Multi-car handle control

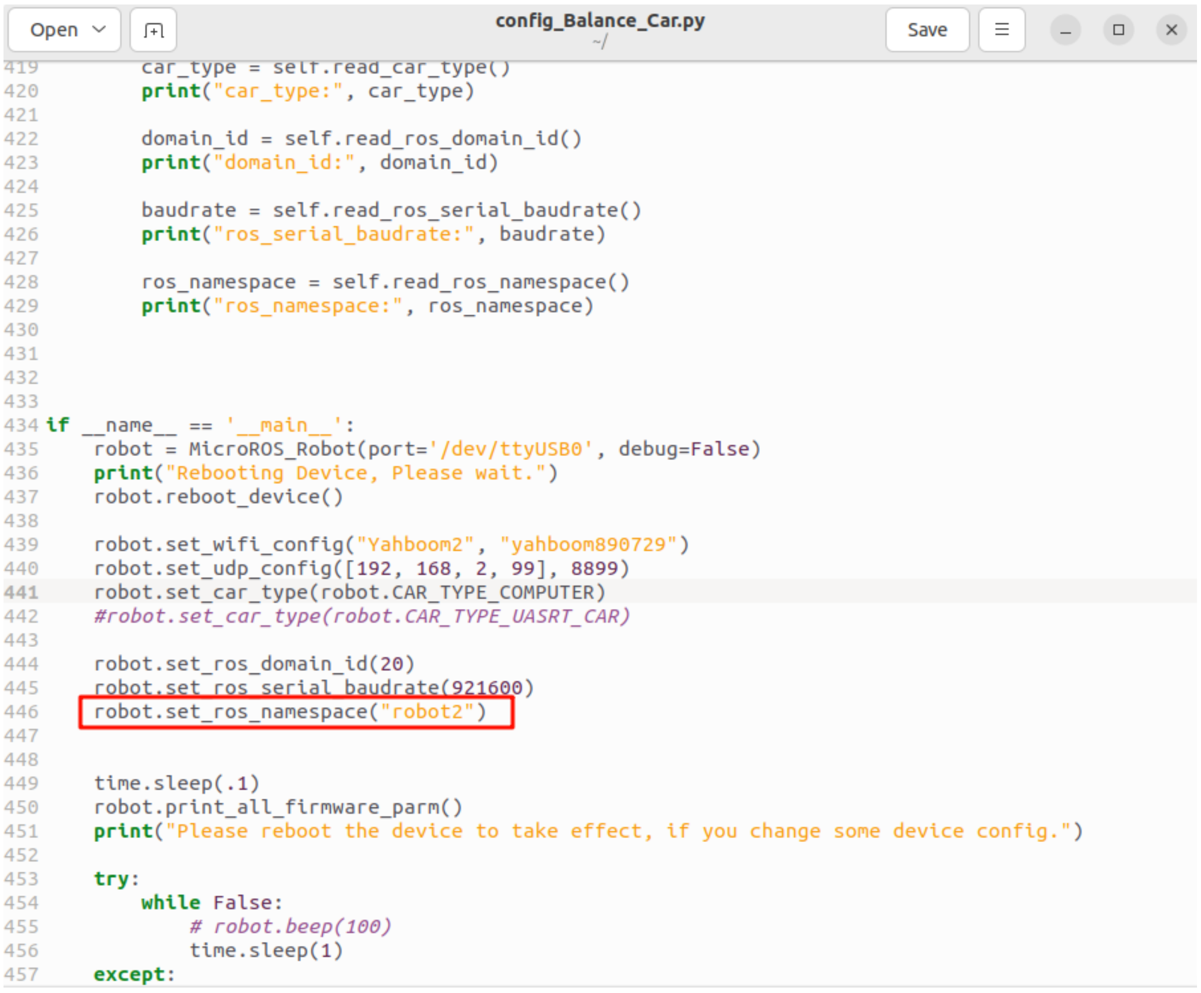
Note: The virtual machine needs to be in the same LAN as the car, and the ROS_DOMAIN_ID needs to be consistent. You can check [Must-read before use] to set the IP and ROS_DOMAIN_ID on the board.

1. Program function description

After the program is started, the handle can be used to control the synchronous movement of the two cars. The handle program here is based on the [ROS robot USB wireless handle] sold by Yabo Intelligent Technology. Other handle programs may not be compatible and need to be modified according to the actual remote control key value.

