ROBOMASTER 2022

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY 学生创新中心 Student Innovation Center

机甲大师校内赛

第二次电控培训

2021.10.23

上海交通大学
RM2022校内赛
电控第二次培训

① 框架解读

② 物资认识

③ 调试指南

④ 安全问题

如何阅读框架

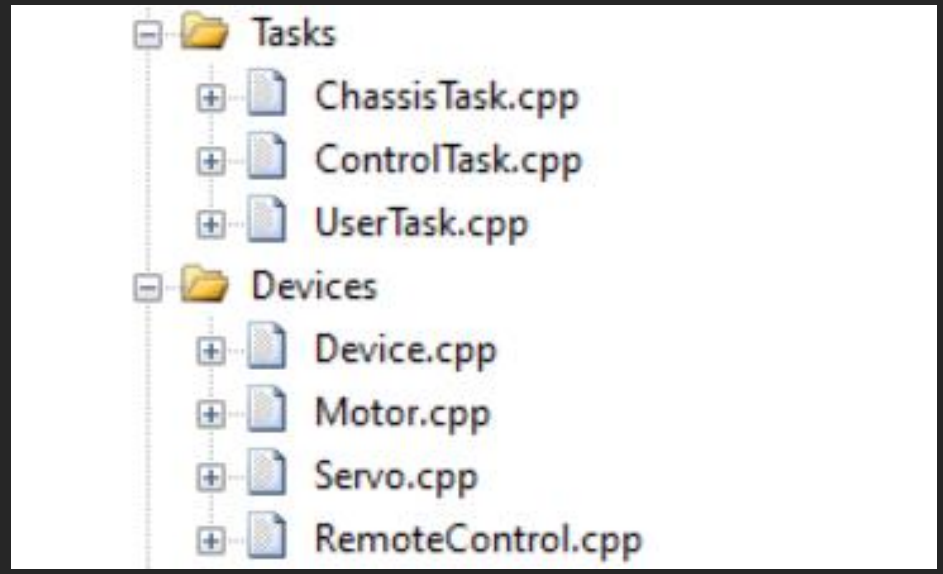
如何阅读框架

ROBOMASTER

0.1 Keil项目的文件树

Tasks：包含具体指挥小车行为的任务

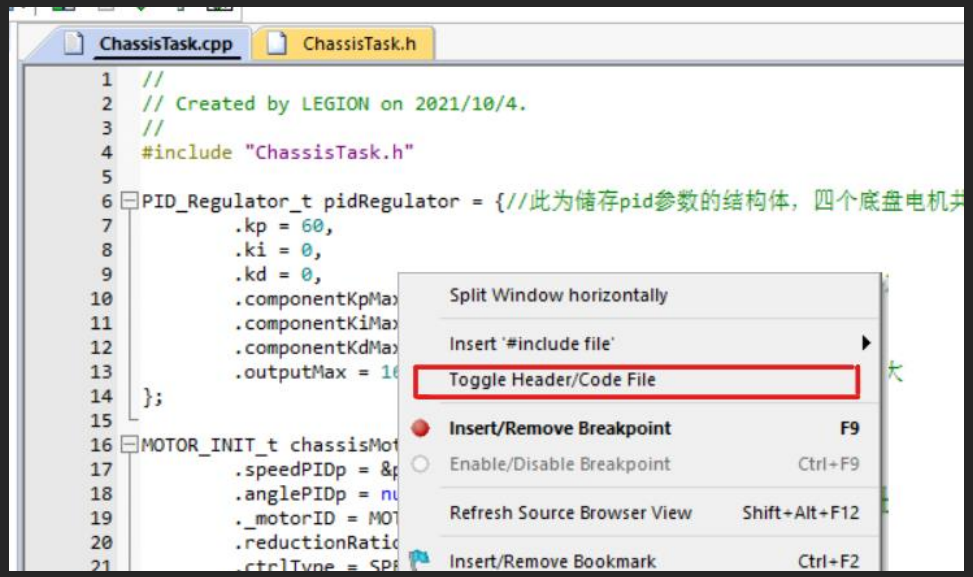
Devices：小车上涉及到的设备类的实现

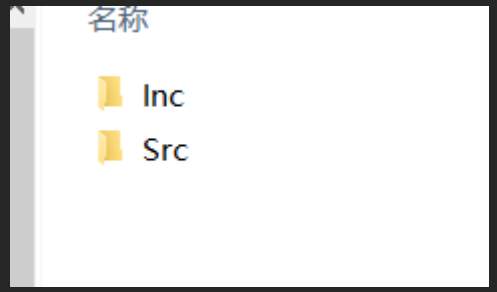


如何阅读框架

ROBOMASTER

0.2 在源文件和头文件之间快速切换





Inc：Include的缩写，存放项目的头文件，具体包括相关函数，变量，结构体，类的声明。

Src：Source的缩写，存放项目的源文件，存放相关实例的实现。

如何阅读框架

ROBOMASTER

举例：

```
/*类型定义------------------------------------------------------------
class Servo :public Device
{
    static uint32_t servo_IDs;
    static void Init();

    uint8_t stopFlag{1};
    SERVO_TYPE_E servoType;

    float angleLimit_Min,angleLimit_Max;

    float duty{0};
    float targetAngle{0};
    float targetSpeed{0};

    void SetTargetSpeed(float _targetSpeed);

    void ErrorHandle() override;


public:
    explicit Servo(SERVO_INIT_T* servoInit);

    void stop();

    void SetTargetAngle(float _targetAngle);

    void Handle() override;
};
```

Servo.h

```
    }
    auto pos = GET_SERVO_POS(deviceID);

    __HAL_TIM_SET_COMPARE(servoInfo[pos].handleTypeDef,servoInfo[pos].timChannel,
            htim1.Instance->ARR*duty);

}

void Servo::stop() {
    if (stopFlag == 0){
        stopFlag = 1;
        auto pos = GET_SERVO_POS(deviceID);
        HAL_TIM_PWM_Stop(servoInfo[pos].handleTypeDef,servoInfo[pos].timChannel);
    }
}
/**
 * @brief 设置舵机角度
 * @param _targetAngle
 */
void Servo::SetTargetAngle(float _targetAngle) {
    //TODO 检查舵机类型,检查角度限位
    if(stopFlag == 1){
        stopFlag = 0;
        auto pos = GET_SERVO_POS(deviceID);
        HAL_TIM_PWM_Start(servoInfo[pos].handleTypeDef,servoInfo[pos].timChannel);
    }
    INRANGE(_targetAngle,angleLimit_Min,angleLimit_Max);
    targetAngle = _targetAngle;
}
/**
```

Servo.cpp

如何阅读框架

ROBOMASTER

举例：

```
void SetTargetAngle(float _targetAngle);
```

在头文件中声明舵机类的成员函数：SetTatgetAngle。无返回值，有一个float类型的参数，函数声明为public

```
void Servo::SetTargetAngle(float _targetAngle) {
    //TODO 检查舵机类型,检查角度限位
    if(stopFlag == 1){
        stopFlag = 0;
        auto pos = GET_SERVO_POS(deviceID);
        HAL_TIM_PWM_Start(servoInfo[pos].handleTypeDef,servoInfo[pos].timChannel);
    }
    INRANGE(_targetAngle,angleLimit_Min,angleLimit_Max);
    targetAngle = _targetAngle;
}
/**
```

在源文件中实现SetTargetAngle，返回值和参数类型必须和声明时相同。

声明为public的函数可以通过对象访问（例如myServo.SetTargetAngle),类的成员函数和变量默认为private，只能被类的成员函数访问。

ROBOMASTER

0.3 快速跳转到实例的声明或实现

```
void CtrlHandle(){
    if (RemoteControl::rcInfo.sRight == DOWN_POS){//右侧三档，急停模式
        ChassisStop();
        UserStop();
    }else {//其他正常模式
        switch (RemoteControl::rcInfo.sLeft) {
            case UP_POS://左侧一档
                ChassisSetVelocity(RemoteControl::rcInfo.right_col*4.2,
                                   RemoteControl::rcInfo.right_rol*4.2,RemoteControl::rcInfo.left_rol*60);
                break;
            case MID_POS://左侧二档
                uint8_t clawState;
                if (RemoteControl::rcInfo.sRight == UP_POS){
                    clawState = 0;
                }else if(RemoteControl::rcInfo.sRight == MID_POS) {
                    clawState = 1;
                }
                break;
            case DOWN_POS:default:
                break;
        }

    }

}
```

如何阅读框架

ROBOMASTER

0.3 快速跳转到实例的声明或实现

```
void CtrlHandle(){
    if (RemoteControl::rcInfo.sRight == DOWN_POS){//右侧三档，急停模式
        ChassisStop();
        UserStop();
    }else {//其他正常模式
        switch (RemoteControl::rcInfo.sLeft) {
            case UP_POS://左侧一档
                ChassisSetVe
                                                          l::rcInfo.left_rol*60);
                break;
            case MID_POS://左
                uint8_t claw
                if (RemoteCo
                    clawStat
                }else if(Rem
                    clawStat
                }
                break;
            case DOWN_POS:de
                break;
        }

    }

}
```

Split Window horizontally

Insert '#include file'

