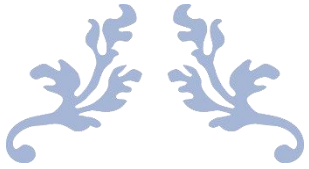


嵌入式技术文档

工程车

MSE-STAR

V2.0

占建

2020.8.18

目录

MSE-STAR

工程车技术文档

一、 需求与分析

1.1 战术地位

战场后勤，辅助位置，给全场己方机器人提供全方位辅助

1.2 战术功能

a. 基本运动功能

b. 救援功能

c. 取弹功能

d. 弹药补给功能

1.3 细节功能

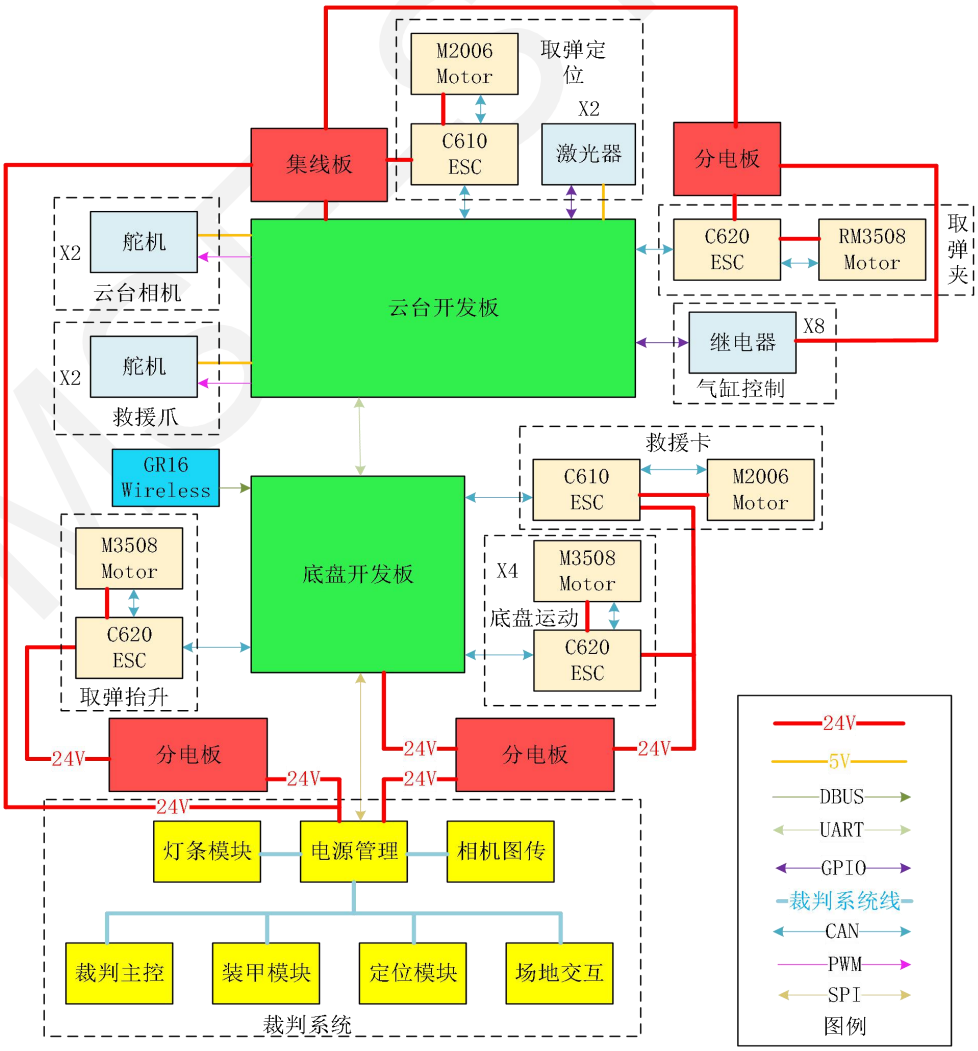
a. 自动取弹：能够一键操作，完成取弹流程，取弹方式包括：一字取弹，十字取弹，四角取弹等三种取弹模式

