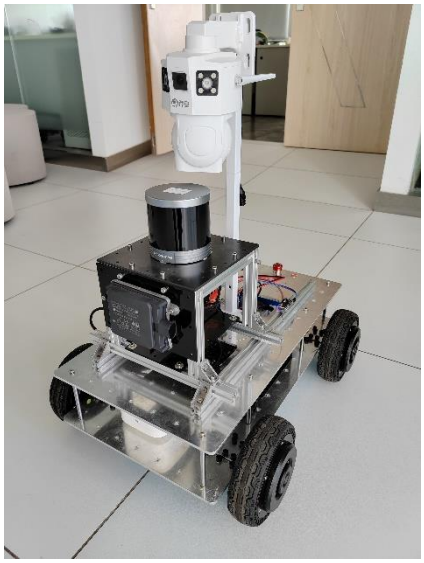
1 简介

该小车是一款简易的 4 轮差速小车，通过 4 个轮毂电机实现小车差速运动，其性能参数如表 1 所示。小车所有的 3D 图纸、PCB 文件、嵌入式端控制代码全部开源，开源地址：https://github.com/RuPingCen/mick_robot_chassis。

表 1 MickX4 小车性能参数

名称	指标
尺寸	600 * 480 * 265 mm
左右轮轴距	400 mm
前后轮轴距	400 mm
轮胎直径	200 mm（8 寸）
最小转弯半径	0
小车最大速度	2.2 m/s (250 R/min)
电机额定扭矩	7 N.m
电机峰值扭矩	18 N.m
负载能力	≤40Kg
续航里程	≥15KM
充电时长	约 4-5 小时



2 小车使用说明

2.1小车面板说明

下图所示，小车车尾面板第一排接口从左到右依次为电量显示模块、急停按钮、三色指示灯、电源按钮，面板第二排接口从左到右依次为电池充电口、外部供电接口（24V/10A）、232 串口、以太网调试口。面板对外提供 24V/10A 的电源用作传感器供电，采用 GX16-2 芯航空插头（**1 号引脚为+VCC，2 号引脚为 GND 注意面板放电口输出电压为 22.4-29.4V**）。

（1）小车状态指示灯

小车面板状态指示灯正常情况为绿灯闪烁，部分异常情况如下：

1、若呈现绿灯闪烁同时红灯常亮则代表急停按钮未松开；

2、若在开机的时候出现黄色灯单独快闪则代表小车未收到遥控器信号；

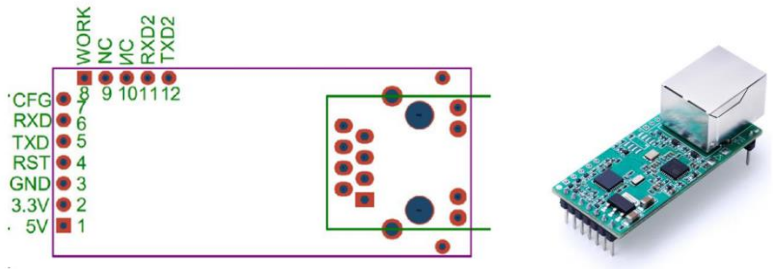
3、若在小车开机时刻出现红灯慢闪 4 次则表示控制板传感器故障（IMU

未初始化）。

（2）以太网调试口

小车面板上以太网调试口 IP 地址固定为“192.168.0.7”，将网线连接面板调试口与电脑，配置电脑 IP 地址为“192.168.0.201”。利用 github 仓库目录下“Reference_Documents/02_调试工具/网络_串口调试工具.exe”工具，将调试工具默认端口设置为 8234，串口波特率为 256000，工作模式设置为 TCP 服务端，即可打印小车调试信息。

小车内部是通过外接串口转以太网模块实现以太网通讯的。再对小车维修时，若设置模块后忘记参数及 IP 地址。可打开小车外壳，找到如下图所示的串口转以太网模块，通过短接 GND 与模块上的复位引脚 RST(引脚编号为 4)进行复位（短接时间≥200ms），网络复位以后网络模块 ip 默认为 192.168.0.7，通讯波特率为 115200。模块引脚如下图所示。



（3）小车底盘充电

对小车电池充电请使用配套的充电器进行充电！充电时将充电器圆孔插头插入小车面板充电口，充电器功率约为 180W，充电器输出电压为 24-29.4V,充电时长约为 4-5 小时。

