2. Development Tutorials (Raspberry Pi)

After the program is downloaded, the chassis will execute actions in the following order:

① **Move forward and backward.**

② **Move left and right (sideways).**

③ **Rotate left and right in place.**

④ **Move left-forward and right-backward.**

⑤ **Drift left and right in place.**

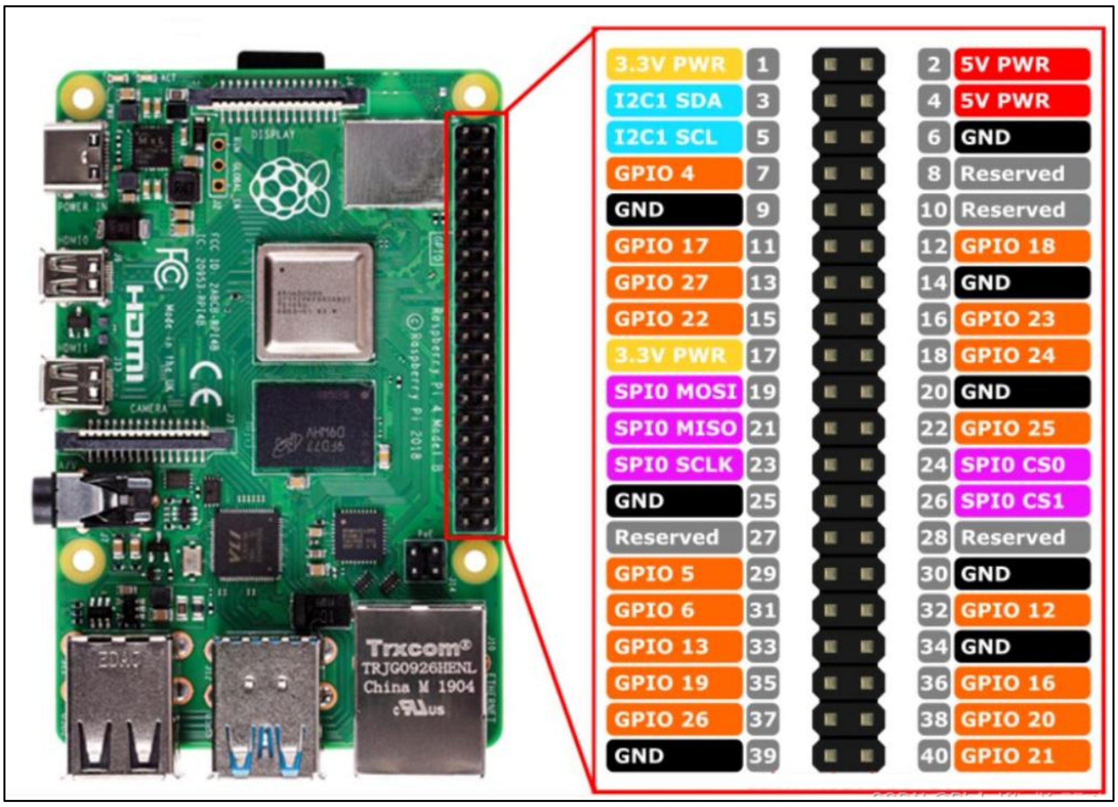
Each action will be executed for 2 seconds, with a 1-second interval between each action in the sequence.

1 Hardware Introduction

1.1 Raspberry Pi 4B

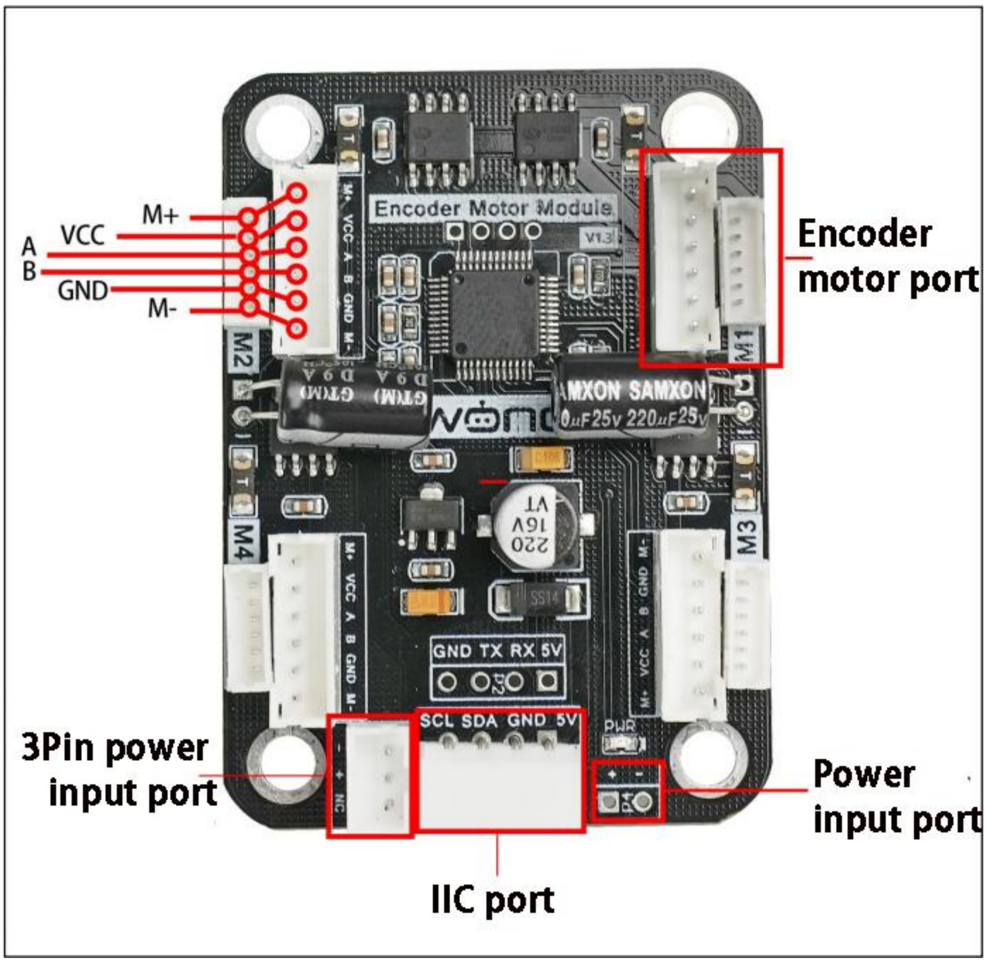
The Raspberry Pi 4B is a compact embedded computer equipped with a 500MHz GPU. It comes with options of 1GB, 2GB, or 4GB LPDDR4 RAM, a Gigabit Ethernet port, Bluetooth 5.0, USB 3.0 ports, and a microHDMI interface. Based on these specifications, the Raspberry Pi offers a robust hardware environment for programming and development.

Below is the pinout diagram for the Raspberry Pi 4B, which will be referenced in the upcoming wiring and development process.



1.2 4-Channel Encoder Motor Driver

This motor driver module is designed to work with microcontrollers to drive TT motors or magnetic encoder motors. Each channel is equipped with an YX-4055AM motor driver chip, which supports a voltage range of DC 3V-12V, depending on the voltage of the connected motor. The interface layout is shown in the diagram below.



Interface Pin Description:

Interface Type	Pin Number	Description
Encoder Motor Interface	GND	Hall power negative
	A	Phase A pulse signal output
	B	Phase B pulse signal output
	VCC	Hall power positive
	M+	Motor power positive
	M-	Motor power negative
	Note: The voltage between VCC and GND is determined	

	by the power supply voltage of the microcontroller, typically 3.3V or 5V. When the motor shaft rotates clockwise, the A channel pulse signal precedes the B channel signal. When the shaft rotates counterclockwise, the A channel signal follows the B channel. The voltage between M+ and M- depends on the motor voltage.	
IIC	SCL	Clock line
	SDA	Bidirectional data line
	GND	Ground
	5V	5V DC output
3Pin power interface	-	Power negative input
	+	Power positive input
	NC	Not connected
Power interface	+	Power positive input
	-	Power negative

