3、Commands and tools

3.1、Start node mode

3.1.1、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.

3.1.2、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 node through the master. Input the following command in the Ubuntu terminal:

```
roscore
```

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

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

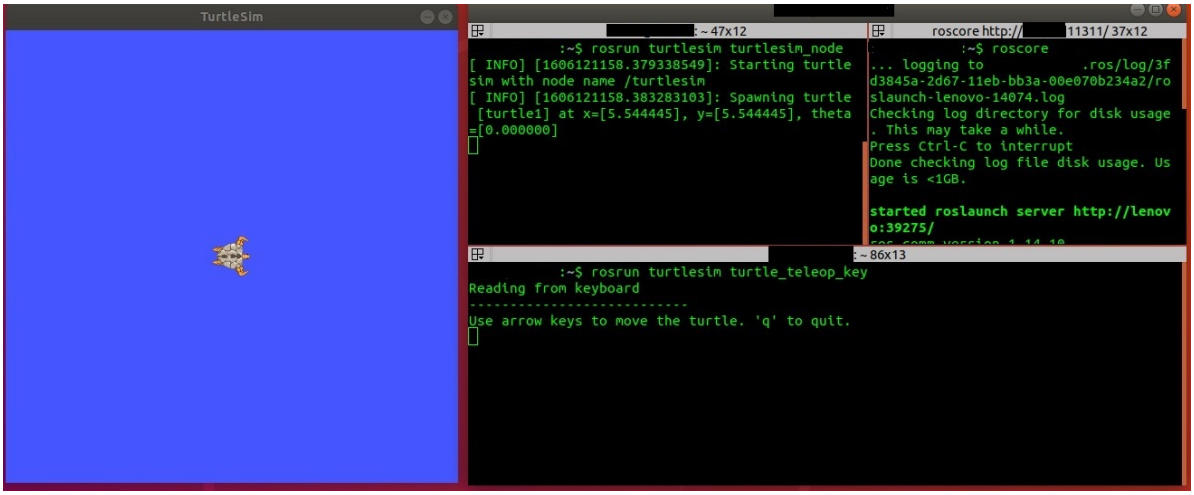
3.1.3、python

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.

3.1.4、Start the turtle

```
roscore
rosrun turtlesim turtlesim_node     # Start the turtle simulator node
rosrun turtlesim turtle_teleop_key  # Start the turtle kayboard 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 【rosrun turtlesim turtle_teleop_key】, and click the keyboard 【up】, 【down】, 【left】, 【right】to control the turtle moves.



And in the rosrun 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]
```

3.1.5、Start two turtles

Input the following command to install function package

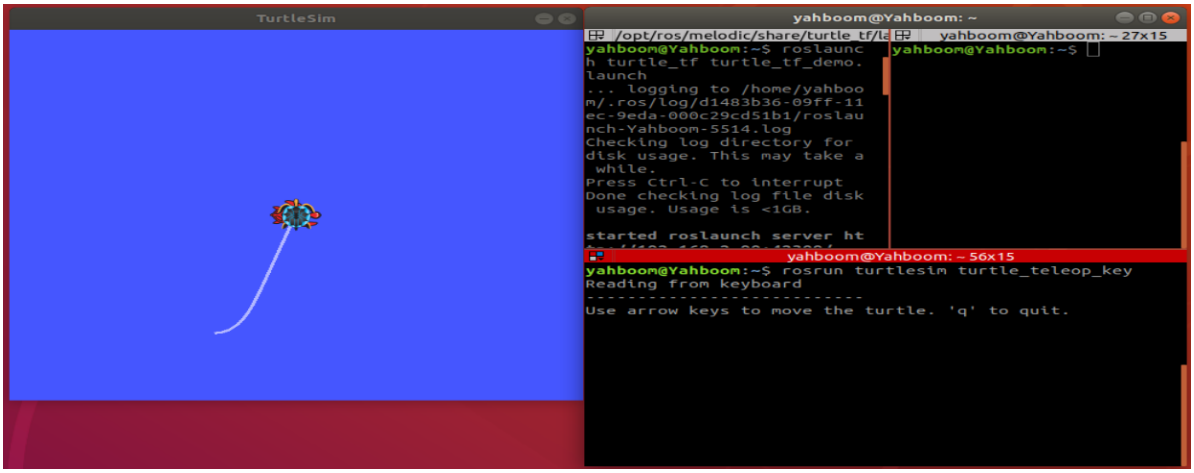
```
sudo apt install ros-melodic-turtle-tf
```

Input the following command to start up function package

```
roslaunch turtle_tf turtle_tf_demo.launch
```

Keyboard control node

```
rosrun turtlesim turtle_teleop_key
```



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.

