

7.3 Commands and tools

7.3.1 Start node mode

1. launch file

There are at least two ways to start a launch file with the `roslaunch` command:

1) 、 Start with the ros package path

Format as shown below:

```
roslaunch package_name launch_file_name
roslaunch pkg_name launchfile_name.launch
```

2) 、 Give the absolute path of the launch file directly

Format as shown below

```
roslaunch path_to_launchfile
```

No matter which method is used to start the launch file, you can add parameters. The more common parameters are as follows

- `--screen`: Output the ros node information to the screen instead of saving it in a log file, which is convenient for debugging
- `arg:=value`: If there are variables to be assigned in the launch file, they can be assigned in this way, for example:

```
roslaunch pkg_name launchfile_name model:=urdf/myfile.urdf
```

or

```
roslaunch pkg_name launchfile_name model:='$(find urdf_pkg)/urdf/myfile.urdf'
```

When the `roslaunch` command runs, it will first check whether the system's `rosmaster` is running. If it is already started, use the existing `rosmaster`; if it is not started, you need to start `rosmaster` first, and then execute the settings in the launch file, one-time Start multiple nodes according to our pre-configuration.

Note: the launch file does not need to be compiled, and you can directly run it in the above way after setting it up.

2. rosrun

The node manager (master) must be started first. The master is used to manage many processes in the system. When each node is started, it must register with the master to manage the communication between the nodes. After the master is started, register each node through the master. Input the following command in the Ubuntu terminal:

```
roscore
```

Start the node, `roslaunch + package name + node name`; the `roslaunch` method can only run one node at a time.

```
roslaunch [--prefix cmd] [--debug] pkg_name node_name [ARGS]
```

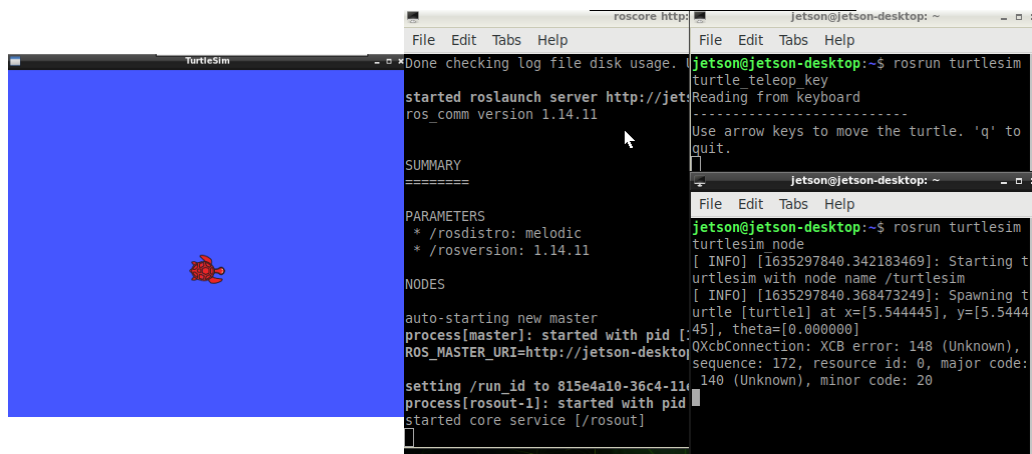
3. python

If it is a python code, you can start it directly in the directory where the py file is located. Pay attention to distinguish python2 and python3.

4. Start the turtle

```
roscore
roslaunch turtlesim turtlesim_node      # Start the turtle simulator node
roslaunch turtlesim turtlesim_teleop_key # Start the turtle keyboard control node
```

After the startup is complete, we can control the movement of the turtle by keyboard input. During keyboard control, the cursor must be under the command line `roslaunch turtlesim turtlesim_teleop_key`, and click the keyboard `↑`, `↓`, `←`, `→` to control the turtle moves.



And in the `roslaunch turtlesim turtlesim_node` terminal will print some turtles log information

```
[ INFO] [1607648666.226328691]: Starting turtlesim with node name /turtlesim
[ INFO] [1607648666.229275030]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

5. Start two turtles

Input the following command to install function package

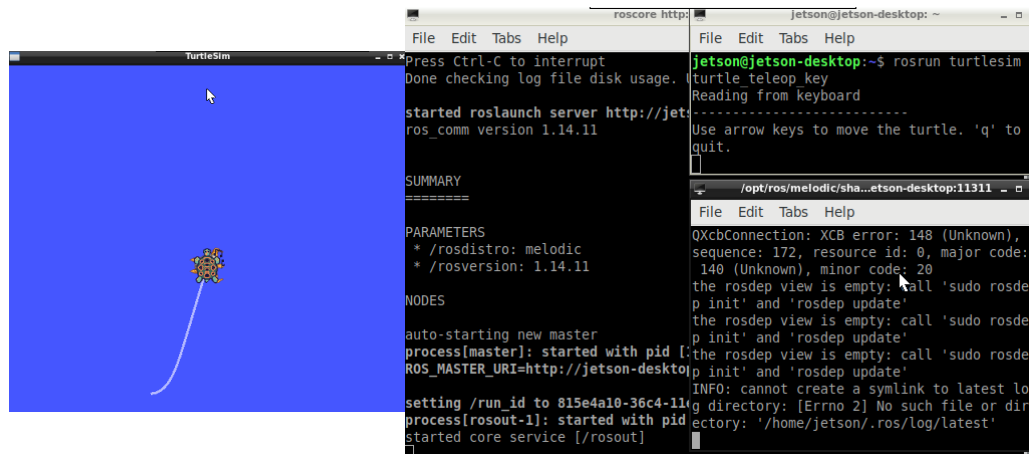
```
sudo apt install ros-melodic-turtle-tf
```

Input the following command to start up function package

