DDS_六轮双驱移动机器人底盘下位机业务流程设计

产品型号	
子系统名称及型号	
模块名称	

版本历史

目录

1 前言

该文档全面梳理了六轮双驱移动机器人底盘（以下简称“机器人底盘”）底层控制的整体框架流程及各个模块功能的实现逻辑，同时对里程计模型涉及的航迹推演算法进行了推导。

1.1 文档目的

该文档为研发人员充分了解机器人底盘控制逻辑提供了参考。

1.2 文档范围

该文档全面梳理了机器人底盘底层控制的设计思路及整体框架流程，并对里程计模型进行了分析。

1.3 参考文献

1.4 定义和缩略语

2 机器人底盘业务流程框架

2.1 简介

机器人底盘控制运行在 FreeRTOS 微操作系统中，通过对任务进行调度实现各模块间的切换。

2.2 整体架构及流程

图 1 是系统架构图，从宏观上描述了系统运行流程。

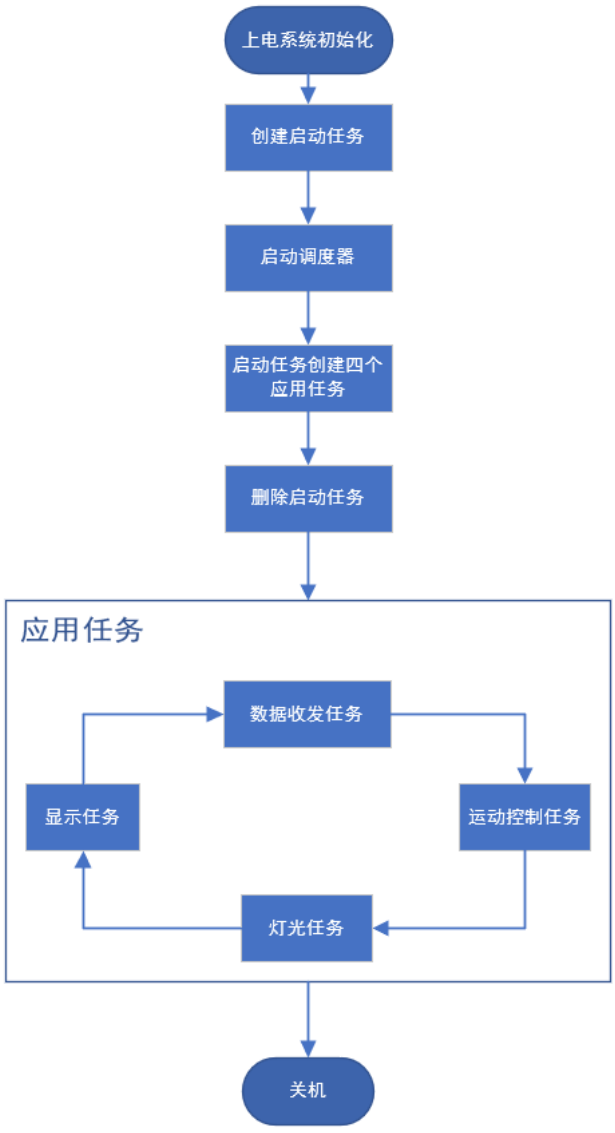

图 1 系统架构图

2.2.1 任务初始化

上电后，首先对硬件初始化，然后创建一个启动任务后就启动调度器，在启动任务里创建四个应用任务。当所有任务创建成功后，启动任务将自己删除，调度器调度任务的执行。任务默认使用固定优先级的抢占式调度策略，对同等优先级的任务采用时间片轮询调度。

首先介绍一下任务创建函数 xTaskCreate()

```
BaseType_t xTaskCreate( TaskFunction_t pxTaskCode,
                        const char * const pcName,
                        const uint16_t usStackDepth,
                        void * const pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t * const pxCreatedTask )
```

参数	说明
pxTaskCode	指向任务入口函数的指针（即实现任务的函数名称）
pcName	任务的描述性名称。主要为了方便调试，但也可用于获取任务句柄
usStackDepth	要分配用于任务堆栈的字数(不是字节)
pvParameters	作为参数传递给创建的任务的一个值
uxPriority	创建任务执行的优先级。优先级范围为 0~configMAX_PRIORITIES，且数值越大优先级越高，0 代表最低优先级。
pxCreatedTask	用来返回正在创建任务的句柄。可以在需要此任务的 API 中调用此句柄，例如修改任务优先级和删除任务。如果你的应用不需要，可以设置为 NULL。

