Development Tutorial (Raspberry Pi Version)

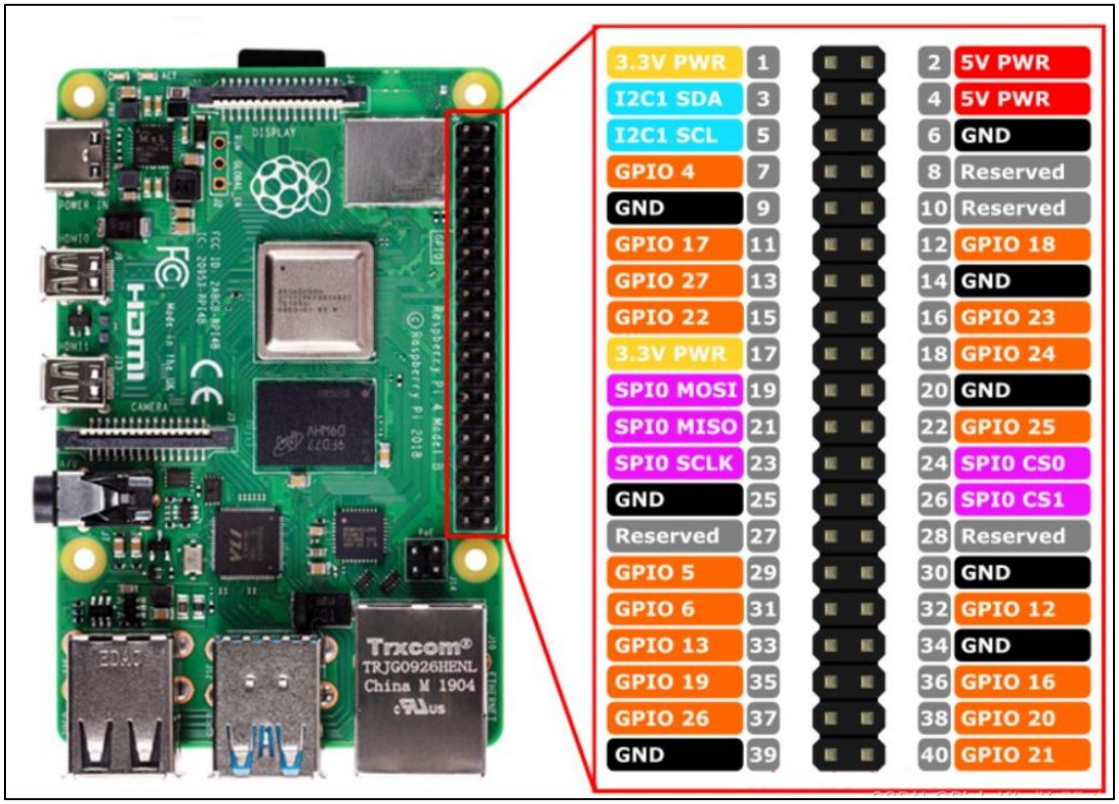
After executing the program, the chassis moves forward for 2 seconds, then reverses for 2 seconds, followed by a left turn for 2 seconds. Finally, it resumes moving forward for another 2 seconds before coming to a stop.