2. Basic settings for multi-machine functions

Taking two cars as an example, it is recommended to use two computers with matching virtual machines, change the config_robot.py files respectively, set robot.set_ros_namespace() to robot1 and robot2 respectively; set robot.set_udp_config() to the IP addresses of the two virtual machines respectively, and **the ROS_DOMAIN_ID of the two cars and the ROS_DOMAIN_ID of the virtual machine need to be set to the same**. Then open the terminal in the /home/yahboom directory and enter `sudo python3 config_Balance_Car.py` to run this program (you need to change the rest of the programs other than running multiple cars and rerun this program).

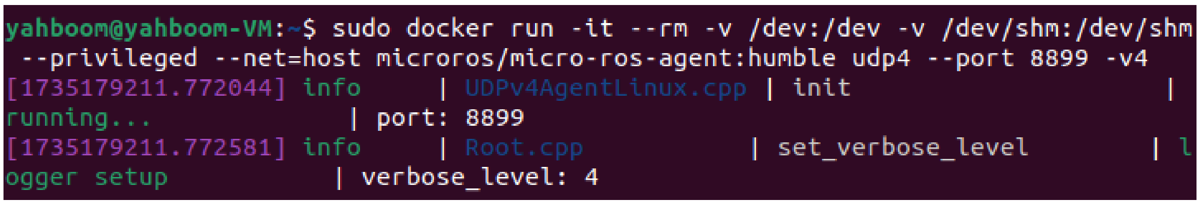


```
        car_type = self.read_car_type()
        print("car_type:", car_type)

        domain_id = self.read_ros_domain_id()
        print("domain_id:", domain_id)

        baudrate = self.read_ros_serial_baudrate()
        print("ros_serial_baudrate:", baudrate)

        ros_namespace = self.read_ros_namespace()
        print("ros_namespace:", ros_namespace)




if __name__ == '__main__':
    robot = MicroROS_Robot(port='/dev/ttyUSB0', debug=False)
    print("Rebooting Device, Please wait.")
    robot.reboot_device()

    robot.set_wifi_config("Yahboom2", "yahboom890729")
    robot.set_udp_config([192, 168, 2, 99], 8899)
    robot.set_car_type(robot.CAR_TYPE_COMPUTER)
    #robot.set_car_type(robot.CAR_TYPE_UASRT_CAR)

    robot.set_ros_domain_id(20)
    robot.set_ros_serial_baudrate(921600)
    robot.set_ros_namespace("robot2")


    time.sleep(.1)
    robot.print_all_firmware_parm()
    print("Please reboot the device to take effect, if you change some device config.")

    try:
        while False:
            # robot.beep(100)
            time.sleep(1)
    except:
```

3. Start and connect the agent

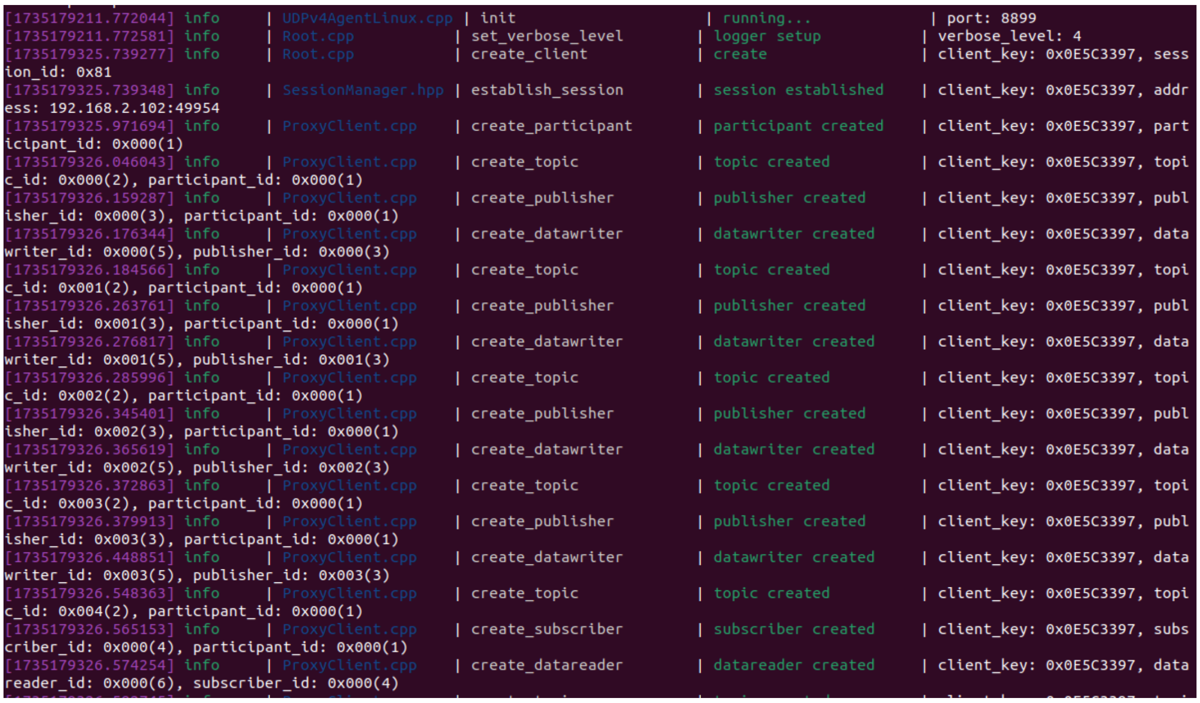
Take the matching virtual machine as an example. In the two virtual machines, enter the following command to start the agent of each car.

```
sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --
net=host microros/micro-ros-agent:humble udp4 --port 8899 -v4
```



```
yahboom@yahboom-VM:~$ sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm
 --privileged --net=host microros/micro-ros-agent:humble udp4 --port 8899 -v4
[1735179211.772044] info     | UDPv4AgentLinux.cpp | init                     |
running...             | port: 8899
[1735179211.772581] info     | Root.cpp            | set_verbose_level        | l
ogger setup              | verbose_level: 4
```

