# 轮 趣 科 技

# CAN 通信控制与反馈

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 | 2024/04/25 | 第一次发布 |
| V1.1 | 2025/01/06 | 章节 1 和 4 新增波特率相关说明 |

网址：www.wheeltec.net

# 目录

# 1. 通讯协议

这一章主要是对我司机器人运动底盘的 CAN 通讯协议做一个详细的说明。我司的 ROS 教育机器人底盘和大型 ROS 科研机器人底盘使用的 CAN 通讯协议是一样的，这种统一性不仅便于维护和升级，而且也为跨平台的协作提供了极大的便利。

文章中的数据上下行都是相对于机器人运动底盘的，上行数据指的是机器人底盘通过 CAN 接口发送出来的状态数据，下行数据指的是机器人底盘通过 CAN 接口接收到的控制信息。

## 1.1 机器人运动速度坐标系说明

我司的机器人可以通过 CAN 通信控制 XYZ 三个方向的运动（Y 方向只有全向机器人才支持）

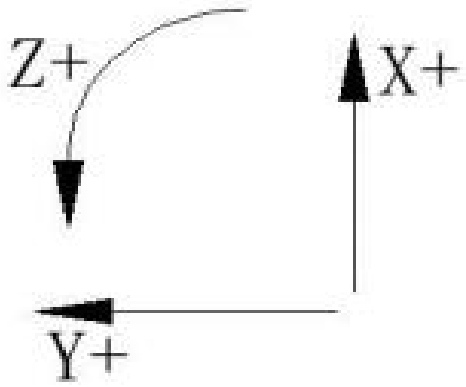



图 1-1 机器人 XYZ 三轴方向

如图 1-1 所示，机器人的 X 轴速度正方向向前，Y 轴速度正方向向左，Z 轴速度正方向为绕 Z 轴(竖直向上)顺时针旋转。

## 1.2 上行数据帧解析

机器人底盘 CAN 上行的数据跟串口上行的数据内容是相同的，机器人底盘的上行数据较多共有 24 字节，发送数据时是一次发送 8 个字节，分 3 次发送，即机器人底盘的上行数据帧共有 3 帧，每一帧都有机器人底盘上不同的信息。接收端可以通过标识符 ID 确认当前接收的是第几组数据，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。

① **帧 ID：0X101**

数据帧说明如下：

帧 ID：0X101

帧类型：标准帧

帧格式：DATA

DLC：8 字节

帧 ID0X101 的上行数据帧内容如下表：

| 数据域 | Rx[0] | Rx[1] | Rx[2] | Rx[3] | Rx[4] | Rx[5] | Rx[6] | Rx[7] |
|---|---|---|---|---|---|---|---|---|
| 内容 | 0X7B | flag_stop | X 轴速度 | | Y 轴速度 | | Z 轴速度 | |
| 数据类型 | uint8 | uint8 | short | | short | | short | |
| 高低位序 | | | MSB | LSB | MSB | LSB | MSB | LSB |

表 1-1 帧 ID 为 0X101 的上行数据帧

以下是数据帧内容的含义:

0X7B：固定内容，0X7B，此处无意义，占一个字节；

Flag_stop：电机使能标志，0x00 时电机使能，其他值失能，占一个字节；

X 轴速度：机器人 X 轴上的速度，单位是 mm/s，占两个字节；

Y 轴速度：机器人 Y 轴上的速度，单位是 mm/s，占两个字节；

Z 轴速度：机器人 Z 轴上的速度放大 1000 倍，单位是 rad/s，占两个字节；

② **帧 ID：0X102**

数据帧说明如下：

帧 ID：0X102

帧类型：标准帧

帧格式：DATA

DLC：8 字节

帧 ID0X102 的上行数据帧内容如下表：

| 数据域 | Rx[0] | Rx[1] | Rx[2] | Rx[3] | Rx[4] | Rx[5] | Rx[6] | Rx[7] |
|---|---|---|---|---|---|---|---|---|
| 内容 | X 轴加速度 | | Y 轴加速度 | | Z 轴加速度 | | X 轴角速度 | |
| 数据类型 | Short | | Short | | Short | | Short | |
| 高低位序 | MSB | LSB | MSB | LSB | MSB | LSB | MSB | LSB |

表 1-1 帧 ID 为 0X102 的上行数据帧

以下是数据帧内容的含义:

X 轴加速度：加速度计 X 轴上的加速度【原始数据】，占两个字节；

Y 轴加速度：加速度计 Y 轴上的加速度【原始数据】，占两个字节；

Z 轴加速度：加速度计 Z 轴上的加速度【原始数据】，占两个字节；

X 轴角速度：陀螺仪 X 轴上的角速度【原始数据】，占两个字节；

③ **帧 ID：0X103**

数据帧说明如下：

帧 ID：0X103

帧类型：标准帧

帧格式：DATA

DLC：8 字节

帧 ID0X103 的上行数据帧内容如下表：

| 数据域 | Rx[0] | Rx[1] | Rx[2] | Rx[3] | Rx[4] | Rx[5] | Rx[6] | Rx[7] |
|---|---|---|---|---|---|---|---|---|
| 内容 | Y 轴角速度 | | Z 轴角速度 | | 电池电压 | | 校验位 | 0X7D |
| 数据类型 | Short | | Short | | Short | | uint8 | uint8 |
| 高低位序 | MSB | LSB | MSB | LSB | MSB | LSB | | |

表 1-1 帧 ID 为 0X103 的上行数据帧

以下是数据帧内容的含义:

