同机筑巡
TJZX TECHNOLOGY

负压吸附爬壁机器人底盘
二次开发说明文档
型号 SC-09

更多产品动态

关注微信公众号

技术支持

同机筑巡
TJZX TECHNOLOGY

目 录

1 概述 1
2 内部电路 1
3 元件说明 2
3.1 电源管理模块 2
3.2 核心控制模块 3
3.2.1 核心控制板 3
3.3 遥控控制模块 5
3.3.1 遥控 5
3.3.2 天空端 5
3.4 电机驱动模块 7
3.4.1 电机驱动板 7
3.4.2 车轮电机 8
3.5 风机控制模块 9
3.5.1 调速器 9
3.5.2 无刷电调 10
3.5.3 涵道风机 11
4 机载计算机接入方式 12
4.1 示例：N100 型号 12
4.2 示例：Jetson Orin NX 型号 13
附件 1 15
附件 2 22

1 概述

SC-09 底盘专为二次开发设计。底盘本身已具备吸附爬壁功能，可直行和转向爬壁，使用遥控远程控制；配备了图传摄像头，可在遥控上实时查看和录制摄像头画面；底盘采用系留供电，供电电压 32V；底盘内分电板预留了 32V 供电接口，核心板预留了 12V 和 5V 供电接口、遥控控制接口以及内部安装空间。开发过程中如有问题请联系我司工程师（扫描封面技术支持二维码添加）。

2 内部电路

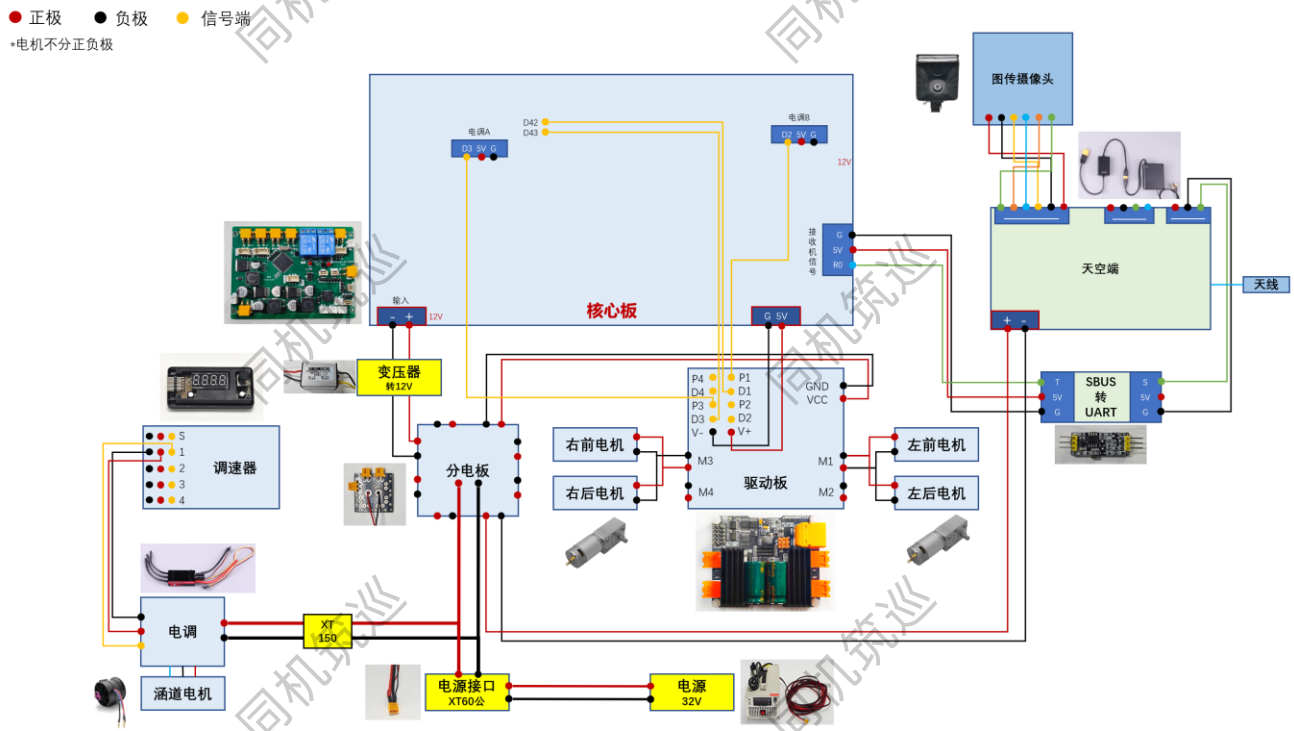

图 1 内部电路连接示意图

图 2 常用插头

3 元件说明

SC-09 底盘主要包含电源管理模块、核心控制模块、遥控控制模块、电机驱动模块和风机控制模块。电源管理模块分为电源接口和分电板，核心控制模块为核心控制板，遥控控制模块分为遥控和天空端，电机驱动模块分为电机驱动板和减速电机，风机控制模块分为调速器、无刷电调和涵道风机。各元件采用魔术贴粘贴于内部空间，便于取下。底盘前盖和后盖均可取下两颗螺丝后开启，如下图所示。

图 3 底盘开盖示意图

3.1 电源管理模块

电源管理模块安装于底盘后方，电源接口采用 XT60 母头，与外部系留电源连接后底盘通电。分电板可分出 8 个接口，为 32V 负载供电，需要时可焊接 XT30PW-F 母头以增加供电接口。