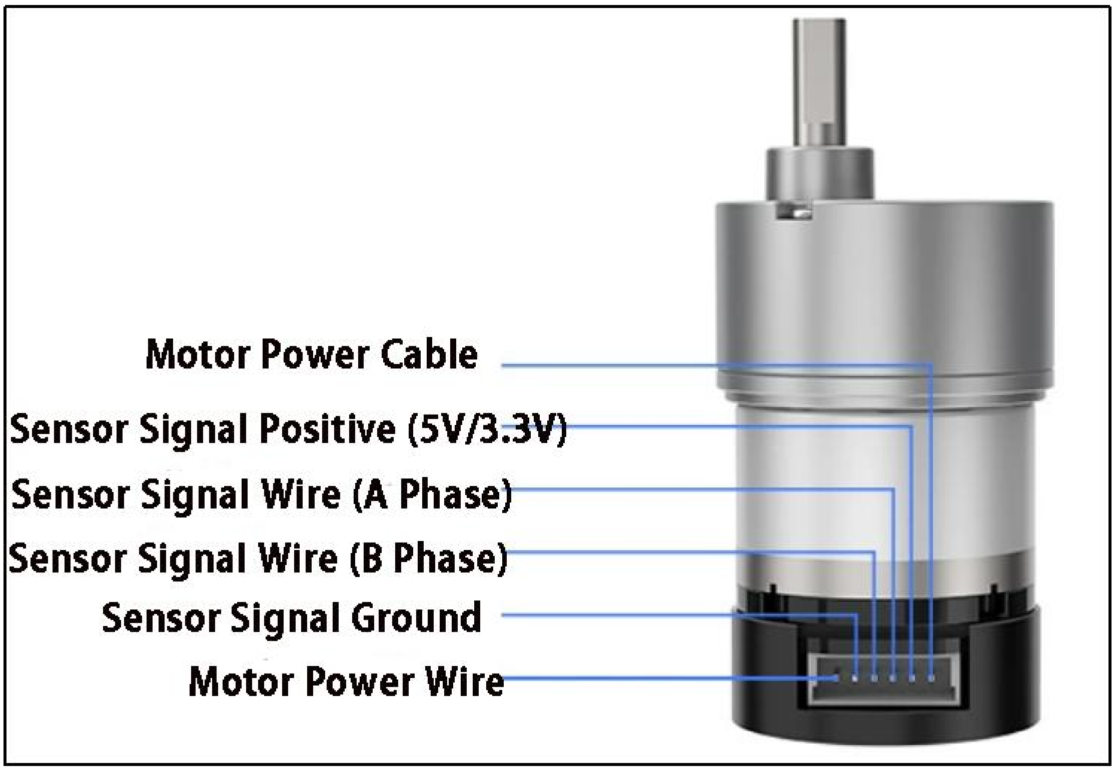
1.3 Encoder Reduction Motor

The chassis is equipped with a motor model JGB37-528R131-08. The designation details are as follows:

- J: DC motor
- GB: Offset output shaft
- 37: Gearbox diameter (mm)

- 528: Motor model number
- R131: Reduction ratio of 131:1
- 08: Rated voltage of 8V

The interface details are shown in the diagram below:

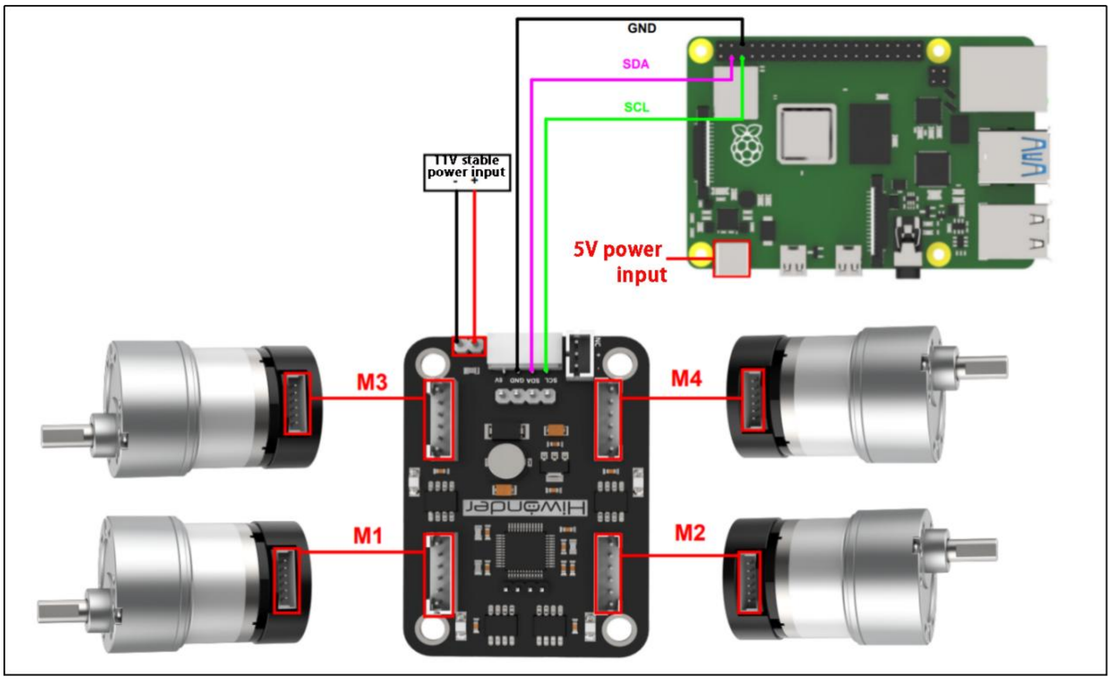

Motor Power Cable
Sensor Signal Positive (5V/3.3V)
Sensor Signal Wire (A Phase)
Sensor Signal Wire (B Phase)
Sensor Signal Ground
Motor Power Wire

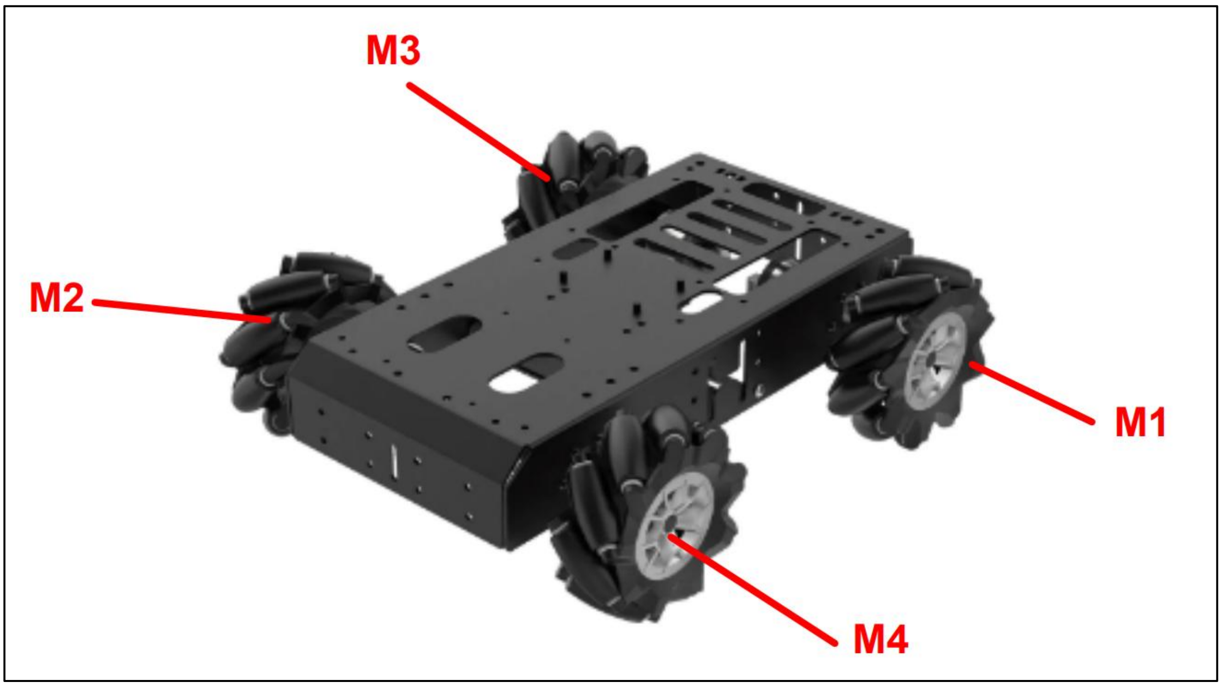
The Hall encoder disk is mounted coaxially with the motor. As the motor rotates, the Hall sensor detects and outputs a series of pulse signals. To determine the direction of rotation, two square wave signals with a certain phase difference are typically output.

2 Wiring

In this example, the Raspberry Pi controller is used along with the 4-channel encoder motor driver module, powered by a 7.4V 3500mAh lithium battery. The wiring diagram is shown below.

In the diagram, the left motor on the front side of the vehicle is labeled M1, and the right motor is labeled M3. The remaining wiring content is a rendered image, where the pin layout matches the physical connections. Please note that the actual application effect is not affected by this rendering, and the real physical setup should be used as the reference.