1 Hardware Introduction

1.1 Raspberry Pi 4B Controller

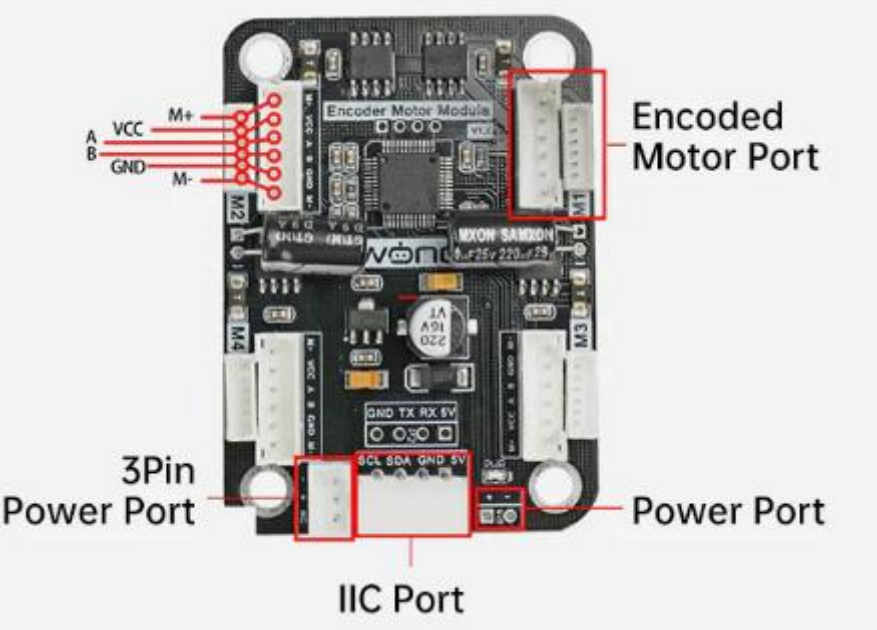
The Raspberry Pi 4B is a compact embedded computer featuring a built-in GPU with a frequency of 500MHz, equipped with 1/2/4GB LPDDR4 memory, a Gigabit network card, Bluetooth 5.0, USB 3.0 interface, and microHDMI interface. With the mentioned configuration, the Raspberry Pi provides a favorable hardware environment for programming and development.

The diagram below illustrates the pin port distribution of the Raspberry Pi 4B, which will be further discussed in the subsequent wiring and development process.



1.2 4-Channel Encoder Motor Driver

This is a motor drive module designed to work with a microcontroller for driving TT motors or magnetic encoder motors. Each channel is equipped with a YX-4055AM motor drive chip, and its voltage range is DC 3V-12V. The specific voltage depends on the voltage requirements of the connected motor. The interface distribution is illustrated in the figure below:



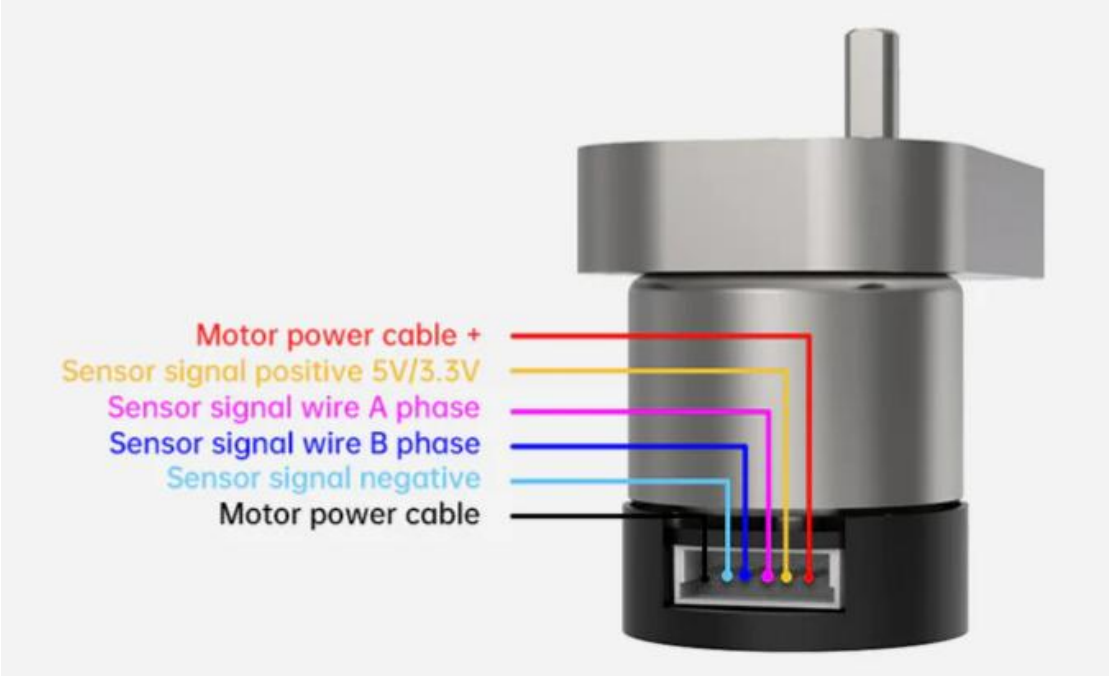
The introduction to the interface on the driver is as below:

Interface type	NO.	Function
Encoder motor interface	GND	Negative electrode of the Hall power
	A	A-phase pulse signal output terminal
	B	B-phase pulse signal output terminal
	VCC	Positive electrode of the Hall power
	M+	Positive electrode of the motor power supply
	M-	Positive electrode of the motor power supply
	Note: 1. The voltage between VCC and GND is determined based on the power supply voltage of the microcontroller used. Typically, 3.3V or 5V is used. 2. When the spindle rotates clockwise, the output pulse signal of channel A is ahead of channel B; when the spindle rotates	

	counterclockwise, the signal of channel A is behind channel B. 3. The voltage between M+ and M- is determined based on the voltage requirements of the motor used.	
IIC	SCL	Clock line
	SDA	Bi-directional data line
	GND	Power ground line
	5V	5V DC output
3Pin power port	-	Power negative electrode
	+	Power positive input
	NC	Empty
Power port	+	Power positive input
	-	Power negative electrode

1.3 Encoder Geared Motor

The chassis uses the motor model JGB3865-520R45-12. In this designation, 'J' stands for a DC motor, 'GB' signifies an eccentric output shaft, '38' denotes the gearbox diameter, '520' represents the motor model, 'R45' indicates a reduction ratio of 1:45, and '12' denotes the rated voltage of 12V. The interface specifications are illustrated in the diagram below:



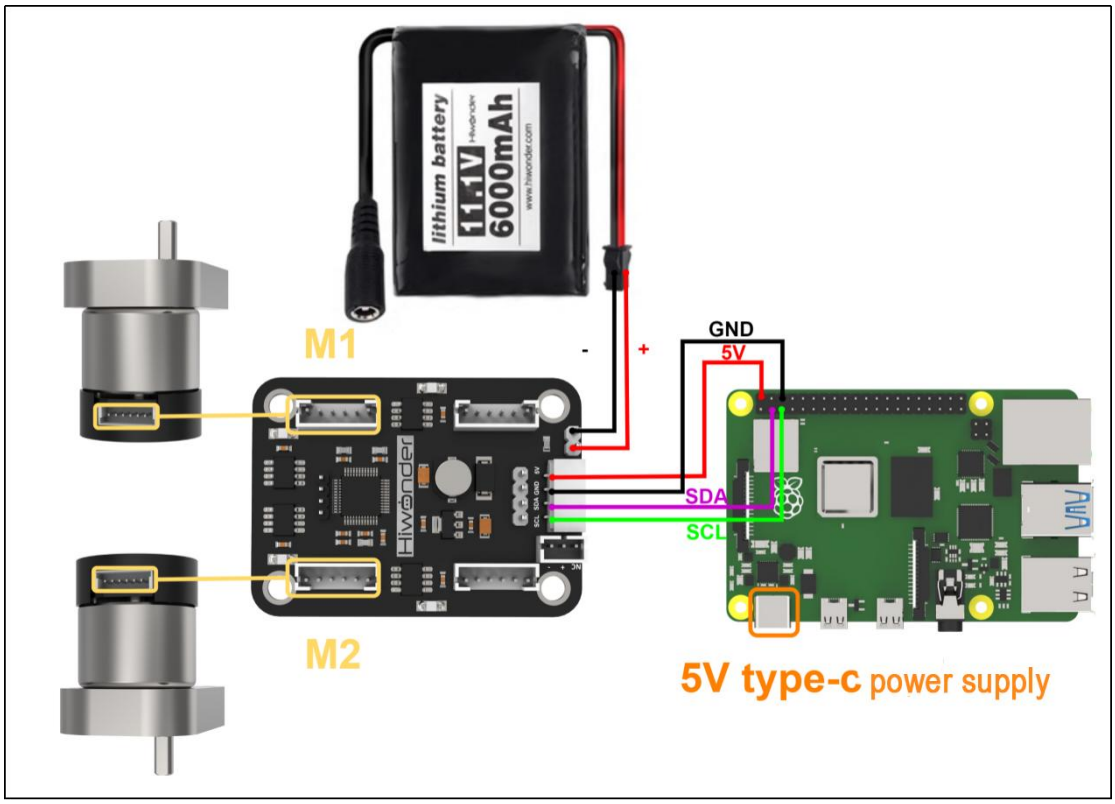
2 Wiring

The example in this section utilizes a Raspberry Pi motherboard and a four-way motor driver module, powered by an 11.1V 6000mAh lithium battery.

