

Handle control

Handle control

1、Operating environment

2、Install driver

3、Steps for usage

4、Handle control turtle

4.1、Start the turtle

4.2、View node

4.3、View the node information

4.4、Handle control turtle

5、Handle control Transbot

5.1、handle control node

5.2、Associated node

5.3、Operating procedures

5.4、Source code analysis

5.5、Handle control effect

Appendix

jetson nano

Raspberry Pi

Precautions for using the handle

1、Operating environment

Operating system: Ubuntu 18.04 LTS

ROS version: melodic

Device: jetson nano、Raspberry Pi、PC、handle (USB receiver)

Handle control function package path: ~/transbot_ws/src/transbot_ctrl

2、Install driver

ROS driver for general Linux handles. The Joy package contains Joy_node, which is a node that connects a general Linux controller to ROS. The node publishes a "/Joy" message, which contains the current state of each button and axis of the handle.

```
sudo apt install ros-melodic-joy ros-melodic-joystick-drivers
```

3、Steps for usage

- Device connection

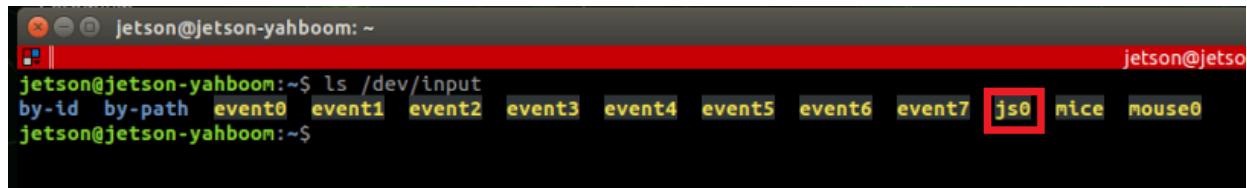
First of all, we need to insert the USB adapter of the wireless controller into Jetson NANO, Raspberry Pi, and PC.

In this lesson, we will insert the USB port of the wireless controller to Jetson NANO board.

- View device

Open the terminal, enter the following command, it shows [js0], this is the wireless controller.

```
ls /dev/input
```



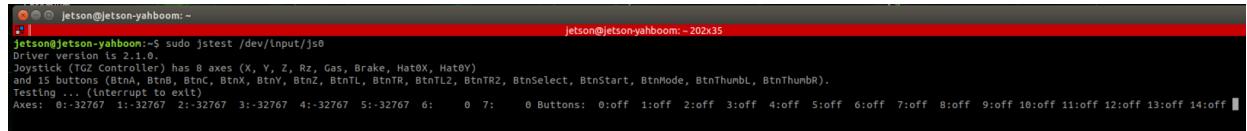
```
jetson@jetson-yahboom:~$ ls /dev/input
by-id by-path event0 event1 event2 event3 event4 event5 event6 event7 js0 mice mouse0
jetson@jetson-yahboom:~$
```

- Test handle

Open the terminal and input the following commands.

As shown in the figure, the wireless handle has 8 axial inputs and 15 key inputs. You can press the keys to test the numbers corresponding to the keys on handle.

```
sudo jstest /dev/input/js0
```



```
jetson@jetson-yahboom:~$ sudo jstest /dev/input/js0
jetson@jetson-yahboom:~$ 202x35
Driver version is 2.1.0.
Joystick (TGZ Controller) has 8 axes (X, Y, Z, Rz, Gas, Brake, Hat0X, Hat0Y)
and 15 buttons (BtnA, BtnB, BtnC, BtnX, BtnY, BtnZ, BtnTL, BtnTR, BtnTL2, BtnTR2, BtnSelect, BtnStart, BtnMode, BtnThumBL, BtnThumBR).
Testing ... (Interrupt to exit)
Axes:  0:-32767  1:-32767  2:-32767  3:-32767  4:-32767  5:-32767  6:   0  7:   0 Buttons: 0:off 1:off 2:off 3:off 4:off 5:off 6:off 7:off 8:off 9:off 10:off 11:off 12:off 13:off 14:off
```

If jstest is not installed, you need run the following command to install it.

```
sudo apt-get install joystick
```

- Run the controller node

Open three terminals, enter the following commands in sequence to view the detailed information, which is the same as [Test Handle].

