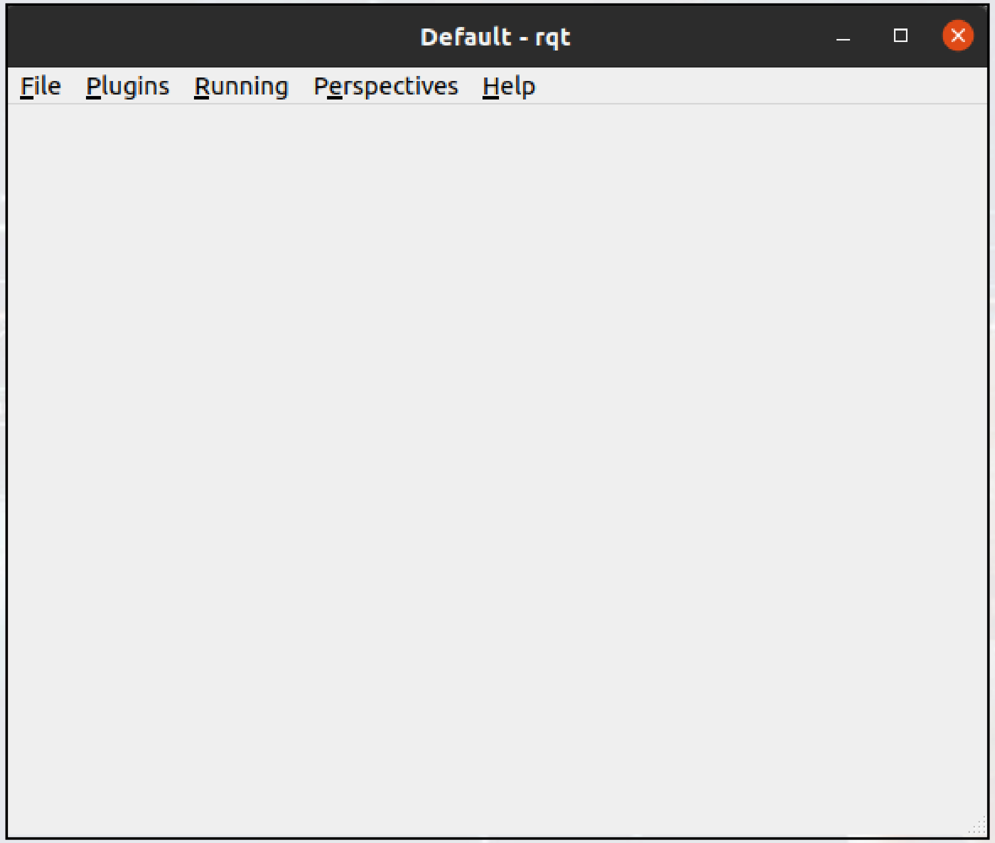


```
rengy@tesr-9939:~$ ros2 topic hz /turtle1/pose
average rate: 62.512
        min: 0.015s max: 0.017s std dev: 0.00036s window: 64
average rate: 62.516
        min: 0.015s max: 0.017s std dev: 0.00034s window: 127
average rate: 62.515
        min: 0.015s max: 0.017s std dev: 0.00032s window: 190
```

Use rqt

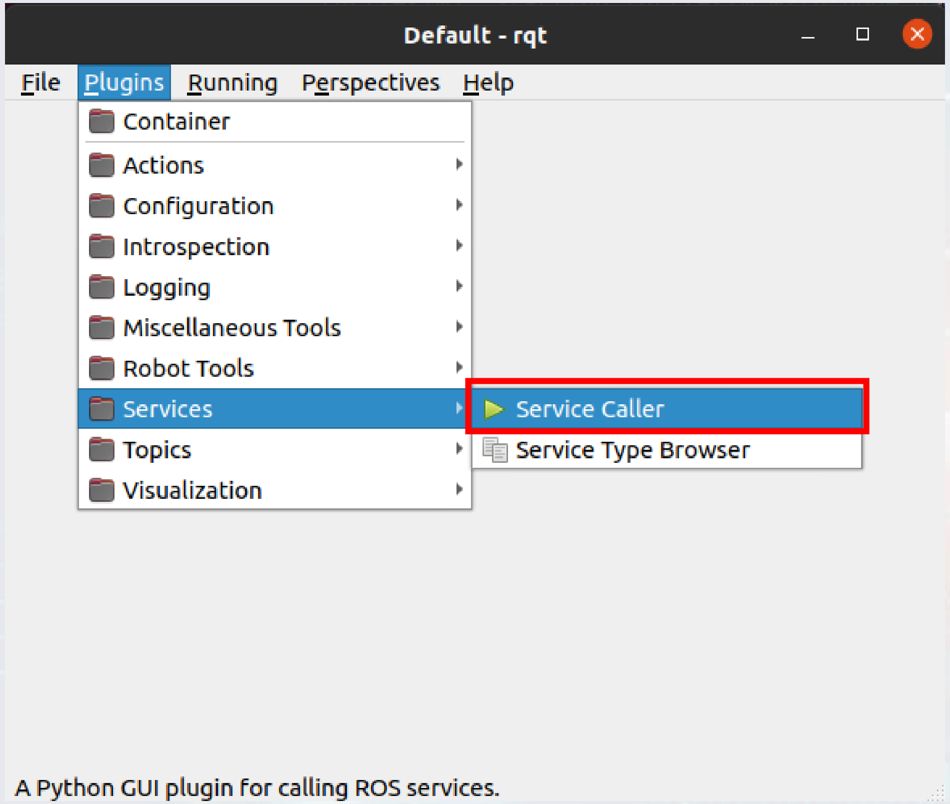
- Open the terminal and run "rqt"

```
rqt
```

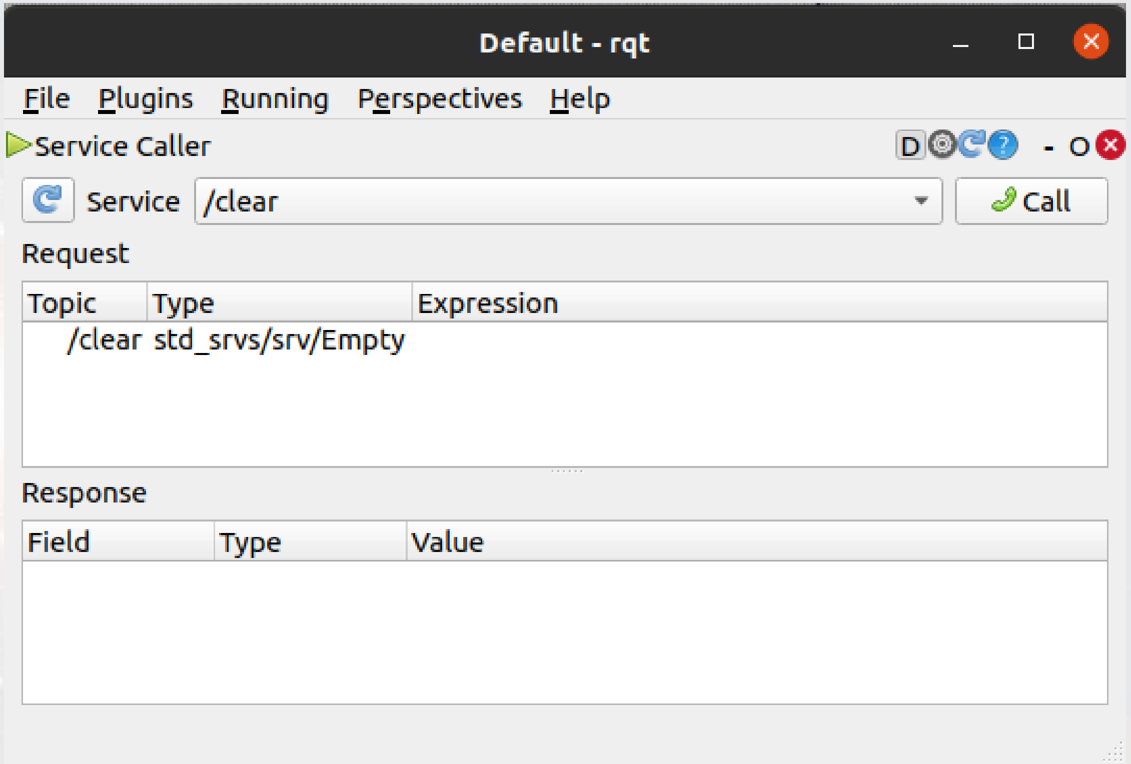


Use rqt

- rqt - will empty for the first operate. So, select Plugin > Services > Sevice Caller



A Python GUI plugin for calling ROS services.



Service /clear

Request

Topic	Type	Expression
/clear	std_srvs/srv/Empty	

Response

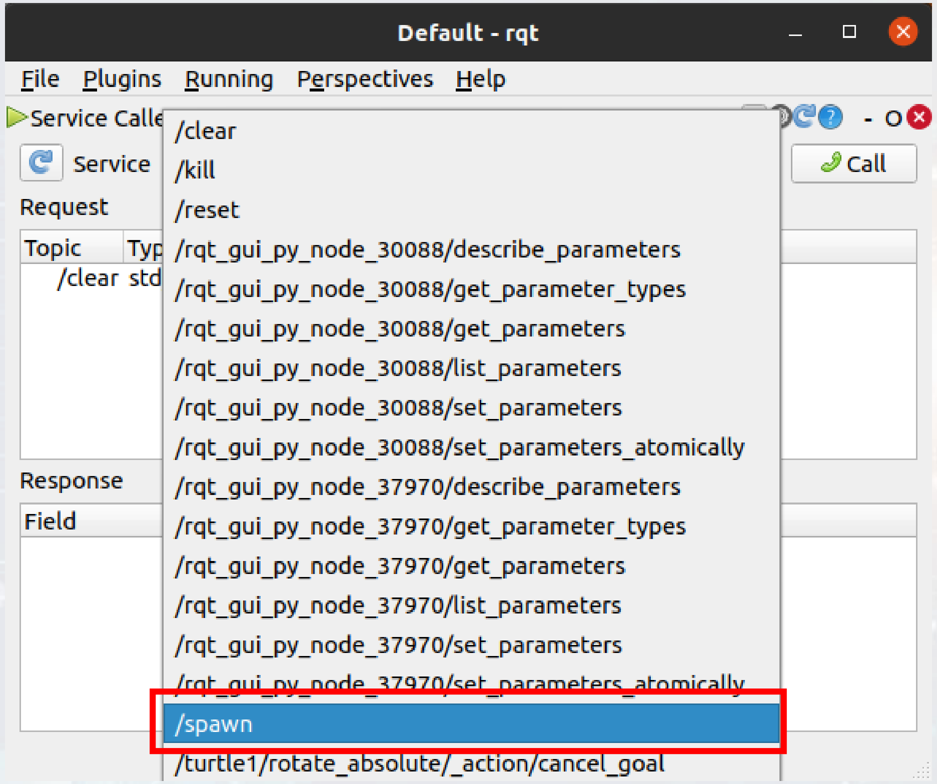
Field	Type	Value

*Use refresh button to the left of Service to ensure all services of your node are available.