Toggle Header/Code File

Insert/Remove Breakpoint F9

Enable/Disable Breakpoint Ctrl+F9

Refresh Source Browser View Shift+Alt+F12

Go To Definition of 'ChassisSetVelocity' F12

Go To Declaration of 'ChassisSetVelocity' Ctrl+F12

Go To Next Reference of 'ChassisSetVelocity' Ctrl+Num +

Go To Previous Reference of 'ChassisSetVelocity' Ctrl+Num -

Show All References of 'ChassisSetVelocity' Ctrl+Shift+F12

如何阅读框架

ROBOMASTER

0.3 快速跳转到实例的声明或实现

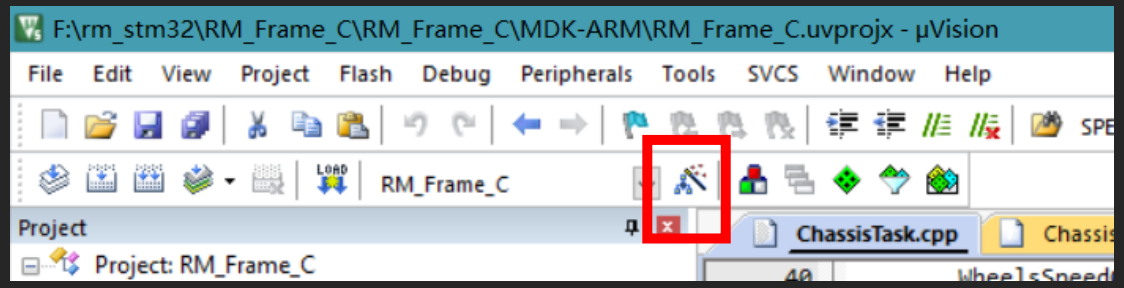
```
45          CMBR.Handle();
46      }
47      /**
48       * @brief 用于控制任务控制底盘速度
49       * @param _fbV 底盘前后方向速度
50       * @param _lrV 底盘左右方向速度
51       * @param _rtV 底盘旋转速度
52       */
53      void ChassisSetVelocity(float _fbV,float _lrV,float _rtV){
54          ChassisStopFlag = 0;
55          FBVelocity = _fbV;
56          LRVelocity = _lrV;
57          RTVelocity = _rtV;
58      }
59      /**
60       * @brief 执行急停模式的底盘任务处理
61       */
62      void ChassisStop(){
```

