Hiwonder

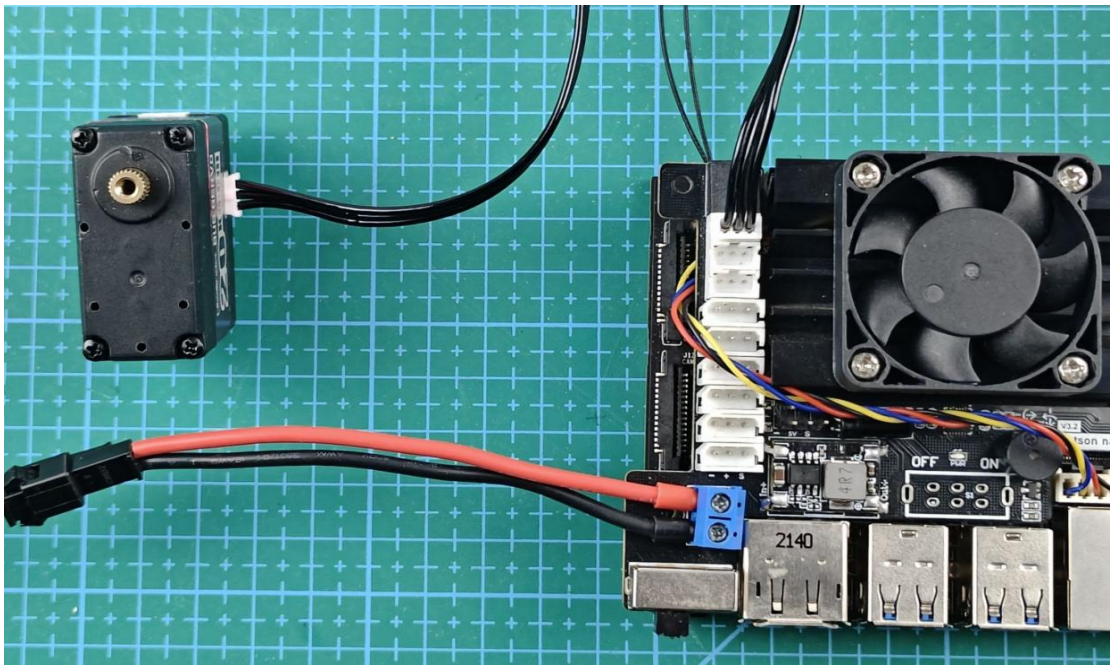
Jetson Nano Development

V1.0

1. Getting Started

1.1 Wiring Instruction

This section employs a Jetson Nano board and Jetson Nano expansion board, powered by a 11.1V 6000mAh lithium battery. Connect the bus servo to any bus interface on the Jetson Nano expansion board.



Note:

① When using our lithium battery, connect the lithium battery interface wire with red to positive (+) and black to negative (-) to the DC port.

② If the interface wire is not connected to the lithium battery, do not directly connect it to the battery interface wire to avoid a short circuit between positive and negative poles, which may cause a short circuit.

1.2 Environment Configuration

```
hiwonder@hiwonder:~/Desktop$ chmod a+x serial_servo sdk/
hiwonder@hiwonder:~/Desktop$
```

Install NoMachine software on PC. The software package is stored in "**2. Software -> Remote Connection Software**". For the detailed operations of Keil5, please refer to the relevant tutorials.

Drag the program and the SDK library files into the Raspberry Pi system image, using the desktop as an example. Note: Ensure the library files are placed in the same directory as the program.

Open the command-line terminal, and input command to add execution permissions:

"**chmod a+x serial_servo sdk/**"

```
hiwonder@hiwonder:~/Desktop$ chmod a+x serial_servo sdk/
hiwonder@hiwonder:~/Desktop$
```

2. Development Case

2.1 Case 1 Retrieve Bus Servo Information

In this case, the servo ID, position, temperature and other information will be displayed in the terminal monitor.

```
id:5
pos:886
dev:38
angle_range:(0, 1000)
voltage_range:(4500, 14000)
temperature_warn:85
temperature:34
vin:11905
lock:0
```

2.1.1 Run Program

1) Open a new terminal and the input the command "**cd Desktop/serial_servo/**" to change to the directory where the program is located.

```
hiwonder@hiwonder:~$ cd Desktop/serial_servo/
hiwonder@hiwonder:~/Desktop/serial_servo$
```

2) Input the command to execute the program:

"**python3 serial_servo_status.py**"。

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_status.py
```

2.1.2 Performance

After the program runs, the servo status information will be printed in the terminal.

