

Dongguan WHEELTEC INTELLIGENT

technology co., LTD

Taobao shop: [minibalance.taobao.com](http://minibalance.taobao.com)

Website: [www.wheeltec.net](http://www.wheeltec.net)

# STM32 Moving Chassis

## Development Manual

Recommend to follow our official account for updated information



Version description:

version	date	Content description
V3.5	2021/01/28	First release

# Preface

The full set of tutorials for ROS navigation robots includes three documents : " STM32 Moving Chassis Development Manual", " Ubuntu Configuration Tutorial", and " ROS Development Tutorial". For the Ubuntu configuration tutorials for virtual machines and Raspberry Pi ( Jetson Nano , Jetson TX2 , industrial computer, etc.), please see " Ubuntu Configuration Tutorial"; for ROS development tutorials, please see " ROS Development Tutorial". The content of this document tutorial is mainly used to explain the kinematic analysis, communication protocol, control mode, etc. of the ROS robot moving chassis. The moving chassis is equipped with two controllers, namely the Raspberry Pi ( Jetson Nano , Jetson TX2 , industrial computer, etc.) and the STM32 controller. Data is transmitted between the two through serial communication. For detailed instructions on the wiring, please see the chapter five. Among them, the Raspberry Pi ( Jetson Nano , Jetson TX2 , industrial computer, etc.) is installed with Ubuntu to run ROS ; the STM32 controller is used to control the moving chassis and collect odometer information, battery information,IMU information.

The task names, function names, etc. appearing in this document are the contents of the built-in program of the moving chassis. If you do not plan to read the program of the robot STM32 controller , you can ignore the program functions and program tasks appearing below (for example: APP\_Show() function, Send task data\_task , data\_transition() function ), without affecting the reading and understanding of the entire development manual.

This document applies to the six robots of the ROS educational robot series: two-wheeled differential car, Ackerman car, Mecanum wheel car, omni-wheel car, crawler car, and four-wheel drive car.

# Directory

Preface.....	2
1. Robot control mode.....	5
1.1 Robot movement speed unit.....	5
1.2 ROS ( serial port 3) control.....	6
1.3 APP control.....	15
1.4 PS2 control.....	18
1.5 Hot-RC remote control.....	19
1.6 CAN control.....	22
1.7 Serial port 1 control.....	24
2. OLED display content.....	27
2.1 OLED specific content.....	27
2.2 OLED universal display content.....	28
2.3 car self-inspection.....	29
3. Elimination of gyroscope zero drift.....	30
4. Robot kinematics analysis.....	31
4.1 Two-wheel differential ( tracked vehicle ) car.....	31
4.2 Ackerman car.....	34
4.3 Mecanum wheel car.....	36
4.4 Omni wheel car.....	38
4.5 Four-wheel drive car.....	39
4.6 PI control program source code.....	41
5. Wiring instructions.....	42
6. Control flow chart.....	45

6.1 Control flowchart of robot motor.....	45
6.2 Robot STM32 program structure diagram.....	46
6.3 Robot controller connection diagram.....	47
<b>7. Matters needing attention.....</b>	<b>48</b>
7.1 About the code.....	48
7.2 About the power interface on the adapter board.....	48
7.3 About the motor.....	48
7.4 About the battery.....	49
<b>8. How to download program to STM32 controller.....</b>	<b>50</b>
8.1 Serial download.....	50
8.2 SWD download.....	51

# 1. Robot control mode

This chapter mainly gives a detailed description of the control and use of the robot. The robot supports 6 control modes: APP remote control, PS2 wired handle control, ROS control, Hot-RC remote control, CAN control, and serial port control . The control mode is displayed in the lower left corner of the OLED display, and the ROS control mode is used by default after booting .

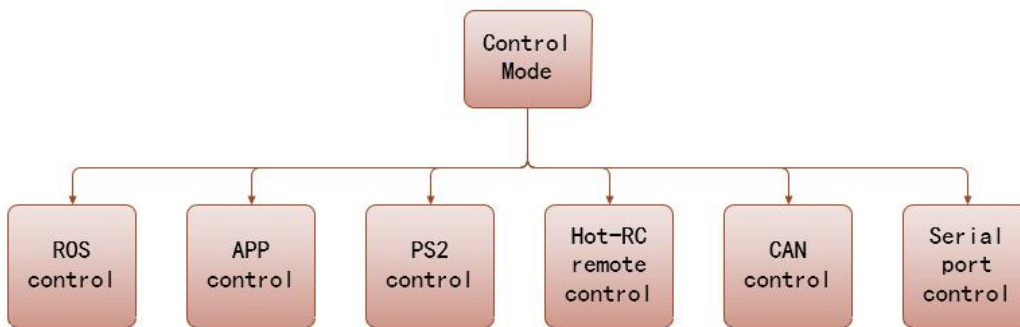


Figure 1-1 Robot control mode

## 1. 1 Robot movement speed unit

Here is an explanation of the unit of robot movement speed, which is m/s ( meters per second ) . The calculation formula of the specific "encoder raw data" converted into " m/s ( meters per second ) " is shown in Figure 1-1-1 .

Encoder\_A\_pr is the raw data of the encoder, CONTROL\_FREQUENCY is the control frequency (unit: HZ ), Encoder\_precision is the accuracy of the encoder (related to the mechanical structure of the motor and the encoder chip), wheel\_perimeter is the circumference of the wheel (unit: meters), and the final MOTOR\_A.Encoder is the actual movement speed of the robot (unit: m / s). Here only the encoder data conversion of A motor is shown , and the encoder calculation conversion of the other B , C and D motors is the same.

$$\text{MOTOR\_A.Encoder} = \text{Encoder\_A\_pr} * \text{CONTROL\_FREQUENCY} * \text{Wheel\_perimeter} / \text{Encoder\_precision};$$

Figure 1-1-1 Robot movement speed conversion formula

The positive direction calibration of XYZ three-axis speed is shown in Figure 1-1-2 below.

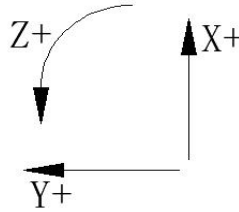


Figure 1 -1-2 The positive direction of robot XYZ three-axis speed

## 1. 2 ROS ( serial port 3) control

After the robot is powered on, the ROS control mode is used by default. This section only describes how to make the robot move through the ROS environment , in particular the principles and ROS some relevant definition of the concept see " ROS Development Manual." For the virtual machine and ros development environment setup and how to remotely log in to the ubuntu system on the robot , please see " Ubuntu Configuration Tutorial".

Here is to use the keyboard on the virtual machine to control the robot's movement.

### ① virtual machine remotely logs in to the robot ubuntu system

It should be noted that using the virtual machine to control the robot movement requires two terminals to be opened separately, and each terminal must be separately remotely logged in to the robot ubuntu system. Enter the command shown in Figure 1-2-1 in the virtual machine to remotely log in to the ubuntu system of the robot .

```
passoni@passoni:~$ ssh wheeltec@192.168.0.100
```

Figure 1-2-1 ssh wheeltec@192.168.0.100

### ② Run the function package

In the first terminal, open the " turn\_on\_wheeltec\_robot " under the " turn\_on\_wheeltec\_robot " function package of the robot to enable the robot control node.



```
wheeltec@wheeltec:~$ roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch
```

Figure 1-2-2 roslaunch turn\_on\_wheeltec\_robot turn\_on\_wheeltec\_robot.launch

In the second terminal, open the keyboard control node of the robot movement. In the second terminal, open the " keyboard\_teleop " keyboard control node under the " wheeltec\_robot\_rc " function package of the robot .

```
wheeltec@wheeltec:~$ roslaunch wheeltec_robot_rc keyboard_teleop.launch
```

Figure 1-2-3 roslaunch wheeltec\_robot\_rc keyboard\_teleop.launch

After opening the keyboard control node, you can see some control prompts. At this time, you can use the keyboard to control the robot movement. The specific keyboard control corresponding effects are shown in Table 1-1 . Press " ctrl+c " or close the terminal directly to exit the control node.

```
Control Your Turtlebot!
-----
Moving around:
  u   i   o
  j   k   l
  m   ,   .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2      turn 1
```

Figure 1-2-4 Open the keyboard control node terminal information

Table 1-1 Description of keyboard control robot motion instructions

Keyboard keys	u	i	o	j	k	l	m	,
Robot achieve effect	Left front move ment	Forw ard move ment	Right front move ment	Turn left	Emer gency stop	Turn right	Left back move ment	Back move ment
Keyboard keys	.	Space	q	z	w	x	e	c

Robot	Right	Emer	Move	Move	line	line	angle	angle
achieve	back	gency	ment	ment	speed	speed	speed	speed
effect	move	stop	speed	speed	+10%	-10%	+10	-10%
	ment		+10%	-10%			%	

### ③ STM32 sends data to ROS

The communication between ROS and STM32 controller (moving chassis) is realized through serial port. STM32 controller uses serial port 3 and the baud rate is 115200. Communication protocol includes: STM32 controller send data to ROS, ROS transmit data to STM32 controller. Open the source code of the STM32 controller. The code for serial communication is in the usartx.c file of the STM32 controller .

STM32 sends data to ROS and uses a date\_task task to execute at a frequency of 20hz . The data sent includes: frame head and end, robot enable flag, robot XYZ three-axis speed, IMU three-axis acceleration, three-axis angular velocity, battery voltage, data check bit , detailed data sent see Table 1-2 below .

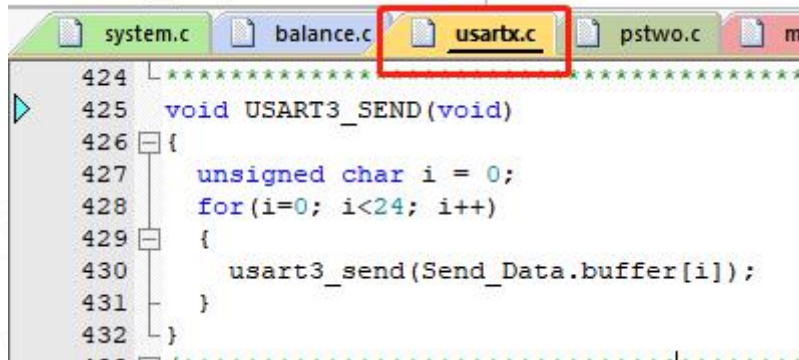
The way of sending data: Pack the data to be sent into an array, the length of the array is 24 bytes, and use the serial port to send out bit by bit. Because the serial port can only send one 8 -bit ( 1 byte) data at a time, the 2 -byte ( short type) data will be split into the upper 8 bits and the lower 8 bits for transmission.

The function assigned to the data before sending it is the "data\_transition()" function in the "usartx.c" file; the function that sends the data is the "USART3\_SEND()" function in the "usartx.c" file.If you need to change the content of the data sent, just change the data\_transition() function; if you need to change the length of the data to send, you need to change " SENT [] " array length of this at the same time also need to modify " USART3\_SEND () " function for the number of cycles and the length of the data received by the ROS terminal must be modified accordingly.

What needs to be explained in the data sent is that the frame header is a fixed value of 0X7B , the frame end is a fixed value of 0X7D , and flag\_stop is the stop flag



bit of the motor ( 0 is enabled, 1 is disabled). The calculation method of the data check bit is BCC check (All data bits (including the frame header) are XOR), and the final result is the data check bit. The calculation process of the data check bit is shown in Figure 1-2-6 .



```

424  L *****
425  void USART3_SEND(void)
426  {
427      unsigned char i = 0;
428      for(i=0; i<24; i++)
429      {
430          usart3_send(Send_Data.buffer[i]);
431      }
432  }
  
```

Figure 1-2-5 USART3\_SEND() function



```

319  L *****
320  函数功能: 计算发送的数据校验位
321  入口参数:
322  返回值: 检验位
323  L *****
324  u8 Check_Sum(unsigned char Count_Number, unsigned char Mode)
325  {
326      unsigned char check_sum=0, k;
327      //发送数据的校验
328      if(Mode==1)
329      for(k=0; k<Count_Number; k++)
330      {
331          check_sum=check_sum^Send_Data.buffer[k];
332      }
333      //接收数据的校验
334      if(Mode==0)
335      for(k=0; k<Count_Number; k++)
336      {
337          check_sum=check_sum^Receive_Data.buffer[k];
338      }
339      return check_sum;
340  }
341  }
  
```

Figure 1-2-6 Data check digit calculation function

Table 1-2 Data sent by STM32 to ROS

data content	Frame header ( Fixed value 0X7B)	flag_stop	robot X axis speed	robot Y axis speed	robot Z axis speed	Accelerometer X axis Acceleration	Accelerometer y axis Acceleration
data Types	Uint8	Uint8	short	short	short	short	short

of												
Occupy byte	1	1	2	2	2	2	2	2	2	2	2	2
Array number	1	2	3	4	5	6	7	8	9	10	11	12
data content	Accelerometer z axis Acceleration	Angular velocity meter X axis Angular velocity	Angular velocity meter Y axis Angular velocity	Angular velocity meter Z axis Angular velocity	battery Voltage	data Check Digit	End of frame ( Fixed value 0X7D)					
Types of	short	short	short	short	short	Uint8	Uint8					
Occupy byte	2	2	2	2	2	1	1					
Array number	13	14	15	16	17	18	19	20	21	22	23	24

There is another point to note here . The raw data of the robot XYZ three-axis speed, accelerometer, angular velocity meter and battery voltage are floating-point data ( float ), because the floating-point data is inconvenient to use the serial port to transmit, so these four before sending the data, amplify the floating-point number by a thousand times (reserve three decimal places), then force the amplified floating-point number into short data, and finally split the short data into two 8 -bit data before sending data. Correspondingly, after the host computer receives the data, it needs to merge the two 8 -bit data of the received data and convert it into a short type, and then convert the unit after shrinking it by a thousand times.

