R2 底盘串口通信协议 v2

下位机（STM32）接收模块实现说明 —— 上位机到 下位机 单向速度指令

项目	内容
协议版本	v2（定长 10 字节，只发速度指令）
上位机实现	ROS 2 包 techx_r2_chassis_control，节点 chassis_serial_node
通信方向	上位机 → 下位机，单向。下位机无需回包，协议不含任何上行帧
文档用途	供下位机开发者实现接收 / 校验 / 解析模块；帧示例与 CRC 由上位机代码现算，可直接作测试向量

> 一句话总结：上位机每 33 ms 发一帧 10 字节数据，帧内就是 vx、vy、omega 三个 int16 小端数值加一个 CRC16。下位机要做的事只有三件——按 10 字节切帧、校验 CRC、把三个数取出来驱动电机；另外建议加一条「超时未收到合法帧就停车」的兜底逻辑。

1. 概述

上位机运行 ROS 2 导航 / 遥控栈，把最终速度指令发布在 `/cmd_vel`（geometry_msgs/Twist）话题上。本节点订阅该话题，取出线速度 x、线速度 y、角速度 z 三个分量，换算成整数后按本协议打包，经串口写入下位机。整条链路是单向的：上位机只发，从不读串口，因此协议里没有应答、没有序号、没有回读状态。

下位机侧要实现的收帧逻辑，本质是一个「固定长度 + 帧头 + CRC」的字节流解析器。需要注意的是：帧头 AA 55 会偶然出现在数据区里（速度值恰好为 0x55AA 或 0xAA55 时），所以不能靠搜索帧头来切帧——必须以「10 字节固定帧 + CRC 校验通过」作为唯一的帧确认条件。这一点在第 6 节展开。

2. 物理层参数

参数	取值	说明
波特率	115200 bps	可通过上位机参数 baudrate 修改，下位机需保持一致
数据位 / 校验 / 停止位	8 / None / 1	即 8N1，无奇偶校验
流控	无	软件与硬件流控都不使用
电平	按硬件设计	上位机侧通过 USB-TTL 适配器转为串口设备节点（默认 /dev/r2_chassis）
帧率	默认 30 Hz	即约 33 ms 一帧；上位机参数 velocity_rate_hz 可调
链路方向	上位机 → 下位机	下位机发回的任何字节上位机都不会读取，请勿依赖上行通道

3. 帧结构（定长 10 字节）

所有多字节数值均为小端（little-endian，低字节在前）。

偏移	长度	字段	类型	单位	说明
0	2	header	uint8[2]	—	固定 0xAA 0x55

偏移	长度	字段	类型	单位	说明
2	2	vx	int16 LE	mm/s	机体坐标系前后方向，前进为正
4	2	vy	int16 LE	mm/s	机体坐标系左右方向，左移为正
6	2	omega	int16 LE	mrad/s	绕 Z 轴角速度，逆时针（左转）为正
8	2	crc16	uint16 LE	—	CRC16-MODBUS，覆盖 byte0 ~ byte7



图 1 帧字节布局（灰色 = 帧头，蓝色 = 速度数据区，橙色 = 校验）

4. 字段语义与数值换算

三个字段都是 int16（-32768 ~ 32767），单位分别为 mm/s、mm/s、mrad/s，即「米每秒 × 1000」和「弧度每秒 × 1000」，全部靠 1000 这一个比例常数换算，下位机侧反算则是除以 1000。符号约定与 geometry_msgs/Twist 一致。

字段	上位机换算（→ 下发值）	下位机反算	实际上限	int16 满量程
vx	`mm/s = round(vx_m_s × 1000)`	`vx_m_s = vx / 1000.0`	±2500	±32767
vy	`mm/s = round(vy_m_s × 1000)`	`vy_m_s = vy / 1000.0`	±2500	±32767
omega	`mrad/s = round(wz_rad_s × 1000)`	`wz_rad_s = omega / 1000.0`	±1500	±32767

实际上限指上位机下发前的限幅：vx/vy 为 ±2500（±2.5 m/s），omega 为 ±1500（±1.5 rad/s），下位机收到的数值不会超出该范围；仍建议下位机做一次范围钳制作为防御性处理。

5. CRC16-MODBUS 校验

校验算法为标准 CRC16-MODBUS：多项式 0x8005（反射形式 0xA001），初值 0xFFFF，输入输出都不做取反，也没有最终异或。计算范围是 byte0 ~ byte7（帧头 + 数据），算出的 16 位结果以小端写入 byte8、byte9（低字节在前）。

5.1 接收端校验要点

- 先按小端把 byte8、byte9 拼成期望值：`expect = byte8 | (byte9 << 8)`
- 再对前 8 字节算 CRC，与 expect 比较，不相等即丢帧
- 多项式、初值、位序三者任一不同都会和上位机对不上，请直接使用下面的参考实现