```
roslaunch turtle_tf turtle_tf_demo.launch
```

Keyboard control node

```
roslaunch turtlesim turtlesim
```



At this time, press the keyboard [Up], [Down], [Left], [Right] to drive the little turtle to move; you can observe one little turtle following the another.

7.3.2 launch file

1. Overview

In ROS, a node program generally can only complete a single-function task, but a complete ROS robot generally has many node programs running at the same time and cooperating to complete complex tasks. Therefore, this requires that the robot must be started when the robot is started. There are many node programs, if one node is started one node, it is more troublesome. Through the launch file and the roslaunch command, multiple nodes can be started at one time, which is convenient for "one-click startup", and rich parameters can be set.

2. File format

The launch file is an xml file

```
<?xml version="1.0"?>
```

Similar to other xml format files, launch files are also written by **tags**, the main **tag** are as follows:

<launch>	<!--Root label-->
<node>	<!--The node and its parameters that need to be started-->
<include>	<!--Include other launch-->
<machine>	<!--Specify the machine to run-->
<env-loader>	<!--Set environment variables-->
<param>	<!--Define the parameter to the parameter server-->
<rosparam>	<!--Load the parameters in the yaml file to the parameter server-->
<arg>	<!--Define variable-->
<remap>	<!--Set topic mapping-->
<group>	<!--Set grouping-->
</launch>	<!--Root tag-->

7.3.3 TF coordinate transformation

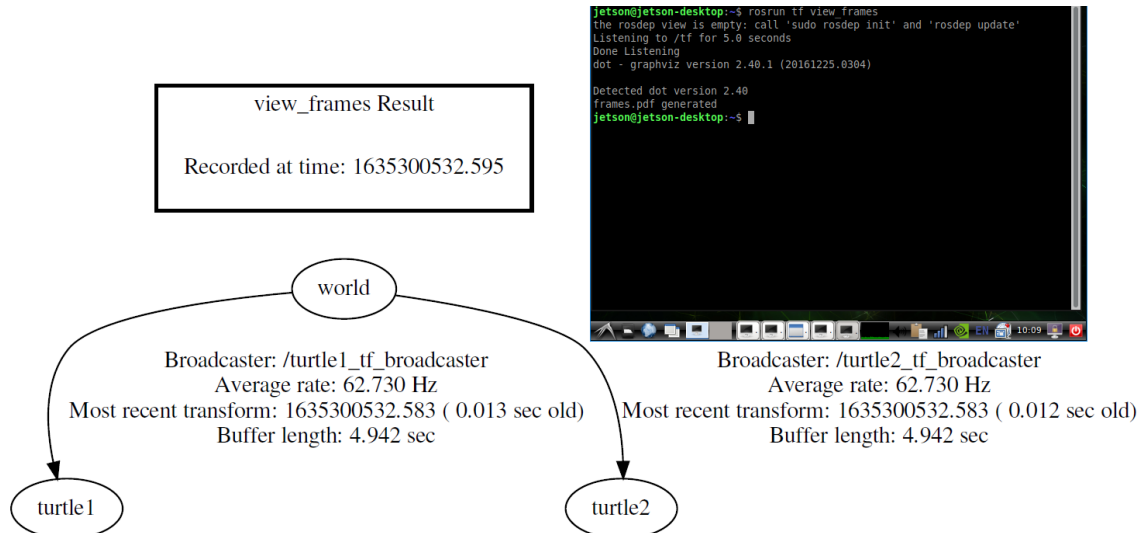
tf is a function package that allows users to track multiple coordinate systems at any time.

1. tf common tools

① view_frames tools

It can monitor all tf coordinate systems broadcast through ROS at the current moment, and draw a tree diagram to show the connection relationship between the coordinate systems, generate a file named frame.pdf, and save it to the local current location.

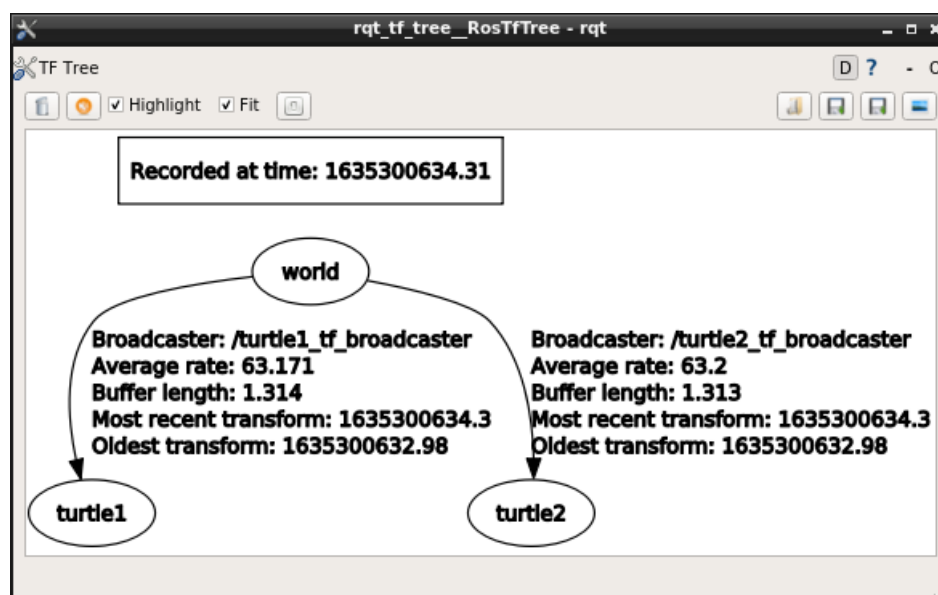
```
roslaunch tf view_frames
```



② rqt_tf_tree tool

Although view_frames can save the current coordinate system relationship in an offline file, it cannot reflect the coordinate relationship in real time, so you can use rqt_tf_tree to refresh the display coordinate system relationship in real time.