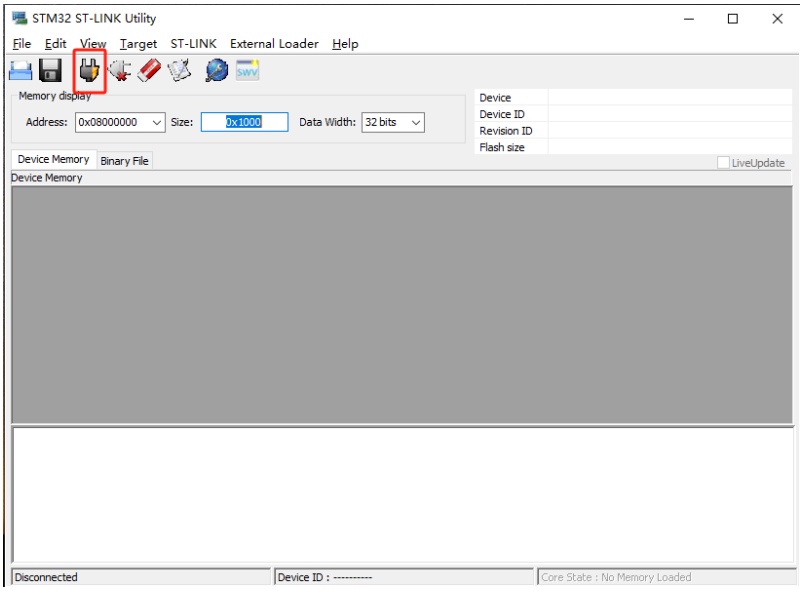
小车充电器背面与正面

2.2 小车固件更新

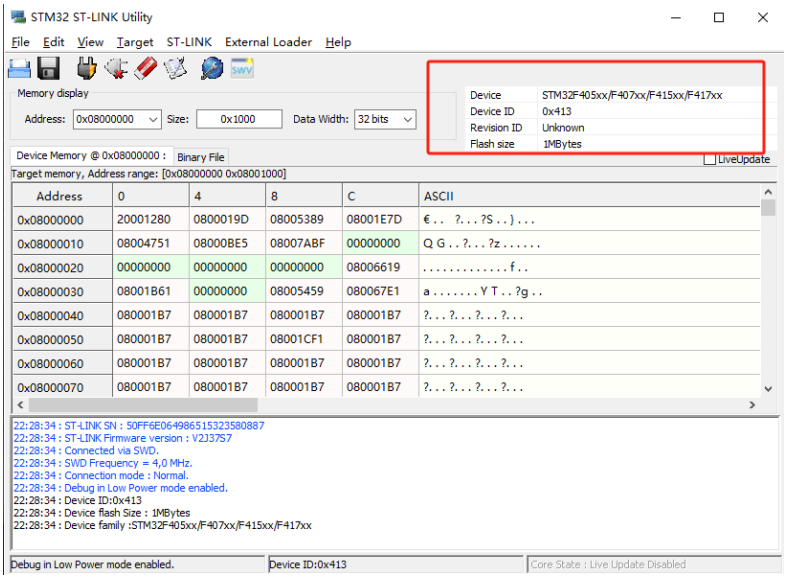
固件可以通过源代码或者 ST-link 工具进行更新，源代码更新方式直接使用 keil 工具编译源代码下载即可。这里主要介绍通过 ST-link 工具下载编译好的 Hex 文件。

从 ST 官方或者本项目的 github 代码仓库目录（mick_robot_chasiss\Reference_Documents\01_固件下载）下载烧写软件（STM32 ST-LINK Utility v4.6.0）。STM32 ST-LINK utility 安装时候一直下一步默认即可。完成安装以后在桌面上找到如下图标：

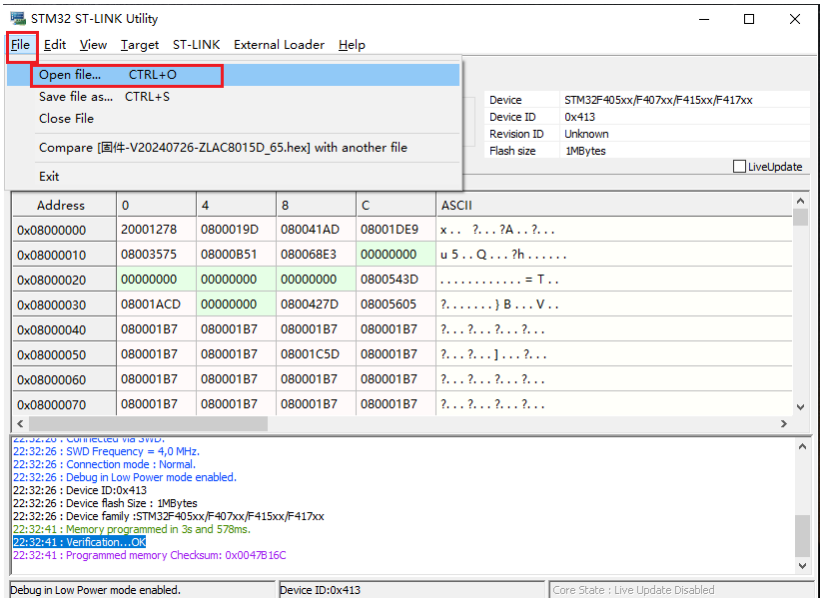
打开软件，使用 STLink 连接到 STM32 板子。接下来点击这个插头一样的东西（如果没有识别到就拔插一下 STLink 下载器）