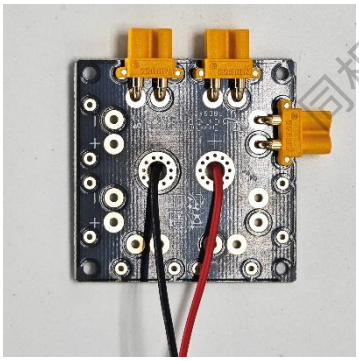

图 4 分电板

3.2 核心控制模块

3.2.1 核心控制板

核心控制板安装于底盘前方。控制板预留了 9 个 12V 供电接口，以及 3 个 5V 供电接口；具备一个 5V 和一个 12V 的继电器；设置了数字信号引脚（包括 PWM 控制）和串口通信引脚。核心控制板的默认板载控制程序见附件 1。

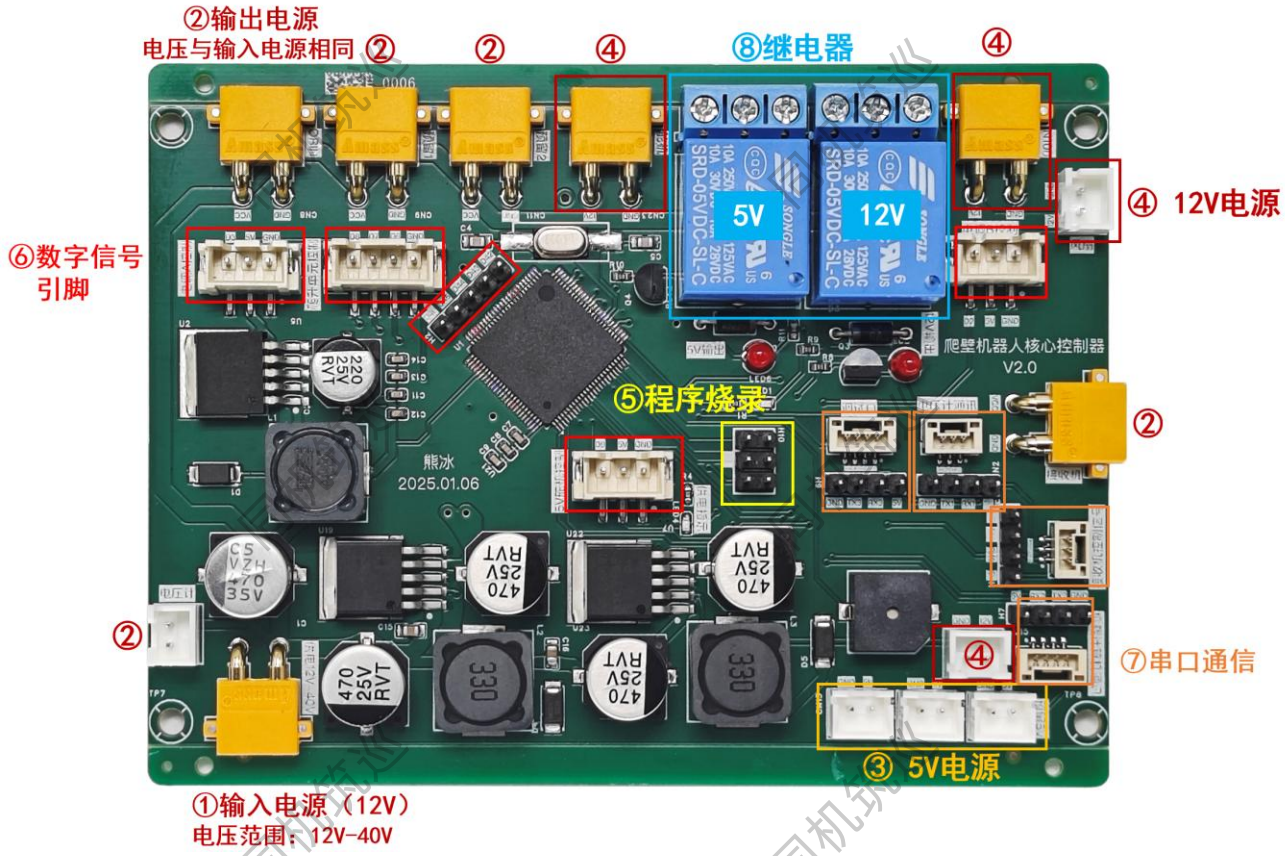

图 5 控制板接口

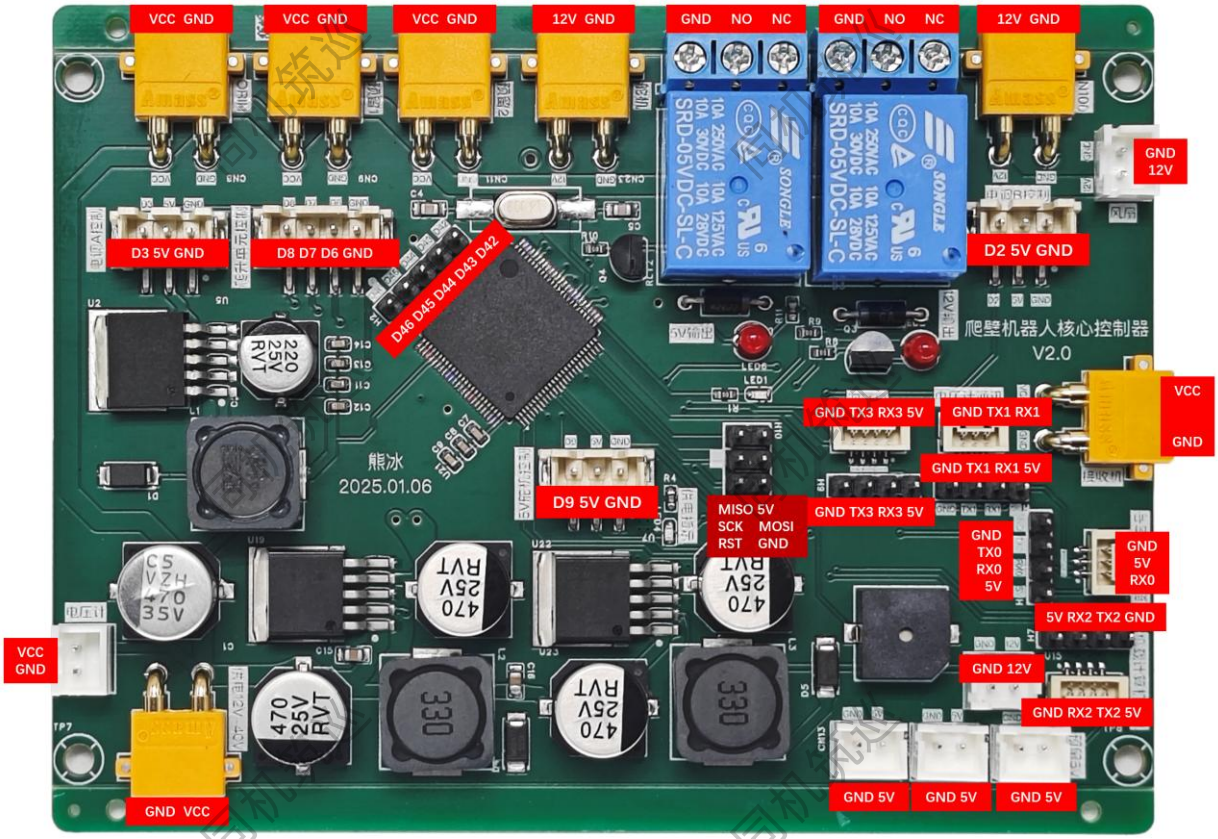