```
roslaunch rqt_tf_tree rqt_tf_tree
```



③ tf_echo tool

Use the tf_echo tool to view the relationship between the two broadcast reference systems.

```
roslaunch tf tf_echo <source_frame> <target_frame>
```

Print the rotation and translation transformation from source_frame to target_frame; for example:

```
roslaunch tf tf_echo turtle1 turtle2
```

```
^Cjetson@jetson-desktop:~$ roslaunch tf tf_echo turtle1 turtle2
At time 1635300703.687
- Translation: [0.000, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.707, 0.707]
             in RPY (radian) [0.000, -0.000, 1.571]
             in RPY (degree) [0.000, -0.000, 90.000]
At time 1635300704.423
- Translation: [0.000, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.707, 0.707]
             in RPY (radian) [0.000, -0.000, 1.571]
             in RPY (degree) [0.000, -0.000, 90.000]
At time 1635300705.431
- Translation: [0.000, 0.000, 0.000]
```

④ static_transform_publisher

Publish the static coordinate transformation between the two coordinate systems, and the relative position of the two coordinate systems will not change.

Command format:

```
static_transform_publisher x y z yaw pitch roll frame_id child_frame_id
period_in_ms
static_transform_publisher x y z qx qy qz qw frame_id child_frame_id
period_in_ms
```

Use in launch:

```
<launch>
  <node pkg="tf" type="static_transform_publisher" name="link1_broadcaster"
  args="1 0 0 0 0 0 1 link1_parent link1 100" />
</launch>
```

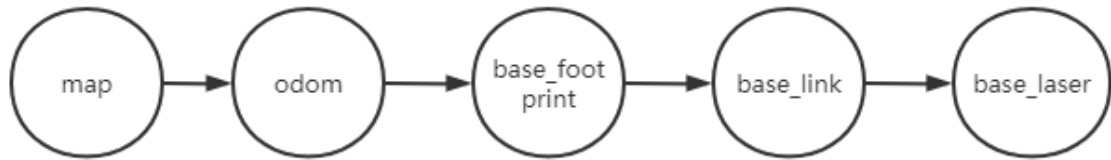
⑤ roswtf plugin

It can analyze your current tf configuration and tries to find common problems.