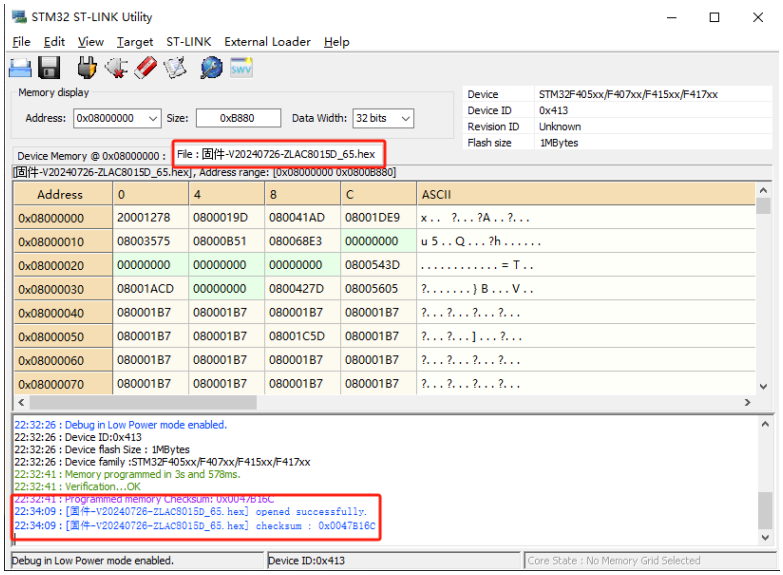
成功以后就可以看到芯片的信息



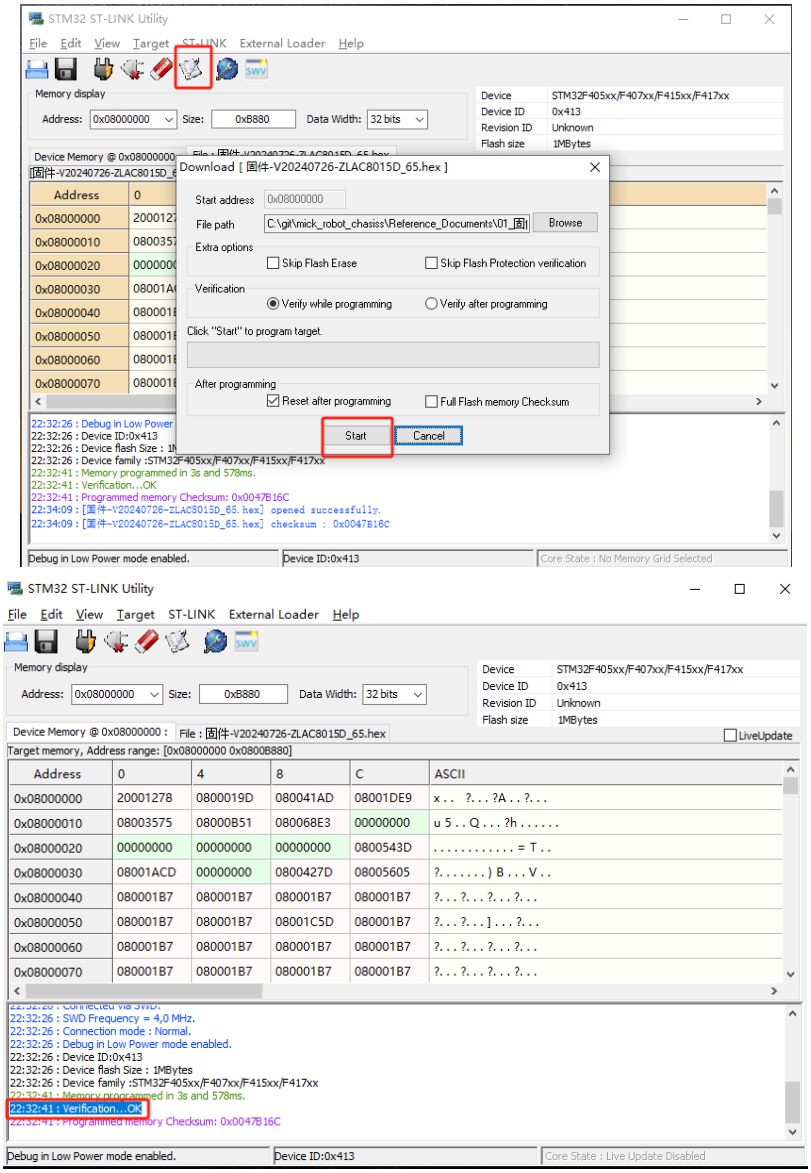
小车的固件在 github 仓库 Hex 目录下，接下来选中需要下载的 hex 文件 MickRobot_Chassis_APS80_Vxxx-xxxxx.hex (黑色小车使用的 8 寸轮胎，这里选中包含“APS80”字段的 hex 固件)