3.2、launch file

3.2.1、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-->
```

3.3、TF coordinate transformation

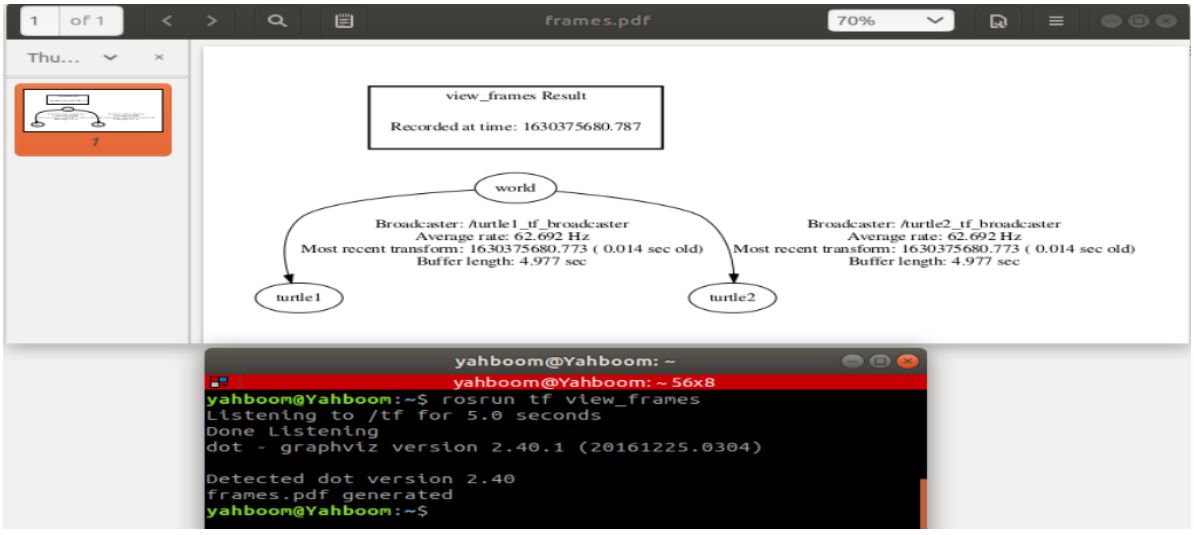
tf is a function package that allows users to track multiple coordinate systems at any time.

3.3.1、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.

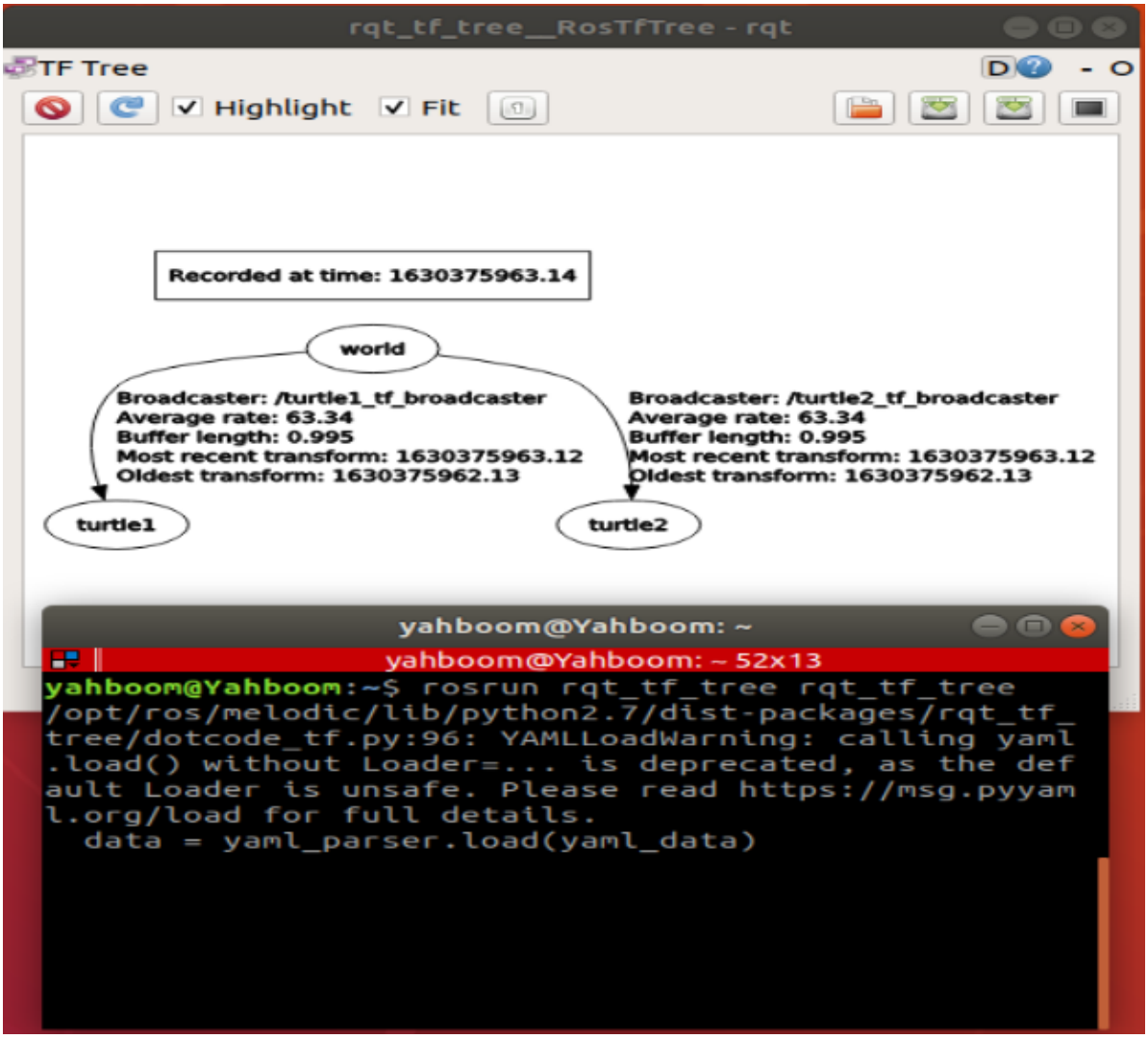
```
rosrun tf view_frames
```



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.