3 Environment Setup and Program Execution

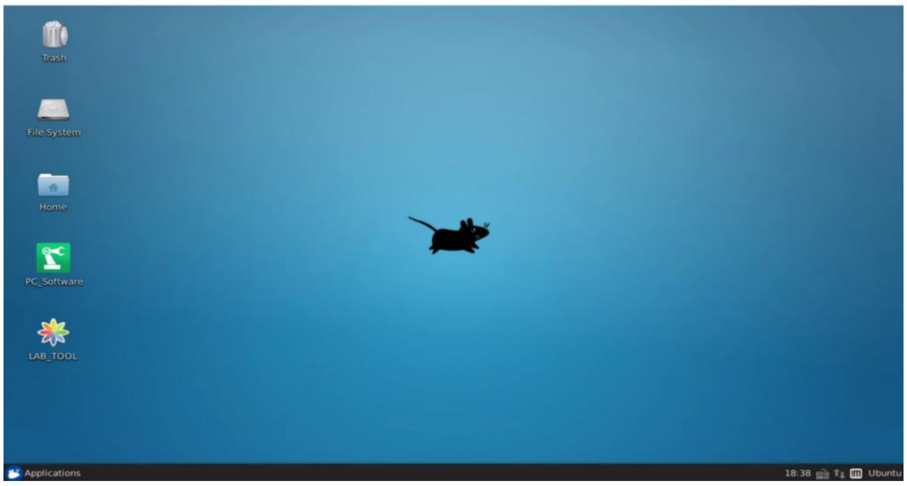
3.1 Environment Setup

Install NoMachine on the computer. The software package can be found under "2. Software Tools\03 Remote Desktop Connection Tools (for Raspberry Pi)". Follow the instructions in the "NoMachine Installation and Usage" document in the folder for installation and configuration.

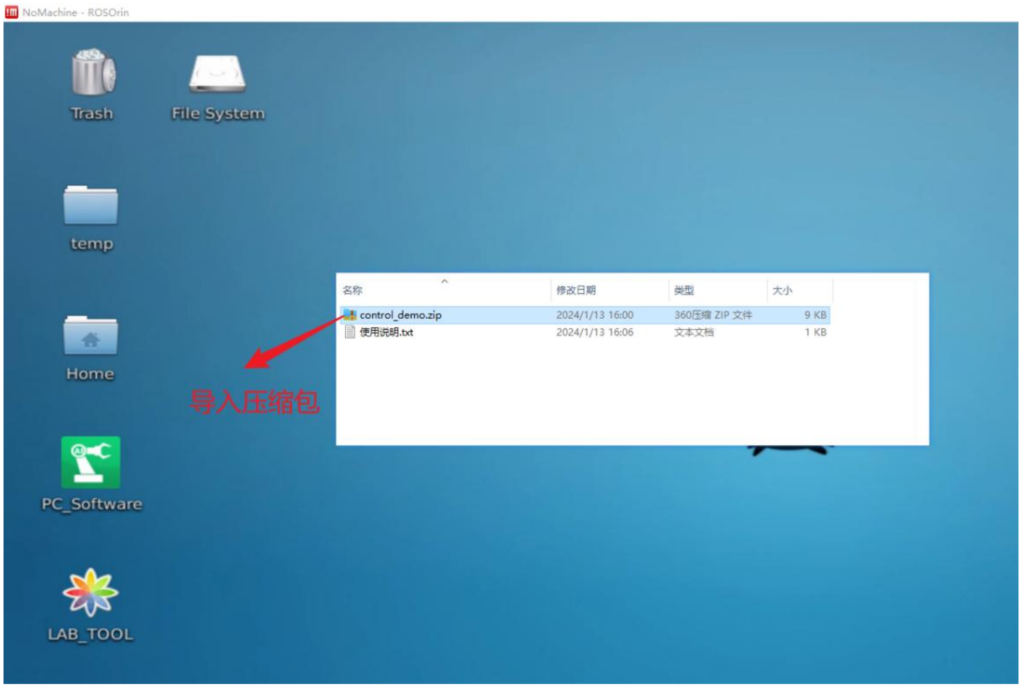
3.2 Executing the Program

Place the Program Files in the Specified Directory

1) Open NoMachine and connect to the robot.

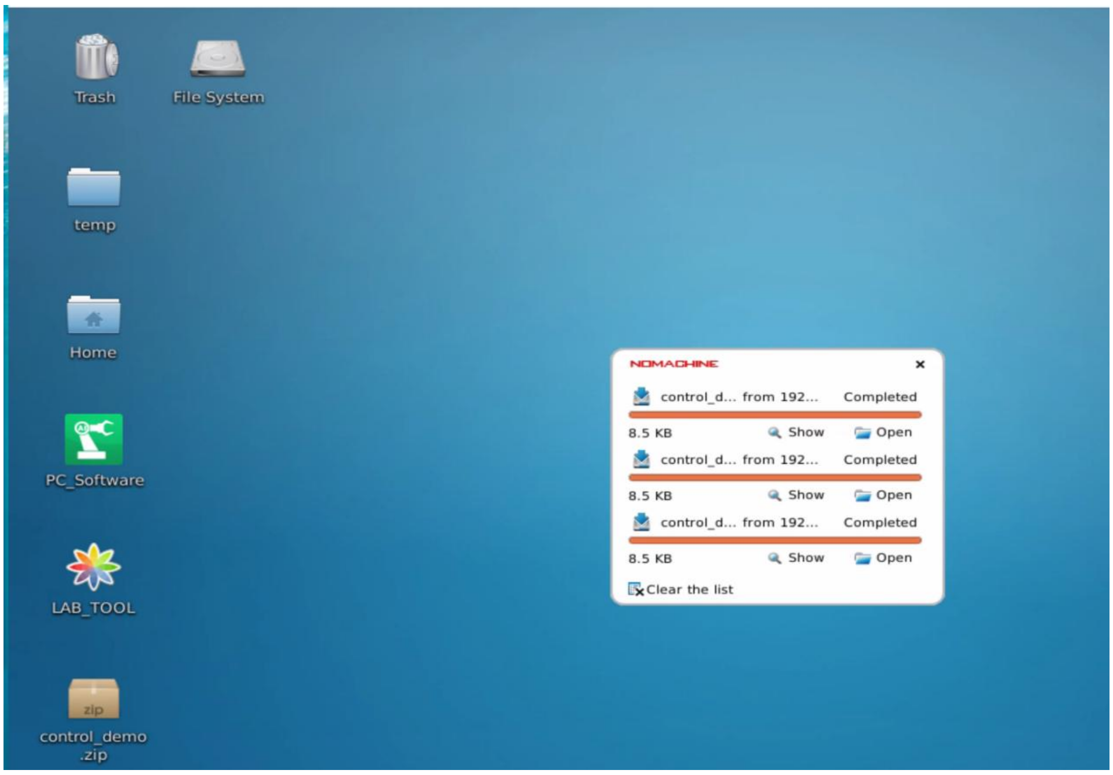


2) Drag the demo file from the path "1. Tutorials\04 Raspberry Pi\3 Programs" to the desktop, as shown in the figure below. Wait for the program to load on the desktop.

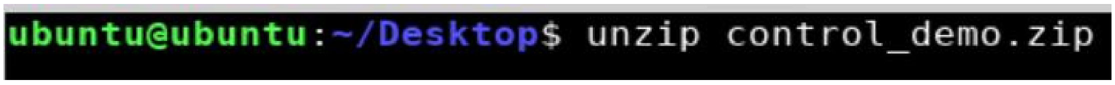


名称	修改日期	类型	大小
control_demo.zip	2024/1/13 16:00	360压缩 ZIP 文件	9 KB
使用说明.txt	2024/1/13 16:06	文本文档	1 KB

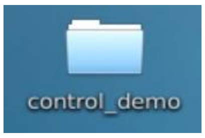
3) Wait for the program to load onto the desktop.



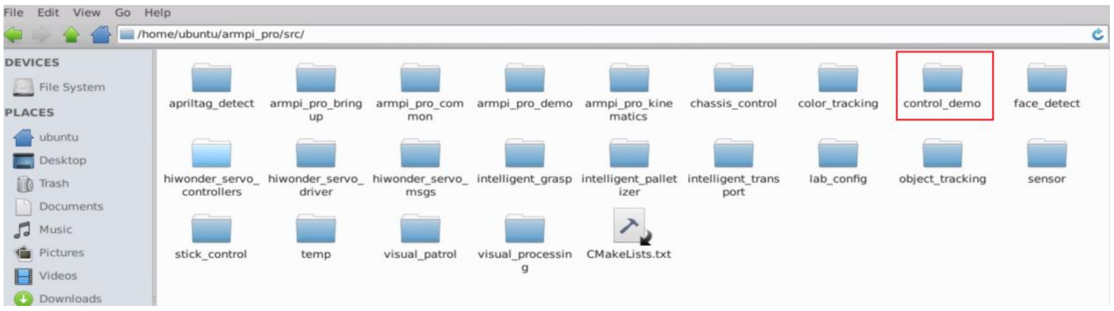
4) Extract the package to the desktop by entering the command: unzip control_demo.zip.