The following explains how to merge two 8-bit data and convert them to short type, that is to get our actual speed and other data: our control quantity unit is mm/s (0.001m/s), and the control quantity direction is from the high 8-bit data The highest bit of the decision.

Example 1 : 21 B6=0010 0001 1011 011 , the highest bit is 0 , positive number, the speed is  $21B6=(2*16+1)*256+(B*16+6)=(2*16+1)*256+(11*16+6)=8630(\text{mm/s})$ .

Example 2 : A1 2F=1010 0001 0010 1111 , the highest bit is 1 , negative number, the speed is  $2^{16}(\text{FF FF}+1)-A1\ 2F=5E\ \text{D0}+1=(5*16+E)*256+(D*16+0)+1=24272(\text{mm/s})$ .

The following picture shows the actual data received through the serial port assistant after we connect to the serial port 3 . ( Note that our serial port 3 is not integrated with CH340 , and we need to use the data cable for communication between ROS and STM32 , so the serial port assistant can communicate with STM32 serial port 3 ).



Figure 1 -2-7 Data sent by car serial port 3

Let's convert the received 24 bytes of data:

The first byte: 0x7B , header;

The second byte: 0X00 , the motor is in a non-stop state;

The 3rd and 4th bytes: X axis speed, high 8 bits 0X01 ( hexadecimal ) = 0000 0001 ( binary ) , low 8 bits 0X01 ( hexadecimal ) = 0000 0001 ( binary ) , the highest bit It is 0 , a positive number ( forward ) , and the speed is  $1*256+1=257(\text{mm/s})$  . This speed is the actual speed calculated by the car on the encoder data.

The 5th and 6th bytes: Y- axis speed, high 8 bits 0X00 ( hexadecimal ) = 0000 0000 ( binary ) , low 8 bits 0X01 ( hexadecimal ) = 0000 0001 ( binary ) , the highest bit It is 0 , a positive number ( left shift ) , and the speed is  $0*256+1=1(\text{mm/s})$  . This speed is the actual speed calculated by the car on the encoder data.

The 7th and 8th bytes: Z axis speed, high 8 bits 0X00 ( hexadecimal ) = 0000 0000 ( binary ) , low 8 bits 0X00 ( hexadecimal ) = 0000 0000 ( binary ) , the highest bit It is 0 , a positive number ( counterclockwise rotation ) , and the speed is  $0*256+0=0(0.001\text{rad/s})$  . This speed is the actual speed calculated by the car on the encoder data.

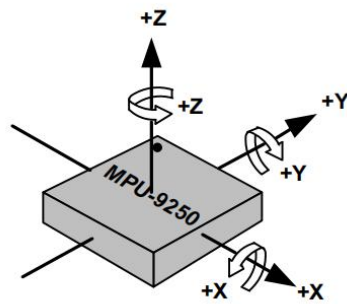


Figure 1 -2-8 Three-axis schematic diagram of MPU9250 accelerometer and angular velocity meter

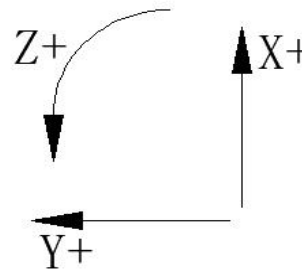


Figure 1 -2-9 Robot XYZ three-axis calibration

The next 12 bytes are the data of the three-axis accelerometer and angular velocity meter. Note that the three-axis direction of XYZ has changed. The positive direction of X- axis is rightward, the positive direction of Y- axis is forward, and the positive direction of Z- axis is ascending. The speed of the accelerometer and angular velocity meter is the speed of rotation around the XYZ three axes, as shown in Figure 1-2-8 above. Because gyro XY robot axis direction with our calibration XY -axis



direction has a difference, so the XY angular velocity and the angular velocity data in the transmission shaft made a correct conversion.

The 9th and 10th bytes: X- axis acceleration, high 8 bits 0XFE ( hexadecimal )=1111 1110 ( binary ) , low 8 bits 0X96 ( hexadecimal )=1001 0110 ( binary ) , the highest bit It is 1 , a negative number, and the size is  $2^{16}(\text{FF} + 1) - \text{FE} \ 96 = 01 \ 69 = 1 * 256 + 105 = 361$  , which is converted into acceleration  $361 / 1672 = 0.2159(\text{m/s}^2)$ .

The 11th and 12th bytes: Y- axis acceleration, high 8 bits 0XFD ( hexadecimal )=1111 1101 ( binary ) , low 8 bits 0XCE ( hexadecimal )=1100 1110 ( binary ) , the highest bit It is 1 , a negative number, and the size is  $2^{16}(\text{FF} + 1) - \text{FD} \ \text{CE} = 02 \ 31 = 2 * 256 + 49 = 561$  , which is converted to acceleration  $561 / 1672 = 0.3355(\text{m/s}^2)$ .

The 13th and 14th bytes: Z- axis acceleration, high 8 bits 0X40 ( hexadecimal ) = 0100 0000 ( binary ) , low 8 bits 0X80 ( hexadecimal ) = 1000 0000 ( binary ) , the highest bit It is 0 , a positive number, and the size is  $64 * 256 + 128 = 16512$  , which is converted to acceleration  $16512 / 1672 = 9.8756(\text{m/s}^2)$ .

The 15th and 16th bytes: X axis angular velocity, high 8 bits 0XFF ( hexadecimal )=1111 1111 ( binary ) , low 8 bits 0XFB ( hexadecimal )=1111 1011 ( binary ) , the highest bit It is 1 , a negative number, and the size is  $2^{16}(\text{FF} + 1) - \text{FF} \ \text{FB} = 00 \ 04 = 0 * 256 + 4 = 4$  , converted to angular velocity  $4 / 3753 = 0.0011(\text{rad/s})$  .

The 17th and 18th bytes: Y- axis angular velocity, high 8 bits 0X00 ( hexadecimal ) = 0000 0000 ( binary ) , low 8 bits 0X07 ( hexadecimal ) = 0000 0111 ( binary ) , the highest bit It is 0 , a positive number, and the size is  $0 * 256 + 7 = 7$  , which is converted to angular velocity  $7 / 3753 = 0.0019(\text{rad/s})$  .

The 19th and 20th bytes: Z axis angular velocity, high 8 bits 0X00 ( hexadecimal ) = 0000 0000 ( binary ) , low 8 bits 0X01 ( hexadecimal ) = 0000 0001 ( binary ) , the highest bit It is 0 , a positive number, and the size is  $0 * 256 + 1 = 1$  , which is converted to angular velocity  $1 / 3753 = 0.0003(\text{rad/s})$  .

The 21st and 22nd bytes: battery voltage, high 8 bits 0X58 ( hexadecimal )=0101 1000 ( binary ) , low 8 bits 0X38 ( hexadecimal )=0011 1000 ( binary ) , the highest bit is 0 , a positive number, the size is  $88*256+56=22584$  , and the voltage size is 22584mv ( millivolt ) .

The 23rd byte: BCC check digit ( exclusive OR of the first 22 bytes ) ,  
 $0X83=0X7B\oplus 0X00\oplus 0X01\oplus 0X01\oplus 0X00\oplus 0X01\oplus 0X00\oplus 0X00\oplus 0XFE\oplus 0X96\oplus 0XFD\oplus 0XCE\oplus 0X40\oplus 0X80\oplus 0XFF\oplus 0XFB\oplus 0X00\oplus 0X07\oplus 0X00\oplus 0X01\oplus 0X58\oplus 0X38$  .

(If you want to verify the result of this check digit, you can use some web and online platforms to calculate the check digit)

The 24th byte: 0X7D , end of frame;

#### ④ STM32 receives the data sent by ROS

The STM32 controller board is equipped with CH340 ( serial port 1) and CP210 ( serial port 3) two serial communication interfaces. The two serial ports receive data processing procedures are exactly the same. By default, CP2102 ( serial port 3) is used for serial communication with ROS. Take the serial port 3 to receive data as an example.

The received data adopts the interrupt receiving method. The received data includes the robot product signal, the enable control flag bit, the robot three-axis target speed, and the data check bit.

The frame head and frame end are fixed values by default; flag\_stop is the enable control bit of the robot, which enable is sent by default; the robot's three-axis target speed is used to control the robot's movement. The specific receiving content is shown in Table 1-3 . It should be noted that the array number in the table is the array number of the data sent by the host computer.



Table 1-3 STM32 receives the data sent by ROS

data content	Frame header (Fixed value 0X7D)	Reserved	Reserved	robot X axis Target speed	robot Y axis Target speed	robot Z axis Target speed	data Check Digit	End of frame (Fixed value 0X7B)			
data Types of	Uin8	Uin8	Uin8	short	short	short	Uin8	Uin8			
Occupy byte	1	1	1	2	2	1	1	1			
Array number	1	2	3	4	5	6	7	8	9	10	11

Note: Differential cars and Ackerman cars do not support Y- axis movement control. Only mecanum-wheel cars and omni-wheel cars support Y- axis movement control.

Among the 7 ways to control the robot, the control priority of ROS is the highest. Whenever the serial port 3 of the STM32 controller receives data, it is forced to enter the ROS mode. The reason for not receiving data in the first 10 seconds is to eliminate the useless data sent during the robot power-on process. To start receiving data after a waiting period of 10 seconds, first detect the data frame header, and start receiving data after detecting the data frame header; after the data is received, verify whether the data check bit at the end of the frame is wrong, and use data only if the data check bit is correct. Please refer to the USART3\_IRQHandler() serial port interrupt function in the usartx.c file for details of the serial port interrupt receiving data .

### 1.3 APP control

The robot supports APP Bluetooth control and online parameter adjustment. In

APP mode, directly use the joystick to control the movement of the robot in space. The speed unit of APP controlling the robot movement is mm/s ( millimeters / second ). The acceleration and deceleration buttons on the diagonally above the joystick will increase / decrease the speed of the robot by 100 ( mm/s ) every time you press it .

While the APP is controlling the robot , the robot will send data to the mobile phone via Bluetooth (the APP supports both WIFI and Bluetooth communication ) . You can see the sent data in the Debug column of the APP. See the APP\_Show() function in the code's show.c file for details on what to send. APP control mode the principle is the Bluetooth ( or WIFI) serial communications control, the robot 's APP control use the serial port 2 , the baud rate is set to 9600 , which the control command is in the serial port 2 receive interrupt service function.

### ① Online tuning

Install the latest version of MiniBalance APP on your Android phone, and then follow the corresponding video tutorials to remotely control the robot or perform online parameter adjustments. In the "Debug" interface, you can click the "parameter x " to customize the name of each channel . The specific effect is shown in Figure 1-3-1 and Figure 1-3-2 . In addition, before adjusting the PID parameters, we need to click [ Get Device Parameters ] ( call up by the menu button in the upper right corner ) , update the robot's PID parameters to the APP , and then drag the slider. When we let go, the APP can send the parameters to the robot.

Parameter 0 is the speed parameter of the car, and adjusting parameter 0 can adjust the speed of the car.

Parameters 1 and 2 are PI parameters for the motion control of the car .

It should be noted that, assuming that the current robot control speed is 40 , when we want to adjust the speed parameter to 150 (out of range), we need to adjust it to 80 for the first time , and then click [ Get device parameters ] again , then the adjust range is changed to 0-160 . Speed range set in the robot is 20- 200. For a detailed explanation of the robot control speed unit, please refer to "Chapter 3 Robot Movement Speed Unit" below.