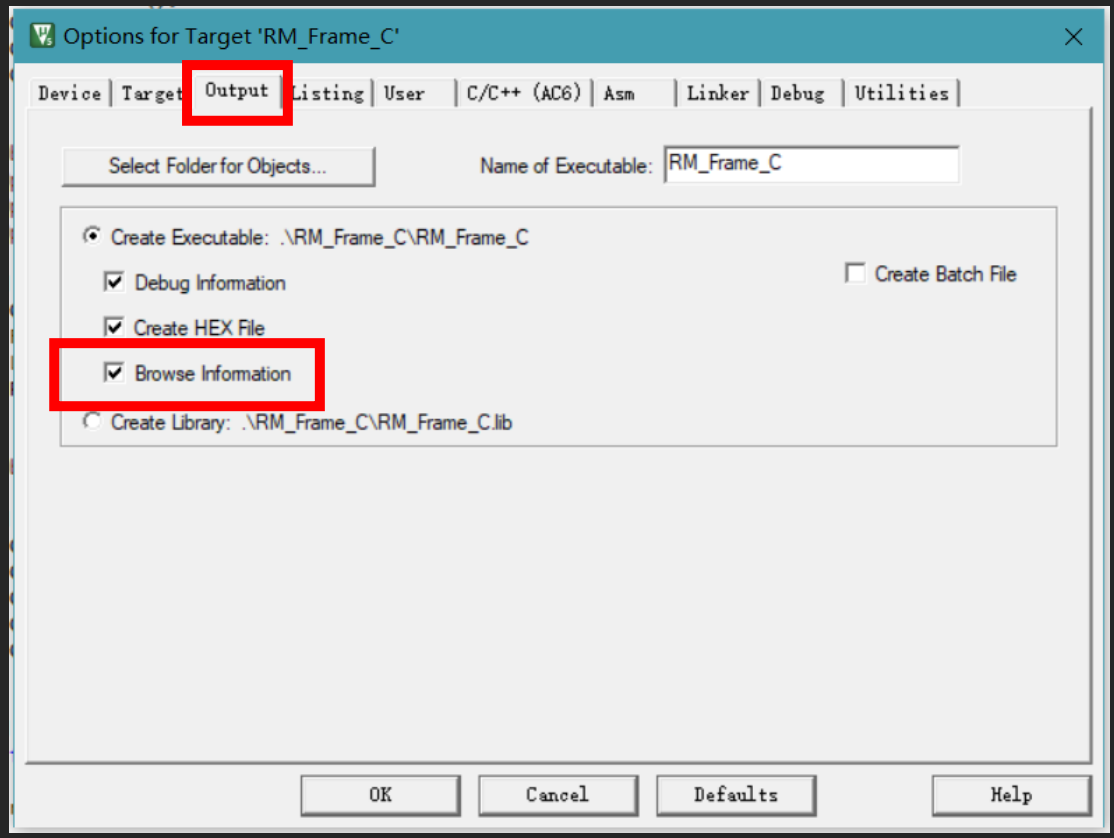
如何阅读框架

ROBOMASTER

0.3 快速跳转到实例的声明或实现

此功能依赖于选项中的Browse Infomation，在勾选之后正确编译，就可以使用。



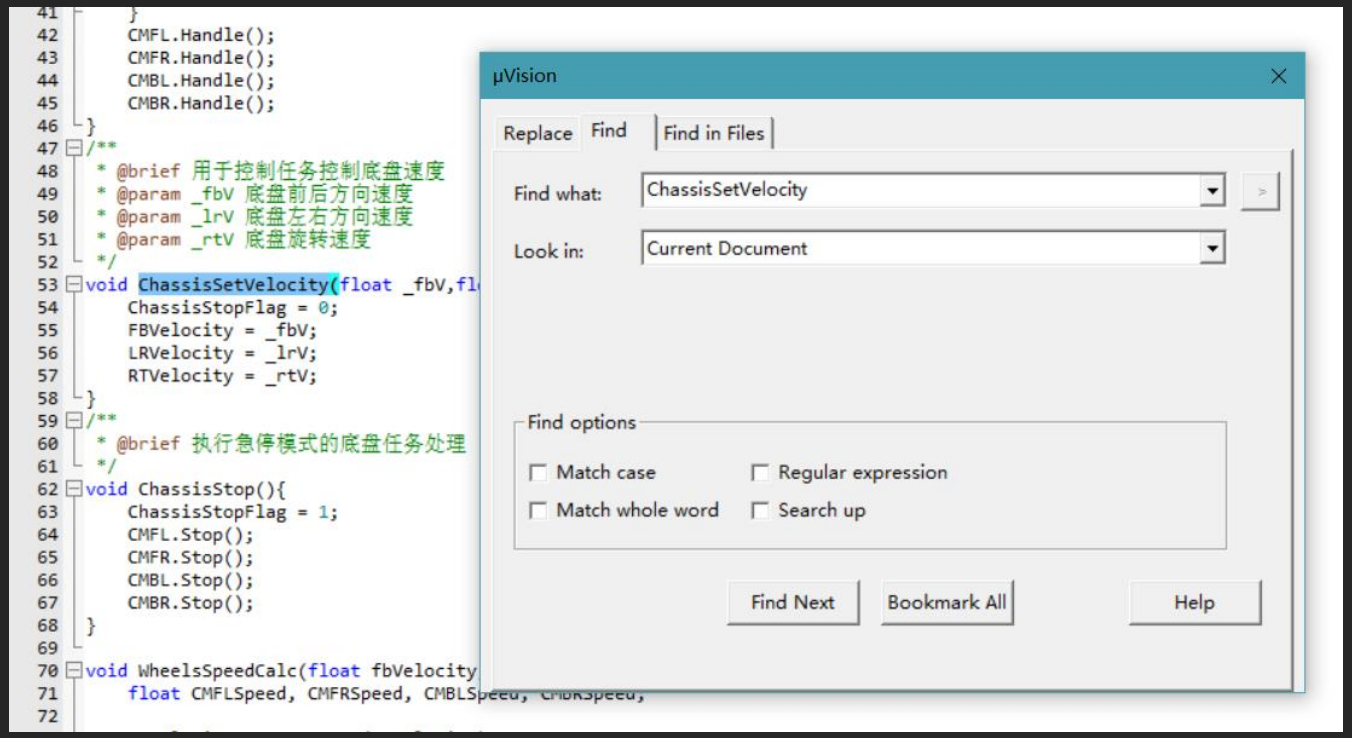


如何阅读框架

ROBOMASTER

0.4 搜索和全局搜索

快捷键是Ctrl + F，可以先双击某个实例，在所选实例高亮后按下搜索快捷键。可以自动输入选定的内容，点击Find Next即可自动跳转并高亮。

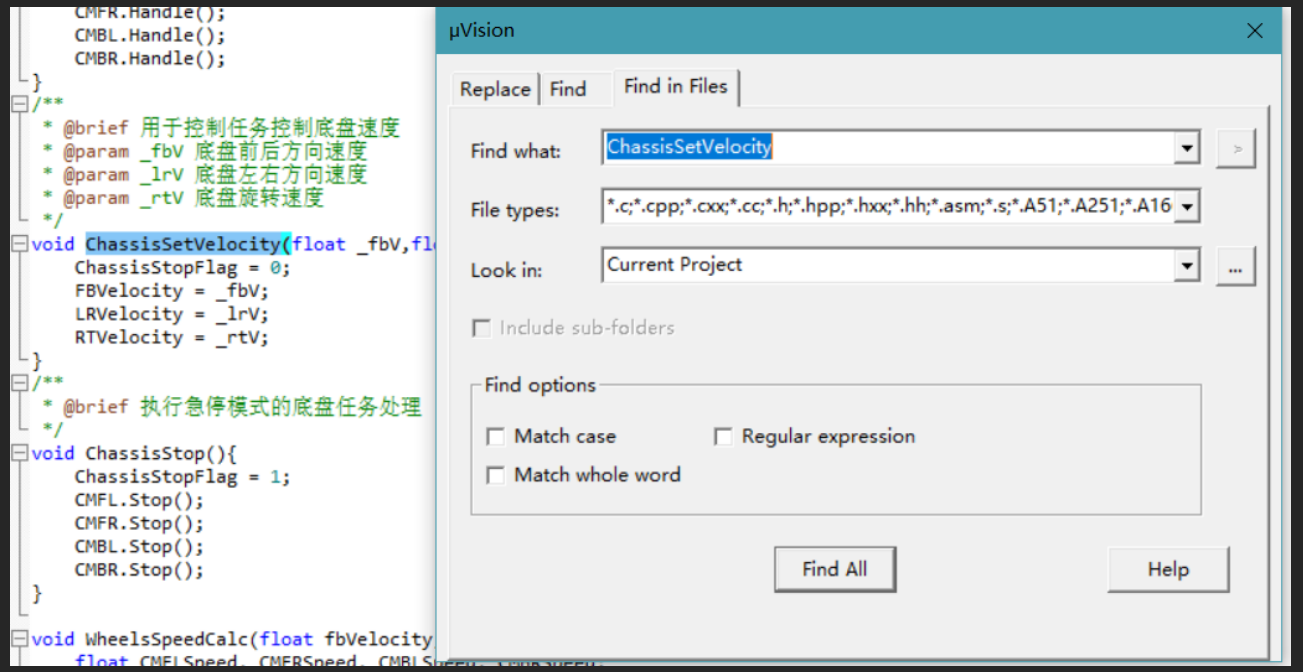


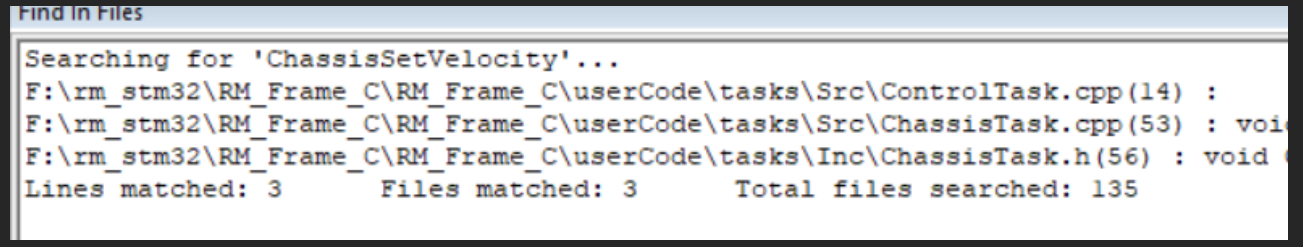
如何阅读框架

ROBOMASTER

0.4 搜索和全局搜索

快捷键是Ctrl + F，操作同前。
区别在于搜索结果会集中显示在下部，点击可以调整。
另外Ctrl + R有替换功能，使用方法略。





电控框架解读

电控框架解读

1.1 任务流程梳理

1.2 设备类详解

1.3用户任务编写建议

1.1 任务流程梳理

底盘任务根据控制任务传入的控制量，计算四个轮子的速度，利用电机类接口控制电机速度。

```
Motor CMFL(MOTOR_ID_1,&chassisMotorInit);//定义左前轮电机
Motor CMFR(MOTOR_ID_2,&chassisMotorInit);//定义右前轮电机
Motor CMBL(MOTOR_ID_3,&chassisMotorInit);//定义左后轮电机
Motor CMBR(MOTOR_ID_4,&chassisMotorInit);//定义右后轮电机

uint8_t ChassisStopFlag = 1; //用于判断底盘是否处于急停模式的标志
float FBVelocity,LRVelocity,RTVelocity;//底盘三个方向上的速度
```

ROBOMASTER

1.1 任务流程梳理

```
void ChassisStart(){
```

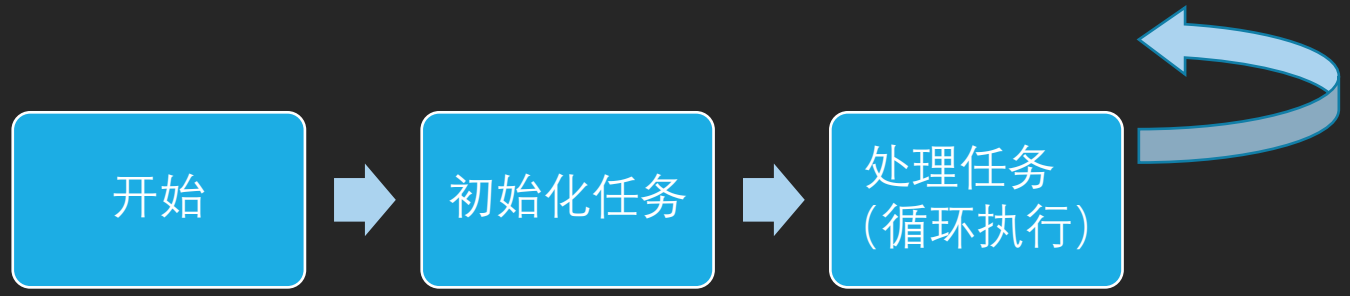
```
/**
 * @brief 底盘任务的处理函数，定时执行
 * @callergraph void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) in Device.cpp
 */
void ChassisHandle() {
```

```
/**
 * @brief 用于控制任务控制底盘速度
 * @param _fbV 底盘前后方向速度
 * @param _lrV 底盘左右方向速度
 * @param _rtV 底盘旋转速度
 */
void ChassisSetVelocity(float _fbV,float _lrV,float _rtV){
```

```
/**
 * @brief 执行急停模式的底盘任务处理
 */
void ChassisStop(){
```

ROBOMASTER

1.1 任务流程梳理




```
void ChassisHandle() {
    if(ChassisStopFlag == 0) {
        WheelsSpeedCalc(FBVelocity, LRVelocity, RTVelocity);
    }
    CMFL.Handle();
    CMFR.Handle();
    CMBL.Handle();
    CMBR.Handle();
}
/**
```

1.1 任务流程梳理

```
void WheelsSpeedCalc(float fbVelocity, float lrVelocity, float rtVelocity) {
    float CMFLSpeed, CMFRSpeed, CMBLSpeed, CMBRSpeed;

    rtVelocity = RPM2RADpS(rtVelocity);

    //计算四个轮子线速度，单位：m/s
    /**
     * @brief 此处四句代码需要结合底盘的三个速度，计算处四个轮子的位置对应的线速度。
     * @param fbVelocity,lrVelocity,rtVelocity
     * @return CMFLSpeed CMFRSpeed CMBLSpeed CMBRSpeed
     */
    CMFLSpeed = 0;
    CMFRSpeed = 0;
    CMBLSpeed = 0;
    CMBRSpeed = 0;

    //计算四个轮子角速度，单位：rad/s
    CMFLSpeed = CMFLSpeed /(WHEEL_DIAMETER/2.0f);
    CMFRSpeed = CMFRSpeed /(WHEEL_DIAMETER/2.0f);
    CMBLSpeed = CMBLSpeed /(WHEEL_DIAMETER/2.0f);
    CMBRSpeed = CMBRSpeed /(WHEEL_DIAMETER/2.0f);
    //控制底盘电机转速
    CMFL.SetTargetSpeed(RADpS2RPM(CMFLSpeed));
    CMFR.SetTargetSpeed(RADpS2RPM(CMFRSpeed));
    CMBL.SetTargetSpeed(RADpS2RPM(CMBLSpeed));
    CMBR.SetTargetSpeed(RADpS2RPM(CMBRSpeed));
}
```

1.1 任务流程梳理

```
//计算四个轮子线速度，单位：m/s
/**
 * @brief 此处四句代码需要结合底盘的三个速度，计算处四个轮子的位置对应的线速度。
 * @param fbVelocity,lrVelocity,rtVelocity
 * @return CMFLSpeed CMFRSpeed CMBLSpeed CMBRSpeed
 */
CMFLSpeed = 0;
CMFRSpeed = 0;
CMBLSpeed = 0;
CMBRSpeed = 0;
```

$$\begin{cases} v_1 = v_y - v_x + a \cdot \omega + b \cdot \omega \\ v_2 = v_y + v_x - a \cdot \omega - b \cdot \omega \\ v_3 = v_y - v_x - a \cdot \omega - b \cdot \omega \\ v_4 = v_y + v_x + a \cdot \omega + b \cdot \omega \end{cases}$$

1.1 任务流程梳理

控制任务借助遥控器设备读入的数值，按照特定的逻辑，为其他设备传入控制值。

```
void CtrlHandle(){
    if (RemoteControl::rcInfo.sRight == DOWN_POS){//右侧三档，急停模式
        ChassisStop();
        UserStop();
    }else {//其他正常模式
        switch (RemoteControl::rcInfo.sLeft) {
            case UP_POS://左侧一档
                ChassisSetVelocity(RemoteControl::rcInfo.right_col*4.2,
                                   RemoteControl::rcInfo.right_rol*4.2,RemoteControl::rcInfo.left_rol*60);
                break;
            case MID_POS://左侧二档
                uint8_t clawState;
                if (RemoteControl::rcInfo.sRight == UP_POS){
                    clawState = 0;
                }else if(RemoteControl::rcInfo.sRight == MID_POS) {
                    clawState = 1;
                }
                break;
            case DOWN_POS:default:
                break;
        }

    }

}
```

1.2 设备类详解

Device.cpp:
提供设备基类，包含部分中断函数和main函数。一般情况下不需要改动。

Motor.cpp:
提供电机类，可用于控制官方物资中的3508和2006电机。

1.2 设备类详解

Motor.cpp:
提供电机类，可用于控制官方物资中的3508和2006电机。

1.2 设备类详解

Motor.cpp:

```
uint8_t stopFlag{1}; //电机急停标志位
C6x0Rx_t feedback;  //电调返回值结构体
PID speedPID,anglePID;  //PID类对象
MOTOR_STATE_t state;    //电机状态结构体
MOTOR_CTRL_TYPE_e ctrlType; //电机控制类型枚举量

float targetSpeed;  //电机目标速度，仅控制类型为单环电机时有效
float targetAngle;  //电机目标角度，仅控制类型为双环时有效
float reductionRatio;   //电机减速比
```

1.2 设备类详解

Motor.cpp:

```
PID_Regulator_t pidRegulator = {//此为储存pid参数的结构体，四个底盘电机共用
        .kp = 60,
        .ki = 0,
        .kd = 0,
        .componentKpMax = 16384,
        .componentKiMax = 0,
        .componentKdMax = 0,
        .outputMax = 16384 //3508电机输出电流上限，可以调小，勿调大
};

MOTOR_INIT_t chassisMotorInit = {//四个底盘电机共用的初始化结构体
        .speedPIDp = &pidRegulator,
        .anglePIDp = nullptr,
        ._motorID = MOTOR_ID_1,
        .reductionRatio = 19.0f,
        .ctrlType = SPEED_Single,
};
```