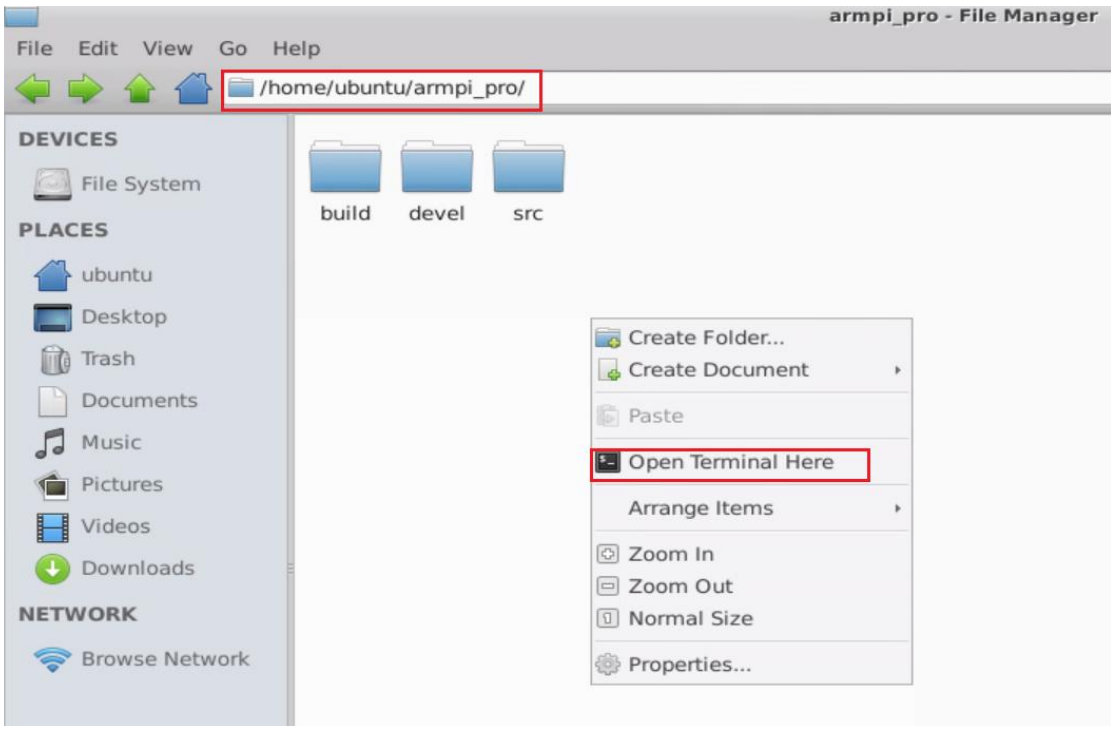
```
ubuntu@ubuntu:~/Desktop$ unzip control_demo.zip
```



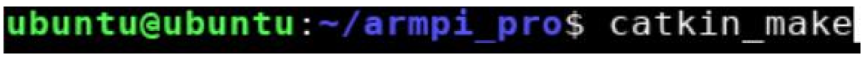
5) Copy the package to the /home/ubuntu/armpi_pro/src/ directory.



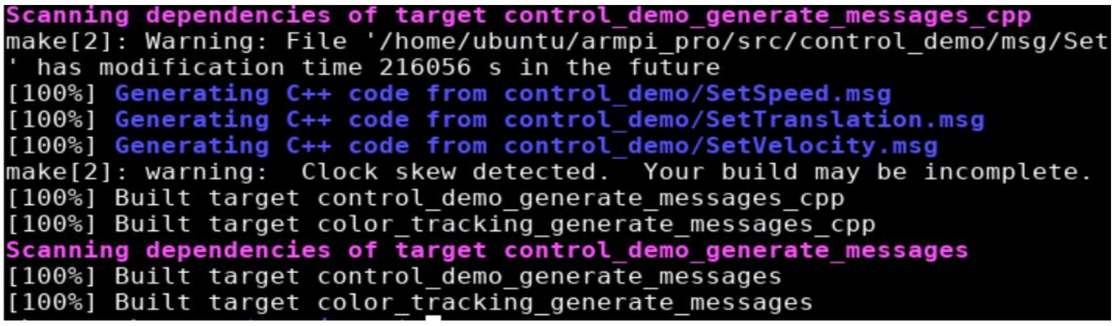
6) Open a terminal in the /home/ubuntu/armpi_pro/ directory.



7) Enter the command catkin_make to compile the package. The compilation will take approximately 4 minutes to complete.



```
ubuntu@ubuntu:~/armpi_pro$ catkin_make
```

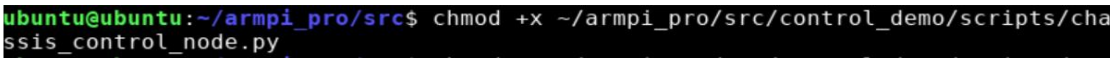


```
Scanning dependencies of target control_demo_generate_messages_cpp
make[2]: Warning: File '/home/ubuntu/armpi_pro/src/control_demo/msg/Set
' has modification time 216056 s in the future
[100%] Generating C++ code from control_demo/SetSpeed.msg
[100%] Generating C++ code from control_demo/SetTranslation.msg
[100%] Generating C++ code from control_demo/SetVelocity.msg
make[2]: warning:  Clock skew detected.  Your build may be incomplete.
[100%] Built target control_demo_generate_messages_cpp
[100%] Built target color_tracking_generate_messages_cpp
Scanning dependencies of target control_demo_generate_messages
[100%] Built target control_demo_generate_messages
[100%] Built target color_tracking_generate_messages
```

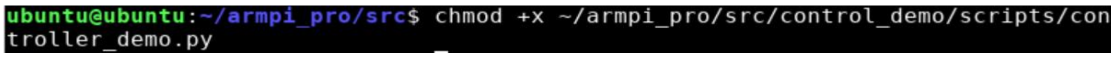
8) Enter the following commands to grant execute permissions to the programs:

chmod +x ~/armpi_pro/src/control_demo/scripts/chassis_control_node.py

chmod +x ~/armpi_pro/src/control_demo/scripts/controller_demo.py



```
ubuntu@ubuntu:~/armpi_pro/src$ chmod +x ~/armpi_pro/src/control_demo/scripts/cha
ssis_control_node.py
```

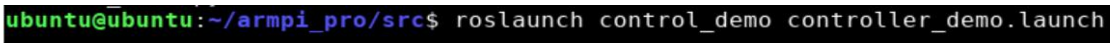


```
ubuntu@ubuntu:~/armpi_pro/src$ chmod +x ~/armpi_pro/src/control_demo/scripts/con
troller_demo.py
```

9) Open a new terminal and enter the following command:

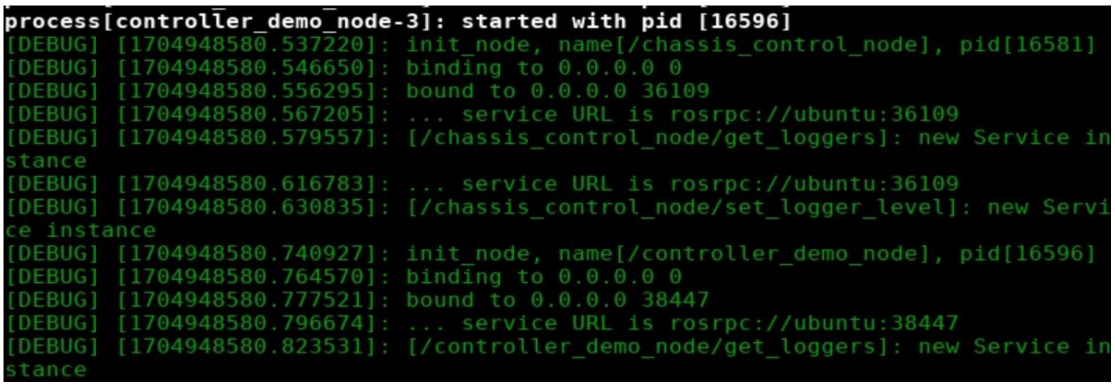
“roslaunch control_demo controller_demo.launch”

This will start the robot's control program. Ensure that I2C communication is connected and the battery is properly powered to avoid errors during program execution.



```
ubuntu@ubuntu:~/armpi_pro/src$ roslaunch control_demo controller_demo.launch
```

10) Wait for the program to load. Once the loading process is complete, the robot will perform the corresponding movements as defined in the program.



```
process[controller_demo_node-3]: started with pid [16596]
[DEBUG] [1704948580.537220]: init_node, name[/chassis_control_node], pid[16581]
[DEBUG] [1704948580.546650]: binding to 0.0.0.0 0
[DEBUG] [1704948580.556295]: bound to 0.0.0.0 36109
[DEBUG] [1704948580.567205]: ... service URL is rosrpc://ubuntu:36109
[DEBUG] [1704948580.579557]: [/chassis_control_node/get_loggers]: new Service in
stance
[DEBUG] [1704948580.616783]: ... service URL is rosrpc://ubuntu:36109
[DEBUG] [1704948580.630835]: [/chassis_control_node/set_logger_level]: new Servi
ce instance
[DEBUG] [1704948580.740927]: init_node, name[/controller_demo_node], pid[16596]
[DEBUG] [1704948580.764570]: binding to 0.0.0.0 0
[DEBUG] [1704948580.777521]: bound to 0.0.0.0 38447
[DEBUG] [1704948580.796674]: ... service URL is rosrpc://ubuntu:38447
[DEBUG] [1704948580.823531]: [/controller_demo_node/get_loggers]: new Service in
stance
```

11) To exit the program, press Ctrl+C in the terminal. This will safely terminate the running program.