Y 轴角速度：陀螺仪 Y 轴上的角速度【原始数据】，占两个字节；

Z 轴角速度：陀螺仪 Z 轴上的角速度【原始数据】，占两个字节；

电池电压：机器人的电池电压大小，单位是 mV（毫伏），占两个字节；

数据校验位：前 22 个字节异或校验（BCC 校验）的结果，CAN 通信自带校验，可以不再进行异或位校验，占一个字节；

0X7D：固定内容，0X7D，此处无意义，占一个字节；

④ 注意事项

1） 底盘三轴速度来源及解析

底盘的三轴速度是通过测量每个轮子的速度，然后通过运动学正解算法，结合底盘的几何参数，计算出底盘在三个方向上的速度。而轮子的速度是由电机的编码器数据计算得到的。

2） 底盘实时姿态数据来源解析

机器人的实时姿态数据是由姿态传感器提供的，其中包括机器人 XYZ 三轴加速度和绕 XYZ 三轴的角速度，这里的 X、Y、Z 与 ROS 机器人运动方向的设置是一致的，X 轴前进为正，Y 轴向左为正，Z 轴逆时针为正，如图 1-2 所示。

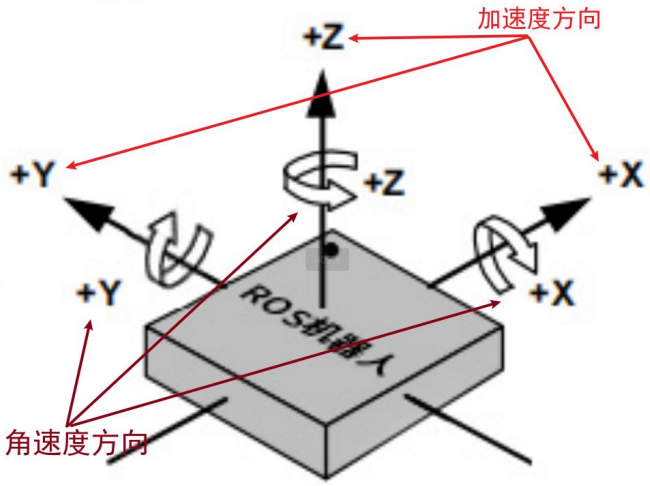



图 1-2 ROS 机器人姿态数据正方向

通过图 1-2 所示，机器人的三轴加速度方向为：X 轴加速度正方向为正前方，Y 轴加速度正方形为正左方，Z 轴加速度正方向为垂直于地面竖直向上。

机器人的三轴角速度方向正方向均为绕着对应轴顺时针旋转，如 Z 轴角速度的正方向为绕 Z 轴顺时针旋转。

以上机器人的姿态数据来源于姿态传感器均为原始数据，因此需要进行量程的转化，加速度计原始数据需要除以 1672 将单位转化 m/s²（米/平方秒）。角速度计原始数据需要除以 3753 将单位转化 rad/s（弧度/秒）。

如下图 1-3、1-4 所示，为底盘上控制板的 IMU 芯片：

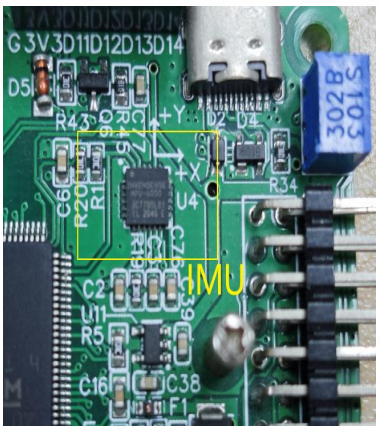



图 1-3 ROS 教育机器人上的 IMU 位置


图 1-4 大型 ROS 科研机器人上的 IMU 位置

图中的 IMU 方向丝印是指 IMU 三轴的原始方向，而数据帧里 IMU 三轴方向是经过我司重定义的，数据帧中的 IMU 三轴方向具体参考上文中图 1-1 机器人 XYZ 三轴方向。

## 1.3 下行数据帧解析

机器人运动底盘通过 CAN 接口接收控制指令，控制指令的帧 ID 为 0X181，底盘通过解析其中的数据内容获得下发的控制指令。

① 帧 ID：0X181

数据帧说明如下：

帧 ID：0X181

帧类型：标准帧

帧格式：DATA

DLC：8 字节

帧 ID0X181 的下行数据帧内容如下表：

| 数据域 | Rx[0] | Rx[1] | Rx[2] | Rx[3] | Rx[4] | Rx[5] | Rx[6] | Rx[7] |
|---|---|---|---|---|---|---|---|---|
| 内容 | X 轴目标速度 | | Y 轴目标速度 | | Z 轴目标速度 | | 预留位 | 预留位 |
| 数据类型 | Short | | Short | | Short | | | |
| 高低位序 | MSB | LSB | MSB | LSB | MSB | LSB | | |

表 1-1 帧 ID 为 0X181 的下行数据帧

以下是数据帧内容的含义：

X 轴目标速度：机器人 X 轴上的目标速度，单位 mm/s，占两个字节；

Y 轴目标速度：机器人 Y 轴上的目标速度，单位 mm/s，占两个字节；

Z 轴目标速度：机器人 Z 轴上的目标速度放大 1000 倍，单位 rad/s，占两个字节；

预留位：Rx[6]、Rx[7]均为预留位，各占一个字节，用户可根据需求自定义。

② **注意事项**

机器人 XYZ 方向与章节 1.1 中介绍的保持一致。

## 1.4 波特率说明

对于 ROS 教育机器人和大型 ROS 科研机器人的 CAN 通讯波特率均为 1M（1000000），S 系列机器人的 CAN 通讯波特率则为 500K（500000），而波特率在源码中的设置以及具体计算方式可通过阅读章节 4.5 进行了解。

# 2. CAN 接口位置

我司的机器人底盘 STM32 控制器上集成有 CAN 总线收发器 VP230 芯片，用户可以通过 STM32 控制器上预留的 CAN 接口与底盘进行 CAN 通信。

