

轮趣科技

串口通信控制与反馈

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	2024/6/4	第一次发布

网址: www.wheeltec.net

目录

1. 通讯协议	3
1.1 机器人运动速度坐标系说明	3
1.2 上行数据帧解析	3
1.3 下行数据帧解析	6
2. 串口接口介绍	8
2.1 机器人串口接口的位置	8
2.2 串口应用	9
2.3 串口驱动安装	9
3. 控制与反馈	12
3.1 Windows 系统	13
3.2 Ubuntu 系统	18
4. 串口 STM32 代码介绍	21
4.1 串口发送数据	21
4.2 发送频率的设置	22
4.3 串口发送前数据处理	23
4.4 串口接收数据	24

1. 通讯协议

这一章主要是对我司机器人运动底盘的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串口通讯协议是一样的**，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向上位机发送的状态数据，下行数据指的是上位机向机器人底盘发送的控制信息。

1.1 机器人运动速度坐标系说明

我司的机器人可以通过串口控制 XYZ 三个方向的运动（Y 方向只有全向机器人支持）

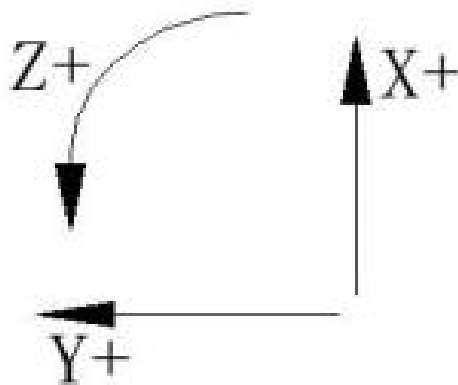


图 1-1 机器人 XYZ 三轴方向

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

机器人运动底盘通过串口发送的数据包格式，如下表所示：

数据内容	帧头(固定 值 0x7B)	flag_stop	X 轴速度		Y 轴速度		Z 轴速度		X 轴加速度		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	
高低位序			MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB
数组号	0	1	2	3	4	5	6	7	8	9	10	11
数据内容	Z 轴加速度	X 轴角速度		Y 轴角速度		Z 轴角速度		电池电压		数据校验 位	帧尾(固定 值 0x7D)	
数据类型	short	short		short		short		short		Uint8	Uint8	
占用字节	2	2		2		2		2		1	1	
高低位序	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB		
数组号	12	13	14	15	16	17	18	19	20	21	22	23

表 1-1 上行数据包格式表

① 数据帧含义

以下是数据帧各数据的含义：

帧头：固定值 0x7B，标识数据包的开始，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；

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；

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，占一个字节；

帧尾：固定值 0x7D，标识数据包的结束，占一个字节；

② 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。

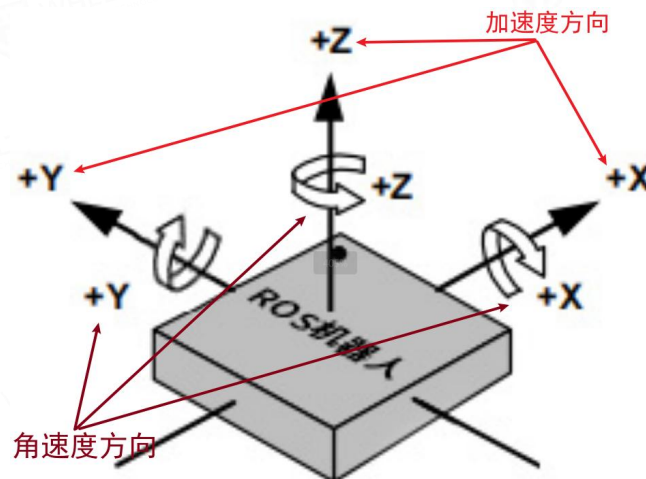


图 1-2 ROS 机器人姿态数据正方向

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 轴加速度正方向为正左方，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/s^2 （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 1-2、1-3 所示，为底盘上控制板的 IMU：

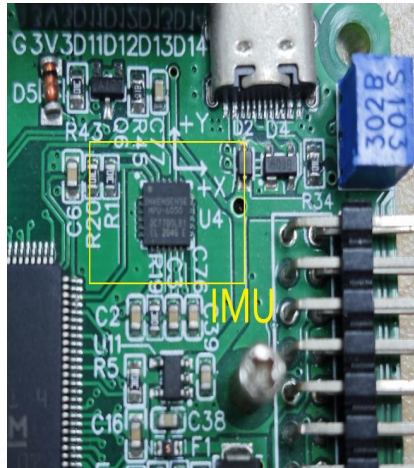


图 1-2 ROS 教育机器人上的 IMU 位置



图 1-3 ROS 大型科研机器人上的 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解析的内容。**

1.3 下行数据帧解析

机器人运动底盘通过串口接收到的数据包格式，如下表所示：

数据内容	帧头（固定值 0x7B）	预留位	预留位	X 轴目标速度		Y 轴目标速度		Z 轴目标速度		数据校验位	帧尾（固定值 0x7D）
数据类型	UInt8	UInt8	UInt8	Short		Short		Short		UInt8	UInt8
占用字节	1	1	1	2		2		2		1	1
高低位序				MSB	LSB	MSB	LSB	MSB	LSB		
数组号	0	1	2	3	4	5	6	7	8	9	10

表 1-2 下行数据包格式表

① 数据帧含义

以下是数据帧各数据的含义：

帧头：固定值 0x7B，标识数据包的开始，占一个字节；

预留位：第 2、3 个字节均为预留位，无意义，各占一个字节。

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；

数据校验位：前 9 个字节异或校验（BCC 校验）的结果，占一个字节；

帧尾：固定值 0x7D，标识数据包的结束，占一个字节；

② 注意事项

机器人三轴速度方向与章节 1.1 中介绍的保持一致。

2. 串口接口介绍

我司的机器人底盘 STM32 控制器上集成有 USB 转串口芯片，用户不仅可以使⤵串口对 STM32 的固件进行一键烧录，还可以通过串口通信与上位机进行通信。本章主要介绍串口的位置以及应用。