b. 电机角度控制：稳定的角度控制，主要在定位电机，抬升电机，取弹爪以及救援卡，防止堵转和便于控制

c. 复位：取弹机构在取弹完成后需要回到中心位置（初始位置），否则机械机构上将无法降下取弹机构

二、 电气架构设计

2.1 电气框图



2.2 框图分析

a. 定位模块是由 M2006 和两个激光组成。激光检测前方是否存在弹药箱，M2006 移动取弹爪到达指定位置

b. 取弹夹由 M3508 进行爪子位置的控制，爪子开合则是利用气缸达到目的

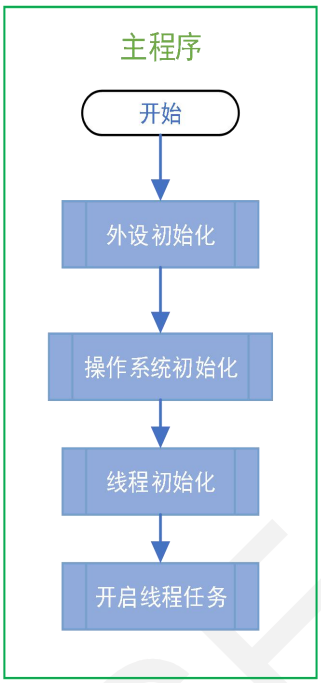
2.3 细节备注

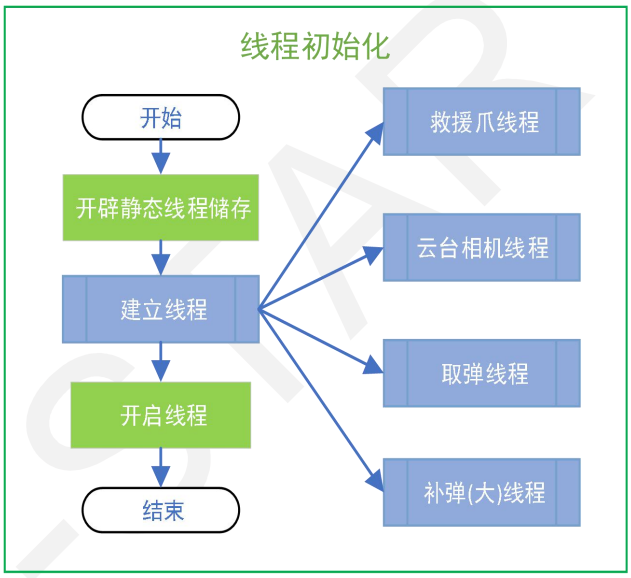
这一代的工程车机械爪利用的 M3508 由机械改造过，增加了减速装置，加大了力矩，所以角度控制的变换映射公式有所改变

