

1.Multi-vehicle control

1. Program function description

Connect the controller to the virtual machine, set the virtual machine to connect to the controller, and after the virtual machine and Muto programs are started, you can control two Mutos at the same time through the controller, and the movements of the two Mutos are synchronized.

The handle remote control will enter sleep mode after being turned on for a period of time, and you need to press the "START" button to end sleep. If you want to **control the operation of the Muto car**, you also need to **press the R2 key** and **release the motion control lock** before you can use the joystick to control the movement of the Muto car.

2. Program reference path

After entering the docker container, the source code of this function is located at

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_multi/launch
```

The file started by the virtual machine, the source code is located in

```
/home/yahboom/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_multi/yahboomcar_multi
```

3. Program startup

Note: Multi-machine communication needs to be set up in advance. That is to say, the virtual machine and Muto (taking two Muto as an example) need to be in the same LAN and Muto docker, and the **ROS_DOMAIN_ID** of the virtual machine must be the same to enable distributed communication.

3.1. Start command

After entering the Muto1 and Muto2 docker containers respectively, enter in the terminal according to the actual car model

```
#Muto1
ros2 launch yahboomcar_multi bringup_multi_launch.xml robot_name:=robot1
#Muto2
ros2 launch yahboomcar_multi bringup_multi_launch.xml robot_name:=robot2

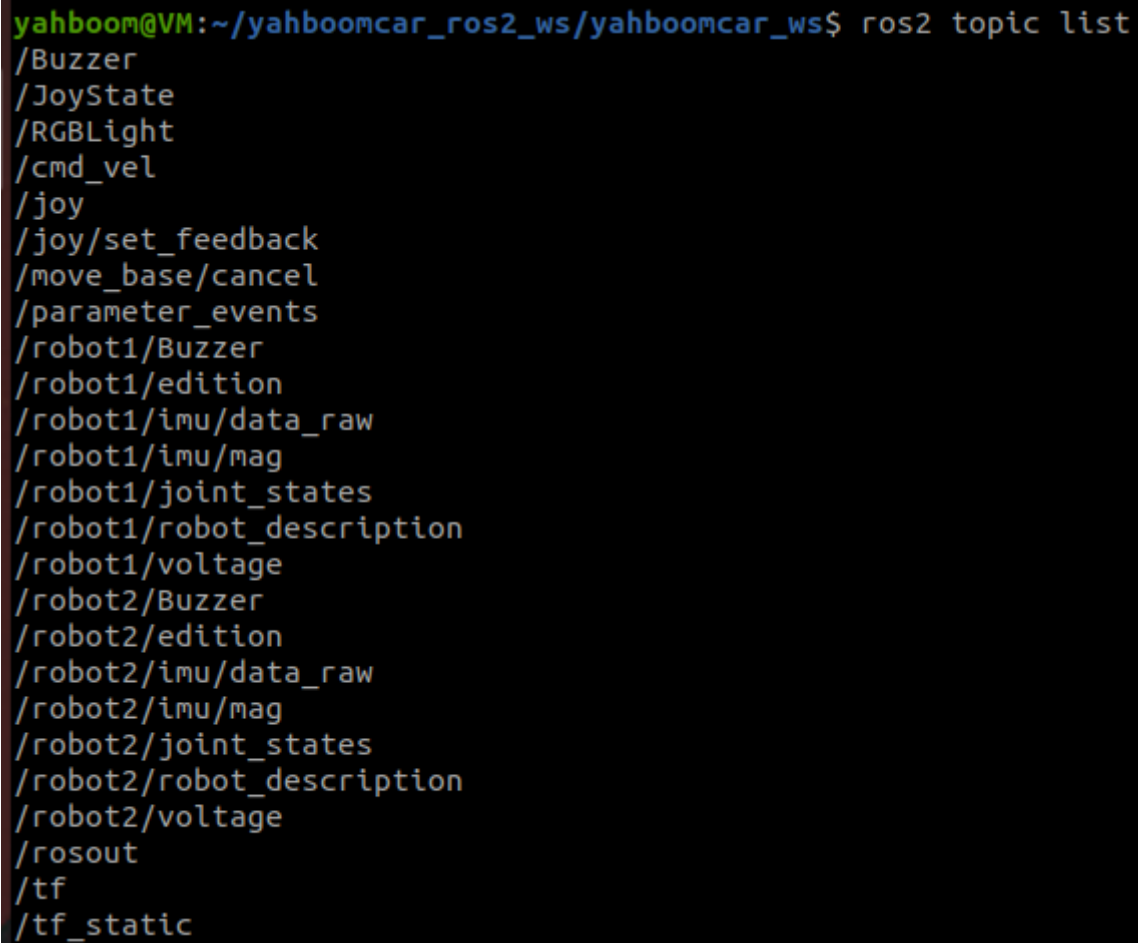
#Start handle control on virtual machine side
ros2 launch yahboomcar_ctrl yahboomcar_joy_launch.py
```

[robot_name] indicates the serial number of the vehicle to start. Currently, the program can choose to start robot1 and robot2.

3.2. View topic nodes

Enter the following command in the virtual machine terminal,