2.1 机器人串口接口的位置

① ROS 教育机器人

以 ROS 教育机器人为例：

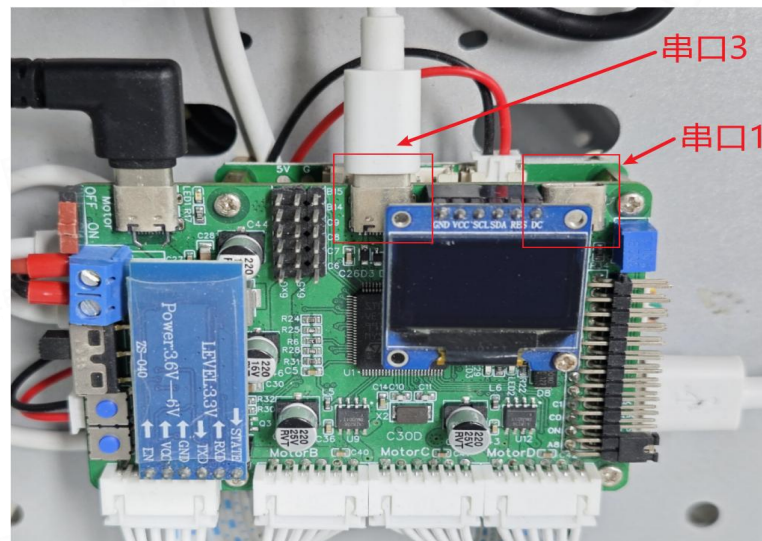


图 2-1 ROS 教育机器人串口位置图

② 大型 ROS 科研机器人

以大型 ROS 科研机器人为例：

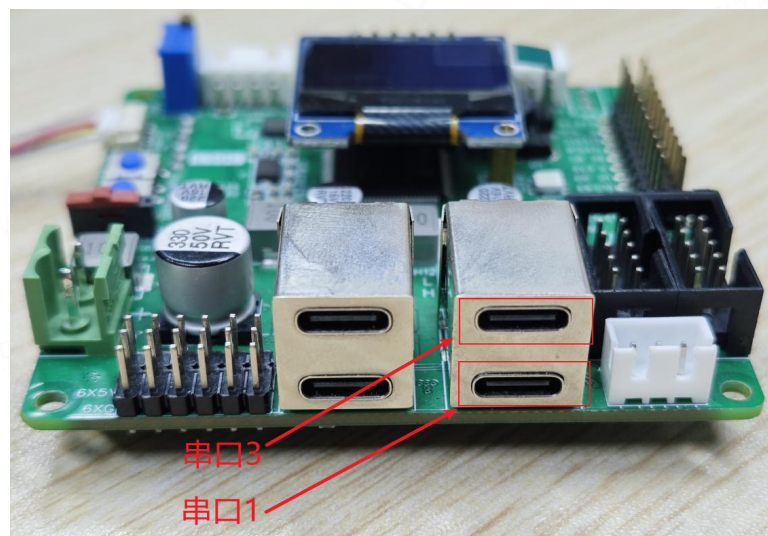


图 2-2 大型 ROS 科研机器人串口位置图

2.2 串口应用

机器人底盘上的串口主要用来连接上位机以及进行 ros 通信使用，串口 1 通常用来烧录底盘的固件以及调试使用，串口 3 通常用来与 ROS 上位机进行通信。

① ROS（串口 3）控制

ROS 和 STM32 控制器（机器人运动底盘）之间通过串口实现通信，STM32 控制器使用的是串口 3，波特率 115200，[通信协议详情见章节 1.通讯协议](#)。

机器人上电后默认是使用 ROS 控制模式，并且 ROS 控制在众多控制中拥有最高优先级。在 STM32 控制器的程序里，通过串口 3 向 ROS 发送机器人运动状态，并通过串口 3 接收 ROS 向机器人底盘发送的运动控制指令。

② 串口 1 控制

串口传输时需要注意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 的数据后机器人进入串口控制模式，显示屏左下角显示“USART”。

同时我们不建议使用大型 ROS 科研机器人和 ROS 教育机器人的串口 1 进行 ROS 通信，原因是机器人的底盘使用串口 1 与 ROS 进行通信时 STM32 控制器重新上电后会出现卡死的情况，原因是串口 1 主要是用于底盘程序的烧录，重新上电后收到的数据帧会被当成烧录程序用的数据包。

