6.2 Using of Handle

Note: In the image operating system provided by Yahboom, the APP remote control process is enabled by default, in order to avoid multiple occupations of internal resources, causing some functions to fail to operate normally.

Before you running the code of this course, please follow the following to close the APP remote control process.

If you want to permanently close the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl disable jetbotmini_start
```

If you want to permanently open the function of the APP that starts automatically after booting, execute the following command:

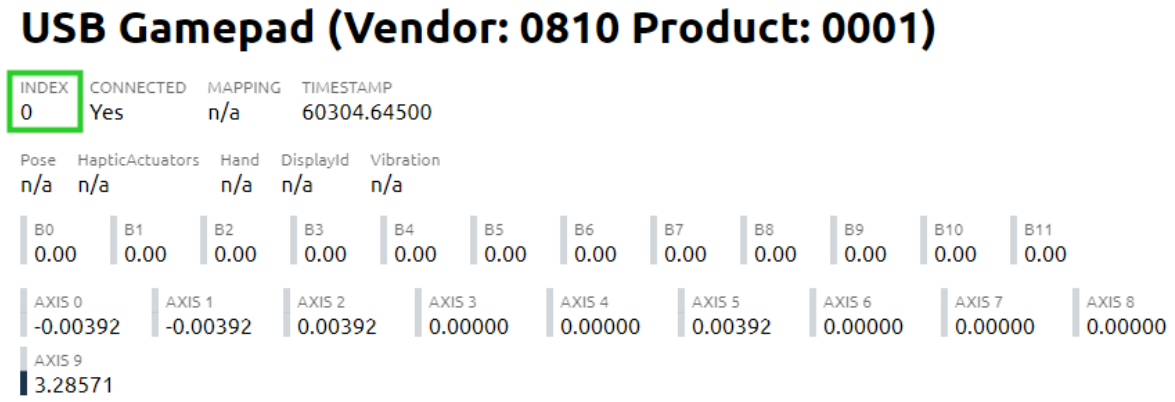
```
sudo systemctl enable jetbotmini_start
```

If you do not restart Jetbotmini to restart the APP function, execute the following command:

```
sudo systemctl stop jetbotmini_start
sudo systemctl start jetbotmini_start
```

6.2.1 Handle key test

First, we plug the controller receiver or the USB plug of the wired controller into the computer, turn on the switch (the wireless controller also needs to install the battery), and open the http://html5gamepad.com webpage , Because maybe your PC has been connected to more than one handle, so the default index of the handle you connected is not 0, so we need to check the index of the handle we are currently using on this web page to use it correctly. **After entering the webpage, you must first press the [START] button to trigger the detection and recognition of the handle and display the corresponding handle information**, the following is the detection interface of my handle, when we press the button of the handle, the above will With the corresponding button pressed, we can view the mapping value of the currently pressed button and then call the corresponding function in our program:



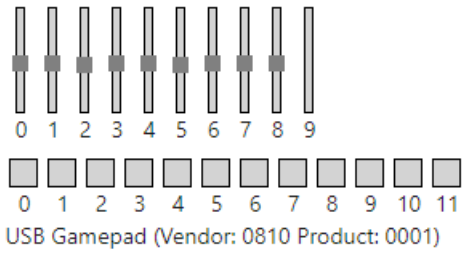
6.2.2 Wired handle to control movement

In this example, we will use a gamepad controller connected to a web browser machine to remotely control Jetbotmini. The first thing we have to do is to create an instance of the'Controller' widget, which we will use to drive our Jetbotmini. The "Controller" widget accepts an "index" parameter, which specifies the number of controllers. This is very useful if you have multiple controllers, or some gamepads appear as multiple controllers. To use your handle to control it correctly, we must set the index value we tested on the handle test webpage mentioned above.

Set the index value in the following unit code block:

```
import ipywidgets.widgets as widgets
controller = widgets.Controller(index=0)  #Replace with the index number of the controller you just tested
display(controller)
```

After setting up and executing the above cell code, it will display the handle key map as shown in the figure below:



Then import the corresponding module package

```
#Function library path import
import RPi.GPIO as GPIO
from jetbotmini import Robot
import traitlets
from jetbotmini import Camera
from jetbotmini import bgr8_to_jpeg
from jetbotmini import Heartbeat
import threading
import time
import smbus
# Thread function operation Library
import inspect
import ctypes

import traitlets
from IPython.display import display
from jetbotmini import Camera, bgr8_to_jpeg
import os
from uuid import uuid1
```

Then execute the following two cell codes to create and open the display camera initialization instance and add a heartbeat connection:

Create and open the display camera initialization instance