本章主要介绍 CAN 接口在 ROS 教育机器人底盘和大型 ROS 科研机器人底盘上的位置。

## 2.1 ROS 教育机器人

以 ROS 教育机器人为例：

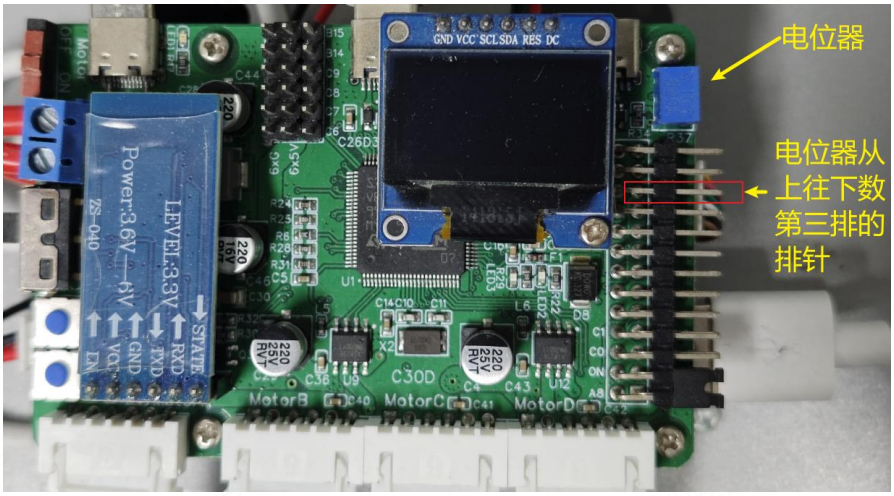



图 2-1　ROS 教育机器人（正面）CAN 接口位置

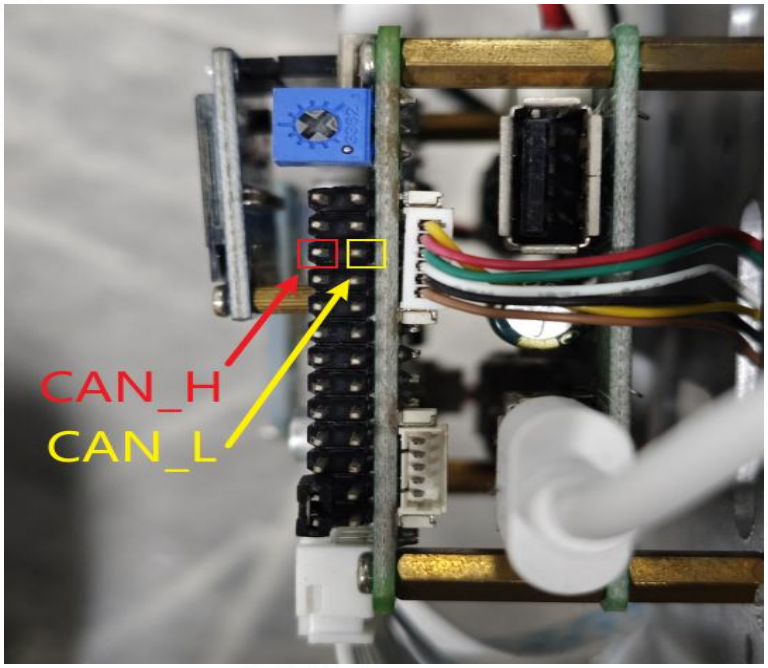



图 2-2　ROS 教育机器人（侧面）CAN 接口位置

## 2.2 大型 ROS 科研机器人

以大型 ROS 科研机器人为例，STM32 控制板上会预留两个 CAN 接口，分别位于正面以及背面，他们均为同一条 CAN 总线上的接口。

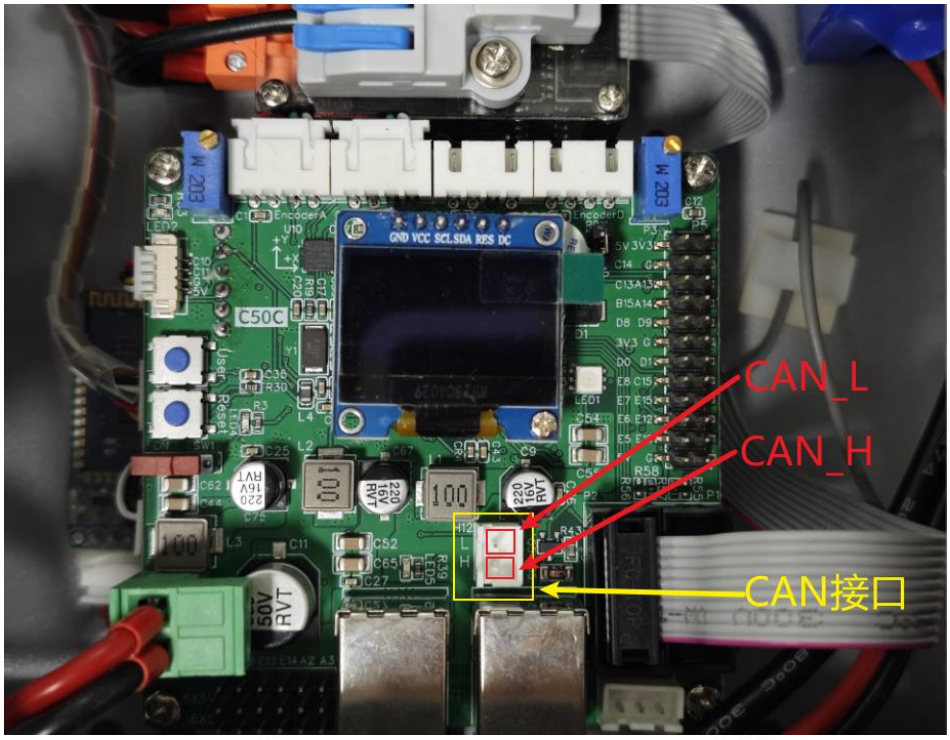



图 2-3　大型 ROS 科研机器人（正面）CAN 接口位置

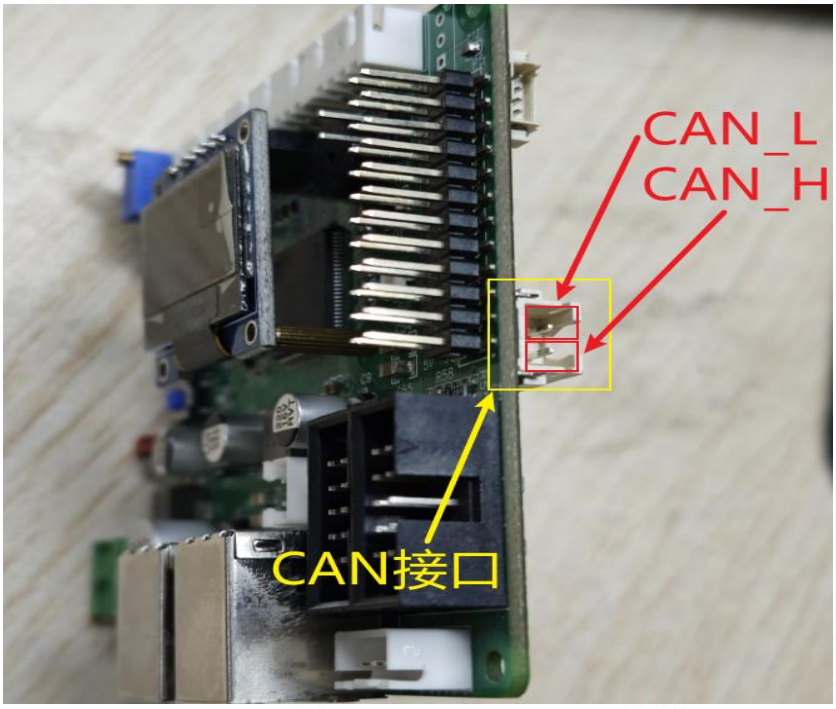



图 2-4　大型 ROS 科研机器人（侧面）CAN 接口位置