1) Connect two DC encoded motors to the M1 and M2 interfaces of four-channel motor driver module.

2) Utilize Dupont wires to connect the IIC interface of the four-channel motor driver module to the corresponding pins on the Raspberry Pi.

3) Connect the positive and negative terminals of the power supply to the power supply interface of the four-channel motor driver module. Here, you can opt to use the positive and negative terminals connected to the 3Pin power interface, or alternatively, solder pins onto the power interface for power supply.



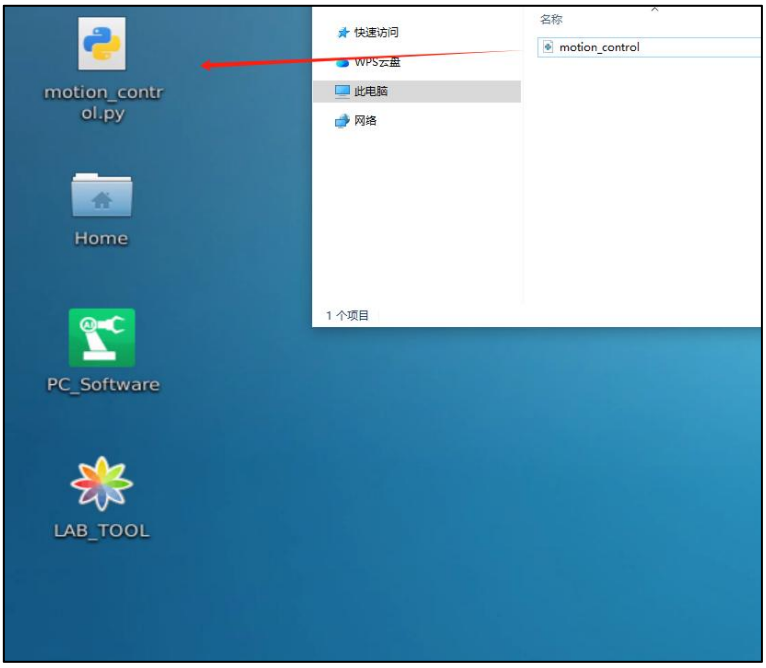
3 Environment Configuration and Program Running

3.1 Environment Configuration

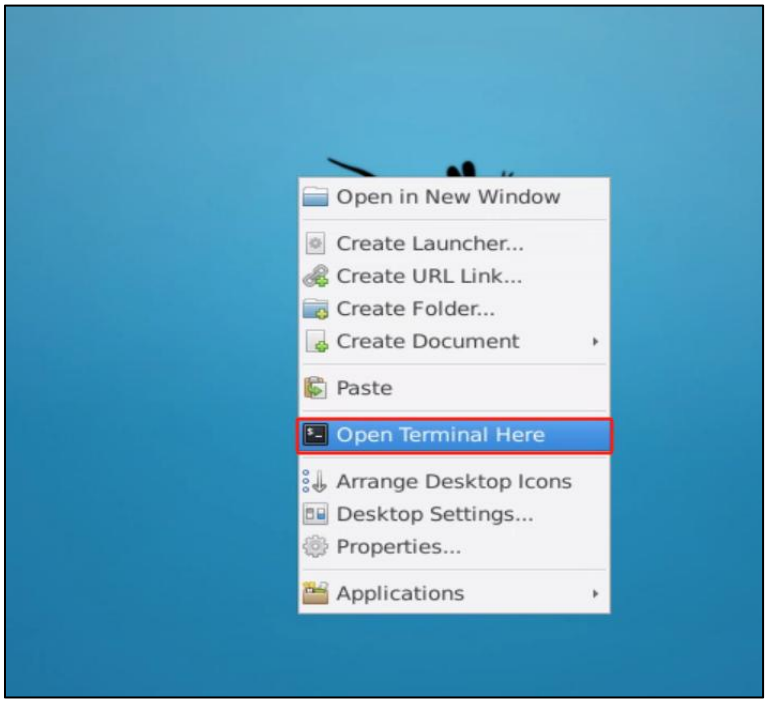
Prior to downloading, ensure that the Arduino IDE is installed on your computer. You can find the software package in the '**2. Software Tools/03 Remote Desktop Connection Tool (Raspberry Pi)**'.

