

# 东莞市微宏智能科技有限公司

淘宝店铺: [minibalance.taobao.com](http://minibalance.taobao.com)

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

## STM32 运动底盘 开发手册

推荐关注我们的公众号获取更新资料



版本说明:

版本	日期	内容说明
V1.0	2020/08/10	第一次发布
V2.0	2020/09/28	第二次发布
V3.0	2020/12/08	第三次发布
V3.5	2021/01/28	多处修正

# 序言

ROS 导航机器人的全套教程包括：“STM32 运动底盘开发手册”，“Ubuntu 配置教程”，“ROS 开发教程”三个文档。关于虚拟机和树莓派（Jetson Nano）的 Ubuntu 配置教程请看“Ubuntu 配置教程”；关于 ROS 开发的教程请看“ROS 开发教程”。该文档教程内容主要用于：讲解 ROS 机器人运动底盘的运动学分析，通信协议、控制模式等。运动底盘上搭载了两个控制器，分别是树莓派（Jetson Nano、Jeston TX2、工控机等）和 STM32 控制器，两者之间通过串口通信来传输数据，关于接线的详细说明请看第五章。其中树莓派（Jetson Nano、Jeston TX2、工控机等）上安装了 Ubuntu，用于运行 ROS；STM32 控制器用于控制运动底盘和采集里程计信息、电池信息、IMU 信息。

本文档中出现的任务名称、函数名称等都是运动底盘内置程序的内容，如果您不打算阅读机器人 STM32 控制器的程序，可以忽略下文出现的程序函数、程序任务（例：APP\_Show() 函数，发送任务 data\_task，data\_transition() 函数），不影响对整个开发手册的阅读和理解。

本文档适用于 ROS 教育机器人系列的：两轮差速小车，阿克曼小车，麦克纳姆轮小车，全向轮车，履带车，四驱车这六款教育机器人。

# 目录

序言.....	2
1. 机器人的控制模式.....	5
1.1 机器人运动速度单位.....	5
1.2 ROS(串口 3)控制.....	6
1.3 APP 控制.....	13
1.4 PS2 控制.....	16
1.5 航模遥控.....	17
1.6 CAN 控制.....	19
1.7 串口 1 控制.....	21
2. OLED 显示内容.....	23
2.1 OLED 具体内容.....	23
2.2 OLED 通用显示内容.....	25
2.3 小车自检.....	25
3. 陀螺仪零点漂移消除.....	26
4. 机器人运动学分析.....	27
4.1 两轮差速(履带车)小车.....	27
4.2 阿克曼小车.....	29
4.3 麦轮小车.....	31
4.4 全向轮小车.....	33
4.5 四驱车.....	35
4.6 PI 控制程序源码.....	37
5. 接线说明.....	38
6. 控制流程图.....	40
6.1 机器人电机的控制流程图.....	40

6.2 机器人 STM32 程序结构图.....	41
6.3 机器人控制器连接图.....	42
7. 注意事项.....	43
7.1 关于代码.....	43
7.2 关于转接板上的电源接口.....	43
7.3 关于电机.....	43
7.4 关于电池.....	43
8. 如何给 STM32 控制器下载程序.....	45
8.1 串口下载.....	45
8.2 SWD 下载.....	46

# 1. 机器人的控制模式

这一章主要是对机器人的控制和使用做一个详细的说明。机器人支持 APP 遥控、PS2 有线手柄、ROS 控制、航模遥控控制、CAN 控制、串口控制这 6 种控制方式。控制方式显示在 OLED 显示屏的左下角，默认开机后使用 ROS 控制的方式。

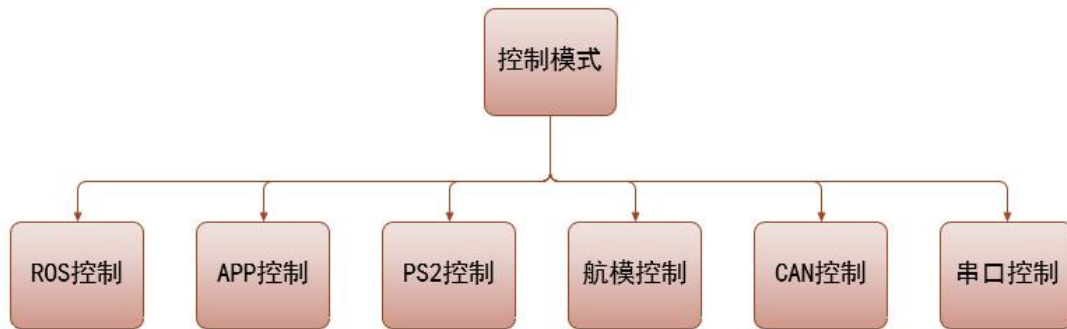


图 1-1 机器人控制模式

## 1.1 机器人运动速度单位

这里说明一下机器人运动速度的单位，是 m/s(米/秒)，具体的“编码器原始数据”转换成“m/s(米/秒)”的计算公式见图 1-1-1。

Encoder\_A\_pr 是编码器原始数据,CONTROL\_FREQUENCY 是控制频率(单位: HZ), Encoder\_precision 是编码器精度(与电机机械结构和编码器芯片有关), wheel\_perimeter 是轮子的周长(单位: 米), 最终得到的 MOTOR\_A.Encoder 就是机器人的实际运动速度(单位: 米/秒)。这里只展示 A 电机的编码器数据转换, 其余的 B、C、D 电机的编码器计算转换同理。

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

图 1-1-1 机器人运动速度转换公式

XYZ 三轴速度正方向标定如下图 1-1-2 所示。

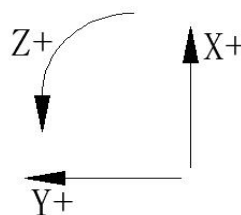


图 1-1-2 机器人 XYZ 三轴速度正方向

## 1.2 ROS(串口 3)控制

机器人上电后默认是使用 ROS 的控制模式。本小节只介绍如何通过 ROS 的环境让机器人动起来，具体的原理以及 ROS 的一些相关定义概念请看“ROS 开发手册”。关于虚拟机和 ros 开发环境搭建以及如何虚拟机如何远程登录机器人上的 ubuntu 系统，请看“Ubuntu 配置教程”。

这里是使用虚拟机端的键盘去控制机器人运动。

### ① 虚拟机远程登录机器人 ubuntu 系统

要注意的是使用虚拟机控制机器人运动需要分别打开两个终端，每个终端都要单独的远程登录机器人 ubuntu 系统。在虚拟机输入如图 1-2-1 所示的指令远程登录机器人的 ubuntu 系统。

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

图 1-2-1 ssh wheeltec@192.168.0.100

### ② 运行功能包

在第一个终端中先打开机器人的“turn\_on\_wheeltec\_robot”功能包下的“turn\_on\_wheeltec\_robot”使能机器人控制节点。

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

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

在第二个终端中打开键盘控制机器人运动的节点，在第二个终端中打开机器人的“wheeltec\_robot\_rc”功能包下的“keyboard\_teleop”键盘控制节点。

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

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

打开键盘控制节点后可以看到一些控制提示信息，这个时候就可以使用键盘去控制机器人运动了，具体的键盘控制对应实现的效果见表 1-1，按“ctrl+c”或者直接关闭终端来退出控制节点。



```
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
```

图 1-2-4 打开键盘控制节点终端的信息

表 1-1 键盘控制机器人运动指令说明

键盘按键	u	i	o	j	k	l	m	,
机器人实 现效果	左前 运动	前进 运动	右前 运动	左转	急停	右转	左后 运动	后退 运动
键盘按键	.	空格	q	z	w	x	e	c
机器人实 现效果	右后 运动	急停	运动 速度 +10%	运动 速度 -10%	线 速度 +10%	线 速度 -10%	角 速度 +10%	角 速度 -10%

### ③ STM32 向 ROS 发送数据

ROS 和 STM32 控制器（运动底盘）之间通过串口实现通信，STM32 控制器使用的是串口 3，波特率是 115200。通信协议包括：STM32 控制器向 ROS 发送数据，ROS 向 STM32 控制器发送数据。打开 STM32 控制器的源码，关于串口通信的代码都在 STM32 控制器的 usartx.c 文件。

STM32 向 ROS 发送数据使用一个 date\_task 的任务按照 20hz 的频率执行。发送的数据包括：帧头帧尾、机器人使能标志位、机器人 XYZ 三轴速度、IMU 三轴加速度、三轴角速度、电池电压、数据校验位,详细发送的数据见下文的表 1-2。

发送数据的方式：将要发送的数据打包到一个数组中，数组的长度是 24 字节，使用串口逐位发送出去。因为串口一次只能发送一个 8 位（1 个字节）的数据，因此 2 个字节（short 型）的数据会拆分成高 8 位和低 8 位发送。

发送数据前对数据赋值的函数是“usartx.c”文件中的“data\_transition()”函数；发送数据的函数是“usartx.c”文件中的“USART3\_SEND()”函数。如果您

需要更改发送的数据内容，直接更改 data\_transition() 函数即可；如果您需要更改发送数据的长度，需要更改“SENT[]”这个数组长度的同时还需要

“USART3\_SEND()”函数中 for 循环的次数，并且 ROS 端接收数据长度也要做相应的修改。

在发送的数据中需要说明的是，其中帧头是固定值 0X7B，帧尾是固定值 0X7D，flag\_stop 是电机的停止标志位（0 是使能、1 是失能）。数据校验位的计算方式是 BCC 校验（将全部数据位（包括帧头）异或），最终得到的结果就是数据校验位，数据校验位的计算过程见图 1-2-6。

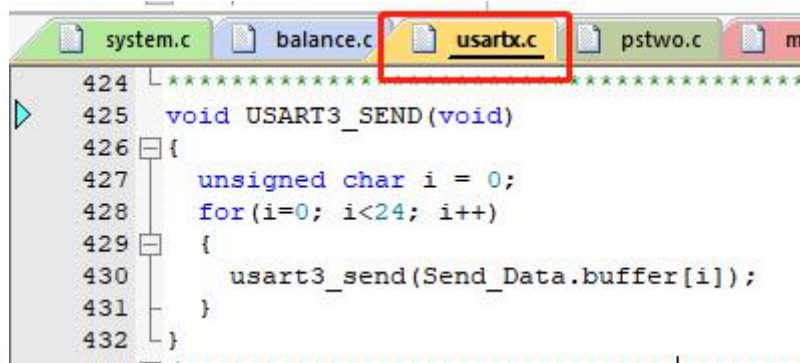


图 1-2-5 USART3\_SEND() 函数



图 1-2-6 数据校验位计算函数



表 1-2 STM32 向 ROS 发送的数据

数据内容	帧头 (固定值 0X7D)	flag_stop	机器人 X 轴 速度		机器人 Y 轴 速度		机器人 Z 轴 速度		加速度计 X 轴 加速度		加速度计 y 轴 加速度	
数据类型	Uint8	Uint8	short		short		short		short		short	
占用字节	1	1	2		2		2		2		2	
数组号	1	2	3	4	5	6	7	8	9	10	11	12
数据内容	加速度计 z 轴 加速度	角速度计 X 轴 角速度	角速度计 Y 轴 角速度		角速度计 Z 轴 角速度		电池 电压		数据 校验位		帧尾 (固定值 0X7B)	
数据类型	short	short	short		short		short		Uint8		Uint8	
占用字节	2	2	2		2		2		1		1	
数组号	13	14	15	16	17	18	19	20	21	22	23	24