1.2 设备类详解

Motor.cpp:

```
/**
 * @brief 用于设置电机速度，只用当电机控制类型对应时才有效
 * @param _targetSpeed 目标速度
 */
void Motor::SetTargetSpeed(float _targetSpeed) {
    stopFlag = 0;
    targetSpeed = _targetSpeed;
}
/**
 * @brief 用于设置电机角度，只用当电机控制类型对应时才有效
 * @param _targetAngle 目标角度
 */
void Motor::SetTargetAngle(float _targetAngle) {
    stopFlag = 0;
    targetAngle = _targetAngle;
}
```

ROBOMASTER

1.2 设备类详解

双环电机PID的定义

```
PID_Regulator_t userPidRegulator = {
        .kp = 60,
        .ki = 0,
        .kd = 0,
        .componentKpMax = 10000,
        .componentKiMax = 0,
        .componentKdMax = 0,
        .outputMax = 10000 //2006电机输出电流上限，可以调小，勿调大
};

PID_Regulator_t userPidAngleRegulator = {
        .kp = 10,
        .ki = 0,
        .kd = 0,
        .componentKpMax = 100,
        .componentKiMax = 0,
        .componentKdMax = 0,
        .outputMax = 1000
};

MOTOR_INIT_t userMotorInit = {
        .speedPIDp = &userPidRegulator,
        .anglePIDp = &userPidAngleRegulator,
        ._motorID = MOTOR_ID_1,
        .reductionRatio = 36.0f,
        .ctrlType = POSITION_Double,
};

Motor UserMotor(MOTOR_ID_5,&userMotorInit);
```

1.2 设备类详解

Servo.cpp:

舵机类可以用于控制常见的180度舵机，也可以通过简单的改写实现对其他类型舵机的控制。

```
typedef struct{
    SERVO_TYPE_E servoType;
    uint32_t servoID;
    float firstAngle,angleLimit_Min,angleLimit_Max;//only valid for POSITION_180 servo
}SERVO_INIT_T;
```

1.2 设备类详解

```
/**
 * @brief 舵机的构造函数，进行舵机的初始化设置
 * @param servoInit 舵机的初始化结构体指针
 */
Servo::Servo(SERVO_INIT_T *servoInit) {
    deviceID = servoInit->servoID;
    deviceType = SERVO;

    angleLimit_Min = servoInit->angleLimit_Min;
    angleLimit_Max = servoInit->angleLimit_Max;

    servo_IDs |= deviceID;
    servoType = servoInit->servoType;
    float duty;
    if(servoType == POSITION_180) {
        duty = AFFINE(0,180,0.025,0.125,servoInit->firstAngle);
    }else if(servoType == SPEED_360){
        duty = 0.075;
    }
    auto pos = GET_SERVO_POS(deviceID);

    __HAL_TIM_SET_COMPARE(servoInfo[pos].handleTypeDef,servoInfo[pos].timChannel,
            htim1.Instance->ARR*duty);

}
```

1.2 设备类详解

RemoteControl.cpp
通过串口接受遥控器接收机的信号并解算，得到对应的数据

```
static RC_Info_t rcInfo;

float left_col,left_rol;//左摇杆横纵位置的相对值
float right_col,right_rol;//右摇杆横纵位置的相对值

float dialWheel;//拨轮位置的相对值

SWITCH_STATE_E sLeft;//左侧开关位置的枚举量
SWITCH_STATE_E sRight;//右侧开关位置的枚举量
```

1.3 用户任务编写建议

1，复习第一次培训内容，补全底盘任务中未完成的部分。
2，学习双环电机的定义方式，在用户任务中定义双环2006，参考底盘任务调用新电机的处理函数和急停函数。
3，学习180度舵机的定义方式，在用户任务中定义。
4，编写测试用控制任务，在官方车上调试验证（或者催机械出车）
5，按需求对框架的设备类扩充
6，编写最终方案的用户任务和测试任务
7，调节电机参数和其他参数

电控物资介绍

电控物资介绍

有关官方物资的详细资料可以在官网上下载

2.1 烧写器

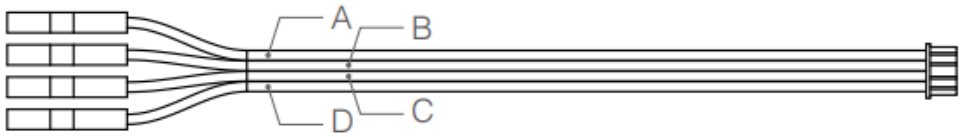
发放的烧写器（stlink），用于将程序烧录进开发板。连接时不使用袋内附带的杜邦线，而是使用另外发放的烧写线。

使用前需要先安装驱动。

若临时出现无法识别可以尝试重新插拔。

2.1 烧写器

SWD 下载线线序



线长 100mm，线序从上到下依次为：

A 黑色（SWDIO），B 黑色（SWCLK），C 黑色（GND），D 黑色（3.3V）

烧写器外壳丝印上已标明线序，左侧的白色方块代表外壳上的缺口，其余线序按顺序接即可。如开发板已有电池供电，则3.3V无需连接。**严防**GND错接至3.3V或5V，可能造成**开发板或烧写器烧毁**！

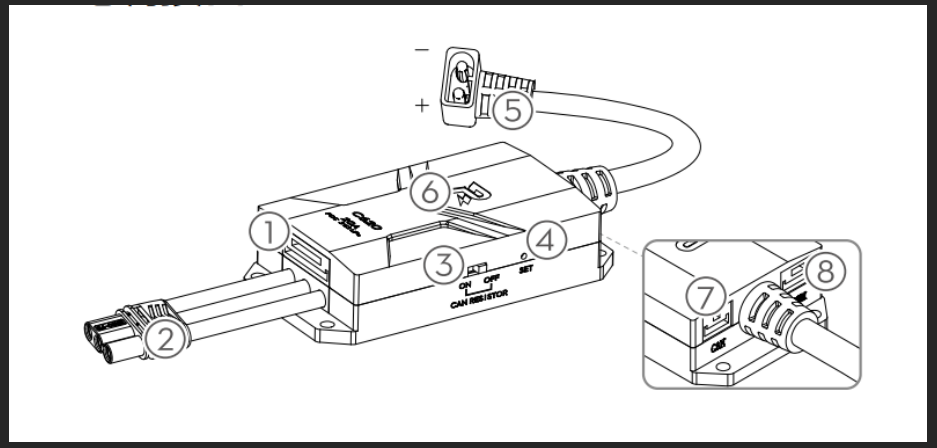
ROBOMASTER

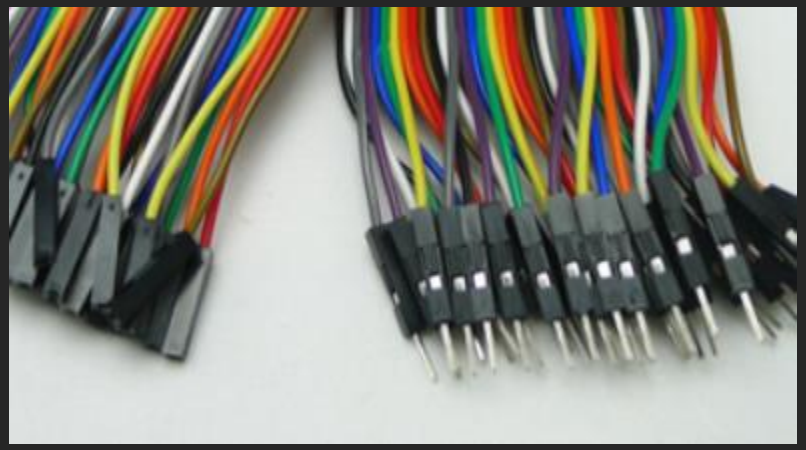
2.2 电机

3508电机和C620电调配套，2006和C610配合使用。
设置ID方法详见电调说明书，其中C620需要细长物体才可以戳入小孔，可以使用取卡针或杜邦线公头。

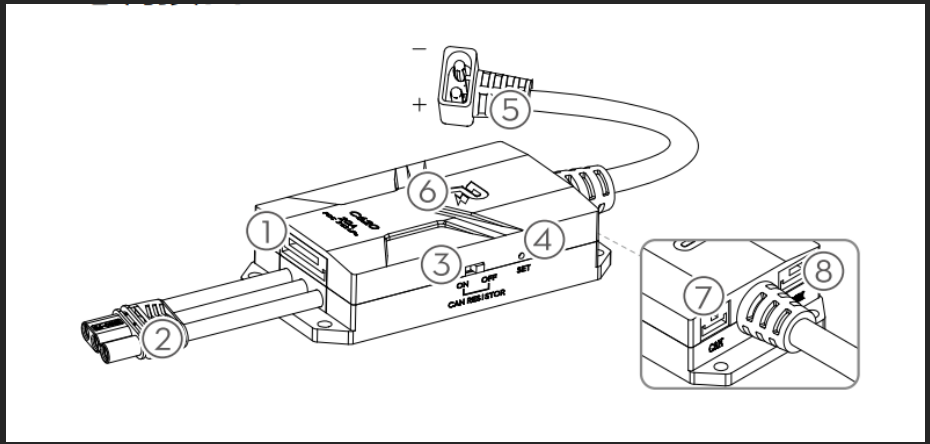
用户对单个 C620 电调进行 ID（支持范围 1-8）设置，具体操作如下：

a. 电调正常工作状态下，短按 1 次 SET 按键，进入独立设置 ID 模式，此时指示灯熄灭。
b. 在独立设置 ID 模式下，短按 SET 按键的次数（不超过 8 次）即为设置的 ID 号。每次有效短按，指示灯将橙灯闪烁 1 次。
c. 若 3 秒未对 SET 按键进行操作，电调将自动保存当前设置 ID 号。设置完 ID 的电调需要重新上电才能进入正常工作状态。





2.2 电机



[1] 7-Pin 电机数据端口
连接 M3508 直流无刷减速电机进行数据交互。
[2] 三相动力线接头
与 M3508 直流无刷减速电机的三相输入接头相连接，连接时请确保电调与电机连线正确（相同颜色的接线匹配连接，并且保证不可逆接头正确匹配连接），切勿接错。

