3.Common commands and tools

3.1. Starting node method

3.1.1 Launch File

There are at least two ways to start the launch file using the roslaunch command:

1. Starting with the help of the ROS package pathThe format is as follows:

```
roslaunch package<name> launch文件名称
roslaunch pkg_name launchfile_name.launch
```

2. Directly provide the absolute path of the launch file

The format is as follows:

```
roslaunch path_to_launchfile
```

Regardless of which method is used to start the launch file, parameters can be added after it. Common parameters include

- `--screen`: Output the information of the ROS node (if any) to the screen instead of saving it in a certain log file, which is more convenient for debugging
- `arg:=value`: If the variables to be assigned in the launch file can be assigned in this way, for example:

```
roslaunch pkg_name launchfile_name model:=urdf/myfile.urdf # launch file
```

or

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

When the roslaunch command is run, it first checks whether the system's rosmaster is running. If it is already started, use the existing rosmaster; If it is not started, the rotor will be started first, and then the settings in the launch file will be executed to start multiple nodes at once according to our pre configured settings.It should be noted that the launch file does not need to be compiled, and after setting it up, it can be run directly using the above method.

3.1.2、rosrun

You must first start the node manager (master), which is used to manage many processes in the system. When each node starts, it must register with the master to manage communication between nodes. After the master is started, register each node node through the master. Enter the command in the Ubuntu terminal:

```
roscore
```

Node startup, 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]
```

ROSRUN will search for an executable program named executable in the package and pass in the optional parameter ARGS.

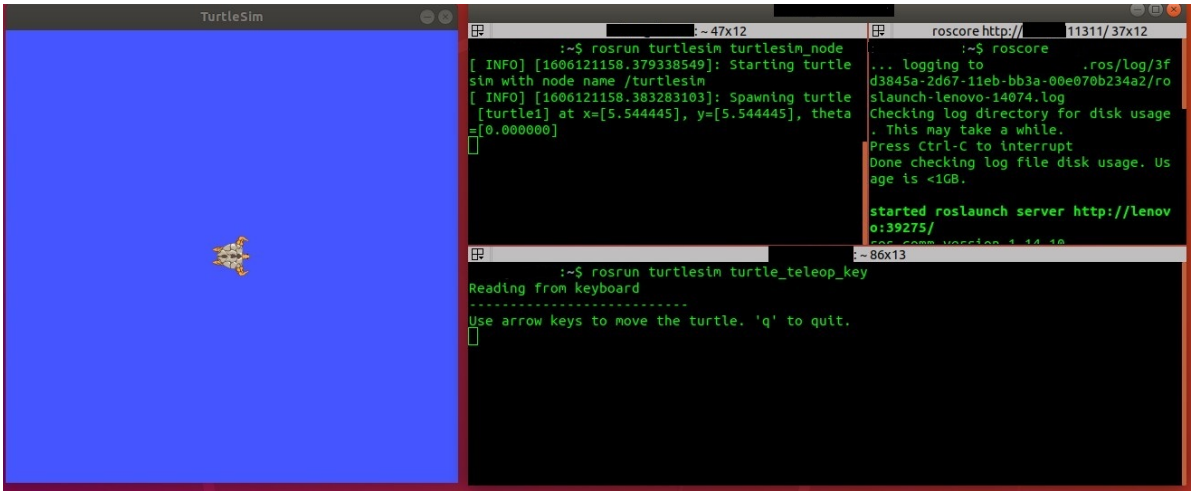
3.1.3、python

If it is Python code, you can directly start it in the directory where the py file is located, paying attention to distinguishing between Python 2 and Python 3.

3.1.4. Start a Little Turtle

```
roscore
rosrun turtlesim turtlesim_node
rosrun turtlesim turtle_teleop_key
```

After the startup is completed, you can control the movement of the small turtle through keyboard input. When operating the keyboard, the cursor must be on the [rosrun turnlesim turnle_teleop_key] command line, and click on the keyboard [Up], [Down], [Left], and [Right] to control the movement of the small turtle.



And in rosrun turnlesim turnlesim_ The node terminal will print some log information of the little turtle

```
[ 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 Little Turtles

Install Feature Pack

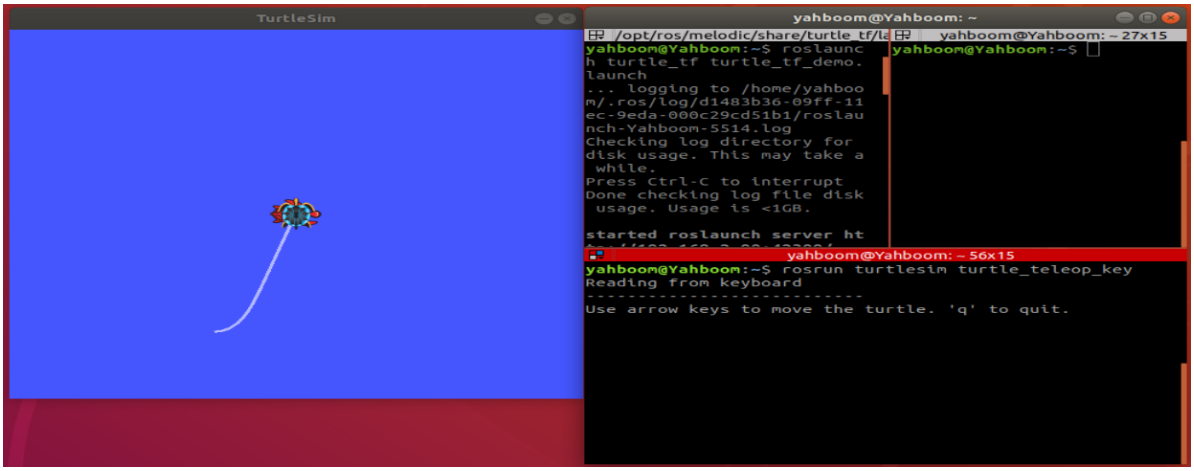
```
sudo apt install ros-melodic-turtle-tf
```

firing

```
roslaunch turtle_tf turtle_tf_demo.launch
```

Keyboard Control Node

```
rosrun turtlesim turtle_teleop_key
```



At this point, press the keyboard [Up], [Down], [Left], and [Right] to drive the little turtle to move; One small turtle can be observed following the movement of the other.