这里还有一个地方需要注意，机器人 XYZ 三轴速度、加速度计、角速度计以及电池电压的原始数据是浮点型的数据（float），因为浮点型数据使用串口传输不方便，所以这四个数据在发送之前先将浮点数放大一千倍（保留小数点后三位），再将放大后的浮点数强制转换成 short 型数据，最后在发送前将 short 型数据拆分成两个 8 位的数据。相应的，上位机端在接收到数据后，需要将接收到的数据两个 8 位数据合并后转换为 short 型，在缩小一千倍来进行单位的转换。

下面讲解一下如何将两个 8 位数据合并后转换为 short 型，即得到我们的实际速度等数据：我们的控制量单位是 mm/s(0.001m/s)，控制量方向由高 8 位数据的最高位决定。

示例 1：21 B6=0010 0001 1011 011 ，最高位为 0，正数，速度大小为 21  
 $B6 = (2 * 16 + 1) * 256 + (11 * 16 + 6) = (2 * 16 + 1) * 256 + (11 * 16 + 6) = 8630(\text{mm/s})$ ;

示例 2: A1 2F=1010 0001 0010 1111, 最高位为 1, 负数, 速度大小为  $2^{16}(\text{FF FF}+1)-\text{A1 2F}=5\text{E D0}+1=(5*16+\text{E})*256+(\text{D}*16+0)+1=24272(\text{mm/s})$ 。

下图为我们连接串口 3 后通过串口助手接收到的实际数据。(注意我们的串口 3 是没有集成 CH340 的, 需要使用我们配备的 ROS 与 STM32 通信用的数据线, 串口助手才可以与 STM32 的串口 3 通信)。



图 1-2-7 小车串口 3 发送的数据

下面对这接收到的 24 个字节数据转换一下:

第 1 个字节: 0X7B, 帧头;

第 2 个字节: 0X00, 电机处于非停止状态;

第 3、4 个字节: X 轴速度, 高 8 位 0X01 (16 进制)=0000 0001 (2 进制)、低 8 位 0X01 (16 进制)=0000 0001 (2 进制), 最高位为 0, 正数(前进), 速度大小为  $1*256+1=257(\text{mm/s})$ , 该速度为小车对编码器数据进行计算得到的实际速度。

第 5、6 个字节: Y 轴速度, 高 8 位 0X00 (16 进制)=0000 0000 (2 进制)、低 8 位 0X01 (16 进制)=0000 0001 (2 进制), 最高位为 0, 正数(左移), 速度大小为  $0*256+1=1(\text{mm/s})$ , 该速度为小车对编码器数据进行计算得到的实际速度。

第 7、8 个字节：Z 轴速度，高 8 位 0X00(16 进制)=0000 0000(2 进制)、低 8 位 0X00(16 进制)=0000 0000(2 进制)，最高位为 0，正数(逆时针旋转)，速度大小为  $0 \times 256 + 0 = 0$  (0.001rad/s)，该速度为小车对编码器数据进行计算得到的实际速度。

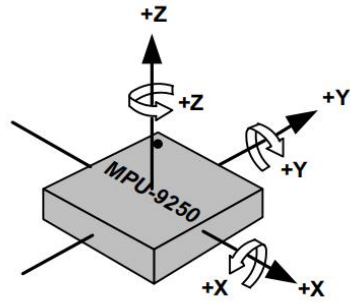


图 1-2-8 MPU9250 加速度计、角速度计三轴示意图

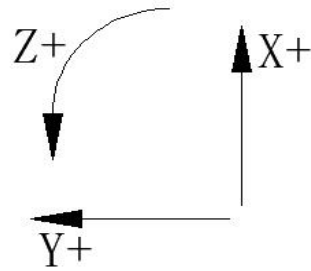


图 1-2-9 机器人 XYZ 三轴方向标定

接下来 12 个字节为三轴加速度计和角速度计的数据，注意这里 XYZ 三轴方向有所改变，X 轴正方向为向右，Y 轴正方向为向前，Z 轴正方向为上升。加速度计和角速度计的速度皆为绕 XYZ 三轴旋转的速度，如上图 1-2-8 所示。因为陀螺仪的 XY 轴方向与我们标定的机器人 XY 轴方向是有差别的，因此在 XY 轴的角速度和角速度数据在发送做了一个矫正的转换。

第 9、10 个字节：X 轴加速度，高 8 位 0XFE(16 进制)=1111 1110(2 进制)、低 8 位 0X96(16 进制)=1001 0110(2 进制)，最高位为 1，负数，大小为  $2^{16}(\text{FF} + 1) - \text{FE}96 = 0169 = 1 \times 256 + 105 = 361$ ，转换为加速度  $361 / 1672 = 0.2159 (\text{m/s}^2)$ 。

第 11、12 个字节：Y 轴加速度，高 8 位 0XFD(16 进制)=1111 1101(2 进制)、低 8 位 0XCE(16 进制)=1100 1110(2 进制)，最高位为 1，负数，大小为  $2^{16}(\text{FF} + 1) - \text{FDC E} = 0231 = 2 \times 256 + 49 = 561$ ，转换为加速度  $125 / 1672 = 0.3355 (\text{m/s}^2)$ 。

第 13、14 个字节：Z 轴加速度，高 8 位 0X40(16 进制)=0100 0000(2 进制)、低 8 位 0X80(16 进制)=1000 0000(2 进制)，最高位为 0，正数，大小为  $64 \times 256 + 128 = 16512$ ，转换为加速度  $15684 / 1672 = 9.8756 (\text{m/s}^2)$ 。

第 15、16 个字节：X 轴角速度，高 8 位 0XFF(16 进制)=1111 1111(2 进制)、低 8 位 0XFB(16 进制)=1111 1011(2 进制)，最高位为 1，负数，大小为  $2^{16}(\text{FF} + 1) - \text{FFFB} = 0004 = 0 \times 256 + 4 = 4$ ，转换为角速度  $9 / 3753 = 0.0011 (\text{rad/s})$ 。

第 17、18 个字节：Y 轴角速度，高 8 位 0X00(16 进制)=0000 0000(2 进制)、低 8 位 0X07(16 进制)=0000 0111(2 进制)，最高位为 0，正数，大小为  $0 \times 256 + 7 = 7$ ，转换为角速度  $7/3753 = 0.0019(\text{rad/s})$ 。

第 19、20 个字节：Z 轴角速度，高 8 位 0X00(16 进制)=0000 0000(2 进制)、低 8 位 0X01(16 进制)=0000 0001(2 进制)，最高位为 0，正数，大小为  $0 \times 256 + 1 = 1$ ，转换为角速度  $1/3753 = 0.0003(\text{rad/s})$ 。

第 21、22 个字节：电池电压，高 8 位 0X58(16 进制)=0101 1000(2 进制)、低 8 位 0X38(16 进制)=0011 1000(2 进制)，最高位为 0，正数，大小为  $88 \times 256 + 56 = 22584$ ，电压大小为 22584mv(毫伏)。

第 23 字节：BBC 校验位(前 22 字节异或)，  
 $0X83 = 0X7B \oplus 0X00 \oplus 0X01 \oplus 0X01 \oplus 0X00 \oplus 0X01 \oplus 0X00 \oplus 0X00 \oplus 0XFE \oplus 0X96 \oplus 0XFD \oplus 0XCE \oplus 0X40 \oplus 0X80 \oplus 0XFF \oplus 0XFB \oplus 0X00 \oplus 0X07 \oplus 0X00 \oplus 0X01 \oplus 0X58 \oplus 0X38$ 。

(如果您想验证这个检验位的结果，可以使用一些网页在线平台来计算校验位)

第 24 个字节：0X7D，帧尾；

#### ④ STM32 接收 ROS 发送过来的数据

STM32 控制器板子上搭载了 CH340(串口 1)和 CP210(串口 3)两路串口通信接口，两路串口接收数据处理的程序完全一致，默认使用 CP2102(串口 3)和 ROS 端进行串口通信，以下以串口 3 接收数据为例进行说明。

接收数据采用中断接收的方式，接收的数据包括机器人产品信号、使能控制标志位、机器人三轴目标速度、数据校验位。

帧头帧尾默认是固定值；flag\_stop 是机器人的使能控制位，默认发送使能；机器人三轴目标速度用于控制机器人运动，具体的接收内容见表 1-3。需要注意的是，表格中的数组号是上位机发送数据的数组号。



表 1-3 STM32 接收 ROS 发送过来的数据

数据内容	帧头 (固定值 0X7D)	预留位	预留位	机器人 X 轴 目标速度		机器人 Y 轴 目标速度		机器人 Z 轴 目标速度		数据 校验位	帧尾 (固定值 0X7B)
数据类型	Uint8	Uint8	Uint8	short		short		short		Uint8	Uint8
占用字节	1	1	1	2		2		1		1	1
数组号	1	2	3	4	5	6	7	8	9	10	11

注意：差速车、阿克曼小车不支持 Y 轴移动控制，麦轮小车和全向轮小车才支持 Y 轴移动控制。

在 7 种控制机器人方式中，ROS 的控制优先级是最高的，在任意时候 STM32 控制器的串口 3 接收到数据，则强制进入 ROS 模式。设置前 10 秒不接收数据是因为消除机器人上电开机过程中发送过来的无用数据。10 秒的等待期后开始接收数据，首先要检测数据帧头，检测到数据帧头后才开始接收数据；在数据接收完成后，再验证帧尾的数据校验位是否出错，数据校验位正确才使用数据。串口中断接收数据详细见 usartx.c 文件的 USART3\_IRQHandler() 串口中断函数。

## 1.3 APP 控制

机器人支持 APP 蓝牙控制以及在线调参。在 APP 模式中，直接使用摇杆控制机器人在空间中的运动。APP 控制机器人运动的速度单位是 mm/s(毫米/秒)，摇杆的斜上方的加速和减速按键，每按下一次机器人的速度增加/降低 100 (mm/s)。

APP 在控制机器人的同时，机器人会通过蓝牙(APP 同时支持 WIFI 和蓝牙通信)发送数据给手机，在 APP 的 Debug 栏里面可以看到显示发送的数据，具体的发送内容可以查看代码的 show.c 文件中的 APP\_Show() 函数。APP 控制模式其原理是通过蓝牙(或 WIFI)实现串口通信控制，该机器人的 APP 控制用的是串口 2，波特率设置为 9600，其控制指令在串口 2 接收中断服务函数中。



### ① 在线调参

将最新版的 MiniBalance APP 安装到安卓手机上，然后根据相应的视频教程操作即可遥控机器人或者进行参数在线调节等操作。在“调试”界面中每个通道的名称可以点击“参数 x”可以自定义修改名称，具体效果见图 1-3-1 和图 1-3-2。另外，在调节 PID 参数之前，我们需要点击[获取设备参数](右上角菜单按钮调出)，把机器人的 PID 参数更新到 APP 上面，然后拖动滑块，当我们松手的时候，APP 即可把参数发送到机器人上面。

参数 0 为小车的速度参数，调整参数 0 可以调节小车的速度。

参数 1、2 是小车运动控制的 PI 参数。

需要注意的是，假设当前机器人控制速度是 40，当我们想把速度参数调到 150（超过范围）时，需要在第一次调到 80 后，再点击一次[获取设备参数]，这时调整范围就改变成 0-160 了。机器人设定了速度范围是 20—200。关于机器人控制速度单位的详细解释，请参考下文的“第三章的机器人运动速度单位部分”。