2.3 串口驱动安装

电脑端连接底盘的串口需要先安装对应的串口驱动，否则电脑将无法识别到底盘的串口，此处以 CH9102 串口芯片为例进行演示。

首先需要在我司提供的资料包内找到串口芯片对应的驱动压缩包。

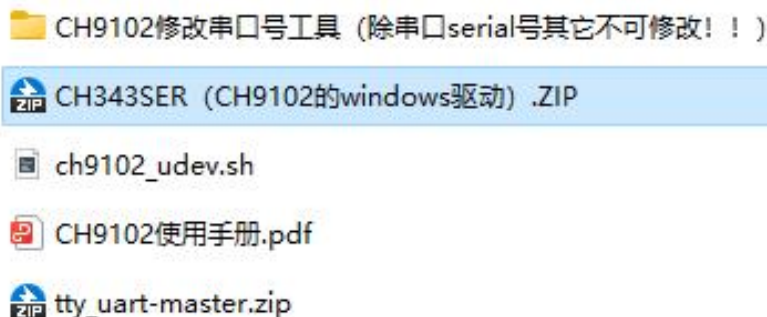


图 2-3 CH343SER

将压缩包解压后可以看到 CH343SER 文件夹，双击打开后找到 Driver 文件夹，里面有 SETUP.EXE 可双击打开，显示界面如图 2-4 所示。

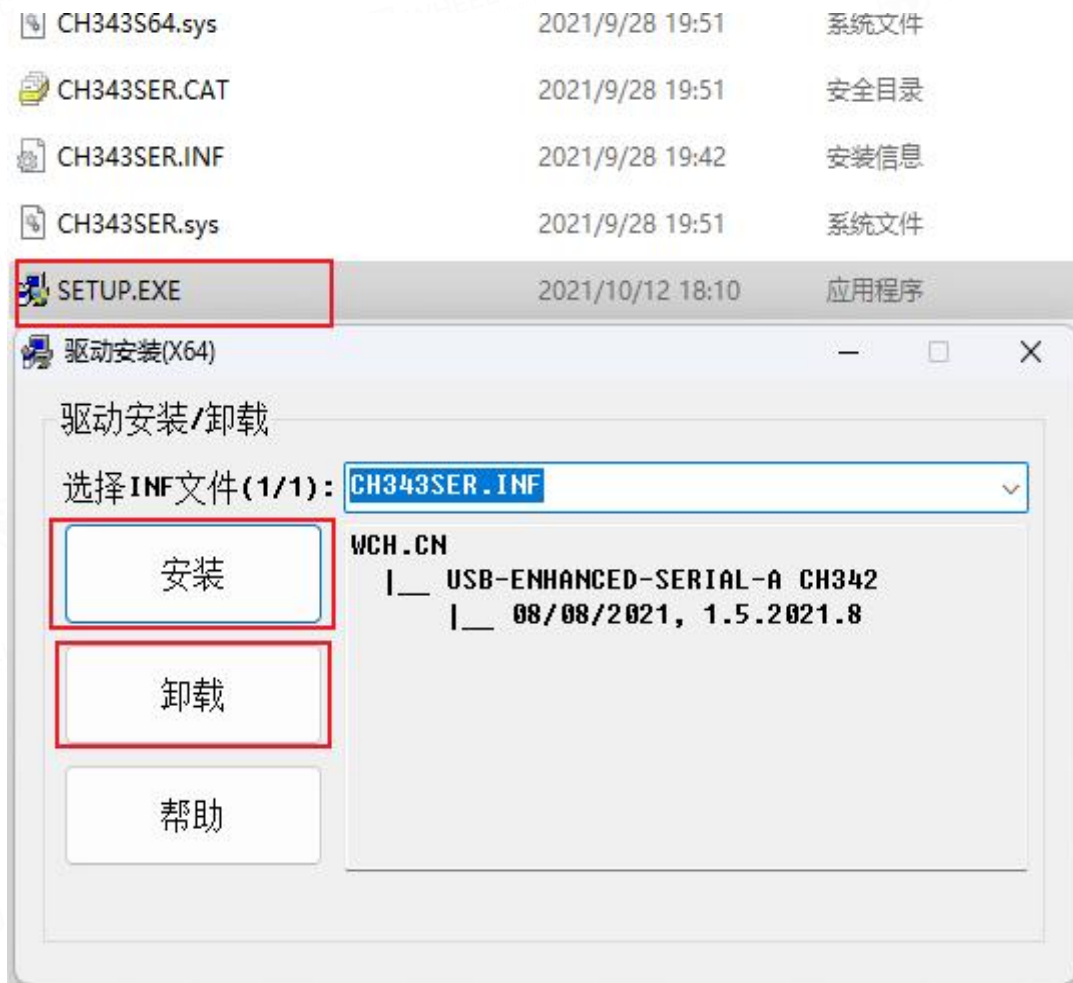


图 2-4 SETUP.EXE

点击安装按钮等待安装完成即可。

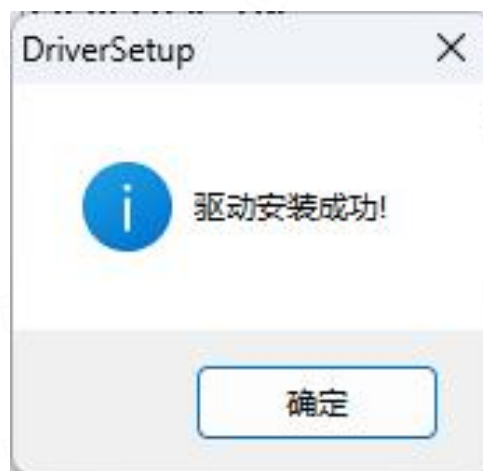


图 2-5 安装成功

最后打开设备管理器确认电脑是否能正常识别到底盘的串口, 如果无法正确是被则点击卸载按钮 (参考图 2-4) 重新进行驱动安装。

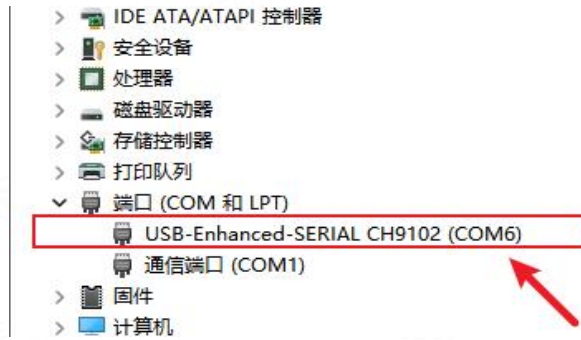


图 2-6 设备管理器面板图

3. 控制与反馈

本章主要介绍在 Windows 和 Ubuntu 系统下如何使用串口接收机器人底盘发出来数据以及对机器人底盘进行控制。

下文均使用 ROS 教育机器人底盘和大型 ROS 科研机器人底盘的 STM32 控制板上的串口 3 来演示不同系统下的上位机使用。

在使用上位机对机器人进行控制前，我们建议**将机器人的电机使能开关关闭**，以防出现上位机发送指令后小车出现乱跑的情况，照成不必要的损失。

ROS 教育机器人的电机使能开关处于 OFF 时电机失能。



图 3-1 ROS 教育机器人电机使能开关

大型 ROS 科研机器人的电机使能开关处于 SW1 时电机失能。

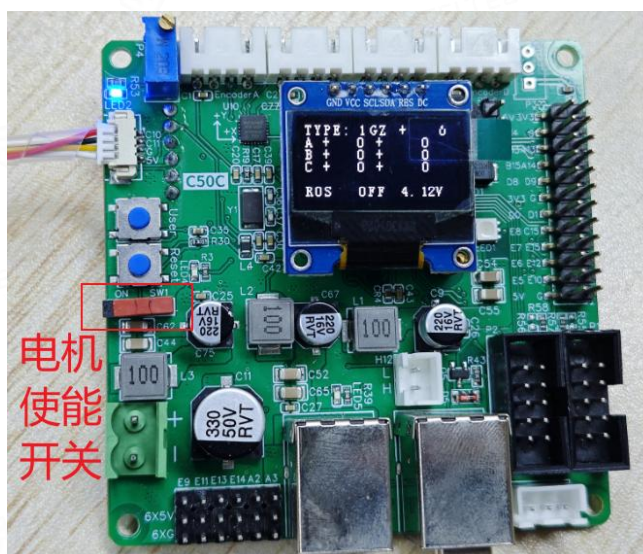


图 3-2 大型 ROS 科研机器人电机使能开关

3.1 Windows 系统

① WHEELTEC—串口助手获取

在 Windows 系统下，用户可以使用我们提供的上位机：WHEELTEC—串口助手，来对机器人进行串口控制以及数据接收。

在您购买了我们的机器人产品之后，可以通过我们提供的资料来下载 WHEELTEC—串口助手软件。

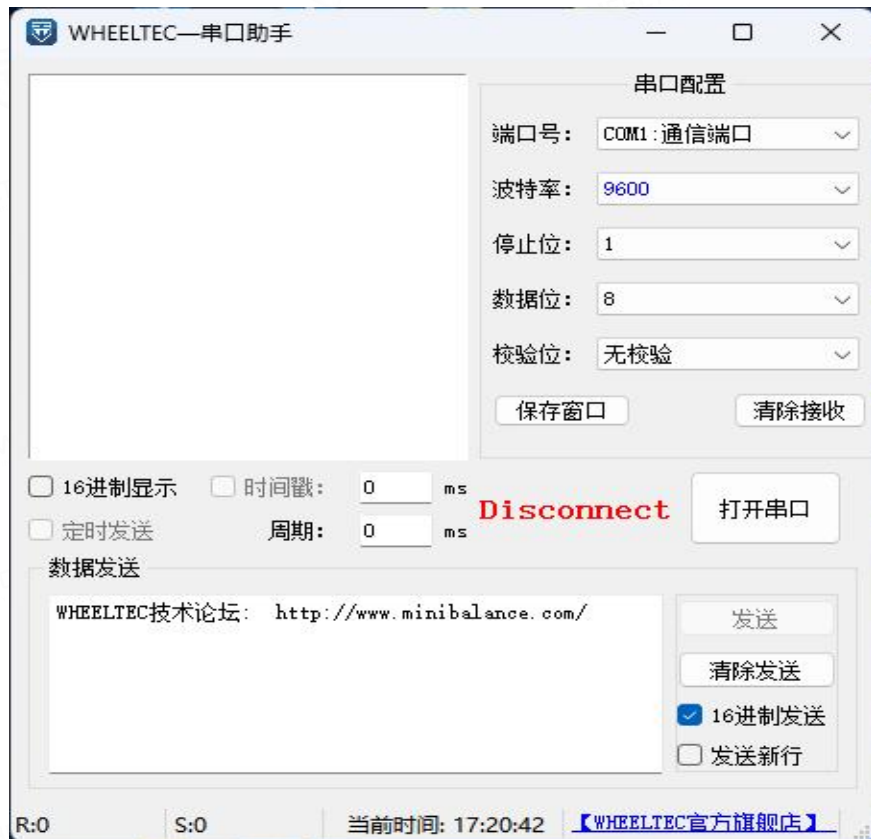


图 3-3 WHEELTEC—串口助手

② 串口助手接收数据

使用串口助手前，我们需要先安装对应的串口驱动，我们的机器人控制板是 USB 转 TTL 串口芯片是 CH9102 或 CH2102,这两款芯片对应的驱动我们也会在资料中为用户进行提供。用户安装完驱动后，使用数据线连接机器人控制板上的 **串口 3** 和电脑的 USB 接口后可以在电脑控制面板的设备管理器中查询到对应的端口号。



图 3-4 设备管理器

接下来我们就可以打开 WHEELTEC—串口助手软件，按照下图 3-5 设置好串口助手后，点击打开串口就可以接收到小车发出的数据帧了。



图 3-5 串口助手接收数据图

