1. Install Handle

Insert the USB port of the handle into the USB port of the Raspberry Pi. (If you are logging in using remote jupyter lab, you need to plug it into the USB port of the computer you are logging in to remotely.)

2. Run handle control code

The following code content needs to be executed according to each actual step, and cannot be run all at once.

Handle remote control In this example, we will use a gamepad controller connected to a web browser machine to print the remote control data of the handle.
Create a handle controller The first thing we need to do is create an instance of the 'Controller' widget, which we will use to drive our USB controller. The 'Controller' widget accepts an 'index' parameter that specifies the number of controllers. If you have multiple controllers, or some game controllers appear in the form of multiple controllers, this is very useful. If only one handle is connected, the default index is 0. If there are multiple handles, the corresponding handle number needs to be entered based on the actual situation: 1. Open[http://html5gamepad.com]（ http://html5gamepad.com ）This webpage 2. Press the button of the handle you are using 3. Remember the corresponding index number that pops up when you press the button Next, we will use this index to create and display controllers.

import ipywidgets.widgets as widgets
controller = widgets.Controller(index=0) #Replace with the index number of the controller you have just tested and are currently using
display(controller)

import threading import time # Thread function operation library import inspect

```
import ctypes
```

```
#Create a method for actively stopping processes
def _async_raise(tid, exctype):
    """raises the exception, performs cleanup if needed"""
    tid = ctypes.c_long(tid)
    if not inspect.isclass(exctype):
        exctype = type(exctype)
    res = ctypes.pythonapi.PyThreadState_SetAsyncExc(tid, ctypes.py_object(exctype))
    if res == 0:
        raise ValueError("invalid thread id")
    elif res != 1:
        # """if it returns a number greater than one, you're in trouble,
        # and you should call it again with exc=NULL to revert the effect"""
        ctypes.pythonapi.PyThreadState_SetAsyncExc(tid, None)

def stop_thread(thread):
    _async_raise(thread.ident, SystemExit)
```

```
# Method of Creating a Handle Remote Rod to Receive and Print Data
##If the simulation mode of the handle is turned on (by pressing the ANALOG key), and the red light is on, the left direction key will not work, and both joysticks will output analog values. The left joystick will have sliding bars 0 and 1, and the right joystick will have sliding bars 2 and 3.
##Program functions:
1. Print which key is pressed each time.
```

```
def USB_Handle():
  delay_time = 0.01
  while 1:
    #Due to individual differences in the joystick handle, the reset value of the remote lever may not necessarily be zero, so it is necessary to filter with 0.1 to avoid misoperation.
     #Print data for the left joystick and directional keys
#A1 upper negative lower positive
    if controller.axes[1].value > 0.1:
      print('L_DOWN=%.2f'%(controller.axes[1].value))
    elif controller.axes[1].value < -0.1:
      print('L_UP=%.2f'%(controller.axes[1].value))
    time.sleep(delay_time)

    if controller.axes[0].value > 0.1:
      print('L_RIGHT=%.2f'%(controller.axes[0].value))
      time.sleep(delay_time)
    elif controller.axes[0].value < -0.1:
      print('L_LEFT=%.2f'%(controller.axes[0].value))
      time.sleep(delay_time)
```

```
if controller.buttons[10].value == True:
    print('L_PRESS')
    time.sleep(delay_time)

if controller.axes[2].value > 0.1:
    print('R_UP=%.2f'%(controller.axes[2].value))
    time.sleep(delay_time)
elif controller.axes[2].value < -0.1:
    print('R_DOWN=%.2f'%(controller.axes[2].value))
    time.sleep(delay_time)

if controller.axes[5].value > 0.1:
    print('R_RIGHT=%.2f'%(controller.axes[5].value))
    time.sleep(delay_time)
elif controller.axes[5].value < -0.1:
    print('R_LEFT=%.2f'%(controller.axes[5].value))
    time.sleep(delay_time)

if controller.buttons[11].value == True:
    print('R_PRESS')
    time.sleep(delay_time)


# NUM1=B0,NUM2=B1,NUM3=B2,NUM4=B3
if controller.buttons[0].value == True:
    print('NUM1')
    time.sleep(delay_time)
if controller.buttons[1].value == True:
    print('NUM2')
    time.sleep(delay_time)
if controller.buttons[2].value == True:
    print('NUM3')
    time.sleep(delay_time)
if controller.buttons[3].value == True:
    print('NUM4')
    time.sleep(delay_time)


# R1=B5,R2=B7
if controller.buttons[5].value == True:
    print('R1')
    time.sleep(delay_time)
if controller.buttons[7].value == True:
```