图 6 控制板引脚标注

如需修改控制程序，应使用烧录线连接程序烧录接口以烧录新的控制程序，连接方式如下图，注意烧录线的凸起处应与核心控制板烧录接口的白色方块同边。

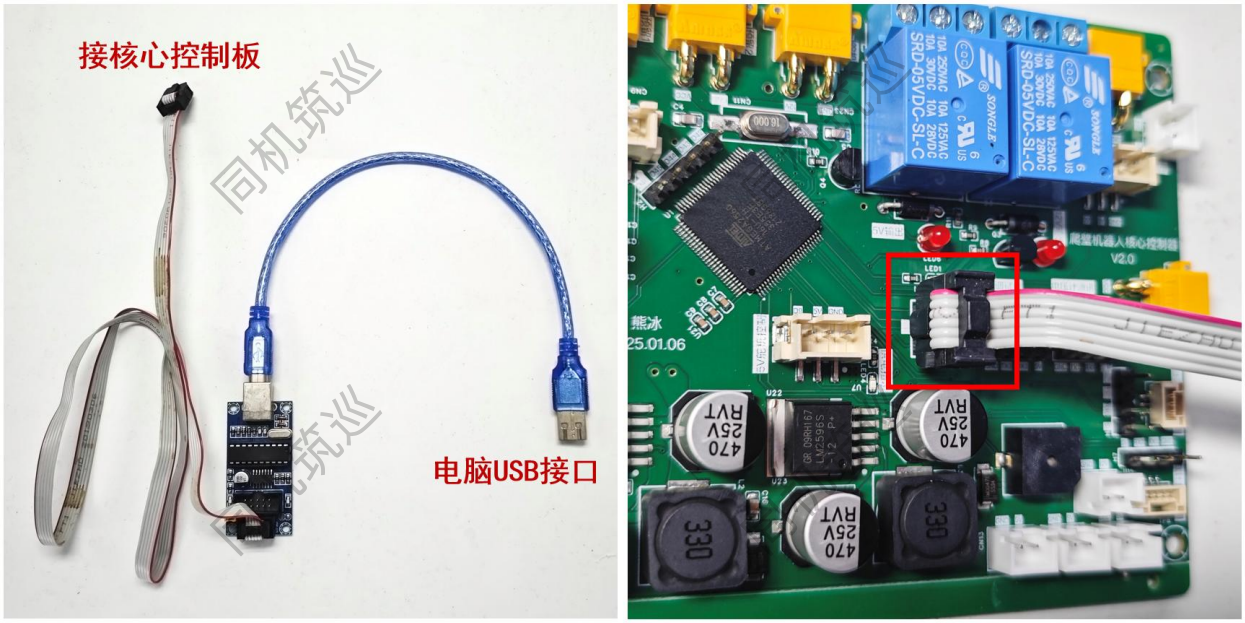

图 7 烧录线连接方式

3.3 遥控控制模块

3.3.1 遥控

SC-09 底盘遥控已使用通道 2（右摇杆前后）和通道 3（左摇杆前后）。点击遥控系统中的“遥控”→“通道设置”，同时使用拨杆或按钮可查看对应通道及值的变化。更多说明见附件 2。