3.3 Program Outcome

After the program is loaded, the robot chassis will execute the following actions in the preset order:

① **Move forward and backward**

② **Move left and right laterally**

③ **Rotate left and right in place**

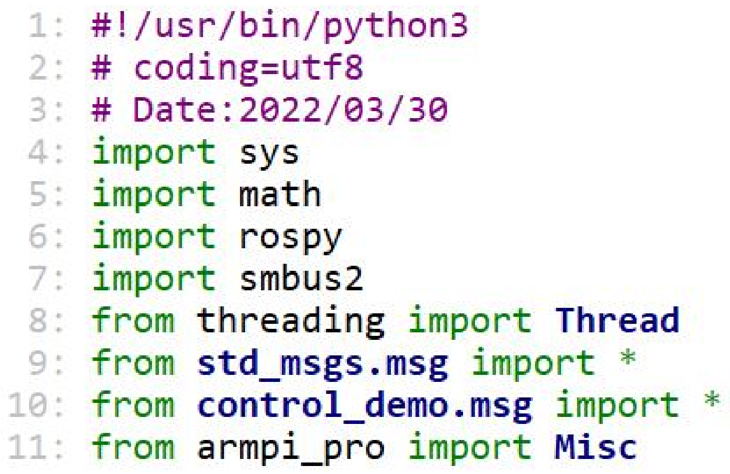
④ **Move diagonally forward-left and backward-right**

⑤ **Drift left and right in place**

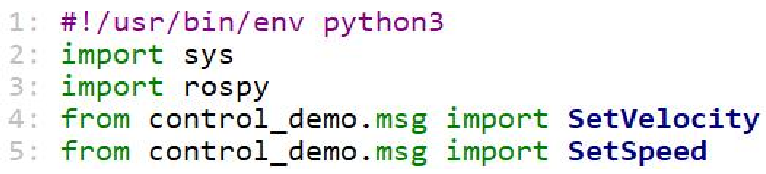
Each action lasts for **2 seconds**, with a **1-second** interval between actions in sequence.

4 Brief Analysis of Example Program

- **Importing Necessary Modules**



```python
#!/usr/bin/python3
# coding=utf8
# Date:2022/03/30
import sys
import math
import rospy
import smbus2
from threading import Thread
from std_msgs.msg import *
from control_demo.msg import *
from armpi_pro import Misc
```



```python
#!/usr/bin/env python3
import sys
import rospy
from control_demo.msg import SetVelocity
from control_demo.msg import SetSpeed
```

The initial setup and library imports in the chassis_control_node.py and controller_demo.py scripts include several critical components. Key points to note are as follows:

① Shebang Line (#!/usr/bin/env python3): This line specifies the interpreter to be used for executing the script. It searches for the python3 interpreter in the system's PATH environment variable and ensures the script is executed with the correct interpreter.

② sys Module: Used for interacting with the system, enabling system-level control and operations.

③ rospy Module: A Python library for ROS (Robot Operating System) that facilitates starting ROS services and executing corresponding functionalities.

④ smbus2 Library: An open-source Python library for I2C communication, enabling control of the robot's motors.

It can be installed using the following command: pip install smbus2

⑤ from control_demo.msg import SetVelocity

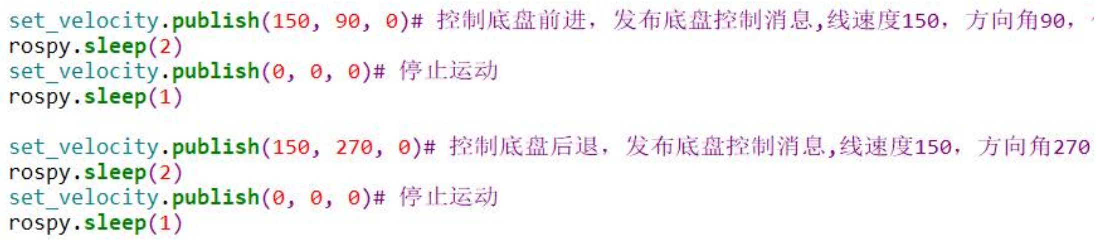
from control_demo.msg import SetSpeed

These imports bring in the SetVelocity and SetSpeed message types, which are crucial for defining and handling motor control commands within the program.

4.1 Motion Functions

1) Forward and Reverse Motion

- In the command publish(150, 90, 0),
- 150 refers to the linear speed of 150mm per second,
- 90 indicates the robot's direction at 90° (moving forward),
- 270 would indicate a reverse direction,
- The last value, 0, refers to zero angular velocity (no rotation).
- This command is implemented in controller_demo.py.

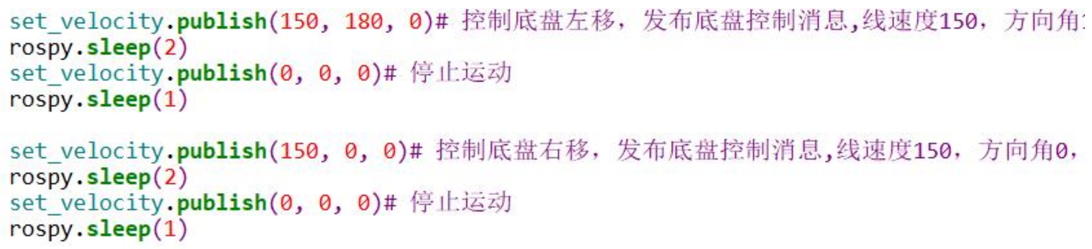


```
set_velocity.publish(150, 90, 0)# 控制底盘前进，发布底盘控制消息,线速度150，方向角90，
rospy.sleep(2)
set_velocity.publish(0, 0, 0)# 停止运动
rospy.sleep(1)

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
rospy.sleep(2)
set_velocity.publish(0, 0, 0)# 停止运动
rospy.sleep(1)
```

2) Left and Right Translation

- For the leftward translation, publish(150, 180, 0) is used, where 180 indicates a direction toward the left.
- The direction angles from 90° to 270° correspond to movements in various angles on the left side of the robot.

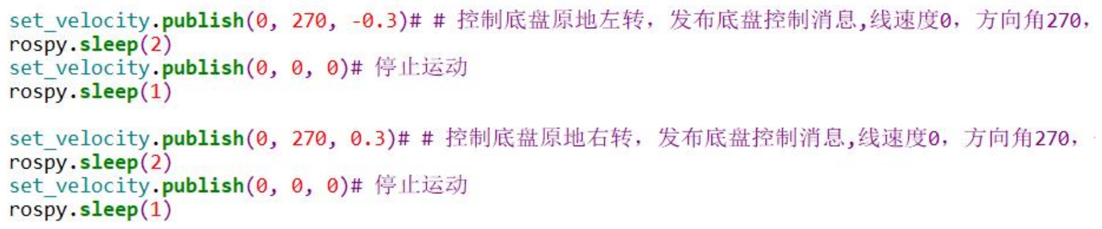
- When the direction is 0°, the robot will translate right, and 270° to 360°, along with 0° to 90°, indicate movements in various angles on the right side of the robot.



```
set_velocity.publish(150, 180, 0)# 控制底盘左移，发布底盘控制消息,线速度150，方向角
rospy.sleep(2)
set_velocity.publish(0, 0, 0)# 停止运动
rospy.sleep(1)

set_velocity.publish(150, 0, 0)# 控制底盘右移，发布底盘控制消息,线速度150，方向角0,
rospy.sleep(2)
set_velocity.publish(0, 0, 0)# 停止运动
rospy.sleep(1)
```

3) Stationary Left and Right Turns

- In the command publish(0, 270, -0.3),
- 0 refers to the linear speed, which is zero (no forward motion),
- 270 is the angle for the turn,
- -0.3 is the angular velocity, where a negative value indicates a leftward turn (clockwise rotation).
- This is implemented for rotating the robot in place.