对于上图中接收到的这一帧数据帧进行解析，含义如下：

完整的数据帧为 7B 00 00 9B 00 00 FF DF 00 60 00 0C 40 A8 FF FD 00 06 00 1E 5B 87 82 7D 共 24 字节。

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 0x00(16 进制)=0000 0000(2 进制)、低 8 位 0x9B(16 进制)=1001 1011(2 进制),最高位为 0，正数(前进)，速度大小为

$0*256+155=155(\text{mm/s})$, 底盘当前 X 轴速度为 155mm/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 $0*256+0=0(\text{mm/s})$, 底盘当前 Y 轴速度为 0mm/s;

第 7、8 个字节: Z 轴速度, 高 8 位 0xFF(16 进制)=1111 1111(2 进制)、低 8 位 0xDF(16 进制)=1101 1111(2 进制),最高位为 1, 负数(顺时针旋转), 速度大小为 $2^{16}(\text{FF FF}+1)-\text{FF DF}=00\ 21(16\ \text{进制})=(0*256+33)/1000=0.033(\text{rad/s})$, 底盘当前 Z 轴速度为 0.033rad/s;

第 9、10 个字节: X 轴加速度, 高八位 0x00(16 进制)=0000 0000(2 进制), 低 8 位 0x60(16 进制)=0110 0000(2 进制),最高位为 0, 正数, 加速度大小为 $0*256+96=96$, 转化为 X 轴加速度 $96/1672=0.0574\ (\text{m/s}^2)$;

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 0x0C(16 进制)=0000 1100(2 进制),最高位为 0, 正数, 加速度大小为 $0*256+12=12$, 转化为 Y 轴加速度 $12/1672=0.0071\ (\text{m/s}^2)$;

第 13、14 个字节: Z 轴加速度, 高八位 0x40(16 进制)=0100 0000(2 进制), 低 8 位 0xA8(16 进制)=1010 1000(2 进制),最高位为 0, 正数, 加速度大小为 $64*256+168=16552$, 转化为 Z 轴加速度 $16552/1672=9.8995\ (\text{m/s}^2)$;

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 0xFD(16 进制)=1111 1101(2 进制),最高位为 1, 负数, 角速度大小为 $2^{16}(\text{FFFF}+1)-\text{FFFD}=00\ 02(16\ \text{进制})=0*256+2=2$, 转化为 X 轴角速度 $2/3753=0.0004\ (\text{rad/s})$;