```
roswtf
```

2. Common coordinate system

The commonly used coordinate system is frame_id, including map, odom, base_link, base_footprint, base_laser, etc.



7.3.4 rqt(QT tool)

Open the command line window and enter `roslaunch rqt` and then double-click the `Tab` key to view the contents of the QT tool in ROS, as shown in the following figure.

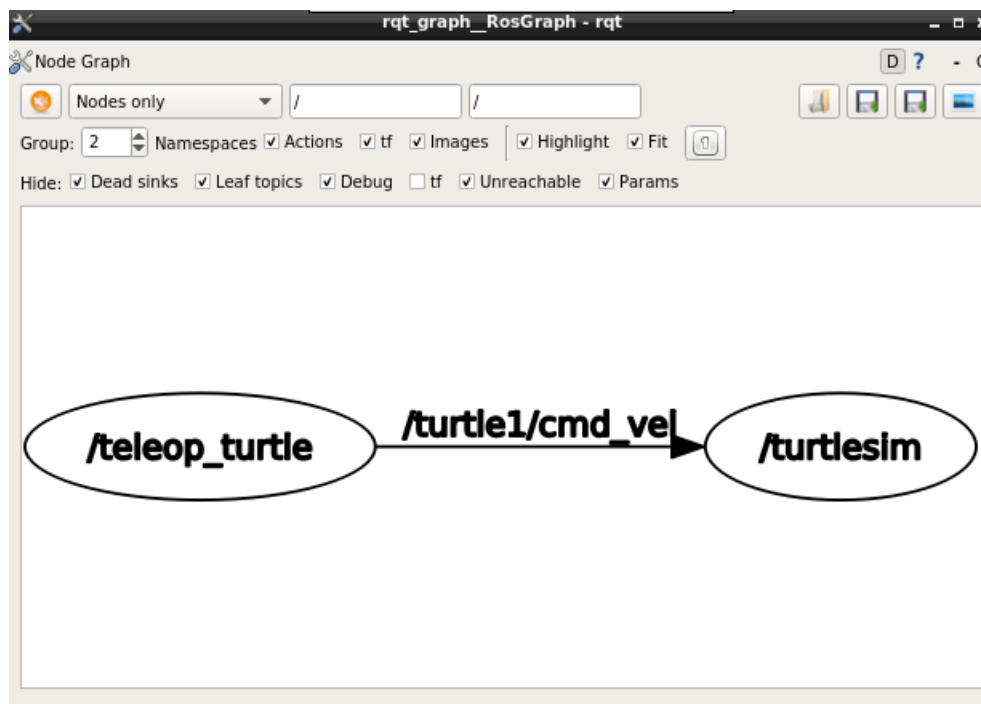
```

jetson@jetson-desktop:~$ roslaunch rqt
rqt_action          rqt_logger_level   rqt_robot_monitor
rqt_bag             rqt_moveit         rqt_robot_steering
rqt_bag_plugins     rqt_msg            rqt_runtime_monitor
rqt_console         rqt_nav_view       rqt_rviz
rqt_dep            rqt_plot           rqt_service_caller
rqt_graph           rqt_pose_view      rqt_shell
rqt_gui            rqt_publisher      rqt_srv
rqt_gui_cpp        rqt_py_common      rqt_tf_tree
rqt_gui_py         rqt_py_console     rqt_top
rqt_image_view     rqt_reconfigure    rqt_topic
rqt_launch         rqt_robot_dashboard rqt_web
  
```

1. rqt_graph

Open the command line window and enter the following command, a dialog window will pop up.

```
roslaunch rqt_graph rqt_graph
```



From the image, we can clearly see that the `/teleop_turtle` node transmits data to the `/turtlesim` node through the `/turtle1/cmd_vel` topic.

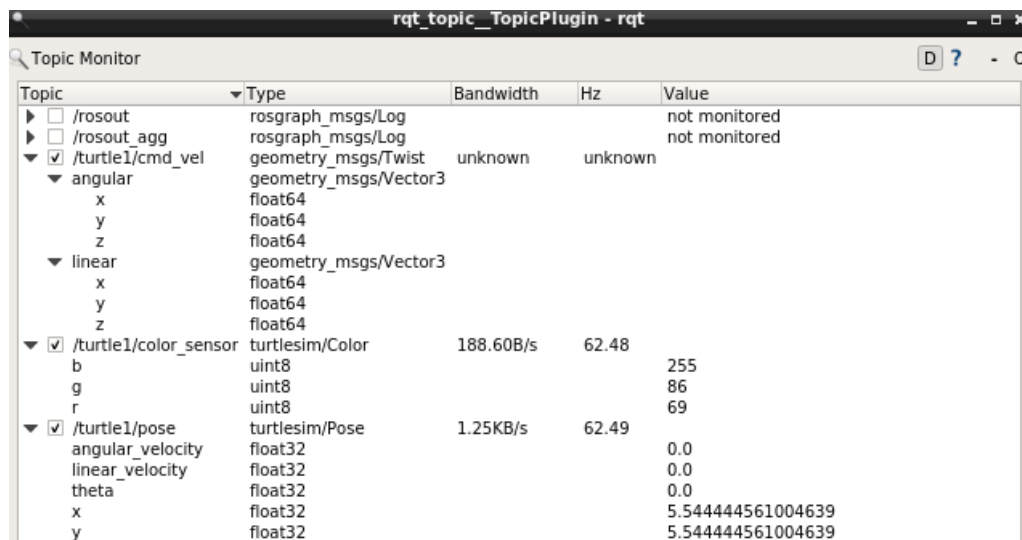
`/teleop_turtle` is a node with Publisher function.

`/turtlesim` is a node with Subscriber function.

`/turtle1/cmd_vel` is the topic of communication between publisher and subscriber.

2. rqt_topic