# 3. 控制与反馈

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机器人底盘发出来的数据，同时会下发控制指令给机器人底盘。例程代码适配的单片机型号是 STM32F103C8T6 或同系列的单片机，并且需要配备 CAN 总线收发器 VP230 芯片对单片机的数据进行转换之后才能正式与机器人进行 CAN 通信。

## 3.1 CAN 通信接线说明

先将烧录完 CAN 发送例程的单片机与 CAN 总线收发器 VP230 芯片进行接线，单片机的 PA11、PA12 引脚接 VP230 的 RXD、TXD 引脚，单片机的 3.3V、GND 引脚接 VP230 的 3.3V、GND 引脚。

以我司的 STM32F103C8T6 板子为例：

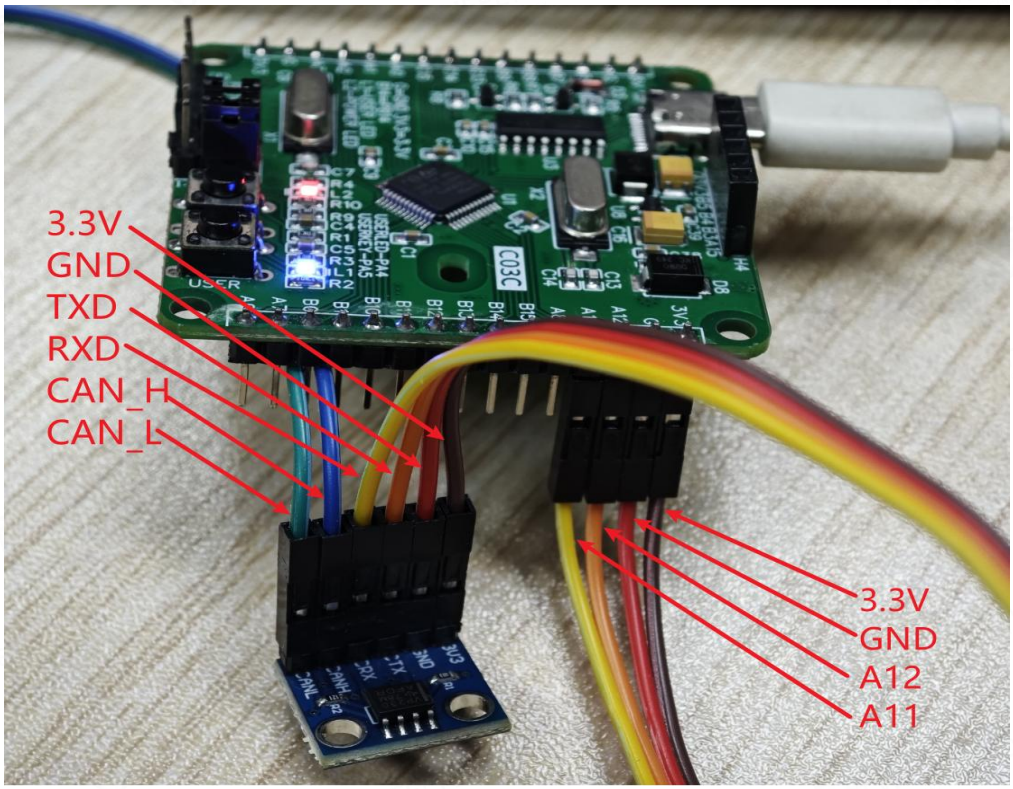



图 3-1　单片机与 VP230 的接线

① ROS 教育机器人

将 CAN 总线收发器 VP230 芯片剩下的两个引脚 CAN_H、CAN_L 与机器人底盘 STM32 控制板上的 CAN_H、CAN_L 相连接（即 H 对 H，L 对 L）。