图 1-3-1 默认调参界面



图 1-3-2 获取设备参数后的界面

（需要自行修改参数名）

② APP 控制机器人



图 1-3-3 APP 控制界面

APP 操作界面对应的每一个操作是实际上向机器人发送不同的指令信息（切换控制方式及界面也同样发送指令），机器人接收到指令信息后作出响应，详细的 APP 操作界面的每个操作对应发送的信息见表 1-4：

表 1-4 APP 界面操作指令说明

APP 摇杆	↑	↗	→	↘	↓	↙	←	↖
机器人接收到的数据	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48
机器人实现效果	前进运动	右前运动	原地右转	右后运动	后退运动	左后运动	原地左转	左前运动
按键	重力		摇杆		按键		减速	加速
机器人接收到的数据	0x49		0X4a		0x4b		0x59	0x58

注意：手机成功连接上小车蓝牙后，需要往前推摇杆 0.5 秒后才能正式控制小车。

## 1. 4PS2 控制

PS2 模式使用的是 PS2 无线手柄，PS2 模式控制的顺序是：上电前先插好手柄，然后接通电源，此时手柄上的指示灯红灯常亮表明正常工作，若指示灯不亮则按一下指示灯上方的按键，按下 START 按键后，进入手柄模式，此时可以看到显示屏的左下角显示“PS2”。

在 PS2 模式中，使用左边遥控控制机器人前后移动，右边的摇杆控制机器人的左右转向(全向运动小车则为左边遥控控制小车在空间中 8 个方位的运动，右边的摇杆控制小车原地自旋转运动)。这样设计的原因是如果前后和左后的控制都使用同一个摇杆容易出现误触转向的情况。左上角两个按键为加速、减速按键。

注意：在 PS2 模式下，要插好 PS2 手柄后再上电，否则容易烧坏手柄以及出现电机乱转的现象。PS2 手柄模式在开机后,需要按下 PS2 手柄上面的 START 按键后才进入 PS2 手柄模式。



图 1-4-1 PS2 手柄实物图

## 1.5 航模遥控

需要注意的是，阿克曼小车不支持航模遥控控制。

航模遥控的正确使用步骤是：小车先接上航模遥控接收器，打开航模遥控的电源，小车上电，航模遥控接收器指示灯常亮说明已经连接上，这时可以看到显示屏左下角显示“R-C”。航模遥控的控制方式是：航模遥控的左边摇杆控制小车前进后退，右边摇杆控制小车左右转向(全向运动小车则为左边遥控控制小车在空间中 8 个方位的运动，右边的摇杆控制小车原地自旋转运动)，同时右边摇杆前后拨动可以控制小车车速即相当于油门(油门功能不支持阿克曼小车)。右上方的 SWC 开关控制小车选择正常模式和低速模式。航模遥控关机之前需要先关闭小车，再关闭遥控器。



图 1-5-1 航模遥控实物图

需要注意的是航模遥控的通道需要按图 1-5-2 所示配置，其中最左边的按键打到中间，其余的四个按键打到下面。这些通道是调节控制的方向的。





图 1-5-2 航模遥控实物图

下面讲解一下航模遥控接收器该如何连接转接板。



图 1-5-3 航模接收器实物图

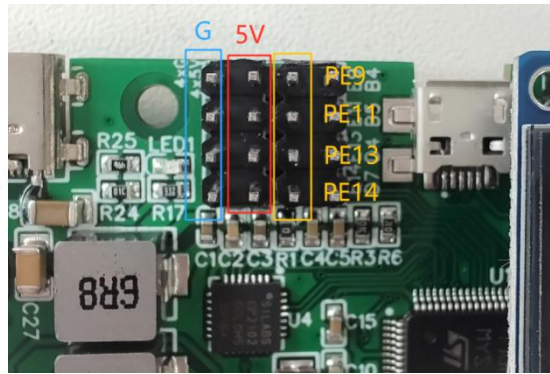


图 1-5-4 转接板上的航模接口

表 1-5 航模遥控引脚对应通道

航模接收器	GND	5V	CH1	CH2	CH3	CH4
转接板	G	5V	PE9	PE11	PE13	PE14

如图 1-5-3、图 1-5-4、表 1-5 所示，航模接收器有三列，分别是 GND、5V 和信号线，我们使用的时候 GND 和 5V 接一对即可，然后信号线的 CH1、CH2、CH3、CH4 分别接转接板上的排针 PE9、PE11、PE13、PE14。PE14 引脚只有全向移动小车才需要用到，用于左右横移。



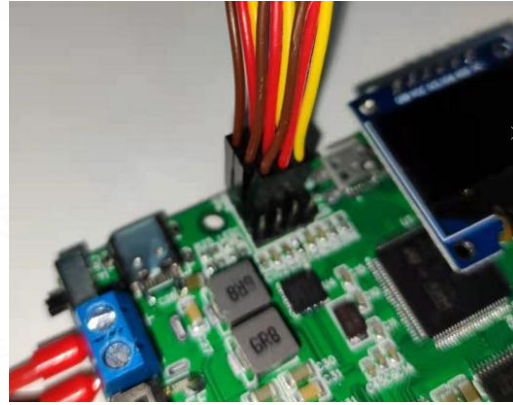


图 1-5-5 航模接线示例

## 1.6 CAN 控制

小车支持 CAN 通信，CAN 通信的接线方式是 CANL 对 CANL, CANH 对 CANH。如果要让小车接收 CAN 控制指令，需要先向 CAN 接口发送使能指令，波特率是 1M。

使能指令格式如下：

标识符 ID: 0X121

帧类型：标准帧

帧格式：DATA

DLC: 8 个字节

表 1-6 CAN 使能指令

数据域	tx[0]	tx[1]	tx[2]	tx[3]	tx[4]	tx[5]	tx[6]	tx[7]
内容	10	12	15	19	24	30	37	Flag

**注意：**该表格中的数据是 10 进制形式

Flag=1 时，CAN 控制使能，之后控制器不再接收其他控制模式的指令。CAN 控制使能后可以看到显示屏下角显示“CAN”，接着我们可以发送 CAN 指令对小车进行控制。控制指令的说明如下：

标识符 ID: 0X121

帧类型：标准帧

帧格式：DATA

DLC:8 个字节

表 1-7 小车接收 CAN 控制指令

数据域	rx[0]	rx[1]	rx[2]	rx[3]	rx[4]	rx[5]	rx[6]	rx[7]
内容	小车	小车	小车	小车	小车	小车		
	X 方向	X 方向	Y 方向	Y 方向	Z 方向	Z 方向	预留位	预留位
	控制量	控制量	控制量	控制量	控制量	控制量		
	高 8 位	低 8 位	高 8 位	低 8 位	高 8 位	低 8 位		

rx[6]、rx[7]是预留的数据位，可以供我们添加自己需要传输的数据，CAN 通信自带 BBC 校验所以这里不需要数据校验位。

注意：差速车、阿克曼小车不支持 Y 轴移动控制，因此 Y 轴方向的控制量默认是 0。

小车在接收 CAN 指令控制的同时也可以发送小车自身的数据。执行 CAN 发送数据的函数为 CAN\_SEND() 位于文件【usartx.c】下，默认已经开启此发送数据的任务并且设定好了要发送的数据，如果需要自定义发送别的内容，只需要替换 CAN\_SEND 函数中的 Send\_Data.buffer[i]的内容即可。

由于要发送的数据较多共 24 字节，发送数据时是一次发送 8 个字节，分 3 次发送，接收小车发送数据的单片机可以通过标识符 ID 确认当前接收到的是第几组的数据，发送第一组数据的标识符是 0X101，发送第二组数据的标识符是 0X102，发送第三组数据的标识符是 0X103。关于小车 CAN 发送的详细内容下表所示，CAN 发送数据的内容与串口 1 发送内容、串口 3 (与 ROS 通信的接口)发送内容是一样的，该部分内容可以查看 1.2 节 ROS(串口 3)控制。

我们配套的 CAN 发送例程中配备了 CAN 接收的中断服务函数用于接收小车发送出来的数据。例程代码适配的单片机型号是 STM32F103RC 或同系列的单片机，同时需要 VP230 芯片对单片机的数据进行转换之后才能正式与小车进行 CAN 通信，其接线如下图所示，单片机的 PD1、PD0 引脚接 VP230 的 D、R 引脚，VP230 的 CANH、CANL 引脚接小车上的 CANH、CANL 引脚(即 H 对 H，L 对 L)。

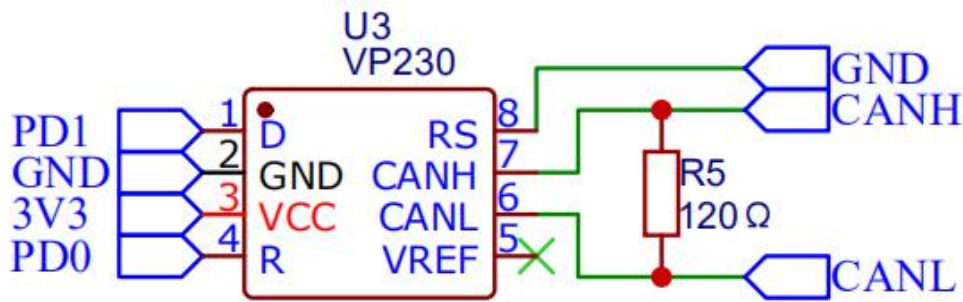


图 1-6-1 单片机与小车之间通过 VP230 转换

如果您想查看 CAN 通信接收到的数据是否正确，可以用上位机连接单片机的串口 1。单片机接收到小车 CAN 通信发过来的信息后，会再通过单片机的串口 1 发送出来，波特率为 9600。

注意：资料提供的 CAN 和串口的例程不是小车上运行的代码，不是说您需要串口通信就需要下载串口例程，小车默认已经有程序（固件）了，不需要下载任何程序即可操作。资料提供的 CAN 和串口的例程是可以运行在其他的单片机上，用于和小车进行通信的程序。

## 1.7 串口 1 控制

STM32 控制器板子上引出了串口 1 (CP210) 和串口 3 (CP210) 两路串口通信接口，两路串口发送（发送的内容也一致）和接收数据的处理程序完全一致，默认使用串口 3 与 ROS 端进行串口通信。关于串口 1 的使用可以查看本文前面的 1.2 节 ROS(串口 3) 控制。

串口传输时需要注意通信双方波特率的设置需要一致，程序中串口 1 设置的波特率是 115200。接收到串口 1 的数据后小车进入串口控制模式，显示屏左下角显示“USART”。

表 1-8 小车接收串口控制指令

数据域	帧头	tx[0]	tx[1]	tx[2]	tx[3]	tx[4]	tx[5]	tx[6]	tx[7]	tx[8]	帧尾
内容	0X7B	预留位	预留位	小车 X 方向控 制量高 8 位	小车 X 方向控 制量低 8 位	小车 Y 方向控 制量高 8 位	小车 Y 方向控 制量低 8 位	小车 Z 方向控 制量高 8 位	小 车 Z 方 向 控 制 量 低 8 位	BBC 校验位	0X7D



注意：差速车、阿克曼小车、履带车、四驱车不支持 Y 轴移动控制，麦轮小车和全向轮小车才支持 Y 轴移动控制。将我们提供的串口发送指令例程下载到另一个单片机中，接通单片机和小车控制器之间的串口通信线，小车进入串口控制模式。

```

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
    
```