成功打开以后这两个地方会有信息提示成功打开文件



接下来点击烧写程序



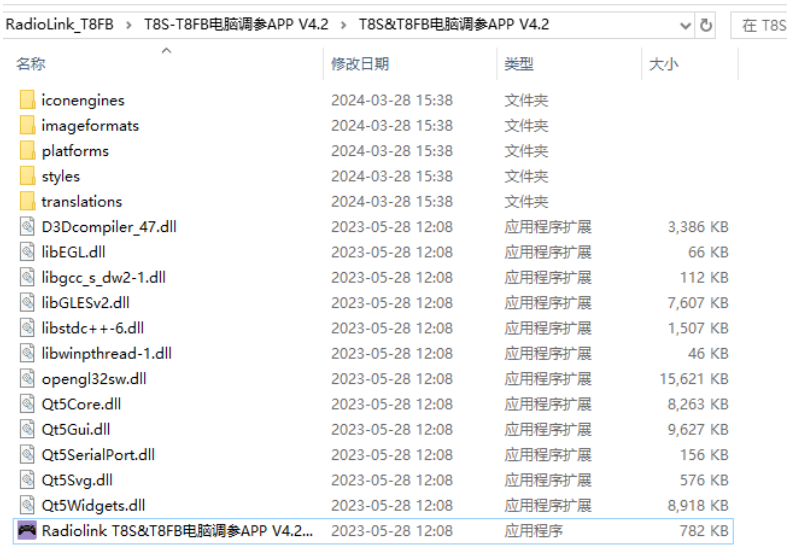
2.3 遥控器操控说明

四轮差速小车的控制与 mick 开源项目中其他小车的操做方法一致。默认采用乐迪 T8FB 遥控器（SBUS 协议）

2.3.1 遥控器配置方法

购买整车时遥控器默认已经配置好，不需要再重新配置。

当自行组装车辆或者更换遥控器时，需利用安卓充电线（micro usb）将遥控器与电脑连接，使用（"Reference\RadioLink_T8FB\T8S-T8FB 电脑调参 APP V4.2\T8S&T8FB 电脑调参 APP V4.2"）遥控器自带的 APP 进行配置。



将第 1 通道（右手摇杆竖直方向通道）反向，保证摇杆拉到最下方的时候遥控器输出最小值，最上方输出最大值。同时，对于水平方向通道而言，拉到最左边输出最小值，最右方向输出最大值，下图是修改后的状态。

2.3.2 遥控器操作说明

这里使用的是左手油门的 RadioLink T8FB 遥控器。

1. 遥控器**左上角的拨动开关**为功能选择按键：置于 L 档，即最上方位置，表示开启自动驾驶（遥控器操作无效），置于 H 档表示由遥控器控制小车，忽略上位机命令。
2. 遥控器**右上角拨动开关**上中下位置分别对应小车 1m/s、2m/s、2.2m/s 速度。

3. 左手边摇杆竖直方向通道（ch3）控制小车前后运动（如下图所示），右边的摇杆水平通道控制小车左右旋转。

2.3.3 遥控器充电

遥控器电池电压低的时候，会发出“滴滴~”的声响，此时打开后盖将电池取出。**（注意电池插头的正负方向，对应遥控器背后”+“，”—“标识）**

使用配套充电器充电，电池充满以后充电器的灯由红色变为绿色。遥控器充电时需使用遥控器对应的充电器，遥控器电池为 2s 电池，电池电压范围（6.4V-8.4V）。

2.4 小车 ROS 接口说明

2.4.1 ROS1 接口

小车 ROS1 节点

https://github.com/RuPingCen/mick_robot_chasiss/tree/master/ROS_Node

该目录中包含两个文件夹：**mick_bringup** 和 **mick_description**，**mick_bringup** 为小车对应的 ROS 驱动节点，**mick_description** 为 urdf 模型。

step1：将 **ROS_Node** 放置于 ROS 工作空间下进行编译

```
cp -r    ROS_Node ~/catkin_ws/src        #拷贝文件到 ROS 工作空间
cd ~/catkin_ws/src
catkin_make
```

step2：将小车 USB 串口连接到电脑，并赋予串口权限

```
sudo chmod 777 /dev/ttyUSB*
```

step3：启动 ROS 节点

```
roslaunch mick_bringup mickrobot-v3.launch
```

小车节点启动以后可以通过 rostopic list 命令查看到该 ROS 节点会对外发布如下 Topic

```
crp@crp-CW65S:~$ rostopic list
/mickrobot/chassis/Imu
/mickrobot/chassis/cmd_vel
/mickrobot/chassis/odom
/mickrobot/chassis/odom/path
/mickrobot/rc_remotes/joy
/rosout
/rosout_agg
crp@crp-CW65S:~$
```