Figure 1-3-1 Default tuning interface



Figure 1-3-2 Interface after obtaining device parameters

(Need to modify the parameter name by yourself)

## ② APP control robot



Figure 1-3-3 APP control interface

Each operation corresponding to the APP operation interface actually sends different instruction information to the robot (switching control methods and interfaces also send instructions), the robot responds after receiving the instruction information, the detailed information sent by each operation of the APP operation interface is shown in the table 1-4 :

Table 1-4 Description of APP interface operation instructions

APP joystick	↑	↗	→	↘	↓	↙	←	↖
Data received by the robot	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48
Robot achieve effect	Forward move ment	Right front move ment	Turn right in place	Right back move ment	Back move ment	Left back move ment	Turn left in place	Left front move ment
button	gravity		Joystick		button	slow down	accelerate	
Data received by the robot	0x49		0X4a		0x4b	0x59	0x58	

Note: After the mobile phone is successfully connected to the Bluetooth of the car, you need to push the joystick forward for 0.5 seconds to formally control the car.

## 1. 4 PS2 control

PS2 mode uses PS2 wireless controller. The sequence of PS2 mode control is: first plug in the PS2 controller before powering on, and then turn on the power. At this time, the red light on the controller is always on to indicate normal operation. If the indicator is off, press the button above the indicator and press the START button to enter the handle mode. At this time, you can see " PS2 " displayed on the lower left corner of the display .

In PS2 mode, use the left joystick to control the robot to move forward and backward, the right joystick to control the left and right steering of the robot (the omnidirectional motion car is the left joystick to control the movement of the car in 8 directions in the space, and the right joystick to control the car in place self-rotating motion ). The reason for this design is that if the same joystick is used for the front and rear and left rear controls, it is easy to accidentally touch the steering. The two buttons in the upper left corner are acceleration and deceleration buttons.

Note: In PS2 mode, the PS2 handle must be plugged in before powering on, otherwise it is easy to burn the handle and cause the motor to rotate randomly. You need to press the Start button on the PS2 handle to enter PS2 handle mode after starting up.



Figure 1 -4-1 PS2 handle physical picture

## 1.5 Hot-RC remote control

It should be noted that the Ackerman car does not support remote control of Hot-RC.

The correct use steps of the Hot-RC remote control are: first connect the Hot-RC remote control receiver to the car, turn on the power of the Hot-RC remote control, power on the car, and the indicator light of the Hot-RC remote control receiver is



always on, indicating that it is connected, and now you can see that the bottom left corner of the screen says "R-C". The control method of the Hot-RC remote control is: the left joystick of the Hot-RC remote control controls the car to move forward and back, the right joystick controls the car to turn left and right (the omnidirectional movement car is the left joystick to control the movement of the car in 8 directions in the space, and the right joystick controls the car rotates in place ), and at the same time, turning the right joystick back and front can control the speed of the car, which is equivalent to the throttle (the throttle function does not support the Ackerman car ). The SWC switch on the upper right controls the car to select normal mode and low speed mode. Before turning off the Hot-RC, you need to turn off the car and then turn off the Hot-RC remote control.



Figure 1-5-1 The physical picture of the remote control aircraft model

It should be noted that the remote control channel of the Hot-RC needs to be configured as shown in Figure 1-5-2 , where the leftmost button is in the middle, and the remaining four buttons are in the bottom. These channels are for adjusting the direction of control.





Figure 1-5-2 The physical image of the remote control aircraft

Next, I will explain how to connect the RC receiver to the adapter board.

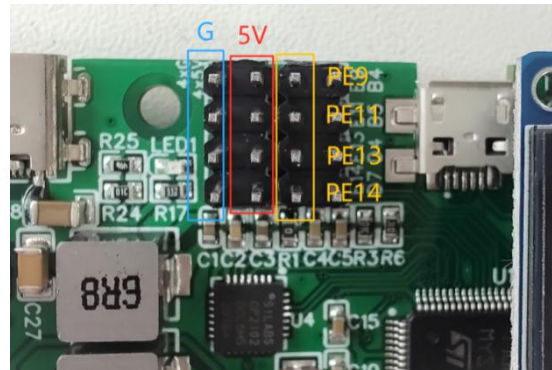


Figure 1-5-3 Physical image of the Hot-RC receiver

Figure 1-5-4 Model aircraft interface on the adapter board

Table 1-5 Corresponding channels of model airplane remote control pins

Hot-RC receiver	GND	5V	CH1	CH2	CH3	CH4
Adapter board	G	5V	PE9	PE11	PE13	PE14

As shown in Figure 1-5-3 , Figure 1-5-4 and Table 1-5 , the Hot-RC receiver has three columns, which are GND , 5V and signal line. When we use it, connect GND and 5V. Then the signal line CH1 , the CH2 , a CH3 , of CH4 are respectively connected to the adapter plate pin PE9 , PE11 , PE13 , PE14 . The PE14 pin is only needed for the omni-directional movement of the car, and is used for left and right traverse.

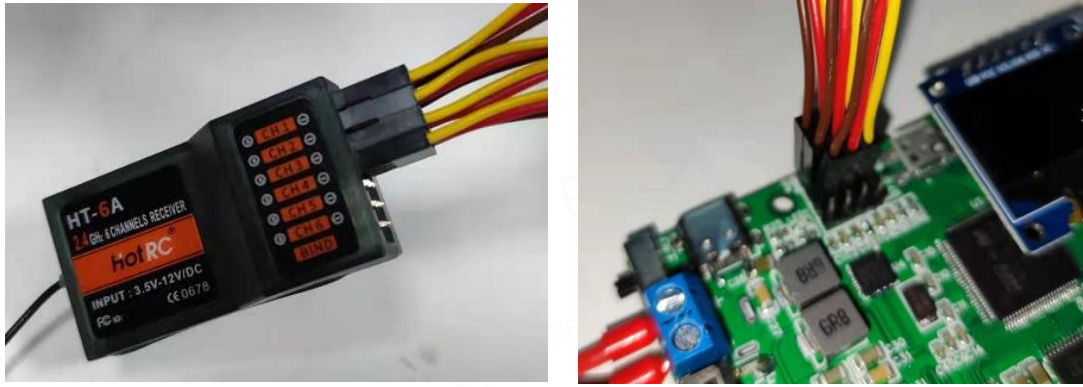


Figure 1 -5-5 Wiring example of airplane model

## 1.6 CAN control

The car supports CAN communication, and the connection mode of CAN communication is CANL to CANL and CANH to CANH . If you want the car to receive CAN control commands, you need to send an enable command to the CAN interface first , and the baud rate is 1M .

The enable command format is as follows:

Identifier ID: 0X121

Frame type: standard frame

Frame format: DATA

DLC: 8 bytes

Table 1-6 CAN enable command

Data field	tx[0]	tx[1]	tx[2]	tx[3]	tx[4]	tx[5]	tx[6]	tx[7]
content	10	12	15	19	24	30	37	Flag

**Note: The data in this table is 10 hexadecimal form**

When Flag=1 , CAN control is enabled, and the controller will no longer receive commands from other control modes. After CAN control is enabled, you can see " CAN " in the lower corner of the display , and then we can send CAN commands to control the car. The description of the control instruction is as follows:

Identifier ID: 0X121

Frame type: standard frame

Frame format: DATA

DLC: 8 bytes

Table 1-7 The car receives CAN control commands

Data field	rx[0]	rx[1]	rx[2]	rx[3]	rx[4]	rx[5]	rx[6]	rx[7]
content	car X direction control amount 8 bits higher	car X direction control amount 8 bits lower	car Y direction control amount 8 bits higher	car Y direction control amount 8 bits lower	car Z direction control amount 8 bits higher	car Z direction control amount 8 bits lower	Reserved	Reserved

rx[6] and rx[7] are reserved data bits for us to add the data we need to transmit.

CAN communication comes with BCC check, so no data check bit is needed here.

Note: Differential cars and Ackerman cars do not support Y- axis movement control, so the control amount in the Y- axis direction is 0 by default .

The car can also send its own data while receiving CAN command control. The function that executes CAN sending data is CAN\_SEND() located in the file [ usartx.c ] . By default, the task of sending data has been opened and the data to be sent has been set. If you need to customize sending other content, just replace CAN\_SEND the content of Send\_Data.buffer[i] in the function CAN\_SEND is sufficient .

Due to the large number of data to send a total of 24 bytes, the data is sent 8 bytes at a time, and sent in 3 times, the SCM receiving the data sent by the car can confirm the group of data currently received through the identifier ID, the identifier for sending the first group of data is 0X101 , the identifier for sending the second group of data is 0X102 , and the identifier for sending the third group of data is 0X103. The detailed content of CAN transmission of the car is shown in the following table. The content of CAN transmission data is the same as that of serial port 1 and serial port 3 ( the interface for communicating with ROS ) . For this part, please refer to section 1.2 ROS ( serial port 3) control.

Our supporting CAN sending routine is equipped with the interrupt service function of CAN receiving to receive the data sent by the car. Single-chip computer models of routine code adaptation is STM32F103RC or with series single chip microcomputer, at the same time, VP230 chip is needed to convert the data of the SCM before formal CAN communication with the car, the wiring as shown below, MCU PD1, PD0 pins are connected to VP230 D, R pins, VP230 CANH, CANL pins are connected to the car CANH, CANL pins ( i.e. H to H , L to L ) .

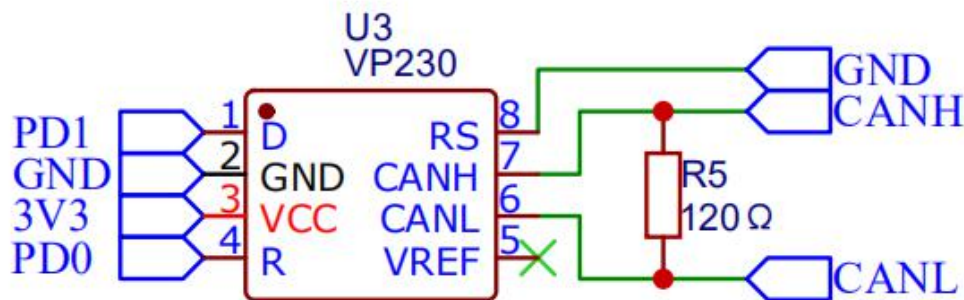


Figure . 1 -6-1 by the microcontroller and the car VP230 Conversion

If you want to check whether the data received by CAN communication is correct, you can use the host computer to connect the serial port 1 of the microcontroller . After the single-chip microcomputer receives the information sent by the CAN communication of the car , it will send it out through the serial port 1 of the single-chip microcomputer , and the baud rate is 9600 .

Note: The CAN and serial port routines provided in the information are not the codes running on the car. It does not mean that you need to download the serial port routines if you need serial communication. The car already has a program (firmware) by default, and you can operate it without downloading any program. The CAN and serial port routines provided by the data are programs that can be run on other microcontrollers to communicate with the car.

## 1. 7 Serial port 1 control

Two serial communication interfaces of serial port 1 (CP210) and serial port 3 (CP210) are led out on the STM32 controller board. The two serial ports send (the content sent is also the same) and receive data processing procedures are exactly the



same. Serial port 3 is used by default for serial port communication with ROS terminal. For the use of serial port 1 , please refer to section 1.2 ROS ( Serial Port 3) control in the previous section of this article .

When serial port transmission, it is necessary to pay attention to the same baud rate settings of both communication parties. The baud rate of serial port 1 in the program is 115200 . After receiving the data from the serial port 1 , the car enters the serial port control mode, and the lower left corner of the display shows " USART " .

Table 1-8 The car receives serial control commands

Data field	Frame header	tx[0]	tx[1]	tx[2]	tx[3]	tx[4]	tx[5]	tx[6]	tx[7]	tx[8]	End of frame
content	0X7B	Reserved	Reserved	The higher 8 bits of the control amount of the car in X direction	The lower 8 bits of the control amount of the car in X direction	The higher 8 bits of the control amount of the car in Y direction	The lower 8 bits of the control amount of the car in Y direction	The higher 8 bits of the control amount of the car in Z direction	The lower 8 bits of the control amount of the car in Z direction	BCC Check Digit	0X7D

Note: Differential cars, Ackerman car, tracked vehicles, and four-wheel drive vehicles do not support Y- axis movement control. Only mecanum-wheel cars and omni-wheel cars support Y- axis movement control. Download the serial port sending instruction routine provided by us to another single-chip microcomputer, connect the serial communication cable between the single-chip microcomputer and the car controller, and the car enters the serial port control mode.

```

usart2_send(0X7B); //帧头
usart2_send(0X00); //预留位
usart2_send(0X00); //预留位
usart2_send(0X01); //X轴速度高8位
usart2_send(0X10); //X轴速度低8位
usart2_send(0X00); //Y轴速度高8位
usart2_send(0X00); //Y轴速度低8位
usart2_send(0X00); //Z轴速度高8位
usart2_send(0X00); //Z轴速度低8位
usart2_send(0X6A); //BBC校验位
usart2_send(0X7D); //帧尾

```

Figure 1-7-1 The control command sent by the serial port command routine to the car

Similarly, you can directly send data to the serial port 1 of the car through the host computer serial port assistant , as shown in Figure 1-7-2 .