```
id:5
pos:886
dev:38
angle_range:(0, 1000)
voltage_range:(4500, 14000)
temperature_warn:85
temperature:34
vin:11863
lock:0
```

The specific meanings of the various pieces of information are as follows:

① **id**: Servo ID. In this example, it is 5.

② **pos**: Current position of the servo. In this example, it is 886.

③ **dev**: Servo deviation. In this example, it is 38.

④ **angle range**: Servo angle range. In this example, it is 0-1000.

⑤ **voltage range**: Servo voltage range. In this example, it is 4.5~14V.

⑥ **temperature warn**: Servo temperature warning threshold. In this example, it is 85°C.

⑦ **temperature**: Current temperature of the servo. In this example, it is 34°C.

⑧ **vin**: Current voltage value of the servo. In this example, it is 11.863V.

⑨ **lock**: Whether the servo is powered on. In this example, it is 0, indicating it is powered off. When it is 1, it indicates it is powered on.

2.1.3 Case Program Analysis

- **Import the Necessary Module**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

First, import the **sys** and **time** modules, as well as the **hiwonder_servo_controller** module. By importing this module, we can use the functions and classes defined within it to control servos.

- **Create Servo Control Oject**

```
servo_control = hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1', 115200)
```

Instantiate the **HiwonderServoController** class to create a **servo_control** object, passing the parameters for the serial servo device path and baud rate.

- **Obtain and Print Servo Status**

```
servo_id = servo_control.get_servo_id()
pos = servo_control.get_servo_position(servo_id)
dev = servo_control.get_servo_deviation(servo_id)
if dev > 125:
    dev = -(0xff - (dev - 1))
angle_range = servo_control.get_servo_range(servo_id)
vin_range = servo_control.get_servo_vin_range(servo_id)
temperature_warn = servo_control.get_servo_temp_range(servo_id)
temperature = servo_control.get_servo_temp(servo_id)
vin = servo_control.get_servo_vin(servo_id)
load_state = servo_control.get_servo_load_state(servo_id)
```

After entering the loop function, various status information of the servo can be obtained by calling the methods of the **servo_control object**. This status information includes the servo ID, position, deviation, angle range, voltage range, over-temperature alarm threshold, current temperature, voltage,

and the power-on status of the servo.

After obtaining this information, these parameters can be printed out.

```
print('id:%s'%(str(servo_id).ljust(3)))
print('pos:%s'%(str(pos).ljust(4)))
print('dev:%s'%(str(dev).ljust(4)))
print('angle_range:%s'%(str(angle_range).ljust(4)))
print('voltage_range:%s'%(str(vin_range).ljust(5)))
print('temperature_warn:%s'%(str(temperature_warn).ljust(4)))
print('temperature:%s'%(str(temperature).ljust(4)))
print('vin:%s'%(str(vin).ljust(4)))
print('lock:%s'%(str(load_state).ljust(4)))
print('')
```

Print out the obtained servo status information. Use string formatting and the **ljust()** method for each print statement to align the output results.

- **Set the Interrupt Flag**

```
except KeyboardInterrupt:
    break
```

Finally, configure the button. When the terminal interface is printing, press Ctrl+C, catch the **KeyboardInterrupt** exception, exit the loop, and terminate the program execution.

2.2 Case 2 Set Servo ID

2.2.1 Run Program

1) Open a new terminal and the input the command "**cd Desktop/serial_servo/**" to change to the directory where the program is located.

