

Robot handle control

Note: The ROS_DOMAIN_ID of the Raspberry Pi and the microROS control board need to be consistent. You can check [MicroROS Control Board Parameter Configuration] to set the microROS control board ROS_DOMAIN_ID. Check the tutorial [Connect MicroROS Agent] to determine whether the IDs are consistent.

1. Program function description

The car is connected to the agent, the receiver of the handle is connected to the Raspberry Pi 5 port, and the program is run, and the car movement, buzzer and gimbal servo can be controlled by remote control.

2. Start and connect to the agent

After successfully starting the Raspberry Pi, enter the following command to start the agent.

```
sh ~/start_agent_rpi5.sh
```

```
pi@raspberrypi:~ $ sh ~/start_agent_rpi5.sh
[1705977423.274347] info      | TermiosAgentLinux.cpp | init
| running...              | fd: 3
[1705977423.274654] info      | Root.cpp               | set_verbose_level      | 1
logger setup              | verbose_level: 4
```

Then, turn on the car switch and wait for the car to connect to the agent. The connection is successful, as shown in the figure below.

```
Subscriber created        | client_key: 0x57E5DE1D, subscriber_id: 0x001(4), partici
pant_id: 0x000(1)
[1705977460.524081] info      | ProxyClient.cpp        | create_datareader      | d
atareader created        | client_key: 0x57E5DE1D, datareader_id: 0x001(6), subscri
ber_id: 0x001(4)
[1705977460.530288] info      | ProxyClient.cpp        | create_topic           | t
opic created             | client_key: 0x57E5DE1D, topic_id: 0x005(2), participant_
id: 0x000(1)
[1705977460.533849] info      | ProxyClient.cpp        | create_subscriber      | s
subscriber created        | client_key: 0x57E5DE1D, subscriber_id: 0x002(4), partici
pant_id: 0x000(1)
[1705977460.540811] info      | ProxyClient.cpp        | create_datareader      | d
atareader created        | client_key: 0x57E5DE1D, datareader_id: 0x002(6), subscri
ber_id: 0x002(4)
[1705977460.548331] info      | ProxyClient.cpp        | create_topic           | t
topic created             | client_key: 0x57E5DE1D, topic_id: 0x006(2), participant_
id: 0x000(1)
[1705977460.553389] info      | ProxyClient.cpp        | create_subscriber      | s
subscriber created        | client_key: 0x57E5DE1D, subscriber_id: 0x003(4), partici
pant_id: 0x000(1)
[1705977460.556688] info      | ProxyClient.cpp        | create_datareader      | d
atareader created        | client_key: 0x57E5DE1D, datareader_id: 0x003(6), subscri
ber_id: 0x003(4)
```

3. Enter the car docker

Open another terminal and enter the following command to enter docker.

```
sh ros2_humble.sh
```

When the following interface appears, you have successfully entered docker. Now you can control the car through commands.

```
pi@raspberrypi:~ $ ./ros2_humble.sh
access control disabled, clients can connect from any host
MY_DOMAIN_ID: 20
root@raspberrypi:/#
```

Enter the following command in the terminal to check whether the controller is mounted normally.

```
ls /dev/input*
```

When this interface appears, it means it was successfully mounted.

```
root@raspberrypi:/# ls /dev/input*
by-id      event0  event2  event4  event6  mice
by-path    event1  event3  event5  js0
```

4. Start the handle control program

Because the handle control function has been automatically activated at boot, if we need to manually enable handle control, we need to turn off the auto-start process at boot.

Terminal input command:

```
ps -A
```

```
root@raspberrypi:/# ps -A
  PID TTY          TIME CMD
    1 pts/0        00:00:00 bash
    7 pts/0        00:00:00 bash
   68 ?            00:00:00 supervisord
  125 ?            00:00:00 sh
  126 ?            00:00:00 bash
  148 ?            00:00:00 python3
  170 ?            00:00:00 ros2
  171 ?            00:00:00 yahboom_joy
  173 ?            00:00:00 joy_node
  198 pts/0        00:00:00 ps
```

You can see that the processes of the handle control node and the handle node are 171 and 173. Turn off these two processes to manually start the handle control.

```
kill -9 171
kill -9 173
```

Manually open the handle control program.

```
#Handle remote control car program
ros2 launch yahboomcar_ctrl yahboom_joy_launch.py
```

```
root@raspberrypi:/# ros2 launch yahboomcar_ctrl yahboomcar_joy_launch.py
[INFO] [launch]: All log files can be found below /root/.ros/log/2024-01-23-03-39-41-513549-raspberrypi-226
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [yahboom_joy-1]: process started with pid [227]
[INFO] [joy_node-2]: process started with pid [229]
[joy_node-2] [INFO] [1705981181.895153074] [joy_node]: No haptic (rumble) available, skipping initialization
[joy_node-2] [INFO] [1705981181.895350000] [joy_node]: Opened joystick: Controller. deadzone: 0.050000
```