```
roslaunch rqt_topic rqt_topic
```



Topic	Type	Bandwidth	Hz	Value
<input type="checkbox"/> /rosout	rosgraph_msgs/Log			not monitored
<input type="checkbox"/> /rosout_agg	rosgraph_msgs/Log			not monitored
<input checked="" type="checkbox"/> /turtle1/cmd_vel	geometry_msgs/Twist	unknown	unknown	
<input checked="" type="checkbox"/> angular	geometry_msgs/Vector3			
x	float64			
y	float64			
z	float64			
<input checked="" type="checkbox"/> linear	geometry_msgs/Vector3			
x	float64			
y	float64			
z	float64			
<input checked="" type="checkbox"/> /turtle1/color_sensor	turtlesim/Color	188.60B/s	62.48	
b	uint8			255
g	uint8			86
r	uint8			69
<input checked="" type="checkbox"/> /turtle1/pose	turtlesim/Pose	1.25KB/s	62.49	
angular_velocity	float32			0.0
linear_velocity	float32			0.0
theta	float32			0.0
x	float32			5.544444561004639
y	float32			5.544444561004639

Through this tool, we can clearly see some real-time changing information of the little turtles.

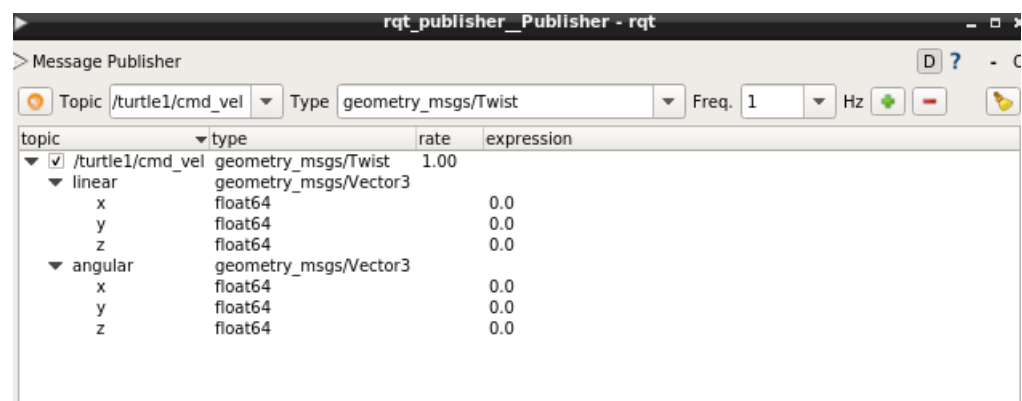
3. rqt_publisher

rqt_publisher provides a GUI plug-in for publishing arbitrary messages with fixed or calculated field values.

Open the command line window and enter the following command, a dialog window will pop up.

```
roslaunch rqt_publisher rqt_publisher
```

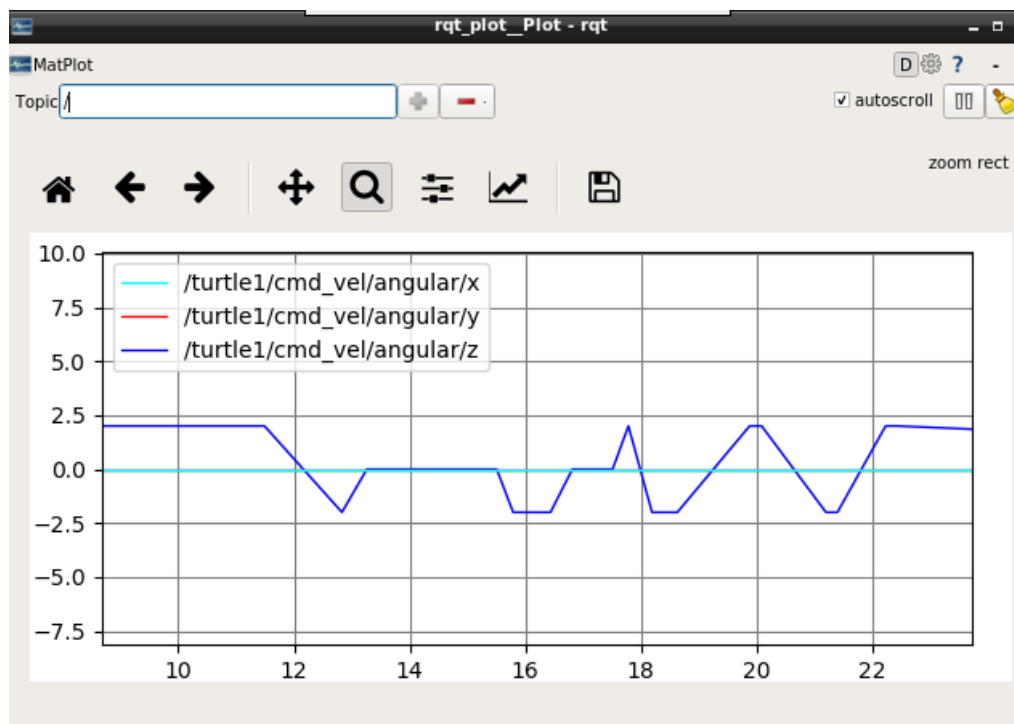
Click the selection box on the right of **Topic** to find the topic of `/turtle1/cmd_vel` we need, and click the plus sign on the right to add it, as shown below:



topic	type	rate	expression
<input checked="" type="checkbox"/> /turtle1/cmd_vel	geometry_msgs/Twist	1.00	
<input checked="" type="checkbox"/> linear	geometry_msgs/Vector3		
x	float64	0.0	
y	float64	0.0	
z	float64	0.0	
<input checked="" type="checkbox"/> angular	geometry_msgs/Vector3		
x	float64	0.0	
y	float64	0.0	
z	float64	0.0	

4. rqt_plot