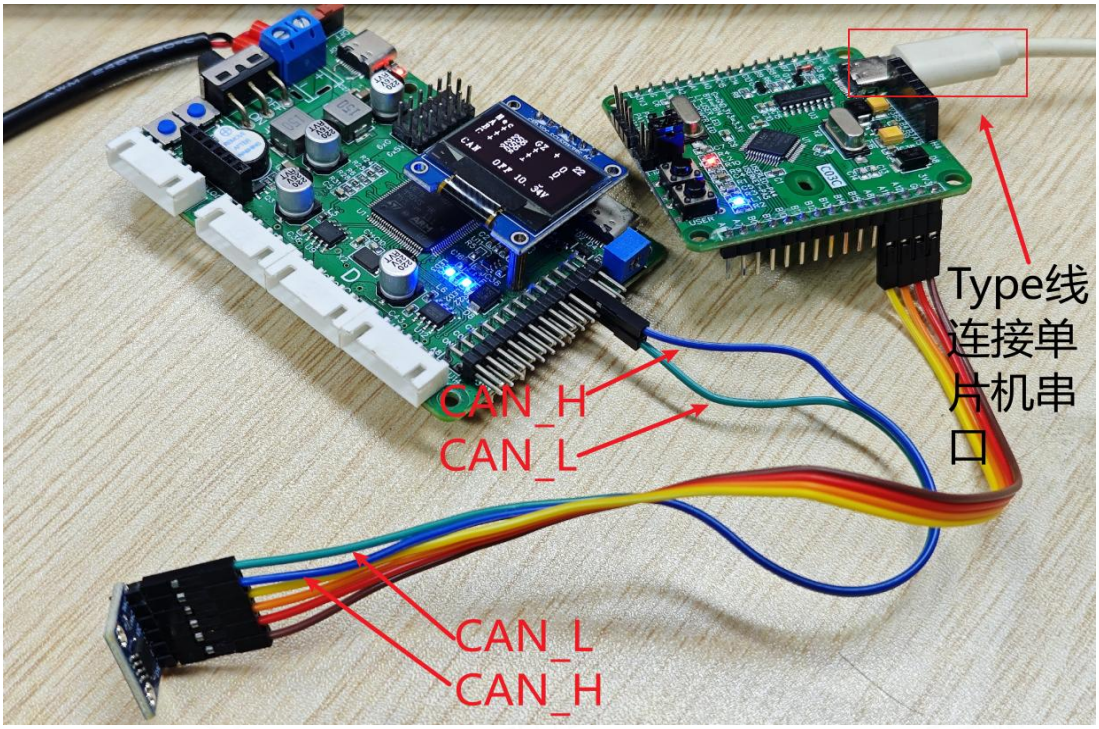



图 3-2　单片机与 ROS 教育机器人 CAN 通信接线

② 大型 ROS 科研机器人

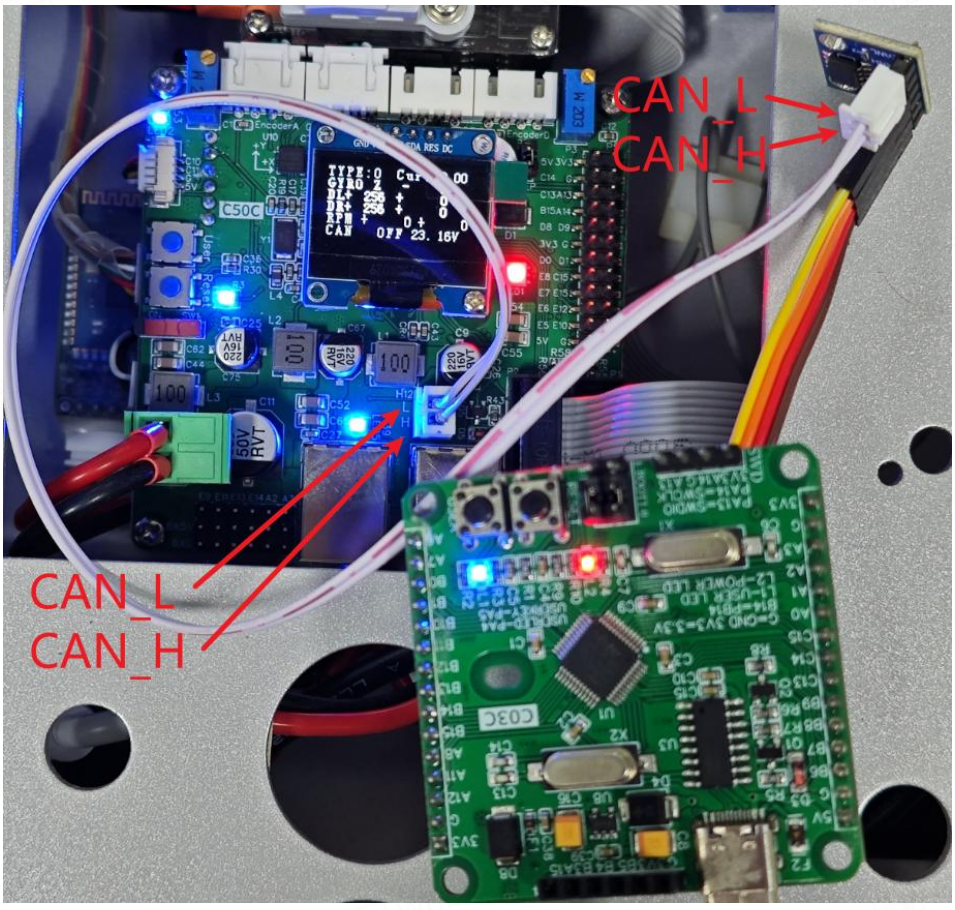



图 3-3　单片机与大型 ROS 科研机器人（正面）CAN 通信接线

将 CAN 总线收发器 VP230 芯片剩下的两个引脚 CAN_H、CAN_L 与机器人底盘 STM32 控制板上的任意 CAN 接口相连接（注意线序，H 对 H，L 对 L）。

## 3.2 CAN 控制模式反馈

按照章节 3.1 接完线后，我们建议先将机器人的底盘电机使能开关关闭，防止测试 CAN 通信例程时，由于机器人的运动造成不必要的损失。

① OLED 显示反馈

将底盘以及烧录 CAN 发送例程的单片机进行上电，当底盘成功接收到 CAN 接口发来的帧 ID 为 0X181 的数据帧后会将控制模式切换为 CAN 控制模式，可以通过底盘的 OLED 显示屏的左下角看到底盘的控制模式变成了 CAN 控制模式。

由于单片机通过 CAN 向机器人底盘发送的数据中只给定 X 轴方向速度，大小为 256mm/s，所以显示屏上电机目标速度为 256。如果没有电机使能开关关掉，可以观察到机器人底盘以 256mm/s 的速度向正前方运动。