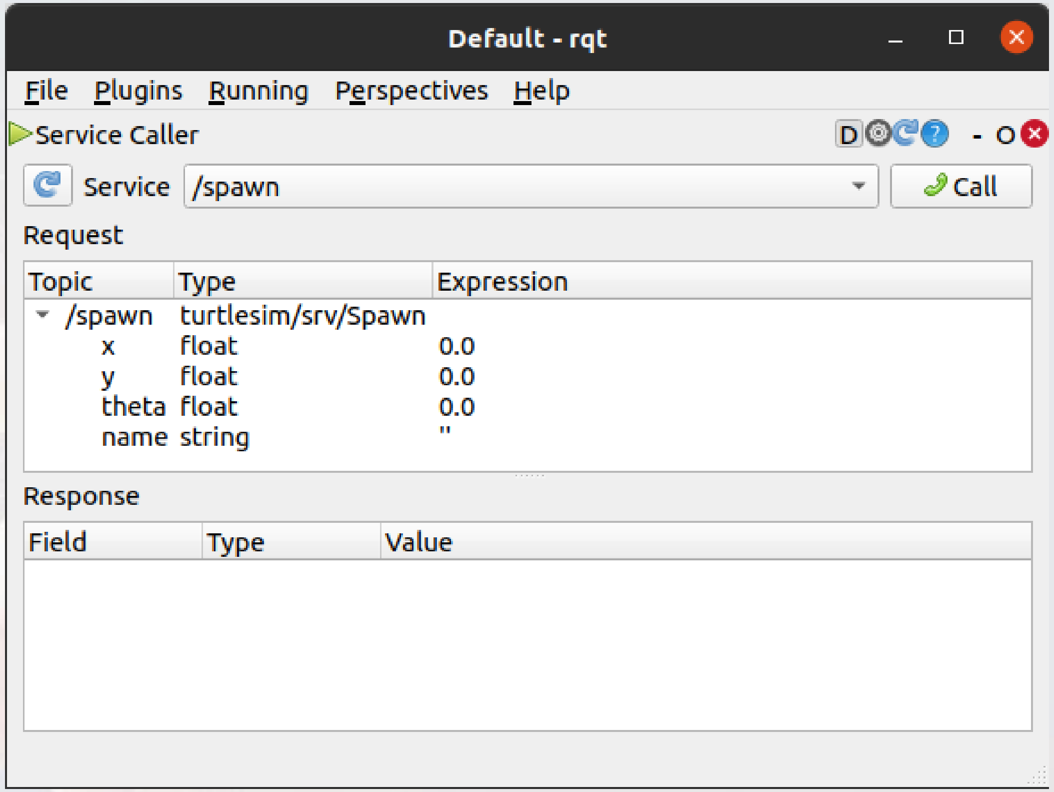
Use rqt

Try the spawn service

- First, click on **Service dropdown** list and select the **/spawn** service.



/clear
/kill
/reset
/rqt_gui_py_node_30088/describe_parameters
/rqt_gui_py_node_30088/get_parameter_types
/rqt_gui_py_node_30088/get_parameters
/rqt_gui_py_node_30088/list_parameters
/rqt_gui_py_node_30088/set_parameters
/rqt_gui_py_node_30088/set_parameters_atomically
/rqt_gui_py_node_37970/describe_parameters
/rqt_gui_py_node_37970/get_parameter_types
/rqt_gui_py_node_37970/get_parameters
/rqt_gui_py_node_37970/list_parameters
/rqt_gui_py_node_37970/set_parameters
/rqt_gui_py_node_37970/set_parameters_atomically
/spawn
/turtle1/rotate_absolute/_action/cancel_goal



Service /spawn

Request

Topic	Type	Expression
/spawn	turtlesim/srv/Spawn	
x	float	0.0
y	float	0.0
theta	float	0.0
name	string	''

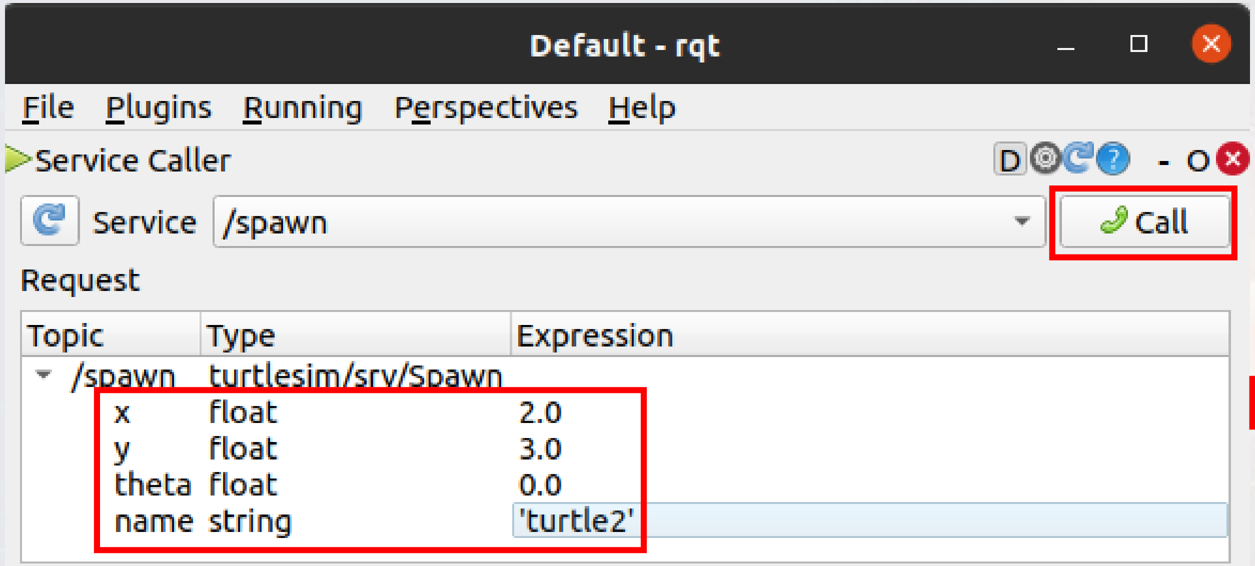
Response

Field	Type	Value

Use rqt

Try the spawn service

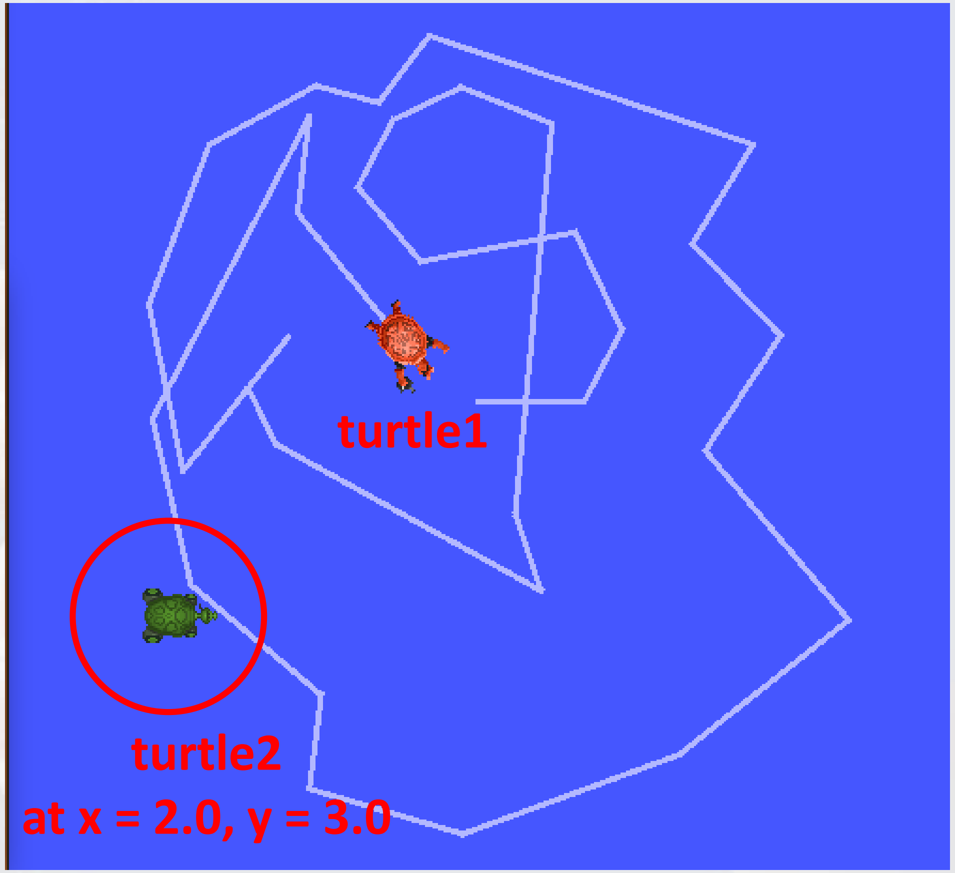
- So, we can renew the coordinate and name of turtle. Such as "x = 2.0, y = 3.0, name = turtle2" and then, Click "Call" to spawn a "turtle2"



Service /spawn

Request

Topic	Type	Expression
/spawn	turtlesim/srv/Spawn	
x	float	2.0
y	float	3.0
theta	float	0.0
name	string	'turtle2'



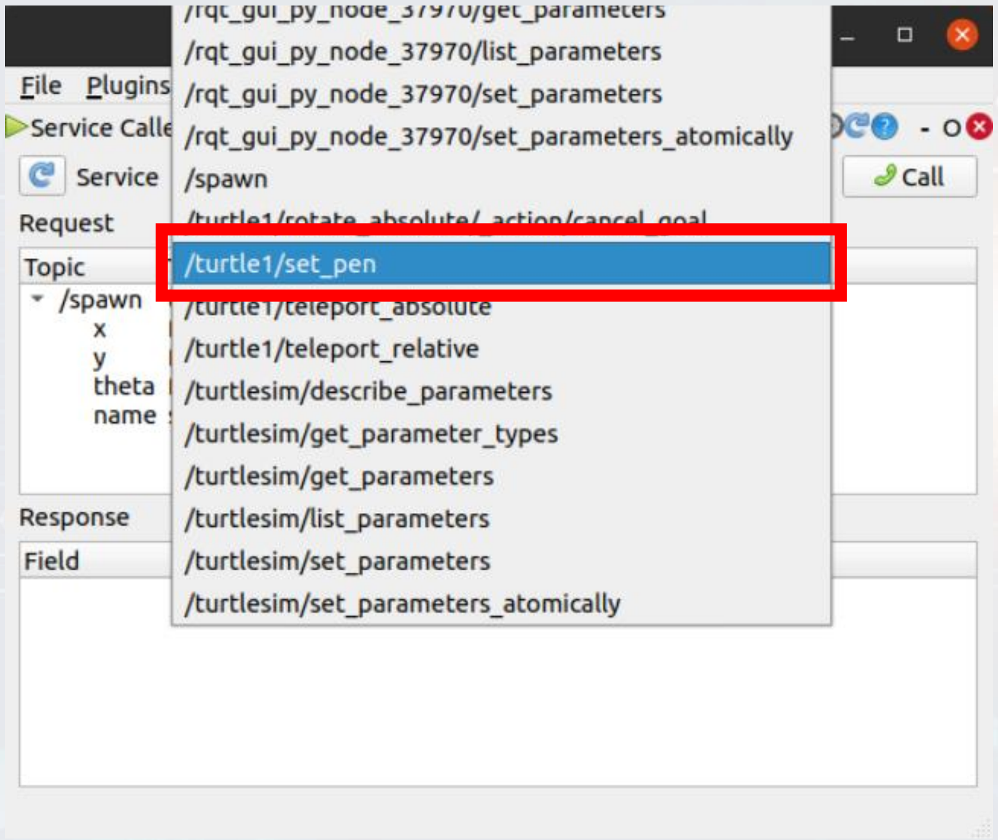
*If you name the spawn turtle same as existing turtle, such as "turtle1" you will get an error message like this;

```
[ERROR] [turtlesim]: A turtle named [turtle1] already exists
```