Different devices (Raspberry Pi, Jetson NANO, PC) and different systems have different handle states.

```
roscore Step.1
rosrun joy joy_node Step.3
rostopic echo joy Step.3
```

```

jetson@jetson-yahboom:~$ roscore
... logging to /home/jetson/.ros/log/8a117abe-0637-11ec-a654-18cc1
89d2896/roslaunch-jetson-yahboom-2218.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.2.91:35151/
ros_comm version 1.14.11

SUMMARY
========
PARAMETERS
  * /rosdistro: melodic

jetson@jetson-yahboom:~$ rosrun joy joy_node
[ WARN] [1629959656.647290938]: Couldn't set gain on joystick force feedback: Bad file descriptor
[ INFO] [1629959656.657392553]: Opened joystick: /dev/input/js0. deadzone_: 0.050000.

jetson@jetson-yahboom:~$ rostopic echo joy
header: # 标题
  seq: 1
  stamp:
    secs: 1629959689
    nsecs: 120446026
  frame_id: "/dev/input/js0" # 设备ID
  axes: [0.0, 0.0, 0.0, 0.2186940312385559, 0.0, 0.0, 0.0, 0.0]
  buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
...
header: # 按键输入
  seq: 2
  stamp:
    secs: 1629959689
    nsecs: 128423930
  frame_id: "/dev/input/js0"
  axes: [0.0, 0.0, 0.0, 0.6636217832565308, 0.0, 0.0, 0.0, 0.0]
  buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
...
header:
  seq: 3
  stamp:
    secs: 1629959689
    nsecs: 144449487
  frame_id: "/dev/input/js0"
  axes: [0.0, 0.0, 0.0, 0.8915147185325623, 0.0, 0.0, 0.0, 0.0]
  buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
...
header:
  seq: 4
  stamp:
    secs: 1629959689
    nsecs: 152497037
  frame_id: "/dev/input/js0"

```

4、Handle control turtle

4.1、Start the turtle

```

roscore
rosrun turtlesim turtlesim_node

```

If you want to drive the little turtle, just give the little turtle a certain speed. Next, check the ROS speed control node.

```

jetson@jetson-yahboom:~$ roscore
... logging to /home/jetson/.ros/log/e8337c1e-0637-11ec-8ec8-18cc189b2896/roslaunch-jetson-yahboom-15142.log
Checking log directory for disk usage. This may take a
TurtleSim

jetson@jetson-yahboom:~$ rosrun turtlesim turtlesim_node
[ INFO] [1629960103.561638823]: Starting turtlesim with node name /turtlesim
[ INFO] [1629960103.600368275]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]

jetson@jetson-yahboom:~$ rostopic list
/rosvout
/rosvout_agg
/turtle1/cmd_vel      # 速度话题
/turtle1/color_sensor # 颜色话题
/turtle1/pose          # 位姿话题
jetson@jetson-yahboom:~$ 
jetson@jetson-yahboom:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist # 话题类型
Publishers: None           # 发布者 (无)
Subscribers:               # 订阅者
  * /turtlesim (http://192.168.2.91:38981/) # 订阅者信息
    (小乌龟节点)
jetson@jetson-yahboom:~$ 

```

4.2、View node

```
rostopic list
```

4.3、View the node information

```
rostopic info /turtle1/cmd_vel
```

4.4、Handle control turtle

Receive the current status information of the handle, and send instructions to the little turtle by pressing the button or shaking the joystick to receive different information feedback from the wireless handle.

Copy the wireless controller control function package to the workspace, compile and update the environment.

```
catkin_make # Compile  
source devel/setup.bash # Update environment  
Note: As long as you modify the C++ code, you must recompile the update to take effect.
```

Input command to start up python code

```
roslaunch transbot_ctrl turtlesim_joy.launch
```