图 1-7-1 串口发送指令例程向小车发送的控制指令

同样的，可以通过上位机串口助手直接给小车的串口 1 发送数据，如图 1-7-2 所示。

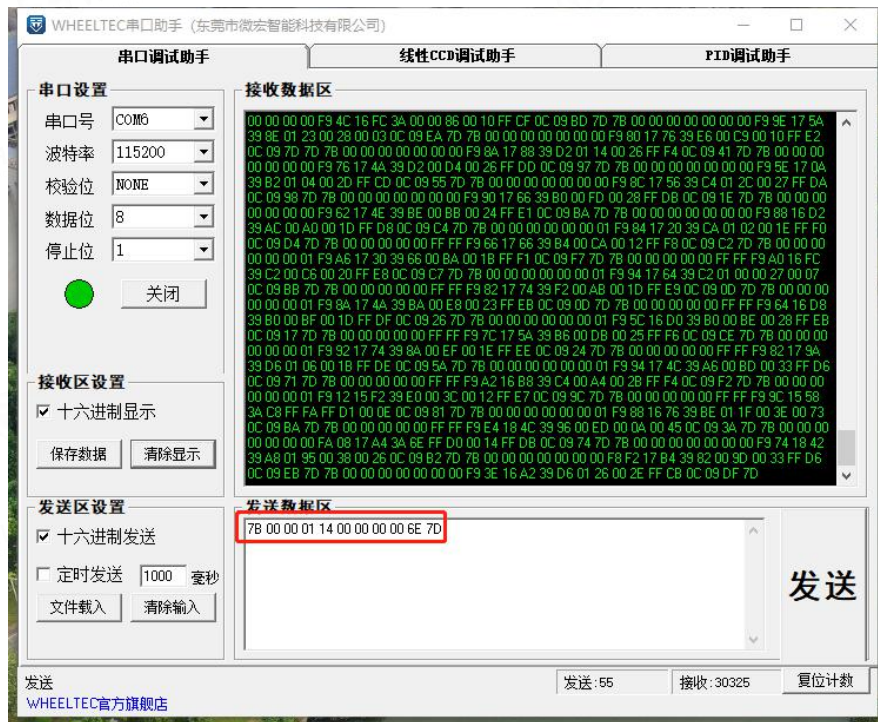


图 1-7-2 串口助手向小车发送的控制指令

串口 1 和串口 3（ROS）控制接收控制的指令相同，关于小车速度控制量和小车发送数据内容的计算转换可以查看我们的 1.2 节 ROS(串口 3)控制。

## 2. OLED 显示内容

### 2.1 OLED 具体内容

机器人配备了 OLED 显示屏，不同型号的机器人显示屏显示的内容大同小异，具体如下所示：

#### ① 履带车

Tank	BIAS	Z +	5	第一行：车型和Z轴角速度零点漂移值
GYRO	Z	+	2	第二行：去除零点漂移值的Z轴角速度
L:+	0	+	0	第三行：左电机目标值和测量值
R:+	0	+	0	第四行：右电机目标值和测量值
MA +	0	MB+	0	第五行：电机的PWM数值
ROS	ON	12.03V		第六行：控制模式 使能开关 电池电压

图 2-1-1 履带车 OLED 显示屏内容

#### ② 阿克曼小车

Akm	BIAS	Z +	5	第一行：车型和Z轴角速度零点漂移值
GYRO	Z	+	2	第二行：去除零点漂移值的Z轴角速度
L:+	0	+	0	第三行：左电机目标值和测量值
R:+	0	+	0	第四行：右电机目标值和测量值
Servo:		+	1500	第五行：舵机值
ROS	ON	12.03V		第六行：控制模式 使能开关 电池电压

图 2-1-2 阿克曼小车 OLED 显示屏内容

#### ③ 两轮差速小车

Diff	BIAS	Z +	5	第一行：车型和Z轴角速度零点漂移值
GYRO	Z	+	2	第二行：去除零点漂移值的Z轴角速度
L:+	0	+	0	第三行：左电机目标值和测量值
R:+	0	+	0	第四行：右电机目标值和测量值
MA +	0	MB+	0	第五行：电机的PWM数值
ROS	ON	12.03V		第六行：控制模式 使能开关 电池电压

图 2-1-3 两轮差速小车 OLED 显示屏内容



#### ④ 全向轮小车

Omni	GZ	+	5	第一行：车型和去除零点的Z轴角速度
A: +	0	+	0	第二行：A电机目标值和测量值
B: +	0	+	0	第三行：B电机目标值和测量值
C: +	0	+	0	第四行：C电机目标值和测量值
MOVE	Z	+	0	第五行：由编码器计算的实时角速度
ROS	ON		12.03V	第六行：控制模式 使能开关 电池电压

图 2-1-4 全向轮小车 OLED 显示屏内容

#### ⑤ 麦轮小车

Mec	GZ	+	9	第一行：车型和去除零点的Z轴角速度
A:+	0	+	0	第二行：A电机目标值和测量值
B:+	0	+	0	第三行：B电机目标值和测量值
C:+	0	+	0	第四行：C电机目标值和测量值
D:+	0	+	0	第五行：D电机目标值和测量值
ROS	ON		12.03V	第六行：控制模式 使能开关 电池电压

图 2-1-5 麦轮小车 OLED 显示屏内容

#### ⑥ 四驱车

4WD	GZ	+	9	第一行：车型和去除零点的Z轴角速度
A:+	0	+	0	第二行：A电机目标值和测量值
B:+	0	+	0	第三行：B电机目标值和测量值
C:+	0	+	0	第四行：C电机目标值和测量值
D:+	0	+	0	第五行：D电机目标值和测量值
ROS	ON		12.03V	第六行：控制模式 使能开关 电池电压

图 2-1-6 四驱车 OLED 显示屏内容

## 2. 2OLED 通用显示内容

### ① 控制模式

其中不同的控制模式对应左下角不同显示内容，具体见表 2-1。

表 2-1 机器人 OLED 显示屏显示控制模式

模式	CAN	手机 APP	PS2 手柄	航模遥控	串口 1	串口 3
显示内容	CAN	APP	PS2	R-C	UART1	UART3

### ② 使能开关

使能开关在机器人 STM32 控制器的左上角，使能开关在“ON”状态下机器人的电机才使能控制。程序里面考虑到机器人的稳定性，STM32 控制器上电之后 10s 内电机是强制失能状态。这个时候即使您的使能开关是在“ON”状态，也是显示“OFF”，10s 后就会更新成“ON”，需要注意的是在程序中设置了这 10s 内串口 1 和串口 3 不接收数据。

## 2. 3 小车自检

小车配备了自检代码，详细的内容见文档“型号、接线、编码器检查教程.doc”。

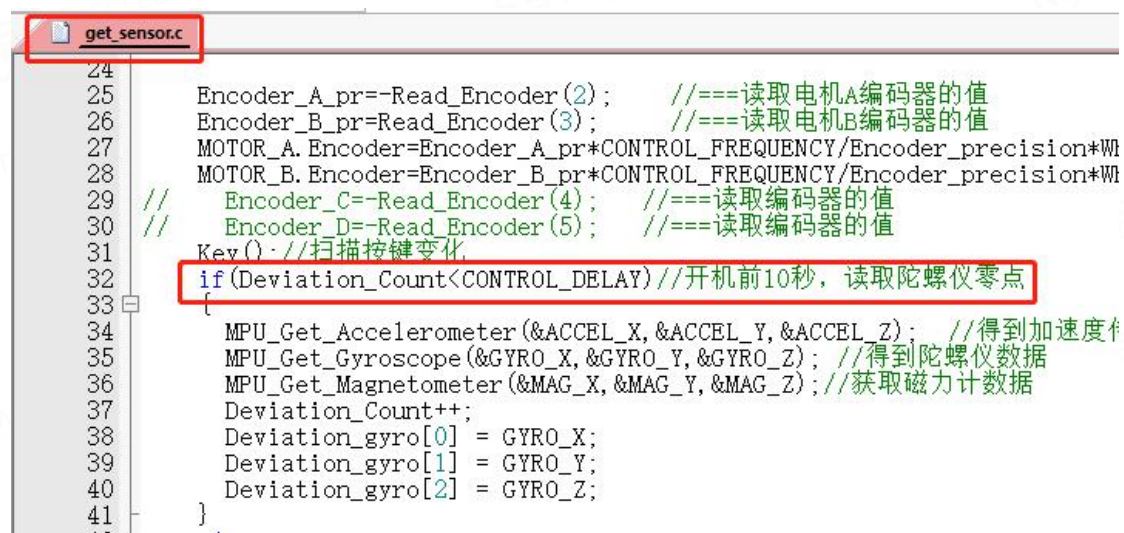
### 3. 陀螺仪零点漂移消除

在 ROS 的导航系统中需要用到 IMU 传感器，在我们的 ROS 机器人系统中，IMU 传感器集成到 STM32 控制器上，由 STM32 控制器采集 IMU 数据后，再发送给 ROS。STM32 运动底盘上使用的 IMU 是 MPU9250，该 IMU 集成了三轴角速度计、三轴加速度计、三轴磁力计，在这里我们只需要用到角速度计和加速度计。陀螺仪不可避免的存在零点漂移的问题，因此在程序中设定了零点漂移消除机制。

在上电的前 10 秒时间内，陀螺仪读取到的是未去除零点漂移值的角速度值，在 10 秒时，程序读取当前的角速度值作为漂移值。10 秒后读取的陀螺仪数据会减去零点偏移量，此时 LED 灯从常亮变成闪烁，后续的读取过程中读取到的角速度值会减去零点漂移值，得到的就是消除零点漂移的角速度。