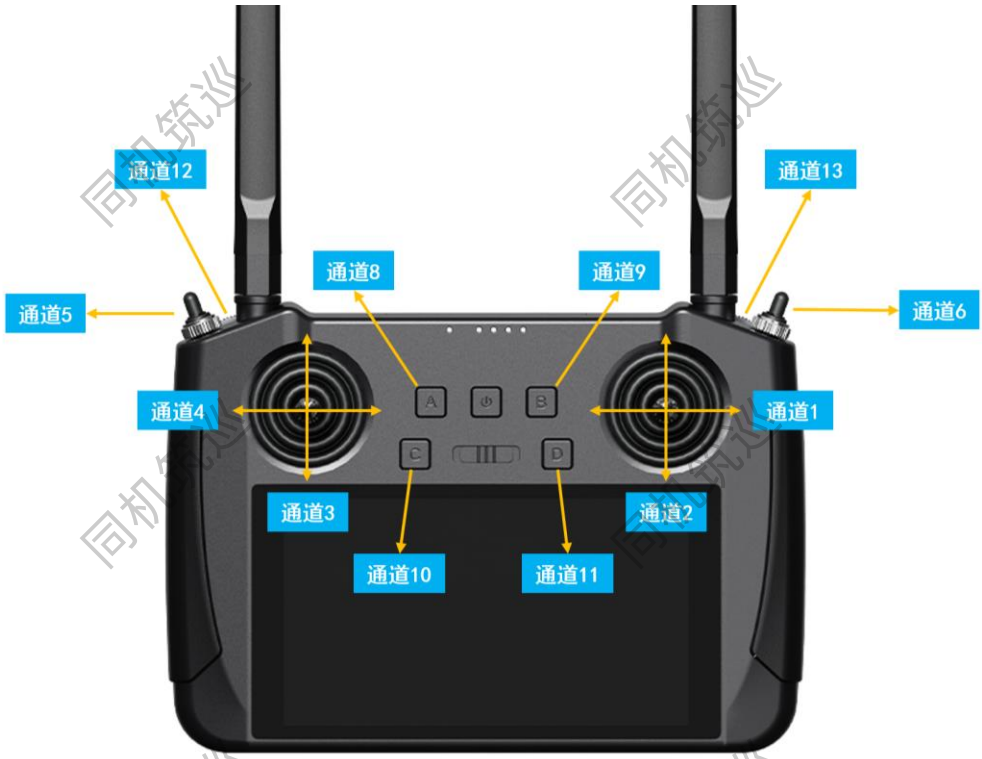

图 8 遥控通道示意图

3.3.2 天空端

天空端是核心控制板和遥控信号的传输介质，且可连接图传摄像头、PWM 负载以及用于串口通信的负载，安装于底盘内部的左前方。天空端与遥控已对频匹配，更多说明见附件 2。


