2. Multi-machine formation

2.1. Introduction

For how to configure multi-machine communication and synchronize time, please refer to the [ros distributed communication] lesson; if there is a network, directly synchronize the network system time without setting.

When using multi-machine control, first make sure that the robots are in the same LAN and configure the same [ROS_MASTER_URI]; there can only be one master for multiple robots to control movement. In this section, the virtual machine is set as the master and other robots as slaves. There can be several slaves; of course, you can also set a robot as the master and the others as slaves.

2.2. Use

Note: The current effect is not good, the code is for reference only

Take the virtual machine as the master and the three robots as slaves as an example; a map must be available before use. The three slaves are [robot1], [robot2], and [robot3], and [robot1] is set as the leader, and [robot2] and [robot3] are set as followers. When playing this function, make sure the venue is large enough to avoid collision. And there is no obstacle avoidance function! ! !

2.2.1, start the robot

Virtual machine side

```
roscore
```

Start command (robot1 side), for the convenience of operation, this section takes [chassis + depth camera] as an example.

```
roslaunch yahboom_navrobo_multi scout_mini_robot_base_multi.launch ns:=robot1 #
Astra + laser + yahboom
```

Start command (robot2 side), for the convenience of operation, this section takes [chassis + depth camera] as an example.

```
roslaunch yahboom_navrobo_multi scout_mini_robot_base_multi.launch ns:=robot2 #
Astra + laser + yahboom
```

More robots are analogous.

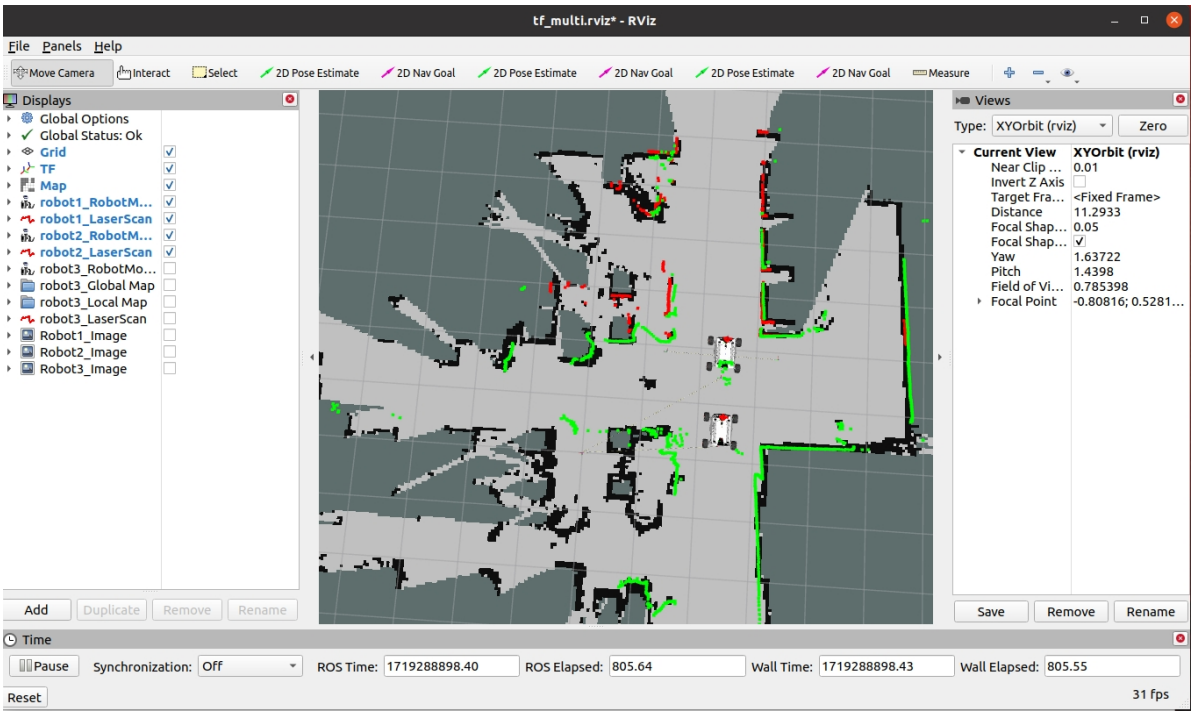
2.2.2, Enable multi-machine formation

Virtual machine side

```
roslaunch yahboom_navrobo_multi tf_queuebroad.launch use_rviz:=true map:=my_map
```

- [use_rviz] parameter: whether to open rviz.
- [map] parameter: map name, map to be loaded.

After startup, the robot needs to be initialized. For specific setting methods, refer to [Multi-machine Navigation Lesson], and the setting is as shown below.



2.2.3, Formation control

Open the dynamic parameter adjustment tool

```
rosrun rqt_reconfigure rqt_reconfigure
```

This tool can be set for each robot separately.