其中：

- /mickrobot/chassis/Imu 对外发布小车自身 IMU 测量数据
- /mickrobot/chassis/odom 小车里程计数据
- /mickrobot/chassis/odom/path 小车里程计数据对应的路径（默认不发布）
- /mickrobot/rc_remotes/joy 遥控器数据（默认不发布）
- /mickrobot/chassis/cmd_vel 小车控制命令接收话题

step4: 新建终端，通过 ROS 话题向 **/mickrobot/chassis/cmd_vel** 话题发布数据控制小车移动（**注意 将遥控器左上角拨码开关拨到最上，表示开启自动驾驶模式**）

```
rostopic pub /mickrobot/chassis/cmd_vel -r 10 geometry_msgs/Twist
"linear:
  x: 0.0
    y: 0.0
    z: 0.0
angular:
    x: 0.0
    y: 0.0
    z: -0.1"
```

角速度方向： 逆时针为正，速度方向：车头方向为 x 方向。

2.4.2 ROS2 接口

在系统安装串口库 serial：

```
git clone https://github.com/ZhaoXiangBox/serial
cd serial && mkdir build
cmake .. && make
sudo make install
```

将 ROS 的驱动板拷贝到工作空间下，进入工作空间主目录:

```
cd bringup_ws
```

使用 colcon build 进行编译:

```
colcon build
```

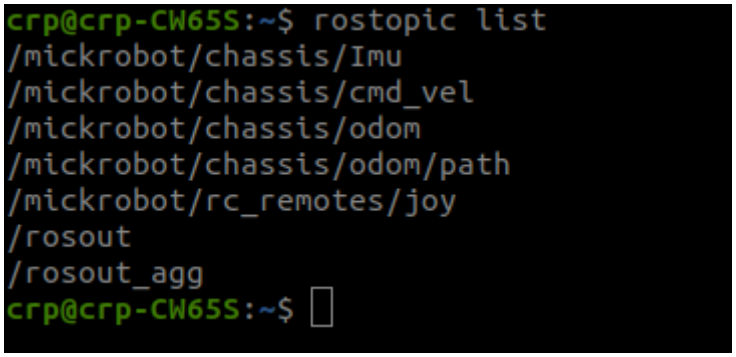
在运行之前，由于节点里面涉及到串口的打开，因此需要先打开串口权限:

```
sudo chmod 777 /dev/ttyUSB0
```

最后在编译无报错之后就可以运行了:

```
sourse install/setup.bash
ros2 run mick_bringup mick_node
```

ROS2 输出的话题名称及消息定义均与 ROS1 接口一致。



注：可以将小车垫高，随后在命令行终端里输入以下指令：

```
ros2 topic pub -r 100 /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 1.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

2.5 小车底盘通讯协议

小车底盘与 PC 机通过串口连接，为增强底盘的通用性，若需要不依赖 ROS 中间件，可以向串口设备按照以下约定的协议发送数据，也可实现小车控制。底盘的通讯接口可参考在线文档：
https://docs.qq.com/sheet/DV2hmSEdSYVVtclB4?tab=bb08j2

2.5.1 命令下发接口

如表 2-1 所示差速底盘速度控制模式下，指令发送的格式为

表 2-1 MickX4 差速小车速度控制指令

帧头		帧长度	命令字	数据					校验位	帧尾	
Byte0	Byte1	Byte2	Byte3	Byte[4:5]	Byte[6:7]	Byte[8:9]	Byte10	Byte11	Byte12	Byte13	Byte14
0XAE	0xEA	1 Byte	0xF3	**X 方向速度（0.01 m/s）**	**Y 方向速度 0x00**	**旋转角速度 (0.01 rad/s)**	保留字（0x00）	保留字（0x00）	数据校验位	0XEF	0xFE

1) **Byte[0:1]**：uint8 类型数据，表示数据帧头，固定值 0XAE、0xEA。
2) **Byte[2]**：uint8 类型数据，表示数据帧长度，数据长度=自身长度（1 个字节）+命令类型（1 个字节）+数据（N 个字节）+校验位（1 个字节）（即除帧头帧尾以外的所有数据位）
3) **Byte[3]**：uint8 类型数据，控制命令 0xF3 表示针对差速底盘下发速度指令控制
4) **Byte[4:5]** : uint16 类型数据，高位在前，X 表示小车底盘 x 方向（车头方向）速度，在发送的时候需要进行手动偏移，偏移量为 10m/s,同时将待发送的速度量 speed_x 放大 100 倍，将其转为正数进行发送。
5) **Byte[6:7]** :uint16 类型数据，表示小车底盘 y 方向（车头方向）速度，差速底盘上设置无效，默认为 0；
6) **Byte[7:8]** :uint16 类型数据，表示小车底盘旋转角速度，单位 0.01rad/s ，偏移量为 10 rad/s；
7) **Byte[10:11]** 两个字节为保留字，固定为 0x00;
8) **Byte[12]**：数据校验位，数据校验位=数据长度+命令类型+所有的数据位（取结果的低 8 位）；
9) **Byte[13:14]**：表示数据帧尾，固定值 0XEF、0xFE;

表 2-2 列举其他控制器向底盘发送控制指令的样例函数

表 2-2 MickX4 差速小车速度控制指令发送函数示例