3.2 Launch Files

3.2.1 Overview

In ROS, a node program can generally only complete tasks with a single function, but a complete ROS robot usually requires multiple node programs to run simultaneously and collaborate with each other to complete complex tasks. Therefore, this requires that multiple node programs must be started when starting the robot. If one node starts one node, it can be quite troublesome. Multiple nodes can be started at once through the launch file and the roslaunch command, facilitating "one click start" and allowing for rich parameter settings.

3.2.1 File Format

A launch file is essentially an XML file that can highlight keywords in certain editors for easy reading, and can be added or not added at the header

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

Similar to other XML format files, launch files are also written using tags, with the main tags being as follows:

```
<launch>
<node>
<include>
<machine>
<env-loader>
<param>
<rosparam>
<arg>
<remap>
</launch>
```

1. Label 【 node 】

The label [node] is the core part of the launch file.

```
<launch>
    <node pkg="package_name" type="executable_file" name="node_name"/>
    <node pkg="another_package" type="another_executable" name="another_node">
</node>
    ...
</launch>
```

wherein

- Pkg is the package name where the node is located
- Type is an executable file in the package. If it is written in Python, it may be a. py file. If it is written in C++, it is the name of the executable file compiled from the source file
- Name is the name of the node after startup, and each node must have its own unique name.

Note: Rosslaunch cannot guarantee the startup order of nodes, so all nodes in the launch file should have robustness to the startup order.You can also set more parameters, as follows:

```
<launch>
    <node
        pkg=""
        type=""
        name=""
        respawn="true"
        required="true"
        launch-prefix="xterm -e"
        output="screen"
        ns="namespace"
    />
</launch>
```

In the above command,

- `respawn`: If the node is shut down, will it automatically restart
- `required`: If this node is closed, do you want to close all other nodes
- `launch-prefix`: Whether to open a new window for execution. For example, when controlling robot movement through a window, a new window should be opened for the control node; Or when the node outputs some information and does not want to mix it with other node information.

- `output`: By default, the information for launching a node will be stored in the following log file, which can be set as a parameter to display the information on the screen

```
/.ros/log/
```

- Classify nodes into different namespaces by adding the prefix specified by ns before the node name. To achieve this type of operation, relative names should be used when defining node names and topic names in the node source file, that is, without the sign/.

The names of calculation sources are divided into:

1. Basic name, for example: topic
2. Global name, for example:/A/topic
3. Relative name, for example: A/topic
4. Private name, for example:~topic

When publishing or subscribing, there is this line of code

```
ros::init(argc, argv, "publish_node");
 ros::NodeHandle nh;
 ros::Publisher pub = nh.advertise<std_msgs::string>("topic",1000);
```

2.Label [remap]

It often appears as a sub tag of the node tag and can be used to modify the topic. In many ROSNODE source files, there may not be a specific topic to receive or send, but only an input_ Topic and output_ Topic substitution requires replacing abstract topic names with specific topic names in specific scenarios during use.

Simply put, the function of remap is to facilitate the application of the same node file to different environments. By using remap to modify the topic externally, there is no need to change the source file.The common usage formats for remap are as follows:

```
<node pkg="some" type="some" name="some">
    <remap from="origin" to="new" />
</node>
```

3.Label 【 include 】

The purpose of this label is to add another launch file to this launch file, similar to the nesting of launch files. Basic format:

```
<include file="path-to-launch-file" />
```

The file path above can provide a specific path, but generally for the portability of the program, it is best to use the find command to provide the file path:

```
<include file="$(find package-name)/launch-file-name" />
```

In the above command, $(find package name) is equivalent to the path of the corresponding package on the local machine. In this way, even if other controllers are replaced, as long as the same package is installed, the corresponding path can be found.Sometimes, another node introduced by a launch may require a unified name, or node names with similar characteristics, such as/my/gps,/my/lidar,/my/imu, which means that nodes have a unified prefix for easy lookup. This can be achieved by setting the ns (namespace) attribute, with the following command:

```
<include file="$(find package-name)/launch-file-name " ns="my" />
```

4.Label [arg]

By using [arg], parameters can be reused and modified in multiple places simultaneously.[arg] Three common methods:

- `<arg name="foo">`: Declare a [arg] but do not assign a value. You can assign values later through the command line or through the [include] tag.
- `<arg name="foo" default="1">`: Assign default values。
- `<arg name="foo" value="1">`: Assign a fixed value。

Assigning values through the command line

```
roslaunch package_name file_name.launch arg1:=value1 arg2:=value2
```

5.Variable Replacement

There are two common forms of variable substitution in launch files

- `$(find pkg)`: For example, $(find rospy)/manifest. xml. If possible, we strongly recommend this package based path setting
- `$(arg arg_name)`: Set the default value first, and if there are no additional assignments, use this default value instead

For example:

```
<arg name="gui" default="true" />
<param name="use_gui" value="$(arg gui)"/>
```

Another example:

```
<node pkg="package_name" type="executable_file" name="node_name" args="$(arg a)
$(arg b)" />
```

After setting in this way, when starting roslaunch, the args parameter can be assigned a value

```
roslaunch package_name file_name.launch a:=1 b:=5
```

6.Label 【 param 】

Unlike [arg], [param] is shared and its values are not limited to value, but can also be files or even one line of commands.