Parameter analysis

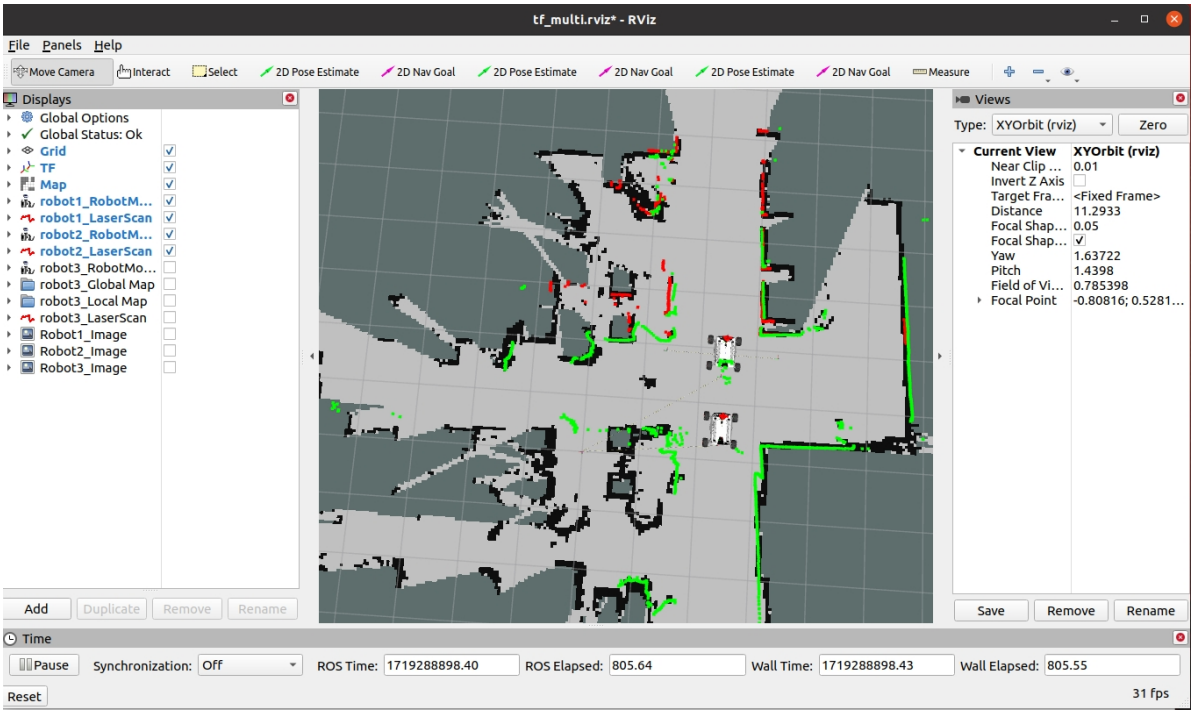
[lin_Kp], [lin_Ki], [lin_Kd]: Car linear speed PID debugging.

[ang_Kp], [ang_Ki], [ang_Kd]: Car angular speed PID debugging.

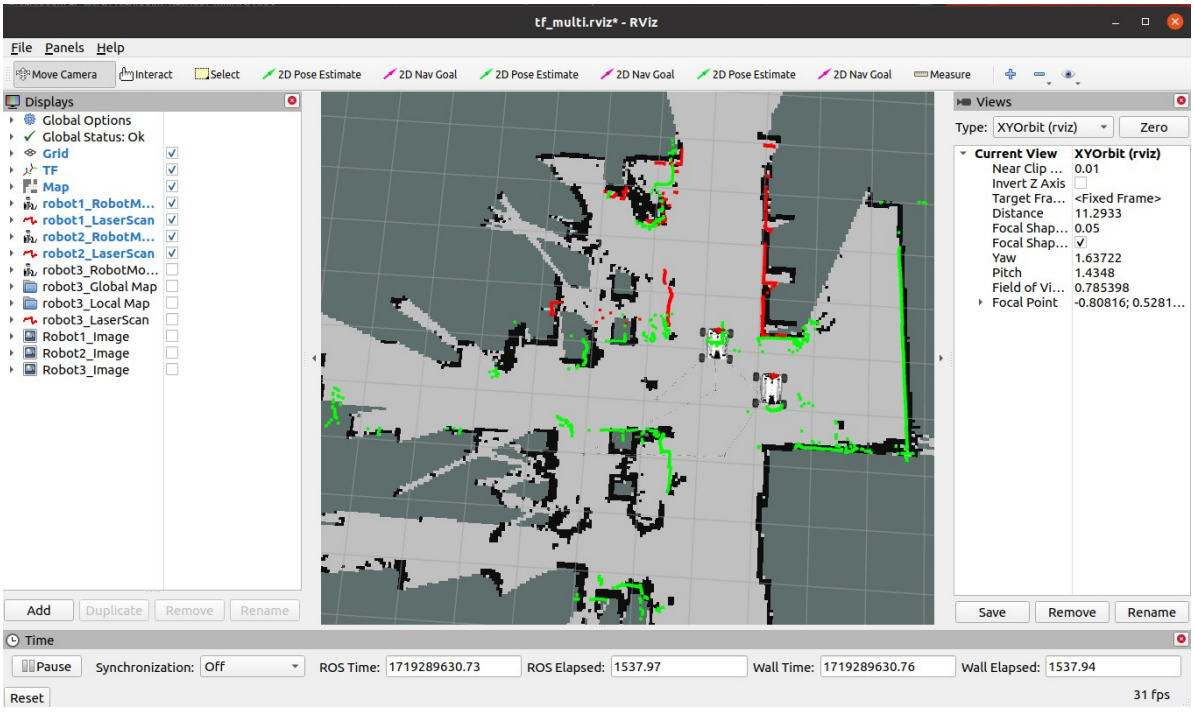
Parameter	Range	Analysis
【teams】	Default 【vertical】	Formation: 【convoy,vertical, horizontal】
【robot_model】	Default 【omni】	Model: 【omni,diff】
【navigate】	【False, True】	Whether to run in navigation mode
【Switch】	【False, True】	Function switch 【Start/Pause】
【dist】	【0.5, 2.0】	Distance between teams