```
rosrun rqt_tf_tree rqt_tf_tree
```



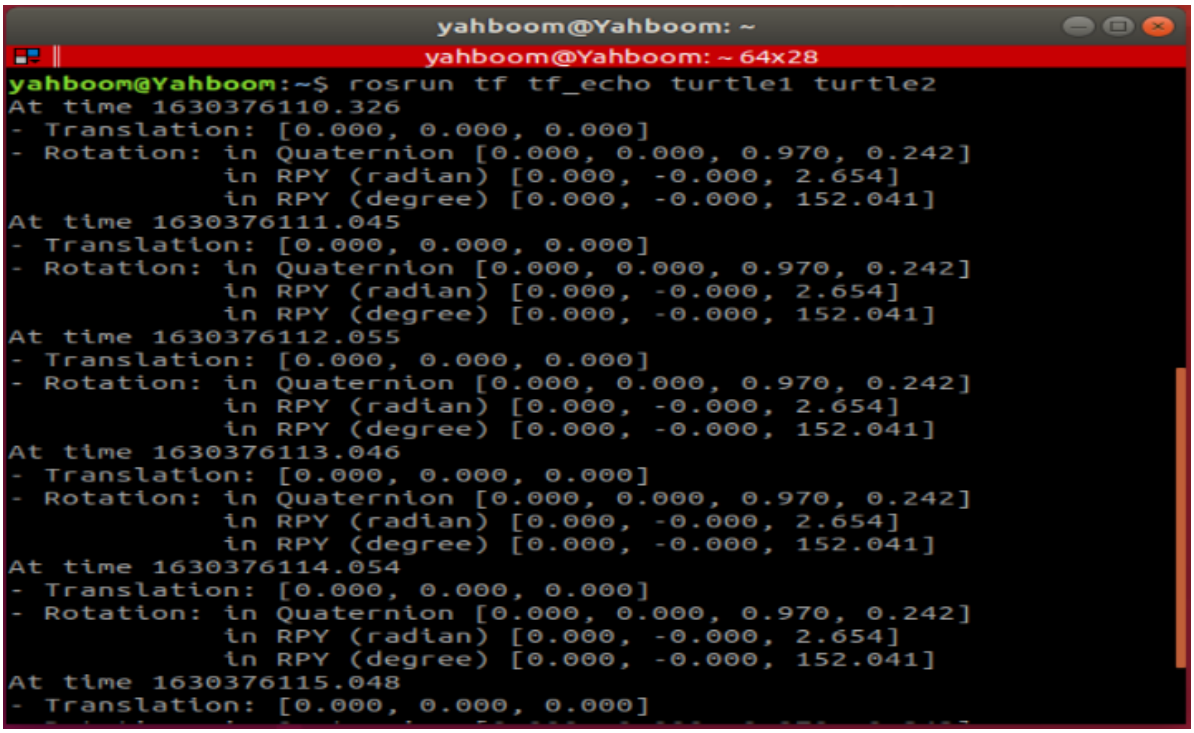
3.tf_echo tool

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

```
rosrun tf tf_echo <source_frame> <target_frame>
```

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

```
rosrun tf tf_echo turtle1 turtle2
```



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

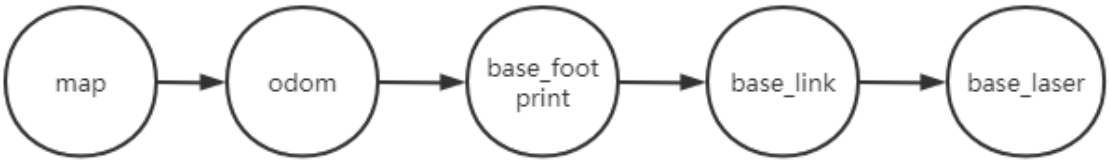
It can analyzes your current tf configuration and tries to find common problems.

```
roswtf
```

```
yahboom@Yahboom:~$ roswtf
Loaded plugin tf.tfwtf
No package or stack in the current directory
================================================================================
Static checks summary:

No errors or warnings
================================================================================
Beginning tests of your ROS graph. These may take a while...
analyzing graph...
... done analyzing graph
running graph rules...
... done running graph rules
running tf checks, this will take a second...
... tf checks complete

Online checks summary:

Found 1 warning(s).
Warnings are things that may be just fine, but are sometimes at fault