```
hiwonder@hiwonder:~$ cd Desktop/serial_servo/
hiwonder@hiwonder:~/Desktop/serial_servo$
```

2) Input the command to execute the program:

"**python3 set_serial_servo_status.py**"

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 set_serial_servo_status.py
set serial servo status

-------------------------
1 id
2 dev
3 save_dev
4 angle_range
5 voltage_range
6 temperature_warn
7 lock
8 help
9 exit
-------------------------
*******current status**********
id:5
dev:-125
angle_range:(0, 1000)
voltage_range:(4500, 14000)
temperature_warn:85
lock:0
******************************
input mode:
```

3) In the command prompt, first enter "1" to enter the ID setting mode. Then input the ID number. If the current servo ID in the program is 5, here is an example of changing it to 3. After entering the new ID, simply press Enter to confirm.

Note: The ID number range for setting is 0-253.

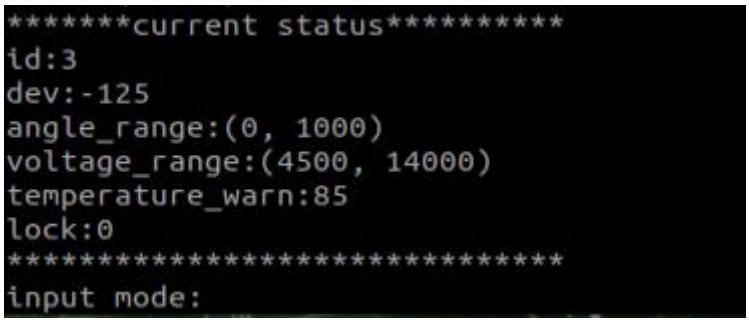
```
atus.py
set serial servo status

-------------------------
1 id
2 dev
3 save_dev
4 angle_range
5 voltage_range
6 temperature_warn
7 lock
8 help
9 exit
-------------------------
*******current status**********
id:5
dev:-125
angle_range:(0, 1000)
voltage_range:(4500, 14000)
temperature_warn:85
lock:0
******************************
input mode:1
current id:3
```

4) To exit the settings, press "**Ctrl+C**".

2.2.2 Performance

After the modification, the feedback interface will directly display the currently set ID.



2.2.3 Case Program Analysis

- **Import the Necessary Module**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

First, import the sys and time modules, as well as the **hiwonder_servo_controller** module. By importing this module, we can use the functions and classes defined within it to control the servos.

- **Create Servo Control Object**

```
servo_control = hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1', 115200)
```

Instantiate an object of the **HiwonderServoController** class to create a **servo_control** object, passing the parameters of the serial servo's device path and baud rate.

- **Enter the Main Loop Function**

```
while True:
    try:
        get_status()
        mode = int(input('input mode:'))
        if mode == 1:
            oldid = int(input('current id:'))
            newid = int(input('new id(0-253):'))
            servo_control.set_servo_id(oldid, newid)
        elif mode == 2:
            servo_id = int(input('servo id:'))
            dev = int(input('deviation(-125~125):'))
            if dev < 0:
                dev = 0xff + dev + 1
            servo_control.set_servo_deviation(servo_id, dev)
        elif mode == 3:
            servo_id = int(input('servo id:'))
            servo_control.save_servo_deviation(servo_id)
        elif mode == 4:
            servo_id = int(input('servo id:'))
            min_pos = int(input('min pos(0-1000):'))
            max_pos = int(input('max pos(0-1000):'))
            servo_control.set_servo_range(servo_id, min_pos, max_pos)
        elif mode == 5:
            servo_id = int(input('servo id:'))
            min_vin = int(input('min vin(4500-14000):'))
            max_vin = int(input('max vin(4500-14000):'))
            servo_control.set_servo_vin_range(servo_id, min_vin, max_vin)
        elif mode == 6:
            servo_id = int(input('servo id:'))
            min_temp = int(input('temperature(0-85):'))
            servo_control.set_servo_temp_range(servo_id, min_temp)
```

In the main loop, the program first calls the **get_status()** function to print the current status of the servo. Then, it selects the operation mode based on user input. Depending on the chosen mode, the program executes corresponding code blocks to configure and control the servo.

- **Print Menu Options**

```
print('''
--------------------------
1 id
2 dev
3 save_dev
4 angle_range
5 voltage_range
6 temperature_warn
7 lock
8 help
9 exit
--------------------------''')
```

To print the menu options, different operation modes are provided for the user to choose from.