After turning on the 【Switch】 function switch, the robot automatically maintains the formation set by the 【teams】 parameter. The 【teams】 parameter only recognizes the most recently set value. As shown in the figure below

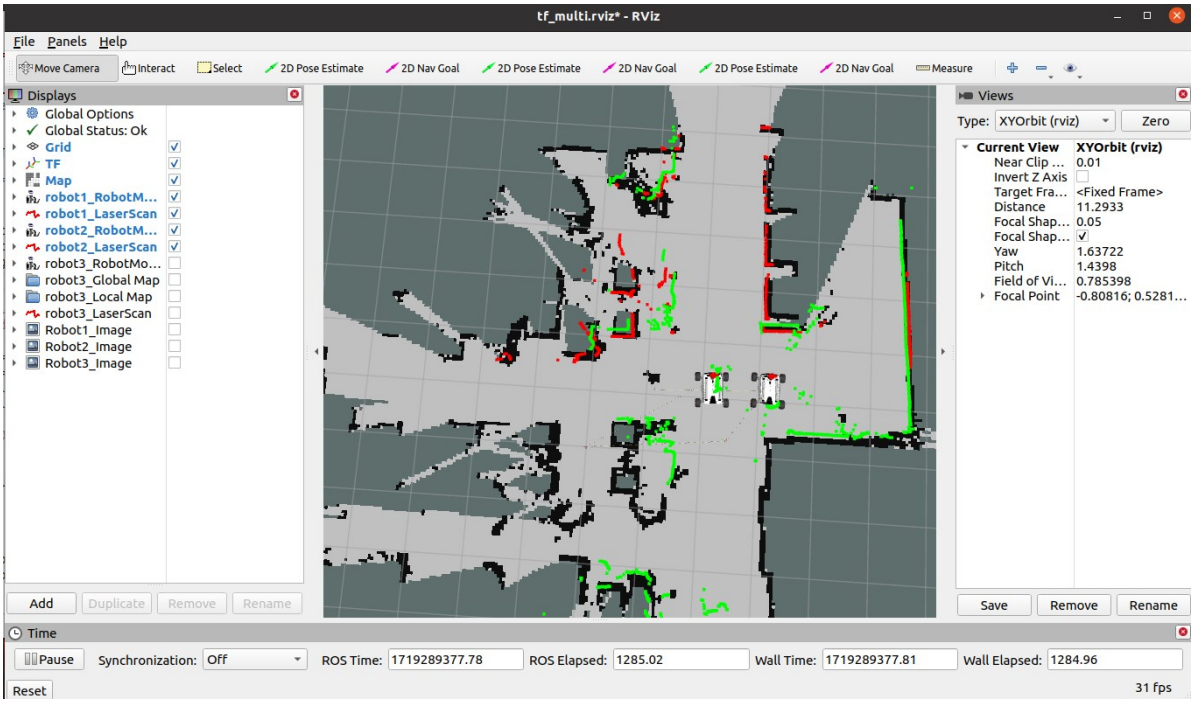
- A vertical column



- Left and right guards (convoy)



- A horizontal column (horizontal)



After the formation is set, when we control [robot1], other robots will automatically maintain the formation without control.