三、 程序流程与思想

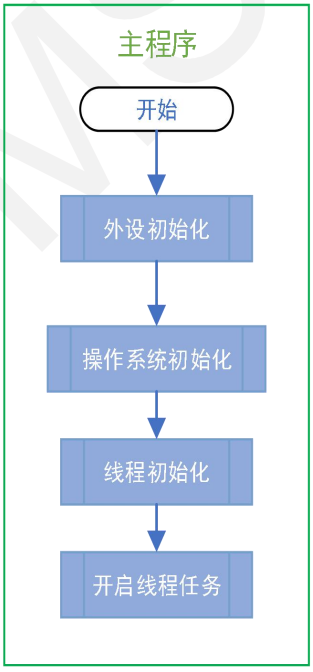
3.1 程序主框架

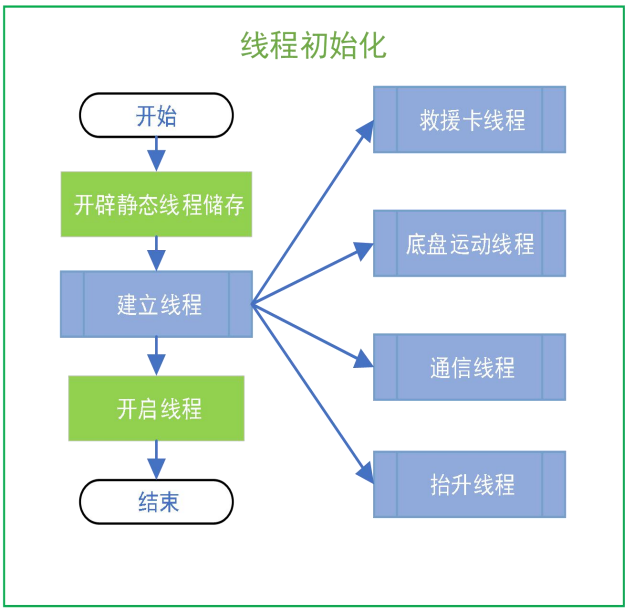
云台板主程序





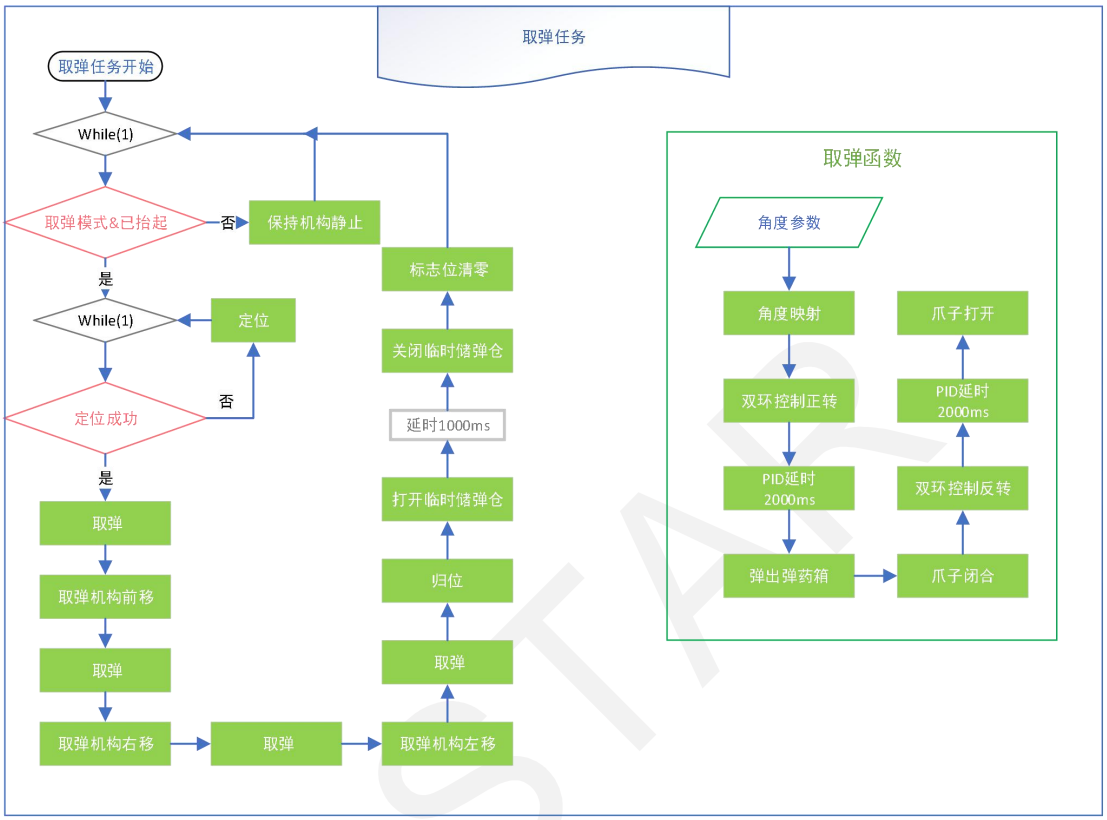
底盘板主程序

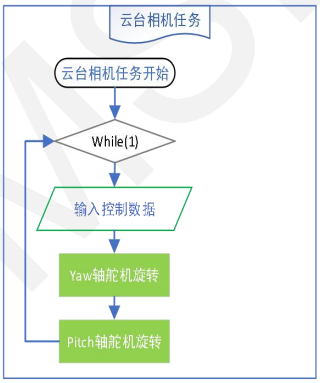