```
camera = Camera.instance(width=224, height=224)
image = widgets.Image(format='jpeg', width=224, height=224)  # this width and height does
camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg
display(image)
```

Add a heartbeat connection

```
def handle_heartbeat_status(change):
    if change['new'] == Heartbeat.Status.dead:
        camera_link.unlink()
        robot.stop()

heartbeat = Heartbeat(period=0.5)
# Attach callback function to heartbeat state
heartbeat.observe(handle_heartbeat_status, names='status')
```

Because we need to perform other operations after implementing the corresponding control of the handle, we need to introduce some operations of the thread here, and run the following code block to create a method to actively stop the process:

```
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)
```

Then we initialize the jetbotmini movement and the onboard LED and buzzer:

Load Robot class

Before we are ready to start programming for Jetbotmini, we need to import the "Robot" class. This class allows us to easily control the motors of Jetbotmini

```
robot = Robot()
```

Initialize the onboard LED and buzzer

```
#Pin definition
LED3BLUE_pin = 23  # BOARD pin 12, BCM pin 18
LED2GREEN_pin = 24  # BOARD pin 12, BCM pin 18
bus = smbus.SMBus(1)
Buzzer_ADD = 0x1B

#Board pin-numbering scheme
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

#Set the pin as the output pin, and the initial state can be selected as high
GPIO.setup(LED3BLUE_pin, GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(LED2GREEN_pin, GPIO.OUT, initial=GPIO.HIGH)
```

Then create a method for real-time control of Jetbotmini movement and control of LED and buzzer by the joystick and remote control:

```
def Jetbotmini_motion():
    while 1:
        if controller.axes[1].value <= 0.1:
            if (controller.axes[0].value <= 0.1 and controller.axes[0].value >= -0.1
                and controller.axes[1].value <= 0.1 and controller.axes[1].value >= -0.1):
                robot.stop()
            else:
                robot.set_motors(-controller.axes[1].value + controller.axes[0].value, -controller.axes[1].value - controller.axes[0].value)

            time.sleep(0.01)
        else:
            robot.set_motors(-controller.axes[1].value - controller.axes[0].value, -controller.axes[1].value + controller.axes[0].value)
            time.sleep(0.01)

        #Blue LED switch
        if controller.buttons[0].value == True:
            GPIO.output(LED3BLUE_pin, GPIO.HIGH)
        elif controller.buttons[2].value == True:
            GPIO.output(LED3BLUE_pin, GPIO.LOW)
        #Green LED switch
        if controller.buttons[3].value == True:
            GPIO.output(LED2GREEN_pin, GPIO.HIGH)
        elif controller.buttons[1].value == True:
            GPIO.output(LED2GREEN_pin, GPIO.LOW)
        #Buzzer control
        if controller.buttons[10].value == True:
            bus.write_byte_data(Buzzer_ADD,0x06,1)
            time.sleep(0.2)
            bus.write_byte_data(Buzzer_ADD,0x06,0)
```

After running the following cell code, turn on the handle to control the independent process of Jetbotmini movement in real time:

```
thread2 = threading.Thread(target=Jetbotmini_motion)
thread2.setDaemon(True)
thread2.start()
```

Now we can control Jetbotmini in real time through the handle.

If we want to modify the functions or expand other modules after running the above code, we can actively end the two threads that were just started. After the expansion, if we want to restart the thread, we can restart the thread by running the code unit block that started the thread above again.