2.3, launch file

tf_queuebroad.launch

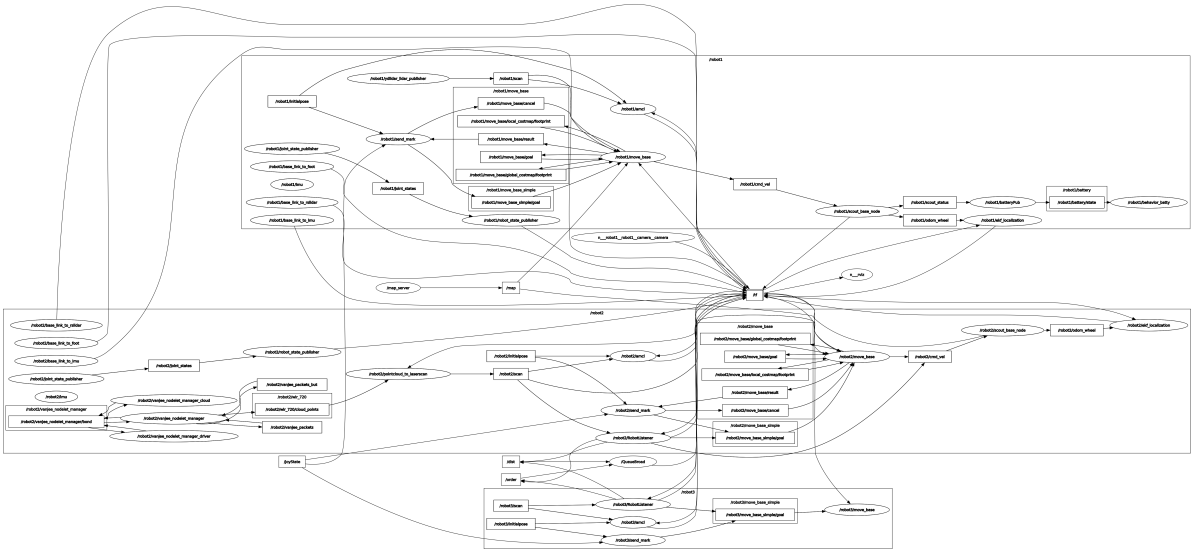
```
<launch>
    <arg name="first_robot1" default="robot1"/>
    <arg name="second_robot2" default="robot2"/>
    <arg name="third_robot3" default="robot3"/>
    <!-- 地图名 || Map name -->
    <arg name="map" default="my_apm"/>
    <!-- 加载地图 || Load map -->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find
yahboomcar_nav)/maps/$(arg map).yaml"/>
    <node pkg="rviz" type="rviz" name="rviz" required="true" args="-d $(find
yahboom_navrobo_multi)/rviz/tf_multi.rviz"/>

    <!-- ############################## first_robot1 ##############################
-->

    <node pkg="yahboom_navrobo_multi" type="broad_queue.py" name="QueueBroad"
output="screen" args="$(arg first_robot1)"/>
    <include file="$(find
yahboom_navrobo_multi)/launch/library/move_base_multi.launch">
        <arg name="ns" value="$(arg first_robot1)"/>
    </include>

    <!-- ############################## second_robot2
############################## -->

    <node pkg="yahboom_navrobo_multi" type="listener.py" name="RobotListener"
output="screen"
          args="$(arg second_robot2) point1" ns="$(arg second_robot2)/">
        <rosparam param="linPIDparam">[1.0, 0, 1.0]</rosparam>
        <rosparam param="angPIDparam">[0.8, 0, 1.0]</rosparam>
```

```
    </node>
    <include file="$(find
yahboom_navrobo_multi)/launch/library/move_base_multi.launch">
        <arg name="ns" value="$(arg second_robot2)"/>
    </include>

    <!-- ############################ third_robot3 ############################
-->

    <node pkg="yahboom_navrobo_multi" type="listener.py" name="RobotListener"
output="screen"
          args="$(arg third_robot3) point2" ns="$(arg third_robot3)/">
        <rosparam param="linPIDparam">[1.0, 0, 1.0]</rosparam>
        <rosparam param="angPIDparam">[0.8, 0, 1.0]</rosparam>
    </node>
    <include file="$(find
yahboom_navrobo_multi)/launch/library/move_base_multi.launch">
        <arg name="ns" value="$(arg third_robot3)"/>
    </include>
</launch>
```

2.4, Framework Analysis

Node View

```
rqt_graph
```



The [QueueBroad] node subscribes to queue instructions from each robot. This node will only recognize the most recent formation setting.

View tf tree

```
rosrun rqt_tf_tree rqt_tf_tree
```

From the above figure, we can see that [robot1] will issue two coordinate systems [point1] and [point2]. [robot2] and [robot3] monitor the relationship between themselves and the coordinate system in real time and make their own coordinate system coincide with the coordinate system. In the [tf_queuebroad.launch] file, we can see that [robot2] follows the [point1] coordinate system; [robot3] follows the [point2] coordinate system.