如果您觉得 10 秒获取的陀螺仪漂移值还不够准确，可以在任意时间双击用户按键（STM32 控制器左下角），重新获取陀螺仪的漂移值。

以上描述的采集陀螺仪数据的过程在 STM32 控制器代码的 get\_sensor.c 文件中，详细见图 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); //得到加速度值
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 }
  
```

图 3-1 采集陀螺仪数据

## 4. 机器人运动学分析

想要让机器人运动，除了提供目标速度还不够，需要将机器人的目标速度转换每个电机实际的目标速度，最终根据电机的目标速度对电机的控制实现对机器人的控制。机器人的目标速度转换成电机的目标速度这个过程叫“运动学分析”，运动学分析又分为正解和逆解，在运动学分析之前先来分别解释一下什么是运动学正解和运动学逆解：

①运动学正解：通过机器人的各轮速度求出机器人 X 轴、Y 轴和 Z 轴方向的速度。

②运动学逆解：通过机器人 X 轴、Y 轴和 Z 轴方向的速度分别求出机器人各轮的速度。

### 4.1 两轮差速(履带车)小车

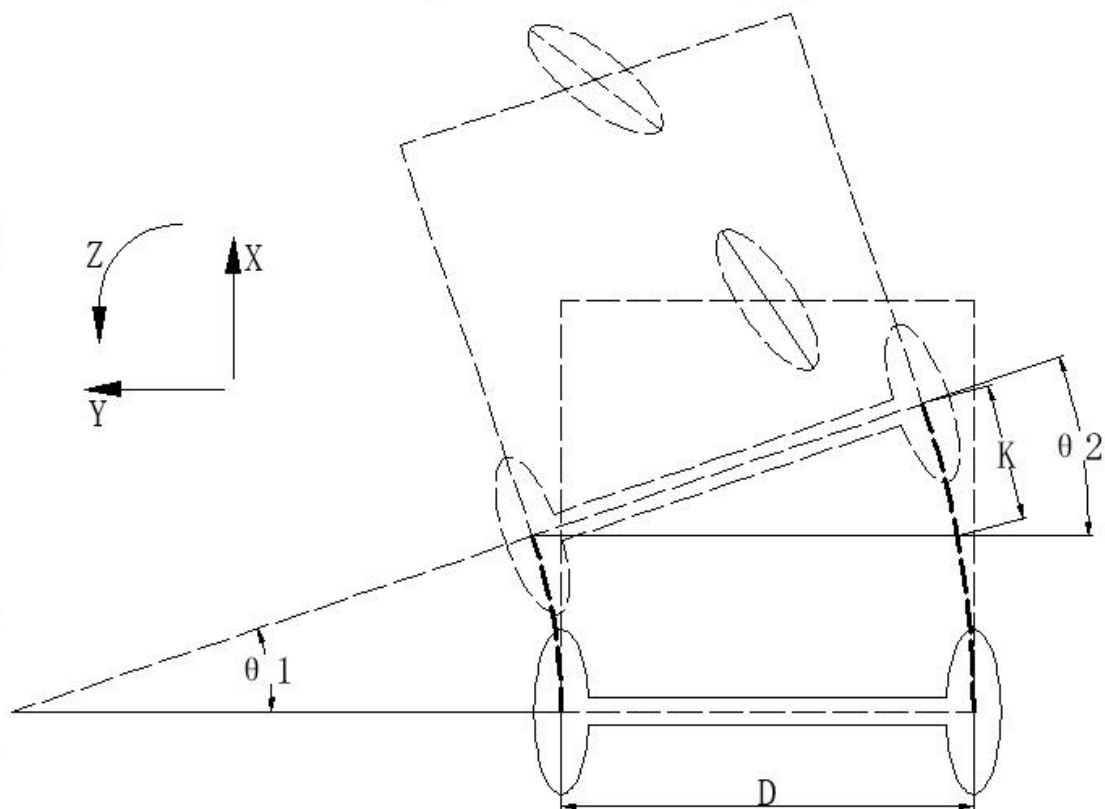


图 4-1 两轮差速小车运动学模型

#### ① 运动学分析

机器人的运动简化模型如图 4-1 所示，X 轴正方向为前进、Y 轴正方向为左



平移、Z 轴正方向为逆时针(后面产品同样，将不再赘述)。机器人两个轮子之间的间距为 D，机器人 X 轴和 Z 轴的速度分别为： $V_x$  和  $V_z$ ，机器人左轮和右轮的速度分别为： $V_L$  和  $V_R$ 。

假设机器人往一个左前的方向行进了一段距离，设机器人的右轮比左轮多走的距离近似为 K，以机器人的轮子上的点作为参考点做延长参考线，可得： $\theta_1 = \theta_2$ 。

由于这个  $\Delta t$  很小 (10ms)，因此角度的变化量  $\theta_1$  也很小，因此有近似公式：

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

由数学分析可以得到下面的式子：

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

由上面的公式和式子可以求解出运动学正解的结果：

机器人 X 轴方向速度  $V_x = \frac{V_L + V_R}{2}$ ，机器人 Z 轴方向速度  $V_z = \frac{V_R - V_L}{D}$ 。

由正解直接反推得出运动学逆解的结果：

机器人左轮的速度  $V_L = V_x - \frac{V_z * D}{2}$ ，机器人右轮的速度  $V_R = V_x + \frac{V_z * D}{2}$ 。

## ② C 语言实现

机器人上面有 2 个带编码器的电机，我们需要通过 C 语言写出上面的运动关系，然后对电机进行控制。代码如下：

```
void Drive_Motor(float vx, float vz)
{
    Target_Left = vx - vz * WIDTH_OF_ROBOT / 2.0f; //计算出左轮的目标速度
    Target_Right = vx + vz * WIDTH_OF_ROBOT / 2.0f; //计算出右轮的目标速度
}
```

以上语句是通过机器人的 X 和 Z 轴方向的速度求两个电机的目标速度大小 (运动学逆解)，其中 WIDTH\_OF\_ROBOT 是两个车轮直线距离的宏定义。



## 4.2 阿克曼小车

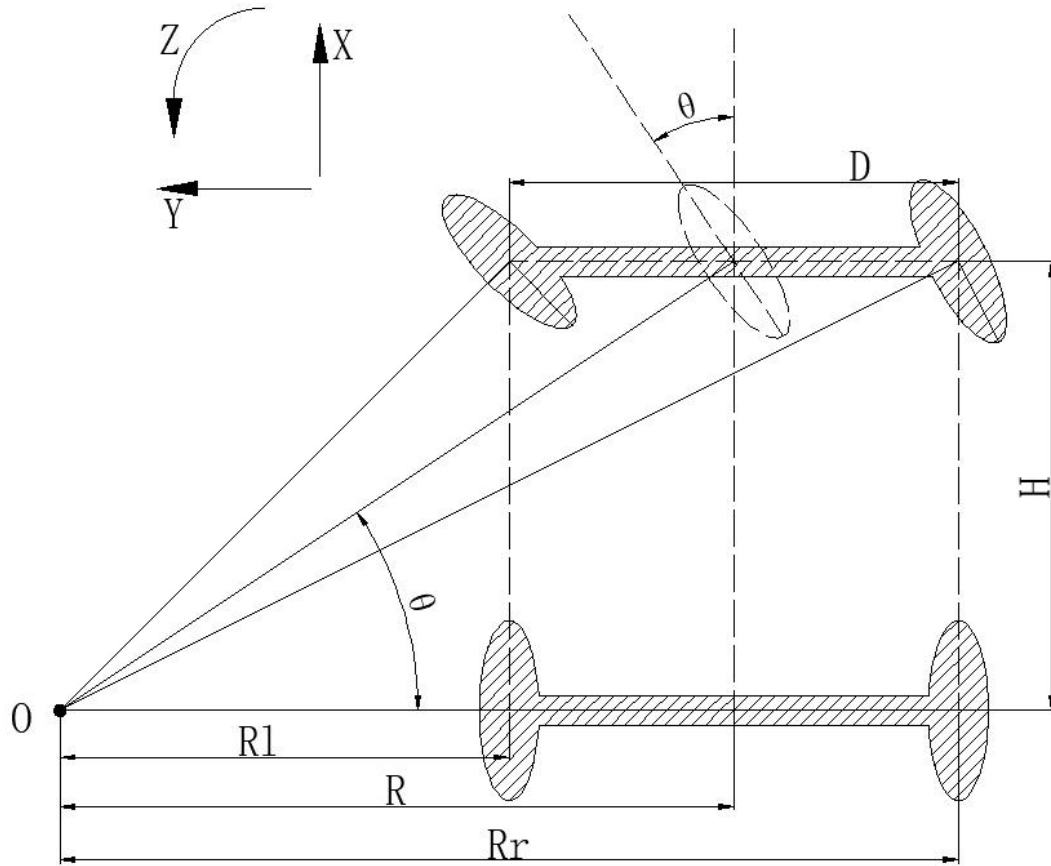


图 4-2 阿克曼小车运动学模型

### ① 运动学分析

阿克曼小车的与两轮差速小车的区别在于,两轮差速小车的前轮为全向轮或者万向轮,而阿克曼小车的前轮则是普通的“单向轮”。而此时要使阿克曼小车实现纯滚动运动,则必须保证小车的四个轮运动方向的法线相交于一点,该点则为转向中心点,如图 4-2 点 O。

为简化模型,设前轮只有一个轮子(实现理论是一致的),位于前轮轴中间位置上,如图 4-2 虚线部分描绘的前轮所示。

由阿克曼小车运动学模型可知,当前轮偏角为  $\theta$  时,小车转向半径为  $R$ 。设小车前进速度为  $V$ (即  $V_x$ ),左轮速度为  $V_L$ ,右轮速度为  $V_R$ ,由角速度的一致性

可得:  $\frac{V}{R} = \frac{V_L}{R_L} = \frac{V_R}{R_R}$ , 又有  $\frac{H}{R} = \tan\theta$ ,  $R_L = R - \frac{D}{2}$ ,  $R_R = R + \frac{D}{2}$ 。

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

## ② C 语言实现

```
void Kinematic_Analysis(float Vx, float Vy, float Vz)
{
    float angle=0;
    MOTOR_A.Target = Vx*(1-(WheelSpacing/(2*AxleSpacing))*tan(Vz*PI/180)); //轮
    上速度要根据前轮实际转角 Vz 来计算
    MOTOR_B.Target = Vx*(1+(WheelSpacing/(2*AxleSpacing))*tan(Vz*PI/180)); //轮
    上速度要根据前轮实际转角 Vz 来计算
    if(Vz>0) angle=Vz*PI/180*Servo_Wheel_ratio_L; //舵机左右两边对称化处理
    else     angle=Vz*PI/180*Servo_Wheel_ratio_R; //舵机左右两边对称化处理
    if(1)    Servo = -angle*K; //这里是舵机转动角度
}
```

函数输入为 X 和 Y 轴速度和前轮转向角度，K 是舵机控制的修正系数，Axle\_Spacing 为小车(前后)轴距参数，Wheel\_spacing 为小车(左右)轮距参数。

### 4.3 麦轮小车

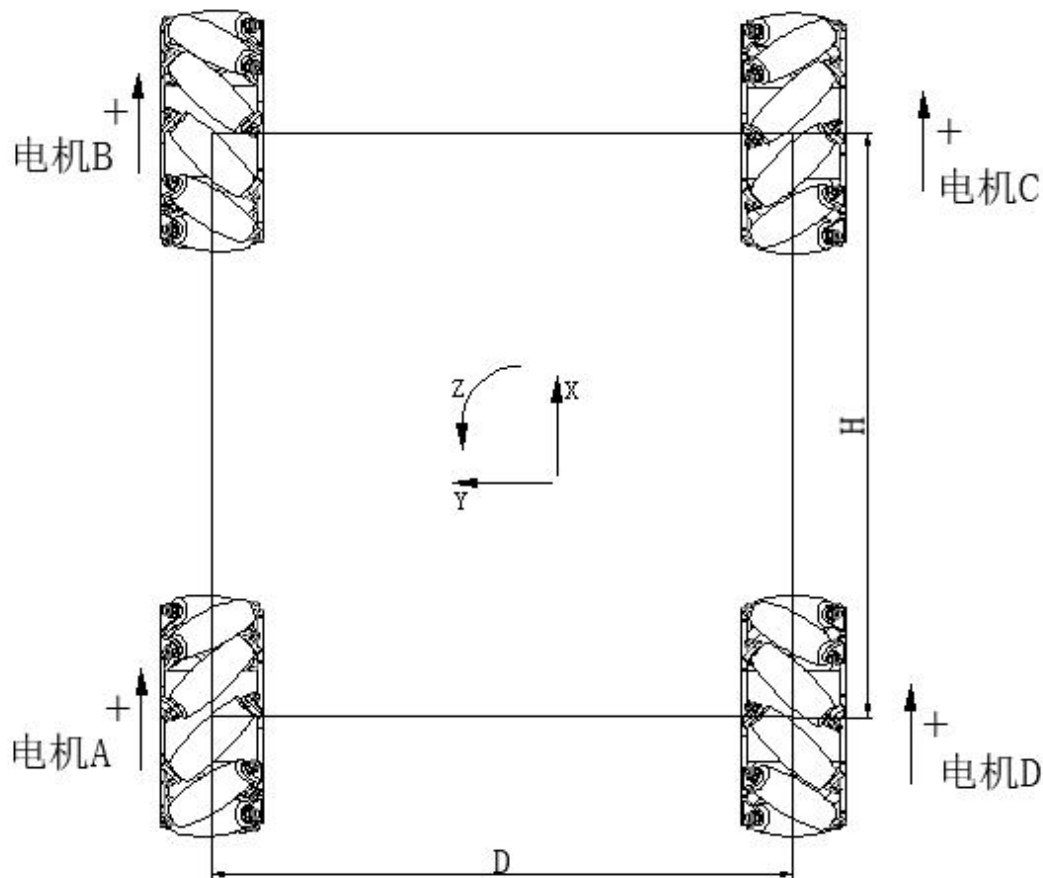


图 4-3 麦轮小车运动学模型

#### ① 运动学分析

为简化运动学数学模型，做下列两点理想化假设：

- (1) 全向轮不与地面打滑，同时地面有足够摩擦力；
- (2) 4 个轮子分布在长方形或者正方形的 4 个角上，轮子之间相互平行。

在这里我们将小车的刚体运动线性分解为三个分量，那么只需计算输出麦轮底盘在沿 X+,Y+方向平移，Z+方向转动时四个轮子的速度，就可以通过公式的合并，计算出这三种简单运动所合成的“平动+旋转”运动时所需要的四个轮子的转速。

其中  $V_A$ 、 $V_B$ 、 $V_C$ 、 $V_D$  分别为 A、B、C、D 四个轮子的转速，也就是电机的转速； $V_x$  为小车沿着 X 轴平移速度， $V_y$  为小车沿着 Y 轴平移速度， $\omega$  为小车沿 Z 轴的旋转速度； $a = \frac{D}{2}$  为小车轮距 D 的一半， $b = \frac{H}{2}$  为小车轴距 H 的一半。

当小车沿着 X 轴平移时:

$$V_A = +V_x$$

$$V_B = +V_x$$

$$V_C = +V_x$$

$$V_D = +V_x$$

当小车沿着 Y 轴平移时:

$$V_A = +V_y$$

$$V_B = -V_y$$

$$V_C = +V_y$$

$$V_D = -V_y$$

当小车绕几何中心自转时:

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

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

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

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

将以上三组方程组合并, 即可根据小车的运动状态解算出四个轮子的转速:

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

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

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

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

## ② C 语言实现