5.2 C 参考实现（可直接抄进 STM32）

```
/* CRC16-MODBUS: poly=0xA001, init=0xFFFF, no reflect in/out, no final xor */
uint16_t crc16_modbus(const uint8_t *buf, uint16_t len)
{
    uint16_t crc = 0xFFFF;
    for (uint16_t i = 0; i < len; i++) {
        crc ^= (uint16_t)buf[i];
        for (int b = 0; b < 8; b++) {
            if (crc & 0x0001) crc = (crc >> 1) ^ 0xA001;
            else              crc = (crc >> 1);
            crc &= 0xFFFF;
        }
    }
    return crc;
}

/* usage: after a full 10-byte frame f[] has arrived */
uint16_t expect = (uint16_t)f[8] | ((uint16_t)f[9] << 8);
if (crc16_modbus(f, 8) != expect) { /* drop this frame */ }
```

6. 接收端实现建议

> 关键坑：帧头不可作为切帧依据。数据区里的 vx / vy / omega 是任意 int16，完全可能等于 0x55AA 或 0xAA55，从而在字节流里造出一个假的 **AA 55**。正确做法是固定 10 字节切帧 + CRC 判定：只有 10 字节对齐且 CRC 通过的帧才采信。若 CRC 失败，不要整帧丢弃后从下一帧开始，而应逐个字节滑动重同步（丢 1 字节再试），这样偶发干扰后最多丢一帧就能重新对齐。

6.1 推荐收帧流程

1. 串口用 DMA + 空闲中断（IDLE）收数据，或中断驱动填环形缓冲，按 10 字节一帧取出；
2. 检查字节 0、1 是否为 0xAA 0x55。不是则丢弃 1 字节，回到上一步；
3. 对前 8 字节算 CRC16，与 byte8 / byte9（小端拼接）比对。失败则丢弃 1 字节并重新对齐；
4. 校验通过：按小端解析 vx、vy、omega，并刷新「最后一次有效帧到达时间」；
5. 把三个值写入电机控制环节（可按需再做死区与斜率限制）；
6. 另设一个定时器：若距上次有效帧超过 300 ~ 500 ms，则强制 vx = vy = omega = 0 并置故障标志。

```
/* 10-byte fixed frame + CRC resync */
static uint8_t  buf[10];
static uint16_t len = 0;
static uint32_t last_ok_ms = 0;

void on_rx_byte(uint8_t b)
{
    buf[len++] = b;
    if (len < 10) return;

    if (buf[0] == 0xAA && buf[1] == 0x55
        && crc16_modbus(buf, 8) == ((uint16_t)buf[8] | ((uint16_t)buf[9] << 8))) {
        int16_t vx    = (int16_t)((uint16_t)buf[2] | ((uint16_t)buf[3] << 8));
        int16_t vy    = (int16_t)((uint16_t)buf[4] | ((uint16_t)buf[5] << 8));
        int16_t omega = (int16_t)((uint16_t)buf[6] | ((uint16_t)buf[7] << 8));
        apply_velocity(vx, vy, omega);   /* units: mm/s, mm/s, mrad/s */
        last_ok_ms = now_ms();
        len = 0;                         /* whole frame consumed */
    } else {
        memmove(buf, buf + 1, 9);        /* shift by 1 byte and retry */
        len = 9;
    }
}

/* watchdog: call from a 100 Hz ~ 1 kHz task */
void safety_tick(void)
{
    if (now_ms() - last_ok_ms > 400) apply_velocity(0, 0, 0);
}
```

6.2 超时停车约定

上位机的行为：只要 200 ms 内没有收到新的 /cmd_vel，它就持续发送全零帧（vx = vy = omega = 0），而不是停发。也就是说链路正常运行时下位机一定会持续收到帧——「收不到帧」本身就意味着异常（上位机崩溃、串口断开、线缆松动）。所以下位机必须自带超时停车，不能只依赖上位机发零帧。

上位机零速帧的固定字节为 `AA 55 00 00 00 00 00 00 CF 09`，可用它作为「停车帧」的识别基准。

7. 示例帧（可直接作单元测试向量）

下表全部由上位机代码 `protocol.py` 现算生成，与实际下发的字节完全一致。建议下位机把这些帧灌进解析函数做回归测试。

场景	vx (m/s)	vy (m/s)	omega (rad/s)	payload byte2~byte7	完整 10 字节帧
静止（停车）	+0.00	+0.00	+0.000	00 00 00 00 00 00	AA 55 00 00 00 00 00 00 CF 09
前进	+0.25	+0.00	+0.000	FA 00 00 00 00 00	AA 55 FA 00 00 00 00 00 DB 53
后退	-0.20	+0.00	+0.000	38 FF 00 00 00 00	AA 55 38 FF 00 00 00 00 DF A5
左横移	+0.00	+0.15	+0.000	00 00 96 00 00 00	AA 55 00 00 96 00 00 00 E2 81
原地右转	+0.00	+0.00	-0.300	00 00 00 00 D4 FE	AA 55 00 00 00 00 D4 FE 11 89
斜向 + 转向	+0.10	+0.05	+0.200	64 00 32 00 C8 00	AA 55 64 00 32 00 C8 00 9E 55
极限值（前进满速）	+2.50	+0.00	+0.000	C4 09 00 00 00 00	AA 55 C4 09 00 00 00 00 03 8C
极限值（旋转满速）	+0.00	+0.00	+1.500	00 00 00 00 DC 05	AA 55 00 00 00 00 DC 05 57 CA