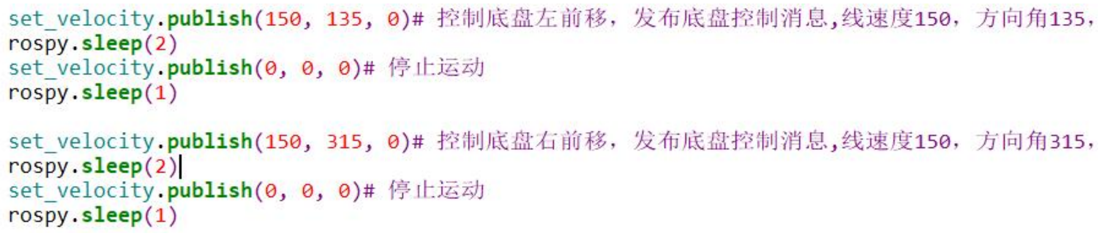


```
set_velocity.publish(0, 270, -0.3)# # 控制底盘原地左转，发布底盘控制消息,线速度0，方向角270,
rospy.sleep(2)
set_velocity.publish(0, 0, 0)# 停止运动
rospy.sleep(1)

set_velocity.publish(0, 270, 0.3)# # 控制底盘原地右转，发布底盘控制消息,线速度0，方向角270,
rospy.sleep(2)
set_velocity.publish(0, 0, 0)# 停止运动
rospy.sleep(1)
```

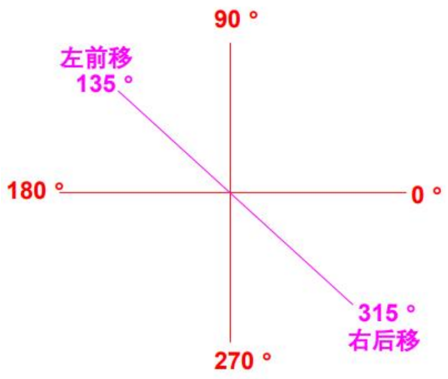
4) Left Forward and Right Reverse Movements

- For the left forward and right reverse motion, publish(150, 135, 0) is used.
- 150 refers to the linear speed of 150mm per second,
- 135° indicates moving 45° leftward (diagonal forward-left direction).
- Similarly, publish(150, 315, 0) would make the robot move 45° rightward in

the opposite diagonal (reverse-right direction).

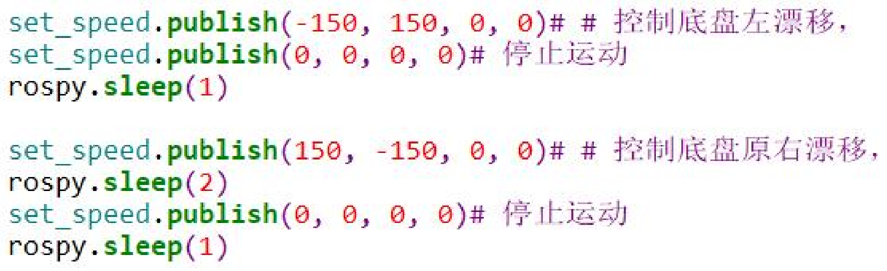


```
set_velocity.publish(150, 135, 0)# 控制底盘左前移，发布底盘控制消息,线速度150，方向角135，
rospy.sleep(2)
set_velocity.publish(0, 0, 0)# 停止运动
rospy.sleep(1)

set_velocity.publish(150, 315, 0)# 控制底盘右前移，发布底盘控制消息,线速度150，方向角315，
rospy.sleep(2)
set_velocity.publish(0, 0, 0)# 停止运动
rospy.sleep(1)
```



5) Left and Right Drifting

- The command publish(-150, 150, 0, 0) is used to perform drifting.
- -150 and 150 represent the speed of the robot's left and right wheels respectively.
- The values v1, v4, v2, v3 represent the left-back wheel, right-back wheel, left-front wheel, and right-front wheel, respectively.
- The sequence of values indicates the desired speed for each wheel to achieve drifting movement.

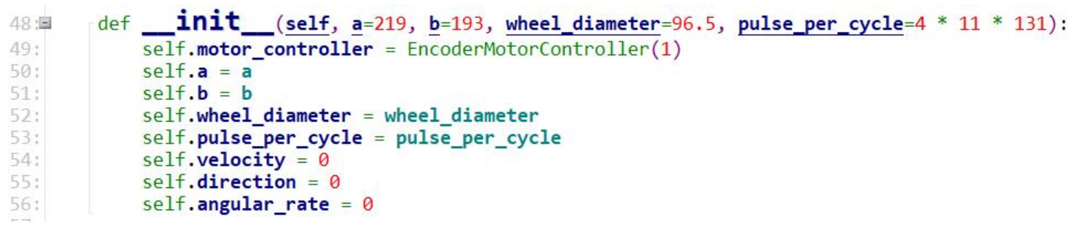


```
set_speed.publish(-150, 150, 0, 0)# # 控制底盘左漂移，
set_speed.publish(0, 0, 0, 0)# 停止运动
rospy.sleep(1)

set_speed.publish(150, -150, 0, 0)# # 控制底盘原右漂移，
rospy.sleep(2)
set_speed.publish(0, 0, 0, 0)# 停止运动
rospy.sleep(1)
```

4.2 Motion Calculation

1) Initialization of Chassis Motion Parameters

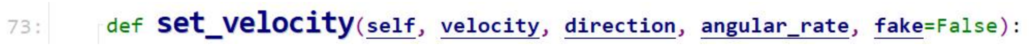
- a = 219: This represents the distance between the left and right wheels of the chassis, 219mm.
- b = 193: This represents the distance between the front and rear wheels, 193mm.
- pulse_per_cycle = 4 * 11 * 131: This calculates the total number of pulse changes per second. The 4x frequency, 11 lines, and 131 reduction ratio determine how pulses are generated during motor operation.



```python
def __init__(self, a=219, b=193, wheel_diameter=96.5, pulse_per_cycle=4 * 11 * 131):
    self.motor_controller = EncoderMotorController(1)
    self.a = a
    self.b = b
    self.wheel_diameter = wheel_diameter
    self.pulse_per_cycle = pulse_per_cycle
    self.velocity = 0
    self.direction = 0
    self.angular_rate = 0
```

2) Motion Calculation in set_velocity Function

- The set_velocity function performs motion calculation, determining the speed of each motor for different motion types.
- velocity: Linear speed, measured in millimeters per second (mm/s).
- direction: The direction of movement, measured in degrees (0~360 degrees), where 270° represents moving forward.
- angular_rate: The angular velocity, representing the rotation speed of the chassis.

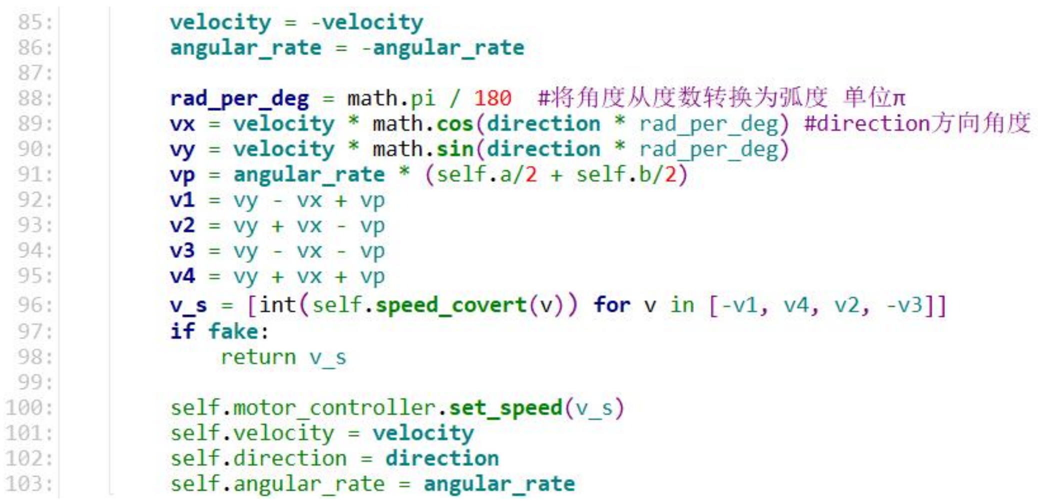


```python
def set_velocity(self, velocity, direction, angular_rate, fake=False):
```

3) Conversion from Degrees to Radians

- rad_per_deg: The conversion factor from degrees to radians, since math.cos and math.sin functions use radians.