```
roslaunch rqt_plot rqt_plot
```



6. rqt_console

The function of the ROS log (log) system is to allow the program to generate some log messages, which are displayed on the screen, sent to a specific topic or stored in a specific log file to facilitate debugging, recording, alarming, etc.

Serial number	Grade	Parsing
1	DEBUG	Debug log for development and testing
2	INFO	Regular log, user-visible level of information
3	WARN	Warning message.
4	ERROR	Error message. Information printed after program error
5	FATAL	Fatal error. Log records of downtime

Log messages in ROS can be divided into 5 levels according to their severity: DEBUG, INFO, WARN, ERROR, FATAL.

As long as the program can run, you can ignore these errors, but the presence of ERROR and FATAL indicates that the program has serious problems that prevent it from running.

```
roslaunch rqt_console rqt_console
```

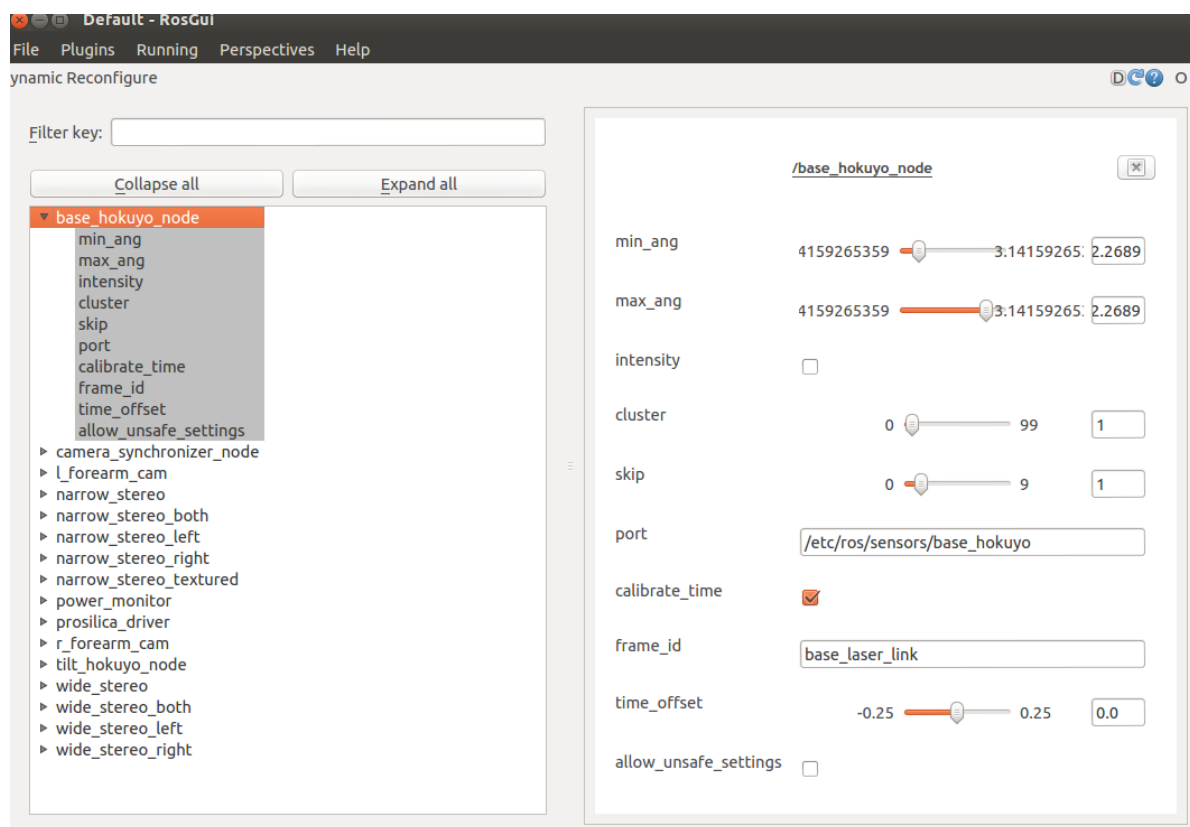
- API

C++ basic API format	C++stream API format	Python Log API
ROS_DEBUG("What needs to be printed");	ROS_DEBUG_STREAM("What needs to be printed" << "hello");	rospy.logdebug("What needs to be printed")