WARNING The following node subscriptions are unconnected:
 * /turtle_pointer:
   * /tf_static
```

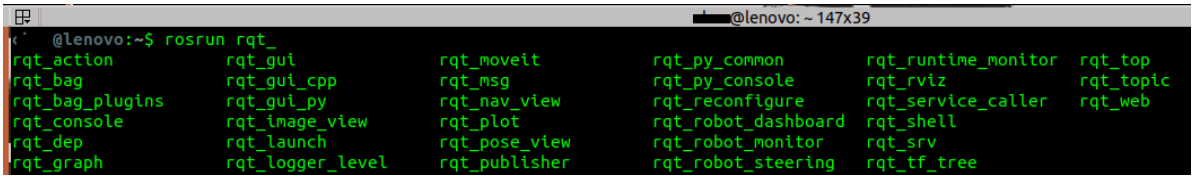
3.3.2、Common coordinate system

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



3.4、rqt(QT tool)

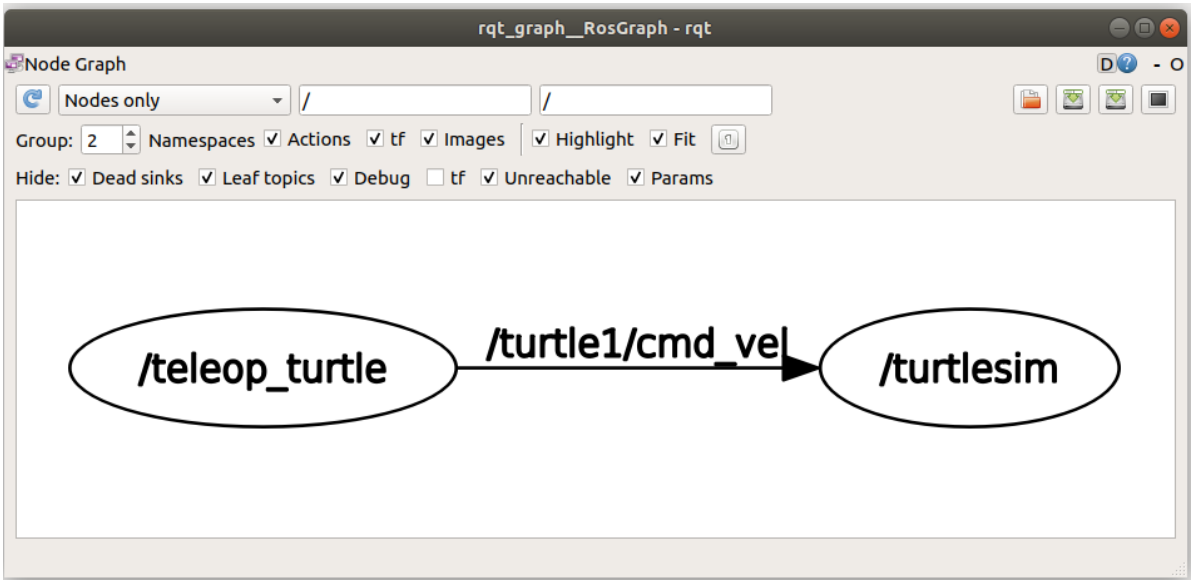
Open the command line window and enter `rosrun rqt` and then double-click the `Tab` key to view the contents of the QT tool in ROS, as shown in the following figure.



1.rqt_graph

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

```
rosrun 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.

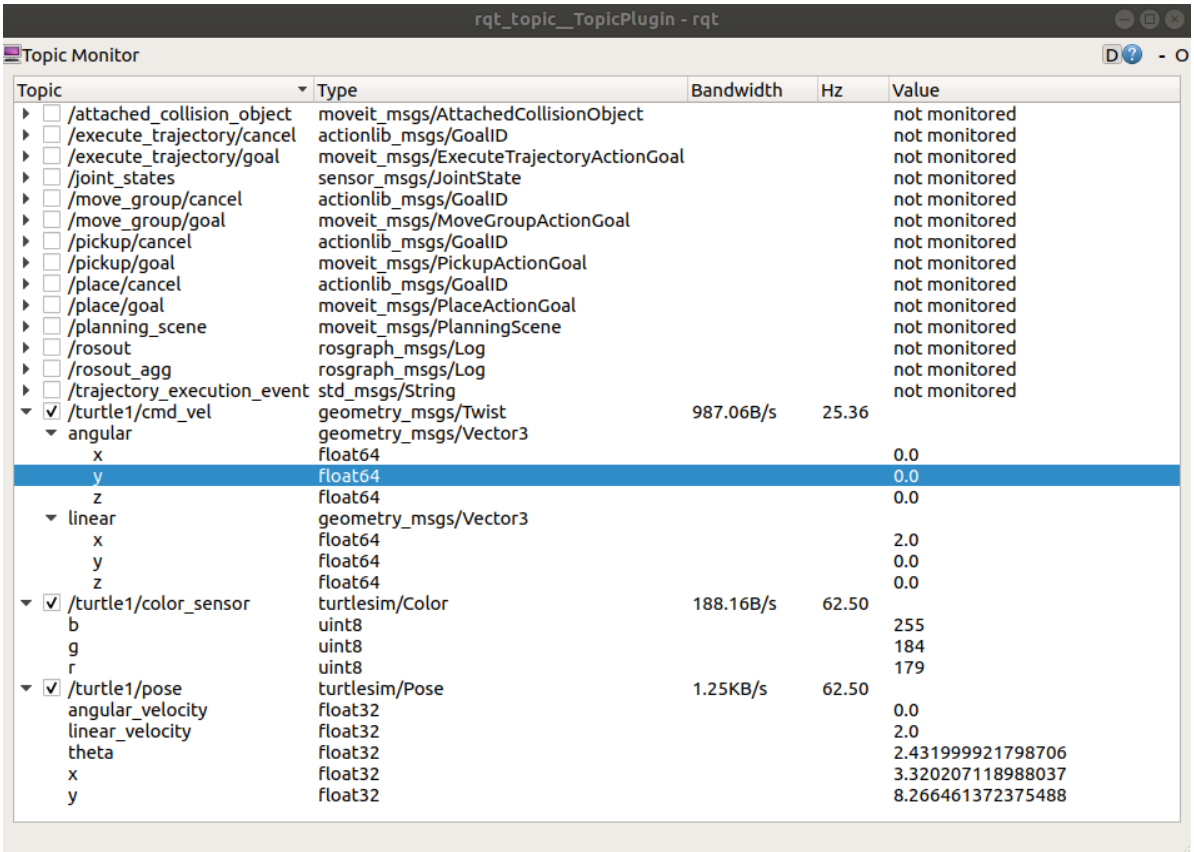
`/teleop_turtle` is a node with Publisher function.