```
void send_speed_to_X4chassis(float x,float y,float w)
{
    uint8_t data_tem[50];
    unsigned int speed_0ffset=10; //速度偏移值  10m/s，把速度转换成正数发送
    unsigned char i,counter=0;
    unsigned int check=0;

    data_tem[counter++] =0xAE;
    data_tem[counter++] =0xEA;
    data_tem[counter++] =0x0B;
    data_tem[counter++] = 0xF3; //针对 MickX4 的小车使用 F3  字段
    data_tem[counter++] =((x+speed_0ffset)*100)/256; // X
    data_tem[counter++] =((x+speed_0ffset)*100);
    data_tem[counter++] =((y+speed_0ffset)*100)/256; // Y
    data_tem[counter++] =((y+speed_0ffset)*100);
    data_tem[counter++] =((w+speed_0ffset)*100)/256; // X
    data_tem[counter++] =((w+speed_0ffset)*100);
    data_tem[counter++] =0x00; //保留字
    data_tem[counter++] =0x00;
    data_tem[2] =counter-2;
    for(i=2;i<counter;i++) //  计算校验值
    {
        check+=data_tem[i];
    }
    data_tem[counter++] =0xEF;
    data_tem[counter++] =0xFE;
    ros_ser.write(data_tem,counter);
```

```
}
```

数据下发示例：假设下发的速度 x=1m/s,则下发的实际值=(1m/s+10)*100 = 1100,对应 16 进制为 0x044C；y=0 m/s,则下发的实际值=(0 m/s+10)*100 = 1000,对应 16 进制为 0x03E8;角速度 w=0 rad/s,则下发的实际值=(0 rad/s+10)*100 = 1000。对应 16 进制为 0x03E8;发送数据时采用大端模式，即高位在前。

AE EA 0B F3 **04 4C** 03 E8 03 E8 00 00 39 EF FE

2.5.2 数据上传协议

2.5.2.1 里程计数据上传

底盘向上位机发送数据的命令字为 0xA7，其中上传的里程计数据包含小车车体坐标系下，X 方向速度（0.001m/s），Y 方向速度（差速底盘固定为 0，0.001m/s），绕小车旋转的角速度（逆时针为正 0.001rad/s）

表 2-3 MickX4 差速小车里程计数据上传指令

帧头		帧长度	命令字	数据			校验位 Byte[12]	帧尾 Byte[13:14]	
Byte[0]	Byte[1]	Byte[2]	Byte[3]	Byte[4:5]	Byte[6:7]	Byte[8:9]	Byte[12]	Byte[13]	Byte14]
0XAE	0xEA	1 Byte	0xA7	**X 方向速度（0.001 m/s）**	**Y 方向速度 0x00**	**旋转角速度 (0.001 rad/s)**	数据校验位	0XEF	0xFE

1) **Byte[0:1]**：uint8 类型数据，表示数据帧头，固定值 0XAE、0xEA。
2) **Byte[2]**：uint8 类型数据，表示数据帧长度，数据长度=自身长度（1 个字节）+命令类型（1 个字节）+数据（N 个字节）+校验位（1 个字节）（即除帧头帧尾以外的所有数据位）
3) **Byte[3]**：uint8 类型数据，控制命令 0xF3 表示针对差速底盘下发速度指令控制
4) **Byte[4:5]** ：int16 类型数据，高位在前，X 表示小车底盘 x 方向（车头方向）速度,单位为 0.001m/s。
5) **Byte[6:7]** :int16 类型数据，表示小车底盘 y 方向（车头方向）速度，差速底盘上设置无效，默认为 0；
6) **Byte[7:8]** :int16 类型数据，表示小车底盘旋转角速度，单位 0.001rad/s；
7) **Byte[9]**：数据校验位，数据校验位=数据长度+命令类型+所有的数据位（取结果的低 8 位）；
8) **Byte[10]**：表示数据帧尾，固定值 0XEF、0xFE;

表 2-4 MickX4 差速小车里程计上传函数