```
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 为小车(前后)轴距参数, Wheel\_spacing 为小车(左右)轮距参数。



## 4.4 全向轮小车

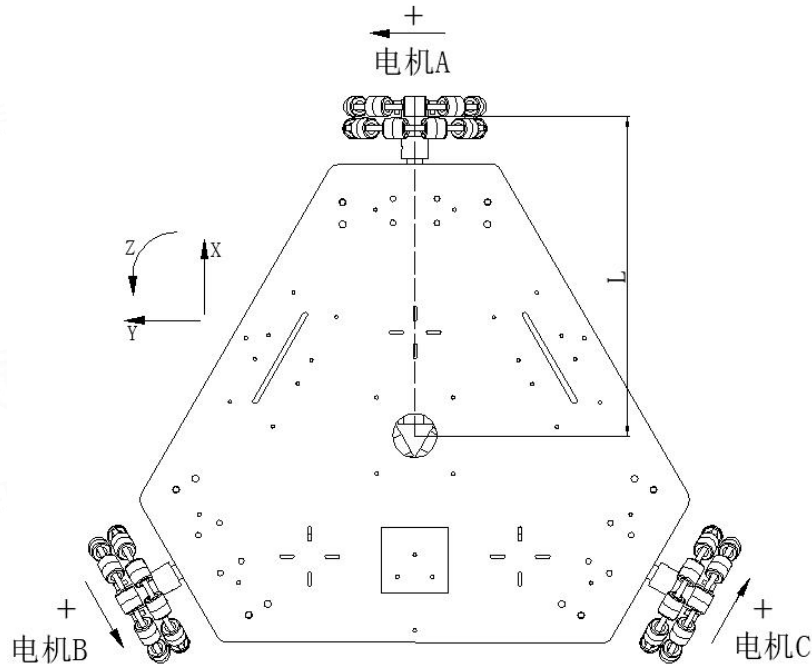


图 4-4 全向轮小车运动学模型

### ① 运动学分析

在运动建模之前，为简化运动学数学模型，做下列几种理想化假设：

- (1) 全向轮不与地面打滑，同时地面有足够摩擦力；
- (2) 电机轴线中心正是底盘重心；
- (3) 各轮之间是绝对的互成  $120^\circ$  安装。

通过简单的速度分解，可以得到以下公式：

$$V_A = V_Y + L_\omega$$

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

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

$$\text{即 } V_A = V_Y + L_\omega,$$

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

$$V_C = +\frac{\sqrt{3}}{2} V_x - \frac{1}{2} V_Y + L_\omega$$

$\omega$  为机器人角速度， $L$  为全向轮中心与底盘中心的距离， $V_A$ 、 $V_B$ 、 $V_C$  分别为 3 个轮子的转速， $V_x$ 、 $V_y$  为机器人的 X、Y 方向的运动速度。

## ② C 语言实现

```
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;
}
```

$\text{Parament.X} = \frac{\sqrt{3}}{2}$ 、 $\text{Parament.Y} = \frac{1}{2}$ 、 $\text{Parament.Z} = 1$ ，对应我们的运动学分析参数。

## 4.5 四驱车

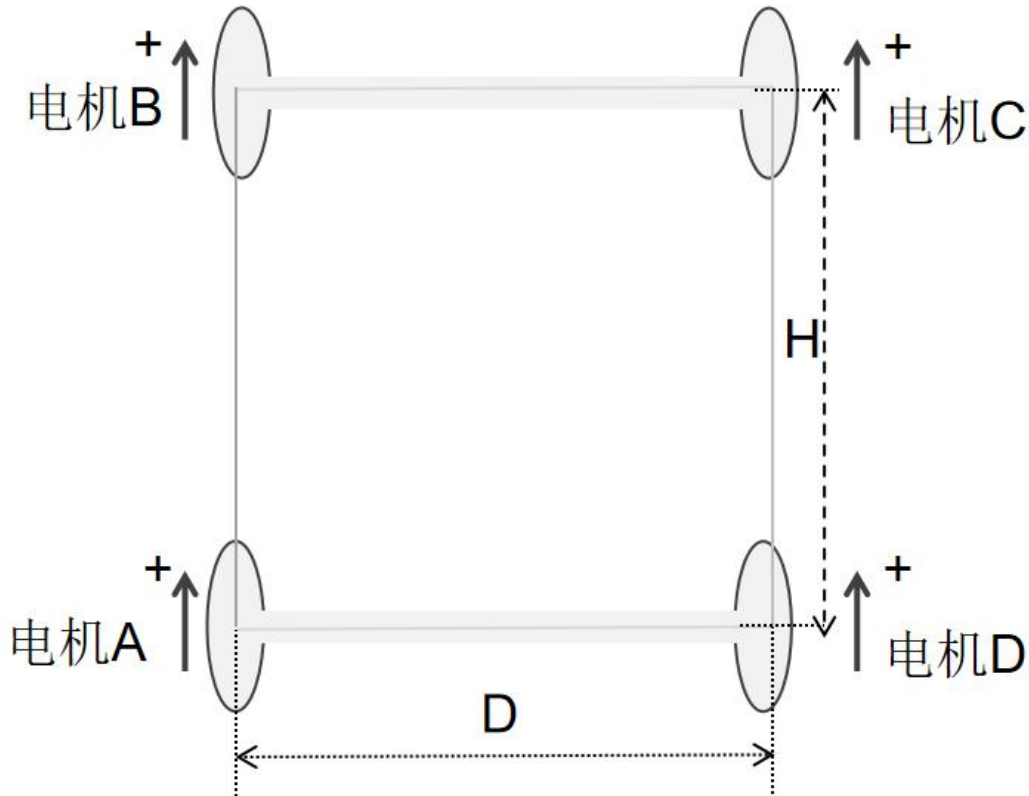


图 4-5 四驱车运动学模型

### ① 运动学分析

为简化运动学数学模型，做下列两点理想化假设：

- (1) 轮子不与地面打滑，同时地面有足够摩擦力；
- (2) 4 个轮子分布在长方形或者正方形的 4 个角上，轮子之间相互平行。

四驱车使用的是橡胶轮。在这里我们将小车的刚体运动线性分解为两个分量，那么只需计算输出底盘在沿 X+方向平移，Z+方向转动时四个轮子的速度，就可以通过公式的合并，计算出这三种简单运动所合成的“平动+旋转”运动时所需要的四个轮子的转速。

其中  $V_A$ 、 $V_B$ 、 $V_C$ 、 $V_D$  分别为 A、B、C、D 四个轮子的转速，也就是电机的转速； $V_x$  为小车沿着 X 轴平移速度， $\omega$  为小车沿 Z 轴的旋转速度； $a = \frac{D}{2}$  为小车轮距 D 的一半， $b = \frac{H}{2}$  为小车轴距 H 的一半。

当小车沿着 X 轴平移时：

$$V_A = +V_x$$

$$V_B = +V_x$$

$$V_C = +V_x$$

$$V_D = +V_x$$

当小车绕几何中心自转时：

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

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

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

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

将以上三组方程组合并，即可根据小车的运动状态解算出四个轮子的转速：

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

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

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

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

## ② C 语言实现

```
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 为小车(前后)轴距参数，Wheel\_spacing 为小车(左右)轮距参数。



## 4. 6PI 控制程序源码

通过运动学分析得到的是电机的目标速度，我们需要把这个目标值送入PID控制器进行速度闭环控制，使得电机的实际输出速度趋近于目标值。程序中的PI控制器源码如下所示：