- format

```
<param name="param_name" type="type1" value="val"/>
<param name="param_name" textfile="$(find pkg)/path/file"/>
<param name="param_name" command="$(find pkg)/exe '$(find pkg)/arg.txt'"/>

<param name="param" type="yaml" command="cat '$(find pkg)/*.yaml'"/>
```

Param can be in the global scope, where its name is the original name, or in a smaller scope, such as node, where its global name is in the form of node/paramFor example, in the global scope, define the following param

```
<param name="publish_frequency" type="double" value="10.0" />
```

Define the following param in the node scope

```
<node name="node1" pkg="pkg1" type="exe1">
    <param name="param1" value="False"/>
 </node>
```

If using the rosparam list to list 【 param 】 in the server, there are

```
/publish_frequency
/node1/param1
```

Note: Although the name 'param' is prefixed with a namespace, it still has a global scope

7.Label 【 rosparam 】

【 param 】 can only operate on a single 【 param 】, and there are only three forms: value, textfile, and command. The returned content is the content of a single

【 param 】.Rosparam can perform batch operations and also includes commands for parameter settings, such as dump, delete, etc

- Load: Load a batch of params from the YAML file, with the following format:

```
<rosparam command="load" file="$(find rosparam)/example.yaml" />
```

- Delete: Delete a param

```
<rosparam command="delete" param="my_param" />
```

- Assignment operation similar to 'param'

```
<rosparam param="my_param">[1,2,3,4]</rosparam>
```

or

```
<rosparam>
a: 1
b: 2
</rosparam>
```

A 'rosparam' can also be placed in a 'node', where the 'param' name is preceded by a node namespace

8.Label [group]

If you want to set the same settings for multiple nodes, such as all in the same specific namespace, remap the same topic, etc., you can use [group]. All common tags can be used for setting in the group, such as

```
<group ns="rosbot">
    <remap from="chatter" to="talker"/>
    <node ... />
    <node ... >
        <remap from="chatter" to="talker1"/>
    </node>
</group>
```

3.3 TF coordinate transformation

Tf is a feature package that allows users to track multiple coordinate systems at any time. Tf maintains the relationships between coordinate frames in a real-time buffered tree structure, and allows users to convert points, vectors, etc. at any time point between any two coordinate frames.The Tf package is to convert the coordinates of a point in one coordinate system into the coordinates of another coordinate system. Sensors can be seen as a coordinate system, machines can be seen as a coordinate system, and obstacles can be seen as a point.After starting two small turtles in 【 3.1.5 】, perform the following operations.

3.3.1 Common Tools for tf

1.view_ Frames tool

Can listen to all the tf coordinate systems broadcasted through ROS at the current time, and draw a tree diagram to represent the connection relationship between 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, but it cannot reflect the coordinate relationship in real-time, so rqt can be used_ tf_ Tree refreshes the display of coordinate system relationships in real-time

```
rosrun rqt_tf_tree rqt_tf_tree
```

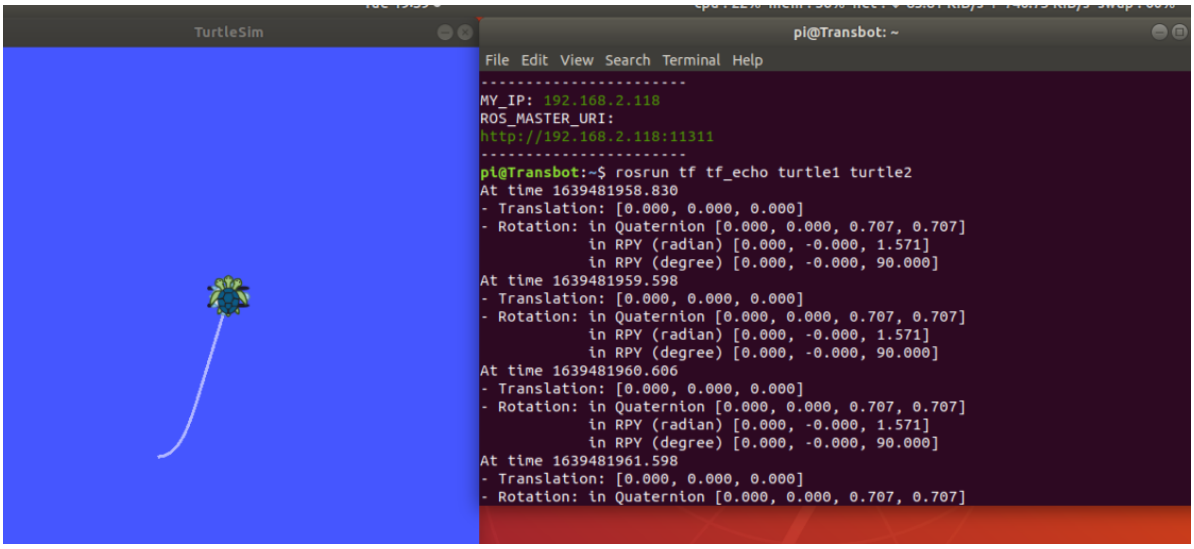
3.tf_ Echo tool

Using tf_ The echo tool can view the relationship between two broadcast reference frames.

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

Print from source_ Frame to target_ Rotation translation transformation of frame; For example:

```
rosrun tf tf_echo turtle1 turtle2
```



4.static_transform_publisher

Publish a static coordinate transformation between two coordinate systems without any relative positional changes. 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>
```

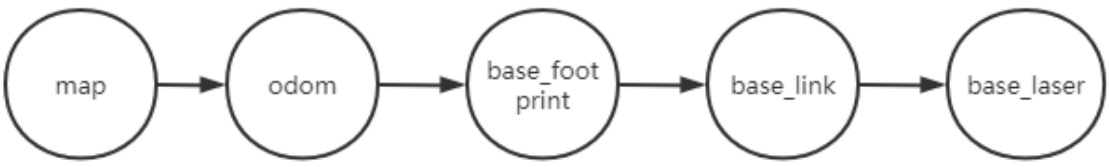
5.roswtf plugin

A plugin that analyzes your current tf configuration and attempts to identify common issues.