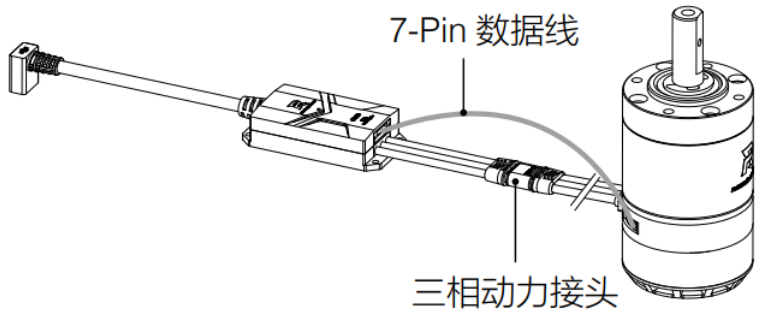
[5] 电源线
连接电源（额定电压 24V）为 C620 电调供电，电源线插头兼容 XT30。
[6] 指示灯
指示当前电调的工作状态，详见“指示灯描述”。
[7] CAN 信号端口
通过 CAN 信号线连接该端口，接收控制板的 CAN 控制指令，CAN 总线比特率为 1Mbps。

2.2 电机

电机电调连线

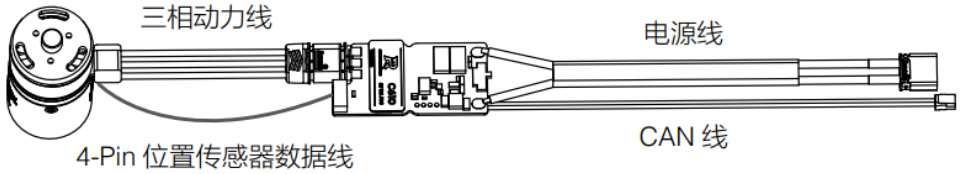
下面以搭配 RoboMaster C620 电调为例，介绍连线方式：

1. 用 7-Pin 数据线分别插入电调和电机的 7-Pin 数据端口，连接电调和电机。
2. 将电机的三相输入接头与电调三相动力线接头相连接，连接时请确保电调与电机连线正确（相同颜色的接线匹配连接，并且保证不可逆接头正确匹配连接），切勿接错。



电机电调连线

下面以搭配 C610 电调为例，介绍连线方式：



1. 将电机的 4-Pin 位置传感器数据线与 C610 电调的 4-Pin 位置传感器数据端口相连接。
2. 将电机的三相动力线与 C610 电调三相动力接头相连接，连接时请确保电调与电机连接正确（保证不可逆接头正确匹配连接），切勿接错。

2.3 中心板

四个GH1.25插口，相互并联，
用于can线连接电调
四个XT30母头，相互并联，
用于给电调供电
一个XT60公头，连接电池座
可以为整块中心板取电
一个8pin插口，用于给主控板
供电和传递can信号

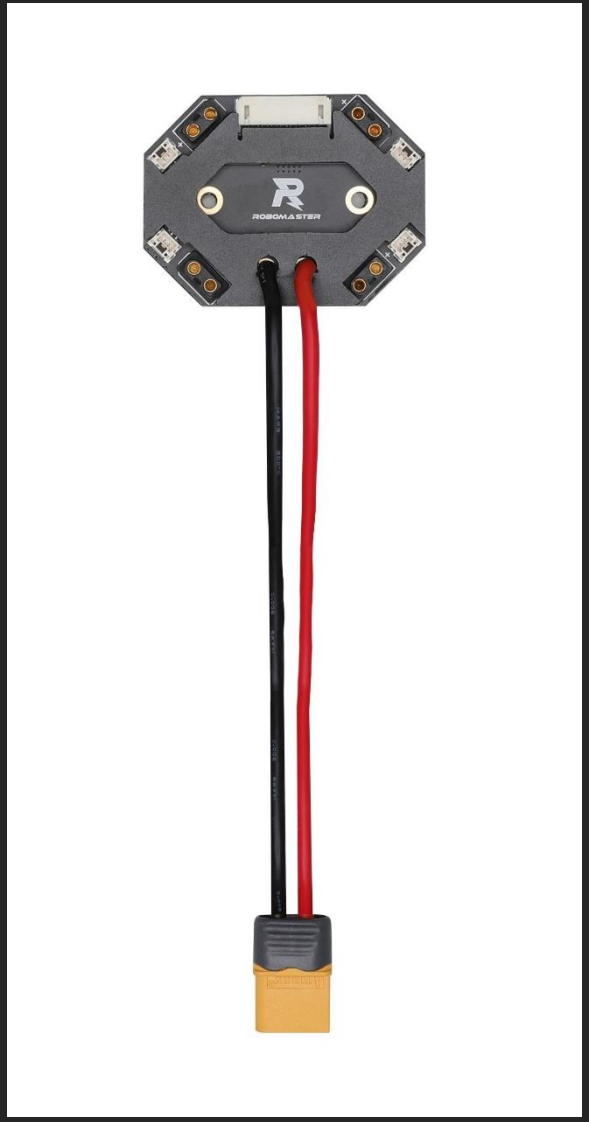
350mm

8-Pin Power Cable Adapter

8-Pin 电源信号转接线

2.3 中心板

中间有两个安装孔，可以用于固定。
另外所有的插头在固定好后，不应该受力。
中心板上表面铺铜带24V电，中心板侧面，尽量不要和铝材和其他导电体接触，防止破损后短路。

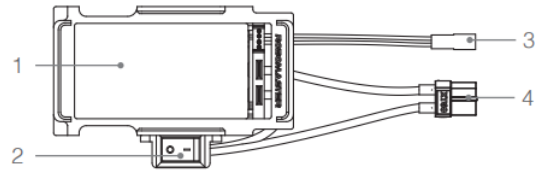


ROBOMASTER

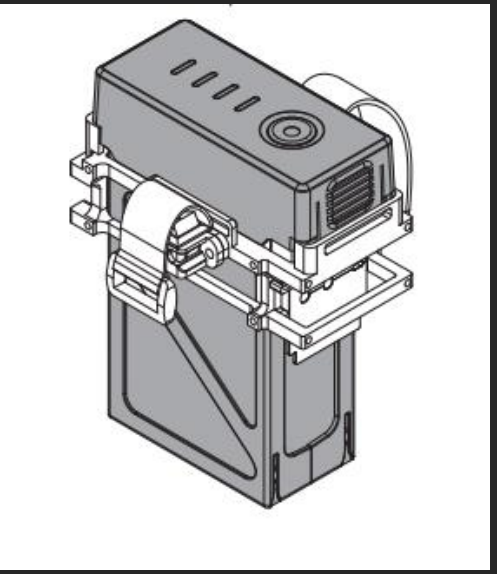
2.4 电池座和电池

只需连接4号接口至中心板即可。
电池开关方式：短按后接长按
电池架固定：受震动可能脱出，需要加固，可使用扎带或魔术贴。
调试没有把握的程序：**手放在电池架开关上！**

电池架部件名称



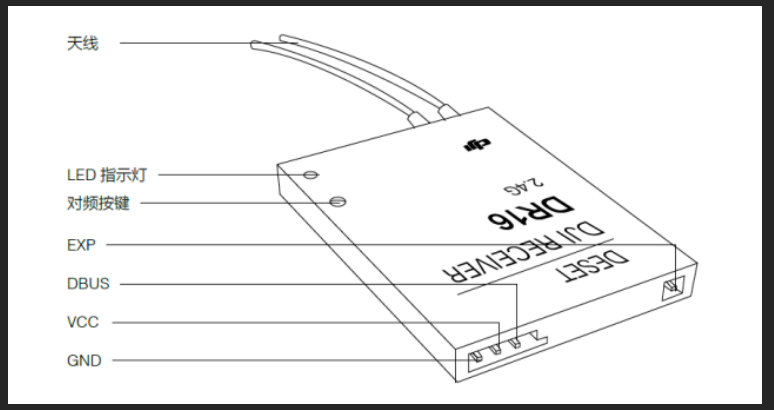
1. 电池置架
   用于安放电池。
2. 电源开关
   开启 / 断开电池供电。
3. 电池信息接口
   可用于监测当前电池电量等数据。
4. 电源输出口
   可连接至机器人或裁判系统。