Input command to start up C++ code

```
roslaunch transbot_ctrl turtlesim_turtle.launch
```

```

/home/jetson/transbot_ws/src/transbot_ctrl/launch/turtlesim_joy.launch http://192.168.2.91:11311
/home/jetson/transbot_ws/src/transbot_ctrl/launch/turtlesim_joy.launch http://192.168.2.91:11311 80x24
jetson@jetson-yahboom:~$ rosrun transbot_ctrl turtlesim_joy.launch
... logging to /home/jetson/.ros/log/d93f08b2-0638-11ec-8b08-18cc189b2896/roslaunch-jetson-yahboom-1387.log
Checking log directory for disk usage. This may take...
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.2.91:36411/
SUMMARY
========
PARAMETERS
  * /rosdistro: melodic
  * /rosversion: 1.14.11
  * /turtlesim_joy/angular_speed_limit: 2.0
  * /turtlesim_joy/linear_speed_limit: 2.0
  * /use_sim_time: False

NODES
/
  joy_node (joy/joy_node)
  turtlesim_joy (transbot_ctrl/turtlesim_joy.py)
  turtlesim_node (turtlesim/turtlesim_node)

ROS_MASTER_URI=http://192.168.2.91:11311

process[turtlesim_node-1]: started with pid [1653]

```

The terminal window shows the output of running the `turtlesim_joy.launch` file. It starts by launching the `turtlesim_joy` node from the `transbot_ctrl` package. The log indicates that it's using ROS version 1.14.11 and the melodic distribution. It lists the parameters set: angular speed limit at 2.0 and linear speed limit at 2.0, and specifies not to use simulated time. The nodes listed are the `joy_node`, `turtlesim_joy` node (which is the one running), and the `turtlesim_node`. The ROS master URI is set to `http://192.168.2.91:11311`. A process named `turtlesim_node-1` has been started with PID 1653.

At this point, we can use the handle to control the little turtle to run.

- Correspondence between the handle and the turtle

Handle	Turtle
Left rocker up	advance
Left rocker down	back
Left rocker left	turn left
Left rocker right	turn right

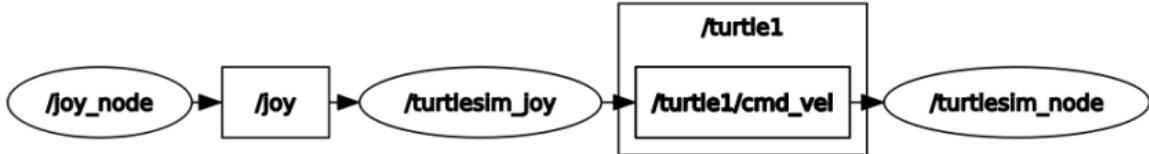
- View node graph

【/joy_node】 : Handle information node

【/turtlesim_joy】 : Handle control node

【/turtlesim_node】 : Turtle node

rqt_graph



5、Handle control Transbot

Handle controls Transbot, which is similar to the handle control of the turtle; let the handle control node establish a connection with the Transbot bottom driver node, change the current state of the handle, send different information to Transbot, and drive Transbot to make different responses. We need to control the buzzer, searchlight, pan-tilt, light bar, robotic arm, and trolley movement (linear velocity, angular velocity).

