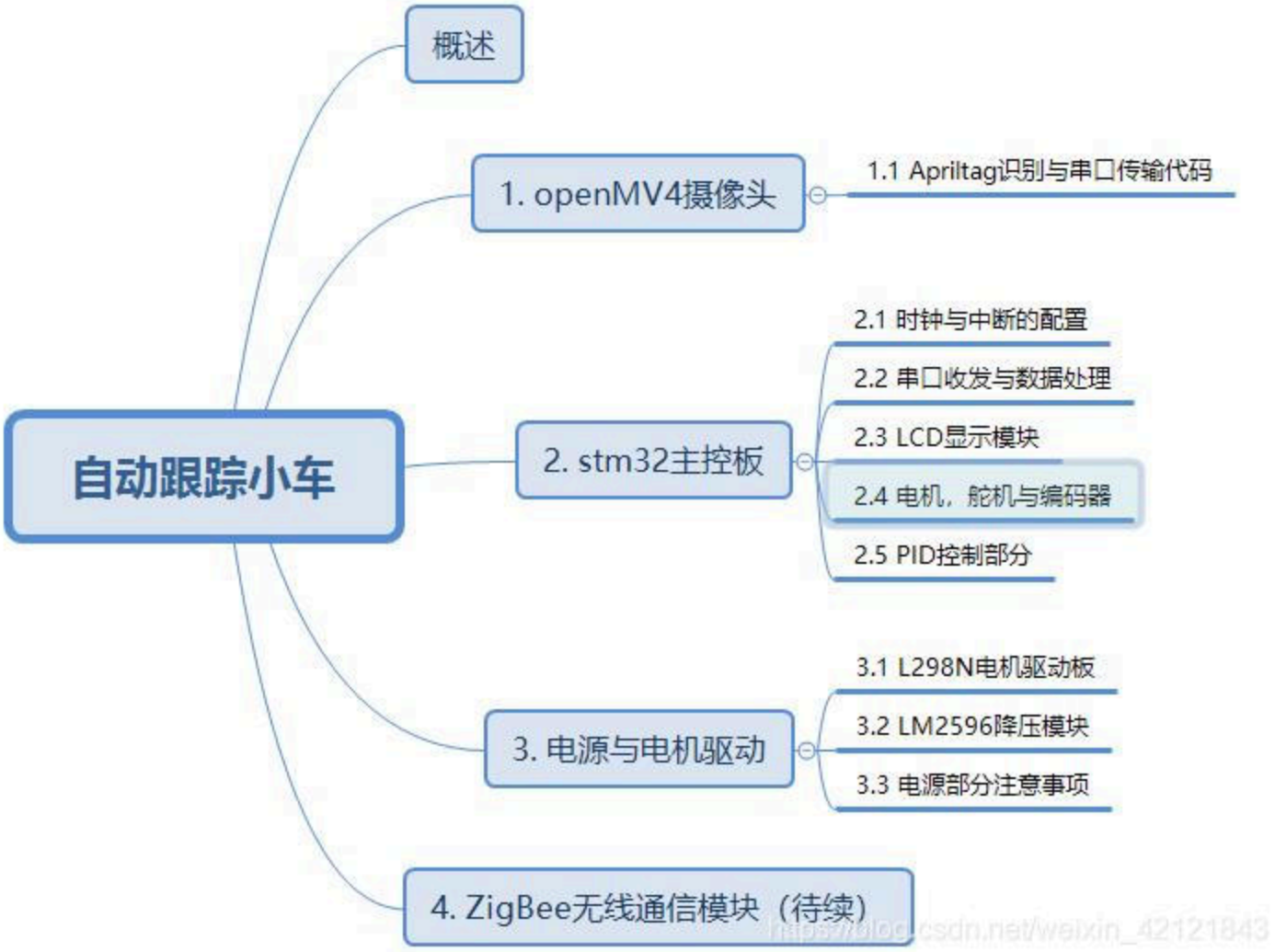