显示屏内容如下图：

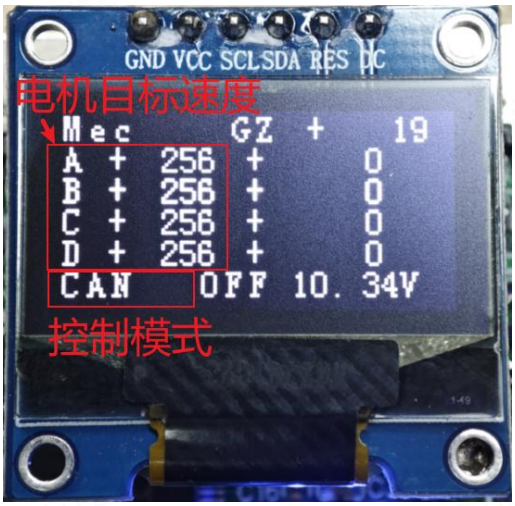



图 3-4 ROS 教育机器人显示屏


图 3-5 大型 ROS 科研机器人显示屏

② 上位机反馈

我司配套的 CAN 发送例程会将单片机通过 CAN 接收的中断服务函数接收到的机器人数据用串口 1 发送出来，用户可以下载安装我司提供的串口上位机——WHEELTEC 串口助手（通过资料获取）来对数据进行观察。

如下图，为单片机接收到机器人自身一帧完整的数据，共 24 字节数据，分别又帧 id 为 0X101、0X102、0X103 的三帧数据组成。


图 3-6　串口助手接收数据图

对于上图中接收到的这三帧数据帧进行解析，含义如下：

完整的数据帧为 7B 00 01 01 00 00 00 0B FF B8 00 70 3D DC FF F3 FF BF FF C8 5A 33 B4 7D 共 24 字节。

第 1 个字节：0x7B，无意义；

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(mm/s)，底盘当前 X 轴速度为 257mm/s；

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 0x00(16 进制)=0000 0000(2 进制),最高位为 0，正数(左移)，速度大小为 0*256+0=0(mm/s)，底盘当前 Y 轴速度为 0mm/s；

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 0x0B(16 进制)=0000 1011(2 进制),最高位为 0，正数(逆时针旋转)，速度大小为(0*256+11)/1000=0.011(rad/s)，底盘当前 Z 轴速度为 0.011rad/s；

第 9、10 个字节：X 轴加速度，高八位 0xFF(16 进制)=1111 1111(2 进制)，低 8 位 0xB8(16 进制)=1011 1000(2 进制),最高位为 1，负数，加速度大小为 2^16(FFFF+1)-FFB8=00 48(16 进制)=0*256+78=78，转化为 X 轴加速度 78/1672=0.0466（$m/s^2$）；

第 11、12 个字节：Y 轴加速度，高八位 0x00(16 进制)=0000 0000(2 进制)，低 8 位 0x70(16 进制)=0111 0000(2 进制),最高位为 0，正数，加速度大小为 0*256+112=112，转化为 Y 轴加速度 112/1672=0.0669（$m/s^2$）；

第 13、14 个字节：Z 轴加速度，高八位 0x3D(16 进制)=0011 1101(2 进制)，低 8 位 0xDC(16 进制)=1101 1100(2 进制),最高位为 0，正数，加速度大小为 61*256+220=15,836，转化为 Z 轴加速度 15836/1672=9.4712（$m/s^2$）；

第 15、16 个字节：X 轴角速度，高八位 0xFF(16 进制)=1111 1111(2 进制)，低 8 位 0xF3(16 进制)=1111 0011(2 进制),最高位为 1，负数，角速度大小为 2^16(FFFF+1)-FFF3=00 0D(16 进制)=0*256+13=13，转化为 X 轴角速度 13/3753=0.0034（rad/s）；

第 17、18 个字节：Y 轴角速度，高八位 0xFF(16 进制)=1111 1111(2 进制)，低 8 位 0xBF(16 进制)=1011 1111(2 进制),最高位为 1，负数，角速度大小为 2^16(FFFF+1)-FFBF=00 41(16 进制)=0*256+65=65，转化为 Y 轴角速度 65/3753=0.0173（rad/s）；

第 19、20 个字节：Z 轴角速度，高八位 0xFF(16 进制)=1111 1111(2 进制)，低 8 位 0xC8(16 进制)=1100 1000(2 进制),最高位为 1，负数，角速度大小为 2^16(FFFF+1)-FFC8=00 38(16 进制)=0*256+56=56，转化为 Z 轴加速度 56/3753=0.0149（rad/s）；

第 21、22 个字节：电池电压，高八位 0x5A(16 进制)=0101 1010(2 进制)，低 8 位 0x33(16 进制)=0011 0011(2 进制),最高位为 0，正数，电池电压大小为 90*256+51=23091(mV)，转化为电压 23091/1000=23.091V；