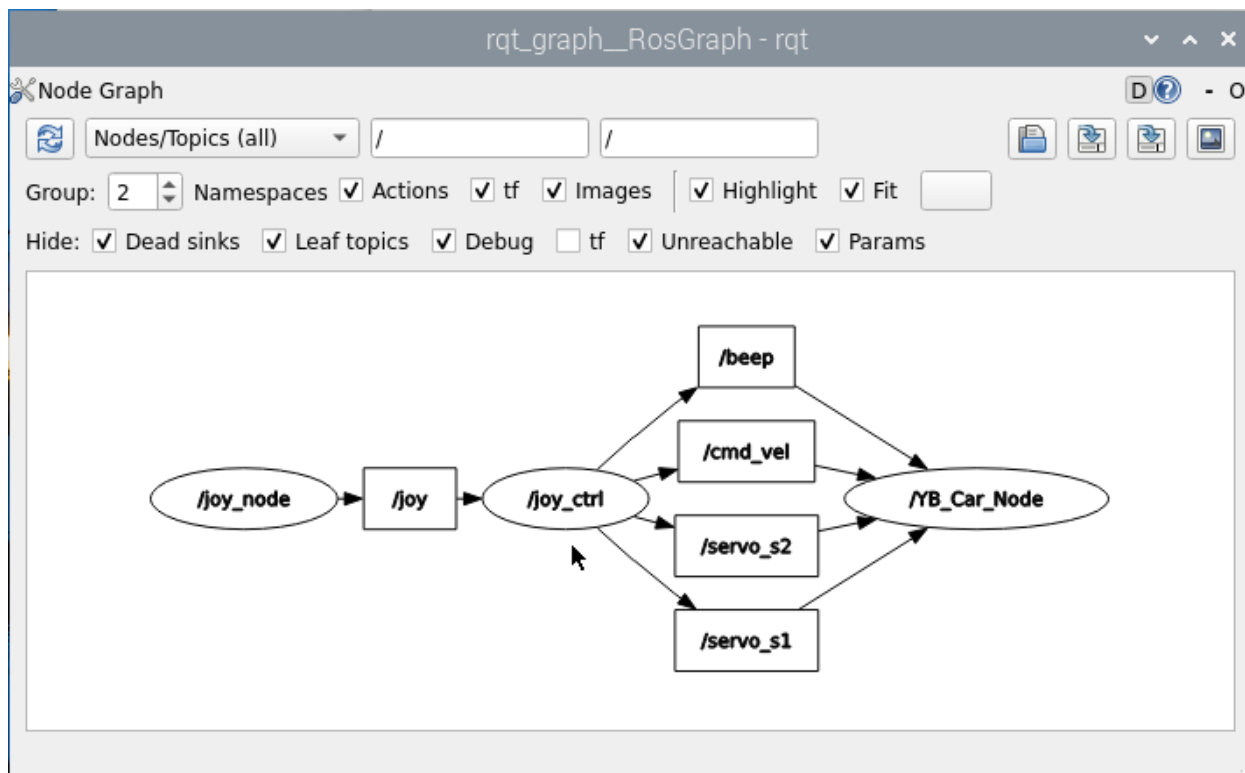
The remote control button description is as follows:

- Left rocker: valid in the front and rear directions, controls the car forward and backward, invalid in the left and right directions
- Right joystick: valid in left and right directions, controls the car to rotate left and right, invalid in forward and backward directions
- START key: buzzer control
- Y key: control the S2 servo upward
- A key: control the S2 servo down
- X key: control the S1 servo to the left
- B key: control the S1 servo to the right
- R1 key: handle control speed switch. After pressing it, you can use the remote control to control the speed of the car. Press it again to lose the handle control speed. It is also a gameplay switch. Press it to stop. Press it again to continue running the functional gameplay program, including radar. Obstacle avoidance, radar guard and more.
- MODE key: Switch modes and use the default mode. After switching modes, if the key value is incorrect, the program will exit with an error.

5、 Node communication diagram

Enter the following command in the terminal.

```
ros2 run rqt_graph rqt_graph
```



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

6、 Code analysis

Source code reference path.

```
/root/yahboomcar_ws/src/yahboomcar_ctrl/launch
```

yahboom_joy_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    node1 = Node(
        package='yahboomcar_ctrl',
        executable='yahboom_joy',
    )
    node2 = Node(
        package='joy',
        executable='joy_node',
    )
    launch_description = LaunchDescription([node1,node2])
    return launch_description
```

Two nodes are running, namely, the yahboom_ctrl handle control node and the joy_node handle key value reading node.

Two node source code reference paths.