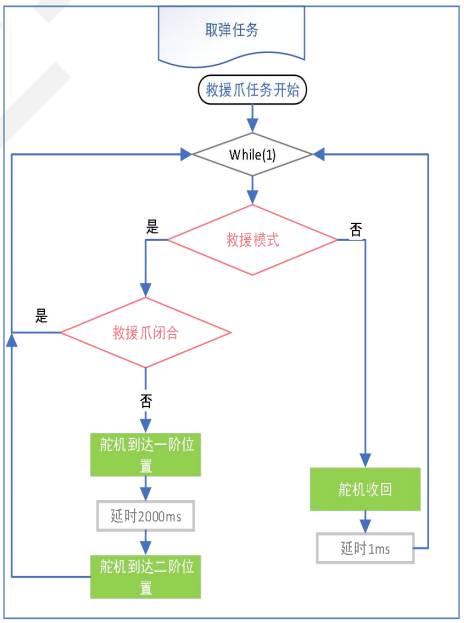


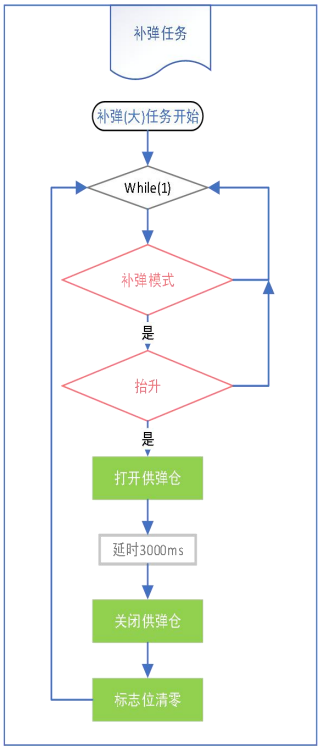
3.2 子程序流程图

云台任务子程序

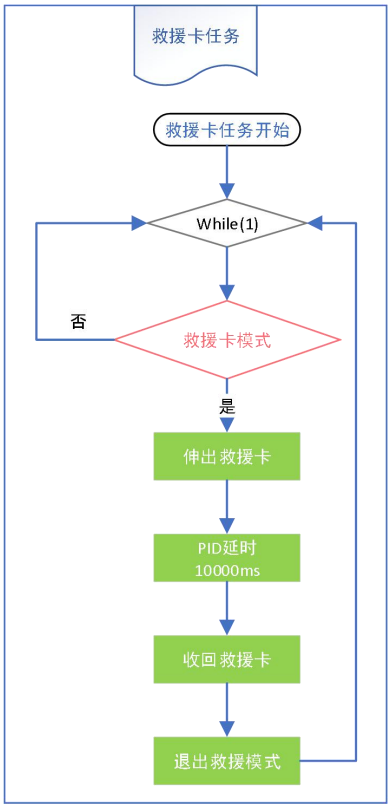


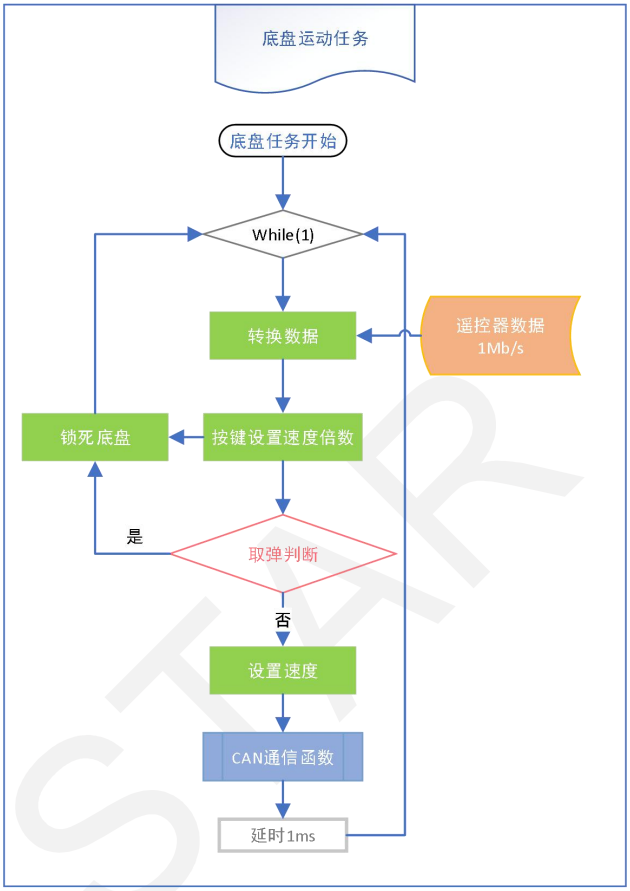


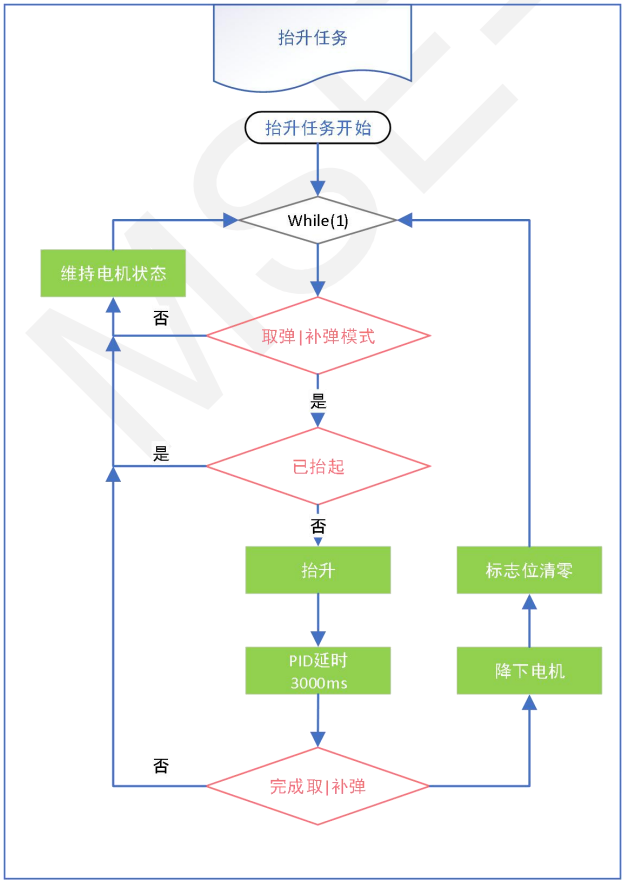


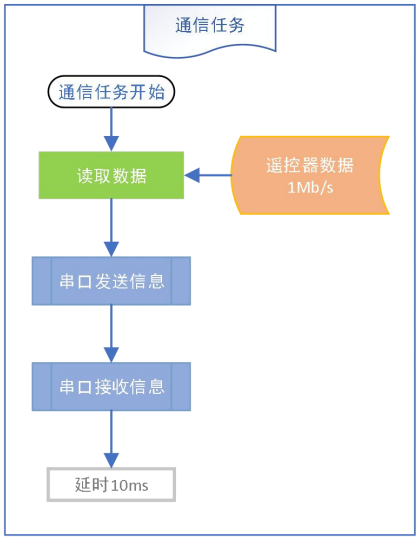


底盘任务子程序


救援卡任务
救援卡任务开始
While(1)
否
救援卡模式
是
伸出救援卡
PID延时
10000ms
收回救援卡
退出救援模式


底盘运动任务
底盘任务开始
While(1)
转换数据
遥控器数据
1Mb/s
锁死底盘
按键设置速度倍数
是
取弹判断
否
设置速度
CAN通信函数
延时1ms