`/turtlesim` is a node with Subscriber function.

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

2.rqt_topic

```
rosrun rqt_topic rqt_topic
```



Topic	Type	Bandwidth	Hz	Value
/attached_collision_object	moveit_msgs/AttachedCollisionObject			not monitored
/execute_trajectory/cancel	actionlib_msgs/GoalID			not monitored
/execute_trajectory/goal	moveit_msgs/ExecuteTrajectoryActionGoal			not monitored
/joint_states	sensor_msgs/JointState			not monitored
/move_group/cancel	actionlib_msgs/GoalID			not monitored
/move_group/goal	moveit_msgs/MoveGroupActionGoal			not monitored
/pickup/cancel	actionlib_msgs/GoalID			not monitored
/pickup/goal	moveit_msgs/PickupActionGoal			not monitored
/place/cancel	actionlib_msgs/GoalID			not monitored
/place/goal	moveit_msgs/PlaceActionGoal			not monitored
/planning_scene	moveit_msgs/PlanningScene			not monitored
/rosout	rosgraph_msgs/Log			not monitored
/rosout_agg	rosgraph_msgs/Log			not monitored
/trajectory_execution_event	std_msgs/String			not monitored
/turtle1/cmd_vel	geometry_msgs/Twist	987.06B/s	25.36	
angular	geometry_msgs/Vector3			
x	float64			0.0
y	float64			0.0
z	float64			0.0
linear	geometry_msgs/Vector3			
x	float64			2.0
y	float64			0.0
z	float64			0.0
/turtle1/color_sensor	turtlesim/Color	188.16B/s	62.50	
b	uint8			255
g	uint8			184
r	uint8			179
/turtle1/pose	turtlesim/Pose	1.25KB/s	62.50	
angular_velocity	float32			0.0
linear_velocity	float32			2.0
theta	float32			2.431999921798706
x	float32			3.320207118988037
y	float32			8.266461372375488

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

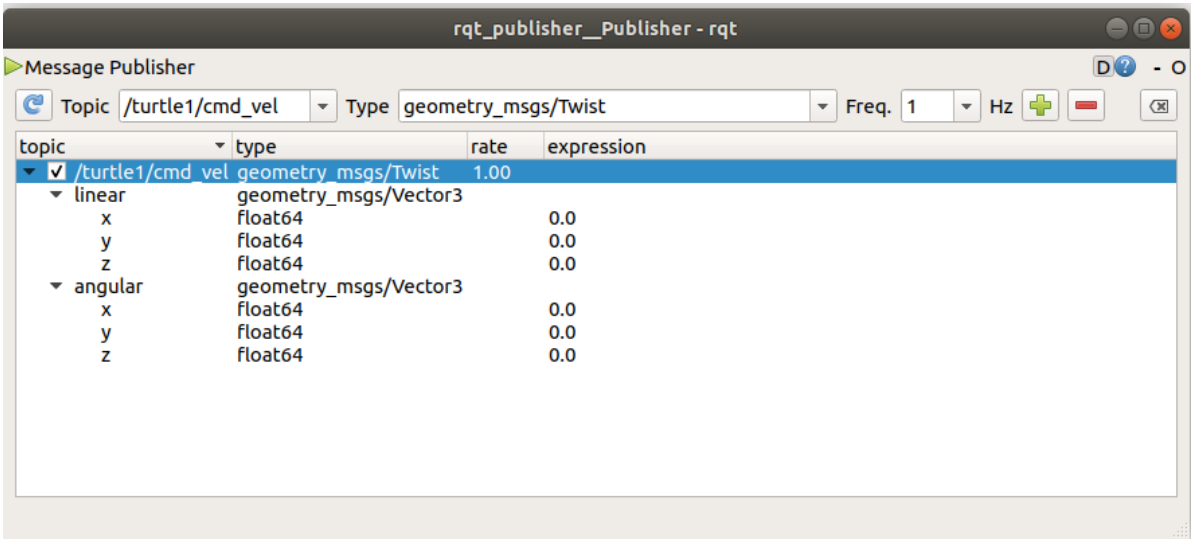
3.rqt_publisher

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.