第 23 个字节：BCC 校验(前 22 字节异或校验)，用于串口通信验证，此处无意义，该数据位的计算方法可参考手册串口通信控制与反馈的第 3 章；

第 24 个字节：0x7D,帧尾，无意义；

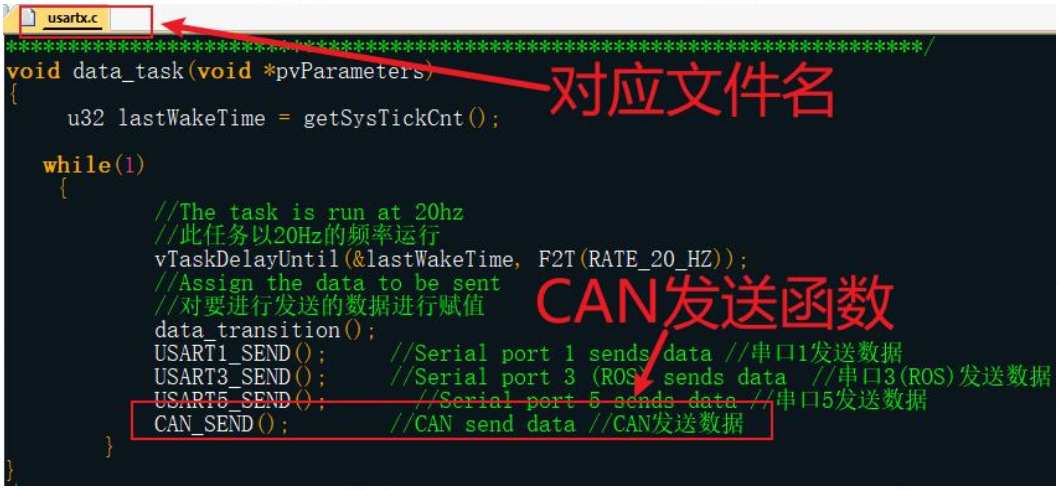
# 4. CAN 通讯 STM32 代码介绍

本章主要是对 CAN 通信相关的代码进行简要的介绍以及指出代码在程序中具体的位置，更加具体的内容见相关视频讲解。

由于我司的 ROS 教育机器人以及大型 ROS 科研机器人的程序结构几乎是一样的，所以本节后续将使用 ROS 教育机器人的程序来进行演示讲解，打开程序的软件为 keil5。

## 4.1 CAN 发送数据

CAN 发送数据的代码主要位于程序中的数据发送任务里，该任务位于 usartx.c 的文件开头处。CAN 发送数据由函数 CAN_SEND（）实现。




```
void data_task(void *pvParameters)
{
    u32 lastWakeTime = getSysTickCnt();

  while(1)
   {
        //The task is run at 20hz
        //此任务以20Hz的频率运行
        vTaskDelayUntil(&lastWakeTime, F2T(RATE_20_HZ));
        //Assign the data to be sent
        //对要进行发送的数据进行赋值
        data_transition();
        USART1_SEND();      //Serial port 1 sends data //串口1发送数据
        USART3_SEND();      //Serial port 3 (ROS) sends data  //串口3(ROS)发送数据
        USART5_SEND();        //Serial port 5 sends data //串口5发送数据
        CAN_SEND();         //CAN send data //CAN发送数据
    }
}
```

图 4-1 CAN 发送数据代码位置图

如图 4-2 所示，为 CAN 发送代码函数的位置。函数中将 0x101、0x102、0x103 三帧数据依次发送。


图 4-2 CAN 发送函数位置图

## 4.2 发送频率的设置

CAN 发送数据的频率是由数据发送任务的执行频率决定的，若需修改 CAN 数据发送的频率则在数据发送任务中将执行频率改为对应的频率即可。

如图 4-3 所示，为数据发送频率修改的位置图。


图 4-3 CAN 发送频率修改位置

程序中对数据的发送频率进行了定义，以下是支持修改的数据发送频率。程序中的 RATE_20_HZ 表示设定的串口发送频率为 20hz（一秒发送 20 次）。

如图 4-4 所示，为程序支持的 CAN 发送频率。

```
#define RATE_1_HZ        1
#define RATE_5_HZ        5
#define RATE_10_HZ       10
#define RATE_20_HZ       20
#define RATE_25_HZ       25
#define RATE_50_HZ       50
```

图 4-4 支持的发送频率

## 4.3 发送前数据处理

由于我司机器人底盘 CAN 上行的数据跟串口上行的数据内容是相同的，所以 CAN 发送数据前的数据处理与串口数据发送数据前的数据处理使用的是相同的代码。CAN 通信将 24 字节的数据处理完后，每 8 个字节作为一帧数据进行发送，且帧 ID 分别为 0x101、0x102、0x103。

如图 4-5 所示，为数据处理和赋值的位置图。


图 4-5 数据处理和赋值位置图

该函数中实现了对上行数据帧中各个数据的处理，其中包括单位转化、运动学分析、数据赋值处理等。由于该函数较长，此处仅对部分代码片段演示以及简要介绍，具体内容见教程视频。

① 单位转化及运动学分析

该代码片段是对 ROS 教育机器人差速车型对底盘的运动学分析，根据电机编码器数据进行运动学正解，以及单位转化，解算出底盘当前的三轴速度。

```
case Diff_Car:
Send_Data.Sensor_Str.X_speed = ((MOTOR_A.Encoder+MOTOR_B.Encoder)/2)*1000;
Send_Data.Sensor_Str.Y_speed = 0;
Send_Data.Sensor_Str.Z_speed = ((MOTOR_B.Encoder-MOTOR_A.Encoder)/Wheel_spacing)*1000;
```

图 4-6 单位转化及运动学分析图

② 数据赋值处理

该代码片段是对底盘的各个数据进行赋值处理，将高于 8 位的数据拆分成多

个 8 位数据分别存储进数据发送缓冲区中（Send_Data.buffer）。

```
Send_Data.buffer[0]=Send_Data.Sensor_Str.Frame_Header; //Frame_heade //帧头
Send_Data.buffer[1]=Flag_Stop; //Car software loss marker //小车软件失能标志位
//The three-axis speed of / / car is split into two eight digit Numbers
//小车三轴速度,各轴都拆分为两个8位数据再发送
Send_Data.buffer[2]=Send_Data.Sensor_Str.X_speed >>8;
Send_Data.buffer[3]=Send_Data.Sensor_Str.X_speed ;
Send_Data.buffer[4]=Send_Data.Sensor_Str.Y_speed>>8;
Send_Data.buffer[5]=Send_Data.Sensor_Str.Y_speed;
Send_Data.buffer[6]=Send_Data.Sensor_Str.Z_speed >>8;
Send_Data.buffer[7]=Send_Data.Sensor_Str.Z_speed ;
```