```
roswtf
```

3.3.2 Common Coordinate Systems

The commonly used coordinate system is the frame_ ID, with map, odom, base_ link， base_ footprint， base_ Laser et al.



- World coordinates (map)

The map coordinate system is a world fixed coordinate system with its Z-axis pointing upwards. The posture of a mobile platform relative to the map coordinate system should not significantly move over time. The map coordinates are discontinuous, which means that the posture of the mobile platform in the map coordinate system can undergo discrete jumps at any time. In a typical setup, the positioning module continuously recalculates the robot's pose in world coordinates based on sensor monitoring to eliminate bias, but may jump when new sensor information arrives. The map coordinate system is useful as a long-term global reference, but jumping makes it a poor reference coordinate for local sensors and actuators.

- Odometer coordinate system (odom)

Odom is a global coordinate system that records the current motion posture of the robot through a odometer. The pose of the mobile platform in the ODOM coordinate system can be moved arbitrarily without any boundaries, making the ODOM coordinate system unable to serve as a long-term global reference. Here we need to distinguish between odom topics, which are two concepts: one is the coordinate system, and the other is the odometer calculated based on encoders (or vision, etc.). But there is also a relationship between the two. The pose matrix transformed from odom topic is odom ->base_ The tf relationship of the link. The odom and map coordinate systems coincide at the beginning of robot motion. However, over time, there is no overlap, and the deviation that occurs is the cumulative error of the odometer. In some calibration sensor collaborative calibration packages, such as amcl, a location estimation is provided, which can obtain a map ->base_ The tf of the link, so the estimated deviation between the position and the odometer position is the coordinate system deviation between the odom and the map. If your odom calculation has no errors, then the tf of map ->odom is 0. The odom coordinate system is useful as a short-term local reference, but offset makes it unsuitable as a long-term reference.

- base_link

The coordinate system of the robot body (base) coincides with the center of the robot, and the origin of the coordinate system is generally the rotation center of the robot.base_ Footprint: Origin is base_ There are some differences in the projection of the link origin on the ground (with different z values).

- The relationship between coordinates

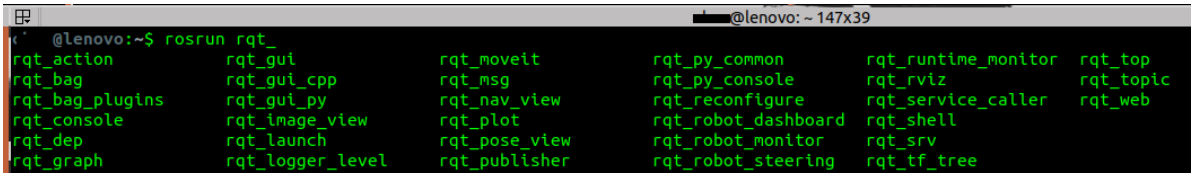
In robot systems, we use a tree to associate all coordinate systems, so each coordinate system has a parent coordinate system and any child coordinate system, as follows: map -->odom -->base_ The world coordinate system is the parent of the odom coordinate system, and the odom coordinate system is the base_ The parent of the link. Although intuitively speaking, map and odom should be connected to base_ Link, this is not allowed because each coordinate system can only have one parent class.

- Coordinate System Permissions

Odom to base_ The conversion of links is calculated and published by the odometer source. However, the positioning module does not publish maps to the base_ The transformation of links. On the contrary, the positioning module first receives the odom to the base_ Link's transformation, and use this information to publish the map to Odom's transformation.

3.4、rqt

Open a command line window and enter rosrun rqt, then double-click the Tab key to view the content contained in the QT tool in ROS, as shown in the following figure:

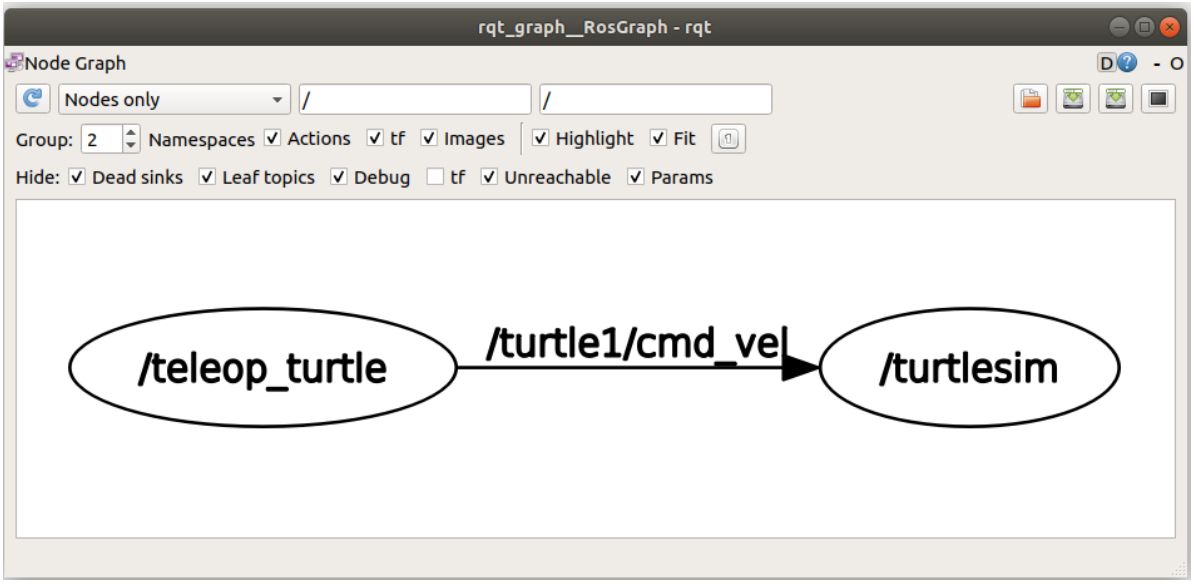


Next, let's take Little Turtle as an example to briefly introduce several commonly used QT tools:

1.rqt_ Visualization of Graph Calculation Graph

Open a command line window and enter the following command to pop up a dialogue window.