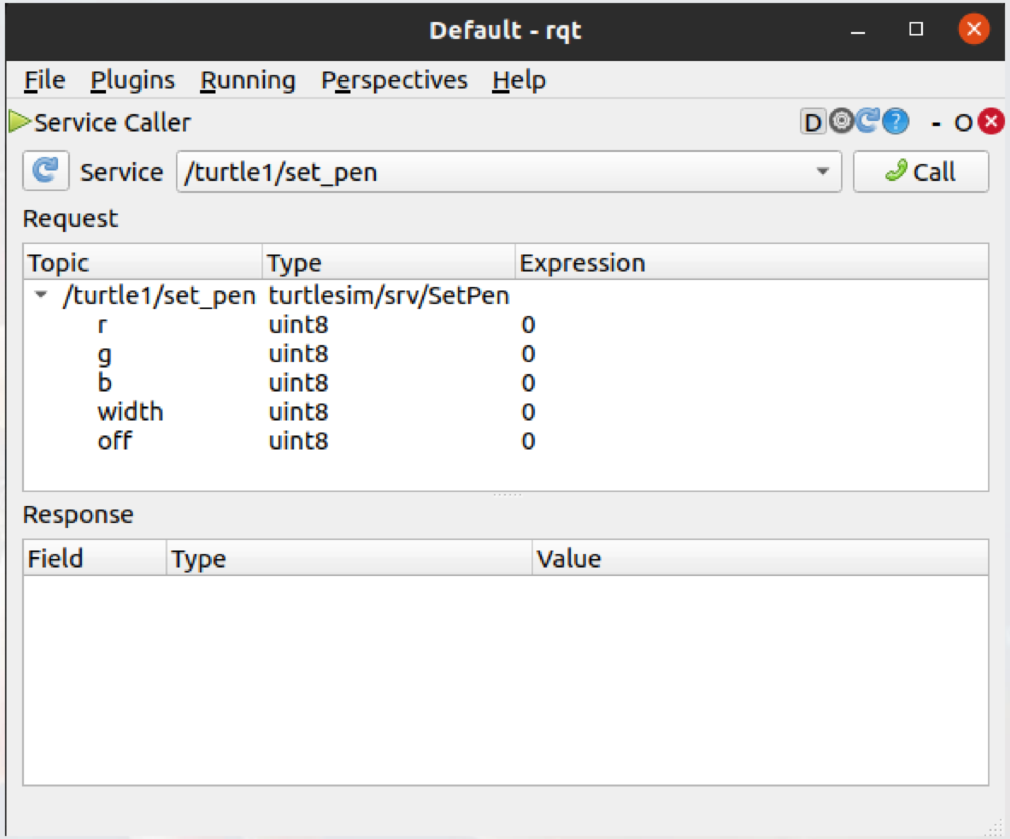
Use rqt

Try the set_pen service

- Now, we have 2 turtle on same panel. So, let give turtle1 unique pen using the /set_pen service.



```
/rqt_gui_py_node_37970/get_parameters
/rqt_gui_py_node_37970/list_parameters
/rqt_gui_py_node_37970/set_parameters
/rqt_gui_py_node_37970/set_parameters_atomically
/spawn
/turtle1/rotate_absolute/_action/cancel_goal
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
```



Service /turtle1/set_pen

Request

Topic	Type	Expression
/turtle1/set_pen	turtlesim/srv/SetPen	
r	uint8	0
g	uint8	0
b	uint8	0
width	uint8	0
off	uint8	0

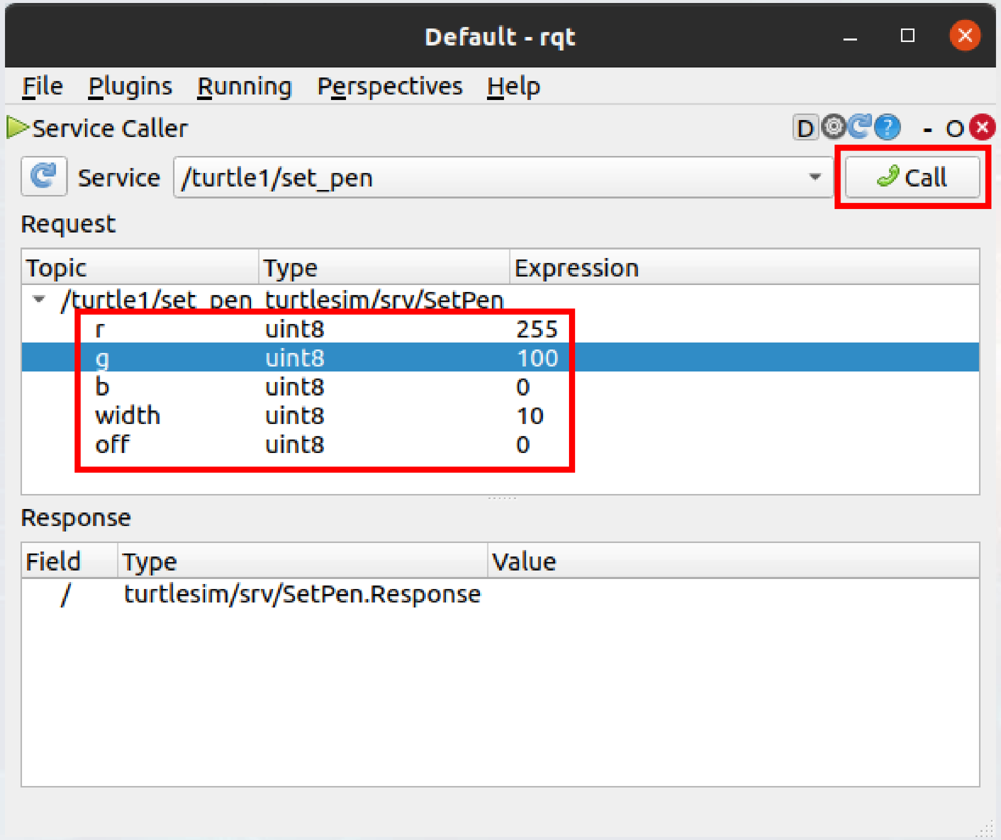
Response

Field	Type	Value

Use rqt

Try the set_pen service

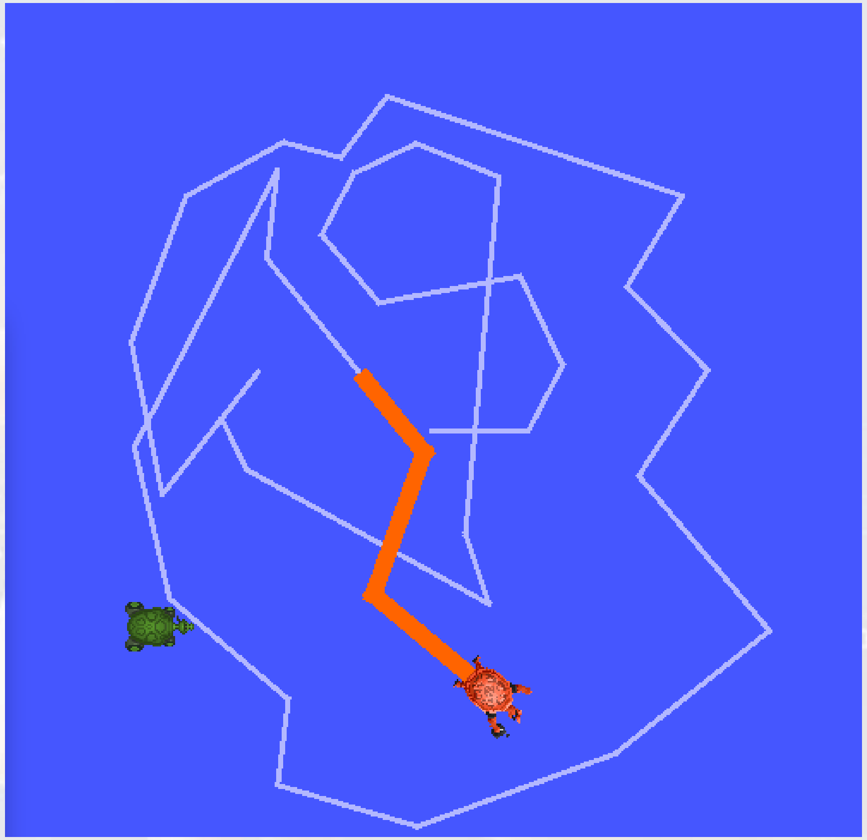
- We can set the pen's color by edit r, g, b and pen's size by edit width. And then click **"Call"**.



Service /turtle1/set_pen

Topic	Type	Expression
/turtle1/set_pen	turtlesim/srv/SetPen	
r	uint8	255
g	uint8	100
b	uint8	0
width	uint8	10
off	uint8	0

Field	Type	Value
/	turtlesim/srv/SetPen.Response	



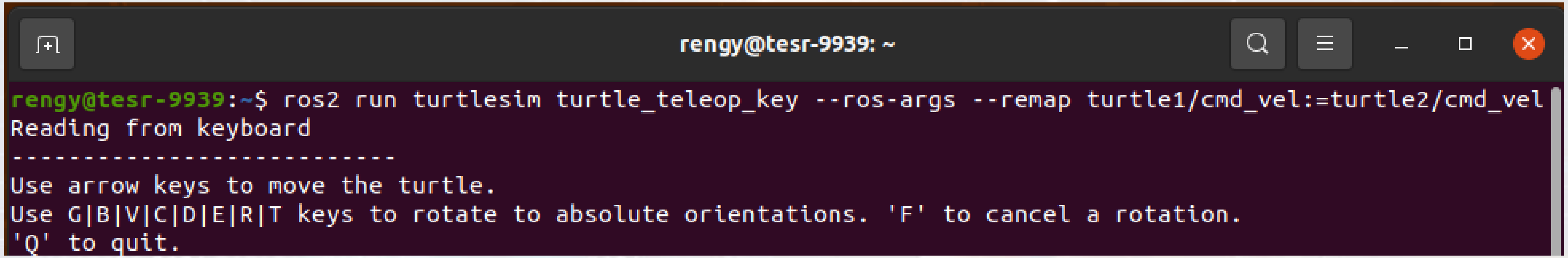
Use rqt

Let's try to remapping the msgs from turtle1 to turtle2

- In new terminal, source ROS 2 and then run:

```
ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/cmd_vel:=turtle2/cmd_vel
```

- Output on terminal should show like this:

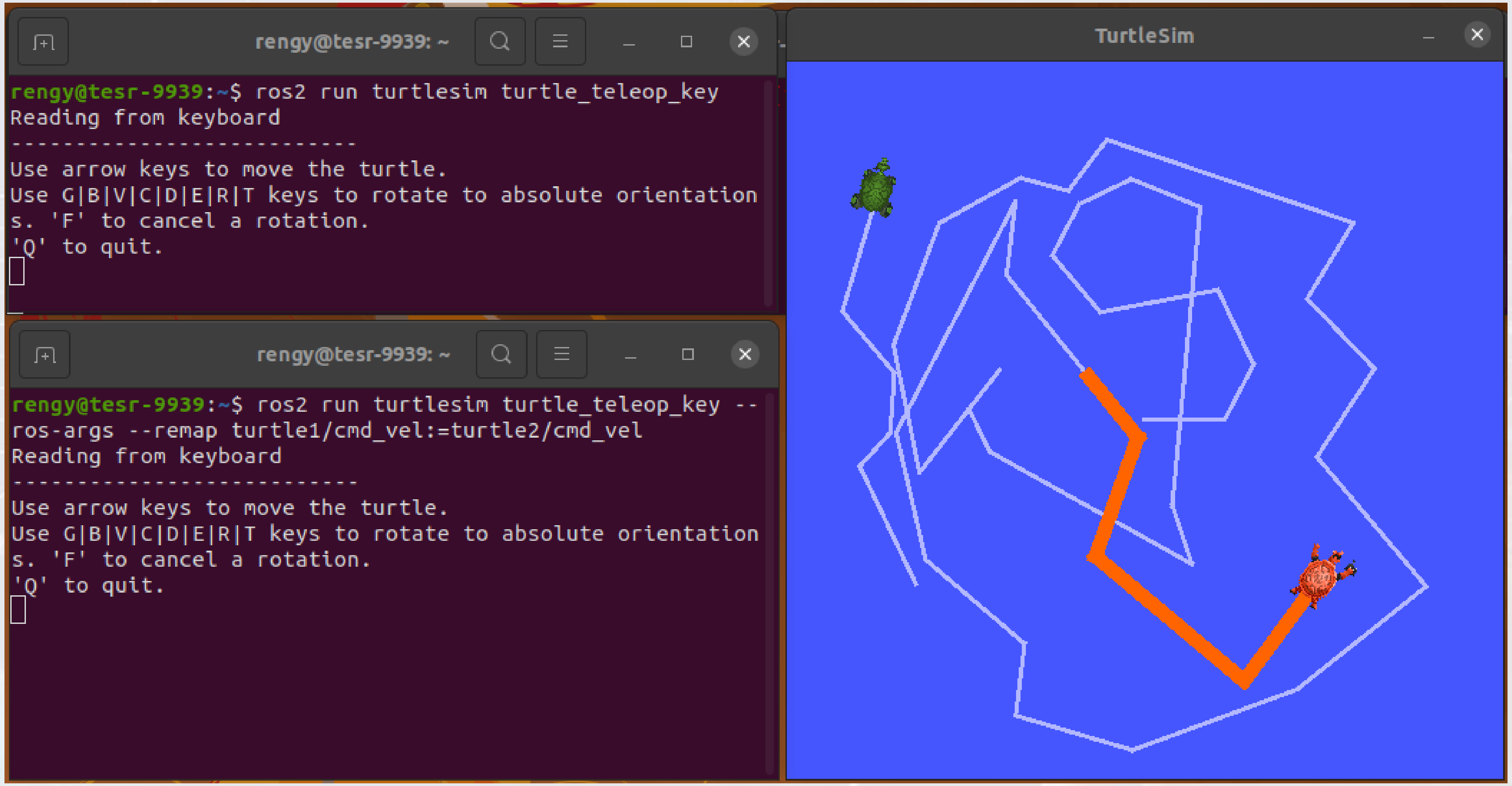


```
rengy@tesr-9939:~$ ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/cmd_vel:=turtle2/cmd_vel
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
'Q' to quit.
```

Use rqt

Now you can control turtle2 and turtle1 separately on each terminal

- Output may be like this



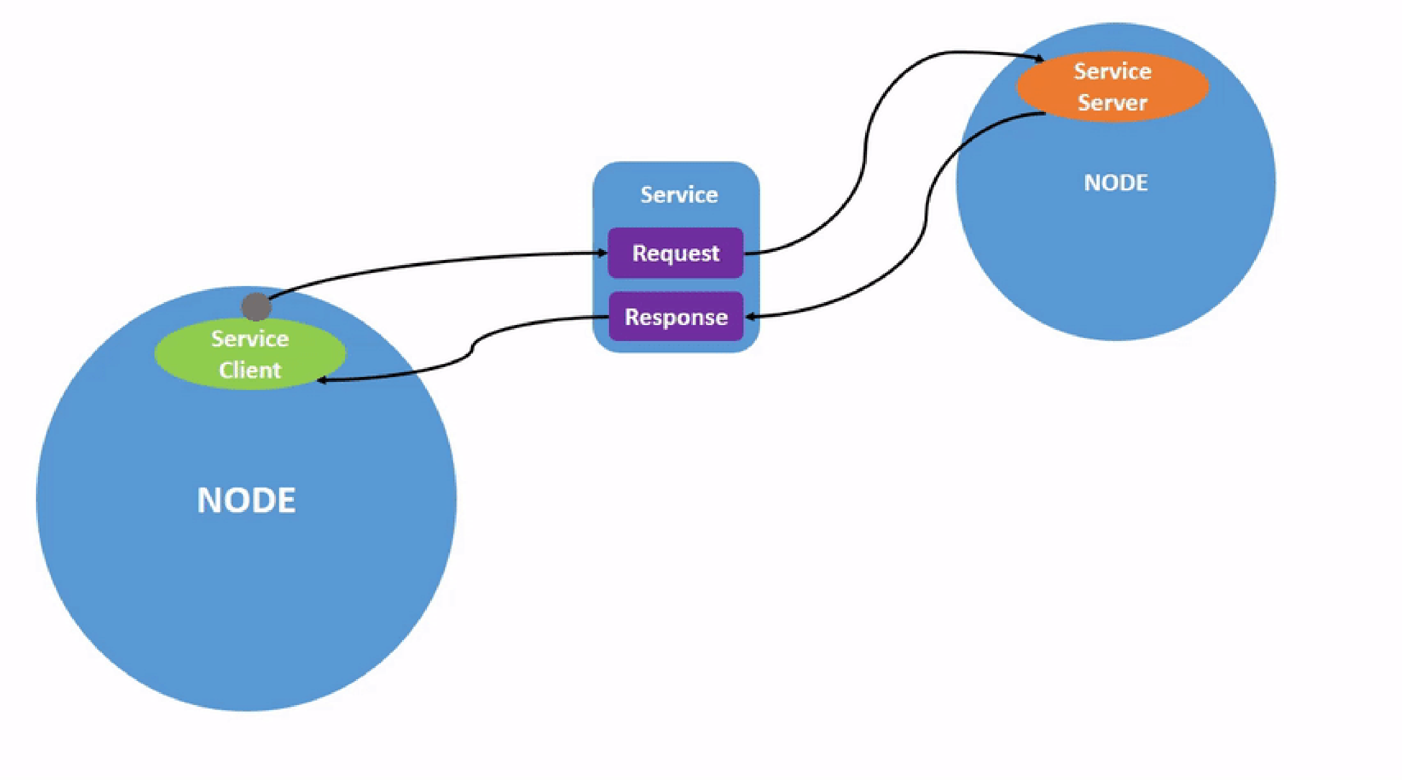
```
rengy@tesr-9939:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientation
s. 'F' to cancel a rotation.
'Q' to quit.
```

```
rengy@tesr-9939:~$ ros2 run turtlesim turtle_teleop_key --
ros-args --remap turtle1/cmd_vel:=turtle2/cmd_vel
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientation
s. 'F' to cancel a rotation.
'Q' to quit.
```

Understanding services

Services is another method ROS2 use to communicate. **Compare to Topic model**, that allow nodes to data streams for continual update. **Service** only provide data when called by client.

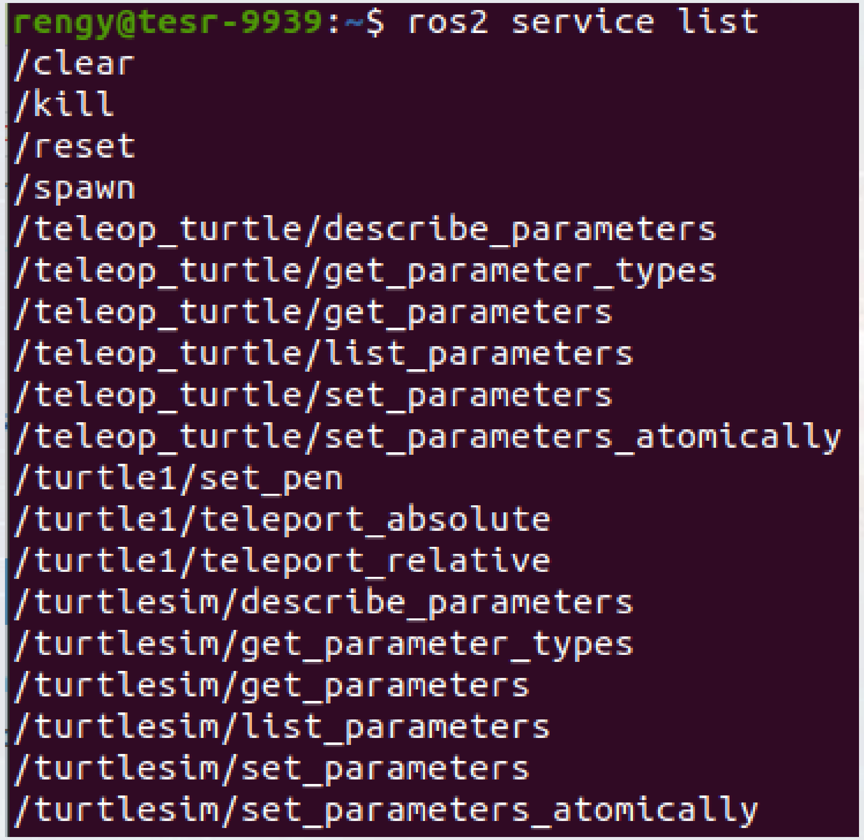
- This **ROS graph** show the **Request** and **Response** message pass through the Service by NODE's **Service Client** to NODE's **Service Server**.



Understanding services

- **ros2 service list**

```
ros2 service list
```



```
rengy@tesr-9939:~$ ros2 service list
/clear
/kill
/reset
/spawn
/teleop_turtle/describe_parameters
/teleop_turtle/get_parameter_types
/teleop_turtle/get_parameters
/teleop_turtle/list_parameters
/teleop_turtle/set_parameters
/teleop_turtle/set_parameters_atomically
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
```

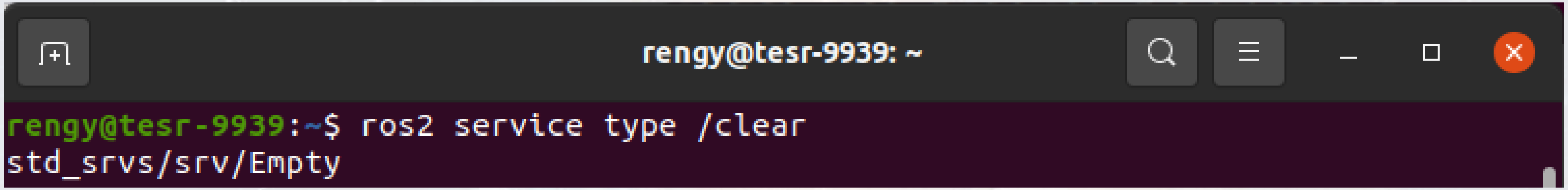
Understanding services

- **service type**

```
ros2 service type /clear
```

Output should return:

```
std_srvs/srv/Empty
```

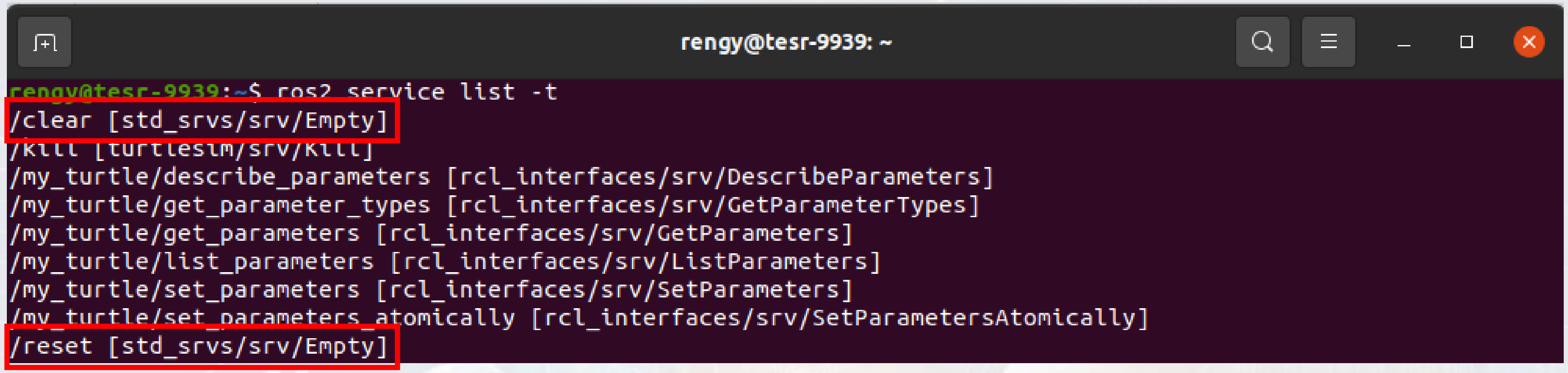


```
rengy@tesr-9939:~$ ros2 service type /clear
std_srvs/srv/Empty
```