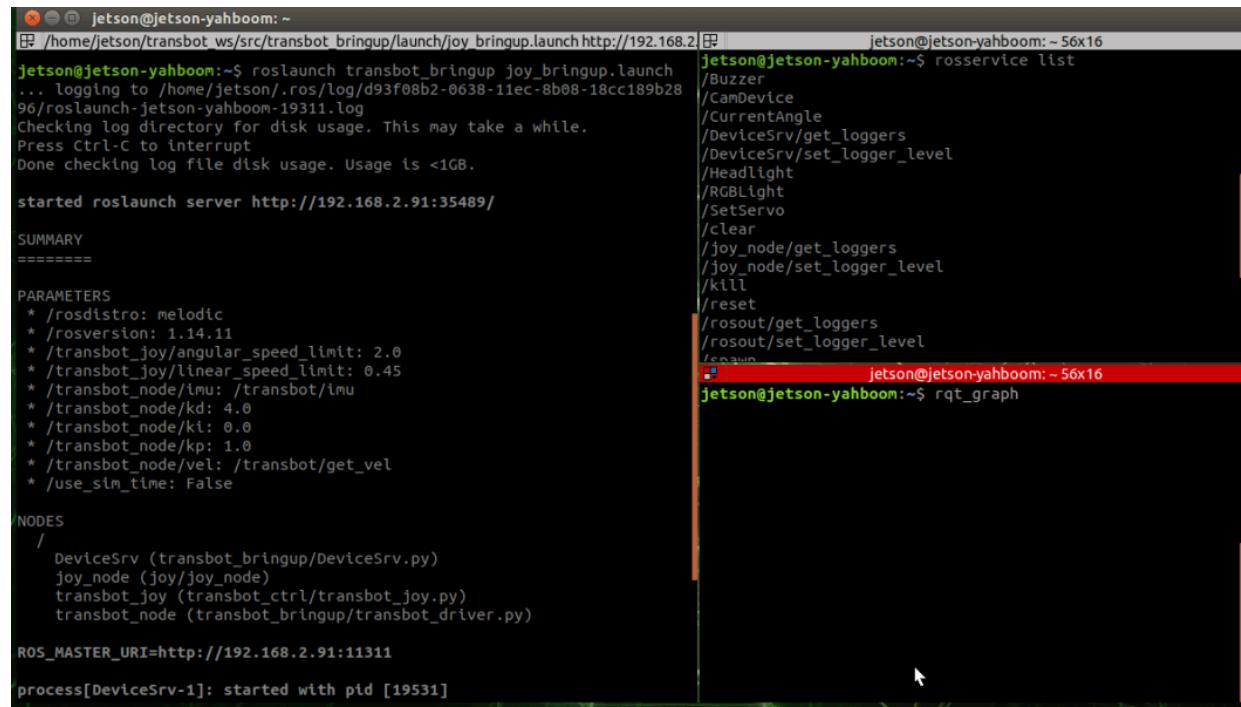
5.1、handle control node

- Corresponding to the MCU coprocessorof Transbot
 - Topic
 - Publish car movement news 【/cmd_vel】
 - Publish robotic arm control message 【/TargetAngle】
 - Publish gimbal servo control message 【/PWMServo】
 - Service (client)
 - Publish buzzer control message 【/Buzzer】
 - Publish searchlight control message 【/Headlight】
 - Publish obtain the current angle of the robotic arm 【/CurrentAngle】
 - Publish running water light control message 【/RGBLight】
 - Other topic
 - Joy_node corresponds
 - Subscribe to the news of the wireless controller 【/joy】
 - corresponding to move_base
 - Issue the cancel motion planning message 【/move_base/cancel】
 - Pause/start of all functions
 - Publish the current Joy state (custom) message 【/JoyState】
 - Node 【DeviceSrv】
 - Get the current camera device 【/CamDevice】
 - Joy_node corresponds

5.2、Associated node

- Low-level driver
 - Topic
 - Subscribe to car sports messages 【/cmd_vel】
 - Subscribe to robotic arm control messages 【/TargetAngle】
 - Subscribe to gimbal servo control messages 【/PWMServo】
 - Service (client)
 - Publish buzzer control message 【/Buzzer】
 - Publish searchlight control message 【/Headlight】
 - Publish obtain the current angle of the robotic arm 【/CurrentAngle】
 - Publish running water light control message 【/RGBLight】
- Joy_node
 - Topic
 - Publish the news of the wireless controller 【/joy】
- DeviceSrv
 - service
 - Enable access to the current camera equipment service 【/CamDevice】

5.3、Operating procedures



```
jetson@jetson-yahboom:~  
[ 1] /home/jetson/transbot_ws/src/transbot Bringup/launch/joy Bringup.launch http://192.168.2.91:35489 jetson@jetson-yahboom: ~ 56x16  
jetson@jetson-yahboom:~$ roslaunch transbot Bringup joy Bringup.launch ... logging to /home/jetson/.ros/log/d93f08b2-0638-11ec-8b08-18cc189b2896/roslaunch-jetson-yahboom-19311.log  
Checking log directory for disk usage. This may take a while.  
Press Ctrl-C to interrupt  
Done checking log file disk usage. Usage is <1GB.  
  
started roslaunch server http://192.168.2.91:35489/  
  
SUMMARY  
=====
```

PARAMETERS

- * /rosdistro: melodic
- * /rosversion: 1.14.11
- * /transbot_joy/angular_speed_limit: 2.0
- * /transbot_joy/llinear_speed_limit: 0.45
- * /transbot_node/imu: /transbot imu
- * /transbot_node/kd: 4.0
- * /transbot_node/kl: 0.0
- * /transbot_node/kp: 1.0
- * /transbot_node/vel: /transbot/get_vel
- * /use_stm_time: False

NODES

- /
- DeviceSrv (transbot Bringup/DeviceSrv.py)
- joy_node (joy/joy_node)
- transbot_joy (transbot_ctrl/transbot_joy.py)
- transbot_node (transbot Bringup/transbot_driver.py)