3.2 Program Running

1) After installing the remote desktop, connect to the robot by dragging the 'motion_control.py' file saved in '**3 Programs**' to the Raspberry Pi desktop and the file from the local PC to the NoMachine remote desktop.



2) Right-click and select ‘Open Terminal Here’.



3) Grant permission to the **'motion_control.py' file by executing the command 'sudo chmod +x motion_control.py'** in the terminal.

```
ubuntu@ubuntu:~/Desktop$ sudo chmod +x motion_control.py
```

4) Run the command ‘**python3 motion_control.py**’ to launch the program.

```
ubuntu@ubuntu:~/Desktop$ python3 motion_control.py
[DEBUG] [1682503464.552223]: init_node, name[/car_control_demo], pid[15297]
[DEBUG] [1682503464.559759]: binding to 0.0.0.0 0
[DEBUG] [1682503464.566620]: bound to 0.0.0.0 34949
[DEBUG] [1682503464.572901]: ... service URL is rosrpc://ubuntu:34949
[DEBUG] [1682503464.579134]: [/car_control_demo/get_loggers]: new Service instan
ce
[DEBUG] [1682503464.600523]: ... service URL is rosrpc://ubuntu:34949
[DEBUG] [1682503464.613758]: [/car_control_demo/set_logger_level]: new Service i
nstance
启动程序...
```

5) After the successful execution of the program, the terminal will display a prompt, and the robot car will commence movement within a second.

6) Press the short-cut ‘Ctrl+C’ on the terminal to terminate the program.

Note: If you encounter a situation where the master node is not activated, as shown in the figure below:

```
ubuntu@ubuntu:~/Desktop$ python3 motion_control.py
Unable to register with master node [http://ubuntu:11311]: master may not be run
ning yet. Will keep trying.
```

Simply press **'Ctrl + C'** in the current terminal to exit the program, and then proceed to enter the command to start the program.

---

4 Program Outcome & Program Analysis

4.1 Program Outcome

After executing the program, the chassis moves forward for 2 seconds, then reverses for 2 seconds, followed by a left turn for 2 seconds. Finally, it resumes moving forward for another 2 seconds before coming to a stop.

4.2 Program Analysis

- **Import Necessary Module**

```
import sys
```

```
import rospy
from chassis_control.msg import SetVelocity
```

Begin by importing '**sys**' for processing command-line parameters or performing operating system-related operations;

Then import 'rospy' for writing ROS nodes in Python to control and communicate with robotic systems;

Finally, import the 'SetVelocityBoard' message type from 'chassis_control.msg' for publishing or subscribing to ROS messages related to chassis control.

- **Initiate ROS Node**

```
rospy.init_node('car_control_demo', log_level=rospy.DEBUG)
```

Initiate the node, and name the node '**car_control_demo**'.

- **Create Chassis Control Publisher**

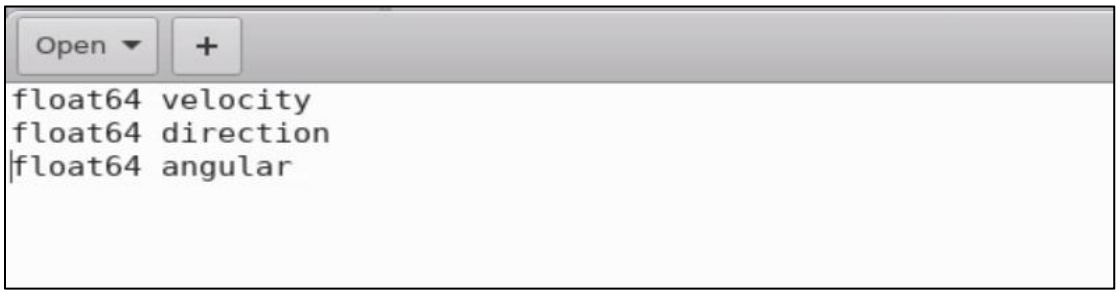
```
set_velocity = rospy.Publisher('/chassis_control/set_velocity', SetVelocity, queue_size=1)
```

Established a ROS publisher to broadcast messages of type 'SetVelocity' to a ROS topic named '/chassis_control/set_velocity'.