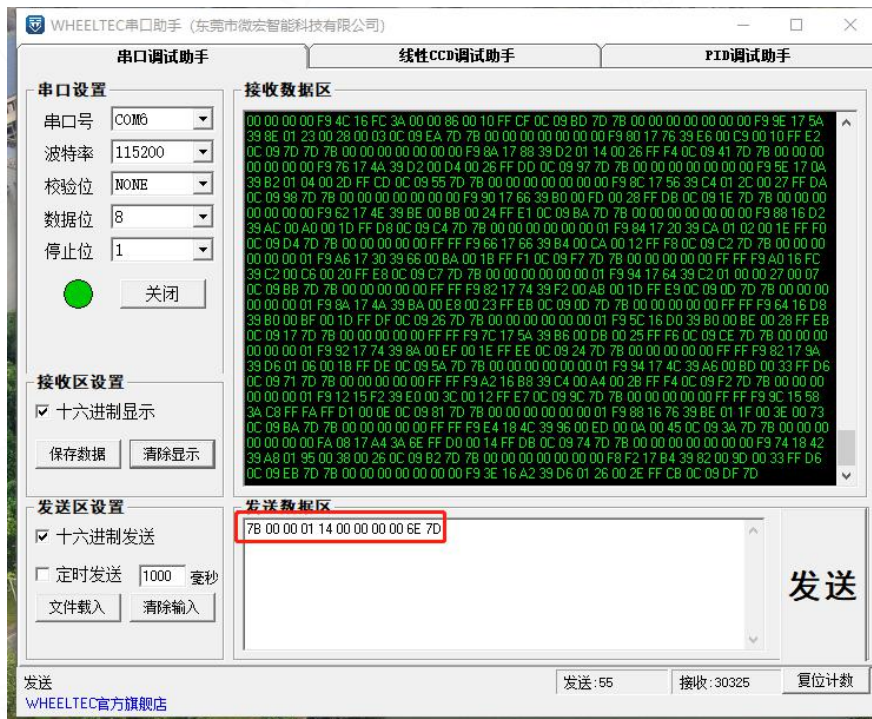


Figure 1-7-2 The control command sent by the serial port assistant to the car

Serial port 1 and serial port 3 ( ROS ) control the same commands for receiving and controlling. For the calculation and conversion of the speed control amount of the car and the content of the data sent by the car, please refer to our section 1.2 ROS ( serial port 3) control.



## 2. OLED display content

### 2.1 OLED specific content

The robot is equipped with an OLED display, and the contents displayed on the display of different types of robots are similar, as follows:

#### ① Tank vehicles

Tank BIAS Z +5 Line 1:Car model and the Z axis angular velocity zero drift value  
 GYRO Z + 2 Line 2:Z-axis angular velocity with zero drift value removed  
 L: + 0 + 0 Line 3:Target and measured values of left motor  
 R: + 0 + 0 Line 4:Target and measured values of right motor  
 MA + 0 MB + 0 Line 5:PWM value of motor  
 ROS ON 12.03V Line 6:control mode enable switch battery voltage

Figure 2-1-1 tracked vehicle OLED display content

#### ② Ackerman car

Akm BIAS Z +5 Line 1:Car model and the Z axis angular velocity zero drift value  
 GYRO Z + 2 Line 2:Z-axis angular velocity with zero drift value removed  
 L: + 0 + 0 Line 3:Target and measured values of left motor  
 R: + 0 + 0 Line 4:Target and measured values of right motor  
 Servo : + 1500 Line 5:Steering gear value  
 ROS ON 12.03V Line 6:control mode enable switch battery voltage

Figure 2 - 1 -2 Ackerman car OLED display content

#### ③ Two-wheel differential car

Diff BIAS Z +5 Line 1:Car model and the Z axis angular velocity zero drift value  
 GYRO Z + 2 Line 2:Z-axis angular velocity with zero drift value removed  
 L: + 0 + 0 Line 3:Target and measured values of left motor  
 R: + 0 + 0 Line 4:Target and measured values of right motor  
 MA + 0 MB + 0 Line 5:PWM value of motor  
 ROS ON 12.03V Line 6:control mode enable switch battery voltage

Figure 2-1-3 two-wheel differential car OLED display content

#### ④ Omni-wheel car

```

Omni  GZ + 5 Line 1:Car model and Z-axis angular velocity with zero drift value removed
A: +  0  + 0 Line 2:Target and measured values of A motor
B: +  0  + 0 Line 3:Target and measured values of B motor
C: +  0  + 0 Line 4:Target and measured values of C motor
MOVE  Z  + 0 Line 5:Real-time angular velocity calculated by the encoder
ROS ON 12.03V Line 6:control mode  enable switch  battery voltage
    
```

Figure 2-1-4 omni wheel car OLED display content

⑤ Mecanum-wheel car

```

Mec   GZ  + 9 Line 1:Car model and Z-axis angular velocity with zero drift value removed
A: +  0  + 0 Line 2:Target and measured values of A motor
B: +  0  + 0 Line 3:Target and measured values of B motor
C: +  0  + 0 Line 4:Target and measured values of C motor
D: +  0  + 0 Line 5:Target and measured values of D motor
ROS ON 12.03V Line 6:control mode  enable switch  battery voltage
    
```

Figure 2-1-5 mecanum wheel car OLED display content

⑥ Four-wheel drive car

```

4WD   GZ  + 9 Line 1:Car model and Z-axis angular velocity with zero drift value removed
A: +  0  + 0 Line 2:Target and measured values of A motor
B: +  0  + 0 Line 3:Target and measured values of B motor
C: +  0  + 0 Line 4:Target and measured values of C motor
D: +  0  + 0 Line 5:Target and measured values of D motor
ROS ON 12.03V Line 6:control mode  enable switch  battery voltage
    
```

Figure 2-1-6 four-wheel drive OLED display content

## 2.2 OLED universal display content

① Control mode

Different control modes correspond to different display contents in the lower left corner, see Table 2-1 for details .

Table 2-1 Robot OLED display control mode

mode	CAN	APP	PS2 handle	Hot-RC remote control	Serial port 1	Serial port 3
Display content	CAN	APP	PS2	RC	UART1	UART3

② Enable switch

The enable switch is at the upper left corner of the robot STM32 controller. The robot motor can only be controlled when the enable switch is in the " ON " state.

Taking into account the stability of the robot in the program , the motor is in a forced disable state within 10s after the STM32 controller is powered on . At this time, even if your enable switch is in the " ON " state, it will display " OFF ", and it will be updated to " ON " after 10s . It should be noted that the serial port 1 and serial port 3 will not receive data within 10s .

## **2. 3 car self-inspection**

The car is equipped with a self-test codes, see the detailed contents of the document "model, wiring, check encoder tutorial .doc ."



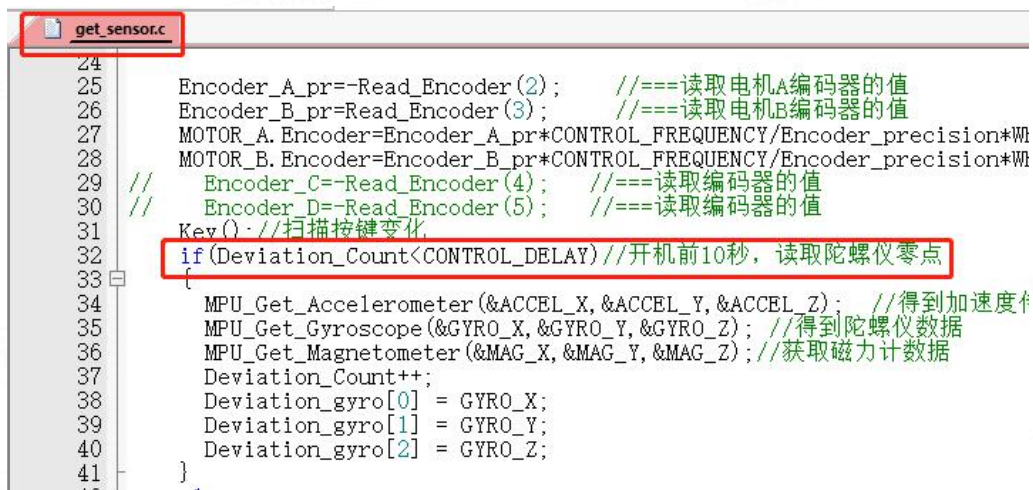
### 3. Elimination of gyroscope zero drift

The IMU sensor is needed in the ROS navigation system . In our ROS robot system, the IMU sensor is integrated into the STM32 controller. The STM32 controller collects the IMU data and sends it to ROS . The IMU used on the STM32 sports chassis is MPU9250 . The IMU integrates a three-axis angular velocity meter, a three-axis accelerometer, and a three-axis magnetometer. Here we only need the angular velocity meter and accelerometer. Gyroscopes cannot avoid the problem of zero drift, so a zero drift elimination mechanism is set in the program.

In the first 10 seconds of power-on, the gyroscope reads the angular velocity value without removing the zero drift value. At the second 10, the program reads the current angular velocity value as the drift value. The gyroscope data read after 10 seconds will subtract the zero point offset. At this time, the LED light changes from constant light to flashing. The angular velocity value read in the subsequent reading process will subtract the zero point drift value, and the result is angular velocity that eliminates the zero drift.

If you feel that the gyroscope drift value obtained in 10 seconds is not accurate enough, you can double-click the user button (the lower left corner of the STM32 controller) at any time to retrieve the gyroscope drift value.

The process of collecting gyroscope data described above is in the `get_sensor.c` file of the STM32 controller code, as shown in Figure 3-1 .



```

24
25 Encoder_A_pr=Read_Encoder(2); //===读取电机A编码器的值
26 Encoder_B_pr=Read_Encoder(3); //===读取电机B编码器的值
27 MOTOR_A.Encoder=Encoder_A_pr*CONTROL_FREQUENCY/Encoder_precision*W
28 MOTOR_B.Encoder=Encoder_B_pr*CONTROL_FREQUENCY/Encoder_precision*W
29 // Encoder_C=Read_Encoder(4); //===读取编码器的值
30 // Encoder_D=Read_Encoder(5); //===读取编码器的值
31 Key(); //扫描按键变化
32 if (Deviation_Count<CONTROL_DELAY) //开机前10秒, 读取陀螺仪零点
33 {
34     MPU_Get_Accelerometer(&ACCEL_X,&ACCEL_Y,&ACCEL_Z); //得到加速度作
35     MPU_Get_Gyroscope(&GYRO_X,&GYRO_Y,&GYRO_Z); //得到陀螺仪数据
36     MPU_Get_Magnetometer(&MAG_X,&MAG_Y,&MAG_Z); //获取磁力计数据
37     Deviation_Count++;
38     Deviation_gyro[0] = GYRO_X;
39     Deviation_gyro[1] = GYRO_Y;
40     Deviation_gyro[2] = GYRO_Z;
41 }

```

Figure 3-1 Collect gyroscope data

## 4. Robot kinematics analysis

In order to make the robot move, it is not enough to provide the target speed. The target speed of the robot needs to be converted to the actual target speed of each motor. Finally, the control of the motor is realized according to the target speed of the motor. The process of converting the target speed of the robot into the target speed of the motor is called “kinematic analysis”. Kinematics analysis is divided into forward and inverse solutions. Before kinematics analysis, let’s explain separately what are forward kinematics and inverse kinematics. :

①Forward kinematics solution: Calculate the speed of the robot in the X , Y and Z directions through the speed of each wheel of the robot .

②Inverse kinematics solution: Calculate the speed of each wheel of the robot by the speed of the robot's X- axis, Y- axis and Z- axis.