第 17、18 个字节: Y 轴角速度, 高八位 0x00(16 进制)=0000 0000(2 进制), 低 8 位 0x06(16 进制)=0000 0110(2 进制),最高位为 0, 正数, 角速度大小为 $0*256+6=6$, 转化为 Y 轴角速度 $6/3753=0.0015\ (\text{rad/s})$;

第 19、20 个字节: Z 轴角速度, 高八位 0x00(16 进制)=0000 0000(2 进制), 低 8 位 0x1E(16 进制)=0001 1110(2 进制),最高位为 0, 正数, 角速度大小为 $0*256+30=30$, 转化为 Z 轴加速度 $30/3753=0.0079\ (\text{rad/s})$;

第 21、22 个字节: 电池电压, 高八位 0x5B(16 进制)=0101 1011(2 进制), 低 8 位 0x87(16 进制)=1000 0111(2 进制),最高位为 0, 正数, 电池电压大小为

$91 \times 256 + 135 = 23431(\text{mV})$ ，转化为加速度 $23431/1000 = 23.431\text{V}$ ；

第 23 个字节：BCC 校验(前 22 字节异或校验)，

$0x82 = 0x7B \oplus 0x00 \oplus 0x00 \oplus 0xFF \oplus 0xDF \oplus 0x00 \oplus 0x60 \oplus 0x00 \oplus 0x0C \oplus 0x40 \oplus 0xA8 \oplus 0xFF \oplus 0xFD \oplus 0x00 \oplus 0x06 \oplus 0x00 \oplus 0x1E \oplus 0x5B \oplus 0x87$ ；大家可以使用 window10/11 自带的计算器验证一下，需要选择【HEX】即 16 进制，异或的运算符号为【XOR】，如图 3-6 所示。



图 3-6 异或校验方法

第 24 个字节：0x7D,帧尾；

③ 串口助手发送数据

串口助手不仅可以通过串口接收数据，还可以发送数据。如下图 3-6，对串口助手的数据发送功能进行介绍。

首先我们将要发送的数据帧输入发送数据栏中，在将 16 进制发送的选项勾选上，点击发送就可以将数据发送出去了。

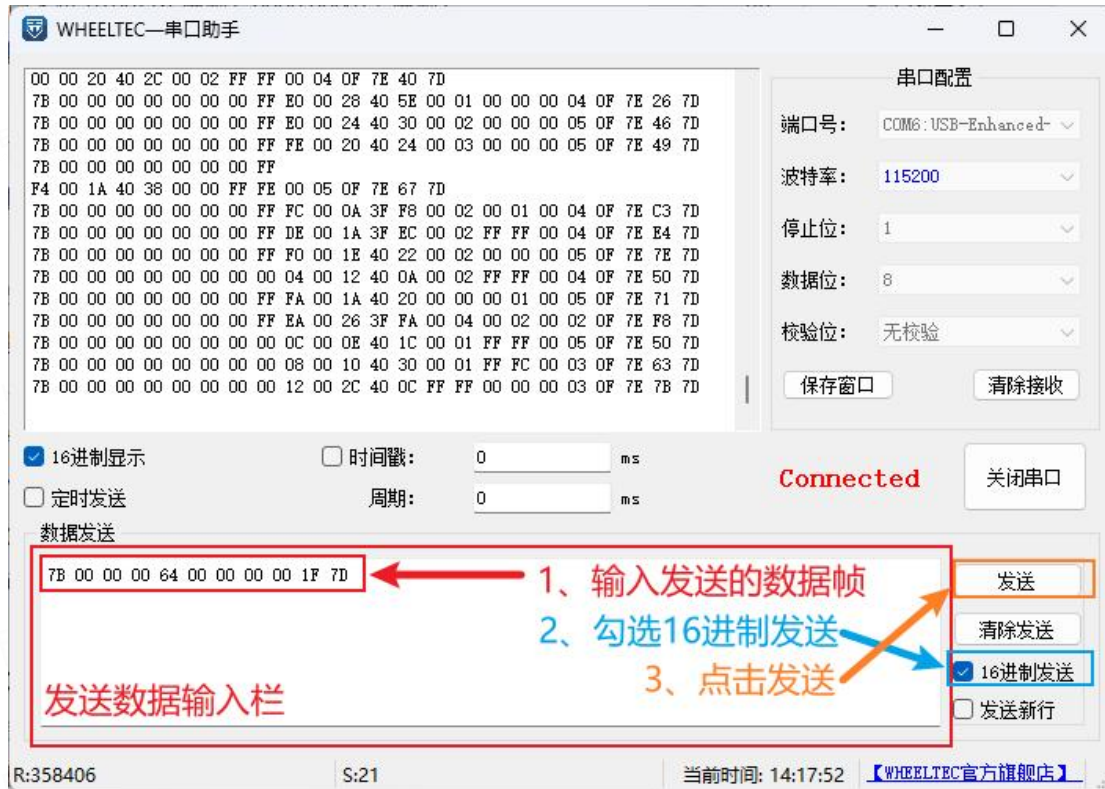


图 3-6 串口助手发送数据图

对于上图中发送到的这一帧数据帧进行解析，含义如下：

第 1 个字节：0x7B，帧头；

第 2、3 个字节：均为 0x00，预留位，数据无意义；

第 4、5 个字节：X 轴目标速度，高 8 位 0x00(16 进制)=0000 0000(2 进制)、低 8 位 0x64(16 进制)=0110 0100(2 进制),最高位为 0，正数(前进)，速度大小为 $0 * 256 + 100 = 100$ (mm/s)，设置底盘当前 X 轴目标速度为 100mm/s；

第 6、7 个字节：Y 轴目标速度，高低位均为 0x00，设置底盘当前 Y 轴目标速度为 0(mm/s)；

第 8、9 个字节：Z 轴目标速度，高低位均为 0x00，设置底盘当前 Z 轴目标速度为 0(mm/s)；

第 10 个字节：BCC 校验(前 9 位字节异或校验)，
 $0x1F = 0x7B \oplus 0x00 \oplus 0x00 \oplus 0x00 \oplus 0x64 \oplus 0x00 \oplus 0x00 \oplus 0x00 \oplus 0x00$ ；

第 11 个字节：0x7D，帧尾。

这个数据帧的控制效果是机器人底盘会以 100mm/s 的速度向前移动，如果将机器人的电机使能开关关闭了，也可以从机器人上的 OLED 显示屏上观察到控制现象。如下图 3-7，以我们大型 ROS 科研机器人(差速车型)上的 OLED 显示