```
stop_thread(thread2)
robot.stop()
```

Handle effect

Code	Handle button	Robot control
axes[0]	Left remote sensing (left positive and right negative)	Turn left turn right
axes[1]	Left remote sensing (upper positive and lower negative)	forward backward
buttons[0]	1	Blue LED off
buttons[1]	2	Green LED on
buttons[2]	3	Blue LED on
buttons[3]	4	Green LED off
buttons[10]	Left stick button	Buzzer

The corresponding complete source code is located:

/home/jetson/Notebooks/English/9.Use_of_handle/1.Wired_handle/Wired_handle.ipynb

6.2.3 Wireless controller FPV remote control

In the previous section, we used a wired handle to remotely control jetbotmini, and below, we use a wireless handle and mobile phone APP to achieve FPV remote control.

In this section, we use the APP. You need to enter the following command to start the APP program. After startup, you can hear three beeps.

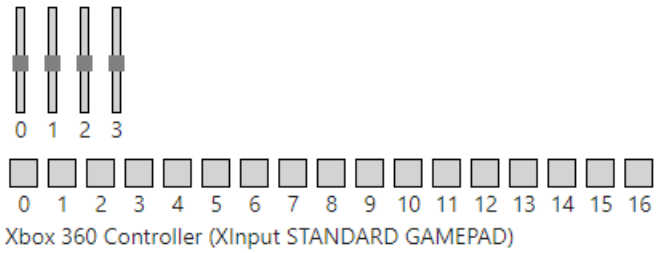
```
sudo systemctl start jetbotmini_start
```

As in the previous section, we will use the gamepad controller connected to the web browser machine to remotely control Jetbotmini. Set the index value.

Set the index value in the following unit code block:

```
import ipywidgets.widgets as widgets
controller = widgets.Controller(index=0)  #Replace with the index number of the controller you just tested
display(controller)
```

After setting up and executing the above cell code, it will display the handle key map as shown in the figure below. Here, the Xbox 360 handle is used, and the key position is different from the wired handle:



Xbox 360 Controller (XInput STANDARD GAMEPAD)

Then import the corresponding module package

```
#Function library path import
import RPi.GPIO as GPIO
from jetbotmini import Robot
import traitlets
from jetbotmini import Camera
from jetbotmini import bgr8_to_jpeg
from jetbotmini import Heartbeat
import threading
import time
import smbus
# Thread function operation Library
import inspect
import ctypes

import traitlets
from IPython.display import display
from jetbotmini import Camera, bgr8_to_jpeg
import os
from uuid import uuid1
```

Then execute the following two cell codes to add a heartbeat connection:

```
def handle_heartbeat_status(change):
    if change['new'] == Heartbeat.Status.dead:
        camera_link.unlink()
        robot.stop()

heartbeat = Heartbeat(period=0.5)
# Attach callback function to heartbeat state
heartbeat.observe(handle_heartbeat_status, names='status')
```

Because we need to perform other operations after implementing the corresponding control of the handle, we need to introduce some operations of the thread here, and run the following code block to create a method to actively stop the process:

```
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)
```

Then we initialize the jetbotmini movement and the onboard LED and buzzer:

Load Robot class

Before we are ready to start programming for Jetbotmini, we need to import the "Robot" class. This class allows us to easily control the motors of Jetbotmini

```
robot = Robot()
```

Initialize the onboard LED and buzzer

```
#Pin definition
LED3BLUE_pin = 23  # BOARD pin 12, BCM pin 18
LED2GREEN_pin = 24  # BOARD pin 12, BCM pin 18
bus = smbus.SMBus(1)
Buzzer_ADD = 0x1B

#Board pin-numbering scheme
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

#Set the pin as the output pin, and the initial state can be selected as high
GPIO.setup(LED3BLUE_pin, GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(LED2GREEN_pin, GPIO.OUT, initial=GPIO.HIGH)
```

Then create a method for real-time control of Jetbotmini movement and control of LED and buzzer by the joystick and remote control:

```
def Jetbotmini_motion():
    while 1:
        #Handle operation code (XBOX360 handle)
        if controller.axes[1].value <= 0:
            robot.set_motors(-controller.axes[1].value*0.6 + controller.axes[2].value/3, -controller.axes[1].value*0.6 - controller.axes[2].value/3)
            time.sleep(0.01)
        else:
            robot.set_motors(-controller.axes[1].value*0.6 - controller.axes[2].value/3, -controller.axes[1].value*0.6 + controller.axes[2].value/3)
            time.sleep(0.01)
        #Blue LED switch
        if controller.buttons[0].value == True:
            GPIO.output(LED3BLUE_pin, GPIO.HIGH)
        elif controller.buttons[2].value == True:
            GPIO.output(LED3BLUE_pin, GPIO.LOW)
        #Green LED switch
        if controller.buttons[3].value == True:
            GPIO.output(LED2GREEN_pin, GPIO.HIGH)
        elif controller.buttons[1].value == True:
            GPIO.output(LED2GREEN_pin, GPIO.LOW)
        #Buzzer control
        if controller.buttons[10].value == True:
            bus.write_byte_data(Buzzer_ADD,0x06,1)
            time.sleep(0.2)
            bus.write_byte_data(Buzzer_ADD,0x06,0)
```

After running the following cell code, turn on the handle to control the independent process of Jetbotmini movement in real time:

```
thread2 = threading.Thread(target=Jetbotmini_motion)
thread2.setDaemon(True)
thread2.start()
```

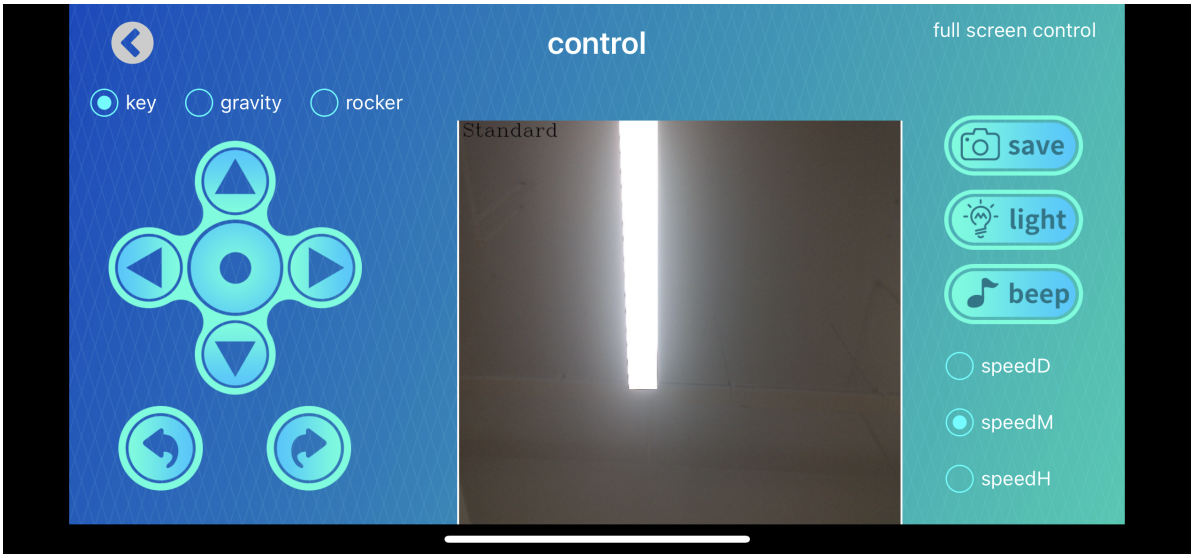
Now we can control Jetbotmini in real time through the handle. Next we install the handle bracket.

Open the button at the bottom of the handle bracket and place it on the table.

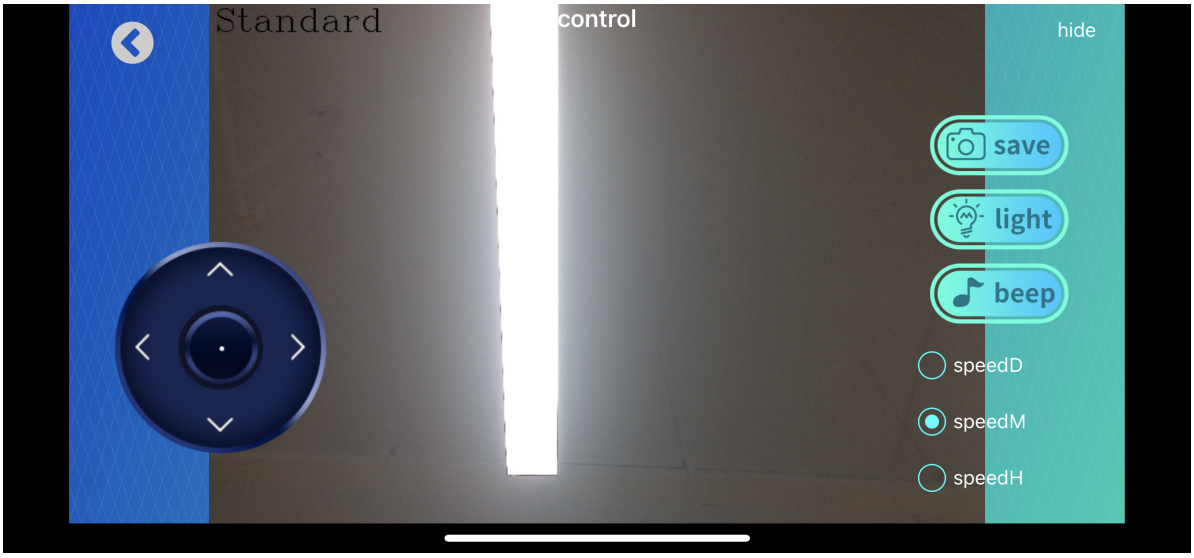
Put the handle into it.

Finally, re-fasten the button at the bottom.

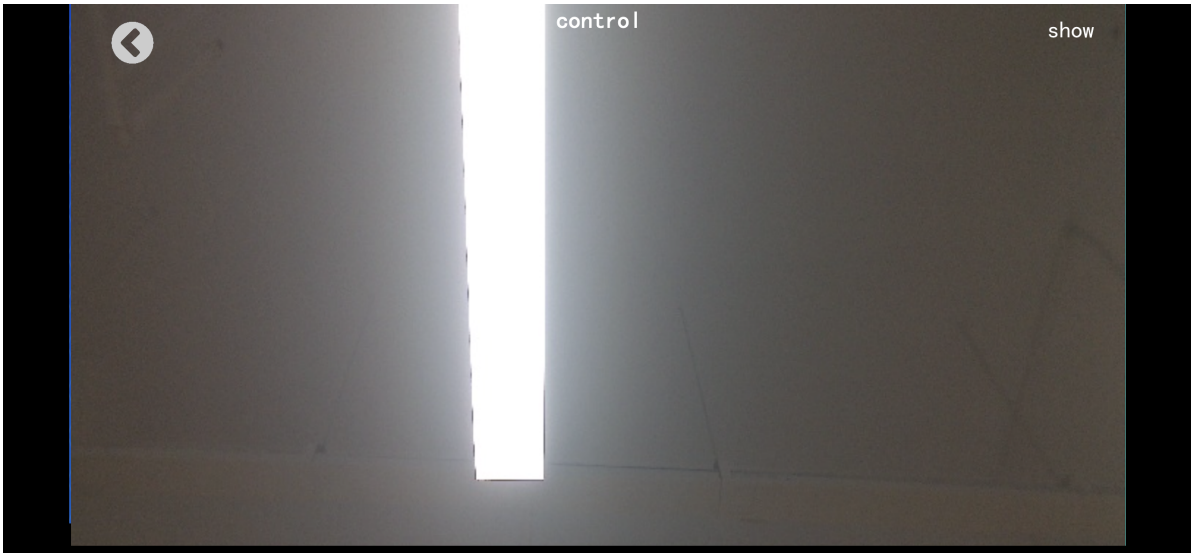
Next, we open the APP according to 2.5.2, after configuration, enter the remote control interface, click on the upper right corner [full screen control]



After entering the interface, click on the upper right corner [Hide]



Two fingers to zoom the screen to a suitable position, the effect is as follows.



Next, we install the mobile phone on the bracket. The top of the handle bracket is a mobile phone holder with an adjustable width, which can be widened upwards and then placed in the mobile phone. The spring of the bracket will automatically clamp the mobile phone firmly. The two plastic screws in the middle of the bracket are used to adjust the tilt angle of the mobile phone. After loosening, the display angle of the mobile phone can be adjusted, and then tighten the fixed angle.

Note: Before loosening the plastic screws, please take care to protect the phone. Due to the heavy weight of the phone, it may turn back. Please adjust it on the table.

If we want to modify the functions or expand other modules after running the above code, we can actively end the two threads that were just started. After the expansion, if we want to restart the thread, we can restart the thread by running the code unit block that started the thread above again.

```
stop_thread(thread2)
robot.stop()
```

Handle effect

Code	Key on handle	Control car
axes[2]	Right remote sensing (left positive and right negative)	Turn left turn right
axes[1]	Left remote sensing (upper positive and lower negative)	forward backward
buttons[0]	A	Blue LED off
buttons[1]	B	Green LED on
buttons[2]	X	Blue LED on
buttons[3]	Y	Green LED off
buttons[10]	Left stick button	buzzer

The corresponding complete source code is located:

/home/jetson/Notebooks/English/9.Use_of_handle/2.FPV_by_wireless_handle/FPV_by_wireless_handle.ipynb