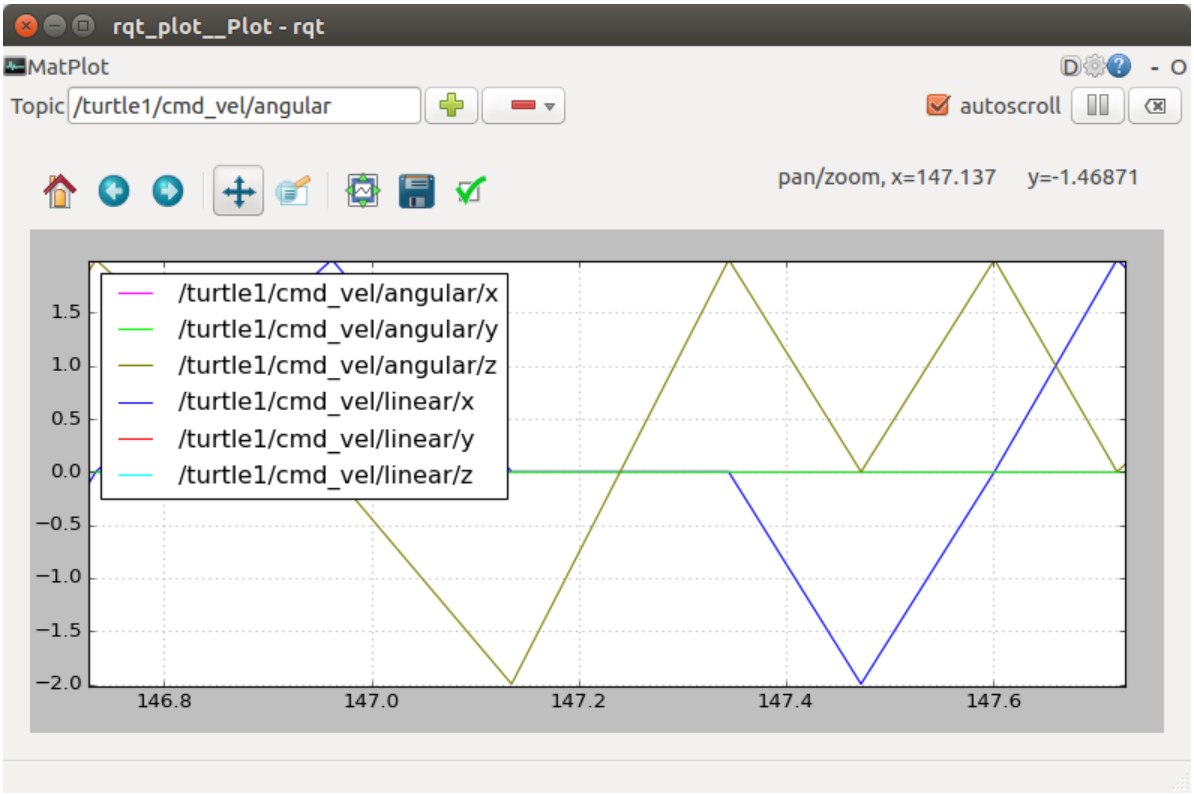
```
rosrun rqt_publisher rqt_publisher
```

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:



4.rqt_plot

```
rosrun rqt_plot rqt_plot
```



6.rqt_console

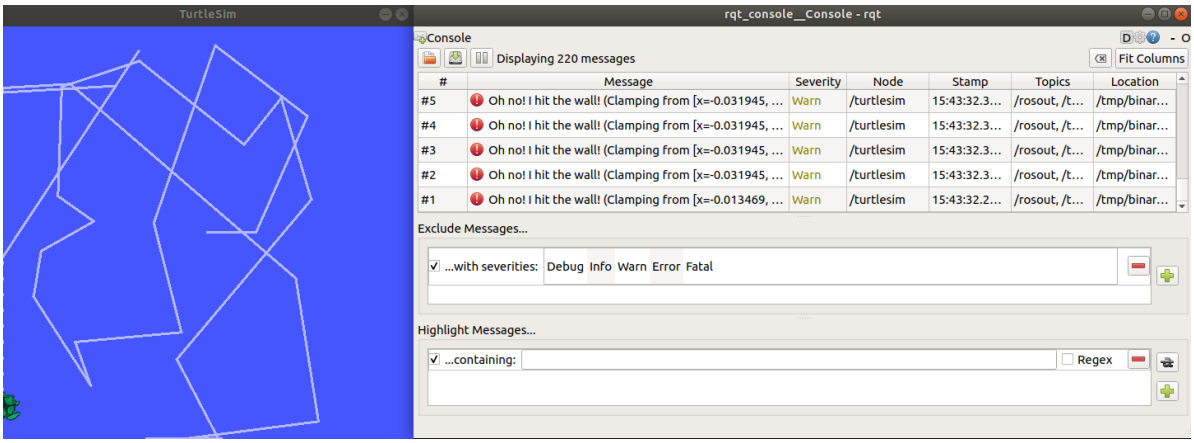
The function of the ROS log (log) system is to allow the program to generate some log messages, which are displayed on the screen, sent to a specific topic or stored in a specific log file to facilitate debugging, recording, alarming, etc.