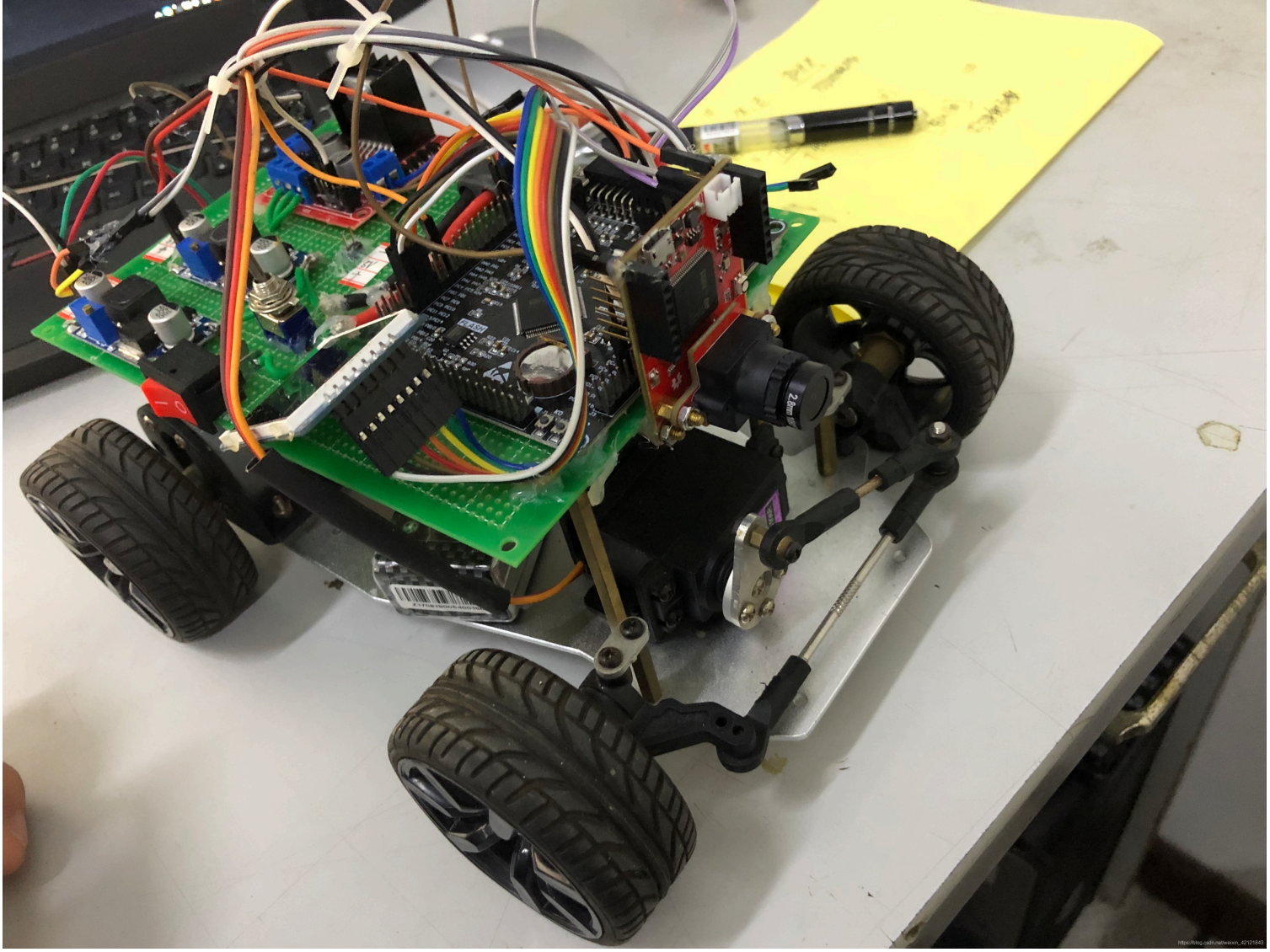
概述