2.2.2 任务调度

启动任务创建完应用任务后，各应用任务即进入就绪状态，调度器开始进行任务调度。应用任务的实现都是在一个 while(1)循环中，不允许退出及有返回值，任务的切换由系统延时函数 vTaskDelayUntil 通过阻塞实现的。高优先级任务先得到执行，同优先级的任务按照时间片轮流执行。以下列出各任

务的优先级,是通过宏来定义的（数值越大，优先级越高。且空闲任务优先级最低）:

任务	宏	值
communicationTask	COMMUNICATION_TASK_PRIO	4
motionControlTask	MOTION_CONTROL_TASK_PRIO	4
showTask	SHOW_TASK_PRIO	3
ledTask	LED_TASK_PRIO	3

调度器首先让最先创建的任务进入运行状态。随后根据延时阻塞函数切换任务。下面介绍 FreeRTOS 官方提供的两个延时函数。

```
void vTaskDelay( TickType_t xTicksToDelay )
```

参数	说明
xTicksToDelay	调用 vTaskDelay 函数的任务在返回就绪态前保持在阻塞态的 tick 中断数（系统每过一段时间会产生 tick 中断以便调度器决定下一个执行的任务）

```
void vTaskDelayUntil( TickType_t * pxPreviousWakeTime,
                      TickType_t xTimeIncrement )
```

参数	说明
pxPreviousWakeTime	pxPreviousWakeTime 保留任务最后一次离开“阻塞”状态的时间（即被唤醒）。此时间用作计算任务下一次离开“阻止”状态的时间的参考点。
xTimeIncrement	固定执行任务的频率

3 功能模块描述

在上述架构流程中，共创建了四个任务，四个任务的状态由调度器进行控制。以下将任务视为模块分别介绍其具体运行逻辑。

3.1 运动控制模块 motionControlTask

根据遥控器或上位机下发的机器人目标速度，解算得到左右驱动轮速度，并通过 CAN 通信，控制驱动器驱动轮毂电机转动。此任务以 100hz 固定频率运行。

3.1.1 运动学模型分析

机器人底盘由两个驱动轮和四个万向轮组成。两个驱动轮分别位于底盘的左右两侧，由两个独立的轮毂电机分别控制，通过不同的双轮速度实现机器人的转向。万向轮的作用仅是支撑机器人，以保持前后平衡。

v_l、v_r分别是当前时刻左轮和右轮的瞬时线速度，l是左右两轮的轮间距，v是当前时刻机器人底盘的瞬时线速度，R是机器人底盘进行转向时的转向半径，ω是机器人底盘进行转向时的转向角速度。其中，l可以通过测量得到，在后续计算过程中作为默认已知的条件。