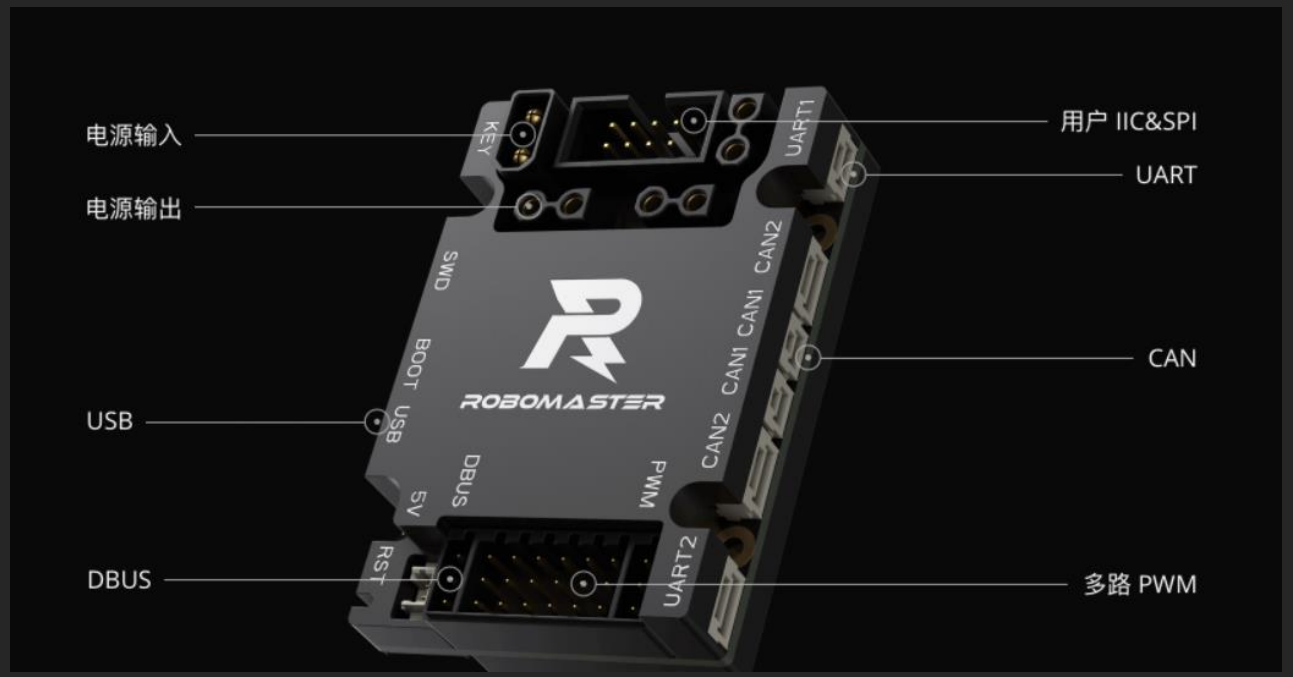


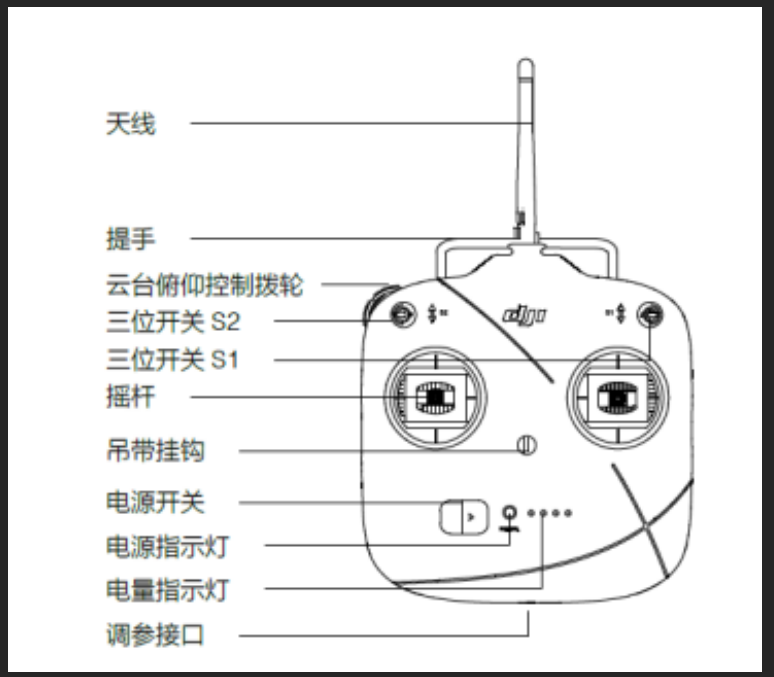
ROBOMASTER

2.5 遥控器和接收机

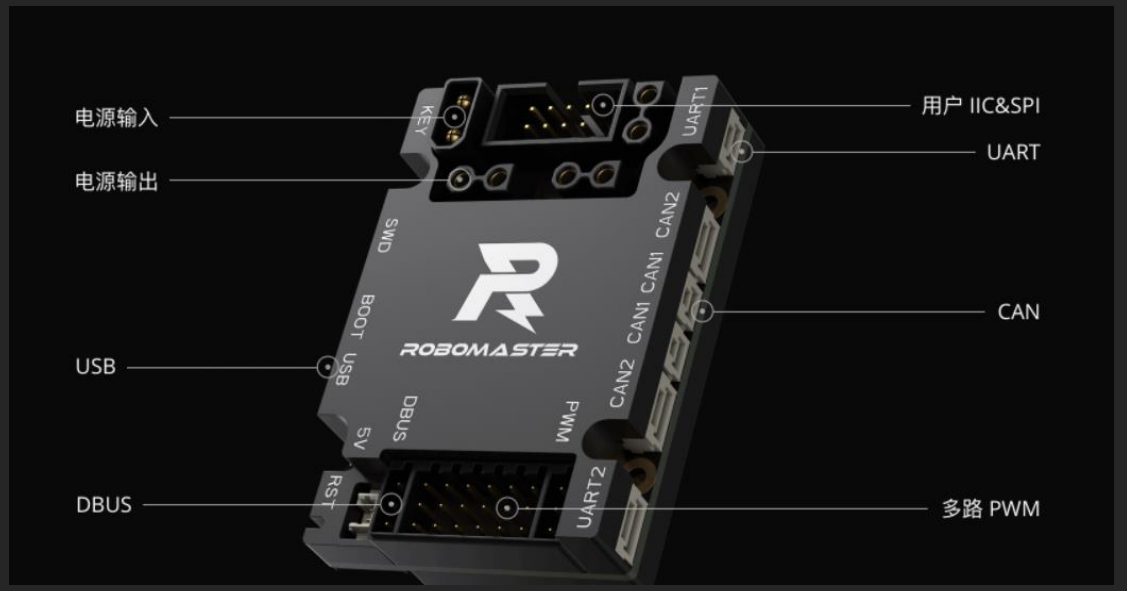
接收机使用附带的线连接至C板Dbus接口，注意不可逆接口的方向！







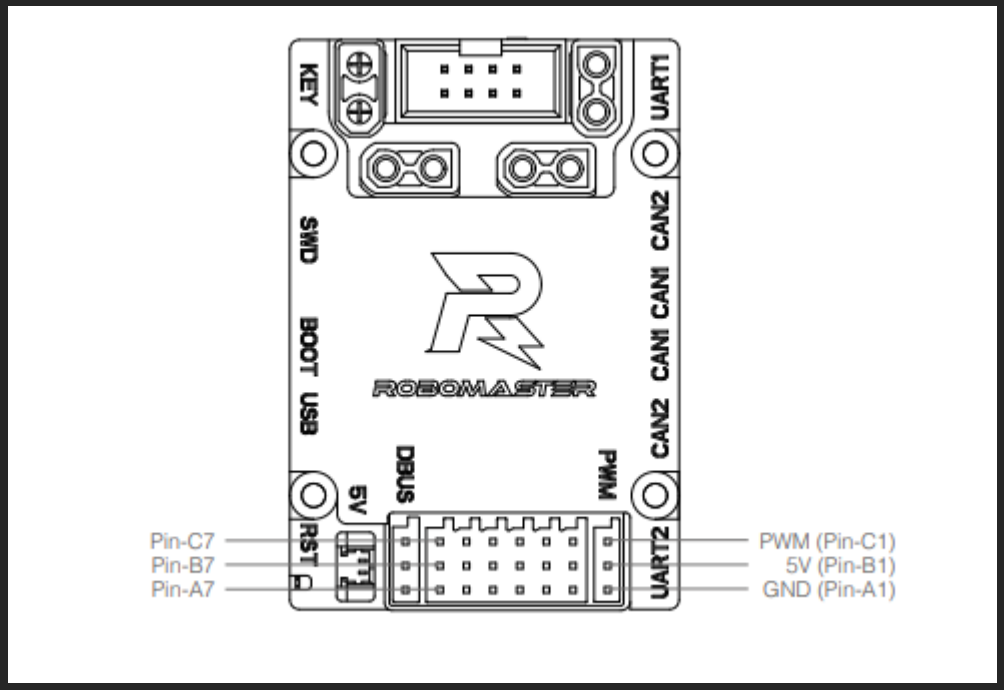
2.5 主控板



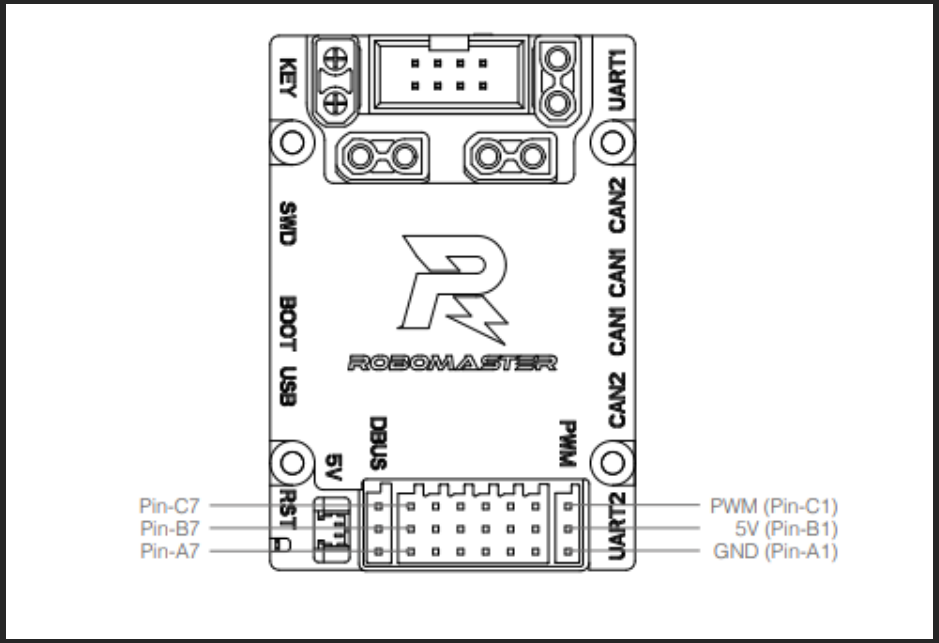
C板连接舵机时，也应注意PWM口的线序，
错误接线也可能烧毁板子！

2.5 主控板

C板连接舵机时，也应注意PWM口的线序，
错误接线也可能**烧毁**板子！



2.5 主控板



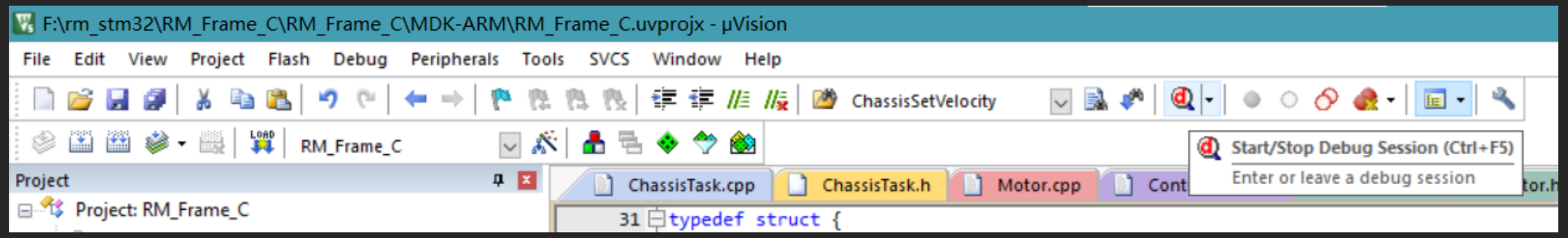
建议使用板上的固定孔固定，同时注意各信号线和电源线的连接。每年都有队伍因为场上线材松动罚站。