天空端主体
工作电压：12.6-25.2V
电源接口
天线
4639D48E
4S-18S
降压模块
工作电压：16.8-75.6V

图 9 天空端说明

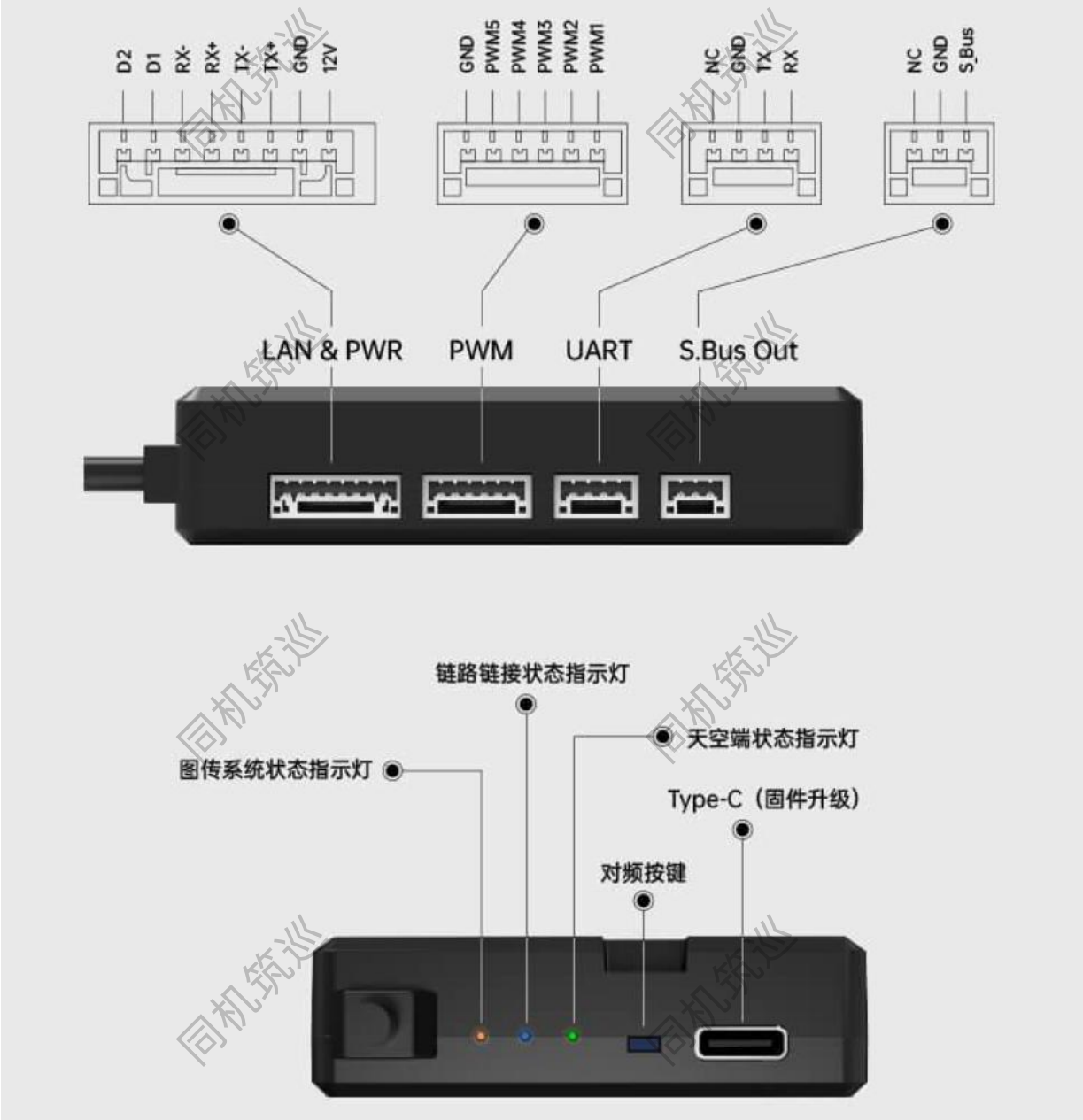

图 10 天空端接口说明

3.4 电机驱动模块

3.4.1 电机驱动板

电机驱动板为四通道，最多可驱动 4 路直流电机，本底盘仅使用其中 2 路，用于控制轮子的运动，驱动板安装于底盘内部的右侧中部位置。

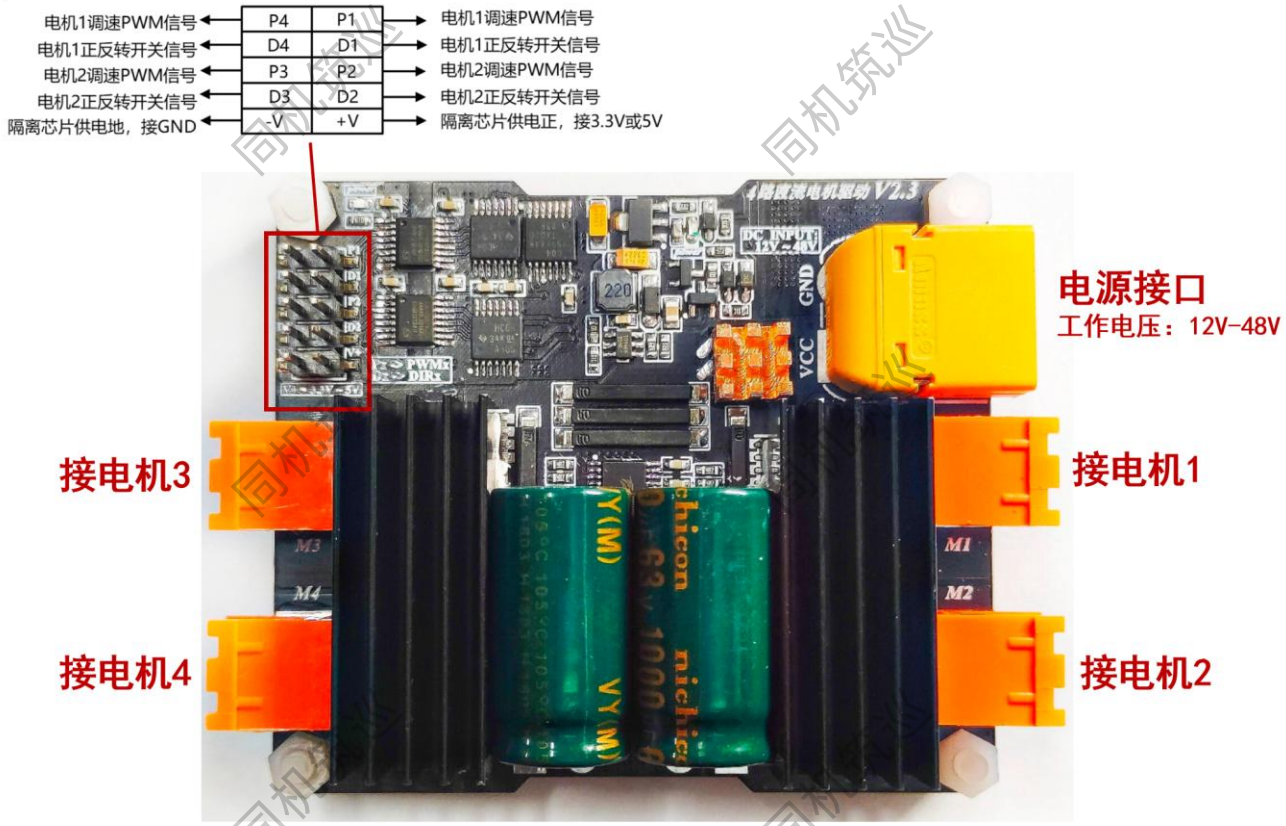

图 11 驱动板接口说明

该驱动板支持 12~48V 电压输入，输入侧有一颗大功率双向瞬态二极管，抑制频繁正反转产生的母线电压尖峰，保护其他模块安全工作；母线上有防反接电路，即使电源反接，模块也不会短路烧毁；具备过压保护和欠压保护，当母线电压高于 51V 触发过压保护，母线电压低于 10.5V 触发欠压保护；功率电路外围保护电路齐全，可频繁正反转，无需延迟，不会轻易损坏；采用简单易用的双线控制，一根控制线通 PWM 信号控制电机启停和转速，另一根控制线通开关信号单独控制电机正反转；逻辑信号与驱动板全隔离，防止功率回路干扰单片机，能有效保护单片机；采用四路全桥分别驱动四个电机，相较于双路复用控制四个电机，具有更强的可控性，能减弱因每路电机特性不同而导致的偏航问题。

表 1 参数

工作电压	12V~48V
单路额定电流	15A
PCB 尺寸	70×60×18.5mm

表 2 控制逻辑（以电机 1 为例）

P1	D1	电机状态
PWM 值	0 或 LOW	正转
PWM 值	1 或 HIGH	反转
0	0（LOW）或 1（HIGH）	制动

3.4.2 车轮电机

车轮电机安装于底盘内部的两侧，与轮子连接。电机的两条接线不分正负极，仅控制电机旋转方向。

图 12 电机

表 3 电机参数

输入电压	12 V
额定转速	35 r/min
额定功率	8.8 W
额定转矩	14 Kg.cm
空载转速	41 r/min
最大空载电流	0.32 A
堵转转矩	30 Kg.cm
最大堵转电流	6.6 A
重量	220 g