ROS_INFO("What needs to be printed");	ROS_INFO_STREAM("What needs to be printed" << "hello");	rospy.loginfo("What needs to be printed")
ROS_WARN("What needs to be printed");	ROS_WARN_STREAM("What needs to be printed" << "hello");	rospy.logwarn("What needs to be printed")
ROS_ERROR("What needs to be printed");	ROS_ERROR_STREAM("What needs to be printed" << "hello");	rospy.logerror("What needs to be printed")
ROS_FATAL("What needs to be printed");	ROS_FATAL_STREAM("What needs to be printed" << "hello");	rospy.logfatal("What needs to be printed")

7. rqt_reconfigure dynamic parameter configuration

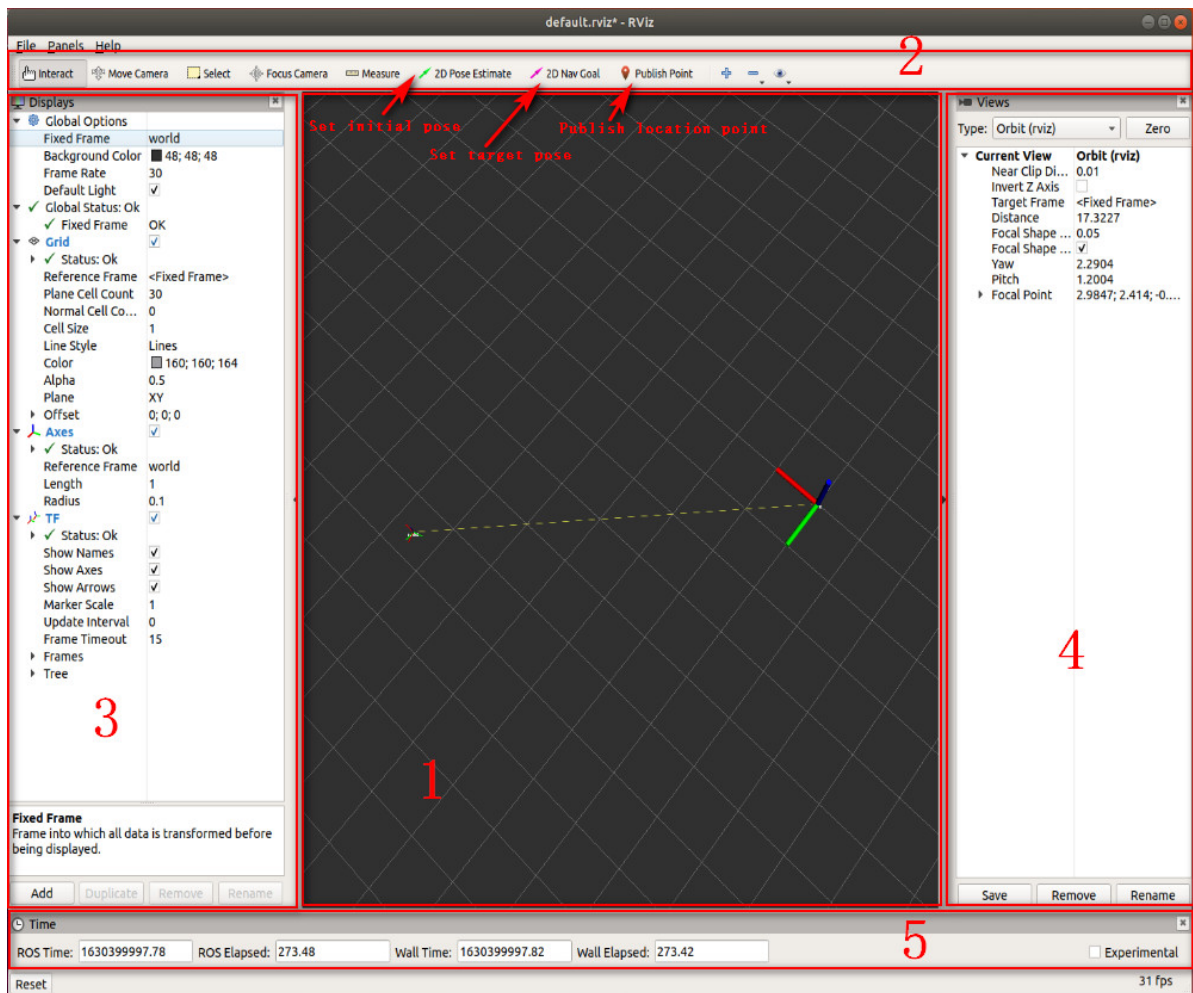
```
roslaunch rqt_reconfigure rqt_reconfigure
```

Picture from ROS wiki:



7.3.5 Rviz

rviz is a graphical tool that comes with ros, which can conveniently perform graphical operations on ros programs. Its use is relatively simple.

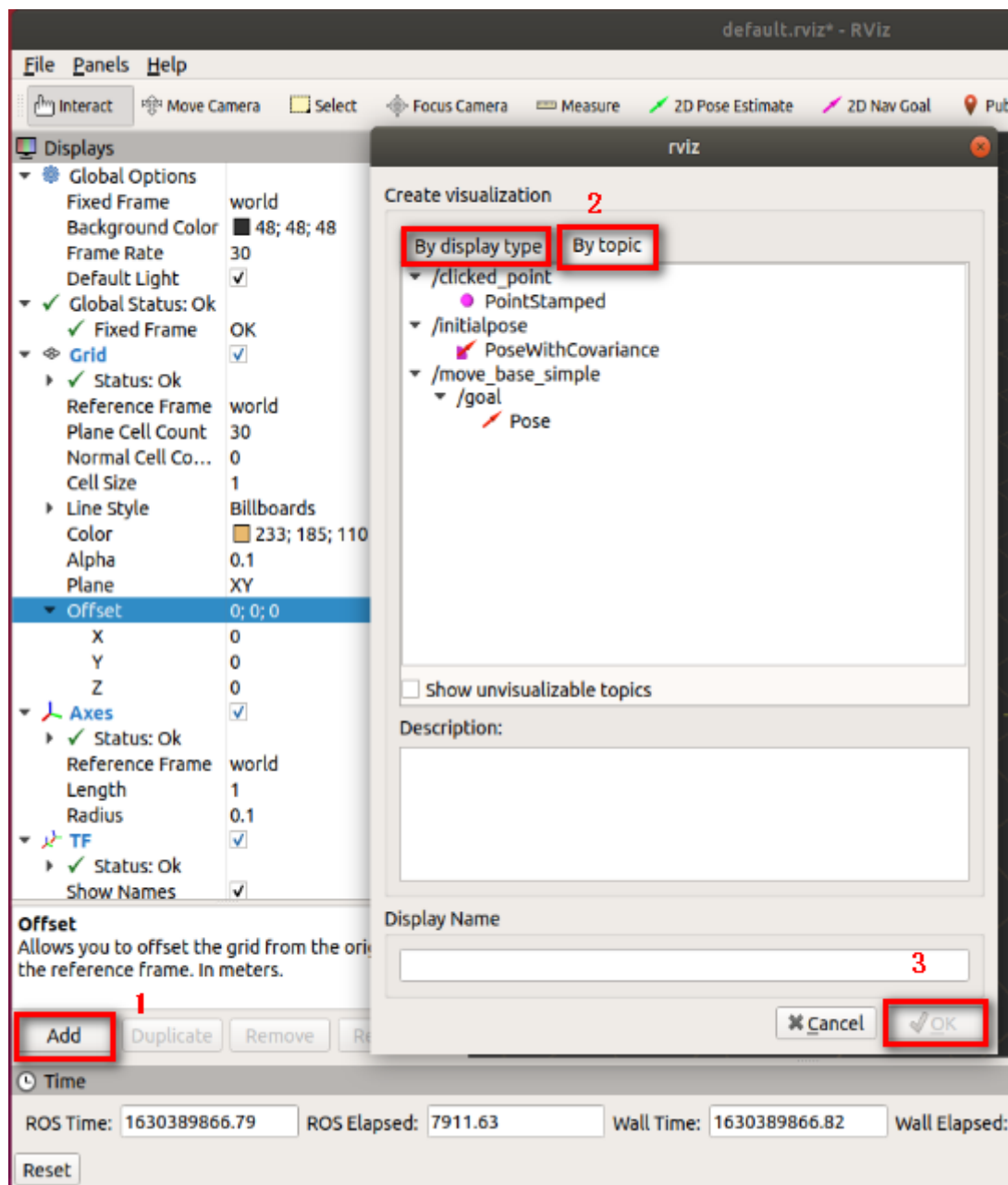


【Set initial pose】、【Set target pose】、【Publish location point】： It is generally used when building a map and navigating.

The rviz interface mainly includes the following parts:

- 1: 3D view area, used to display data visually, there is no data at present, so it is displayed in black.
- 2: Toolbar, providing tools for viewing angle control, target setting, publishing location, etc.
- 3: Display item list, used to display the currently selected display plug-in, you can configure the properties of each plug-in.
- 4: Viewing angle setting area, you can choose a variety of viewing angles.
- 5: Time display area, showing the current system time and ROS time.

- Add display



Step 1: Click the 【Add】 button. A selection box will pop up.

Step 2: You can choose to add through the display type 【By display type】 , but you need to modify the corresponding topic yourself before the coordinate system can be displayed; you can also select the topic 【By topic】 to directly add it and it can be displayed normally.

Step 3: Click 【OK】 .

7.3.6 ROS commonly used commands

command	Function
catin_create_pkg	Information for creating a feature package
rospack	Get information about the feature pack
catkin_make	Compile the function package in the workspace
rosdep	Automatically install other packages that the feature package depends on
roscd	Function package directory jump
roscp	Copy the files in the function package
rosed	Edit the files in the feature pack
roslaunch	Run the executable file in the feature pack
roslaunch	Run the startup file