5.2 Using of the handle

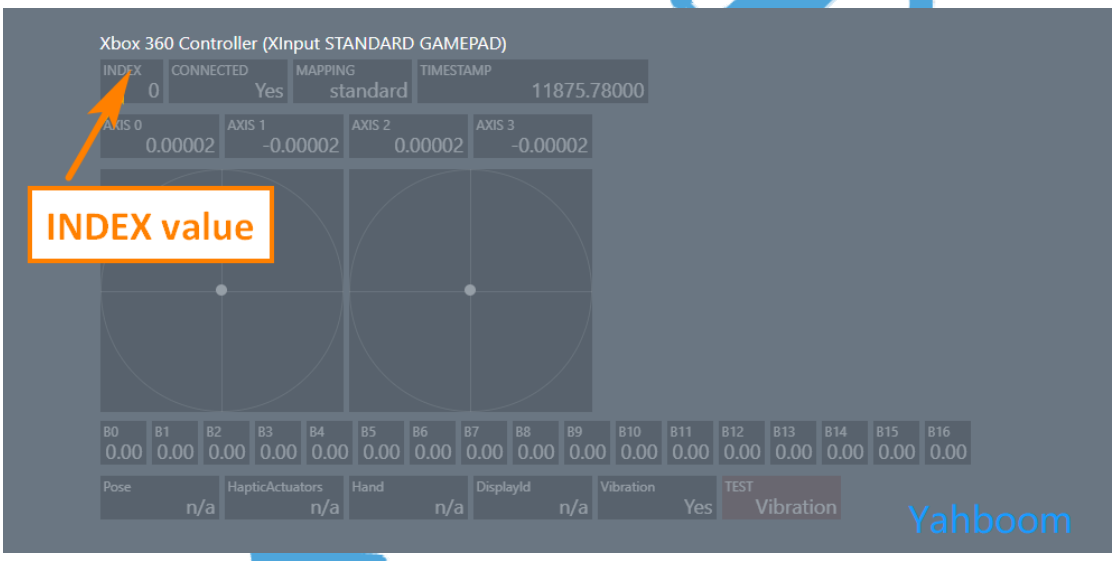
1. Handle key test

First of all, we open the http://html5gamepad.com webpage, and connect your Handle to your computer.

Because maybe your PC computer can not only connect a handle, so the default value of the index of the handle you connect is not 0, so we need to go to this page to view the handle we are currently using. The index can be used correctly.

After entering the webpage, the button on Handle must be pressed to trigger the detection and recognition.

The handle displays the corresponding handle information. The following is the detection interface of my handle. When we press the button of the handle, the corresponding button will also be pressed. We can view the mapped value of the currently pressed button and then call it into the corresponding function in our program.



2. The handle controls the omnidirectional movement

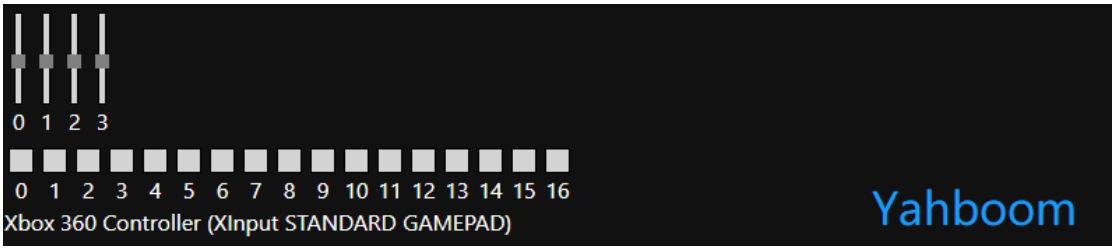
In this example, we will remotely control Jetbot using a gamepad controller connected to the web browser machine.

First, we need to create an instance of the 'Controller' widget, which we will use to drive our Jetbot. The "Controller" widget accepts an "index" parameter that specifies the number of controllers.

This is useful if you have multiple controllers, or if some gamepads appear as multiple controllers. In order to properly use your handle to control, we need to set the index value tested by the handle test page mentioned above:

Set the index value in the following code:

After executing the above code, the handle key map shown in the figure below will be displayed:



Then, import the corresponding module package. Code as shown below:

Import some module

```
#Import function library path
from servoserial import ServoSerial
from jetbot import Robot
import traitlets
from jetbot import Camera
from jetbot import bgr8_to_jpeg
from jetbot import Heartbeat
import threading
import time
# Thread function operation library
import inspect
import ctypes

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

Then, execute the code for the next two cells, create and open the display camera initialization instance and add a heartbeat connection:
Code as shown below:

Create and open a display camera initialization instance

```
camera = Camera.instance(width=224, height=224)
image = widgets.Image(format='jpeg', width=224, height=224)  # this width and height doesn't necessarily have to match the camera
camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)
display(image)
```

Add heartbeat connection

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

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


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

Code as shown below:

```
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)
```

Create a method to detect the button action to control the motion of the PTZ camera up, down, left, and right:
Code as shown below:

```
'''Servo Part'''
servo_device = ServoSerial()
def camUpFunction():
    global updownpulse
    updownpulse+=15
    if updownpulse>4095:
        updownpulse=4095
    servo_device.Servo_serial_control(2, updownpulse)

def camDownFunction():
    global updownpulse
    updownpulse-=15
    if updownpulse<1300:
        updownpulse=1300
    servo_device.Servo_serial_control(2, updownpulse)

def camLeftFunction():
    global leftrightpulse
    leftrightpulse+=15
    if leftrightpulse>3600:
        leftrightpulse=3600
    servo_device.Servo_serial_control(1, leftrightpulse)

def camRightFunction():
    global leftrightpulse
    leftrightpulse-=15
    if leftrightpulse<600:
        leftrightpulse=600
    servo_device.Servo_serial_control(1, leftrightpulse)

def camservoInitFunction():
    global leftrightpulse, updownpulse
    leftrightpulse = 2048
    updownpulse = 2048
    servo_device.Servo_serial_control(1, 2048)
    time.sleep(0.1)
    servo_device.Servo_serial_control(2, 2048)
```

Initialize the PTZ camera position and create an instance that controls the Jetbot motion:
Code as shown below:

Initialize the position of the PTZ camera

Run the following cell code to initialize the PTZ project to the initial location

```
camservoInitFunction()
```

Load the Robot class

This class allows us to easily control the JetBot motor

```
robot = Robot()
```

In order to increase the visual effect of the point, we open a thread to make the on board LED light under the LOGO realize the breathing light effect:
Code as shown below:

Create onboard breathing light method

```
def BLN_Onboard():
    global i , k
    i = k = 0
    while True:
        if k == 0:
            robot.set_bln(i)
            i += 0.01
            if( i >= 1 ):
                k = 1
            time.sleep(0.005)
        elif k == 1:
            robot.set_bln(i)
            i -=0.01
            if i <= 0 :
                k = 0
            time.sleep(0.005)
```

Turn on the onboard breathing light independent process by running the cell code below

```
thread1 = threading.Thread(target=BLN_Onboard)
# thread1.setDaemon(True)
thread1.start()
```

Create a handle rocker to control the Jetbot motion in real time:
Code as shown below:

Create a handle to control the movement of Jetbot in real time

Please use the ANOLOG button in the middle of the handle to switch to the simulation mode before using.

Program features:

1. Left rocker control Jetbot movement, right rocker control the servo movement
2. Press the SELECT button to reset the PTZ angle
3. Press the L side button No.1 to control the PTZ rise, press the L side button No.2 to control the gimbal decline.

In the default code, the Yahboom handle is used by default. For other handles, please refer to the key table to change the value.

If you are using Yahboom accessory, please use the code section ---1
If you are using the Xbox360's handle, please use the code section ---2

```
def jetbot_motion():
    count1 = count2 = count3 = count4 =  count5 = 0
    while 1:
        #Robot car Left and right DC motor
        #Handle control code---1(Jetbot Yahboom handle)
        #Yahboom Rocker reset value is 0.0039,
        #In order to prevent the motor from being abnormal, the following code is added to operate.
        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)
          #Handle control code---2(Xbox360手柄)
#         if controller.axes[1].value <= 0:
#             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)
```

```
        #Servo control camera up and down
        if controller.axes[2].value == 1:
            count1  += 1
            if count1 >= 3:
                camDownFunction()
                count1 = 0
        elif controller.axes[2].value == -1:
            count1  += 1
            if count1 >= 3:
                camUpFunction()
                count1 = 0
        else:
            count1 = 0
        #Servo control camera left and right
        if controller.axes[5].value == 1:
            count2  += 1
            if count2 >= 3:
                camRightFunction()
                count2 = 0
        elif controller.axes[5].value == -1:
            count2  += 1
            if count2 >= 3:
                camLeftFunction()
                count2 = 0
        else:
            count2 = 0
        #Servo control camera up down, left and right is reset
        if controller.buttons[8].value == True:
            count3 += 1
            if count3 >= 3:
                camservoInitFunction()
                count3 = 0
        else:
            count3 = 0

        #Servo control servo rise and decline
        if controller.buttons[6].value == True:
            count4 += 1
            if count4 >= 3:
                robot.set_vertical_motors(1)
                count4 = 0
        elif controller.buttons[4].value == True:
            count4 += 1
            if count4 >= 3:
                robot.set_vertical_motors(-1)
```

```
                count4 = 0
        else:
            robot.set_vertical_motors(0)
            count4 = 0
```

After running the following code, you can open the handle to control the independent process of Jetbot motion in real time:

Control the independent process of Jetbot motion in real time by running the cell code below to open the handle

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

If you need to modify the functions or extend other modules after running the above code, you can actively terminate the two threads that have just been opened.

After the extension, if you want to re-open the thread, you can re-open the thread in the code unit block that opens the thread by running it again.

```
Ending onboard breathing RGB process
```

```
stop_thread(thread1)
```

```
Ending Jetbot movement process
```

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

The corresponding complete source code is located at:
/home/jetbot/Notebook/10.Using of Handle/Handle control.ipynb