抬升任务
抬升任务开始
While(1)
维持电机状态
否
取弹|补弹模式
是
是
已抬起
否
抬升
标志位清零
PID延时
3000ms
降下电机
否
完成取|补弹


通信任务
通信任务开始
读取数据
遥控器数据
1Mb/s
串口发送信息
串口接收信息
延时10ms

3.3 线程描述

3.3.1 主线程

底盘运动：接收遥控器信息进行运动分解，然后实现底盘运动操控

云台相机运动：接收来自底盘的信息，控制相机位置，实现视角移动

3.3.2 取弹线程

通过 M2006 和激光器定位，再借助气缸和 M3508 电机完成弹药箱的夹取，同时注意应该有三种取弹模式，目前实现一种取弹模式

3.3.3 救援线程

救援包括救援卡和救援爪，救援卡由底盘控制，救援爪由于机械结构的原因，由云台控制，救援爪在救援模式下都需要开启，抓牢被救援单位。救援爪在机械方面的设计导致其需要两段夹取。

3.3.4 补弹线程

补弹包括大弹丸小弹丸补弹，都需要将补弹机构抬起，与取弹线程相同，然后开始补弹。目前实现大弹丸补弹

3.4 中断服务函数描述

3.4.1 遥控器接收中断

接收操作手遥控器数据，控制底盘运动和得到云台信息

3.4.2 串口接收中断

工程车单位以底盘为控制核心，由他统一接收发送控制信息，云台底盘通过串口通信，由于算法原因，需要在串口接收中断中将数据按算法处理出结果。

四、 算法详叙

4.1 PID 算法

由于实际生活中能够实现的都是离散化的系统，所以只考虑离散 PID

离散化的 PID 公式（位置式）：

$$U(\mathrm{k}) = K_p(err(k) + \frac{T}{T_I}\sum err(k) + \frac{T_D}{T}(err(k) - err(k-1)))$$

一般记为

$$U(k) = K_P err(k) + K_i\sum err(k) + K_d(err(k) - err(k-1))$$

则有传统意义上的 P, I, D 三个参数量

另外离散化的 PID 公式（位置式）：

因为 $\Delta U(k) = U(k) - U(k-1)$

由位置式的公式可得

$$\Delta U(k) = K_p(err(k) - err(k-1)) + K_i err(k) + K_d(err(k) - 2err(k) + err(k-2))$$