ROS_MASTER_URI=http://192.168.2.91:11311

process[DeviceSrv-1]: started with pid [19531]

```
jetson@jetson-yahboom:~$ rosservice list  
/Buzzer  
/CamDevice  
/CurrentAngle  
/DeviceSrv/get_loggers  
/DeviceSrv/set_logger_level  
/Headlight  
/RGBLight  
/SetServo  
/Clear  
/joy_node/get_loggers  
/joy_node/set_logger_level  
/Kill  
/Reset  
/Rosout/get_loggers  
/Rosout/set_logger_level  
/Run  
jetson@jetson-yahboom:~$ rqt_graph  
jetson@jetson-yahboom:~$
```

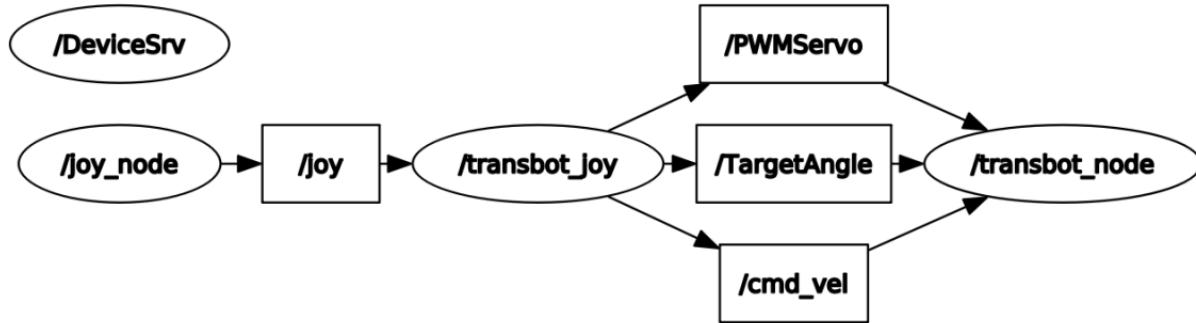
Start up command

```
roslaunch transbot Bringup joy Bringup.launch
```

View node graph

```
rqt_graph
```

Can only see the connections between topics



View service list

```
rosservice list
```

The system will print the following:

```
/Buzzer
/CamDevice
/CurrentAngle
/DeviceSrv/get_loggers
/DeviceSrv/set_logger_level
/Headlight
/RGBLight
/SetServo
/clear
/joy_node/get_loggers
/joy_node/set_logger_level
/kill
/reset
... ...
```

5.4、Source code analysis

- launch file

Code path: `~/transbot_ws/src/transbot_bringup/launch/joy_bringup.launch`

```
<launch>
    <!--Start the camera device service-->
    <node pkg="transbot_bringup" type="DeviceSrv.py" name="DeviceSrv"
output="screen"/>
    <!--Start Transbot bottom node-->
```

```

<node pkg="transbot_bringup" type="transbot_driver.py" name="transbot_node"
required="true" output="screen">
    <param name="imu" value="/transbot/imu"/>
    <param name="vel" value="/transbot/get_vel"/>
    <param name="kp" value="1.0"/>
    <param name="ki" value="0.0"/>
    <param name="kd" value="4.0"/>
</node>
<!--Start handle control node-->
<include file="$(find transbot_ctrl)/launch/transbot_joy.launch"/>
</launch>

```

Node 【DeviceSrv】 starts the camera device service: obtains the current camera device number, and publishes whether it is currently connected to the Astra camera.

- py file

Code path: ~/transbot_ws/src/transbot_ctrl/scripts/transbot_joy.py

```

arm_thread = threading.Thread(target=self.analyse_PWM())
arm_thread.setDaemon(True)
arm_thread.start()

```

Start the thread, get the currently connected camera device in the 【self.analyse_PWM()】 function, which is convenient for processing different cameras, and turn on the loop to control the gimbal servo and robotic arm.

```

def buttonCallback(self, joy_data):
    """
    Obtain the wireless controller receiving signal
    [1: Clamp, 2: Release, upper left: joint2+, lower left: joint2-, left left:
    joint1 upper, left and right joint1: lower]
    """

    if not isinstance(joy_data, Joy): return
    # rospy.loginfo("joy_data.buttons:", joy_data.buttons)
    # rospy.loginfo("joy_data.axes:", joy_data.axes)
    if self.user_name == "pi": self.user_pi(joy_data)
    else: self.user_jetson(joy_data)
    # rospy.loginfo("linear_Gear: {},angular_Gear:
    {}".format(self.linear_Gear,self.angular_Gear))
    #rospy.loginfo("linear_speed: {},angular_speed:
    {}".format(linear_speed,angular_speed))