■ '/chassis_control/set_velocity': This is the ROS topic for publishing messages.

■ 'SetVelocity': The message type being published.

For details about this message type, you can navigate to the path '/home/ubuntu/armpi_pro/src/chassis_control/msg/' in the system. Double-click to open the 'set_velocity.msg' file, as illustrated in the figure below:



The three parameters mentioned above are all of type 'float64' floating-point numbers, designed for more accurate control of the motor speed and steering angle of the car chassis. 'Velocity' represents the linear speed direction, 'direction' represents the yaw angle of the chassis, and the final 'angular' represents the angular velocity of the yaw angle.

■ 'queue_size=1': This is the publisher's message queue size, and this parameter can be kept as the default.

In the installed system image, the initiated service can be utilized to receive the entire topic message '**/chassis_control/set_velocity**.' Subsequently, the system subscribes to it, allowing the complete topic message to be written into the IIC communication. This, in turn, controls the car chassis to operate based on different speed values.

- **Main Function Controls Chassis Motion**

```
def main():
    print("启动程序...")
    rospy.sleep(1)
    set_velocity.publish(150, 90, 0)# 控制底盘前进，发布底盘控制消息,线速度 150，方向角 90，偏航角速度 0(小于 0，为顺时针方向)
    rospy.sleep(2)
    set_velocity.publish(0, 0, 0)# 停止运动
    rospy.sleep(1)
```

```
        set_velocity.publish(150, 270, 0)# 控制底盘后退，发布底盘控制消息，线速度 150，方向角 270，偏航角速度 0(小于 0，为顺时针方向)
        rospy.sleep(2)
        set_velocity.publish(0, 0, 0)# 停止运动
        rospy.sleep(1)
        set_velocity.publish(0, 90, 0.1)# # 控制底盘逆时针旋转，发布底盘控制消息,线速度 0，方向角 90，偏航角速度 0.1(小于 0，为顺时针方向)
        rospy.sleep(2)
        set_velocity.publish(0, 0, 0)# 停止运动
        rospy.sleep(1)
        set_velocity.publish(150, 90, 0)# 控制底盘前进，发布底盘控制消息,线速度 150，方向角 90，偏航角速度 0(小于 0，为顺时针方向)
        rospy.sleep(2)
        set_velocity.publish(0, 0, 0)# 停止运动
        rospy.sleep(1)
```

The above code snippet represents the main function executed by the program, and it can be used to modify the motion behavior of the car. Let's take the example of 'set_velocity.publish(150, 90, 0)':

In the ROS operating system, the 'rospy.Publisher' object named 'set_velocity,' previously defined, is employed to issue commands for chassis control. The parameters within the brackets are as follows:

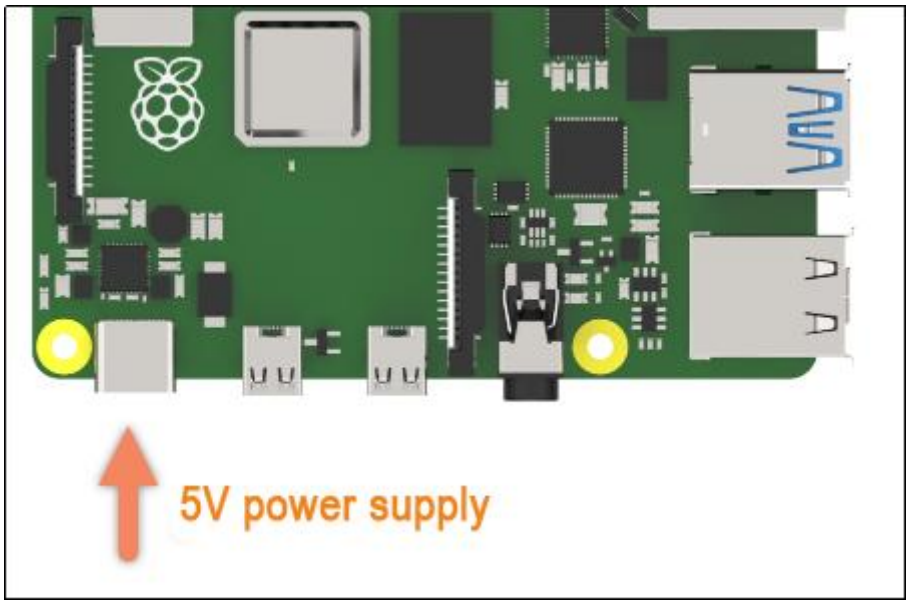
The first parameter, '150,' signifies the motor speed in millimeters per second, with a range of '-300~300.' Negative values indicate reverse motor rotation.