屏为例，可以观察到显示屏上的电机目标速度变成了 100mm/s。



图 3-7 大型 ROS 科研机器人控制现象图

3.2 Ubuntu 系统

① cutecom 获取方法

在 Linux 系统下，用户可以通过安装 cutecom 来进行机器人底盘的串口收发数据。cutecom 是 Linux 系统下的图形界面形式的串口工具。用户可以在 Ubuntu 系统下使用终端来进行安装。

首先打开终端(ctrl+alt+T 键)，输入安装指令：

```
sudo apt-get install cutecom
```

此时会从 Ubuntu 的软件仓库中下载并安装 CuteCom。

```
下列【新】软件包将被安装：
cutecom
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 44 个软件包未被升级。
需要下载 98.2 kB 的归档。
解压后会消耗 354 kB 的额外空间。
获取:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic/universe amd64 cutecom amd64
0.30.3-1 [98.2 kB]
已下载 98.2 kB，耗时 1 秒 (138 kB/s)
正在选中未选择的软件包 cutecom。
(正在读取数据库 ... 系统当前共安装有 316637 个文件和目录。)
正准备解包 .../cutecom_0.30.3-1_amd64.deb ...
正在解包 cutecom (0.30.3-1) ...
正在设置 cutecom (0.30.3-1) ...
正在处理用于 desktop-file-utils (0.23-1ubuntu3.18.04.2) 的触发器 ...
正在处理用于 man-db (2.8.3-2ubuntu0.1) 的触发器 ...
正在处理用于 gnome-menus (3.13.3-11ubuntu1.1) 的触发器 ...
正在处理用于 mime-support (3.60ubuntu1) 的触发器 ...
wheeltec-client@ubuntu:~$
```

图 3-8 cutecom 安装

接下来我们可以通过终端来打开 cutecom，输入以下指令：

Sduo cutecom

此时会有如下图 3-9 的一个窗口弹出，就说明我们的 cutecom 安装成功了。

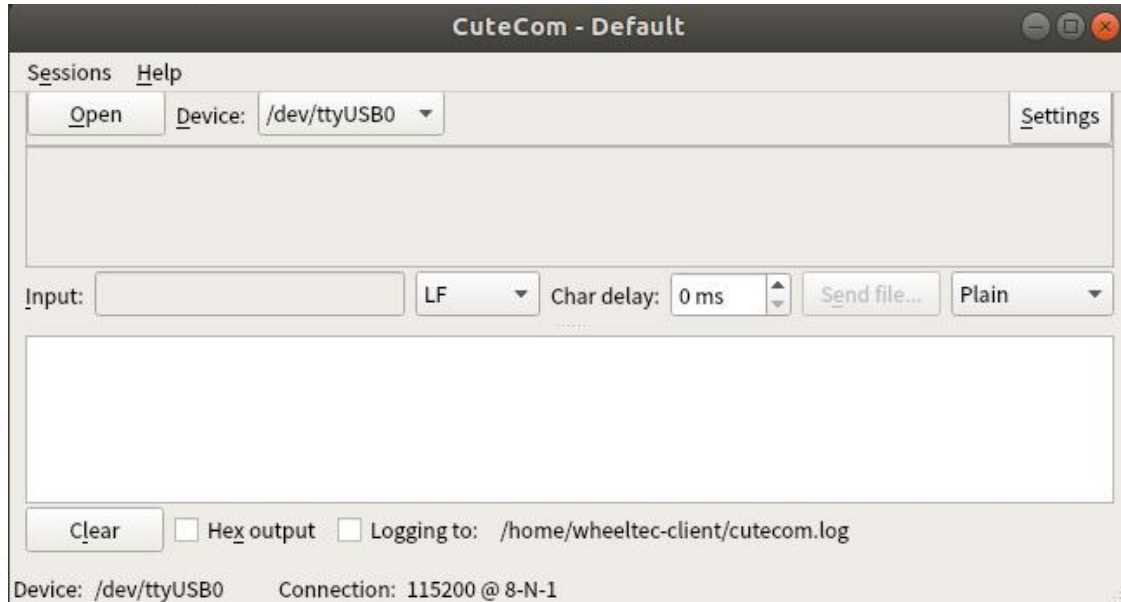


图 3-9 cutecom 界面图

② cutecom 接收数据

首先点击 cutecom 界面图右上角处的 Settings 按钮打开串口设置界面，并按照下图 3-10 进行配置。配置好后点击 Open 按钮与机器人底盘进行串口通信。