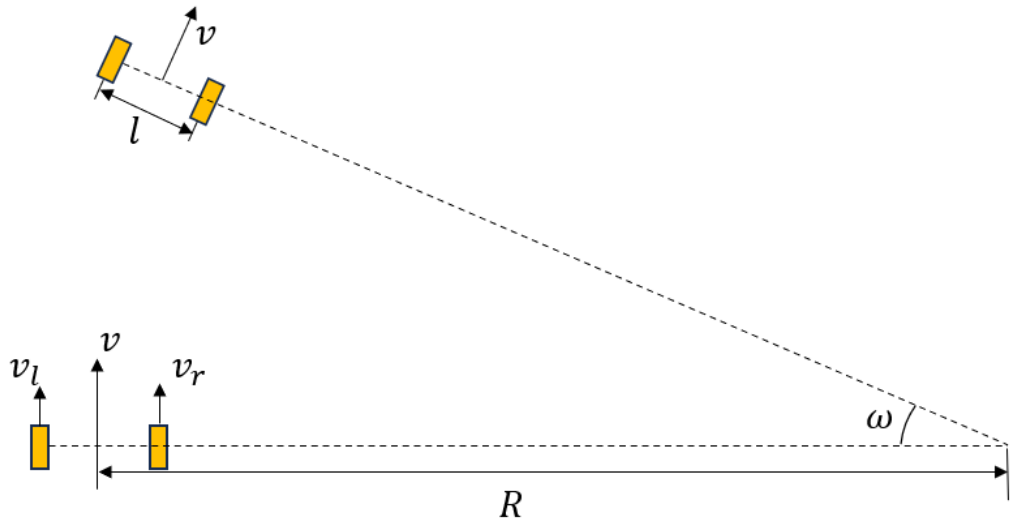

图 2 机器人转向模型

1) 两轮速度→机器人速度（运动学逆解）

问题抽象：已知左轮和右轮的瞬时线速度v_l、v_r，求机器人瞬时线速度v和转向角速度ω

a) 行进过程中，左轮和右轮转向角速度是一致的

$$\omega = \frac{v_l}{R+\frac{l}{2}} = \frac{v_r}{R-\frac{l}{2}}$$

$$R = \frac{l(v_l+v_r)}{2(v_l-v_r)} \tag{3-1}$$

$$\Rightarrow \omega = \frac{v_l-v_r}{l}$$

b） 机器人瞬时线速度v直接取左轮和右轮瞬时线速度v_l、v_r的平均值

$$v=\frac{v_l+v_r}{2} \tag{3-2}$$

这部分由`inverseSolutionKinematics()`实现。

```
motor_left_.line_speed_target = robot_x - robot_z * WHEELSPACING / 2;
motor_right_.line_speed_target = robot_x + robot_z * WHEELSPACING / 2;
```

2） 机器人速度→两轮速度（运动学正解）

问题抽象：已知机器人瞬时线速度v和转向角速度ω，求左右轮的瞬时线速度v_l、v_r。

联立v和ω表达式

$$\begin{cases} v=\dfrac{v_l+v_r}{2} \\ \omega=\dfrac{v_l-v_r}{l} \end{cases} \tag{3-3}$$

这部分在赋值数据时`getToSendData()`解算实现

```
send_data_.SENSOR_STR.robot_x=
((motor_left_.line_speed_feedback   +   motor_right_.line_speed_feedback)/2)*
1000; //小车 x 速度
send_data_.SENSOR_STR.robot_y = 0;//小车 y 速度
send_data_.SENSOR_STR.robot_z=
((motor_right_.line_speed_feedback-
motor_left_.line_speed_feedback)/WHEELSPACING)*1000;//小车 z 速度
```

3.1.2 航迹推算

上位机接收到机器人底盘发送的速度信息，用其作为里程计模型的数据来源。构建里程计模型使用航迹推算算法。

航迹推算是一种利用现在物体位置及速度推定未来位置方向的航海技术，移动机器人运动过程也大量使用，但容易受到误差累积的影响。下图是机器人底盘在移动过程中位置示意图。

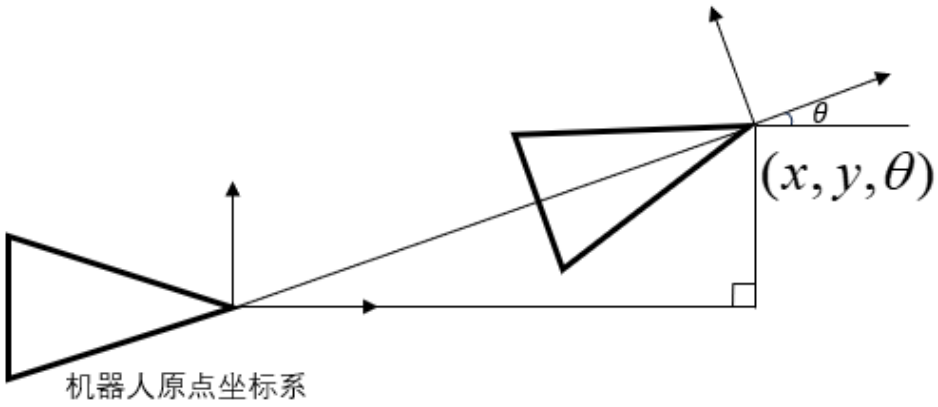

图 3 机器人底盘轨迹

(x,y,θ) 为底盘当前位姿

$(dx,dy,d\theta)$ 为运动学解算增量

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} \tag{3-4}$$

其中，$T_R^W = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 为从机器人坐标系到世界坐标系的旋转变换矩阵