如何调试程序

ROBOMASTER

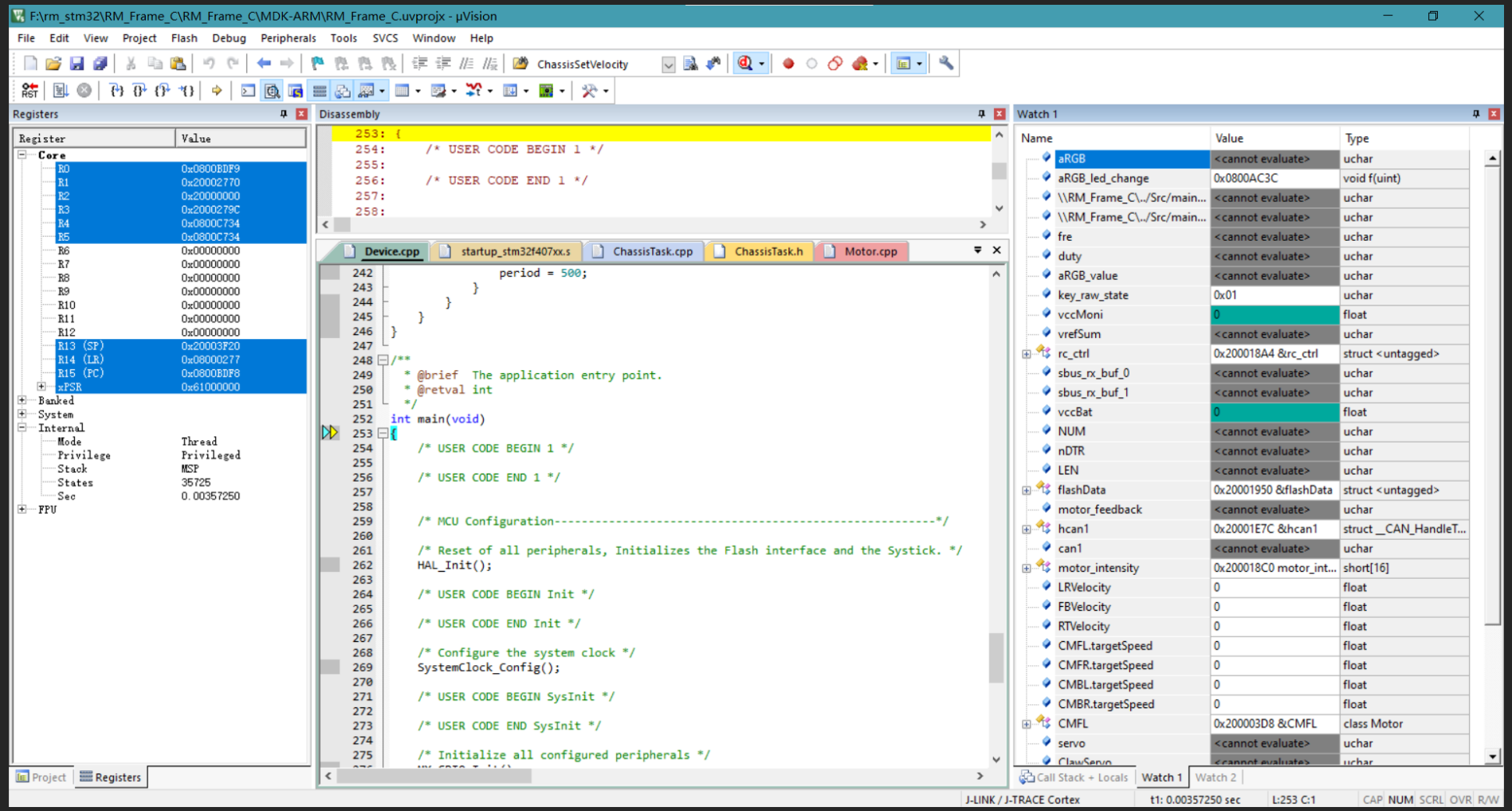
3.1 keil的Debug功能

成功编译并烧录程序后，保持烧写器的连接，即可在keil中进入debug模式。可以在其中动态追踪全局变量的数值，也可以利用断点，单步执行等功能来验证程序逻辑的同时观测局部变量的数值。



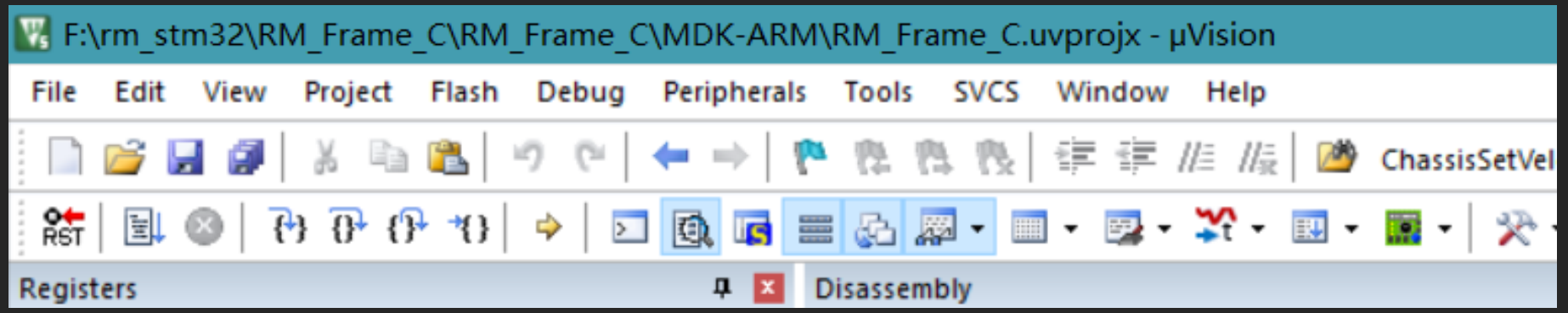
ROBOMASTER

3.1 keil的Debug功能



ROBOMASTER

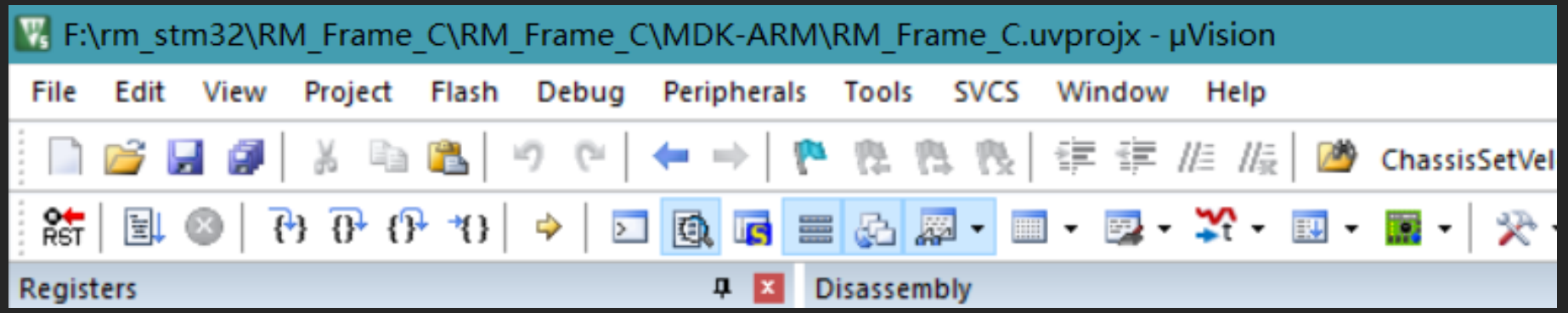
3.1 keil的Debug功能



进入Debug模式新增的工具栏：
重置程序；　　运行程序；停止程序；
单步执行（进行跳转）；单条执行（不跳转）；跳出当前函数体；运行到光标处
显示下一步执行的代码；
堆栈调用窗口 变量观察窗口