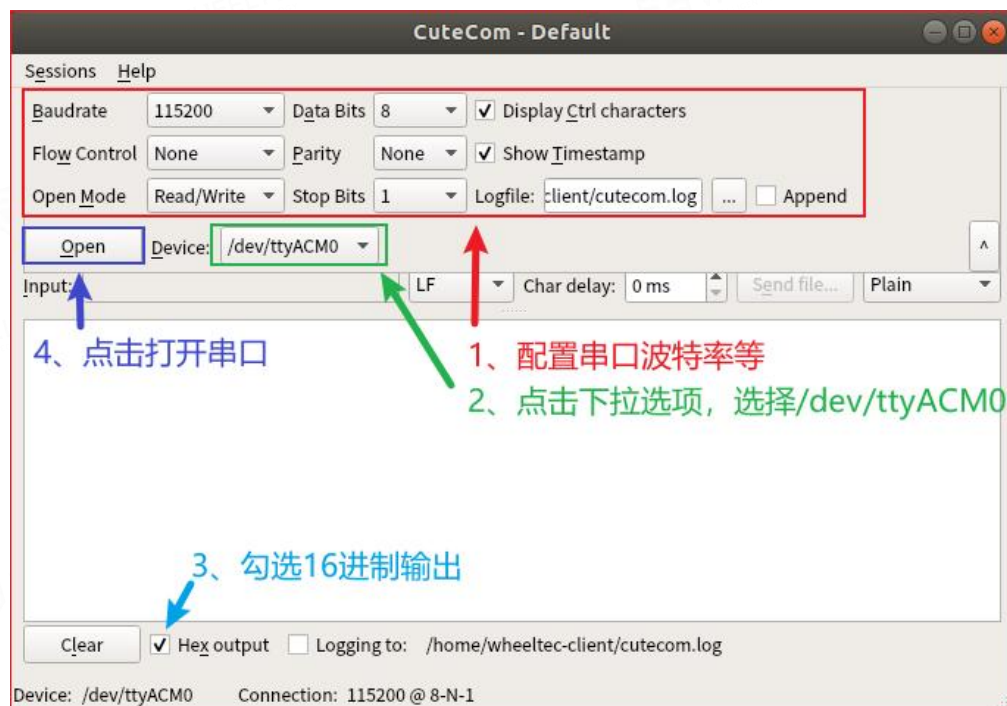


图 3-10 cutecom 配置图

打开串口后 cute 就能收到机器人底盘上发的数据了。

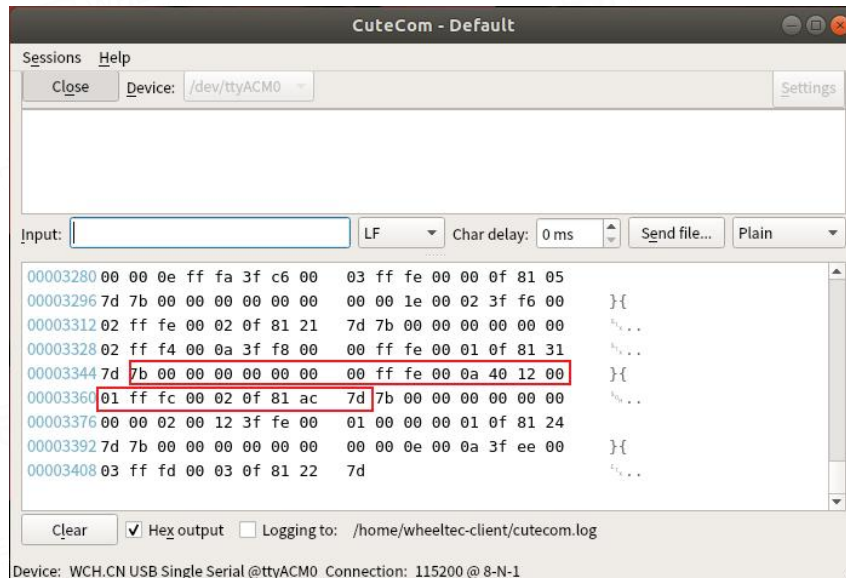


图 3-11 cutecom 接收数据

图 3-11 中的红框为收到的一帧数据帧。数据的具体含义这里不在进行相关介绍，具体的解算过程可以在[章节 3.1 中串口助手接收数据](#)有做详细的介绍。

③ cutecom 发送数据

cutecom 也可通过串口向机器人发送数据，将发送的数据格式选择为 Hex，接着把数据帧写入数据输入框内，敲下回车发送数据，数据会显示在数据显示框内。

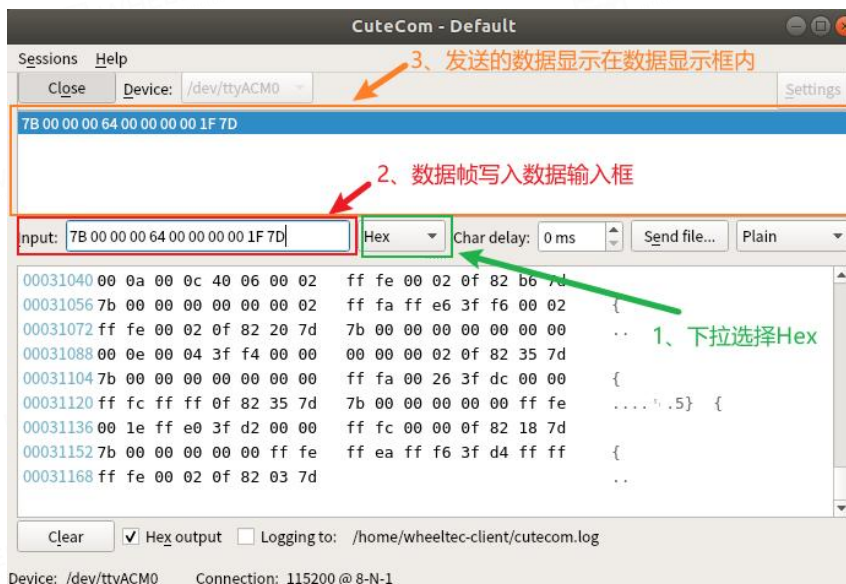


图 3-11 cutecom 发送数据

数据发送成功的现象与串口助手发送数据成功的现象一样，详情可参考[章节 3.1 中的串口助手发送数据](#)。

4. 串口 STM32 代码介绍

本章主要是对串口通信相关的代码进行简要的介绍以及指出代码在固件中具体的位置，**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 串口发送数据

串口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。串口 1 和 3 发送数据分别由 `USART1_SEND()` 和 `USART2_SEND()` 实现。