- **Set the Interrupt Flag**

```
    except KeyboardInterrupt:
        break
```

Finally, configure the button. When the terminal interface is printing, press Ctrl+C, catch the **KeyboardInterrupt** exception, exit the loop, and terminate the program execution.

2.3 Case 3 Control Servo to Rotate

2.3.1 Run Program

1) Open a new terminal and the input the command "**cd Desktop/serial_servo/**" to change to the directory where the program is located.

```
hiwonder@hiwonder:~$ cd Desktop/serial_servo/
hiwonder@hiwonder:~/Desktop/serial_servo$
```

2) Input the command to execute the program:

"**python3 serial_servo_move_demo.py**"

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_move_demo.py
serial servo move between 500 - 1000
```

2.3.2 Performance

After the program runs, the servo will rotate back and forth between positions 0 and 1000 with a 1-second interval.

2.3.3 Case Program Analysis

- **Import the Necessary Module**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

First, import the sys and time modules, as well as the **hiwonder_servo_controller** module. By importing this module, we can use the functions and classes defined within it to control the servos.

- **Create Servo Control Object**

```
servo_control = hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1', 115200)
```

Instantiate an object of the **HiwonderServoController** class to create a

servo_control object, passing the parameters of the serial servo's device path and baud rate.

- **Enter the Main Loop Function**

```
while True:
    try:
        servo_id = 6  # servo ID(0-253)
        position = 500  # position(0-1000)
        duration = 500  # time(20-30000)
        servo_control.set_servo_position(servo_id, position, duration)
        time.sleep(duration/1000.0 + 0.1)  # It takes an additional 100ms to fully rotate to the designat

        servo_id = 6
        position = 1000
        duration = 500
        servo_control.set_servo_position(servo_id, position, duration)
        time.sleep(duration/1000.0 + 0.1)
    except KeyboardInterrupt:
        servo_id = 6
        position = 0
        duration = 500
        servo_control.set_servo_position(servo_id, position, duration)
        break
```

In the main loop, the program sets the position and running time of the servo by calling the **set_servo_position()** method. First, it sets the servo ID to 1, position to **500**, and running time to **500ms**. Then, it calls **set_servo_position()** to control the servo to move to the specified position.

Next, it uses the **time.sleep()** method to wait for the servo to complete its movement. To ensure the servo reaches the target position completely, an additional delay of 100ms is added. Then, it sets the servo position to 1000 and calls **set_servo_position()** again to control the servo to move to the new position.

- **Set the Interrupt Flag**

```
except KeyboardInterrupt:
    break
```

Finally, configure the button. When the terminal interface is printing, press Ctrl+C, catch the **KeyboardInterrupt** exception, exit the loop, and terminate the program execution.

2.4 Case 4 Adjust Servo Speed

2.4.1 Run Program

1) Open a new terminal and the input the command “**cd**

Desktop/serial_servo/" to change to the directory where the program is located.

```
hiwonder@hiwonder:~$ cd Desktop/serial_servo/
hiwonder@hiwonder:~/Desktop/serial_servo$
```

2) Input the command to execute the program:

"**python3 serial_servo_move_demo.py**"

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_speed_demo.py
serial servo move at different speed
```

2.4.2 Performance

After the program runs, the servo behaves as follows:

① The servo starts from position 500.

② It rotates to position 1000 over 0.5 seconds.

③ It rotates to position 500 over 1.5 seconds.

④ It rotates to position 0 over 2.5 seconds.

⑤ It rotates to position 500 over 3.5 seconds.

2.4.3 Case Program Analysis

- **Import the Necessary Module**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

First, import the sys and time modules, as well as the **hiwonder_servo_controller** module. By importing this module, we can use the functions and classes defined within it to control the servos.

- **Create Servo Control Object**

```
servo_control = hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1', 115200)
```

Instantiate an object of the **HiwonderServoController** class to create a **servo_control** object, passing the parameters of the serial servo's device path and baud rate.