下面是代码实现：

```
Robot_Pos.X += (Robot_Vel.X * cos(Robot_Pos.Z) - Robot_Vel.Y * sin(Robot_Pos.Z)) *
Sampling_Time;//计算 x 方向的位移
Robot_Pos.Y += (Robot_Vel.X * sin(Robot_Pos.Z) + Robot_Vel.Y * cos(Robot_Pos.Z)) *
Sampling_Time;//计算 y 方向的位移，
Robot_Pos.Z += Robot_Vel.Z * Sampling_Time; //角位移
```

其中，`Robot_Pos.Z` 为角位移，即公式中的 θ，由角速度乘以采样时间得到。

3.1.3 整理变量说明

变量	类型	说明
motor_left_	MOTOR_PARAMETER 结构体 成员： rotate_speed_target：目标转速 rotate_speed_feedback：反馈转速 line_speed_target：目标线速度 line_speed_feedback：反馈线速度	机器人左电机速度参数，存储目标速度和反馈速度
motor_right_		机器人右电机速度结构体，同上
robot_x_	Float	机器人 x 轴速度
robot_y_	Float	机器人 y 轴速度
robot_z_	Float	机器人 z 轴速度
robot_control_mode_	枚举类型： ROBOT_CONTROL_MODE 可选值为： CONTROL_MODE_UNKNOW：未知方式控制 CONTROL_MODE_REMOTE：遥控器控制 CONTROL_MODE_ROS：ROS 上位机控制	

3.1.4 模块流程详解

运动控制模块首先根据变量 robot_control_mode_ 对不同控制方式（见模式切换）下发的速度进行解析 setRobotSpeed()，并分别赋值给 robot_x_、robot_y_、robot_z_，再根据机器人运动学逆解 inverseSolutionKinematics()由机器人速度得到左右轮（关节）速度，最后交给驱动器 setMotorSpeed()进行实际速度控制。

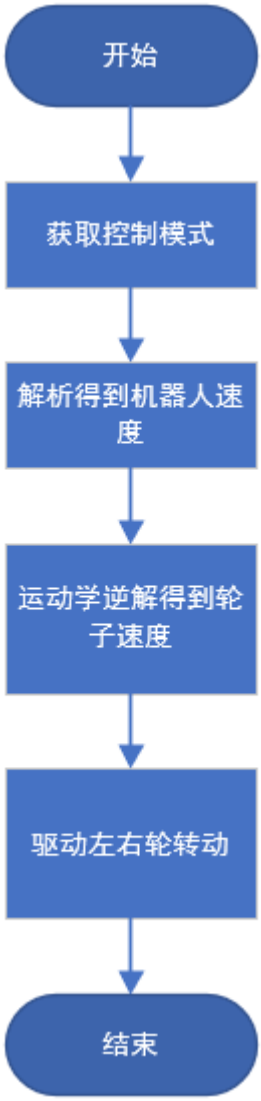

图 4 运动控制流程图

为防止底盘与上位机失联状态下，机器人底盘速度失控，设置速度时需进行数据帧判断。首先记录最新一次接收到 ros 下发速度的时间

```
recive_time_ = getSysTickCnt();
```

然后在设置速度时，获取当前时间

```
unsigned int current_time = getSysTickCnt();
```

在对速度赋值时，通过获取速度时间是否先于设置速度时间且两个时间间隔不大于 300ms 来作为上位机与控制板是否失联的条件，如果失联则将机器人速度置 0

```
else if (robot_control_mode_ == CONTROL_MODE_ROS)
    {
        unsigned int recive_time = recive_time_;
```

```
        if (current_time > recive_time && (current_time - recive_time) >
300)
        {
            robot_x_ = 0;
            robot_y_ = 0;
            robot_z_ = 0;
         }
        else
        {
            robot_x_ = recive_robot_speed_x_;
            robot_y_ = recive_robot_speed_y_;
            robot_z_ = recive_robot_speed_z_;
        }
    }
```