### 4. 1 Two-wheel differential ( tracked vehicle ) car

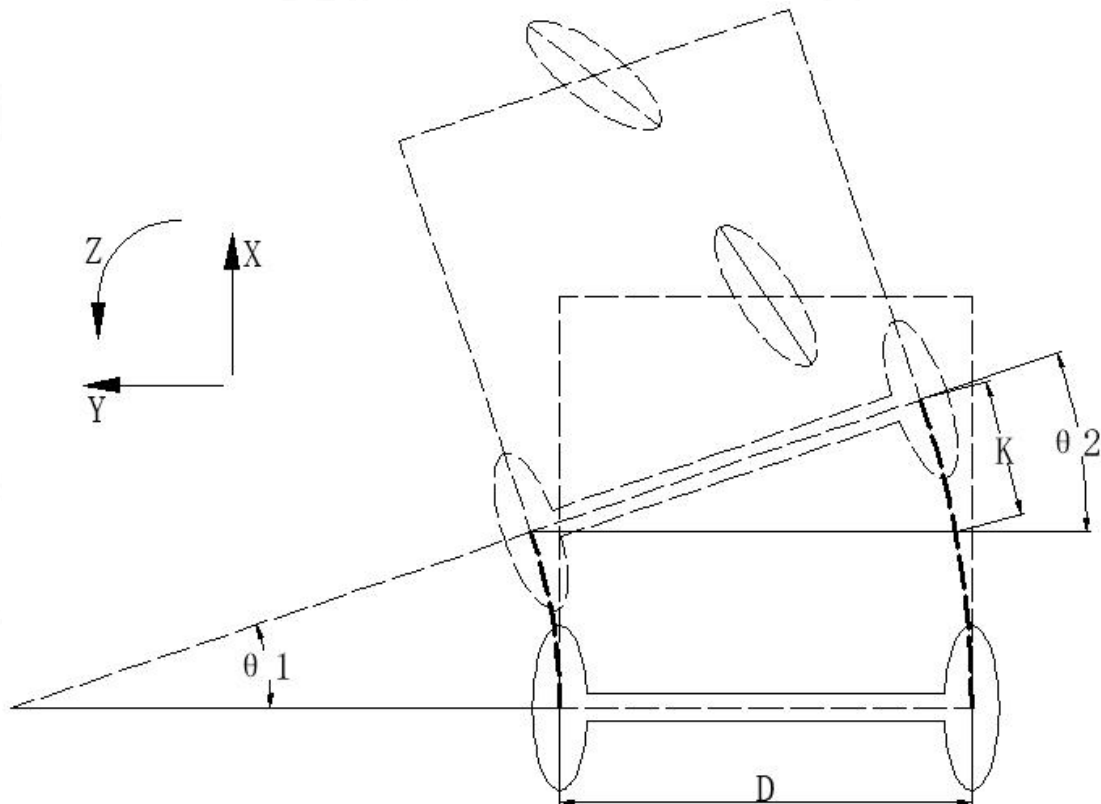


Figure 4-1 Kinematics model of two-wheel differential car

① kinematic analysis

The simplified motion model of the robot is shown in Figure 4-1, X-axis positive direction is forward, the Y-axis positive direction is the left translation, the Z-axis positive direction is counterclockwise ( the product below are the same, will be omitted ) . The distance between the two wheels of the robot is  $D$  , the speeds of the X- axis and Z- axis of the robot are respectively:  $V_x$  and  $V_z$  , and the speeds of the left and right wheels of the robot are respectively:  $V_L$  and  $V_R$  .

Assuming that the robot travels in a left-front direction for a certain distance, the distance that the right wheel of the robot travels more than the left wheel is approximately  $K$ , and the point on the wheel of the robot is used as the reference point to extend the reference line, and then:  $\theta_1 = \theta_2$  .

Since this  $\Delta t$  is very small ( 10ms ), the amount of angle change  $\theta_1$  is also very small, so there is an approximate formula:

$$\theta_2 \approx \sin (\theta_2) = \frac{K}{D}$$

From mathematical analysis, the following formula can be obtained:

$$K = (V_R - V_L) * \Delta t, \quad \omega = \frac{\theta_1}{\Delta t}$$

The result of the positive kinematics solution can be solved by the above formula:

The speed in the X- axis direction of the robot  $V_x = \frac{V_L + V_R}{2}$  , and the speed in

the Z- axis direction of the robot  $V_z = \frac{V_R - V_L}{D}$  .

The result of the inverse kinematics solution is obtained directly from the positive solution:



The speed of the left wheel of the robot  $V_L = V_x - \frac{V_z * D}{2}$  , the speed of the right

wheel of the robot  $V_R = V_x + \frac{V_z * D}{2}$  .

## ② C language implementation

There are two motors with encoders on the robot. We need to write the above motion relationships in C language, and then control the motors. Code show as below:

```
void Drive_Motor(float vx, float vz)
{
    Target_Left = vx - vz * WIDTH_OF_ROBOT / 2.0f; // Calculate the target speed of
the left wheel
    Target_Right = vx + vz * WIDTH_OF_ROBOT / 2.0f; // Calculate the target speed of
the right wheel
}
```

The above statement is to find the target speed of the two motors (inverse kinematics solution) through the speed of the X and Z axis of the robot , where WIDTH\_OF\_ROBOT is the macro definition of the linear distance between the two wheels.

## 4.2 Ackerman car

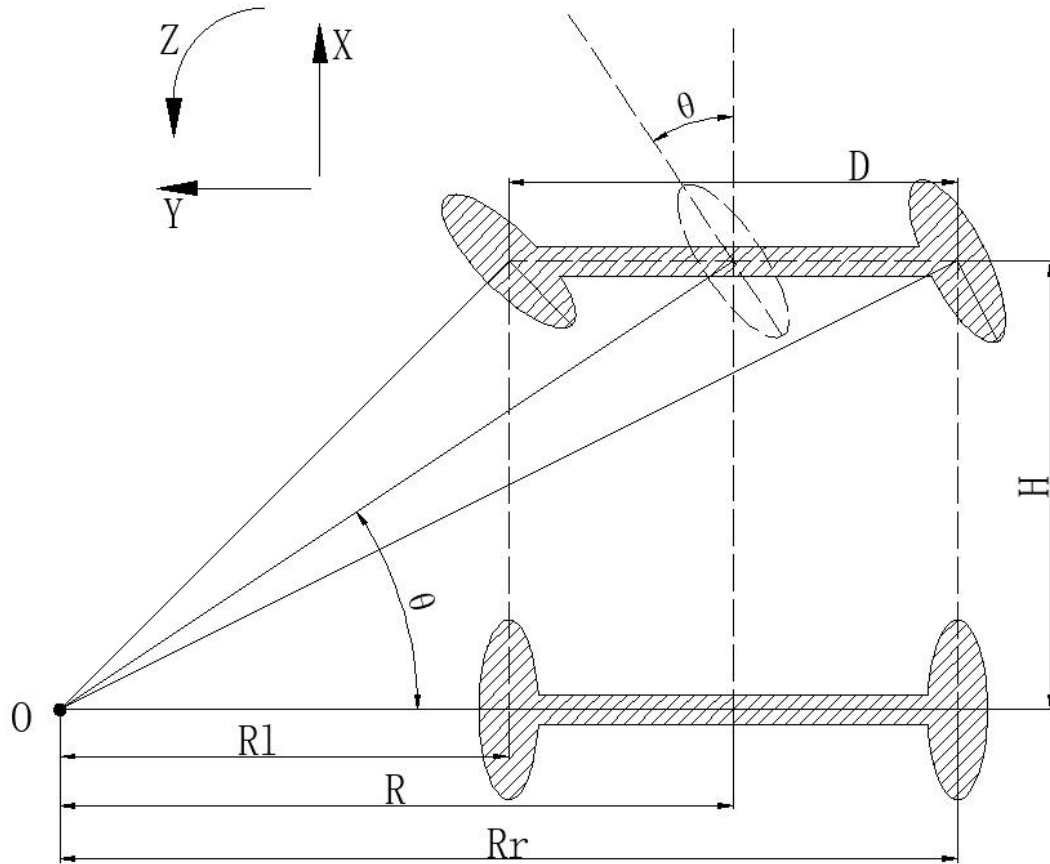


Figure 4 -2 Kinematics model of Ackerman car

### ① kinematic analysis

The difference between the Ackerman car and the two-wheel differential car is that the front wheels of the two-wheel differential car are omnidirectional wheels or universal wheels, while the front wheels of the Ackerman car are ordinary "one-way wheels". At this time to make Ackerman car to achieve pure rolling motion, it must ensure that the four wheels of the car normal direction of movement intersect at one point, the turning point compared with the center point , as shown in Figure 4-2 point O .

To simplify the model, suppose that the front wheel has only one wheel (the realization theory is the same ) , which is located in the middle of the front axle, as shown in the front wheel depicted by the dotted line in Figure 4-2 .

As seen from the Ackerman car kinematics model, When the current wheel

Angle is  $\theta$ , the car steering radius is  $R$ . Let the forward speed of the car be  $V$  (i.e.  $V_x$ ), the left wheel velocity  $V_L$  and right wheel velocity  $V_R$ , the angular

velocity can be obtained by the consistency:  $\frac{V}{R} = \frac{V_L}{R_L} = \frac{V_R}{R_R}$ ,

there  $\frac{H}{R} = \tan\theta$ ,  $R_L = R - \frac{D}{2}$ ,  $R_R = R + \frac{D}{2}$ . Then

$$V_L = \frac{V}{R} R_L = V * (1 - \frac{D * \tan\theta}{2H}), \quad V_R = \frac{V}{R} R_R = V * (1 + \frac{D * \tan\theta}{2H})$$

## ② C language implementation

```
void Kinematic_Analysis(float Vx, float Vy, float Vz)
{
    float angle=0;
    MOTOR_A.Target = Vx*(1-(WheelSpacing/(2*AxleSpacing))*tan(Vz*PI/180)); //The
    wheel speed should be calculated according to the actual front wheel angle Vz
    MOTOR_B.Target = Vx*(1+(WheelSpacing/(2*AxleSpacing))*tan(Vz*PI/180)); //The on-
    wheel speed should be calculated according to the actual front wheel angle Vz
    if(Vz>0) angle=Vz*PI/180*Servo_Wheel_ratio_L; // Symmetry processing on the left and
    right sides of the servo
    else angle=Vz*PI/180*Servo_Wheel_ratio_R; // Symmetry processing on the left and
    right sides of the servo
    if(1) Servo = -angle*K; // Here is the rotation angle of the servo
}
```

The input of the function is X and Y axis speed and front wheel steering angle,  $K$  is the correction coefficient of the steering gear control,  $Axle\_Spacing$  is the wheelbase parameter of the car ( front and rear ),  $Wheel\_spacing$  is the wheelbase parameter of the car ( left and right ).

### 4.3 Mecanum wheel car

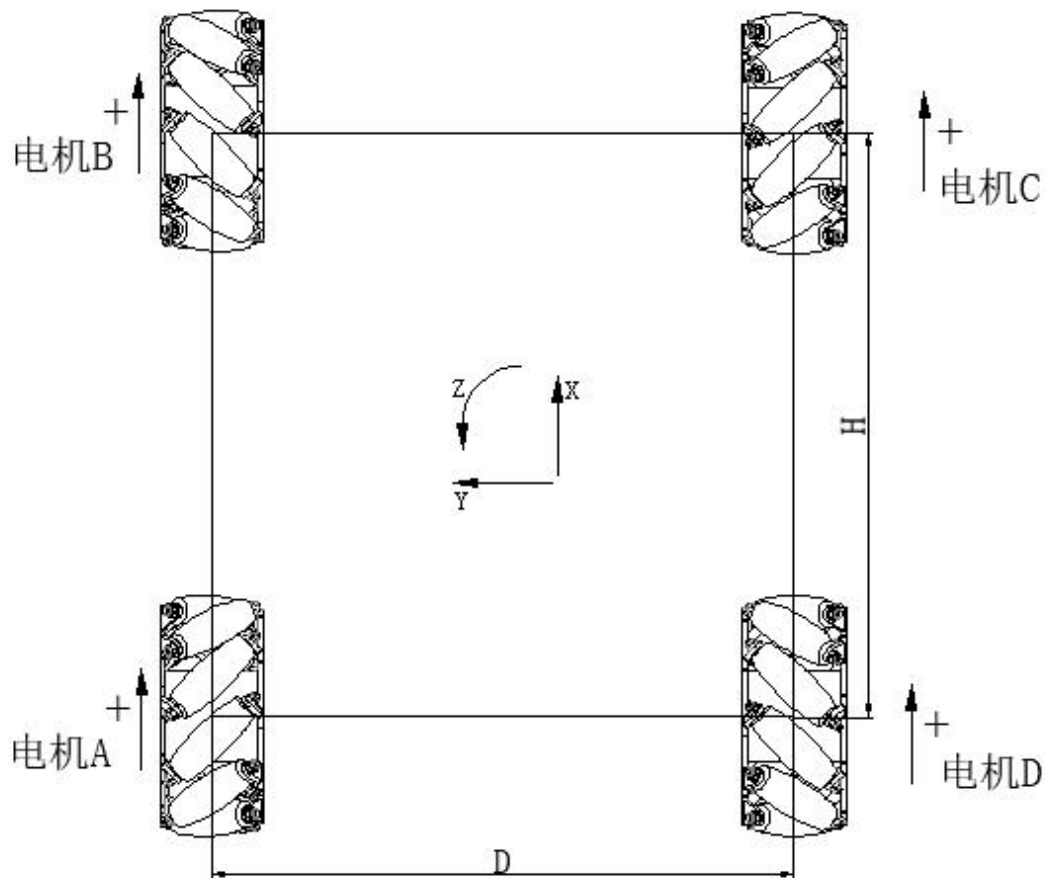


Figure 4 -3 Kinematics model of mecanum wheel car

#### ① kinematic analysis

To simplify the mathematical model of kinematics, the following two idealized assumptions are made:

- (1) The omni wheel does not slip with the ground, and the ground has sufficient friction;
- (2) The 4 wheels are distributed on the 4 corners of the rectangle or square, and the wheels are parallel to each other.