V2.0 版本的代码都采取的是位置式 PID，便于理解运用

4.2 斜波函数

为了避免 PID 调节的响应速度和超调量的制约，同时为了保护电机防止瞬时电流过大，采取了斜波函数进行缓和

参考博客：

https://blog.csdn.net/u010632165/article/details/104729090/

MSE-STAR

工程车代码中有所更改，主要是变更采样时间为固定次数。

斜波函数主要就是将当前值与目标值分为多阶输出，从而达到缓冲目的，数据曲线的直观效果就是从阶跃变成斜波。

4.3 位操作处理串口数据

由于目前采取的是串口传输板间通信,然后就出现了收到的信息出现了乱序的问题，导致程序出问题。

依据传输数据的特点：9 个模式加 3 个标识符，加上串口只能够传输 char 型(8 位)的数据并且仅传一个 char 型的数据不用担心数据紊乱。

7	6	5	4	3	2	1	0
	MODE				UP	TAKE	RESCUE

所以利用位操作，将 8 位的前 3-6 位作为 9 个模式的存储位置，0-2 位为三个标识位储存位置

位 7　保留

位 6:3 MODE：将九个模式的二进制储存起来

0000：常规模式　0001：补弹模式　0010：登岛模式

0011：一字取弹　0100：四角取弹　0101：十字取弹

0110：救援爪模式　0111：打开爪子　1000：救援卡模式

位 2　UP：标识符，标识取弹机构是否已经抬起

由软件置 1 或清零

1：已抬升

0：未抬升

位 1　TAKE：标识符，标识取/补弹是否完成

由软件置 1 或清零

1：已完成

0：未完成

位 0　RESCUE：标识符，标识是否救援爪抓住目标单位

由软件置 1 或清零

1：已抓住

0：为抓住

五、 测试方案与数据记录

5.1 机械爪角度控制

测试方案：a.设置固定角度，看机械爪到达情况

b.遥控器控制，根据遥控器的数据来达到指定角度

数据：暂无

六、 未来技术展望

6.1 PID 算法的升级---ADRC

PID 算法存在着其自身的缺陷，所以出现新的调节算法--ADRC，其全称叫做 Active Disturbance Rejection Control，中文名是自抗扰控制技术。这项控制算法是由中科院的韩京清教授提出的。韩教授继承了经典 PID 控制器的精华，对被控对象的数学模型几乎没有任何要求，又在其基础上引入了基于现代控制理论的状态观测器技术，将抗干扰技术融入到了传统 PID 控制当中去，最终设计出了适合在工程实践中广泛应用的全新控制器。

MSE-STAR

ADRC 算法的参数较之 PID 增加至 6 个，一般来说，ADRC 控制器包括三个组件：跟踪微分器，非线性状态反馈（非线性组合），扩张观测器。

ADRC 是可以直接适配到使用了 PID 算法的地方的，所以可以考虑使用。

6.2 板间通信改为 CAN 通信

这个比较简单，目前使用是使用串口来进行板间通信，存在数据传输上的问题，所以考虑和组内其他单位统一，都采取 CAN 来通信。可能下一版本就会改成 CAN 通信

MSE-STAR

附录

- **部分源代码**

位操作代码

```
#define SetBit(x, y)    (x |= (1<<y))    // 特定位置 1,x 为目标,y 为第几位
#define ClearBit(x, y)  (x &= ~(1<<y))   // 特定位清 0
#define GetBit(x, y)    (x &= (1<<y))    // 特定位取值
#define ReveBit(x, y)   (x ^= (1<<y))    // 特定位取反
/**
    * @brief  将信息存入一个字符
    * @note
    * @author 占建
    * @param  int mode           模式
              int up_finish      抬升标识位
              int take_finish    取弹标识位
              int rescue         救援标识位
    * @retval
    */
uint8_t deal_masge_put(int mode, int up_finish, int take_finish, int rescue)
{
    int postion =0;
    uint8_t masge=0;
    /*处理 mode,将 mode 放在 3-6 bit 位上*/
        for(int i=0;i<4;i++)
        {
            postion = i+3;//3-6 位作为 mode 的位置
            if(mode%2==1)
            SetBit(masge,postion);
            else
            ClearBit(masge,postion);
            mode/=2;
        }
    /*处理 up_finish,在 2bit 位*/
        if(up_finish == 1)
            SetBit(masge,2);
        else
            ClearBit(masge,2);
    /*处理 take_finish,在 1bit 位*/
        if(take_finish == 1)
            SetBit(masge,1);
        else
            ClearBit(masge,1);
    /*处理 rescue,在 0bit 位*/
        if(rescue == 1)
```

MSE-STAR

```
                SetBit(masge,0);
            else
                ClearBit(masge,0);
      return masge;
}

/**
    * @brief  将信息取出
    * @note
    * @author 占建
    * @param  uint8_t masge  待处理信息
    * @retval
    */
void deal_masge_get(uint8_t masge)
{
    int postion =0,copy_mode=0;
    char copy;

        copy=masge;
        receive_masge[0]= GetBit(copy,0);

        copy=masge;
        receive_masge[1]= GetBit(copy,1)/2;

        copy=masge;
        receive_masge[2]= GetBit(copy,2)/4;
    /*得到 mode*/
    for(int i=0;i<4;i++)
    {
        copy=masge;
        postion=i+3;
        copy_mode+= GetBit(copy,postion)/8;
    }
    receive_masge[3]= copy_mode;
}
```

其他源码部分参考 GitHub 库：
https://github.com/HUST-MSE-STAR/Engineer

MSE-STAR