ROBOMASTER

3.1 keil的Debug功能



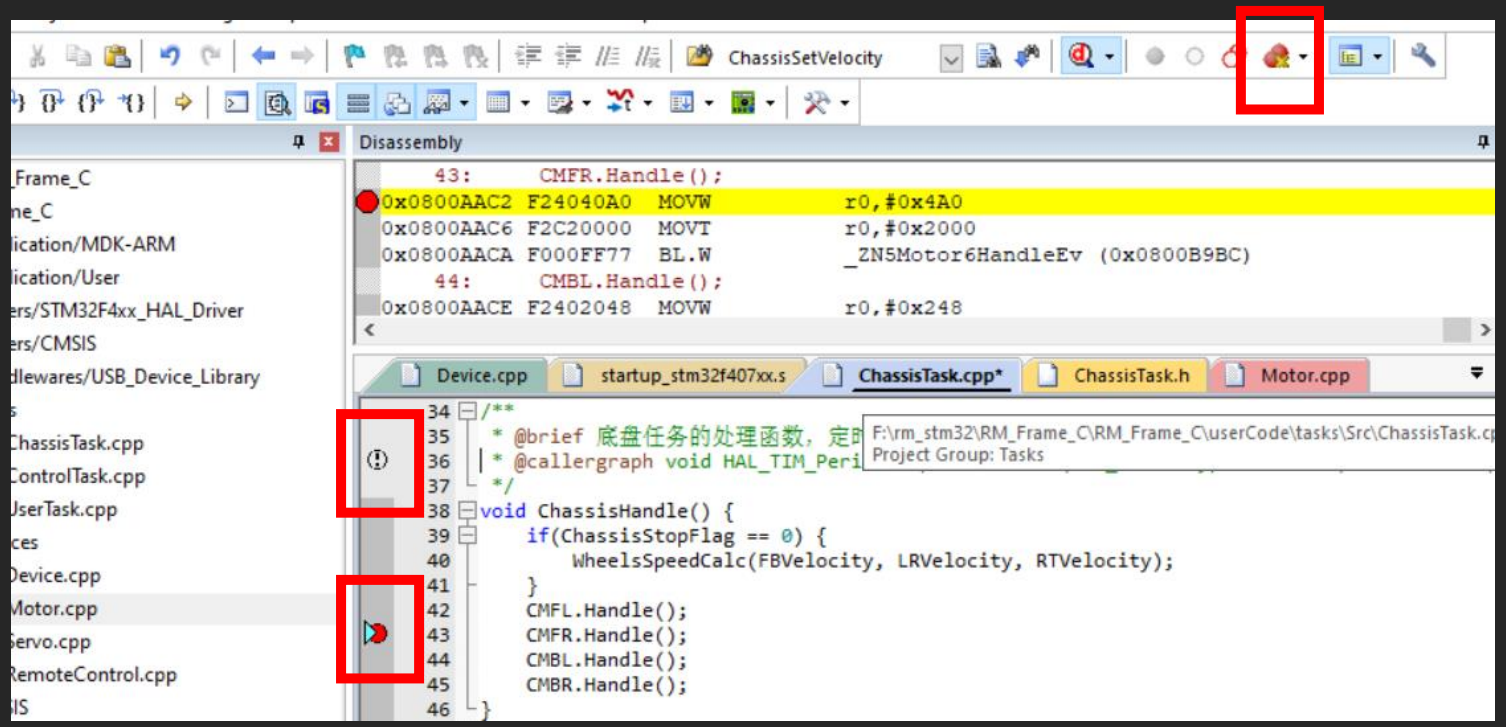
进入Debug模式新增的工具栏：
重置程序；　　运行程序；停止程序；
单步执行（进行跳转）；单条执行（不跳转）；跳出当前函数体；运行到光标处
显示下一步执行的代码；
堆栈调用窗口 变量观察窗口

ROBOMASTER

3.1 keil的Debug功能

添加断点：在代码左侧点击，出现红色点即成功，再次点击消失，右上角位置可以一键取消所有断点

若代码点击后为黑色感叹号，且所在位置为浅色，说明此位置没有对应的代码。即程序不可能执行至此处。

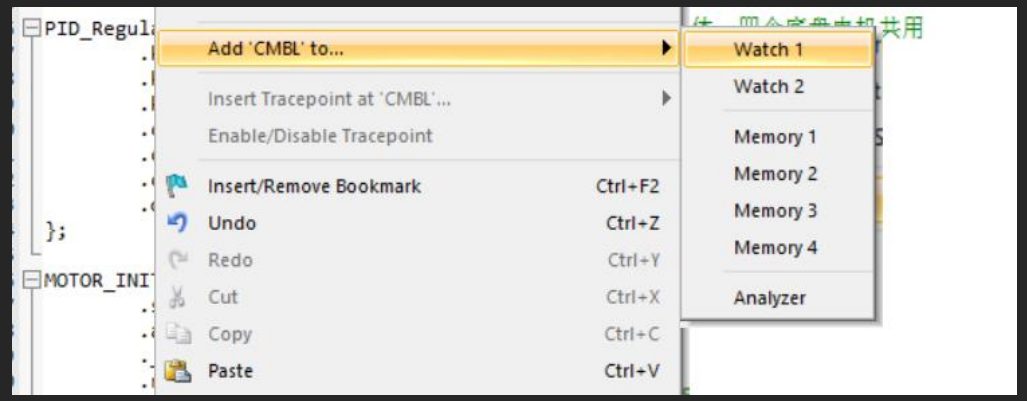


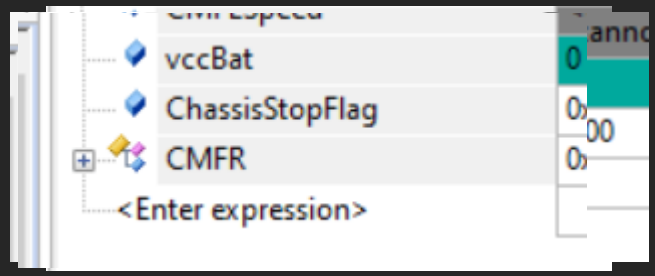
如何调试程序

ROBOMASTER

3.1 keil的Debug功能

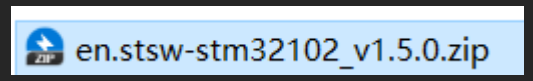
监测变量：在变量上右键，watch1和watch2是两个窗口，功能相同，任选其一。
或直接在窗口内位置输入变量名称



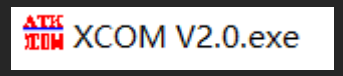


3.2 利用printf（）调试

框架内利用usb虚拟串口实现了printf（）功能，使用时需安装驱动



同时需要任意串口调试软件，以XCOM为例



ROBOMASTER

3.2 利用printf（）调试

XCOM设置如下，串口选择处的端口号可能不同，波特率对于虚拟串口来说没有意义。其他设置默认

如何调试程序

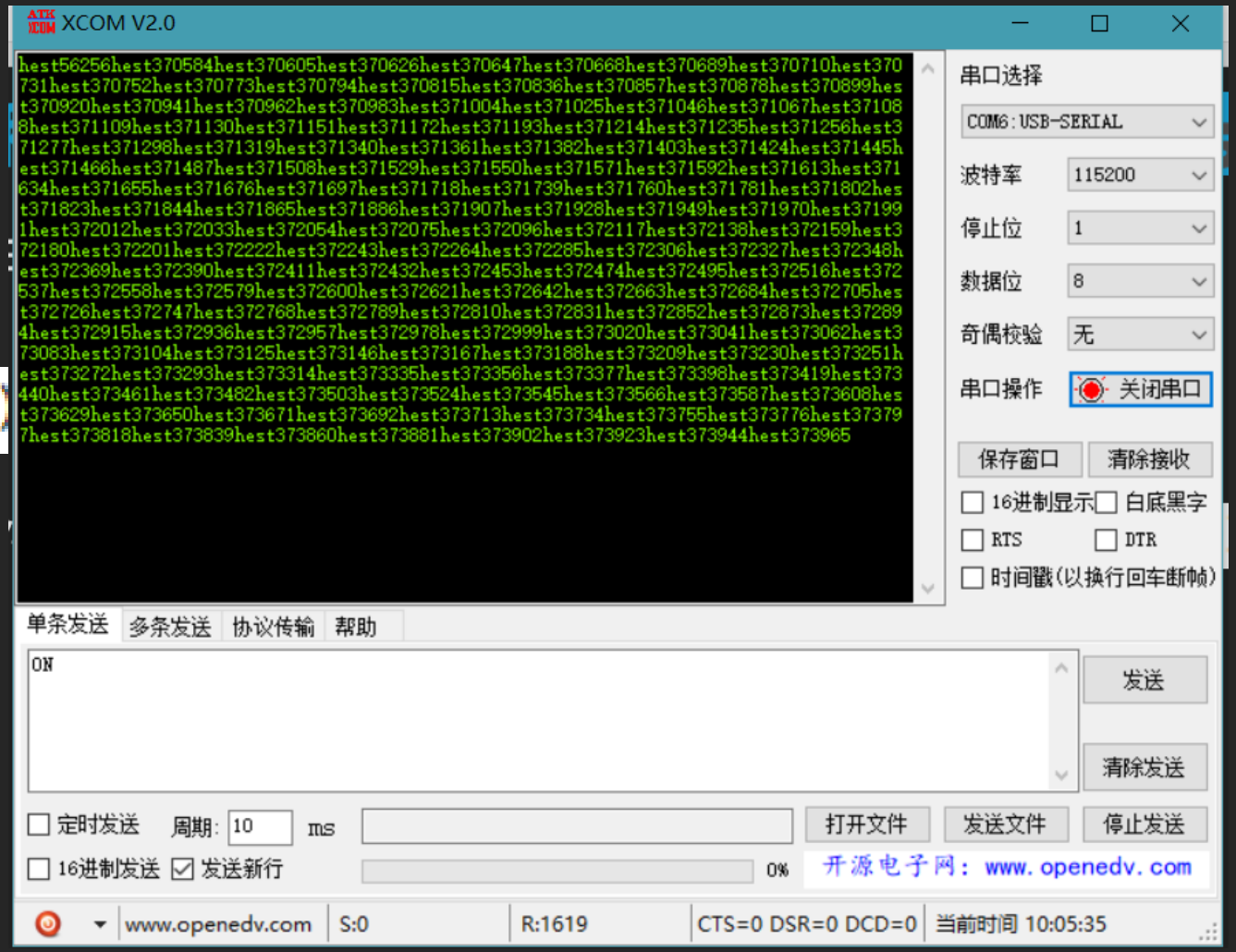
ROBOMASTER

3.2 利用printf（）调试

在程序任意位置写下

```
usart_printf("hest%d",HAL_GetTick());
```

可以收到结果，建议此功能仅在调试过程中使用，同时尽量避免调用频率过高。



安全注意事项

4.1 调试善用安全模式

遥控器右侧拨码拨到最下为急停模式，同时也可直接关闭遥控器开关触发开门狗。
调试新程序可以把手放在电池开关上。
放在桌子上调车一定要垫起来。

4.2 电气安全

接线时防止接错，短接
当闻道绝缘皮的香味时意味着短路，立刻关掉电池开始排查
电池有断路保护，如果发现接好线后打开电池就会自动关闭说明电源有明显短路。
如果需要焊线，可以使用c200公共资源。及时关闭焊台，注意保护好电线无裸露。