```
rosrun rqt_graph rqt_graph
```



From the image, we can clearly see/delete_ Turtle node through/turtle1/cmd_ The vel topic is used to transfer data to/tourlesim nodes.

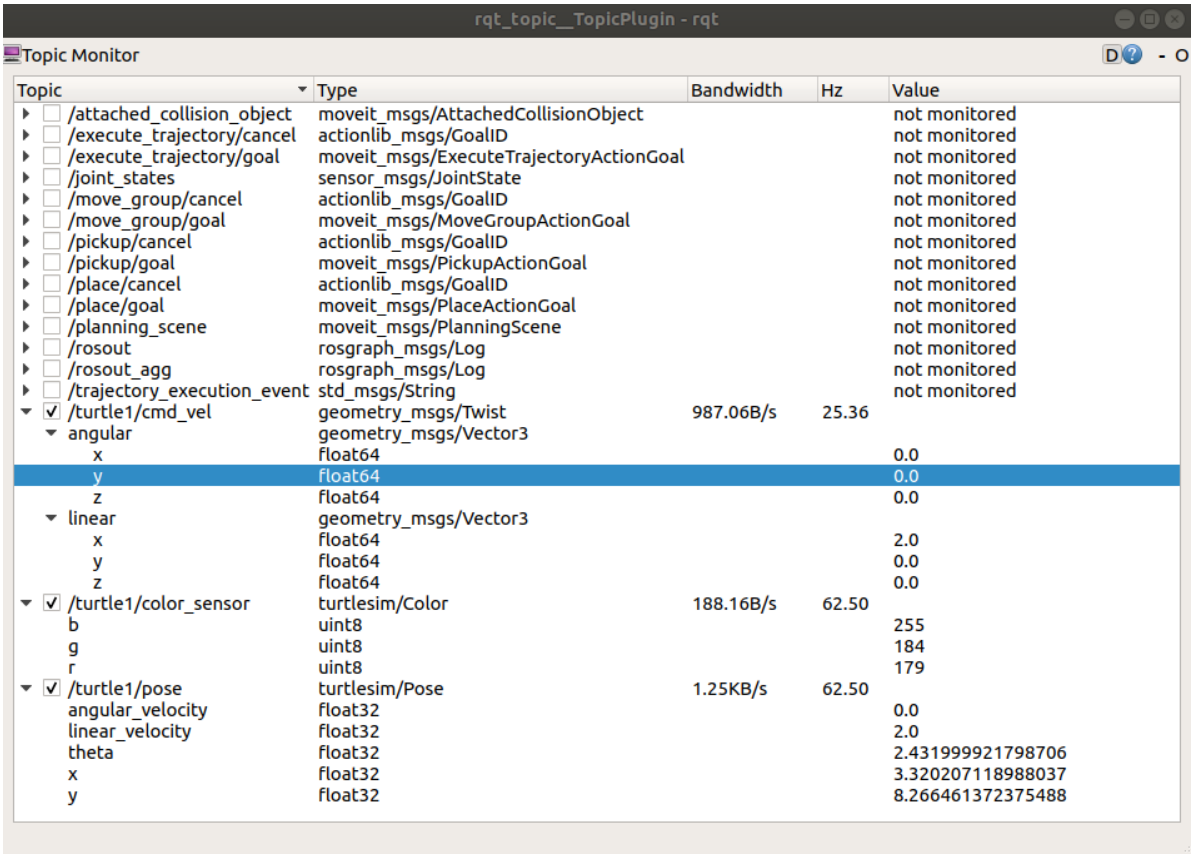
/teleop_ Turtle is a node with Publisher functionality.

/Turtlesim is a node that has the ability to subscribe.

/turtle1/cmd_ Vel is the topic of communication between publisher and subscriber.

2.rqt_topic