The second parameter, '90,' denotes the direction of the car's movement in degrees, with a range of '0~360.' Here, 90 degrees corresponds to the car's forward direction, while 270 degrees aligns with the car chassis facing backward.

The third parameter, '0,' represents the steering angle of the car, measured in 5-degree increments per second, with a range of '-2~2.' Positive values indicate counterclockwise rotation, while negative values indicate clockwise rotation.

5 Development Notices

1) Considering the rated operating voltage of the Raspberry Pi 4B motherboard is 5V, it's essential to note that the four-channel encoded IIC interface (5V, GND, SCL, SDA) solution from the motor drive module cannot be directly used for power supply. This is because the 5V of this interface only supports voltage input and cannot be used for output. However, using other interfaces of the motor driver module to power the Raspberry Pi GPIO is not recommended. It is advised to adopt a dual power supply solution, providing a separate external power supply to the motor drive module. For instance, using an 11.1V lithium battery (fully charged at 12V) to power the motor drive module, and supplying Raspberry Pi 4B with an additional 5V3A power supply for independent power.



Here, we clarify that powering the Raspberry Pi through the GPIO port is not recommended due to the following reasons:

a. The Raspberry Pi lacks an additional protection circuit for the GPIO port. If power is supplied solely through the GPIO, unstable voltage may lead to current reactions that could potentially damage the built-in CPU, posing a risk to the device.

b. Simultaneously, inadequate external current through the GPIO port may result in improper functioning of the Raspberry Pi 4B, while excessively high current can lead to damage to the built-in components. Therefore, it is advisable to power the Raspberry Pi 4B through the type-C port, using a stable power source such as a power bank with specified current and voltage (5V, 2.5A or 5V, 3A) to ensure the normal operation of the Raspberry Pi 4B motherboard.

c. Using a separate power supply for the Raspberry Pi not only ensures its normal operation but also preserves the 5V and GND pins of the Raspberry Pi, facilitating future expansions of the Raspberry Pi.

2) Coding Pin Wiring Diagram of Raspberry Pi

The diagram below illustrates the physical pin layout of the Raspberry Pi, featuring two encoding methods: BCM encoding and wiringPi encoding. It is evident from the picture that when writing code, you have the flexibility to set different encoding methods within the program for effective programming.

wiringPi code	BCM code	Function name	Physical pin code BOARD		Function name	BCM code	wiringPi code
		3.3V	1	2	5V		
8	2	SDA.1	3	4	5V		
9	3	SCL.1	5	6	GND		
7	4	GPIO.7	7	8	TXD	14	15
		GND	9	10	RXD	15	16
0	17	GPIO.0	11	12	GPIO.1	18	1
2	27	GPIO.2	13	14	GND		
3	22	GPIO.3	15	16	GPIO.4	23	4
		3.3V	17	18	GPIO.5	24	5
12	10	MOSI	19	20	GND		
13	9	MISO	21	22	GPIO.6	25	6
14	11	SCLK	23	24	CE0	8	10
		GND	25	26	CE1	7	11
30	0	SDA.0	27	28	SCL.0	1	31
21	5	GPIO.21	29	30	GND		
22	6	GPIO.22	31	32	GPIO.26	12	26
23	13	GPIO.23	33	34	GND		
24	19	GPIO.24	35	36	GPIO.27	16	27
25	26	GPIO.25	37	38	GPIO.28	20	28
		GND	39	40	GPIO.29	21	29