Here we linearly decompose the rigid body motion of the car into three components, then only need to calculate the speed of the four wheels when the wheel-wheel chassis is moving along the X+,Y+ and Z+ directions, and then the formula can be combined to calculate the rotational speed of the four wheels required for the "translation + rotation" movement synthesized by these three simple

movements .

Wherein ,  $V_A, V_B, V_C, V_D$  respectively A , B , C , D four wheel rotational speed, i.e. the rotational speed of the motor;  $V_x$  is the translation speed of the car along the X axis,  $V_y$  is the translation speed of the car along the Y axis, and  $\omega$  is the rotation

speed of the car along the Z axis;  $a = \frac{D}{2}$  is half of the tread D of the car, and  $b = \frac{H}{2}$

is half of the wheelbase H of the car.

When the car moves along the X axis:

$$V_{A=+} V_x \quad V_{B=+} V_x \quad V_{C=+} V_x \quad V_{D=+} V_x$$

When the car moves along the Y axis:

$$V_{A=+} V_y \quad V_{B=-} V_y \quad V_{C=+} V_y \quad V_{D=-} V_y$$

When the car rotates around the geometric center:

$$V_{A=+} \omega (a+b) \quad V_{B=+} \omega (a+b) \quad V_{C=+} \omega (a+b) \quad V_{D=+} \omega (a+b)$$

Combine the above three sets of equations to calculate the speed of the four wheels according to the motion state of the car:

$$V_A = V_x + V_y - \omega (a+b)$$

$$V_B = V_x - V_y - \omega (a+b)$$

$$V_C = V_x + V_y + \omega (a+b)$$

$$V_D = V_x - V_y + \omega (a+b)$$

## ② C language implementation

```
void Drive_Motor(float vx, float vy, float vz)
{
    MotorTarget.A = (vx+vy-vz*(Wheel_spacing+Wheel_axlespacing));
    MotorTarget.B = (vx-vy-vz*(Wheel_spacing+Wheel_axlespacing));
    MotorTarget.C = (vx+vy+vz*(Wheel_spacing+Wheel_axlespacing));
    MotorTarget.D = (vx-vy+vz*(Wheel_spacing+Wheel_axlespacing));
}
```

Wheel\_axlespacing is the wheelbase parameter of the car ( front and rear ) , and Wheel\_spacing is the wheelbase parameter of the car ( left and right ) .



## 4. 4 Omni wheel car

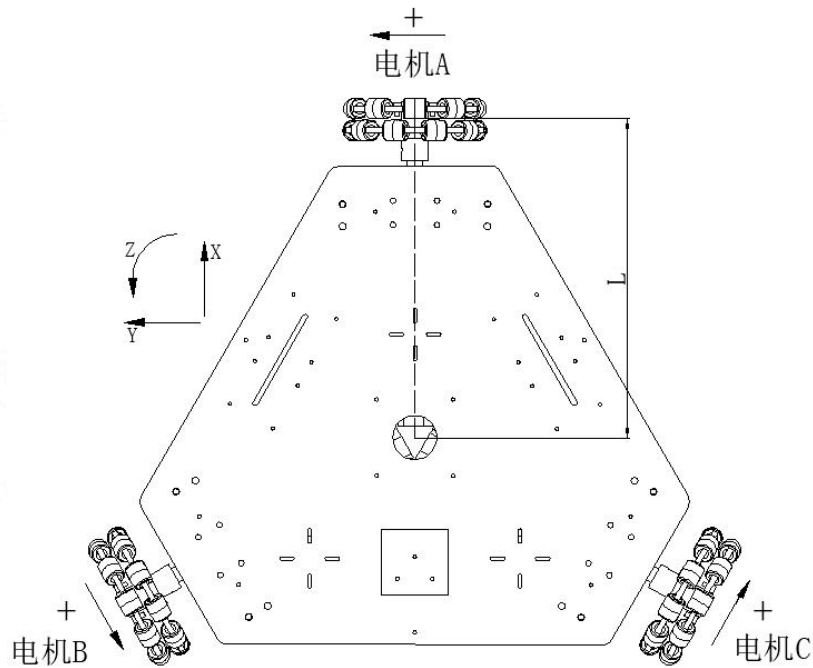


Figure 4 -4 Kinematics model of omni wheel car

### ① kinematic analysis

Before motion modeling, in order to simplify the mathematical model of kinematics, the following idealized assumptions are made:

- (1) The omni wheel does not slip with the ground, and the ground has sufficient friction;
- (2) The axis center of the motor is exactly the center of gravity of the chassis;
- (3) Each wheel is installed at  $120^\circ$  absolutely mutually.

Through simple speed decomposition, the following formula can be obtained:

$$V_A = V_Y + L\omega$$

$$V_B = -\cos 30^\circ V_x - \sin 30^\circ V_Y + L\omega$$

$$V_C = +\cos 30^\circ V_x - \sin 30^\circ V_Y + L\omega$$

That is ,  $V_A = V_Y + L\omega$

$$V_B = -\frac{\sqrt{3}}{2} V_x - \frac{1}{2} V_Y + L\omega ,$$

$$V_C = +\frac{\sqrt{3}}{2}V_x - \frac{1}{2}V_y + L\omega$$

$\omega$  is the angular velocity of the robot,  $L$  for the whole distance to the wheel center and the center of the chassis, and  $V_A, V_B, V_C$  respectively 3-wheel rotational speed,  $V_x$  and  $V_y$  are the motion speeds of the robot in  $X$  and  $Y$  directions.

### ② C language implementation

```
void Drive_Motor(float vx, float vy, float vz)
{
    MotorTarget.A = +vy+vz*Parament.Z;
    MotorTarget.B = -vx*Parament.X-vy*Parament.Y+vz*Parament.Z;
    MotorTarget.C = +vx*Parament.X-vy*Parament.Y-vz*Parament.Z;
}
```

$Parament.X = \frac{\sqrt{3}}{2}$ 、 $Parament.Y = \frac{1}{2}$ 、 $Parament.Z = 1$ , corresponding to our

kinematic analysis parameters.

## 4.5 Four-wheel drive car

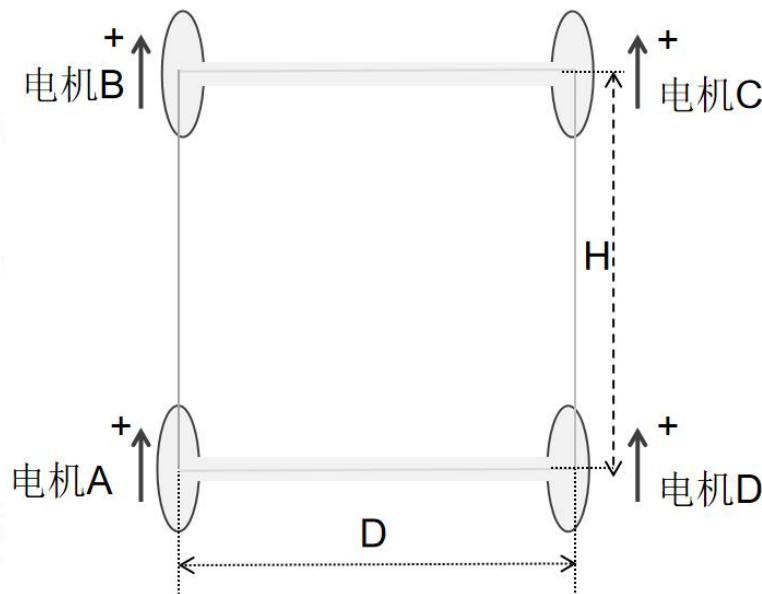


Figure . 4 -5 four-wheel kinematics model

### ① kinematic analysis

To simplify the mathematical model of kinematics, the following two idealized assumptions are made:

( 1 ) The wheels do not slip against the ground, and the ground has sufficient friction;

( 2 ) The 4 wheels are distributed on the 4 corners of the rectangle or square , and the wheels are parallel to each other.

Four-wheel drive vehicles use rubber wheels. Here we linearly decompose the rigid body motion of the car into two components, then just calculate and output the speed of the four wheels when the chassis translates in the X+ direction and rotates in the Z+ direction. Through the combination of the formula, we can calculate the rotation speed of the four wheels when the "translational + rotational" movement synthesized by these three simple movements is needed.

Where  $V_A, V_B, V_C$  and  $V_D$  are the rotating speeds of wheels A, B, C and D respectively, i.e. the rotational speed of the motor;  $V_x$  is the translation speed of the

car along the X axis;  $\omega$  is the rotation speed of the car along the Z axis;  $a = \frac{D}{2}$  is half

of the trolley wheelbase D,  $b = \frac{H}{2}$  is half of the trolley wheelbase H.

When the car moves along the X axis:

$$V_A = +V_x \quad V_B = +V_x \quad V_C = +V_x \quad V_D = +V_x$$

When the car rotates around the geometric center:

$$V_A = +\omega (a+b) \quad V_B = +\omega (a+b) \quad V_C = +\omega (a+b) \quad V_D = +\omega (a+b)$$

Combine the above three sets of equations to calculate the speed of the four wheels according to the motion state of the car:

$$V_A = V_x - \omega (a+b)$$

$$V_B = V_x - \omega (a+b)$$

$$V_C = V_x + \omega (a+b)$$

$$V_D = V_x + \omega (a+b)$$

## ② C language implementation

```
void Drive_Motor(float vx,float vy,float vz)
{
    MotorTarget.A    = (vx-vz*(Wheel_spacing+Wheel_axlespacing));
    MotorTarget.B    = (vx-vz*(Wheel_spacing+Wheel_axlespacing));
    MotorTarget.C    = (vx+vz*(Wheel_spacing+Wheel_axlespacing));
    MotorTarget.D    = (vx+vz*(Wheel_spacing+Wheel_axlespacing));
}
```

Wheel\_axlespacing is the wheelbase parameter of the car ( front and rear ) , and Wheel\_spacing is the wheelbase parameter of the car ( left and right ) .

## 4. 6 PI control program source code

What is obtained through kinematic analysis is the target speed of the motor. We need to send this target value to the PID controller for speed closed-loop control, so that the actual output speed of the motor approaches the target value. The PI controller source code in the program is as follows:

```

/*****
* Function: Incremental PI controller speed control
* Entry parameters: Encoder : encoder measurement value, Target : target speed
* Return value: Pwm : Motor PWM
* Function description: pwm+=Kp[e ( k ) -e(k-1)]+Ki*e(k) incremental PI control
*****/
int Incremental_PI_A (float Encoder,float Target)
{
    static float Bias,Pwm,Last_bias;
    Bias=Target-Encoder;           // Calculate the deviation
    Pwm+=Velocity_KP*(Bias-Last_bias)+Velocity_KI*Bias;    // Incremental PI
controller
    if (Pwm>7200) Pwm=7200;
    if (Pwm<-7200) Pwm=-7200;
    Last_bias=Bias;                //Save the last bias
    return Pwm;                    // Incremental output
}

```

## 5. Wiring instructions

This chapter mainly demonstrates several key wiring instructions. Please refer to the figure directly for specific wiring. The STM32 controller integrates dual 5V power supplies: The STM32 controller has two 5V power outputs; one 5V power supply supplies power to the STM32 controller and peripherals (encoders, Bluetooth, handles, etc.), and the other 5V power supply outputs to the Raspberry Pi power supply.