```
void Chassis_Odom_Upload_Message(float odom_vx, float odom_vy, float odom_w)
{
        unsigned char senddata[25];
        unsigned char i=0,j=0;
        unsigned int sum=0x00;
        int16_t tem =0;

        senddata[i++]=0xAE;
        senddata[i++]=0xEA;
        senddata[i++]=0x01;//数据长度在后面赋值
        senddata[i++]=0xA7; //命令字

        tem = (int16_t)((odom_vx*1000)/1); //odom_vx 最大值 65.535
        senddata[i++] = tem>>8;
        senddata[i++] = tem;
        tem = (int16_t)((odom_vy*1000)/1);
        senddata[i++] = tem>>8;
        senddata[i++] = tem;
        tem = (int16_t)((odom_w*1000)/1);
        senddata[i++] = tem>>8;
        senddata[i++] = tem;

        senddata[2]=i-1; //数据长度
        for(j=2;j<i;j++)
                sum+=senddata[j];

        senddata[i++]=sum;
        senddata[i++]=0xEF;
        senddata[i++]=0xFE;

        UART_send_buffer(USART1,senddata,i);
}
```

其中：odom_vx, odom_vy, odom_w 分别表示小车车体坐标系下 x 方向速度，y 方向速度以及旋转角速度。

例如，odom_vx=2.1 m/s, odom_vy=0, odom_w=1.6 rad/s。首先将 odom_vx 放大 1000 倍，将其单位转化为 0.001m/s;同理将 odom_w 放大 1000 倍，将其单位转化为 0.001rad/s。此时，odom_vx 对应十进制数 2100，转为为十六进制为 0x834，odom_w 对应十进制数为 1600，十六进制数为 0x640。此时发送的数据为：
AE EA 09 A7 08 34 00 00 06 40 32 EF FE

2.5.2.2 IMU 数据上传协议

小车控制板安装有型号为 MPU6050 的 6 轴惯性测量单元，对外提供控制板自身的加速度和陀螺仪测量值。

表 2-3 MickX4 差速小车板载 IMU 数据上传指令

帧头		帧长度	命令字	IMU 数据				校验位	帧尾	
Byte[0]	Byte[1]	Byte[2]	Byte[3]	Byte[4:9]	Byte[10:15]	Byte[16:21]	Byte[22:27]	Byte[28]	Byte[29]	Byte[30]

0XAE	0xEA	1 Byte	0xA0	**x,y,z 轴加速度（原始数据）**	**x,y,z 轴角速度（原始数据）**	**x,y,z 轴磁力计（原始数据、保留位）**	**Roll-Pitch-Yaw 欧拉角（0.01 度）**	校验位	0XEF	0xFE

1) **Byte[0:1]**：uint8 类型数据，表示数据帧头，固定值 0XAE、0xEA。
2) **Byte[2]**：uint8 类型数据，表示数据帧长度，数据长度=自身长度（1 个字节）+命令类型（1 个字节）+数据（N 个字节）+校验位（1 个字节）（即除帧头帧尾以外的所有数据位）
3) **Byte[3]**：uint8 类型数据，控制命令 0xA0 表示底盘向 PC 端发送 IMU 测量数据帧
4) **Byte[4:5]:** int16 类型，高位在前，x 轴方向加速度原始输出数据；
5) **Byte[6:7]:** int16 类型，高位在前，y 轴方向加速度原始输出数据；
6) **Byte[8:9]:** int16 类型，高位在前，z 轴方向加速度原始输出数据；
7) **Byte[10:11]:** int16 类型，高位在前，绕 x 轴旋转角速度原始输出数据；
8) **Byte[12:13]:** int16 类型，高位在前，绕 y 轴旋转角速度原始输出数据；
9) **Byte[14:15]:** int16 类型，高位在前，绕 z 轴旋转角速度原始输出数据；
10) **Byte[16:17]:** int16 类型，高位在前，x 轴磁场原始输出数据；(保留)
11) **Byte[18:19]:** int16 类型，高位在前，y 轴磁场原始输出数据；(保留)
12) **Byte[20:21]:** int16 类型，高位在前，z 轴磁场原始输出数据；(保留)
13) **Byte[22:23]:** int16 类型，高位在前，欧拉角，Roll 角，单位 0.01 度；