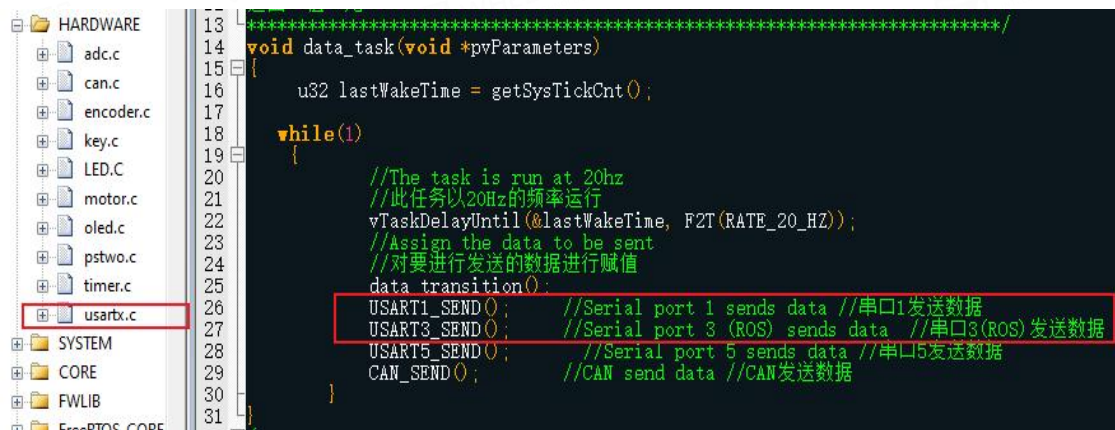


图 4-1 串口发送代码位置图

如图 4-2 所示，为串口 1 和 3 发送代码函数的位置。

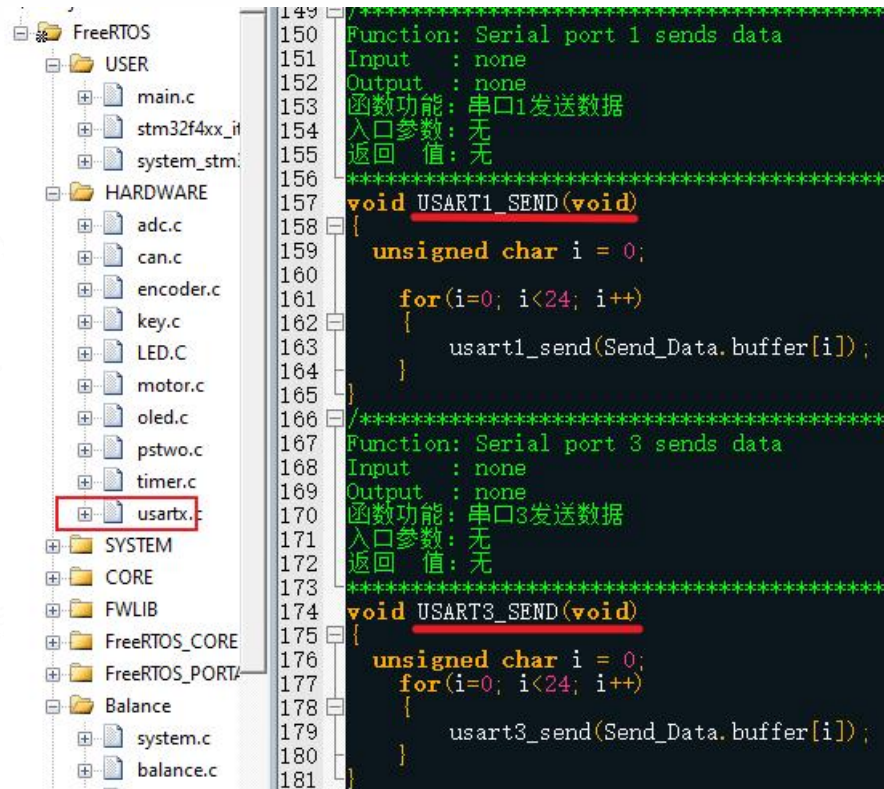


图 4-2 串口发送函数位置图

4.2 发送频率的设置

串口发送数据的频率是由数据发送任务的执行频率决定的，若需修改串口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。

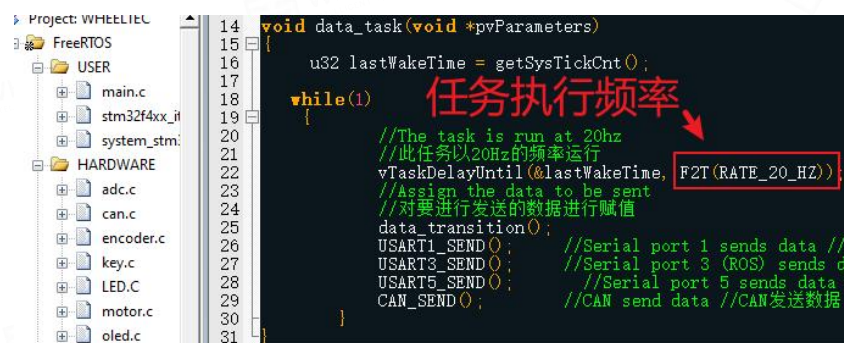


图 4-3 串口发送频率修改位置

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，为程序支持的串口发送频率。

```
#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 串口发送前数据处理

出于兼容性等多方面的考虑，我司在串口通信中，数据以字节（byte）为单位进行传输的，所以部分数据需要处理为多个 8 位数据进行存储与发送。

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 串口接收数据

底盘接收到的数据处理位于程序的串口中断,串口 1 跟串口 3 的接收中断处理逻辑是一样的,下文将采用串口 3 的代码进行逻辑讲解。

如图 4-8 所示,为串口 1 接收中断位置



图 4-8 串口 1 接收中断位置

如图 4-9 所示,为串口 3 接收中断位置



图 4-9 串口 3 接收中断位置

① 串口数据接收逻辑

如图 4-10 所示,为串口接收数据的处理逻辑,主要分为五个步骤。

```

Receive_Data.buffer[Count]=Usart_Receive;
// Ensure that the first data in the array is FRAME_HEADER
// 确保数组第一个数据为FRAME_HEADER
if(Usart_Receive == FRAME_HEADER||Count>0)
    Count++;
else
    Count=0;
if (Count == 11)
{
    Count=0; //Prepare for the serial port data to be refill into the array //为串口数据重新填入数组做准备
    if(Receive_Data.buffer[10] == FRAME_TAIL)
    {
        //Data exclusionary or bit check calculation, mode 0 is sent data check
        //数据异或位校验计算, 模式0是发送数据校验
        if(Receive_Data.buffer[9] ==Check_Sum(9, 0))
    }

```

①接收到每一字节的数据都写入接收数组
 ②接收到帧头后开始将后续数据依次填入接收缓冲区
 ③判断接收缓冲区里数据包的长度
 ④验证数据包最后一位是否为帧尾
 ⑤进行异或校验, 检查数据帧的数据是否正确

图 4-10 串口接收数据处理逻辑

② 阿克曼车型 Z 轴速度处理

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。如图 4-11 所示, 为阿克曼车型的 Z 轴速度处理代码的位置。

```

if(Car_Mode==Akm_Car)
{
    Move_Z=Vz_to_Akm_Angle(Move_X, Vz);
}
else
{
    Move_Z=XYZ_Target_Speed_transition(Receive_Data.buffer[7],Receive_Data.buffer[8]);
}

```

阿克曼车型
 其他车型

图 4-11 阿克曼车型 z 轴特殊处理