3.5 风机控制模块

3.5.1 调速器

调速器用于控制涵道风机的转速，从而调节底盘吸附力，安装于底盘内部右侧。调速器的最大输出信号值为 2200us，但在本底盘中，考虑电压与功率的关系，设置了最大限位值为 1735。

图 13 调速器说明

表 4 调速器参数

输入电压	5-6V
输出信号宽度	约 800-1732us（本底盘最大限位）
最大可输出信号宽度	800-2000us

3.5.2 无刷电调

无刷电调安装于底盘内部的涵道风机上，解析来自调速器的信号从而控制涵道风机转速。油门信号线连接调速器，电源输入端连接电源接口，电机连接端连接涵道风机的电机。

图 14 无刷电调说明

3.5.3 涵道风机

涵道风机安装于底盘内部中间位置，与底板和顶板相接。涵道风机的三条接线仅控制电机旋转方向，即出风方向，出风方向错误时，调换连接任意两条线即可。

图 15 涵道风机

4 机载计算机接入方式

4.1 示例：N100 型号

图 16 魔方 M6S N100 mini PC

该型号机载计算机工作电压为 12V，可使用核心控制板的 12V 电源接口为其供电，或使用分电板供电（分电板与 N100 之间需连接 32V 转 12V 变压器），电源接口为 TYPE-C 口，接入后需设置上电自启动。

N100 为 windows 系统，使用便捷，可直接通过 USB 接口接入其他负载，还具备 HDMI 接口、耳机接口、SD 卡接口和网口等。安装时建议拆卸外壳以减小重量和体积。

4.2 示例：Jetson Orin NX 型号

图 17 Jetson Orin NX

该型号机载计算机工作电压为 12V-26V，可使用核心控制板或分电板为其供电（若使用分电板供电，需要接入合适的变压模块），电源接口为 XT30U 公头。

图 18 电源接口

Orin NX 为基于 Linux 的 ubuntu 系统，更建议专业开发者使用。该型号具备 USB、micro HDMI、UART、I2C 等接口以接入其他负载。

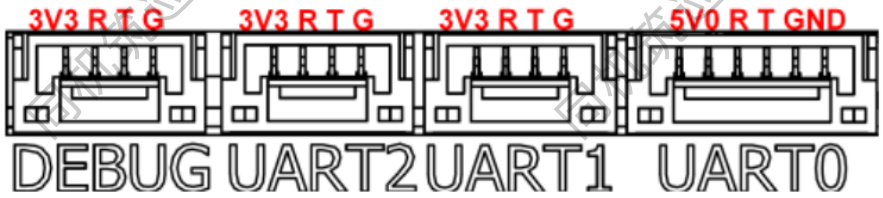

图 19 接口定义

表 5 接口引脚定义

	PIN1	PIN2	PIN3	PIN4	PIN5	PIN6
DEBUG	3V3	RX	TX	GND	无	无
UART2	3V3	RX	TX	GND	无	无
UART1	3V3	RX	TX	GND	无	无
UART0	5V0	5V0	RX	TX	GND	GND