自动跟踪小车
概述
1. openMV4摄像头
1.1 Apriltag识别与串口传输代码
2. stm32主控板
2.1 时钟与中断的配置
2.2 串口收发与数据处理
2.3 LCD显示模块
2.4 电机，舵机与编码器
2.5 PID控制部分
3. 电源与电机驱动
3.1 L298N电机驱动板
3.2 LM2596降压模块
3.3 电源部分注意事项
4. ZigBee无线通信模块（待续）

小车外形：



功能简介

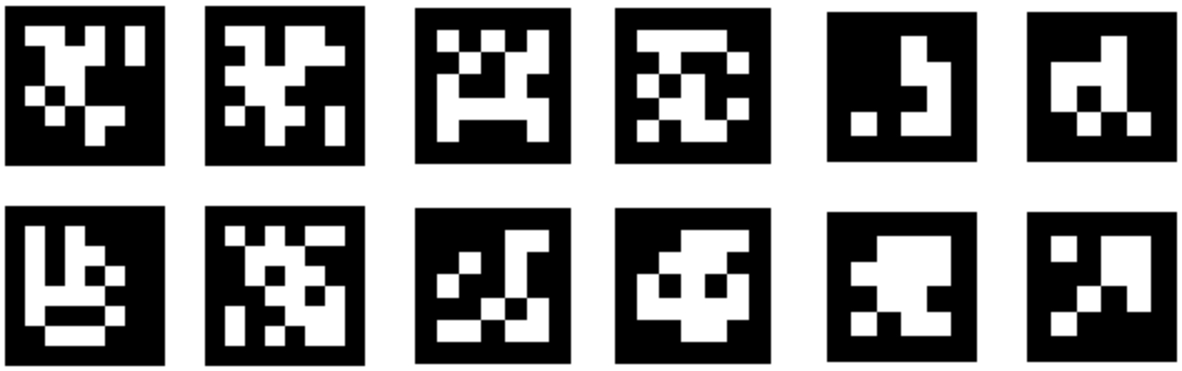
利用摄像头识别前车尾部的AprilTag，得到前车位置，传回stm32主控板处理，使两车在行驶时保持恒定距离，实现自动跟车。

1.openMV4摄像头

1.1 Apriltag识别与串口传输

AprilTag是一个视觉基准库，在AR，机器人，相机校准领域广泛使用。通过特定的标志（与二维码相似，但是降低了复杂度以满足实时性要求），可以快速地检测标志，并计算相对位置。

Apriltag示例：



通过识别Apriltag，可以得到x,y,z三个方向的距离以及偏移角度。这里只需要三维的距离即可，通过串口传回stm32.