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.

```
rosrun rqt_console rqt_console
```



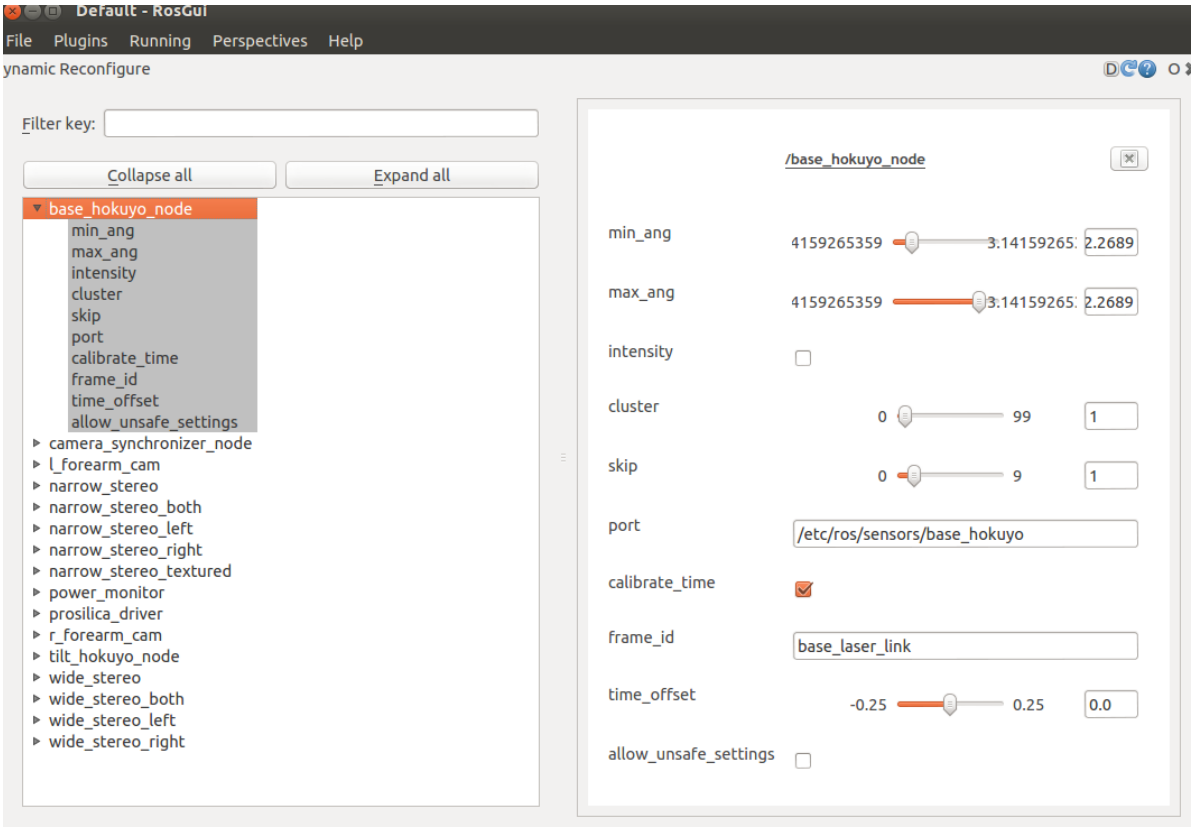
- API

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

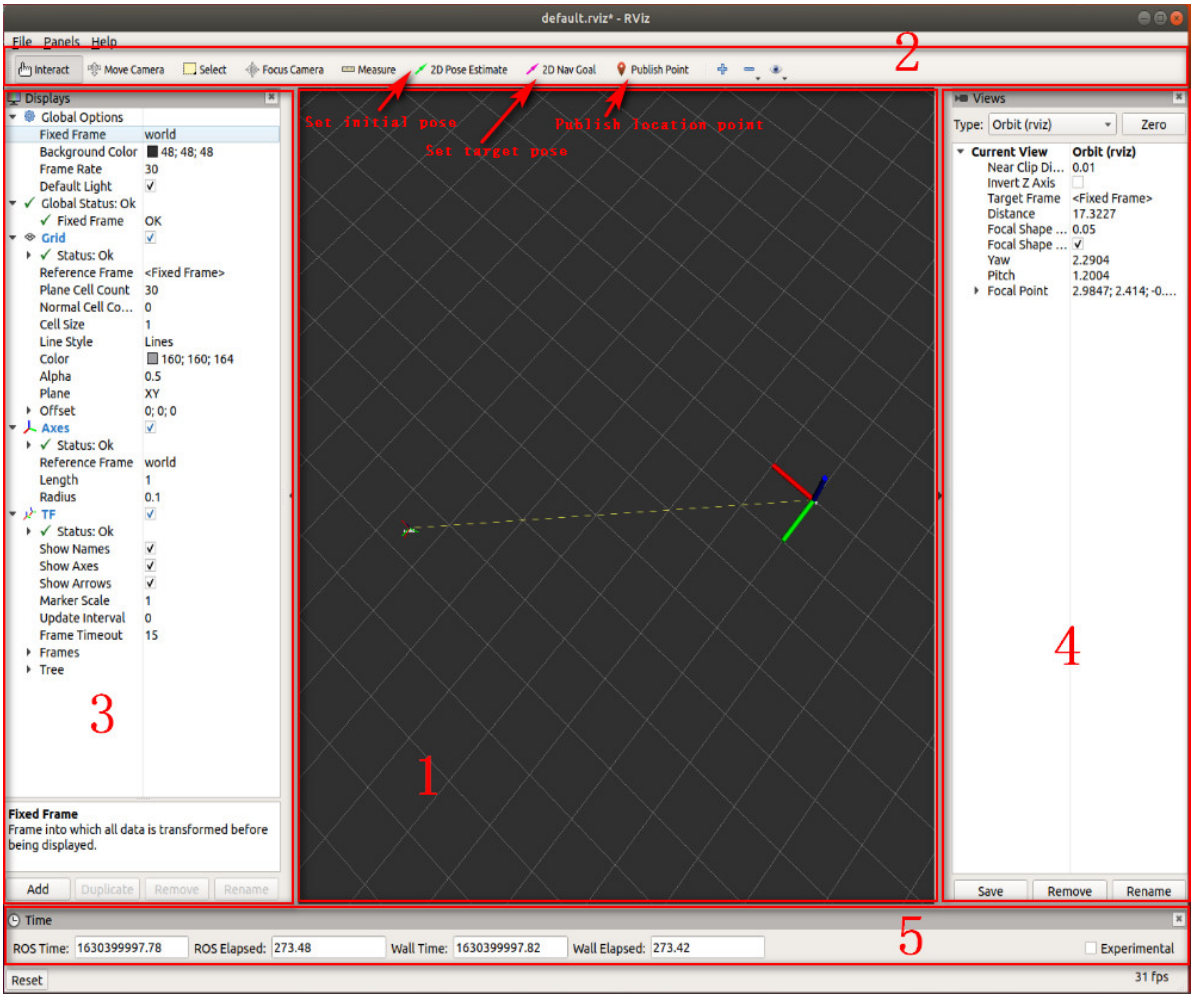
```
rosrun rqt_reconfigure rqt_reconfigure
```