```
ros2 topic list
```

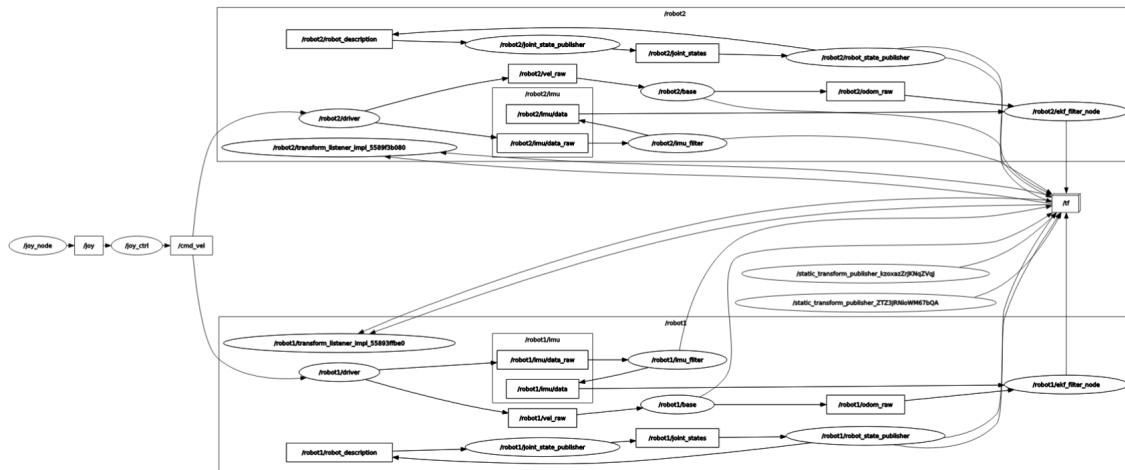
A terminal window with a black background and green text. The prompt is 'yahboom@VM:~/yahboomcar_ros2_ws/yahboomcar_ws\$'. The command 'ros2 topic list' has been executed, and the output is a list of topics. The topics are: /Buzzer, /JoyState, /RGBLight, /cmd_vel, /joy, /joy/set_feedback, /move_base/cancel, /parameter_events, /robot1/Buzzer, /robot1/edition, /robot1/imu/data_raw, /robot1/imu/mag, /robot1/joint_states, /robot1/robot_description, /robot1/voltage, /robot2/Buzzer, /robot2/edition, /robot2/imu/data_raw, /robot2/imu/mag, /robot2/joint_states, /robot2/robot_description, /robot2/voltage, /rosout, /tf, and /tf_static.

```
yahboom@VM:~/yahboomcar_ros2_ws/yahboomcar_ws$ ros2 topic list
/Buzzer
/JoyState
/RGBLight
/cmd_vel
/joy
/joy/set_feedback
/move_base/cancel
/parameter_events
/robot1/Buzzer
/robot1/edition
/robot1/imu/data_raw
/robot1/imu/mag
/robot1/joint_states
/robot1/robot_description
/robot1/voltage
/robot2/Buzzer
/robot2/edition
/robot2/imu/data_raw
/robot2/imu/mag
/robot2/joint_states
/robot2/robot_description
/robot2/voltage
/rosout
/tf
/tf_static
```

3.3. View the node communication diagram

Enter the following command in the virtual machine terminal,

```
ros2 run rqt_graph rqt_graph
```



It can be seen that the virtual machine published the topic speed of /cmd_vel, and the chassis of robot1 and robot2 have subscribed to the topic, so when the virtual machine publishes the topic data, both Muto will receive it and then control the movement of their respective chassis.

4. Core code analysis

The code on the virtual machine side is the same as the controller control code on the Muto side. I won't go into details here. Let's mainly look at the content of the Muto side. The startup file is bringup_multi_launch.xml.

```
<launch>
  <arg name="robot_name" default="robot1"/>
  <group>
    <push-ros-namespace namespace="$(var robot_name)"/>
    <!--driver_node-->
    <node name="driver" pkg="yahboomcar_bringup" exec="muto_driver"
output="screen">
      <param name="imu_link" value="$(var robot_name)/imu_link"/>
      <remap from="cmd_vel" to="/cmd_vel"/>
    </node>
  </group>
  <include file="$(find-pkg-share
yahboomcar_description)/launch/description_multi_$(var robot_name).launch.py"/>
</launch>
```

The xml format is used here to write the launch file, which facilitates us to add namespaces in front of multiple nodes. Adding namespace is to solve the conflict caused by the same node name. We have two Mutos here. Take the chassis program as an example. If both chassis nodes are named driver, then after multi-machine communication is established, this is not allowed. So we add the namespace **namespace** before the node name. In this way, the chassis node name of car 1 is /robot1/driver, and the chassis node program of car 2 is /robot2/driver. In the launch file in XML format, a group is used to specify that within this group, both the node name and the topic name need to be added with **namespace**.

```
<group>
  <push-ros-namespace namespace="$(var robot_name)"/>
</group>
```

To reference the parameters defined in the launch file in XML format, use **\$(var robot_name)**. The parameter passed in here is robot_name. When entering the command, you can enter it in the command line.

```
ros2 launch yahboomcar_multi bringup_multi_launch.xml robot_name:=robot1
```

One thing to note here is that after adding the namespace, we remapped the topic name of /robot1/cmd_vel to /cmd_vel in order to receive topic messages sent by the virtual machine.

```
<node name="driver" pkg="yahboomcar_bringup" exec="muto_driver" output="screen">
  <param name="imu_link" value="$(var robot_name)/imu_link"/>
  <remap from="cmd_vel" to="/cmd_vel"/>
</node>
```

Regarding the passing of parameters, adding the namespace needs to be stated in the launch file, such as the following parameters,

```
<param name="imu_link" value="$(var robot_name)/imu_link"/>
```

It can be seen that the **imu_link** parameter is assigned to **\$(var robot_name)/imu_link**.