### ① Raspberry Pi power supply

The 5V power circuit of the Raspberry Pi is integrated on the adapter board of the STM32 controller, and it uses a TYPE-C to TYPE-C cable that can pass a current of more than 3A .



Figure 5-1-1 Raspberry Pi power supply wiring

### ② Serial communication between Raspberry Pi and STM32 controller

Because the Raspberry Pi is used as a host computer to communicate with the STM32 controller, the default selection is the serial port 3 integrated with the CP2102 level conversion chip .



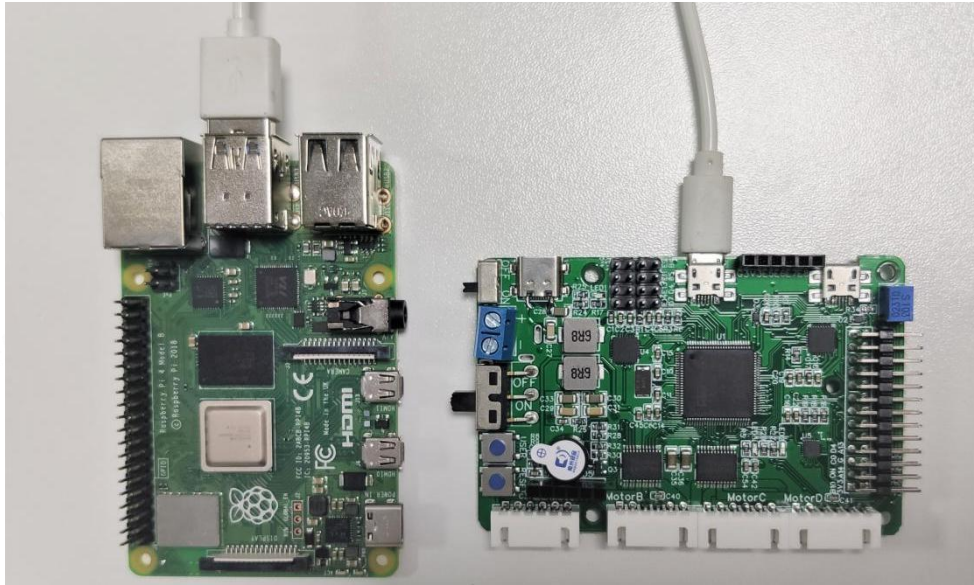


Figure 5-1-2 Raspberry Pi and STM32 communication

③ Raspberry Pi connect to the navigation radar

The connection between the Raspberry Pi and the radar here is a normal Micro-USB cable. The Raspberry Pi also communicates with the radar while powering the radar.

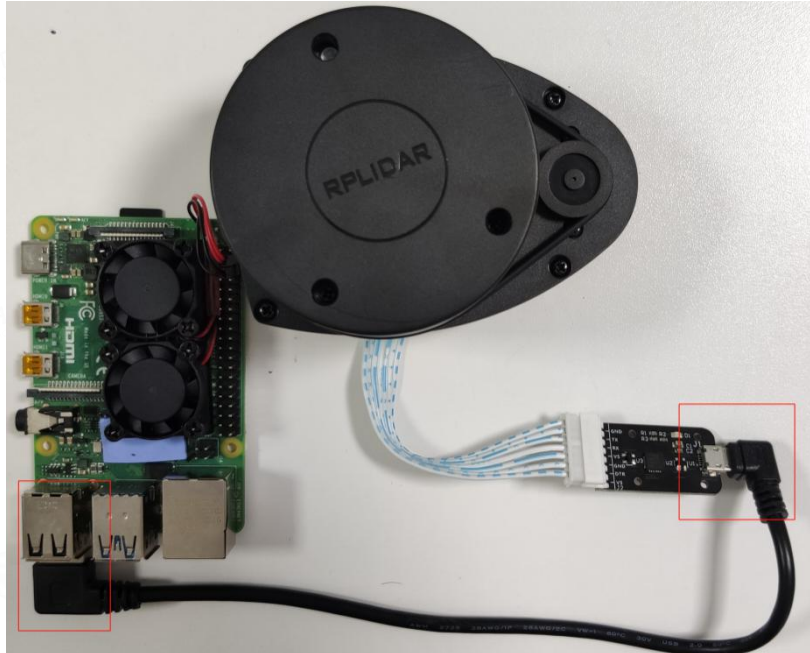


Figure 5-1-3 Wiring of Raspberry Pi and Lidar

④ A detailed description diagram of the peripherals of the STM32 controller

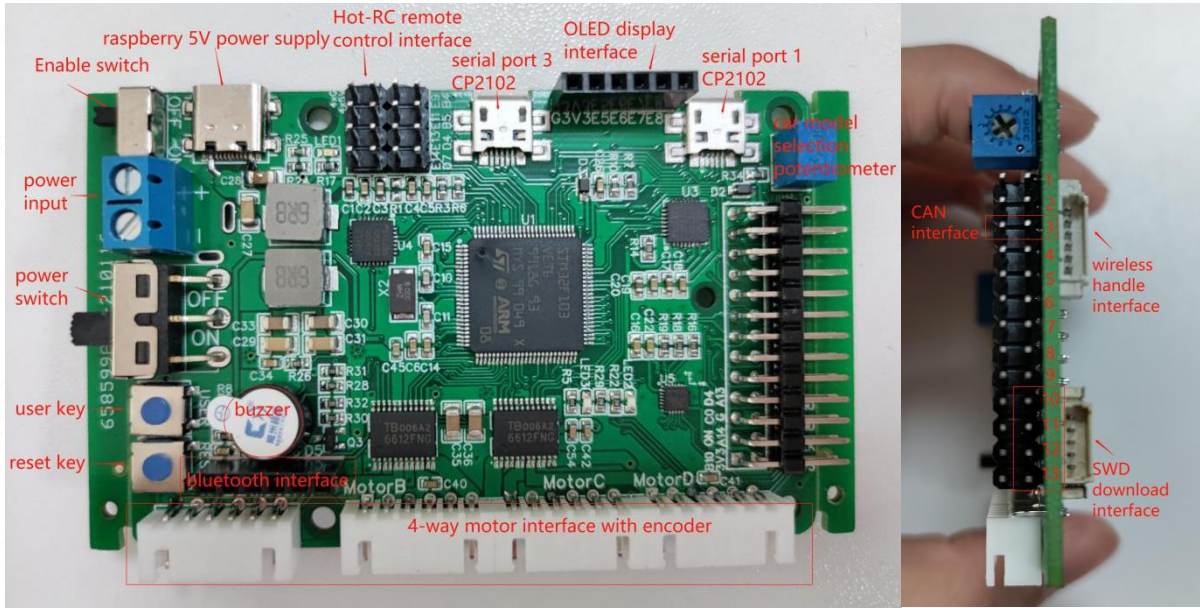


Figure 5-1-4 STM32 controller description diagram

# 6. Control flow chart

## 6. 1 Control flowchart of robot motor

The robot supports 6 control modes, and the principles of these 6 control modes are realized by changing the target speed of the robot. The target speed obtains the actual output of each motor through the kinematic analysis function, and finally realizes the speed control of the motor through the PID controller (PID speed control function).

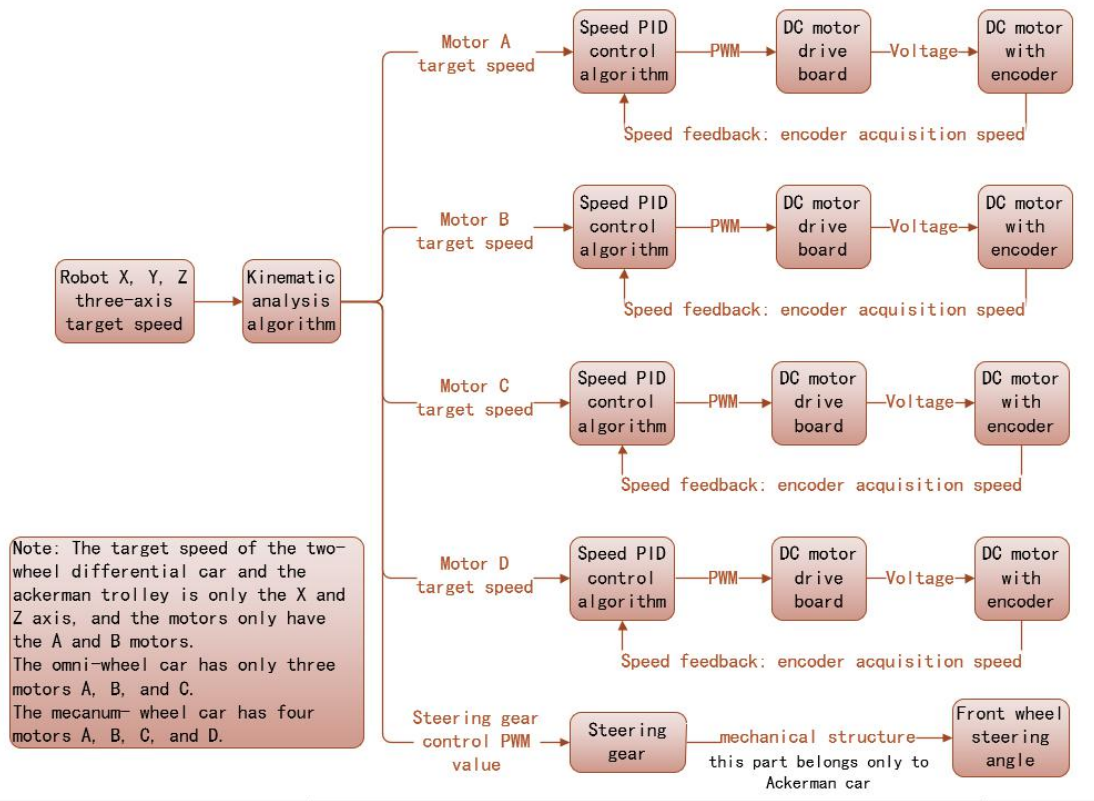


Figure 6-1 Robot motor control flow chart



## 6.2 Robot STM32 program structure diagram

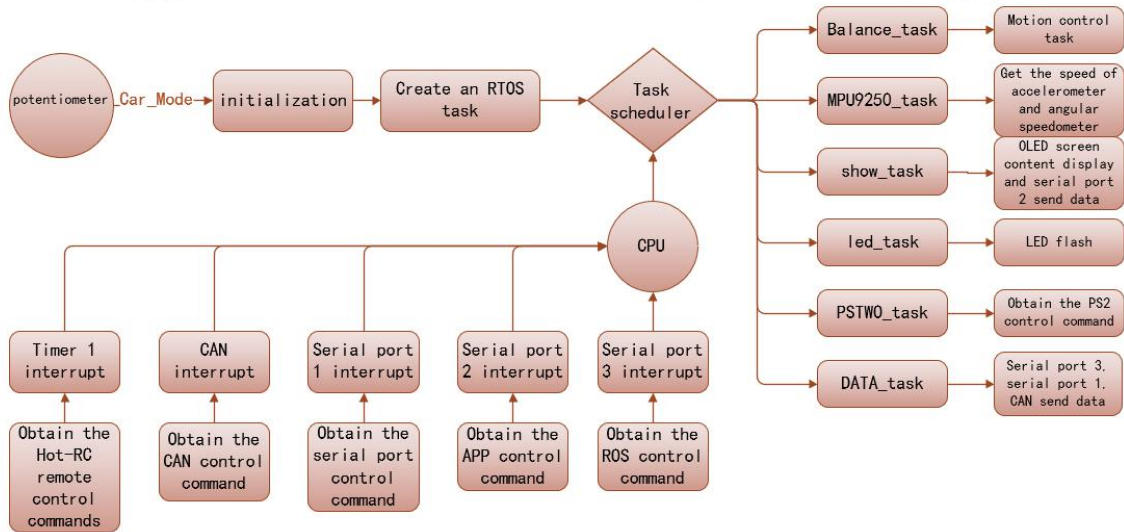


Figure 6-2 The program execution flowchart of the robot STM32 controller

The RTOS task scheduler determines the execution order of tasks according to the priority of the task ( the task order in Figure 6-2 does not represent the task priority, the specific task priority needs to check the priority setting in the program), and the execution time of each task is very short, so it is almost equivalent to executing all tasks at the same time. If an interrupt occurs during this period, it will respond to the interrupt. The serial port 2 interrupt is used for APP Bluetooth control, and the serial port 3 interrupt is used to receive information from ROS .