```
rosrun rqt_topic rqt_topic
```



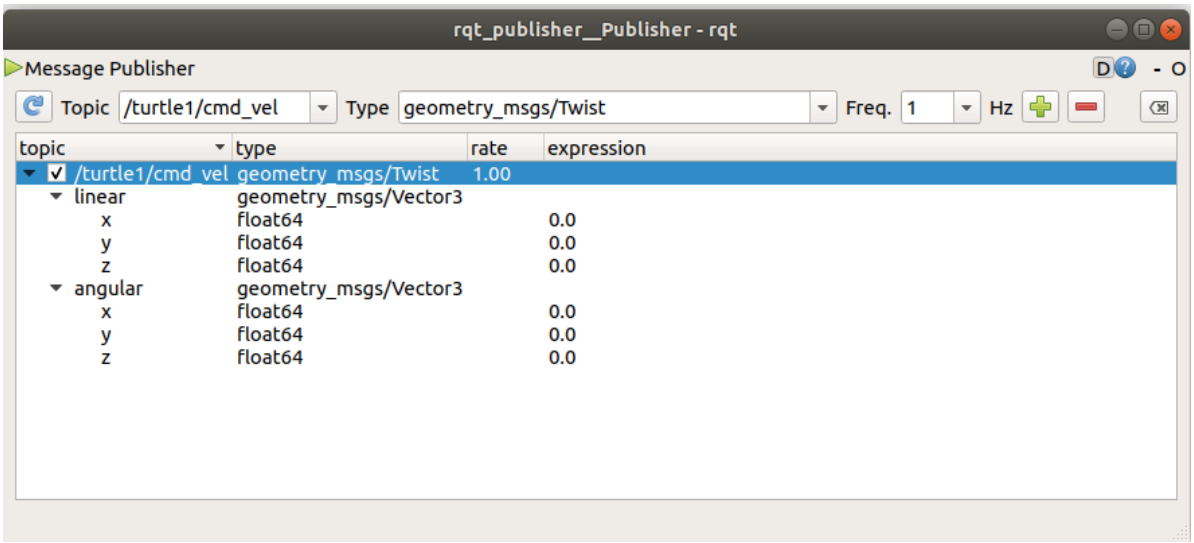
Through this tool, we can clearly see some real-time changes in the information of the little turtle.

3.rqt_publisher

rqt_ Publisher provides a GUI plugin for publishing any message with fixed or calculated field values. Open a command line window and enter the following command to pop up a dialogue window.

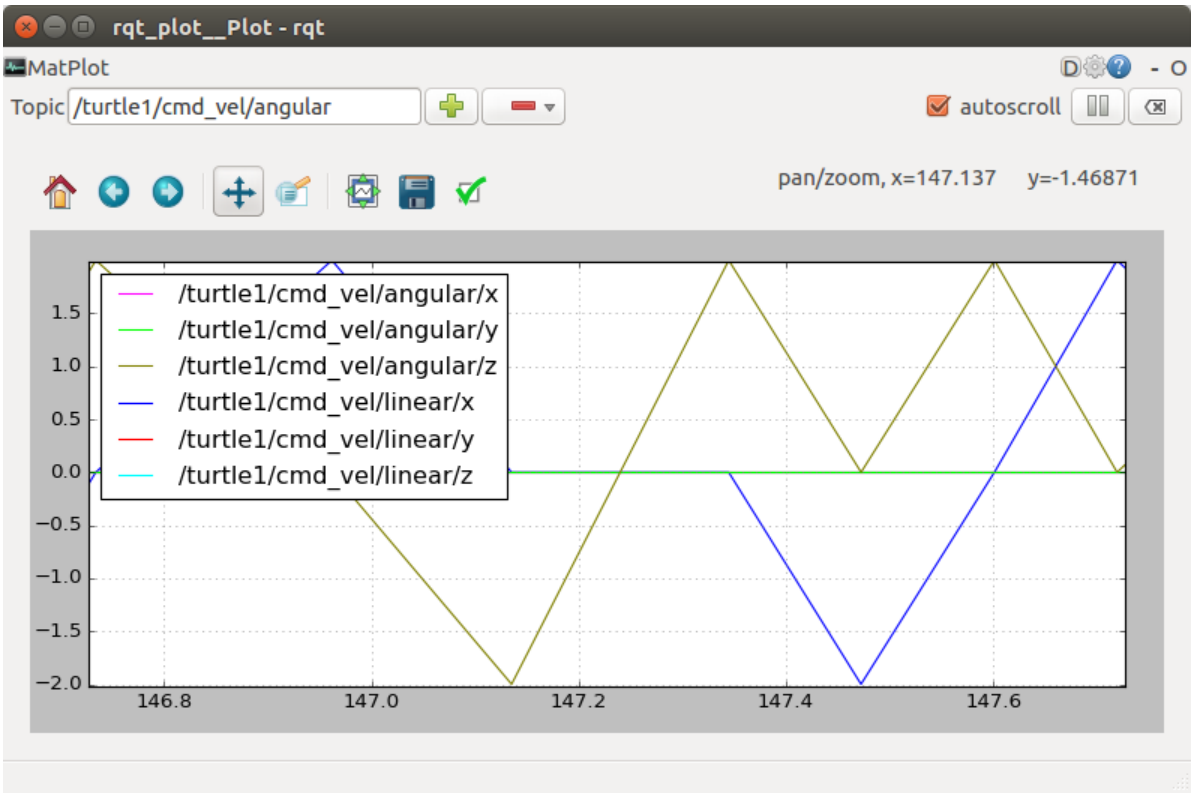
```
rosrun rqt_publisher rqt_publisher
```

Click on the selection box on the right side of the Topic to find the/tourle1/cmd we need_ For the vel topic, click the plus sign on the right to add it, as shown below:



4.rqt_plot

```
rosrun rqt_plot rqt_plot
```



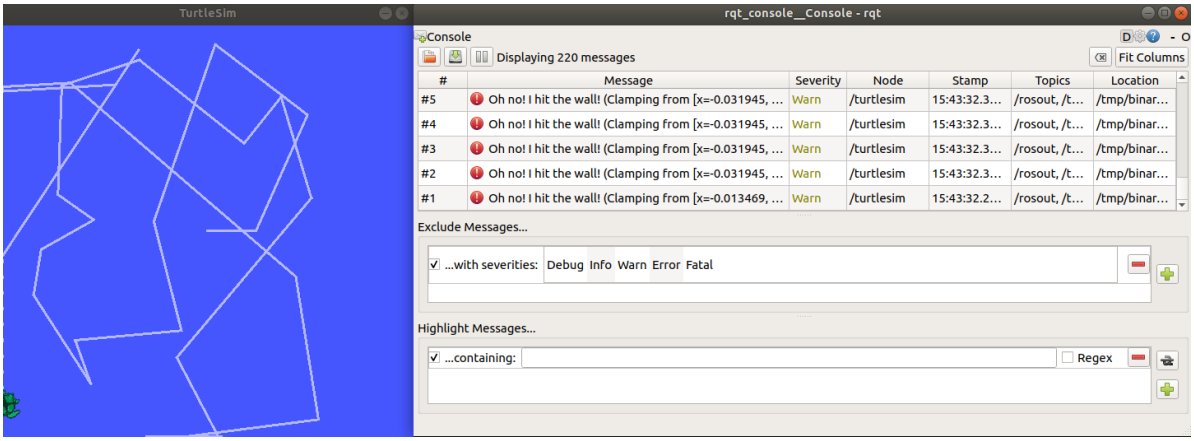
6.rqt_console

The function of the ROS log system is to allow programs to generate log messages that are displayed on the screen, sent to specific topics, or stored in specific log files for easy debugging, recording, alarm, and more.

Serial Number	grade	analysis
1	DEBUG	Debug logs for development and testing purposes
2	INFO	Regular logs, user visible level information
3	WARN	Warning message.
4	ERROR	Error message. Information printed after program error
5	FATAL	Fatal error. Logging of downtime

The log messages in ROS can be classified into 5 levels based on their severity: DEBUG, INFO, WARN, ERROR, and FATAL. As long as the program can run, there is no need to pay attention, but the appearance of ERROR and FATAL indicates that the program has serious problems that prevent it from running.