附件 1

核心控制板默认板载控制程序

```
#include <Arduino.h>
#include <Servo.h>
#include <Arduino_FreeRTOS.h>
#include <Wire.h>
#include <SoftwareSerial.h>
#include <avr/wdt.h>

// 串口定义
HardwareSerial& receiverSerial = Serial;
HardwareSerial& debugSerial = Serial3;

// 引脚定义
#define LEFT_MOTOR_PWM 3 //左轮电机
#define LEFT_MOTOR_DIR 43
#define RIGHT_MOTOR_PWM 2 //右轮电机
#define RIGHT_MOTOR_DIR 42
#define RELAY_PIN 46
#define BUZZER_PIN 12

// 定义通道的最小、中间和最大值
#define CHANNEL_MIN 272
#define CHANNEL_MID 992
#define CHANNEL_MAX 1712

// 电机 PWM 范围；最大区间 0-254
#define MOTOR_MIN 0
#define MOTOR_MAX 100

// 通道变量
volatile int channelValues[16] = {0};

// 信号状态和超时检测
volatile unsigned long lastSignalTime = 0;
const unsigned long signalTimeout = 1000;  // 超时时间设置为 1000 毫秒
volatile int signal_connected = 0;

// 声明任务函数
void SBUS_LISTEN_TASK(void *pvParameters);
void SERVO_CONTROL_TASK(void *pvParameters);
void LED_BUZZER_TASK(void *pvParameters);
void SAFETY_MONITOR_TASK(void *pvParameters);

void setup() {
  // 初始化串口
```

```
  receiverSerial.begin(115200);
  debugSerial.begin(115200);

  // 初始化控制引脚
  pinMode(LEFT_MOTOR_PWM, OUTPUT);
  pinMode(LEFT_MOTOR_DIR, OUTPUT);
  pinMode(RIGHT_MOTOR_PWM, OUTPUT);
  pinMode(RIGHT_MOTOR_DIR, OUTPUT);
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);

  // 初始停止电机
  analogWrite(LEFT_MOTOR_PWM, 0);
  analogWrite(RIGHT_MOTOR_PWM, 0);
  digitalWrite(LEFT_MOTOR_DIR, LOW);
  digitalWrite(RIGHT_MOTOR_DIR, LOW);

  // 初始化看门狗，设置超时时间为 2 秒
  wdt_enable(WDTO_2S);

  // 创建任务
  xTaskCreate(SBUS_LISTEN_TASK, "SBUS_LISTEN", 256, NULL, 1, NULL);
  xTaskCreate(SERVO_CONTROL_TASK, "SERVO_CONTROL", 256, NULL, 1, NULL);
  xTaskCreate(LED_BUZZER_TASK, "LED_BUZZER", 128, NULL, 1, NULL);
  xTaskCreate(SAFETY_MONITOR_TASK, "SAFETY_MONITOR", 256, NULL, 1, NULL);  // 创
建安全监控任务
}

void loop() {
  // 空循环，所有操作在 FreeRTOS 任务中处理
}

/* 接收机串口监听任务 */
void SBUS_LISTEN_TASK(void *pvParameters) {
  while (1) {
    receiverSerial.flush();

    while (!receiverSerial.available()); // 等待串口有数据

    int incomingByte = receiverSerial.read();
    // 检查帧头
    if (incomingByte == 0x0F) {
      byte data[34];
```

```
      int i = 0;

      // 等待并读取整个数据帧
      while (i < 34) {
        if (receiverSerial.available()) {
          data[i++] = receiverSerial.read();
        }
      }

      // 检查帧尾
      if (data[32] == 0x00) {  // 根据您的描述，这里检查第 33 个字节是否为 0

        // 更新最后接收信号的时间及信号连接状态
        lastSignalTime = millis();
        signal_connected = 1;

        // 解析各通道数据
        for (int j = 0; j < 16; j++) {
          channelValues[j] = data[j * 2] << 8 | data[j * 2 + 1];
        }

        // 可选：输出通道数据以供调试
        // for (int j = 0; j < 16; j++) {
        //   debugSerial.print("Channel ");
        //   debugSerial.print(j + 1);
        //   debugSerial.print(": ");
        //   debugSerial.println(channelValues[j]);
        // }

      } else {
        receiverSerial.println("Frame error");
      }
    }

    // taskYIELD();  // 释放 CPU 时间片 不知道有没有用

    // 处理串口断开连接的情况 断开后会重连接
    if (!receiverSerial) {
      receiverSerial.end();
      vTaskDelay(pdMS_TO_TICKS(100));
      receiverSerial.begin(115200);
    }
  }
}
```

```
/* 电机控制任务 */
void SERVO_CONTROL_TASK(void *pvParameters) {
    receiverSerial.println("SBUS_LISTEN");  // 似乎用于初始化或标志开始，具体功能根据实际调整
    vTaskDelay(pdMS_TO_TICKS(2000));  // 初始延迟以稳定系统

    while (1) {
      /******* 车轮运动控制 *******/
      // 左轮控制：3 号通道（数组索引为 2）
      if (channelValues[2] < CHANNEL_MID){ // 左轮向后
        int speed = map(channelValues[2], CHANNEL_MIN, CHANNEL_MID, MOTOR_MAX, MOTOR_MIN);
        digitalWrite(LEFT_MOTOR_DIR, HIGH);
        analogWrite(LEFT_MOTOR_PWM, speed);
      }
      if (channelValues[2] > CHANNEL_MID){ // 左轮向前
        int speed = map(channelValues[2], CHANNEL_MID, CHANNEL_MAX, MOTOR_MIN, MOTOR_MAX);
        digitalWrite(LEFT_MOTOR_DIR, LOW);
        analogWrite(LEFT_MOTOR_PWM, speed);
      }
      if (channelValues[2] == CHANNEL_MID || channelValues[2] < CHANNEL_MIN || channelValues[2] > CHANNEL_MAX) { // 左轮停止
        digitalWrite(LEFT_MOTOR_DIR, LOW);
        analogWrite(LEFT_MOTOR_PWM, 0);
      }

      // 右轮控制：2 号通道（数组索引为 1）
      if (channelValues[1] < CHANNEL_MID){ // 右轮向后
        int speed = map(channelValues[1], CHANNEL_MIN, CHANNEL_MID, MOTOR_MAX, MOTOR_MIN);
        digitalWrite(RIGHT_MOTOR_DIR, HIGH);
        analogWrite(RIGHT_MOTOR_PWM, speed);
      }
      if (channelValues[1] > CHANNEL_MID){ // 右轮向前
        int speed = map(channelValues[1], CHANNEL_MID, CHANNEL_MAX, MOTOR_MIN, MOTOR_MAX);
        digitalWrite(RIGHT_MOTOR_DIR, LOW);
        analogWrite(RIGHT_MOTOR_PWM, speed);
      }
      if (channelValues[1] == CHANNEL_MID || channelValues[1] < CHANNEL_MIN || channelValues[1] > CHANNEL_MAX) { // 右轮停止
        digitalWrite(RIGHT_MOTOR_DIR, LOW);
```

```
        analogWrite(RIGHT_MOTOR_PWM, 0);
      }

      vTaskDelay(pdMS_TO_TICKS(20));  // 小延迟帮助调度器管理其他任务
    }
}

/* 安全保障任务 */
void SAFETY_MONITOR_TASK(void *pvParameters) {
    while (1) {
        wdt_reset();  // 看门狗复位
        if (millis() - lastSignalTime > signalTimeout) {
            signal_connected = 0;
            // 如果超时没有接收到信号
            // 将 channelValues 中所有值设置为 0
            for (int i = 0; i < 16; i++) {
                channelValues[i] = 0;
            }
            // 1. 车轮停止
            digitalWrite(LEFT_MOTOR_DIR, LOW);
            analogWrite(LEFT_MOTOR_PWM, 0);
            digitalWrite(RIGHT_MOTOR_DIR, LOW);
            analogWrite(RIGHT_MOTOR_PWM, 0);
            // 2. 蜂鸣器发出声音
            digitalWrite(BUZZER_PIN, HIGH);
            vTaskDelay(pdMS_TO_TICKS(100));  // 蜂鸣器响 100 毫秒
            digitalWrite(BUZZER_PIN, LOW);
        }
        vTaskDelay(pdMS_TO_TICKS(1000));  // 每 100 毫秒检查一次
    }
}

/* LED 和蜂鸣器状态控制任务 */
void LED_BUZZER_TASK(void *pvParameters) {
  pinMode(LED_BUILTIN, OUTPUT);  // 设置内置 LED 引脚为输出模式
  bool hasBuzzed = false;        // 防止重复触发蜂鸣器

  while (1) {
    // LED 闪烁控制
    digitalWrite(LED_BUILTIN, HIGH);  // 打开 LED
    vTaskDelay(pdMS_TO_TICKS(500));   // 等待 500 毫秒
    digitalWrite(LED_BUILTIN, LOW);   // 关闭 LED
    vTaskDelay(pdMS_TO_TICKS(500));   // 再等待 500 毫秒
```

```
    // 蜂鸣器控制
    bool isValid = false;
    for (int i = 0; i < 16; i++) {
      if (channelValues[i] != 0) {
        isValid = true;
        break;
      }
    }

    if (isValid && !hasBuzzed) {
      hasBuzzed = true;
      // 发出三声蜂鸣
      debugSerial.println("Buzzer triggered");
      for (int i = 0; i < 3; i++) {
        digitalWrite(BUZZER_PIN, HIGH);
        vTaskDelay(pdMS_TO_TICKS(200)); // 蜂鸣器响 100 毫秒
        digitalWrite(BUZZER_PIN, LOW);
        vTaskDelay(pdMS_TO_TICKS(200)); // 蜂鸣器停 100 毫秒
      }
    } else if (!isValid) {
      hasBuzzed = false; // 如果数据无效，重置标志，以便再次触发
    }
  }
}
```