```

5.5、Handle control effect



Appendix

jetson nano

```
joy_data.buttons:
header:
  seq: 335
  stamp:
    secs: 1628324636
    nsecs: 962988952
  frame_id: "/dev/input/js0"
axes: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

axes (8)

Code analysis	Handle button	Car control
axes[0]	Left rocker	
axes[1]	Left rocker	advance and back
axes[2]	Right rocker	left and right
axes[3]	Right rocker	
axes[4]		
axes[5]		

Code analysis	Handle button	Car control
axes[6]	Left button	Arm-1
axes[7]	Left button	Arm-2

buttons (15)

Code analysis	Handle button	Car control
buttons[0]	A	camera platform move down
buttons[1]	B	camera platform move right
buttons[2]		
buttons[3]	X	camera platform move left
buttons[4]	Y	camera platform move up
buttons[5]		
buttons[6]	L1	Arm-clip close
buttons[7]	R1	running water light
buttons[8]	L2	Arm-clip open
buttons[9]	R2	control switch
buttons[10]	SELECT	search light on high frame rate camera
buttons[11]	START	buzzer
buttons[12]		
buttons[13]	Press left rocker	Linear speed [0.15 , 0.3 , 0.45]
buttons[14]	Press right rocker	Angular speed [0.5 , 1 , 1.5 , 2]

Raspberry Pi

```
joy_data.buttons: header:
  seq: 264
  stamp:
    secs: 1628326479
    nsecs: 848359307
  frame_id: "/dev/input/js0"
axes: [-0.0, -0.0, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

axes (8)

Code analysis	Handle button	Car control
axes[0]	Left rocker	
axes[1]	Left rocker	advance and back
axes[2]	L2(press:-1, release:1)	Arm clip
axes[3]	Right rocker	left and right
axes[4]	Right rocker	
axes[5]	R2(press:-1, release:1)	control switch
axes[6]	Left button	Arm-1
axes[7]	Left button	Arm-2

buttons (11)

Code analysis	Handle button	Car control
buttons[0]	A	camera platform move down
buttons[1]	B	camera platform move right
buttons[2]	X	camera platform move left
buttons[3]	Y	camera platform move up
buttons[4]	L1	Arm clip
buttons[5]	R1	running water light
buttons[6]	SELECT	search light on high frame rate camera
buttons[7]	START	buzzer
buttons[8]	MODE	
buttons[9]	Press left rocker	Linear speed [0.15 , 0.3 , 0.45]
buttons[10]	Press right rocker	Angular speed [0.5 , 1 , 1.5 , 2]

Precautions for using the handle

- After plugging and unplugging the handle receiving head, the handle program needs to be restarted, otherwise the car will not be able to be controlled.
- After starting the handle control program, if the handle cannot control the car, it may be caused by the wrong handle control mode. You can press and hold the handle mode button for about 15 seconds to switch modes. After the green indicator light is always on, press the start button again. If the buzzer sounds, it means the switching is successful. If there is no response, you can press and hold the mode button on the handle again for 15 seconds.

Jetson series support mode: PC/PCS mode. In PC mode, the POWER MODE indicator light is red by default. You can connect the handle receiver to the usb port of the computer to connect to the wireless handle. Enter the URL in the browser: <https://gamepad-tester.com/>. Pressing the button URL will display the change of the button value, as shown in the following figure:



Raspberry Pi series support mode: X-BOX mode. In X-BOX mode, the default POWER MODE indicator light is green. You can connect the handle receiver to the usb port of the computer to connect to the wireless handle. Enter the URL in the browser: <https://gamepad-tester.com/>. Pressing the button URL will display the change of the button value, as shown in the following figure:



- After re-plugging the handle receiver or restarting the motherboard, the handle will reset to the factory mode. If it cannot be controlled, you need to switch the mode again every time you plug or restart.

- In the case of unsuccessful matching, the POWER MODE indicator light will flash red and green all the time, and will not light up after a few seconds of sleep.