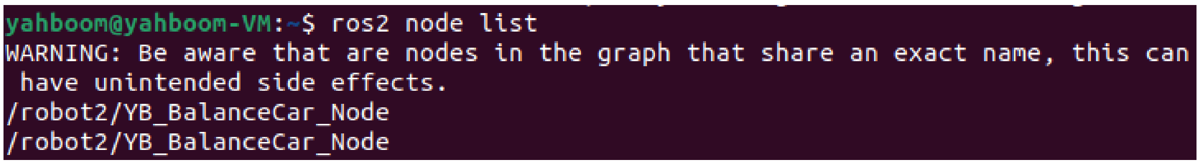
Then, turn on the switches of the two cars and wait for the two cars to connect to their respective agents. If the connection is successful, the terminal display is as shown in the figure below.



```
[1735179211.772044] info     | UDPv4AgentLinux.cpp | init                     | running...             | port: 8899
[1735179211.772581] info     | Root.cpp            | set_verbose_level        | logger setup           | verbose_level: 4
[1735179325.739277] info     | Root.cpp            | create_client            | create                 | client_key: 0x0E5C3397, sess
ion_id: 0x81
[1735179325.739348] info     | SessionManager.hpp | establish_session         | session established    | client_key: 0x0E5C3397, addr
ess: 192.168.2.102:49954
[1735179325.971694] info     | ProxyClient.cpp     | create_participant       | participant created    | client_key: 0x0E5C3397, part
icipant_id: 0x000(1)
[1735179326.046043] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x000(2), participant_id: 0x000(1)
[1735179326.159287] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0E5C3397, publ
isher_id: 0x000(3), participant_id: 0x000(1)
[1735179326.176344] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0E5C3397, data
writer_id: 0x000(5), publisher_id: 0x000(3)
[1735179326.184566] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x001(2), participant_id: 0x000(1)
[1735179326.263761] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0E5C3397, publ
isher_id: 0x001(3), participant_id: 0x000(1)
[1735179326.276817] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0E5C3397, data
writer_id: 0x001(5), publisher_id: 0x001(3)
[1735179326.285996] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x002(2), participant_id: 0x000(1)
[1735179326.345401] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0E5C3397, publ
isher_id: 0x002(3), participant_id: 0x000(1)
[1735179326.365619] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0E5C3397, data
writer_id: 0x002(5), publisher_id: 0x002(3)
[1735179326.372863] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x003(2), participant_id: 0x000(1)
[1735179326.379913] info     | ProxyClient.cpp     | create_publisher         | publisher created      | client_key: 0x0E5C3397, publ
isher_id: 0x003(3), participant_id: 0x000(1)
[1735179326.448851] info     | ProxyClient.cpp     | create_datawriter        | datawriter created     | client_key: 0x0E5C3397, data
writer_id: 0x003(5), publisher_id: 0x003(3)
[1735179326.548363] info     | ProxyClient.cpp     | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x004(2), participant_id: 0x000(1)
[1735179326.565153] info     | ProxyClient.cpp     | create_subscriber        | subscriber created     | client_key: 0x0E5C3397, subs
criber_id: 0x000(4), participant_id: 0x000(1)
[1735179326.574254] info     | ProxyClient.cpp     | create_datareader        | datareader created     | client_key: 0x0E5C3397, data
reader_id: 0x000(6), subscriber_id: 0x000(4)
```