```
/root/yahboomcar_ws/src/yahboomcar_ctrl/yahboomcar_ctrl
```

yahboom_joy.py

```
#Create subscriber subscription/joy topic data
self.sub_joy = self.create_subscription(Joy, 'joy', self.buttonCallback, 10)

#Create a publisher to publish speed, servo, buzzer topic data and function gameplay
switches
self.pub_cmdVel = self.create_publisher(Twist, 'cmd_vel', 10)
self.pub_Buzzer = self.create_publisher(UInt16, "beep", 1)
self.pub_JoyState = self.create_publisher(Bool, "JoyState", 10)
self.pub_Servo1 = self.create_publisher(Int32, "servo_s1", 10)
self.pub_Servo2 = self.create_publisher(Int32, "servo_s2", 10)

#Callback
def buttonCallback(self, joy_data):
    if not isinstance(joy_data, Joy): return
    self.user_jetson(joy_data)

#Process key function user_jetson
def user_jetson(self, joy_data):
    #cancel nav
    if joy_data.buttons[7] == 1: self.cancel_nav()
    #Buzzer
    if joy_data.buttons[11] == 1:
        b = UInt16()
        self.Buzzer_active = not self.Buzzer_active
        b.data = self.Buzzer_active
        self.pub_Buzzer.publish(b)
    #Gear adjustment, press the rocker to adjust the gear
    xlinear_speed = self.filter_data(joy_data.axes[1]) * self.xspeed_limit *
self.linear_Gear
    angular_speed = self.filter_data(joy_data.axes[2]) * self.angular_speed_limit *
self.angular_Gear
    if xlinear_speed > self.xspeed_limit: xlinear_speed = self.xspeed_limit
    elif xlinear_speed < -self.xspeed_limit: xlinear_speed = -self.xspeed_limit
    if angular_speed > self.angular_speed_limit: angular_speed =
self.angular_speed_limit
    elif angular_speed < -self.angular_speed_limit: angular_speed = -
self.angular_speed_limit
    twist = Twist()
    twist.linear.x = xlinear_speed
    twist.linear.y = 0.0
    twist.angular.z = angular_speed
    #Determine whether the speed can be controlled, that is, whether the R1 key is
    pressed
    if self.Joy_active == True:
        self.pub_cmdVel.publish(twist)
```

#The following is the data for processing the servo control. After pressing it, the angle will increase/decrease by 1. If you continue to press it, the angle will continue to increase and decrease.

```
if not joy_data.buttons[1] == 0:
    print("Up")
    self.PWMServo_X += 1
    if self.PWMServo_X <= -90: self.PWMServo_X = -90
    elif self.PWMServo_X >= 90: self.PWMServo_X = 90
    print("self.PWMServo_X: ",self.PWMServo_X)
    print("self.PWMServo_Y: ",self.PWMServo_Y)
    servo1_angle = Int32()
    servo1_angle.data = self.PWMServo_X
    self.pub_Servo1.publish(servo1_angle)

if not joy_data.buttons[3] == 0:
    print("Down")
    self.PWMServo_X -= 1
    if self.PWMServo_X <= -90: self.PWMServo_X = -90
    elif self.PWMServo_X >= 90: self.PWMServo_X = 90
    print("self.PWMServo_X: ",self.PWMServo_X)
    print("self.PWMServo_Y: ",self.PWMServo_Y)
    servo1_angle = Int32()
    servo1_angle.data = self.PWMServo_X
    self.pub_Servo1.publish(servo1_angle)

if not joy_data.buttons[0] == 0:
    print("Left")
    self.PWMServo_Y -= 1
    if self.PWMServo_Y <= -90: self.PWMServo_Y = -90
    elif self.PWMServo_Y >= 20: self.PWMServo_Y = 20
    servo2_angle = Int32()
    servo2_angle.data = self.PWMServo_Y
    self.pub_Servo2.publish(servo2_angle)
    print("self.PWMServo_X: ",self.PWMServo_X)
    print("self.PWMServo_Y: ",self.PWMServo_Y)

if not joy_data.buttons[4] == 0:
    print("Right")
    self.PWMServo_Y += 1
    if self.PWMServo_Y <= -90: self.PWMServo_Y = -90
    elif self.PWMServo_Y >= 20: self.PWMServo_Y = 20
    servo2_angle = Int32()
    servo2_angle.data = self.PWMServo_Y
    self.pub_Servo2.publish(servo2_angle)
    print("self.PWMServo_X: ",self.PWMServo_X)
    print("self.PWMServo_Y: ",self.PWMServo_Y)
```

7. Variables corresponding to remote control key values

Based on the default mode [Controller], the key values corresponding to the remote control are as follows.

remote control event	Corresponding variable
Left joystick up	axes[1]=1
Left joystick down	axes[1]=-1
Right joystick to left	axes[2]=1
Right joystick to the right	axes[2]=-1
Button X pressed	button[3]=1
ButtonB pressed	button[1]=1
Button Y pressed	button[4]=1
Button R1 pressed	button[7]=1
Button start pressed	button[11]=1
Left joystick pressed	button[13]=1
Right joystick pressed	button[14]=1

Combined with the above source code for easy understanding, when these values change, it means that the remote control is pressed and the corresponding program can be executed. To view other buttons, you can subscribe to the /joy topic. Enter the following command in the terminal.

```
ros2 topic echo /joy
```

```
header:
  stamp:
    sec: 1705982833
    nanosec: 926566533
  frame_id: joy
```

```
axes:
```

```
- -0.0
- -0.0
- -0.0
- -0.0
- 1.0
- 1.0
- 0.0
- 0.0
```

```
buttons:
```

```
- 0
- 0
- 0
- 0
- 0
- 0
- 0
- 0
- 0
```

I