```
rosrun rqt_console rqt_console
```

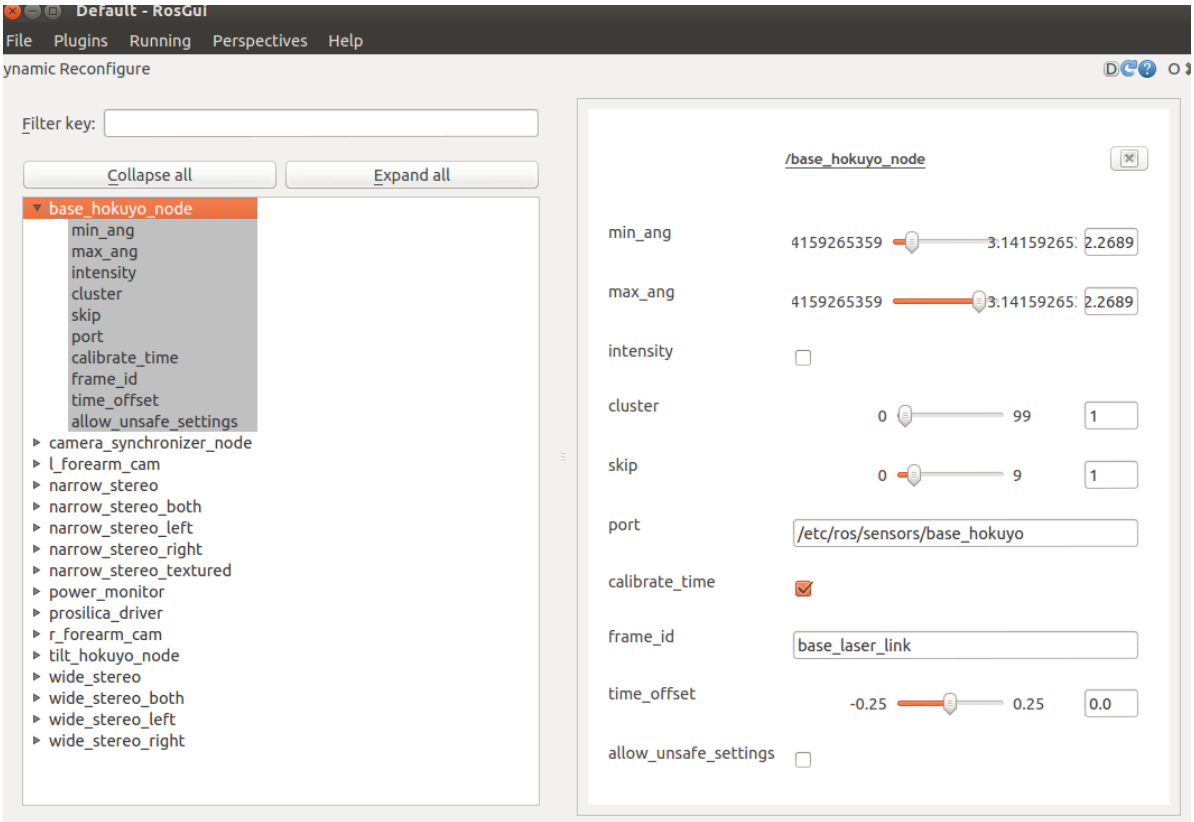


The log output tool is part of the ROS logging framework and is used to display the output information of nodes. From the graph, we can see that it again prompts that the turtle has hit the wall.

7.rqt_reconfigure

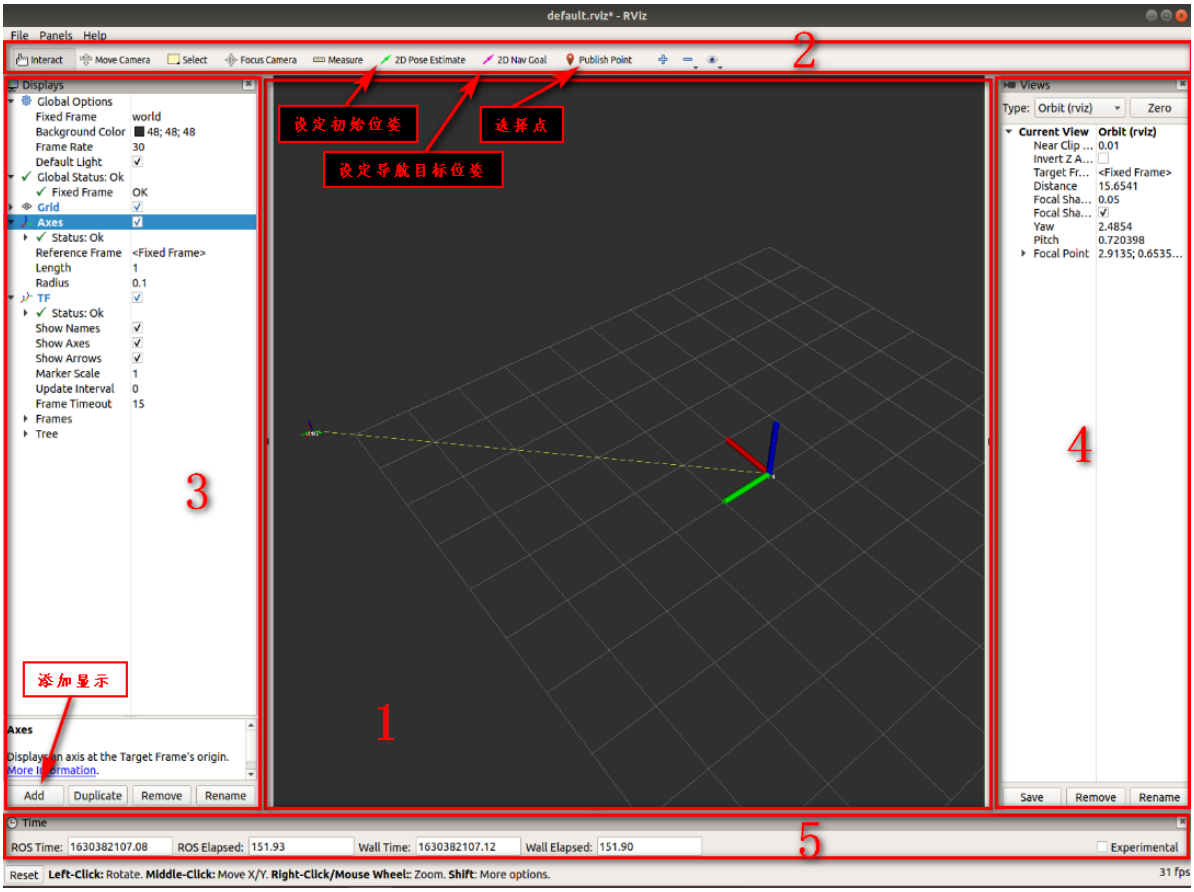
```
rosrun rqt_reconfigure rqt_reconfigure
```

Image source ROS wiki:



3.5、Rviz

Rviz is a graphical tool that comes with ROS, which allows for convenient graphical operation of ROS programs. Its use is also relatively simple.



[Set initial position], [Set target position], [Publish location point]: Generally used during image navigation.The rviz interface mainly includes the following parts:

1: The 3D view area is used for visualizing data. Currently, there is no data available, so it is displayed in black.

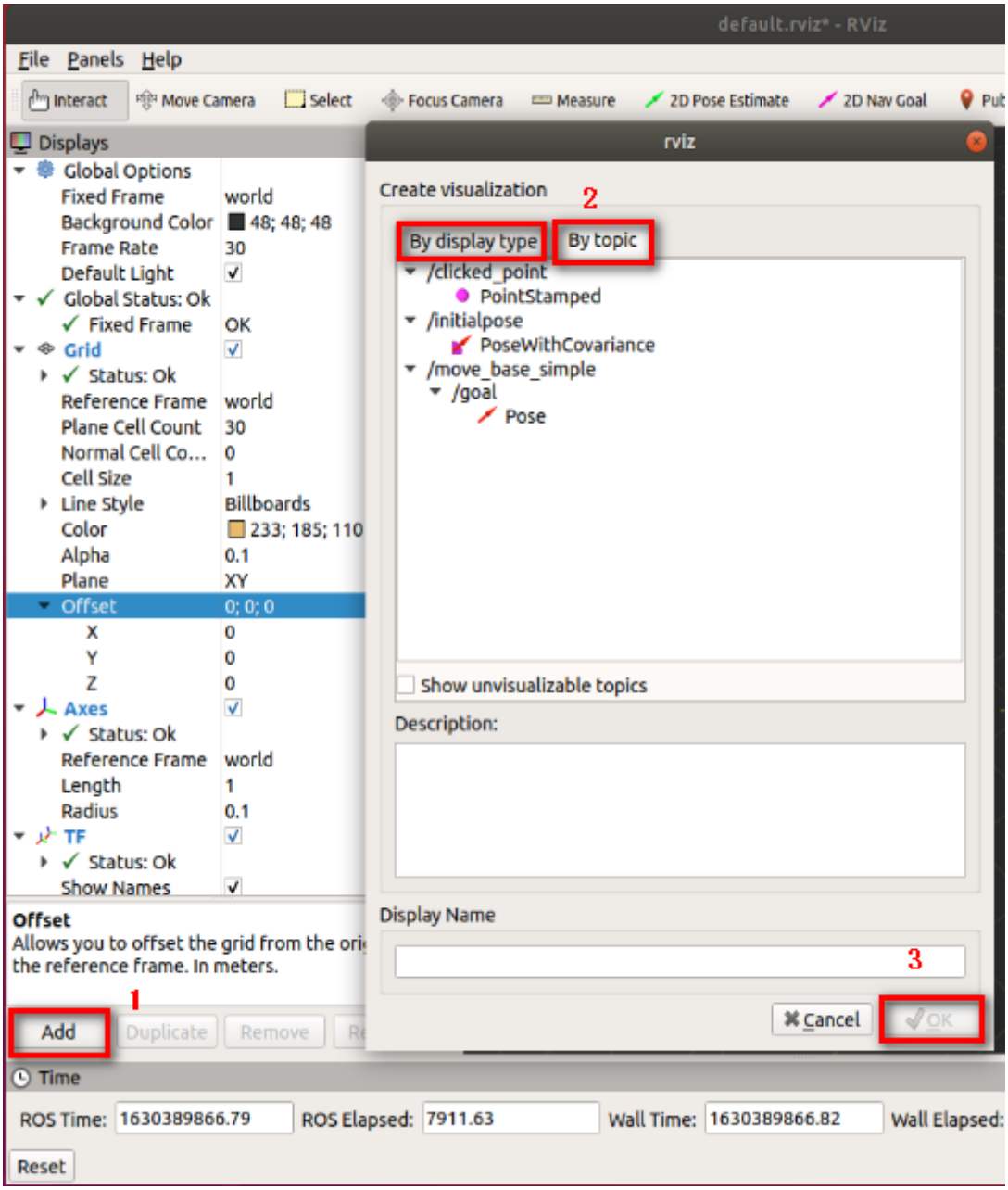
2: Toolbar, providing tools such as perspective control, target setting, and publishing location.

3: Display item list, used to display the currently selected display plugin, and the properties of each plugin can be configured.

4: The viewing angle setting area allows for multiple observation angles to be selected.

5: The time display area displays 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 add the topic directly by selecting 'By topic' to display it normally.

Step 3: Click [OK] to proceed.

3.6. ROS Common Commands

command	role
catin_create_pkg	Information on creating feature packs
rospack	Obtain information about feature packs
catkin_make	Compiling feature packages in the workspace
rosdep	Automatically install other packages dependent on feature packs
roscd	Function Pack Directory Jump

command	role
roscp	Copy files from the feature pack
rosed	Editing files in the feature pack
rosrun	Run executable files in the feature pack
roslaunch	Run startup file