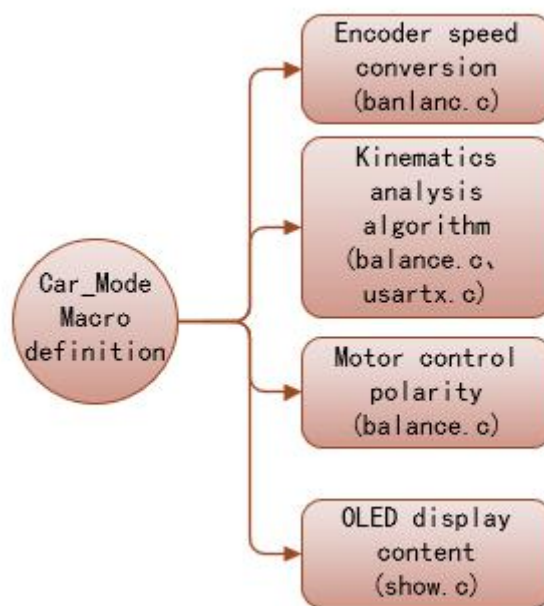


Figure 6 -3 Part of the program affected by Car\_Mode

### 6.3 Robot controller connection diagram

Many controllers and peripherals are used in the robot, including: Raspberry Pi ( Jetson Nano ), laser radar, STM32 controller, motor, encoder, dual-channel drive, Bluetooth, PS2 handle, Hot-RC remote control, gyroscope, etc., At the same time, serial port 1 and CAN interface are provided to facilitate users to expand control. The connections between these controllers and controllers, and peripherals and controllers are shown in Figure 6-3 .

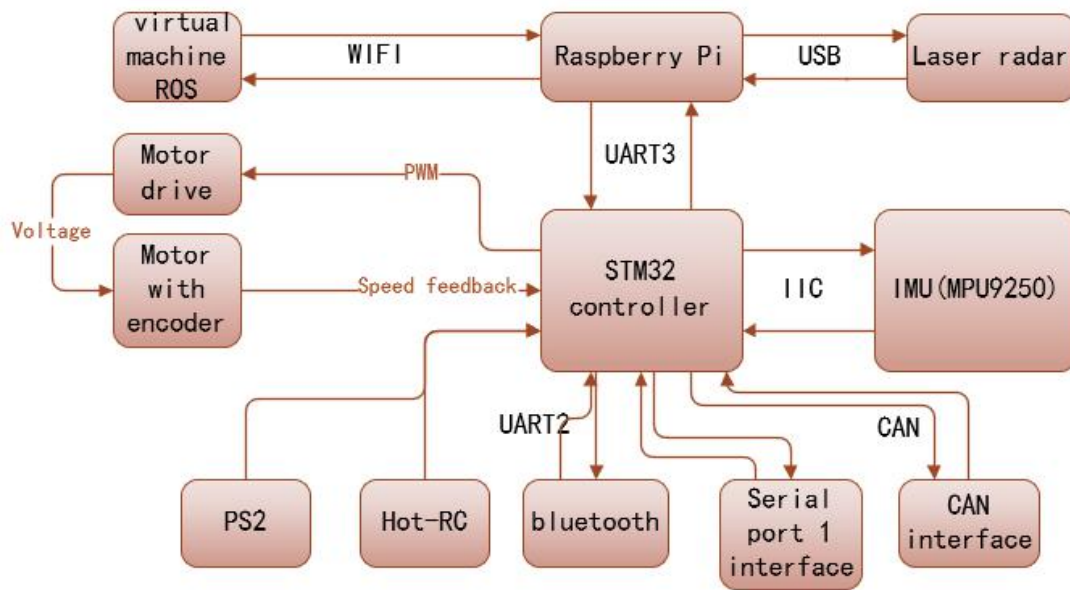


Figure 6-4 Robot controller connection diagram



## 7. Matters needing attention

### 7.1 About the code

The programming method on the robot STM32 controller is based on the RTOS system, which is different from the interrupt control method. The RTOS is executed in the form of tasks in turn, and tasks with higher priority have higher execution priority (interrupt priority is higher than task priority). It should be noted that if a task has an execution logic error, the program will be stuck at the error and cannot continue. For example: If there is a sending task of serial port 3 in the program, but the serial port 3 is not initialized or the initialization code is wrong, the program will get stuck when executing the sending task of serial port 3. Therefore, if the program is stuck during debugging and cannot be executed normally, you need to check whether there is a bug stuck o task by task .

### 7.2 About the power interface on the adapter board

The 5V and 3.3V pins on the adapter board can supply external power, but they cannot carry loads with too much current. Among them, 5V output is not recommended to carry a load exceeding 1.5A, and 3.3V output is not recommended to carry a load exceeding 200mA. As you can see from the schematic diagram of the adapter board provided by us, the adapter board is equipped with two 5V power circuits, one of which supplies power to the basic peripherals, and the other independently supplies power to the Raspberry Pi ( Jetson Nano ) ( USB female socket ). Wiring instructions are explained in detail in Chapter 5 " 5.Wiring Instructions".

### 7.3 About the motor

Avoid blocking the motor during use, otherwise it is easy to burn the motherboard. Before the robot is completely tested and passed, please stand up the robot and let the motor hang in the air to avoid unnecessary damage caused by

misoperation.

## 7.4 About the battery

The battery voltage is displayed on the display. When the battery is low (less than 10.8V ), please charge it in time. The battery comes with over-discharge and over-charge protection. Please do not use the battery when charging the battery. The battery charging connection is shown in Figure 7-4-1 .



Figure 7-4-1 Wiring diagram of robot battery and charger

After the battery is charged, the cover of the charging port should be closed to prevent the battery from being disconnected by accidental touch, as shown in Figure 7-4-2 .



Figure 7-4-2 Robot battery charging cable interface

## 8. How to download program to STM32 controller

The STM32 controller can download the program through the serial port or the SWD interface. The serial port is downloaded via the USB data cable, and it is given by default. The SWD interface recommends using the STlink of the metal shell to download, and you need to bring your own.

### 8.1 Serial download

The motherboard uses a one-click download circuit, which is very convenient for downloading programs. All you need is a microUSB cable.

#### ① Hardware preparation

hardware:

1. STM32 controller
2. MicroUSB mobile phone data cable

#### ② Software preparation

Software: mcuisp recovering software (included in the attached information), the corresponding USB to TTL module CH340G driver. There are also drivers in the attached materials. If the driver installation is really difficult, download a driver wizard.

After the installation is successful, you can open the device manager to see, you can see that the driver has been installed successfully, otherwise there will be a red exclamation mark.



Figure 8-1-1 Device manager view CP210x driver

The wiring of the serial port download program is very simple, just connect the

data cable to the computer and the board. Open the mcuisp software in the attached document and set it according to the operation sequence in Figure 8-1-2 .

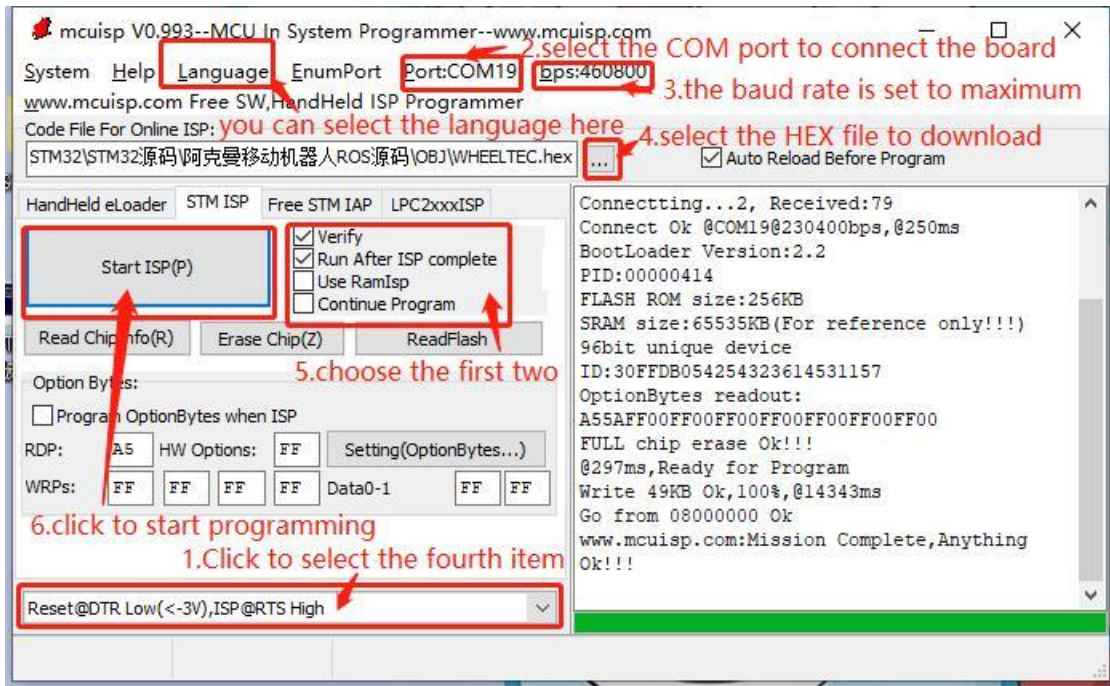


Figure 8-1-2 mcuisp download program configuration description

OK, everything is ready, and then click to start programming, the program can be downloaded. Since the option to execute after programming is checked, the program will run automatically after downloading. (Note: It is forbidden to check the Program OptionBytes when ISP . If you are using an F4 board, the baud rate needs to be set to 76800)

## 8.2 SWD download

The STM32 controller can download the program through the SWD interface, there are marks on the motherboard, PA13 and PA14 respectively .

### ① Hardware preparation

1. STM32 controller
2. STlink

### ② Software preparation

Install the corresponding STlink or Jlink driver.

After the installation is successful, you can open the device manager to see if the STLink device is displayed again .

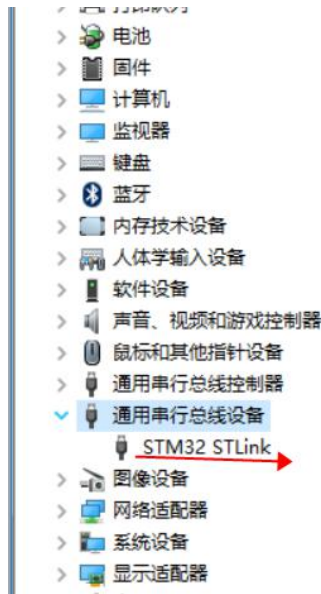


Figure 8-2-1 Device manager view STlink driver

You can see that the driver has been successfully installed!

### ③ Wiring

STlink ----- STM32 controller  
SWDIO-----PA13  
SWCLK-----PA14  
GND-----GND

OK, everything is ready.

### ④ program

Click the button pointed by the arrow in Figure 8-2-2 , and the program can be downloaded! Since the option to execute after programming is checked, the program will run automatically after downloading. The default program configuration is for STlink , if you need to configure Jlink download, you need to modify the MDK settings.



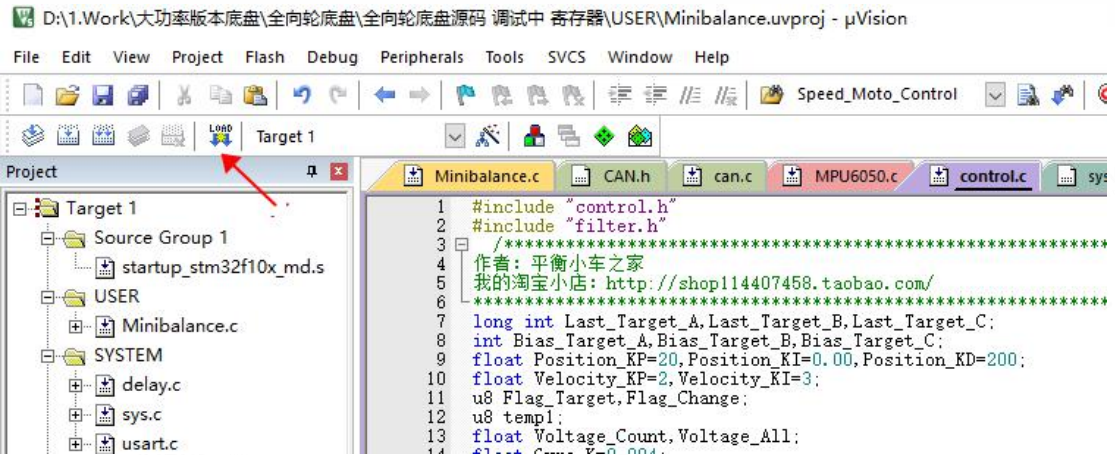


Figure 8-2-2 STLink download program interface