图 4-7 数据赋值处理图

# 4.4 CAN 接收数据

底盘接收到的数据处理位于程序的 CAN 接收中断里，在接收中断中主要进行的是数据帧的筛选、三轴目标速度解析以及阿克曼车型 Z 轴数据的特殊处理。

如图 4-8 所示，为 CAN 接收中断函数。

```
main.c  can.c
301 Function: CAN receives interrupt service function, conditional compilation
302 Input   : none
303 Output  : none
304 函数功能：CAN接收中断服务函数，条件编译
305 入口参数：无
306 返回  值：无
307 ******************************************************************************/
308 #if CAN1_RX0_INT_ENABLE //Enable RX0 interrupt //使能RX0中断
309 void CAN1_RX0_IRQHandler(void)
310 {
311     u32 id;
312     u8 ide,rtr,len;
313
314     u8 temp_rxbuf[8];
315
316     CAN1_Rx_Msg(0,&id,&ide,&rtr,&len,temp_rxbuf);
317     if(id==0x181)
318     {
319         float Vz;
320         CAN_ON_Flag=1, PS2_ON_Flag=0,Remote_ON_Flag=0,APP_ON_Flag=0,Usart1_ON_Flag=0,Usart5_ON_Flag=0;
321         command_lost_count=0;
322         //Calculate the three-axis target velocity, unit: m/s
323         //计算三轴目标速度，单位：m/s
324         Move_X=((float)((short)((temp_rxbuf[0]<<8)+(temp_rxbuf[1]))))/1000;
325         Move_Y=((float)((short)((temp_rxbuf[2]<<8)+(temp_rxbuf[3]))))/1000;
326         Vz    =((float)((short)((temp_rxbuf[4]<<8)+(temp_rxbuf[5]))))/1000;
327         if(Car_Mode==Akm_Car)
328         {
329             Move_Z=Vz_to_Akm_Angle(Move_X, Vz);
330         }
331         else
332         {
333             Move_Z=((float)((short)((temp_rxbuf[4]<<8)+(temp_rxbuf[5]))))/1000;
334         }
335     }
336 }
337 #endif
```

对应的文件名

CAN1接收中断函数

①筛选帧ID为0x181的数据帧

②三轴目标速度解析

③阿克曼车型z轴数据特殊处理

图 4-8 CAN 接收中断函数

① 帧 ID 筛选

CAN 数据帧中自带帧 ID，底盘可以根据我们的需要过滤消息。底盘只接收与其帧 ID 匹配的消息，从而减少处理无关信息的负担，提高效率。在 CAN 接收中断函数中，底盘只对帧 ID 为 0x181 的数据帧进行处理，其他无关的数据帧

将被过滤掉。

② 三轴目标速度解析

由 1.2 节的下行数据帧解析可知，上位机下发的三轴目标速度拆分为 2 个 8 位数据，将目标速度的高 8 位与低 8 位分别存储进数据帧里。底盘接收到数据帧后需要将位 8 数据进行解析并重组成三轴目标速度。

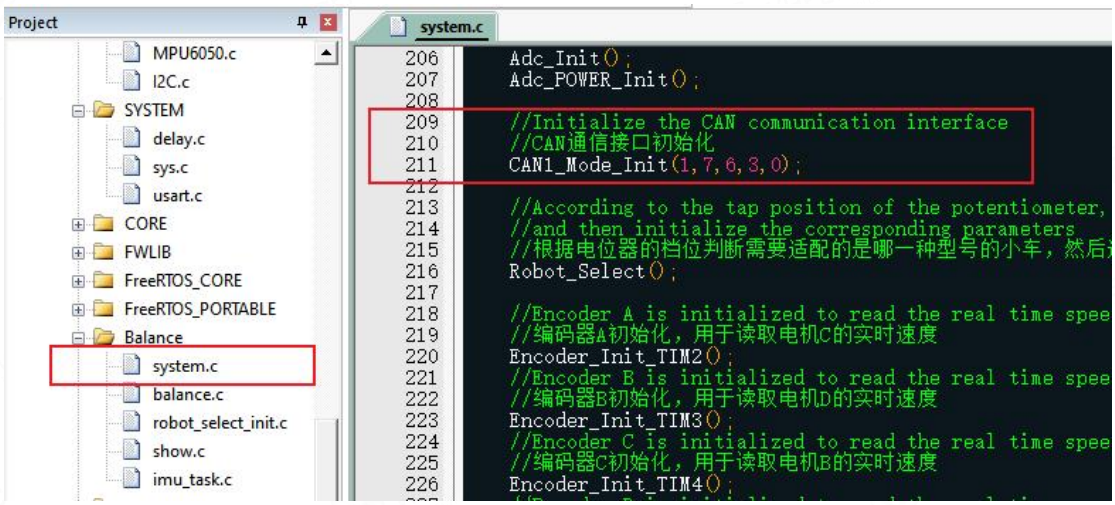
图 1-10 的三轴目标速度解析代码段中，将目标速度的高 8 位进行左移 8 位后与目标速度的低 8 位相加得到目标速度，同时除以 1000 将单位从 mm/s 转化为 m/s。

③ 阿克曼车型 Z 轴数据处理

由于阿克曼车型的转向结构较为是由舵机实现的，所以需要结合接收到的 X 轴与 Z 轴目标速度来进行前轮转向转角的解算，并且在会判断当前目标速度要求的转弯半径是否小于最小转弯半径。

## 4.5 波特率设置及计算

源码中 CAN 通讯波特率的设置位于 system.c 文件下的第 209 行处的 CAN1_Mode_Init()函数。




用户可通过修改传入函数的参数进而修改 CAN 通讯的波特率，修改方式如下：

CAN 初始化函数声明如下：

```
u8 CAN1_Mode_Init(u8 tsjw,u8 tbs2,u8 tbs1,u16 brp,u8 mode)
```

由函数声明可知将 CAN1 的 tsjw 设置为 1，tbs2 设置为 7，tbs1 设置为 6，

brp 设置为 3。

CAN 通讯波特率设置计算方法如下：

Baud rate=Fpclk1/((tbs1+tbs2+1)*brp)

CAN1 在 STM32F407VET6 上为 APB1 总线上的外设，因此 Fpclk1 为 APB1 当前的时钟频率，即 Fpclk1=42M。

将各值代入上式可得：

Baud rate=Fpclk1/((tbs1+tbs2+1)*brp)

=42M/((6+7+1)*3)

=1M