Understanding services

- **service list type**
  - To see the type of all services you can also add **"-t"** after **"list"** command:

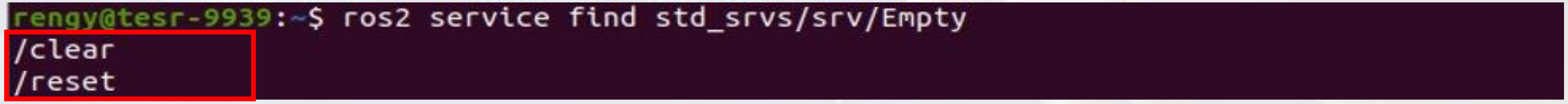
```
ros2 service list -t
```



```
rengy@tesr-9939:~$ ros2 service list -t
/clear [std_srvs/srv/Empty]
/kill [turtlesim/srv/Kill]
/my_turtle/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/my_turtle/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/my_turtle/get_parameters [rcl_interfaces/srv/GetParameters]
/my_turtle/list_parameters [rcl_interfaces/srv/ListParameters]
/my_turtle/set_parameters [rcl_interfaces/srv/SetParameters]
/my_turtle/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
/reset [std_srvs/srv/Empty]
```

- **service find**
  - You can find all of the service that have the same type. Using this command below:

```
ros2 service find std_srvs/srv/Empty
```



```
rengy@tesr-9939:~$ ros2 service find std_srvs/srv/Empty
/clear
/reset
```

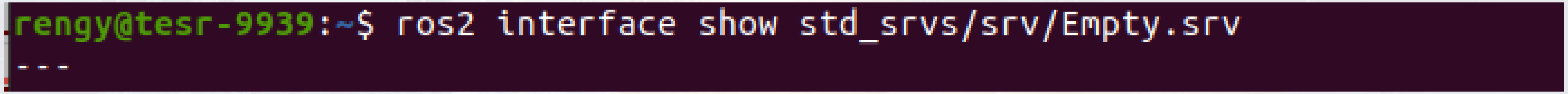
Understanding services

- **interface show**
  - You can show the response structure of service. Using this command below:

```
ros2 interface show std_srvs/srv/Empty.srv
```

  - Output will return:

```
---
```

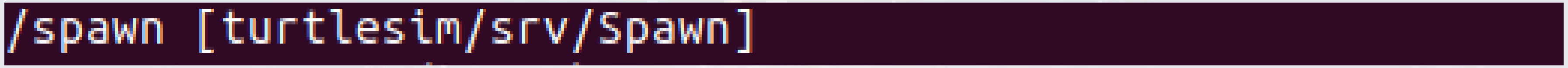


```
rengy@tesr-9939:~$ ros2 interface show std_srvs/srv/Empty.srv
---
```

* --- use as a seperates the request structure(above) from the response structure(below). But the Empty type doesn't send or receive any data. So, the structure is blank.

Understanding services

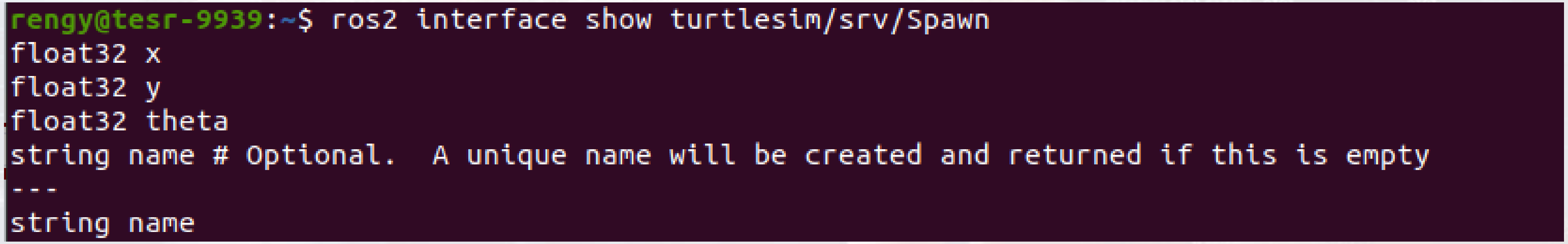
- **interface show**
  - Let’s introspect a service with a type that sends and receives data, like **/spawn**. From the results of **"ros2 service list –t"**, you can see **/spawn** type. (turtlesim/srv/Spawn)



```
/spawn [turtlesim/srv/Spawn]
```

  - So, we can show the response structure of **/spawn** use command below.

```
ros2 interface show turtlesim/srv/Spawn
```



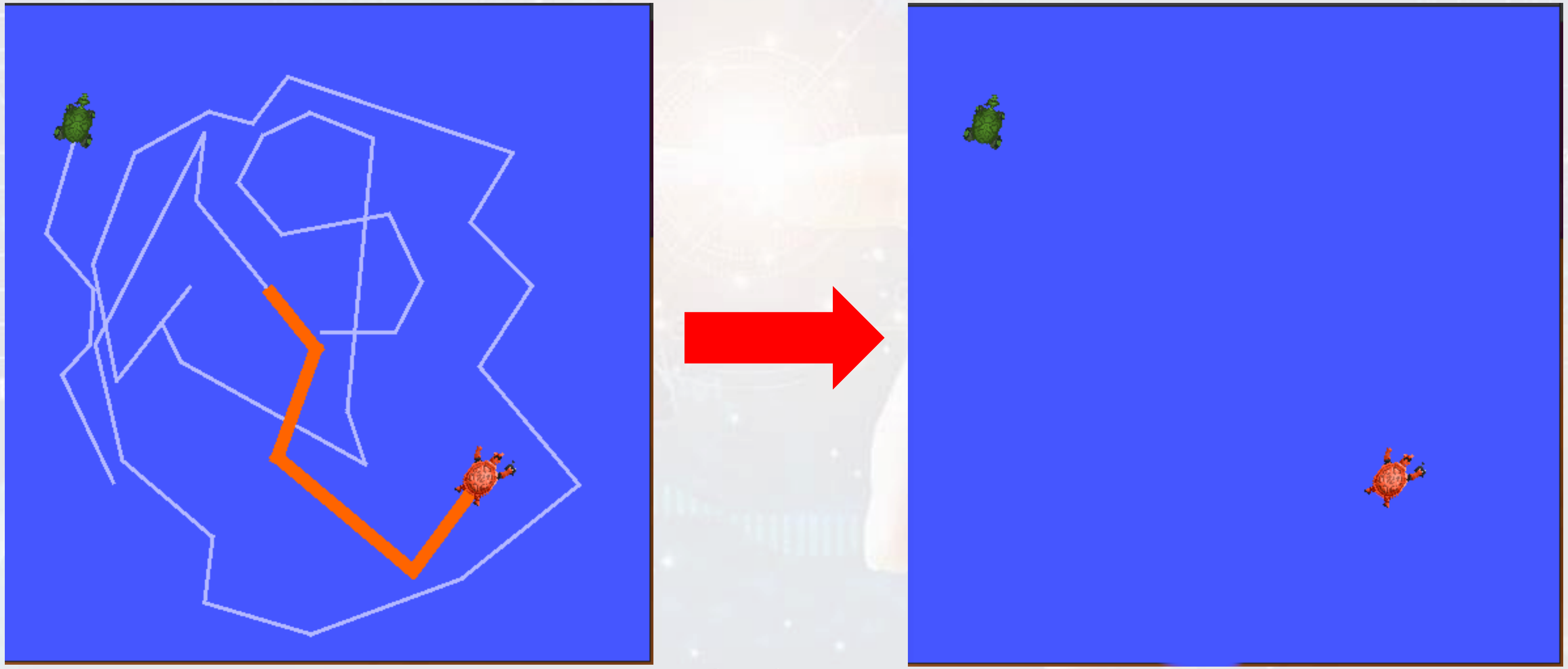
```
rengy@tesr-9939:~$ ros2 interface show turtlesim/srv/Spawn
float32 x
float32 y
float32 theta
string name # Optional.  A unique name will be created and returned if this is empty
---
string name
```

Understanding services

- **Service call**
  - We will try to clean the turtlesim's drawn using /clear service using command below.

```
ros2 service call /clear std_srvs/srv/Empty
```

  - Output after call /clear service.

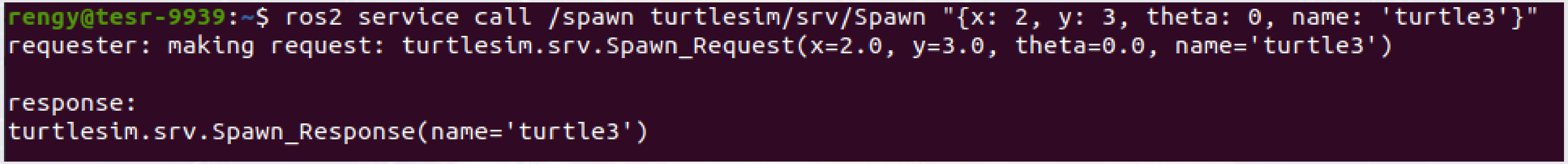


Understanding services

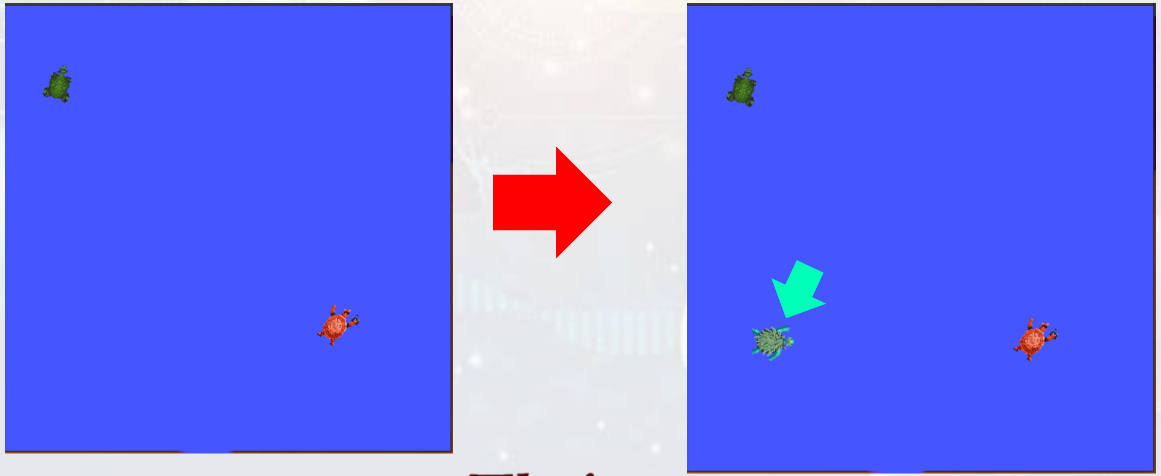
- **Service call**
  - We will also try to spawn a new turtle too..

```
ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 3, theta: 0, name: 'turtle3'}"
```

  - Output after call /clear service.



```
rengy@tesr-9939:~$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 3, theta: 0, name: 'turtle3'}"
requester: making request: turtlesim.srv.Spawn_Request(x=2.0, y=3.0, theta=0.0, name='turtle3')

response:
turtlesim.srv.Spawn_Response(name='turtle3')
```