Check the currently started node. In the two virtual machines, randomly select one and open the terminal to enter,

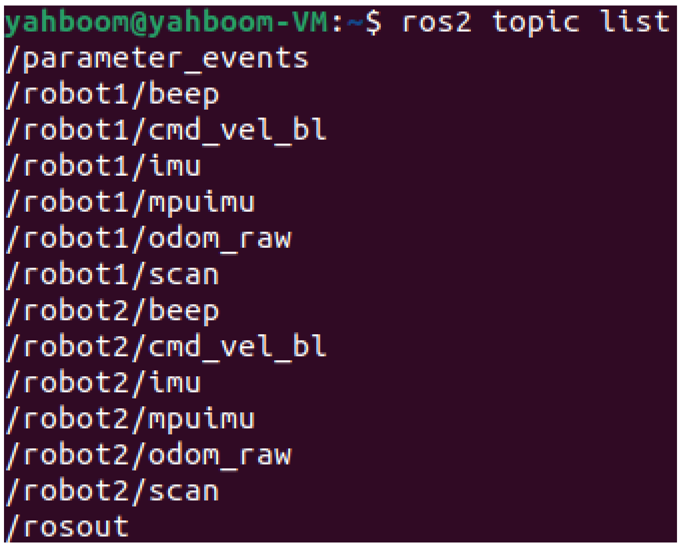
```
ros2 node list
```



```
yahboom@yahboom-VM:~$ ros2 node list
WARNING: Be aware that are nodes in the graph that share an exact name, this can
 have unintended side effects.
/robot2/YB_BalanceCar_Node
/robot2/YB_BalanceCar_Node
```

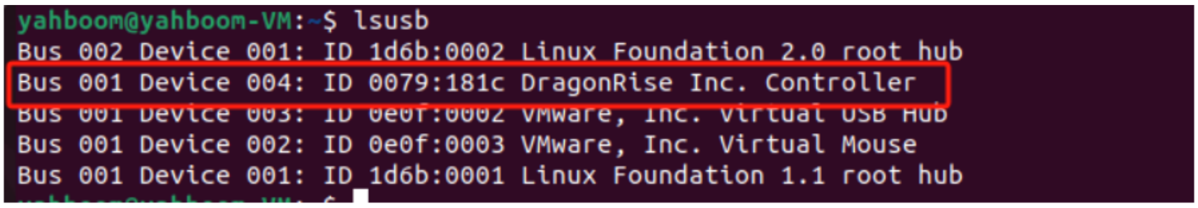
As shown in the figure above, the nodes of both cars have been started. Check the current topic information, input in the terminal,

```
ros2 topic list
```


```
yahboom@yahboom-VM:~$ ros2 topic list
/parameter_events
/robot1/beep
/robot1/cmd_vel_bl
/robot1/imu
/robot1/mpuimu
/robot1/odom_raw
/robot1/scan
/robot2/beep
/robot2/cmd_vel_bl
/robot2/imu
/robot2/mpuimu
/robot2/odom_raw
/robot2/scan
/rosout
```

3. Start the controller control program

Connect the controller receiver to any virtual machine. You need to ensure that the virtual machine can recognize the controller receiver. If it is connected as shown in the figure below, it means it is connected.


```
yahboom@yahboom-VM:~$ lsusb
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 004: ID 0079:181c DragonRise Inc. Controller
Bus 001 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 001 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

If it is not connected, check [Virtual Machine] -> [Removable Devices] in the menu bar on the virtual machine toolbar, and check whether [DragonRise Controller] is checked.

3.1 Run command

In the virtual machine of the connected handle receiver, take the matching virtual machine as an example, input in the terminal,

```
ros2 launch yahboomcar_multi yahboomcar_multi_joy_launch.py
```

Observe the handle indicator light, if it is always on, it means the connection is successful. After the program starts, press [START], and the buzzers of the two cars will sound at the same time. Press the R1 key, and the terminal prints the following information, turning on the handle control. You can use the left joystick up and down to control the two cars to move forward and backward at the same time; you can use the right joystick left and right to control the two cars to turn left and right at the same time;