14) **Byte[24:25]:** int16 类型，高位在前，欧拉角，Pitch 角，单位 0.01 度；
15) **Byte[26:27]:** int16 类型，高位在前，欧拉角，Yaw 角，单位 0.01 度；
16) **Byte[28]**：数据校验位，数据校验位=数据长度+命令类型+所有的数据位（取结果的低 8 位）；
17) **Byte[29:30]**：表示数据帧尾，固定值 0XEF、0xFE;

2.5.2.3 遥控器数据协议

小车遥控器数据上传使用命令字 0xA3。

表 2-3 MickX4 差速小车遥控器数据上传指令

帧头		**帧长度**	**命令字**	**IMU 数据**					**校验位**	**帧尾**	
Byte[0]	Byte[1]	Byte[2]	Byte[3]	Byte[4:11]	Byte[12:15]	Byte[16]	Byte[17]	Byte[18]	Byte[19]	Byte[20]	Byte[21]
0XAE	0xEA	1 Byte	0xA3	**ch1-ch4 (每个通道占用 2 字节)**	**SW1-SW4 (每个通道占用 1 字节)**	**Type 遥控器类型**	**错误码**	**保留字**	校验位	0XEF	0xFE

1) **Byte[0:1]**：uint8 类型数据，表示数据帧头，固定值 0XAE、0xEA。

2) **Byte[2]**：uint8 类型数据，表示数据帧长度，数据长度=自身长度（1 个字节）+命令类型（1 个字节）+数据（N 个字节）+校验位（1 个字节）（即除帧头帧尾以外的所有数据位）；
3) **Byte[3]**：uint8 类型数据，控制命令 0xA3 表示底盘向 PC 端发送遥控器测量数据帧；
4) **Byte[4:5]:** int16 类型，高位在前，遥控器 Ch1 摇杆通道原始输出数据；
5) **Byte[6:7]:** int16 类型，高位在前，遥控器 Ch2 摇杆通道原始输出数据；
6) **Byte[8:9]:** int16 类型，高位在前，遥控器 Ch3 摇杆通道原始输出数据；
7) **Byte[10:11]:** int16 类型，高位在前，遥控器 Ch4 摇杆通道原始输出数据；
8) **Byte[12]:** uint8 类型，高位在前，遥控器 SW1 拨动开关原始输出数据；
9) **Byte[13]:** uint8 类型，高位在前，遥控器 SW2 拨动开关原始输出数据；
10) **Byte[14]:** uint8 类型，高位在前，遥控器 SW3 拨动开关原始输出数据；
11) **Byte[15]:** uint8 类型，高位在前，遥控器 SW4 拨动开关原始输出数据；
12) **Byte[15]:** uint8 类型， Type-遥控器类型（1:DJI-DBUS 2:SBUS）；
13) **Byte[16]:** uint8 类型，错误码((0x00 表示在正常，0xA1 通讯丢失);
14) **Byte[17]:** uint8 类型，保留字(0x00)
15) **Byte[18]**：数据校验位，数据校验位=数据长度+命令类型+所有的数据位（取结果的低 8 位）；
16) **Byte[20:21]**：表示数据帧尾，固定值 0XEF、0xFE;

2.5.2.4 GPIO 状态上传

小车控制板输入/输出端口状态数据上传使用命令字 0xAC。

表 2-3 MickX4 差速小车输入输出端口状态数据上传指令

帧头		帧长度	命令字	端口数据		校验位	帧尾	
Byte[0]	Byte[1]	Byte[2]	Byte[3]	Byte[4]	Byte[5]	Byte[6]	Byte[7]	Byte[8]
0XAE	0xEA	1 Byte	0xAC	**输入端口 (占用 1 字节)**	**输出端口 (占用 1 字节)**	校验位	0XEF	0xFE

1) **Byte[0:1]**：uint8 类型数据，表示数据帧头，固定值 0XAE、0xEA。
2) **Byte[2]**：uint8 类型数据，表示数据帧长度，数据长度=自身长度（1 个字节）+命令类型（1 个字节）+数据（N 个字节）+校验位（1 个字节）（即除帧头帧尾以外的所有数据位）；
3) **Byte[3]**：uint8 类型数据，控制命令 0xAC 表示底盘向 PC 端发送遥控器测量数据帧；

4) **Byte[4]:** uint8 类型，输入端口；每一个 bit 位对应一个通道，例如 bit0 表示通道 0 的状态。每一个 bit 位上数据 0 表示端口电平为 0 电平，数据 1 表示端口电平是 1 电平。
5) **Byte[5]:** uint8 类型，输输出端口；每一个 bit 位对应一个通道，例如 bit0 表示通道 0 的状态。每一个 bit 位上数据 0 表示端口电平为 0 电平，数据 1 表示端口电平是 1 电平。
6) **Byte[6]**：数据校验位，数据校验位=数据长度+命令类型+所有的数据位（取结果的低 8 位）；
7) **Byte[7:8]**：表示数据帧尾，固定值 0XEF、0xFE;

2.6 扩展支架尺寸

小车上面版有两个 20 型材可做为外部固定支架，型材中心间距为 213mm, 车头方向有安装孔阵列作为备用，安装孔从中心线出发间隔 45mm 向两侧扩展，以间距 50mm 沿纵向方向扩展。