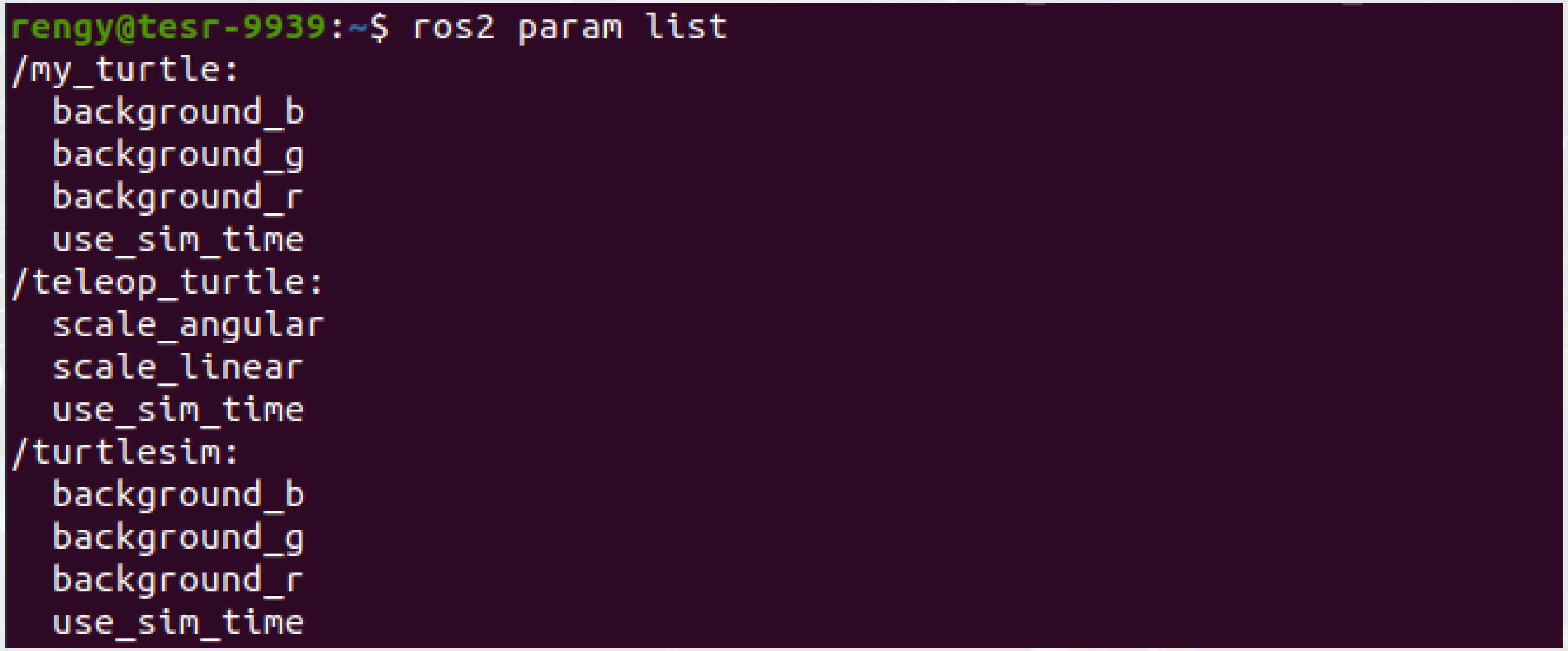


Understanding parameters

Parameters is a configuration value of a node. You can think of parameters as node settings. A node can store parameters as **integers, floats, booleans, strings,** and **lists**.

- **ros2 param list**
  - Open terminal and type command below to see the list of all parameters.

```
ros2 param list
```

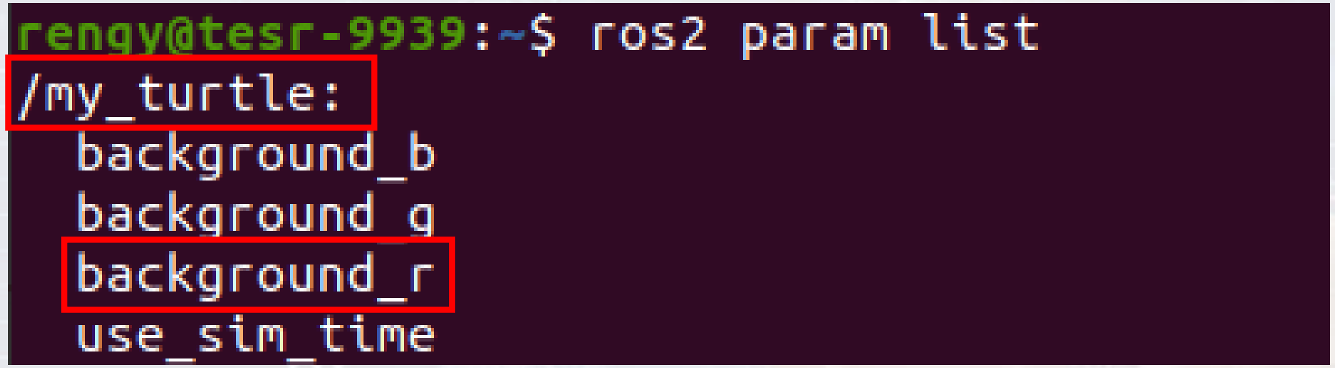


```
rengy@tesr-9939:~$ ros2 param list
/my_turtle:
  background_b
  background_g
  background_r
  use_sim_time
/teleop_turtle:
  scale_angular
  scale_linear
  use_sim_time
/turtlesim:
  background_b
  background_g
  background_r
  use_sim_time
```

Understanding parameters

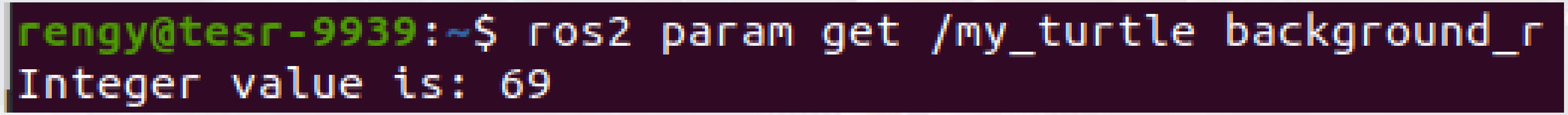
- **param get**
  - To show the value of parameter we use **"param get"** follow by **node** and **param_name** as below:

```
ros2 param get /my_turtle background_r
```



```
rengy@tesr-9939:~$ ros2 param list
/my_turtle:
  background_b
  background_g
  background_r
  use_sim_time
```

  - What output return should be like this.



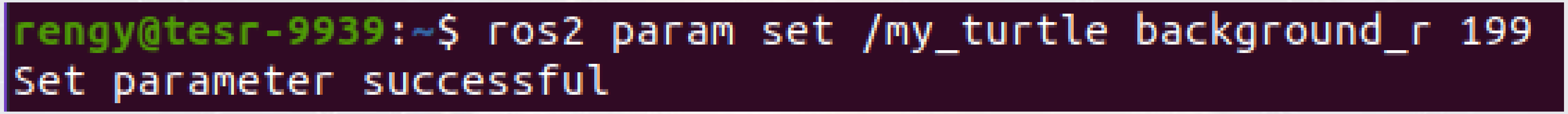
```
rengy@tesr-9939:~$ ros2 param get /my_turtle background_r
Integer value is: 69
```

Understanding parameters

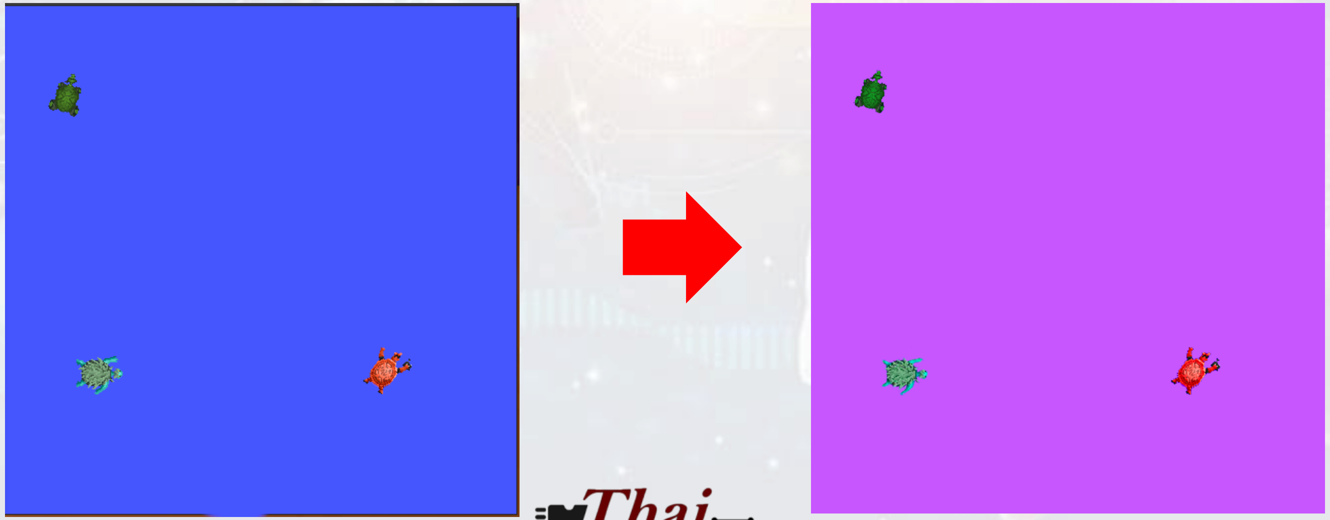
- **param set**
  - To set the value of parameter we use **"param set"** follow by **node, param_name** and **value** you needed as below:

```
ros2 param set /my_turtle background_r 199
```

  - Output and the turtlesim background will change to something like below.



```
rengy@tesr-9939:~$ ros2 param set /my_turtle background_r 199
Set parameter successful
```

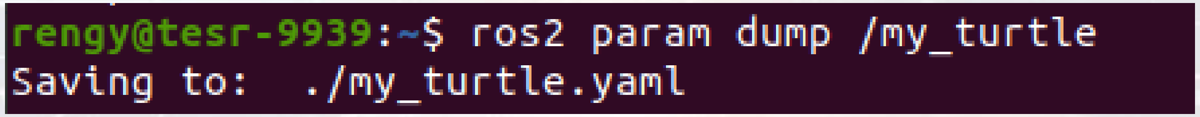


Understanding parameters

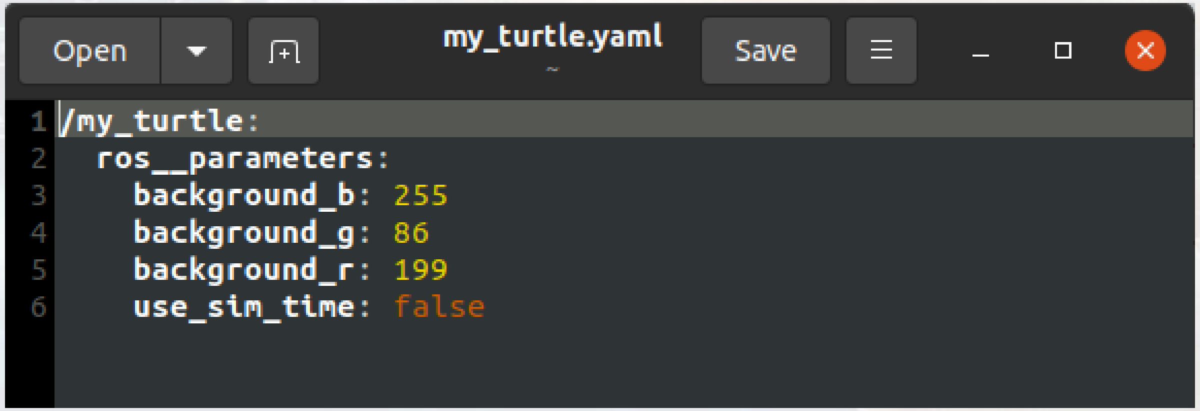
- **param dump**
  - You can dump node's current parameter values into file to save them for later use command below:

```
ros2 param dump /my_turtle
```