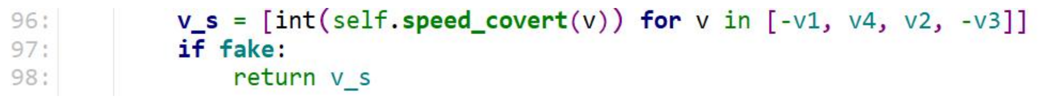
- Next, the horizontal (vx) and vertical (vy) speed components are calculated based on the input linear speed (velocity) and direction (direction).
- vp: A variable that is determined by the rotational speed (angular_rate), the distance between the left and right wheels (self.a), and the distance between the front and rear wheels (self.b). This is used to calculate the speed of the chassis rotation.



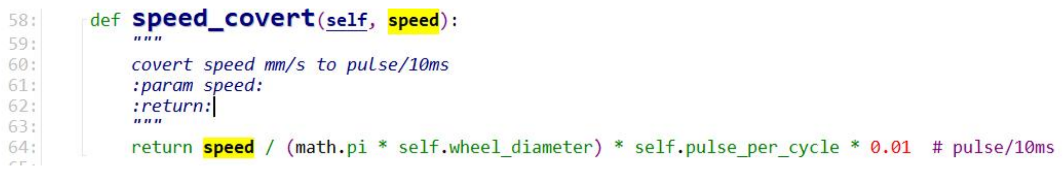
```
        velocity = -velocity
        angular_rate = -angular_rate

        rad_per_deg = math.pi / 180  #将角度从度数转换为弧度 单位π
        vx = velocity * math.cos(direction * rad_per_deg) #direction方向角度
        vy = velocity * math.sin(direction * rad_per_deg)
        vp = angular_rate * (self.a/2 + self.b/2)
        v1 = vy - vx + vp
        v2 = vy + vx - vp
        v3 = vy - vx - vp
        v4 = vy + vx + vp
        v_s = [int(self.speed_covert(v)) for v in [-v1, v4, v2, -v3]]
        if fake:
            return v_s

        self.motor_controller.set_speed(v_s)
        self.velocity = velocity
        self.direction = direction
        self.angular_rate = angular_rate
```

4) Conversion of Linear Speed to Wheel Speed

- The circumference of each wheel is calculated as math.pi * self.wheel_diameter.
- The wheel speed is then converted to the number of pulses generated per rotation.
- pulse_per_cycle = 4 * 11 * 131 = 5764 is the number of pulses generated by the motor per full wheel rotation.
- By multiplying by a 10ms time interval, the number of pulses for the motor in 10ms is calculated. This conversion is essential because the motor controller and encoder use pulses to quantify and control motor speed.



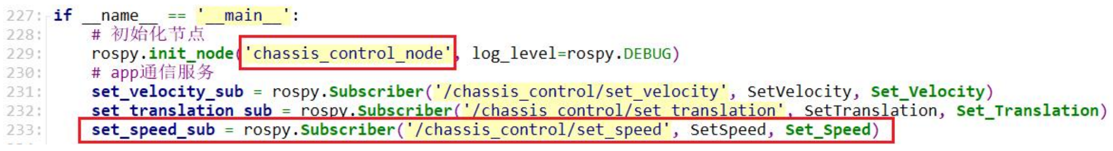
```
        v_s = [int(self.speed_covert(v)) for v in [-v1, v4, v2, -v3]]
        if fake:
            return v_s
```



```
def speed_covert(self, speed):
    """
    covert speed mm/s to pulse/10ms
    :param speed:
    :return:
    """
    return speed / (math.pi * self.wheel_diameter) * self.pulse_per_cycle * 0.01  # pulse/10ms
```

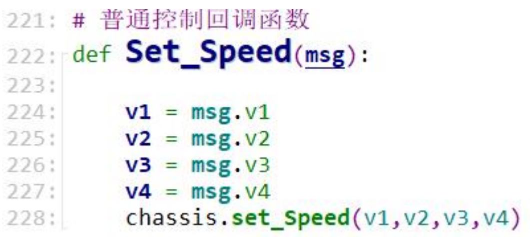
4.3 Setting Up Drifting Motion

1) In the chassis_control_node.py program, add a topic subscription and initialize the "chassis_control_node" node. Then, set up the topic subscription:



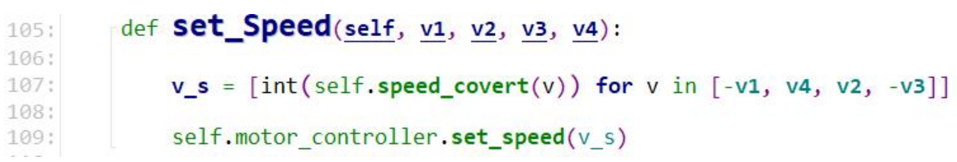
```
if __name__ == '__main__':
    # 初始化节点
    rospy.init_node('chassis_control_node', log_level=rospy.DEBUG)
    # app通信服务
    set_velocity_sub = rospy.Subscriber('/chassis_control/set_velocity', SetVelocity, Set_Velocity)
    set_translation_sub = rospy.Subscriber('/chassis_control/set_translation', SetTranslation, Set_Translation)
    set_speed_sub = rospy.Subscriber('/chassis_control/set_speed', SetSpeed, Set_Speed)
```

2) Add the Set_Speed(msg) callback function to handle the incoming messages.



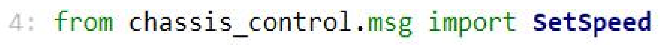
```
# 普通控制回调函数
def Set_Speed(msg):

    v1 = msg.v1
    v2 = msg.v2
    v3 = msg.v3
    v4 = msg.v4
    chassis.set_Speed(v1,v2,v3,v4)
```

3) In the MecanumChassis class, implement the set_Speed(self, v1, v2, v3, v4) function to convert the velocity values and set the speeds of each motor.

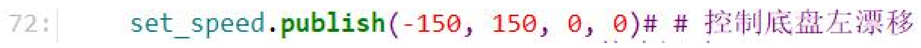


```
def set_Speed(self, v1, v2, v3, v4):

    v_s = [int(self.speed_covert(v)) for v in [-v1, v4, v2, -v3]]

    self.motor_controller.set_speed(v_s)
```

4) In the controller_demo.py program, import the SetSpeed message and publish the speeds for the two rear motors.

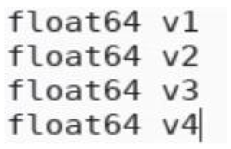


```
from chassis_control.msg import SetSpeed
```



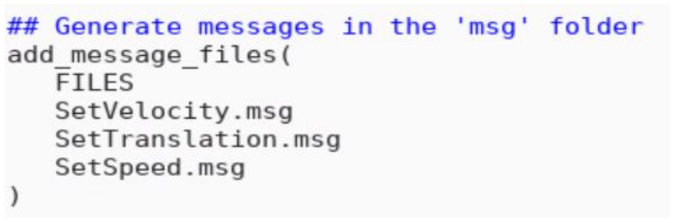
```
    set_speed.publish(-150, 150, 0, 0)# # 控制底盘左漂移
```

5) Add a new SetSpeed.msg file in the path /home/ubuntu/armpi_pro/src/chassis_control/msg/. The content of the message is as follows:



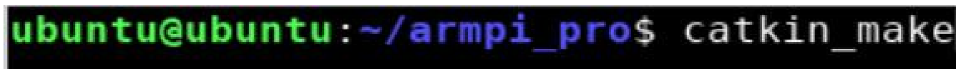
```
float64 v1
float64 v2
float64 v3
float64 v4
```

6) Modify the CMakeLists.txt file in the /home/ubuntu/armpi_pro/src/control_demo/ path to include the SetSpeed.msg message.



```
## Generate messages in the 'msg' folder
add_message_files(
   FILES
   SetVelocity.msg
   SetTranslation.msg
   SetSpeed.msg
)
```

7) The provided example is already set up, but if new messages are added or changes are made, you need to navigate to the /home/ubuntu/armpi_pro directory and run the following command to compile the changes:



```
ubuntu@ubuntu:~/armpi_pro$ catkin_make
```

4.4 I2C Communication for Motor Speed Control

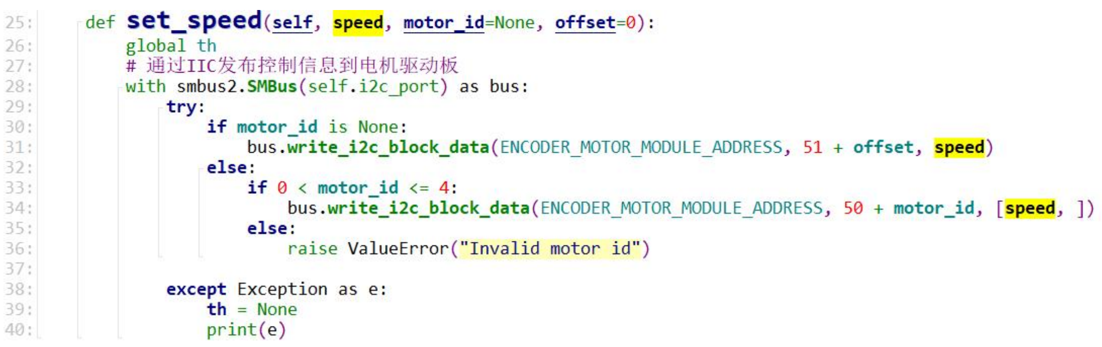
The smbus2.SMBus is used to create an I2C communication instance, allowing communication between the Raspberry Pi and the four-channel encoder motor module via the SMBus/I2C interface.