附件 2

遥控及天空端说明

1 通道定义

MK15 手持地面站拥有 13 个物理通道以及 16 个通讯通道。其中第 10 至第 14 通讯通道默认与 PWM 第一至第五通道映射。

通道序号	物理通道类型	默认物理开关	备注
1	副翼摇杆	J1	
2	升降摇杆（美国手）	J2	
3	油门摇杆（美国手）	J3	
4	方向摇杆	J4	
5	三档开关 SA	SA	
6	三档开关 SB	SB	
7	三档开关 SC	SC	
8	按键 A	A	
9	按键 B	B	
10	按键 C	C	PWM1
11	按键 D	D	PWM2
12	旋转拨轮 LD	LD	PWM3
13	旋转拨轮 RD	RD	PWM4
14			PWM5
15			探照灯 云台俯仰
16			副探照灯 云台一键回中

2 链路数据流示意图

遥控数据流


机载电脑
思翼AI跟踪模块
HDMI相机
天空端HDMI输入模块
（选配）4S~18S降压模块
网口FPV摄像头
ZT6
ZT30
ZR30
光电吊舱
（云台相机）
电源输入（XT30接头）
3S ~ 6S（11 ~ 25.2V）
*早期版本只支持3S~4S
请注意甄别
ZR10
A8 mini
A2 mini
舵机
LAN & PWR
LAN & PWR
PWM信号输出
PWM
与飞控数传通信
UART
S.Bus遥控信号输出
S.Bus Out
N7飞行控制器

天空端数据流

3 指示灯状态定义

- 红灯常亮：地面端与天空端未通信
- 红灯快闪：对频中
- 红灯慢闪：固件不匹配
- 红灯三闪：链路初始化失败
- 红灯四闪：地面端摇杆需要校准
- 红绿交替闪烁：安卓系统意外关闭
- 红绿黄交替慢闪：图传启动中
- 红绿黄交替快闪：固件升级中
- 黄灯慢闪：地面端电源电压异常
- 黄灯两闪：地面端蓝牙未识别
- 黄灯三闪：链路数据量一级告警
- 黄灯四闪：链路数据量二级告警
- 黄红：地面端温度一级报警
- 黄红红：地面端温度二级报警
- 黄红红红：地面端温度三级报警
- 绿灯常亮、闪烁：闪烁速度越快，信号强度越差
- 绿灯常亮：有效包 100%
- 绿灯闪烁（1Hz）：有效包 99%~95%
- 绿灯闪烁（间隔 3/5 秒）：有效包 75%~50%
- 绿灯闪烁（间隔 3/10 秒）：有效包 50%~25%
- 绿灯闪烁（间隔 1/25 秒）：有效包小于 25%
- 绿红：天空端温度一级报警
- 绿红红：天空端温度二级报警
- 绿红红红：天空端温度三级报警

遥控指示灯

红灯常亮：地面端与天空端未通信

红灯快闪：对频中

红灯慢闪：固件不匹配

红灯三闪：链路初始化失败

红绿黄交替慢闪：图传启动中

黄灯闪烁：电压告警（输入电压低于 12V）

绿灯常亮、闪烁：闪烁速度越快，信号强度越差

绿灯常亮：有效包 100%

绿灯闪烁（1Hz）：有效包 99%~95%

绿灯闪烁（间隔 3/5 秒）：有效包 75%~50%

绿灯闪烁（间隔 3/10 秒）：有效包 50%~25%

绿灯闪烁（间隔 1/25 秒）：有效包小于 25%

绿红交替闪烁：开始无线对频（上电三次触发）

绿红：天空端温度一级报警

绿红红：天空端温度二级报警

绿红红红：天空端温度三级报警

天空端指示灯