  - Output on terminal and inside .yaml should be as below.



```
rengy@tesr-9939:~$ ros2 param dump /my_turtle
Saving to:  ./my_turtle.yaml
```



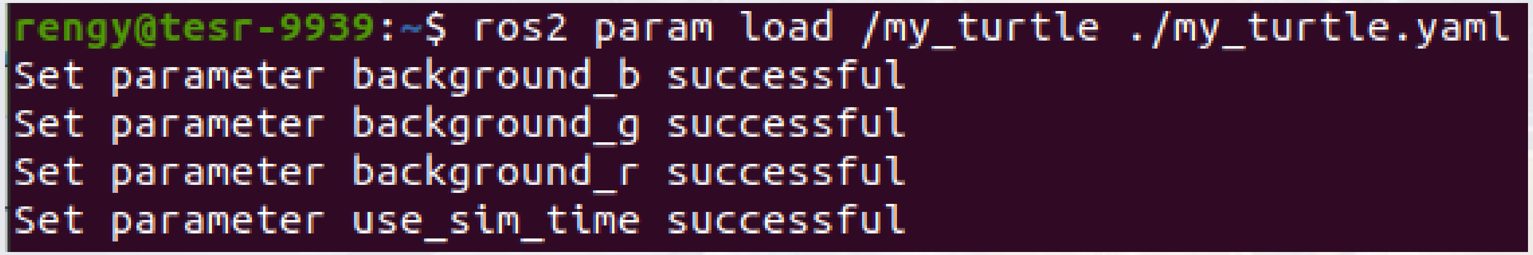
```yaml
/my_turtle:
  ros__parameters:
    background_b: 255
    background_g: 86
    background_r: 199
    use_sim_time: false
```

Understanding parameters

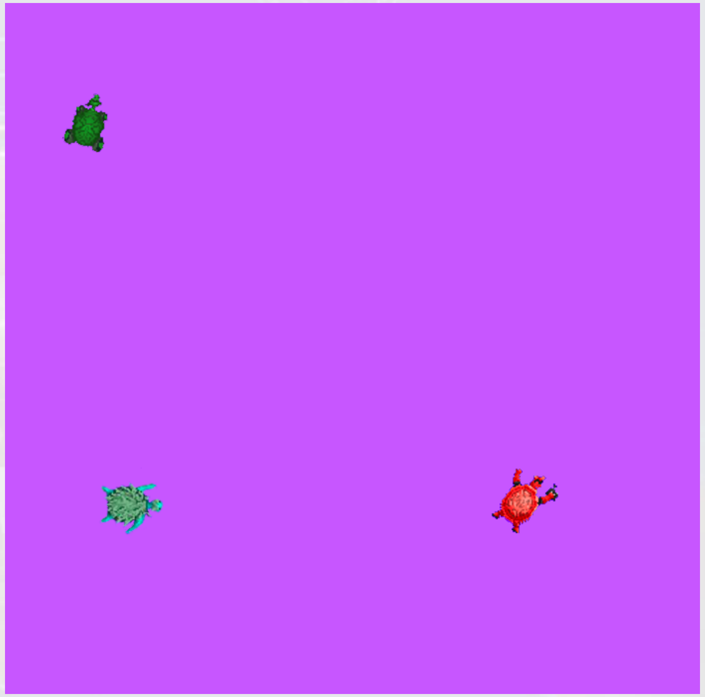
- **param load**
  - You can load parameters from a file to a currently running node using the command:

```
ros2 param load /my_turtle ./my_turtle.yaml
```

  - Output on terminal and turtlesim should be as below.



```
rengy@tesr-9939:~$ ros2 param load /my_turtle ./my_turtle.yaml
Set parameter background_b successful
Set parameter background_g successful
Set parameter background_r successful
Set parameter use_sim_time successful
```



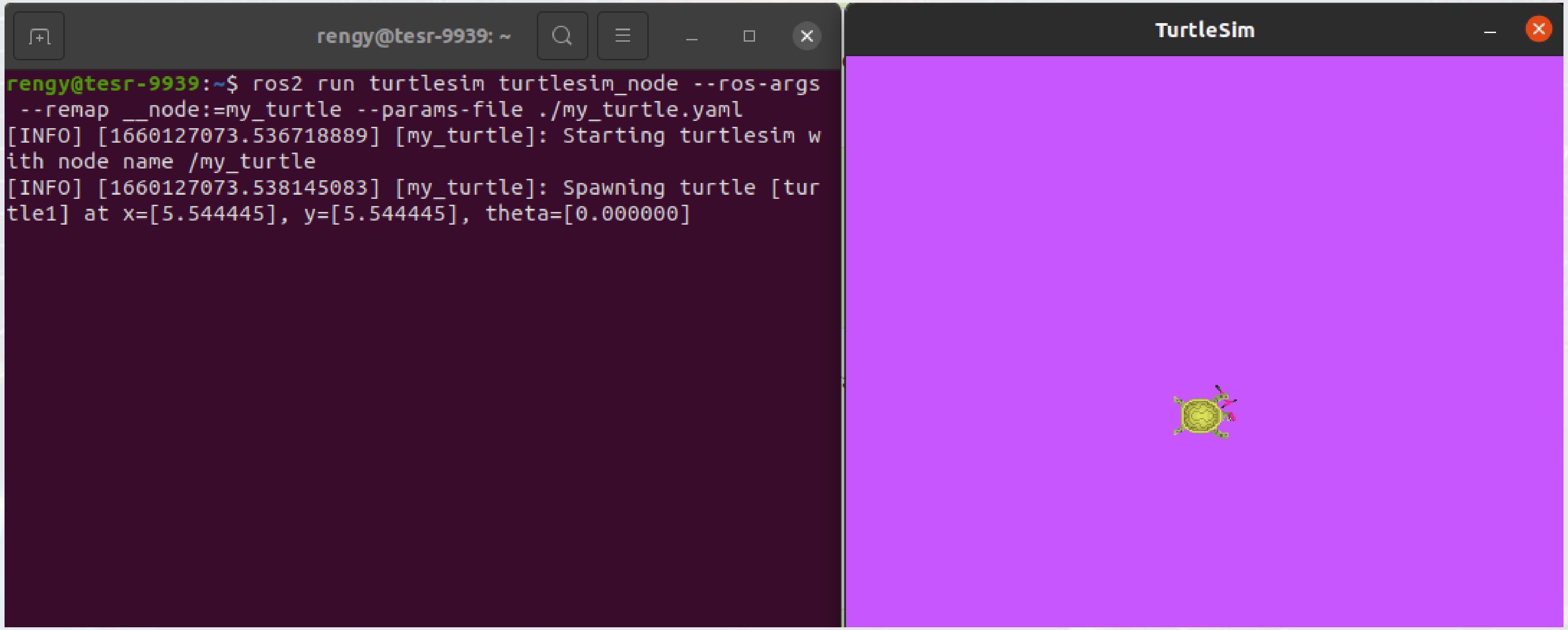
Understanding parameters

- **Load parameter file on node startup**
  - To start the same node using your saved parameter values, use:

    *you must load param on startup with the same name as dumped param file.

```
ros2 run turtlesim turtlesim_node --ros-args --remap __node:=my_turtle --params-file ./my_turtle.yaml
```

  - Output on terminal and turtlesim should be as below.

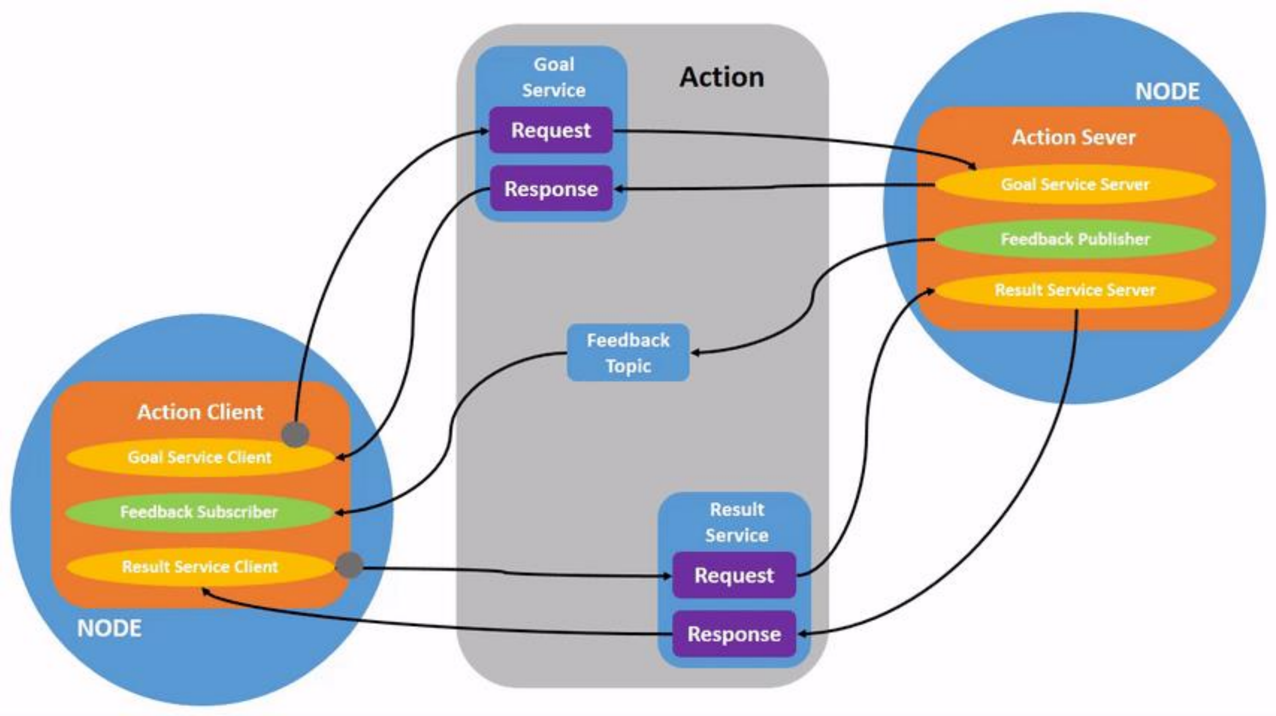


```
rengy@tesr-9939:~$ ros2 run turtlesim turtlesim_node --ros-args
 --remap __node:=my_turtle --params-file ./my_turtle.yaml
[INFO] [1660127073.536718889] [my_turtle]: Starting turtlesim w
ith node name /my_turtle
[INFO] [1660127073.538145083] [my_turtle]: Spawning turtle [tur
tle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

Understanding actions

Action is the method ROS2 use to communicate, consist by **goal, feedback and result.** Action use client-server model **similar to topic's publisher-subscriber model** and **functionality similar to services**, except action are preemptable (you can cancel while executing). They also provide steady feedback, as opposed to services which return a single response.

- This **ROS graph** show the communicate of **NODE's Action Client** to **NODE's Action Server** pass through the **Action** consist by **Goal Service, Feedback Topic** and **Result Service.**

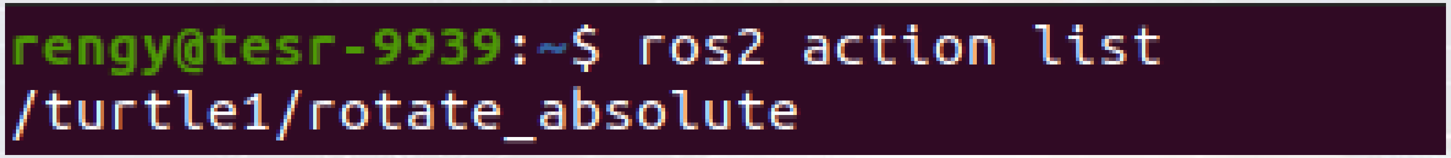


Understanding actions

- **ros2 action list**
  - To identify all the actions in the ROS graph, run the command:

```
ros2 action list
```

  - Output will return as.

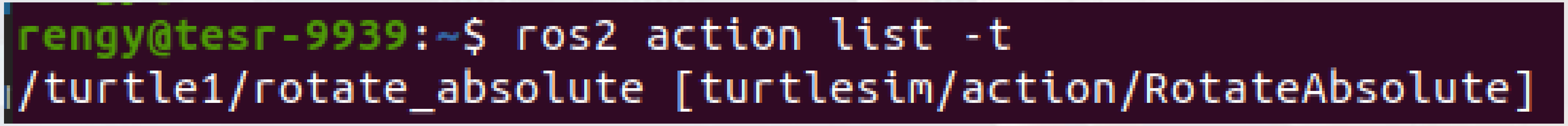


```
rengy@tesr-9939:~$ ros2 action list
/turtle1/rotate_absolute
```

  - Can also show type of action by add "-t" after "list" as below:

```
ros2 action list -t
```

  - Output will return as.



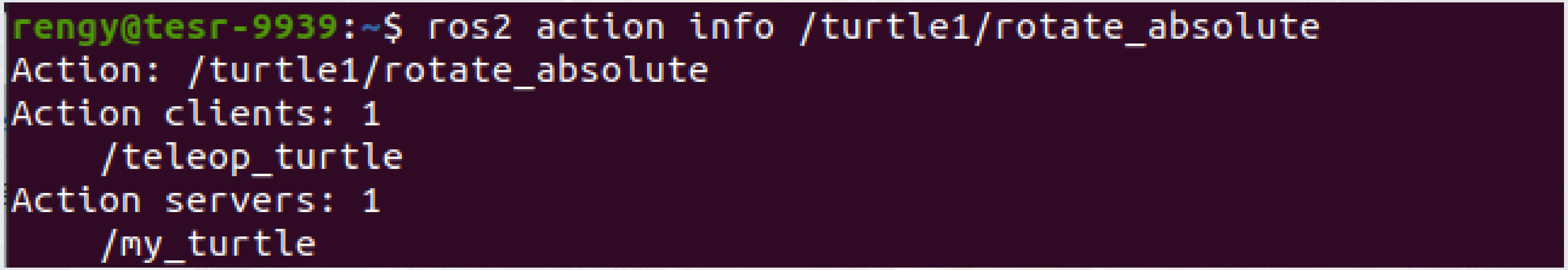
```
rengy@tesr-9939:~$ ros2 action list -t
/turtle1/rotate_absolute [turtlesim/action/RotateAbsolute]
```

Understanding actions

- **action info**
  - You can further introspect the **/turtle1/rotate_absolute** action with the command:

```
ros2 action info /turtle1/rotate_absolute
```

  - Output will return as.



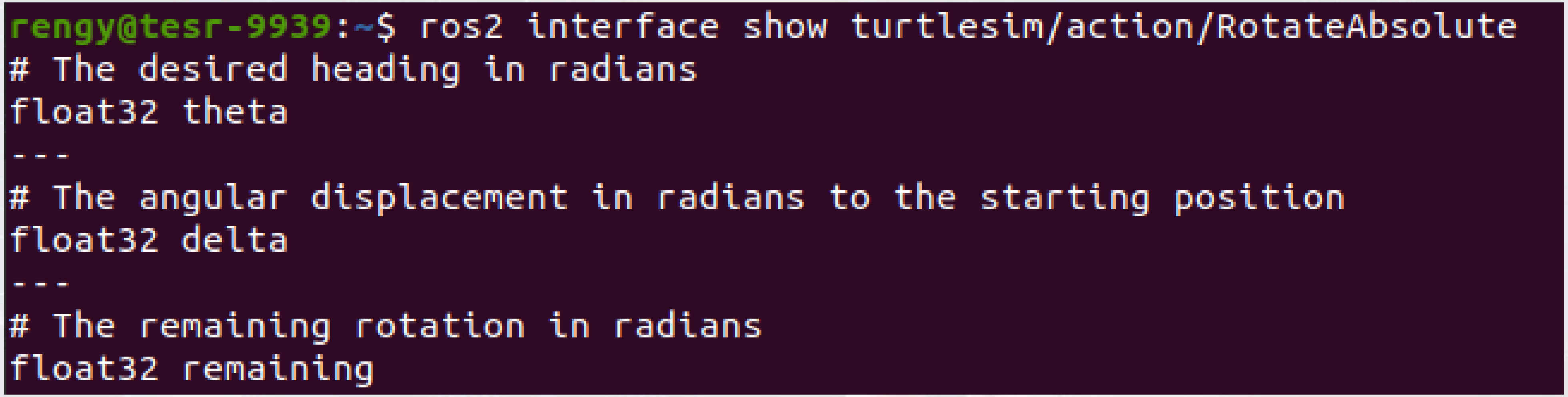
```
rengy@tesr-9939:~$ ros2 action info /turtle1/rotate_absolute
Action: /turtle1/rotate_absolute
Action clients: 1
    /teleop_turtle
Action servers: 1
    /my_turtle
```

Understanding actions

- **Interface show**
  - You must check the structure of action before send or execute an action goal by:

```
ros2 interface show turtlesim/action/RotateAbsolute
```

  - Output will return as.



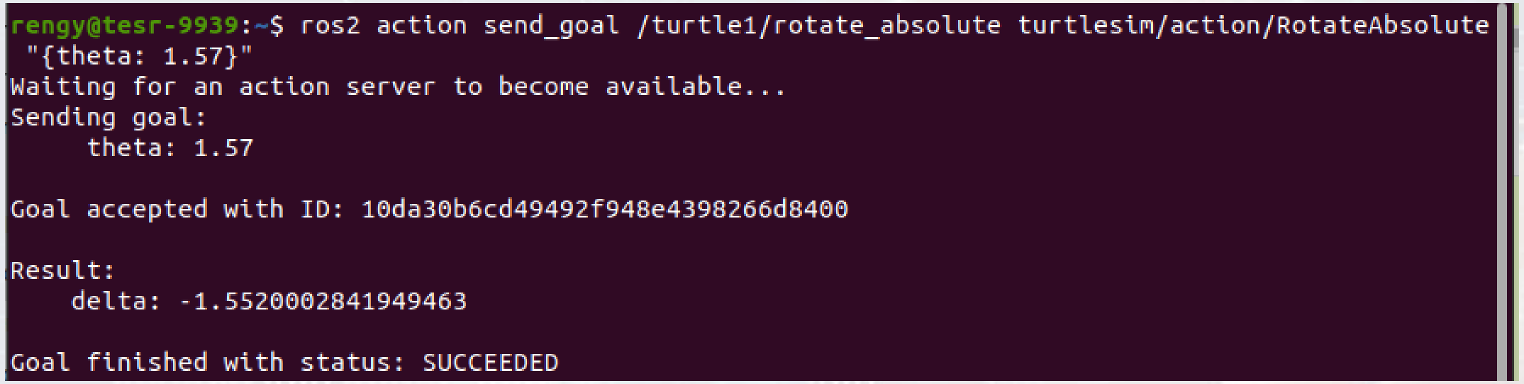
```
rengy@tesr-9939:~$ ros2 interface show turtlesim/action/RotateAbsolute
# The desired heading in radians
float32 theta
---
# The angular displacement in radians to the starting position
float32 delta
---
# The remaining rotation in radians
float32 remaining
```

Understanding actions

- **action send goal**
  - Now let’s send an action goal from the command line with the following syntax:

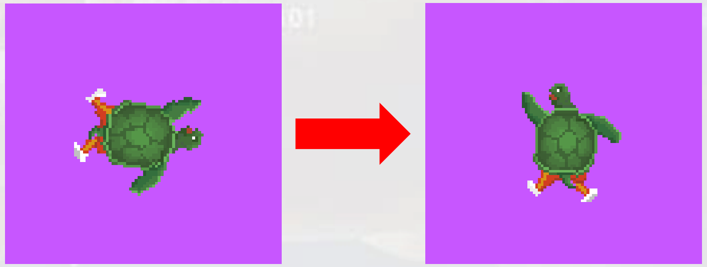
```
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
```

  - Output will be same as below:



```
rengy@tesr-9939:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute
 "{theta: 1.57}"
Waiting for an action server to become available...
Sending goal:
     theta: 1.57

Goal accepted with ID: 10da30b6cd49492f948e4398266d8400

Result:
    delta: -1.5520002841949463

Goal finished with status: SUCCEEDED
```

  - So, the turtle in turtlesim should rotated by 1.57 theta.

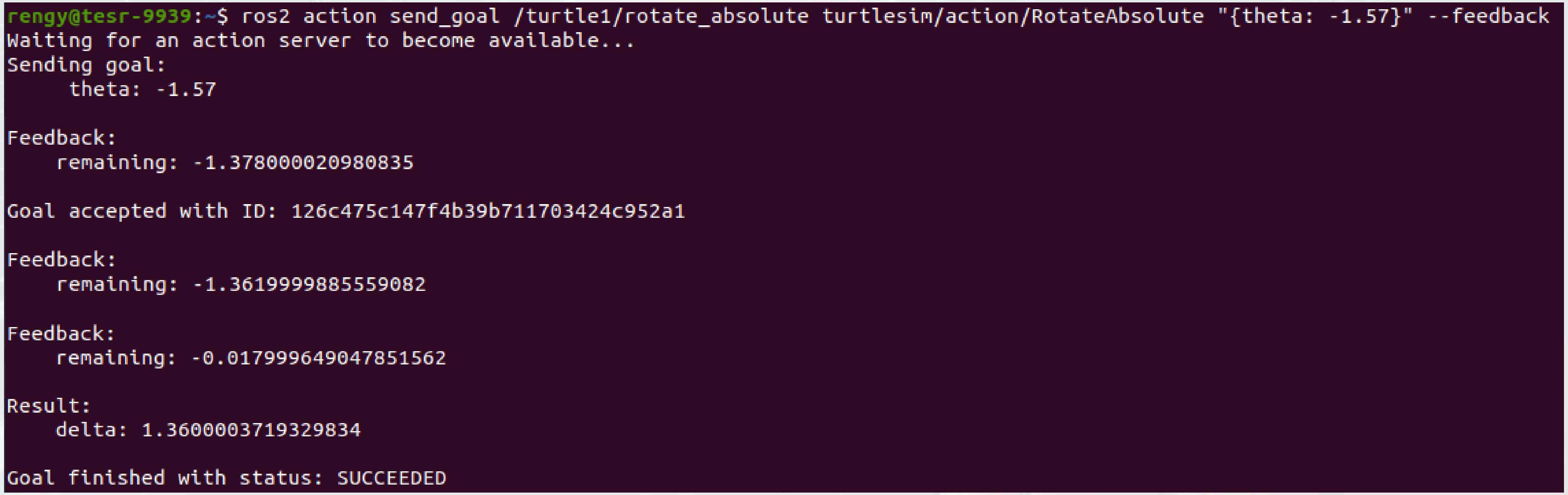


Understanding actions

- **action send goal**
  - You can also see the feedback of this goal by add "--feedback" after the command:

```
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: -1.57}" --feedback
```

  - Output on terminal will be same as below:



```
rengy@tesr-9939:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: -1.57}" --feedback
Waiting for an action server to become available...
Sending goal:
     theta: -1.57

Feedback:
    remaining: -1.378000020980835

Goal accepted with ID: 126c475c147f4b39b711703424c952a1

Feedback:
    remaining: -1.3619999885559082

Feedback:
    remaining: -0.017999649047851562

Result:
    delta: 1.3600003719329834

Goal finished with status: SUCCEEDED
```