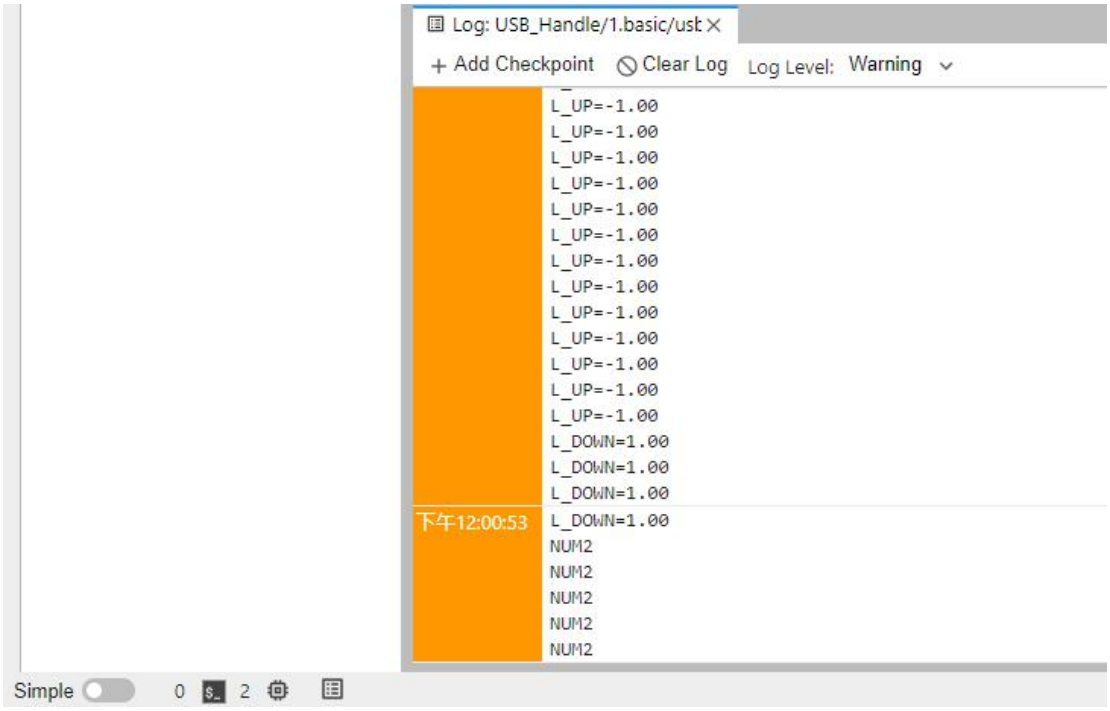
```
        print('R2')
        time.sleep(delay_time)

    # L1=B4,L2=B6
    if controller.buttons[4].value == True:
        print('L1')
        time.sleep(delay_time)
    elif controller.buttons[6].value == True:
        print('L2')
        time.sleep(delay_time)

    # SELECT=B8
    if controller.buttons[8].value == True:
        print('SELECT')
        time.sleep(delay_time)
    # START=B9
    if controller.buttons[9].value == True:
        print('START')
        time.sleep(delay_time)
```

```
#Start the thread of the handle by running the following cell code
#Wait for the handle control thread to start before operating the handle.
thread = threading.Thread(target=Arm_Handle)
thread.setDaemon(True)
thread.start()
```

```
#End handle thread.
#If a thread fails to start or end,
#Please restart the kernel and run it step by step again.
stop_thread(thread)
```

3. Use handle

Click on the debugging information window in the bottom left corner, and a debugging information window will pop up. Pressing different keys will print the corresponding data.



At the same time, the plugin above will display the operation of the handle in a graphical manner.

```
import ipywidgets.widgets as widgets
controller = widgets.Controller(index=0)
display(controller)
```