```
[yahboom_joy-2] [INFO] [1735197789.731509428] [joy_ctrl]: Play:true
```

The remote control button description is as follows:

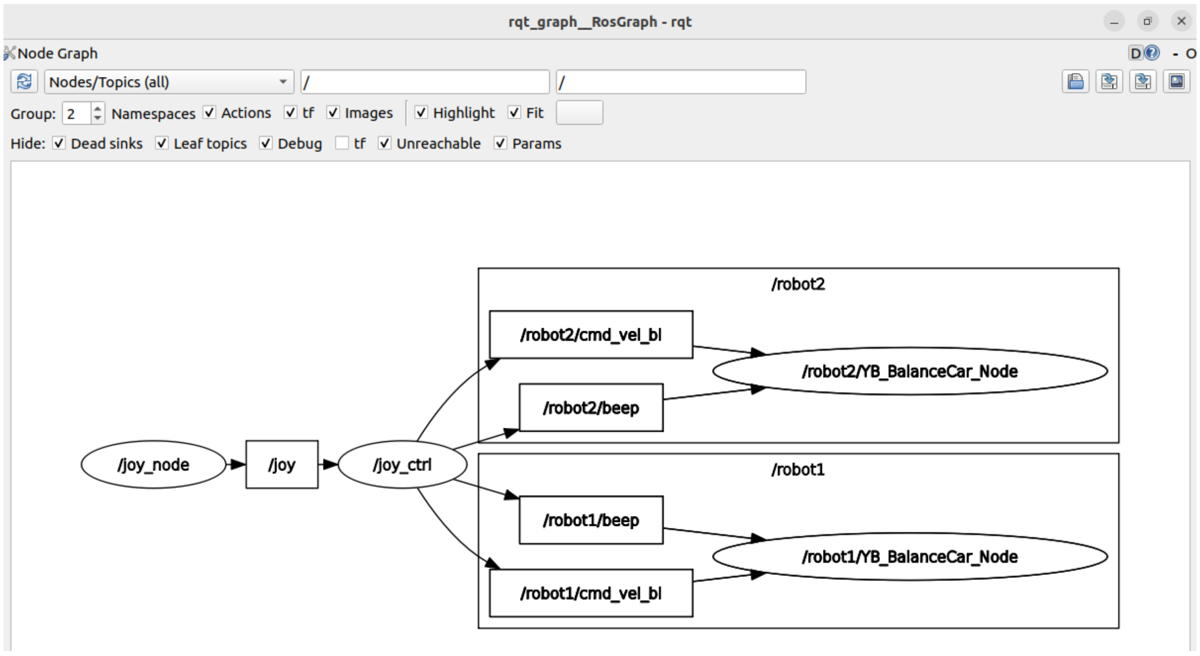
- Left joystick: front and back directions are valid, controlling the car to move forward and backward, and left and right directions are invalid
- Right joystick: left and right directions are valid, controlling the car to turn left and right, and front and back directions are invalid

- START button: buzzer control
- R1 button: handle control speed switch, press it to control the car speed with the remote control
- MODE button: switch mode, use the default mode, after switching modes, if the key value is incorrect, the program will report an error and exit.

4. View the node communication graph

Select one of the two virtual machines at random and enter in the terminal,

```
ros2 run rqt_graph rqt_graph
```



If it is not displayed at the beginning, select [Nodes/Topics(all)] and click the refresh button in the upper left corner.

5. Source code analysis

Source code reference path (taking the supporting virtual machine as an example):

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

multi_yahboom_joy.py,

```
#Create topic publisher
#create pub
self.pub_goal = self.create_publisher(GoalID,"move_base/cancel",10)
self.pub_JoyState = self.create_publisher(Bool,"JoyState", 10)
#cmd_vel
self.pub_cmdVel_r1 = self.create_publisher(Twist,'/robot1/cmd_vel_bl',10)
self.pub_cmdVel_r2 = self.create_publisher(Twist,'/robot2/cmd_vel_bl',10)
self.pub_cmdVel_r3 = self.create_publisher(Twist,'/robot3/cmd_vel_bl',10)
#beep
self.pub_Buzzer_r1 = self.create_publisher(UInt16,"/robot1/beep", 1)
self.pub_Buzzer_r2 = self.create_publisher(UInt16,"/robot2/beep", 1)
self.pub_Buzzer_r3 = self.create_publisher(UInt16,"/robot3/beep", 1)
```

```
#Create topic subscriber, subscribe to /joy node information
self.sub_Joy = self.create_subscription(Joy,'joy', self.buttonCallback,10)
#Callback function
def buttonCallback(self,joy_data):
if not isinstance(joy_data, Joy): return
self.user_jetson(joy_data)
#Process remote control key values, for detailed code, refer to user_jetson
function
#Publish speed topic
self.pub_cmdVel_r1.publish(twist)
self.pub_cmdVel_r2.publish(twist)
#Publish buzzer topic
self.pub_Buzzer_r1.publish(b)
self.pub_Buzzer_r2.publish(b)
```