7.1 逐字节解释示例：前进 0.25 m/s

字节位置	值	含义
byte 0	AA	帧头高字节，固定 0xAA
byte 1	55	帧头低字节，固定 0x55
byte 2	FA	vx 低字节：250 mm/s = 0x00FA
byte 3	00	vx 高字节
byte 4	00	vy 低字节：0 mm/s
byte 5	00	vy 高字节
byte 6	00	omega 低字节：0 mrad/s
byte 7	00	omega 高字节
byte 8	DB	CRC16 低字节
byte 9	53	CRC16 高字节

整帧：`AA 55 FA 00 00 00 00 00 DB 53`；CRC16-MODBUS 计算结果为 `0x53DB`（小端写入即 DB 53）。

8. 上位机侧行为（对接需知）

参数（ROS 2 参数名）	默认值	对下位机的影响
velocity_rate_hz	30.0	发包频率，决定下位机收到帧的节奏
cmd_vel_timeout_sec	0.2	超过该时长无新指令即发零速帧
send_zero_on_timeout	true	置 false 则超时后直接停发（此时下位机的超时兜底更关键）
max_vx_m_s / max_vy_m_s	2.5 / 2.5	下发前的限幅，vx / vy 不会超过 ±2500
max_wz_rad_s	1.5	下发前的角速度限幅，omega 不会超过 ±1500
baudrate	115200	必须与下位机一致
port	/dev/...	上位机侧串口设备名（默认 /dev/r2_chassis），与下位机无关

8.1 上电与重启行为

- 上位机节点启动后立刻按帧率发帧，在收到第一条 /cmd_ve1 之前发的都是零速帧；
- 没有任何握手、心跳或使能流程。下位机随时上电都能从下一帧开始正常接收；
- 下位机重启不需要通知上位机，只要按 10 字节切帧即可重新同步；
- 上位机不会因为下位机不回包而报错——它根本不读串口。

9. 下位机自检清单

#	检查项	通过标准
1	串口参数	115200 8N1，无流控
2	收帧切分	以 10 字节为单位处理，不用搜帧头切帧
3	CRC 实现	对 byte0~7 算 0xA001 / 0xFFFF，结果与 byte8、byte9（小端）一致
4	字节序	vx / vy / omega 按小端拼 int16，负数能正确解析（如 0xFF9C = -100）
5	重同步	灌入含假帧头或坏 CRC 的脏字节流后，能在 1 ~ 2 帧内恢复对齐
6	超时停车	停发数据 300 ~ 500 ms 后电机速度归零并置标志
7	示例帧回归	第 7 节所有示例帧的解析结果与表中数值一致
8	零速帧	收到全零帧后电机确实停止（而不是仅写入变量）

> 负数字节序自检用例：int16 -100 = 0xFF9C，小端下发为 **9C FF**（低字节 9C 在前，高字节 FF 在后）。若解析出 +155 之类的怪值，说明字节序或符号扩展写错了。

附录 A 上位机发送侧实现（Python，节选自 protocol.py）

```
FRAME_LEN = 10
HEADER    = b"\xAA\x55"
VX_SCALE = VY_SCALE = OMEGA_SCALE = 1000.0   # m/s->mm/s, rad/s->mrad/s

def pack_velocity_frame(vx_mm_s, vy_mm_s, omega_mrad_s):
    payload = bytearray(HEADER)
    payload.extend(struct.pack("<h", clamp_i16(vx_mm_s)))
    payload.extend(struct.pack("<h", clamp_i16(vy_mm_s)))
    payload.extend(struct.pack("<h", clamp_i16(omega_mrad_s)))
    payload.extend(struct.pack("<H", crc16_modbus(bytes(payload))))
    return bytes(payload)          # always exactly 10 bytes

# node side, inside a 30 Hz timer:
#   frame = pack_velocity_frame(vx*1000, vy*1000, wz*1000)
#   serial.write(frame); serial.flush()      # write only, never read
```

附录 B Python 侧解析参考（可当对照实现）

```
def parse_velocity_frame(raw: bytes):
    assert len(raw) == 10 and raw[0:2] == b"\xAA\x55"
    expect = raw[8] | (raw[9] << 8)
    assert crc16_modbus(raw[:8]) == expect, "crc mismatch"
    vx, vy, omega = struct.unpack_from("<hhh", raw, 2)
    return vx / 1000.0, vy / 1000.0, omega / 1000.0   # m/s, m/s, rad/s
```

文档由 `docs/build_protocol_pdf.py` 自动生成；协议若有改动，重新运行该脚本即可得到与新版固件一致的新版 PDF。协议权威定义见 `techx_r2_chassis_control/protocol.py` 文件头注释。