- **Control Servo Speed**

```
servo_id = 6  # servo ID (0-253)
position = 500  # position(0-1000)
duration = 1000  # time(20-30000)
servo_control.set_servo_position(servo_id, position, duration)
time.sleep(duration/1000.0 + 0.1)

servo_id = 6
position = 1000
duration = 500
servo_control.set_servo_position(servo_id, position, duration)
time.sleep(duration/1000.0 + 0.1)

servo_id = 6  # servo ID (0-253)
position = 500  # position(0-1000)
duration = 1000  # time(20-30000)
servo_control.set_servo_position(servo_id, position, duration)
time.sleep(duration/1000.0 + 0.1)

servo_id = 6
position = 0
duration = 500
servo_control.set_servo_position(servo_id, position, duration)
time.sleep(duration/1000.0 + 0.1)
```

First, set the servo ID to 6, position to 500, and running time to 1000ms. Then, call the set_servo_position() method to control the servo to move to the specified position.

Next, use the **time.sleep()** method to wait for the servo to complete its movement. To ensure the servo reaches the target position completely, an additional delay of 100ms is added. Then, set the servo position to 1000 and call **set_servo_position()** again to control the servo to move to the new position. Similarly, use the **time.sleep()** method to wait for the servo to complete its movement.

Then, set the servo position to 500 and call **set_servo_position()** again to control the servo to move to the new position.

Finally, set the servo to 0 and call **set_servo_position()** again to control the servo to move to the new position. After each servo movement, use the **time.sleep()** method to wait for the servo to complete its movement.

- **Set the Interrupt Flag**

```
except KeyboardInterrupt:
    break
```

Finally, configure the button. When the terminal interface is printing, press Ctrl+C, catch the **KeyboardInterrupt** exception, exit the loop, and terminate

the program execution.

2.5 Case 5 Teaching Record Operations

2.5.1 Run Program

1) Open a new terminal and the input the command "**cd Desktop/serial_servo/**" to change to the directory where the program is located.

```
hiwonder@hiwonder:~$ cd Desktop/serial_servo/
hiwonder@hiwonder:~/Desktop/serial_servo$
```

2) Input the command to execute the program:

"**python3 set_serial_servo_record.py**"

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_speed_demo.py
serial servo move at different speed
```

2.5.2 Performance

After the program runs, return the servo to the central position, power it off, and prompt: '**Start turning the servo**.' At this point, move the servo arm to any angle and note the current position. Press key 0; the servo powers on, returns to the neutral position, and then rotates to the angle recorded as pos.

2.5.3 Case Program Analysis

- **Import the Necessary Module**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

First, import the sys and time modules, as well as the **hiwonder_servo_controller** module. By importing this module, we can use the functions and classes defined within it to control the servos.

- **Create Servo Control Object**

```
servo_control = hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1', 115200)
```

Instantiate an object of the **HiwonderServoController** class to create a **servo_control** object, passing the parameters of the serial servo's device path

and baud rate.

- **Obtain and Print Servo ID**

```
servo_id = servo_control.get_servo_id()
print('id:',servo_id)
```

Call **get_servo_id()** to retrieve the servo's ID and print it out.

- **Control Servo Back to Central Position**

```
servo_control.set_servo_position(servo_id, 500, 500)
```

Call **set_servo_position()** to move the servo back to position 500 and set the running time to 1000ms.

- **Setting the servo to power-off mode.**

```
servo_control.unload_servo(servo_id,False)
```

Call **unload_servo()** to power off the servo, allowing it to be manually manipulated.

- **Obtain Input Mode**

```
mode = int(input('input mode:'))
```

Use the **input()** function to obtain the mode input from the user and convert it to an integer type.

- **Perform operations based on the obtained mode**

```
if mode == 0:
    servo_control.get_servo_position(servo_id)
    pos = servo_control.get_servo_position(servo_id)
    servo_control.set_servo_position(servo_id, 0, 500)
    time.sleep(3)
    servo_control.set_servo_position(servo_id, pos, 500)
```

If the mode is 0, first call **get_servo_position()** to retrieve the current position of the servo. Then set the servo position to 0 and wait for 3 seconds. Finally, restore the servo position to its previous position.

Users can also modify the servo ID as needed.

- **Set the Interrupt Flag**

```
    except KeyboardInterrupt:
        break
```

Finally, configure the button. When the terminal interface is printing, press Ctrl+C, catch the **KeyboardInterrupt** exception, exit the loop, and terminate the program execution.