3.2数据收发模块 communicationTask

接收遥控器和上位机下发的数据，并获取打包需要上传到上位机的数据。同时获取当前机器人底盘被控模式（通过设置标志位）。该任务以 20hz 固定频率执行。

3.2.1 整理变量说明

变量	说明
remote_flag_	遥控器控制标志位 0：关闭 1：开启 初始值：0 转换条件：遥控器打开
ros_flag_	ros 上位机控制标志位 0：关闭 1：开启 初始值：0 转换条件：遥控器关闭，上位机有下发速度
sbus_channel_data_transformed_	存放遥控器转换后的通道数据结构体 类型：SBUS_CHANNEL_STRUCT 成员：1~10 个通道数据
usart5_recive_buffer_ [MAX_BUFF_LEN]	存放从上位机接收到的数据的数组。 MAX_BUFF_LEN：数组最大长度
control_speed_level_	存放遥控器下发的机器人速度等级
control_director_level_	存放遥控器下发的机器人运动方向
recive_robot_speed_x_	存放由串口数据解析得到的上位机下发的机器人 x 轴速度

recive_robot_speed_y_	存放由串口数据解析得到的上位机下发的机器人 y 轴速度
recive_robot_speed_z_	存放由串口数据解析得到的上位机下发的机器人 z 轴速度

3.2.2 模块流程详解

图 5 数据收发流程

1） 数据收发

a） 机器人底盘与遥控器

控制板的串口 5 接收遥控器发送数据，格式采用 SBUS 协议。控制板解析 SBUS 信号代码如下：

```
unsigned char sbusTransform(unsigned char* ucBuf)
{
    if (ucBuf[23] == 0)
    {
     sbus_channel_data_transformed_.remote_receiver_connect_state = 1;
     sbus_channel_data_transformed_.channel_1=((int16_t)ucBuf[1]>>0|
((int16_t)ucBuf[2] << 8)) & 0x07FF;
         sbus_channel_data_transformed_.channel_2 = ((int16_t)ucBuf[2] >> 3 |
((int16_t)ucBuf[3] << 5)) & 0x07FF;
         sbus_channel_data_transformed_.channel_3 = ((int16_t)ucBuf[3] >> 6 |
((int16_t)ucBuf[4] << 2) | (int16_t)ucBuf[5] << 10) & 0x07FF;
         sbus_channel_data_transformed_.channel_4 = ((int16_t)ucBuf[5] >> 1 |
((int16_t)ucBuf[6] << 7)) & 0x07FF;
         sbus_channel_data_transformed_.channel_5 = ((int16_t)ucBuf[6] >> 4 |
((int16_t)ucBuf[7] << 4)) & 0x07FF;
         sbus_channel_data_transformed_.channel_6 = ((int16_t)ucBuf[7] >> 7 |
((int16_t)ucBuf[8] << 1) | (int16_t)ucBuf[9] << 9) & 0x07FF;
         sbus_channel_data_transformed_.channel_7 = ((int16_t)ucBuf[9] >> 2 |
((int16_t)ucBuf[10] << 6)) & 0x07FF;
         sbus_channel_data_transformed_.channel_8 = ((int16_t)ucBuf[10] >> 5 |
((int16_t)ucBuf[11] << 3)) & 0x07FF;
         sbus_channel_data_transformed_.channel_9 = ((int16_t)ucBuf[12] << 0 |
((int16_t)ucBuf[13] << 8)) & 0x07FF;
         sbus_channel_data_transformed_.channel_10 = ((int16_t)ucBuf[13] >> 3 |
((int16_t)ucBuf[14] << 5)) & 0x07FF;
         return 1;
    }
    else
    {
         sbus_channel_data_transformed_.remote_receiver_connect_state = 0;
         return 0;
    }

}
```