```
/******  
* 函数功能：增量PI 控制器速度控制  
* 入口参数：Encoder：编码器测量值、Target：目标速度  
* 返回值：Pwm：电机 PWM  
* 函数说明：pwm+=Kp[e(k)-e(k-1)]+Ki*e(k) 增量式PI 控制  
*****/  
int Incremental_PI_A (float Encoder,float Target)  
{  
    static float Bias,Pwm,Last_bias;  
    Bias=Target-Encoder;          //计算偏差  
    Pwm+=Velocity_KP*(Bias-Last_bias)+Velocity_KI*Bias;    //增量式PI 控制器  
    if (Pwm>7200) Pwm=7200;  
    if (Pwm<-7200) Pwm=-7200;  
    Last_bias=Bias;                //保存上一次偏差  
    return Pwm;                    //增量输出  
}
```

## 5. 接线说明

这一章主要演示一下几个关键的接线说明，具体的接线请直接看图。STM32 控制器上集成双路 5V 电源：STM32 控制器上带有两路 5V 电源输出；一路 5V 电源给 STM32 控制器以及外设（编码器、蓝牙、手柄等）供电，一路 5V 电源输出给树莓派供电。

### ① 树莓派电源供电

树莓派的 5V 电源电路集成在了 STM32 控制器的转接板上，使用的是可过 3A 以上电流的 TYPE-C 转 TYPE-C 线。



图 5-1-1 树莓派供电接线

### ② 树莓派和 STM32 控制器串口通信

因为树莓派这里是作为上位机和 STM32 控制器进行通信，默认选择的是集成了 CP2102 电平转换芯片的串口 3。

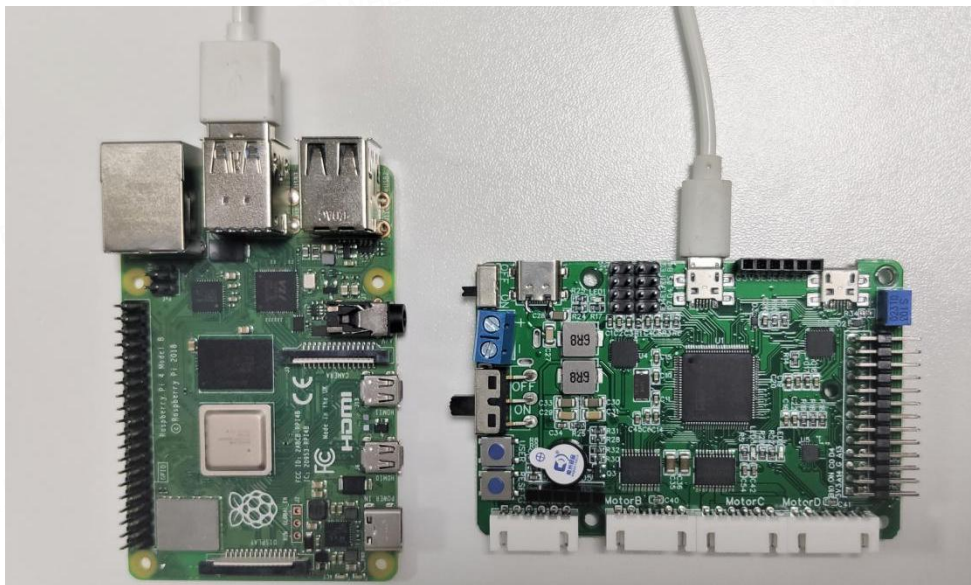


图 5-1-2 树莓派和 STM32 通信

### ③ 树莓派连接导航雷达

树莓派和激光雷达之间的连接这里用的是普通 Mirco-USB 线。树莓派在给激光雷达供电的同时也和激光雷达进行通信。

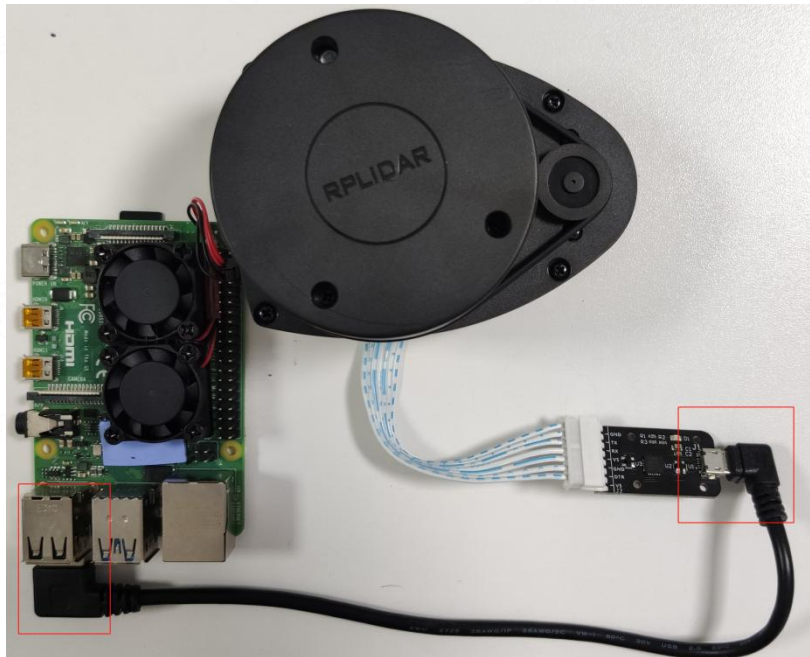


图 5-1-3 树莓派和激光雷达接线

### ④ STM32 控制器的外设详细说明图

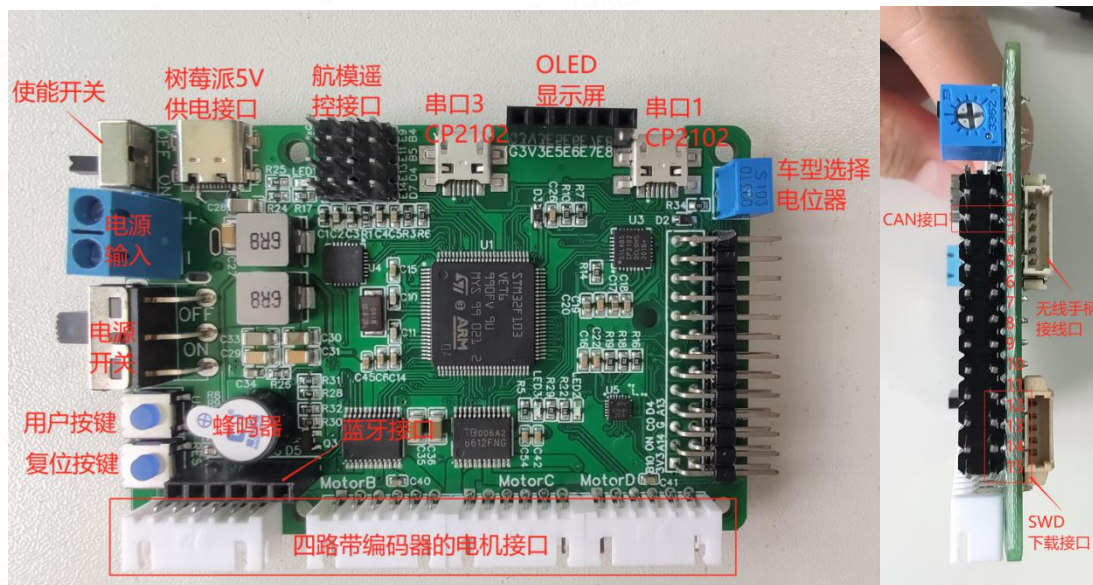


图 5-1-4 STM32 控制器说明图



## 6. 控制流程图

### 6.1 机器人电机的控制流程图

机器人支持 6 种控制模式，这 6 种控制模式实现控制的原理都是通过改变机器人的目标速度来实现控制的。目标速度经过运动学分析函数得出每个电机的实际输出，最后通过 PID 控制器（PID 速度控制函数）来实现电机的速度控制。

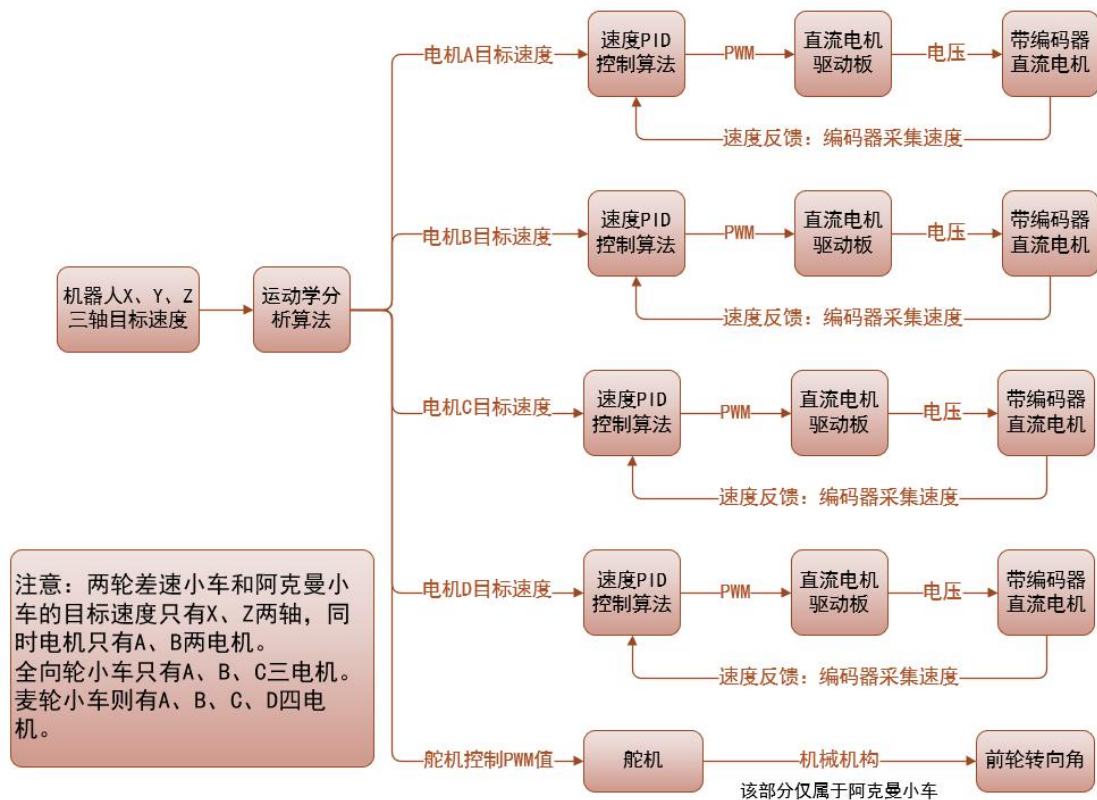


图 6-1 机器人电机控制流程图



## 6.2 机器人 STM32 程序结构图

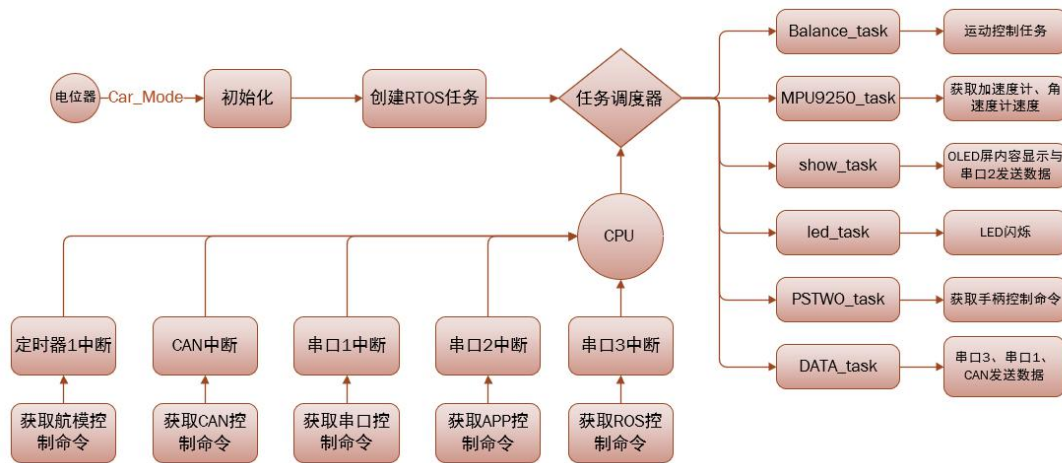


图 6-2 机器人 STM32 控制器程序执行流程图

RTOS 任务调度器根据任务的优先级决定任务的执行顺序（图 6-2 中的任务排序不代表任务优先级，具体任务优先级需要查看程序中优先级的设定），每个任务执行的时间很短，因此几乎等效于所有任务同时执行，期间如果发生中断则去响应中断。串口 2 中断用于 APP 蓝牙控制，串口 3 中断用于接收 ROS 传过来的信息。

### Car\_Mode 影响到的程序部分

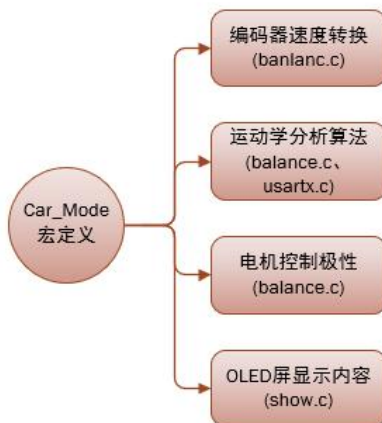


图 6-3 Car\_Mode 影响到的程序部分

## 6.3 机器人控制器连接图

在机器人中用到很多控制器和外设，包括：树莓派（Jetson Nano）、激光雷达，STM32 控制器，电机、编码器、双路驱动、蓝牙、PS2 手柄、航模遥控、陀螺仪等，同时提供了串口 1 和 CAN 接口方便用户拓展控制，这些控制器与控制器，外设与控制器之间的连接，如图 6-3 所示。

小车外设关系图

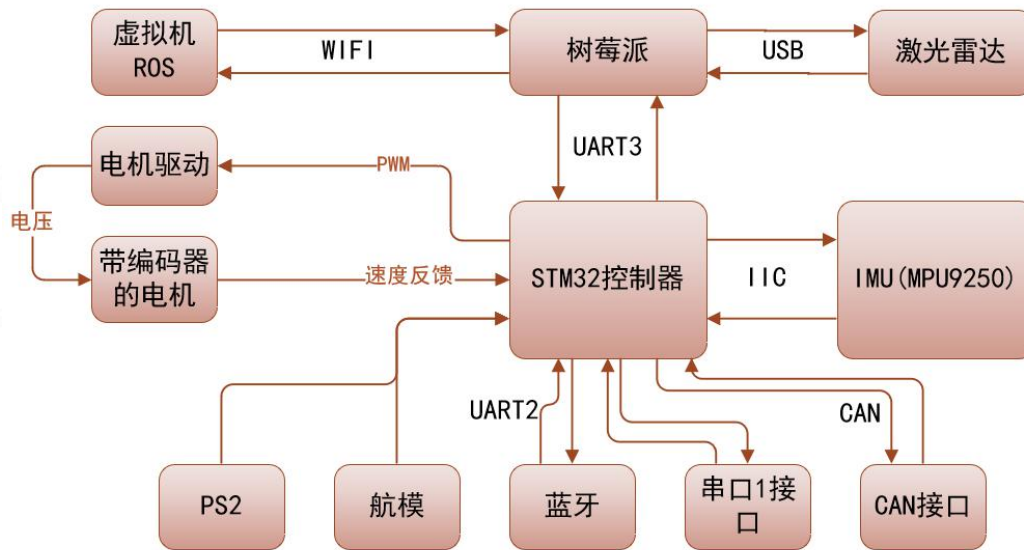