Picture from ROS wiki:



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.

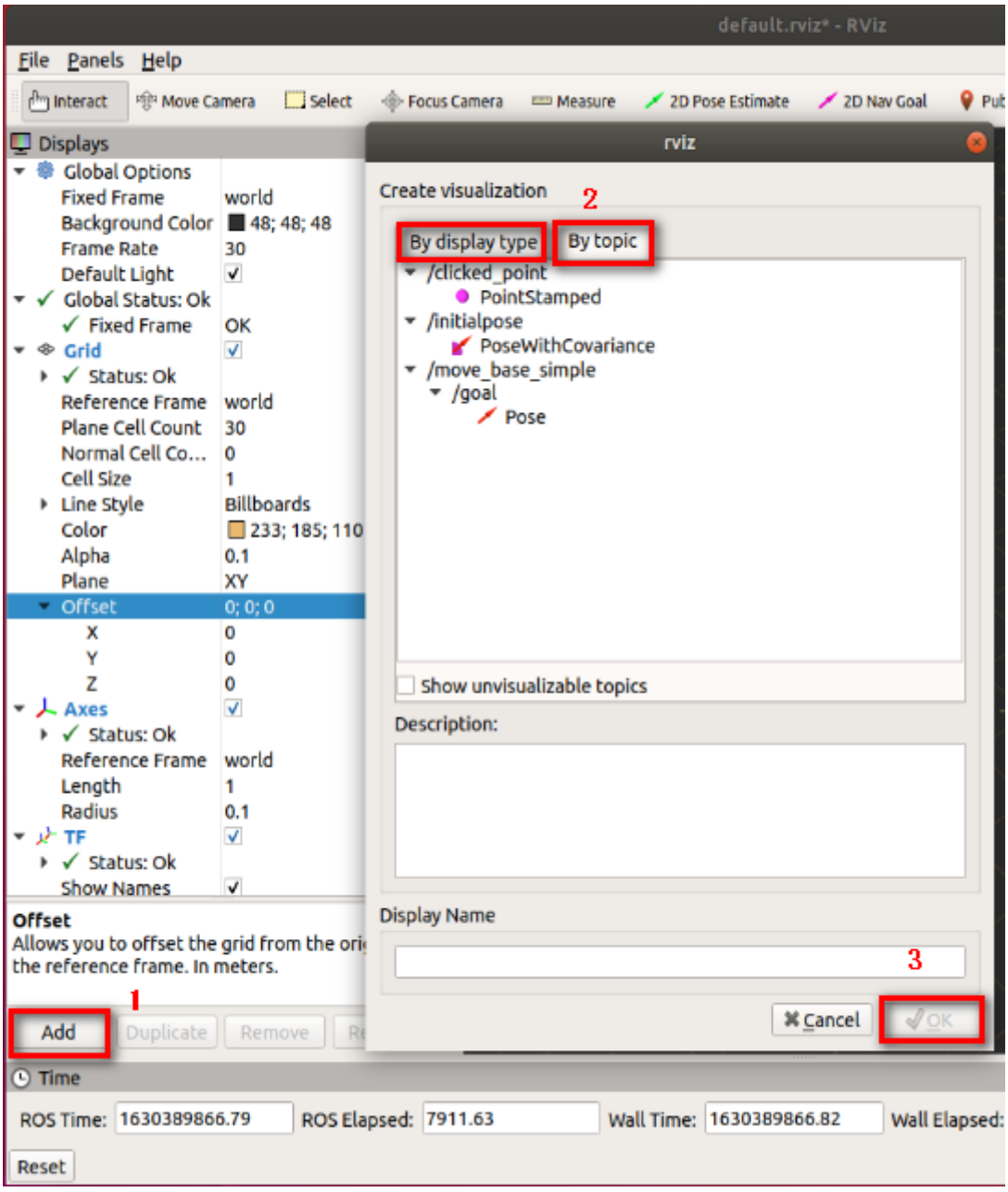
2: Toolbar, providing tools for viewing angle control, target setting, publishing location, etc.

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】.

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
rosrun	Run the executable file in the feature pack
roslaunch	Run the startup file