b) 机器人底盘与上位机

底盘通过串口 2 与上位机通信，在中断接收上位机发送的控制命令。通过 `getToSendData()` 函数给待发送数据赋值。

2） 模式切换

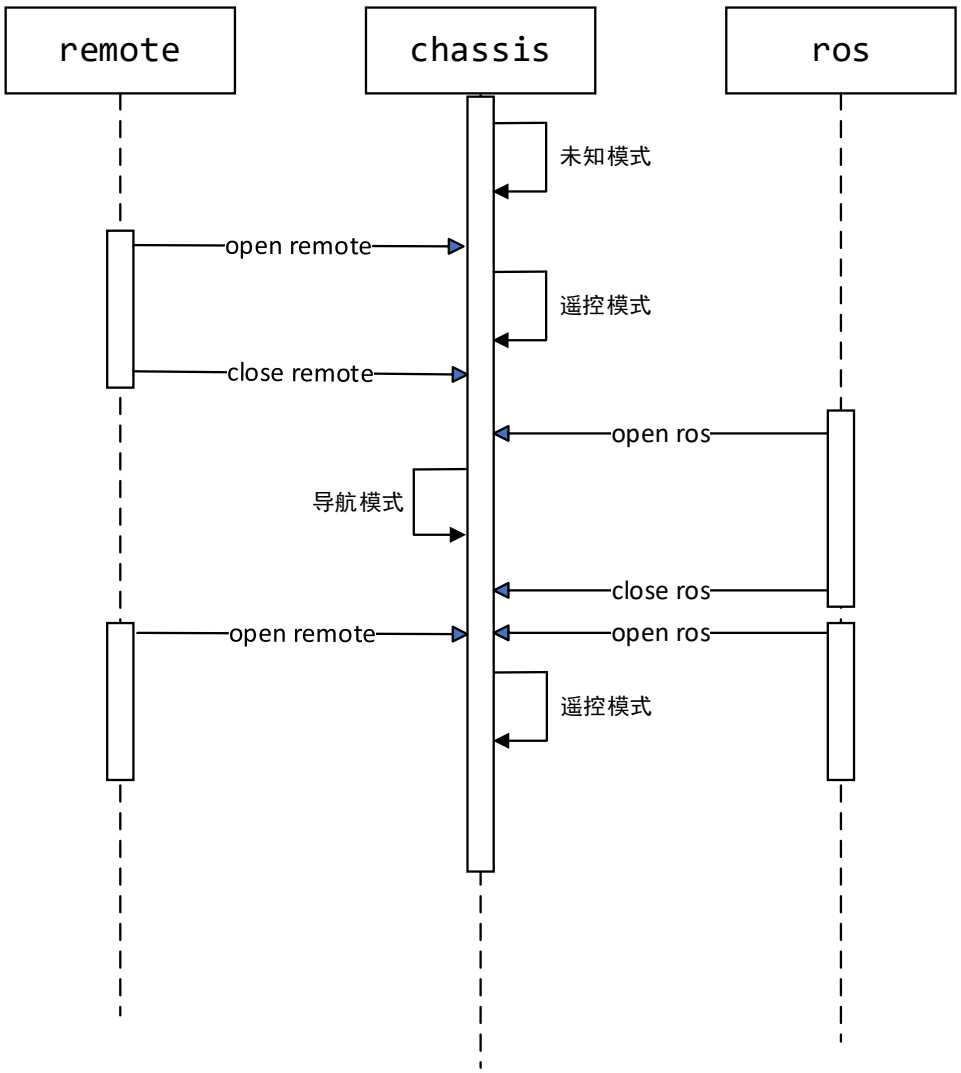

图 6 模式切换时序图

系统初始化后，remote_flag_、ros_flag_都为 0，机器人保持静止。遥控器开启后，触发串口 5 中断，进入UART5_IRQHandler()，接收遥控器发送的数据。然后对数据进行解析，并将标志位 remote_flag_置 1。上位机开启后，触发串口 2 中断，进入UART2_IRQHandler()，将标志位 ros_flag_置 1 上电后，循环执行各应用任务。

注意：遥控器优先级最高

3.3灯光控制模块 ledTask

机器人底盘主要用到的灯光为转向灯。目前控制左右转向灯比较粗糙，通过判断转向角速度来设置。当转向角速度为正时（这里为了消除抖动，避免灯光处于一直闪烁，设置了一个阈值 0.1），左转向灯闪烁；当转向灯速度为负时，右转向灯闪烁。默认灯是熄灭的。该任务以 5hz 频率执行。

3.4显示控制模块 showTask

4 存在问题

1) 执行数据收发任务时，上传的频率会逐渐变小，具体原因还在排查，后续对任务、中断占用 CPU 时间的时间进行查看
2) 遥控器开启一段时间后，切换为上位机控制，码表信息会丢失
3) 开机经常得开几次，继电器会跳，可能是电源板控制问题