图 6-4 机器人控制器连接图

## 7. 注意事项

### 7.1 关于代码

机器人 STM32 控制器上的程序写法是基于 RTOS 系统，与中断控制的方式不同，RTOS 是任务形式轮流执行，优先级高的任务执行优先级更高（中断优先级高于任务优先级）。需要说明的是，如果某个任务有执行逻辑的错误的话，程序会卡在该错误处无法继续执行下去。例如：假如程序中有一个串口 3 的发送任务，但是串口 3 没有初始化或者是初始化的代码出错，那么在执行串口 3 发送任务时，程序就会卡住。因此如果遇到调试过程中程序卡住无法正常执行，需要逐个任务排查是否有 bug 卡住了。

### 7.2 关于转接板上的电源接口

转接板上面的 5V 和 3.3V 引脚可以对外供电，但是不可带太大电流的负载，其中 5V 输出不建议带超过 1.5A 的负载，3.3V 输出不建议带超过 200mA 的负载。在我们提供的转接板原理图上可以看到，转接板上配备了两路 5V 电源电路，其中一路给基础外设供电，另一路单独给树莓派（Jetson Nano）供电（USB 母座）。关于接线说明在第 5 章“5. 接线说明”有详细的讲解。

### 7.3 关于电机

使用的过程中要避免电机堵转，否则很容易烧毁主板。在机器人没有完全测试通过之前，请把机器人架起来，让电机悬空，避免误操作导致电机乱转引起不必要的损坏。

### 7.4 关于电池

显示屏上显示了电池的电压，电量低时（低于 10.8V）请及时充电。电池自带过放和过充保护。电池充电的时候请不要使用电池，电池充电接线如图 7-4-1 所示。



图 7-4-1 机器人电池与充电器接线图

电池充电结束后，应该将充电口的盖子合上，避免误触导致电池断路，具体如图 7-4-2 所示。



图 7-4-2 机器人电池充电线接口



## 8. 如何给 STM32 控制器下载程序

STM32 控制器可以通过串口或者 SWD 接口下载程序。串口是通过 USB 数据线下载，默认有赠送，SWD 接口建议使用金属外壳的 STLink 下载，需要自备。

### 8.1 串口下载

主板采用了一键下载电路，下载程序非常方便。只需一根 MicroUSB 手机数据线就行了。

#### ① 硬件准备

硬件：

1. STM32 控制器
2. MicroUSB 手机数据线

#### ② 软件准备

软件：mcuisp 烧录软件（附送的资料中有），相应的 USB 转 TTL 模块 CP2102 的驱动。附送的资料里面也有驱动，如果驱动安装实在困难，就下载个驱动精灵吧。

安装成功后可以打开设备管理器看看，可以看到驱动已经安装成功，否则会有红色的感叹号。



图 8-1-1 设备管理器查看 CP210x 驱动

串口下载程序的接线非常简单，数据线连接电脑和板子即可。打开附送资料里面的 mcuisp 软件，根据图 8-1-2 中的操作顺序设置即可。

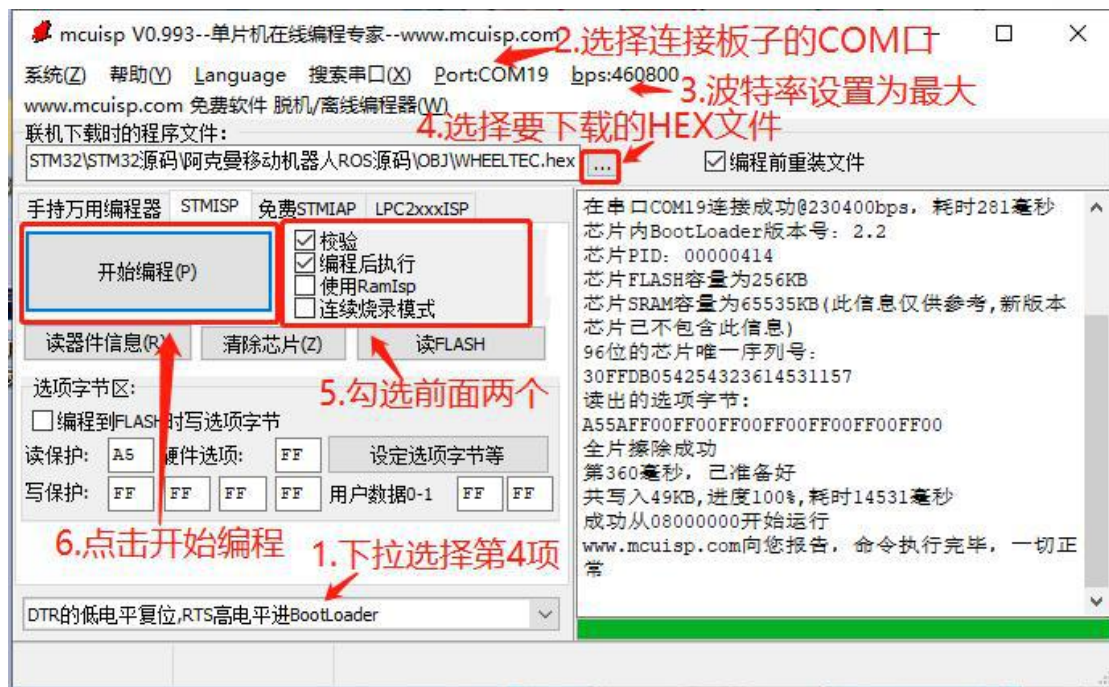


图 8-1-2 FlyMcu 下载程序配置说明

OK, 一切准备就绪, 然后点击开始编程, 程序就可以下载了。因为勾选了编程后执行, 所以程序下载完后, 会自动运行。(需要注意的是: 禁止勾选编程到FLASH时写选项字节, 如果您使用的是F4的板子, 波特率需要设置为76800)

## 8.2 SWD 下载

STM32 控制器可以通过 SWD 接口下载程序, 在主板上上面有标识, 分别是 PA13 和 PA14。

### ① 硬件准备

1. STM32 控制器
2. STlink

### ② 软件准备

对应的 STlink 或者 Jlink 驱动的安装。

安装成功后可以打开设备管理器查看是否又显示 STLink 设备。



图 8-2-1 设备管理器查看 STlink 驱动

可以看到驱动已经安装成功！

### ③ 接线

STlink ----- STM32 控制器  
SWDIO-----PA13  
SWCLK-----PA14  
GND-----GND

OK, 一切准备就绪。

### ④ 下载程序

点击图 8-2-2 中的箭头所指的按钮，程序就可以下载了！因为勾选了编程后执行，所以程序下载完后，会自动运行。默认程序的配置是针对 STlink 的，如需配置 Jlink 下载，需要修改 MDK 设置。

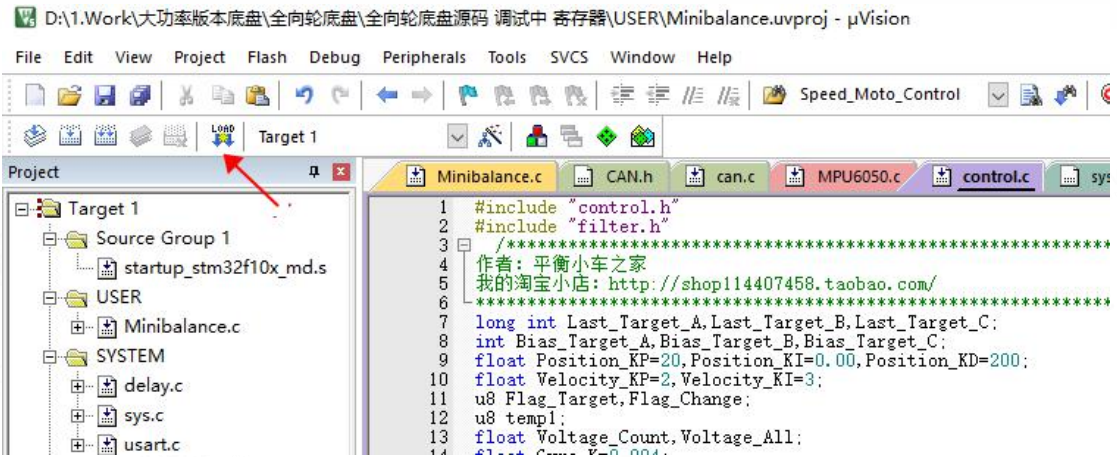


图 8-2-2 STlink 下载程序界面