1. If motor_id is not specified, the function will send the same speed command to all motors.

2. If a motor_id is specified, the function will send the speed command only to the specific motor. The motor_id value should be between 1 and 4.

The speed command is sent to the motor control module using the bus.write_i2c_block_data function. The motor module's I2C address is defined as ENCODER_MOTOR_MODULE_ADDRESS = 0x34.

Here's an example of how this can be implemented:



```
    def set_speed(self, speed, motor_id=None, offset=0):
        global th
        # 通过IIC发布控制信息到电机驱动板
        with smbus2.SMBus(self.i2c_port) as bus:
            try:
                if motor_id is None:
                    bus.write_i2c_block_data(ENCODER_MOTOR_MODULE_ADDRESS, 51 + offset, speed)
                else:
                    if 0 < motor_id <= 4:
                        bus.write_i2c_block_data(ENCODER_MOTOR_MODULE_ADDRESS, 50 + motor_id, [speed, ])
                    else:
                        raise ValueError("Invalid motor id")

            except Exception as e:
                th = None
                print(e)
```

5 Development Considerations

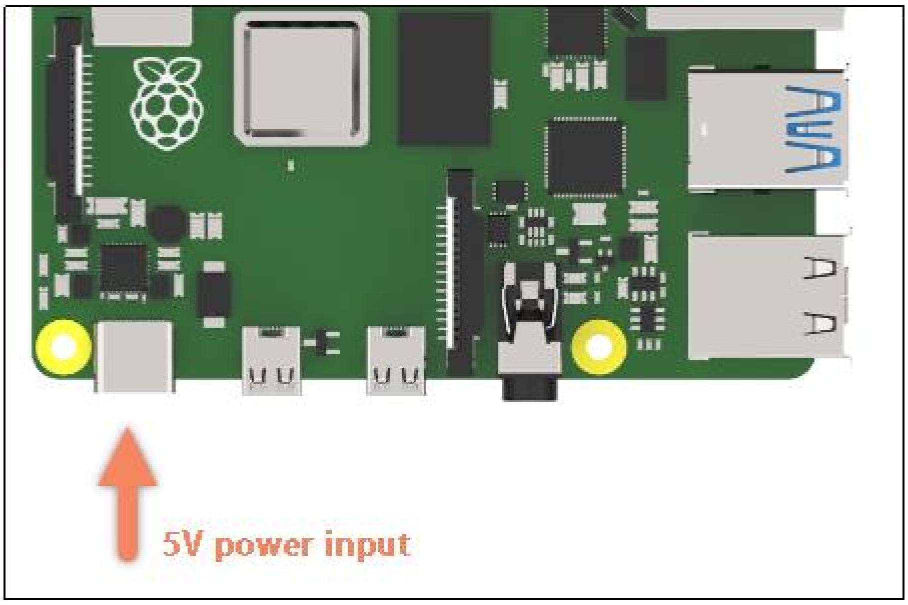
1. Power Supply for Raspberry Pi 4B:

The Raspberry Pi 4B operates with a rated voltage of 5V. It cannot directly use the I2C interface (5V, GND, SCL, SDA) of the four-channel encoder motor driver module for power supply. This is because the 5V pin on the motor driver module only supports voltage input, not output. Additionally, it is not recommended to power the Raspberry Pi GPIO pins through other interfaces of the motor driver module.

Recommended Power Solution:

A dual power supply setup is suggested. Use an external power supply to power the motor driver module (e.g., an 11.1V lithium battery, fully charged to

12V), while powering the Raspberry Pi 4B with a separate 5V, 3A power supply.



2. Why Not Power Through GPIO Pins?

It is important not to power the Raspberry Pi through its GPIO pins for the following reasons:

- No Protection Circuit: The GPIO pins do not have extra protection circuitry. Supplying power through the GPIO can cause voltage instability, which may lead to excessive current that could damage the internal CPU and components.
- Current Instability: If the current from the GPIO is insufficient or too low, the Raspberry Pi 4B may fail to operate properly. On the other hand, if the current is too high, it could damage the internal components of the Raspberry Pi.

3. Recommended Power Supply for Raspberry Pi:

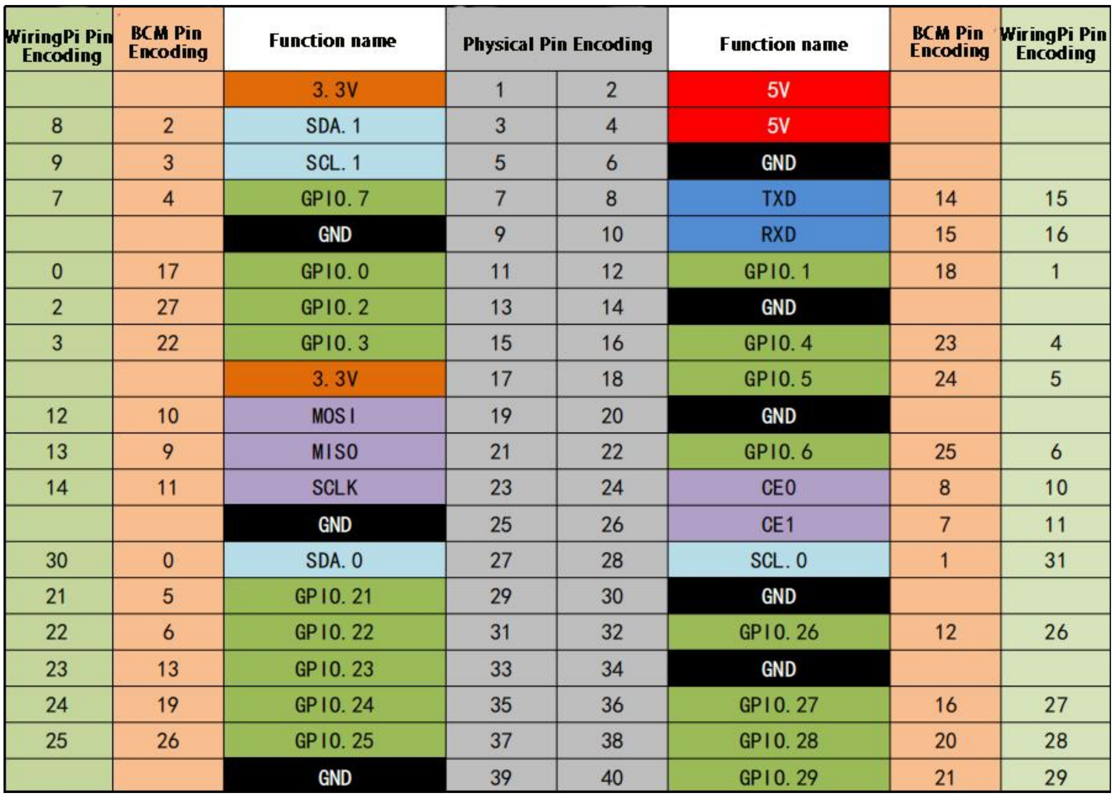
To ensure stable operation, it is recommended to use a Type-C power supply for the Raspberry Pi. A power bank with stable current and voltage (e.g., 5V, 2.5A or 5V, 3A) will suffice and guarantee the normal operation of the Raspberry Pi 4B.

Using a dedicated power supply for the Raspberry Pi not only ensures stable operation but also saves the 5V and GND pins for other uses, which is beneficial for expanding the capabilities of the Raspberry Pi.

By following these power guidelines, you can ensure the stable performance of both the Raspberry Pi and the motor driver module without risk of damage due to improper power supply configurations.

4. Raspberry Pi Encoder Pin Wiring Diagram

The following diagram illustrates the physical pinout of the Raspberry Pi. It shows two types of pin numbering systems: BCM (Broadcom) and wiringPi. When programming, you can choose either encoding method and set it accordingly in your code.



WiringPi Pin Encoding	BCM Pin Encoding	Function name	Physical Pin Encoding		Function name	BCM Pin Encoding	WiringPi Pin Encoding
		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