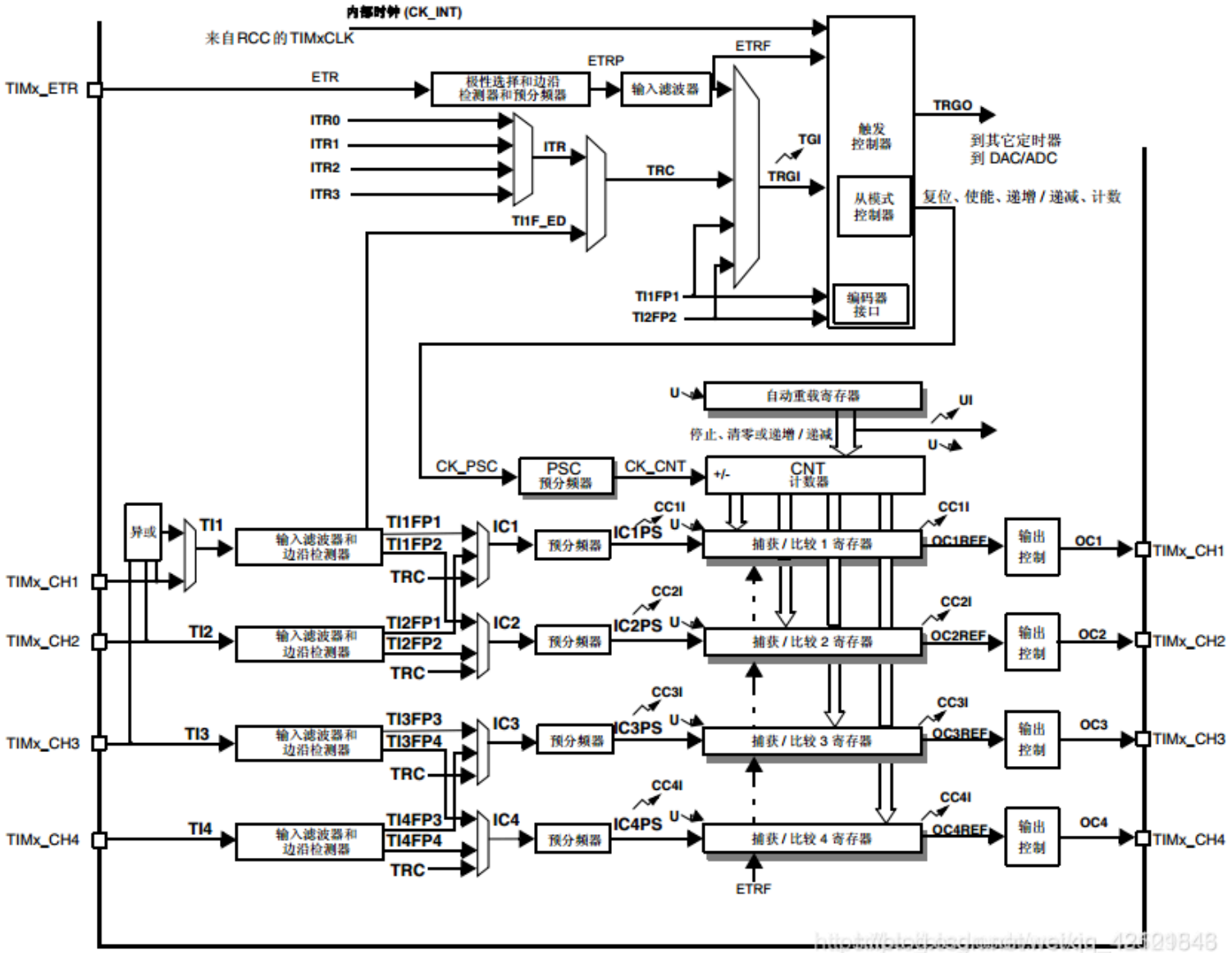
```
import sensor, image, time, math,pyb
from pyb import UART

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA) # we run out of memory if the resolution is much bigger.
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False)  # must turn this off to prevent image washout...
sensor.set_auto_whitebal(False)  # must turn this off to prevent image washout...
clock = time.clock()
uart = UART(3, 115200)#串口波特率

f_x = (2.8 / 3.984) * 160 # find_apriltags defaults to this if not set
f_y = (2.8 / 2.952) * 120 # find_apriltags defaults to this if not set
c_x = 160 * 0.5 # find_apriltags defaults to this if not set (the image.w * 0.5)
c_y = 120 * 0.5 # find_apriltags defaults to this if not set (the image.h * 0.5)

def degrees(radians):
    return (180 * radians) / math.pi

while(True):
    clock.tick()
    img = sensor.snapshot()
    for tag in img.find_apriltags(fx=f_x, fy=f_y, cx=c_x, cy=c_y): # defaults to TAG36H11
        img.draw_rectangle(tag.rect(), color = (255, 0, 0))
        img.draw_cross(tag.cx(), tag.cy(), color = (0, 255, 0))
        print_args = (tag.x_translation(), tag.y_translation(), tag.z_translation())
            #degrees(tag.x_rotation()), degrees(tag.y_rotation()), degrees(tag.z_rotation()
        # Translation units are unknown. Rotation units are in degrees.
       # print("Tx %f, Ty %f, Tz %f" % print_args)
        uart.write("A%.2f,B%.2f,C%.2f," % print_args+'\r\n')#设置特定格式，以便于stm32分割取得数
        #pyb.delay(500)
```

```
# print(clock.fps())
```

2. STM32主控板（型号为F407）

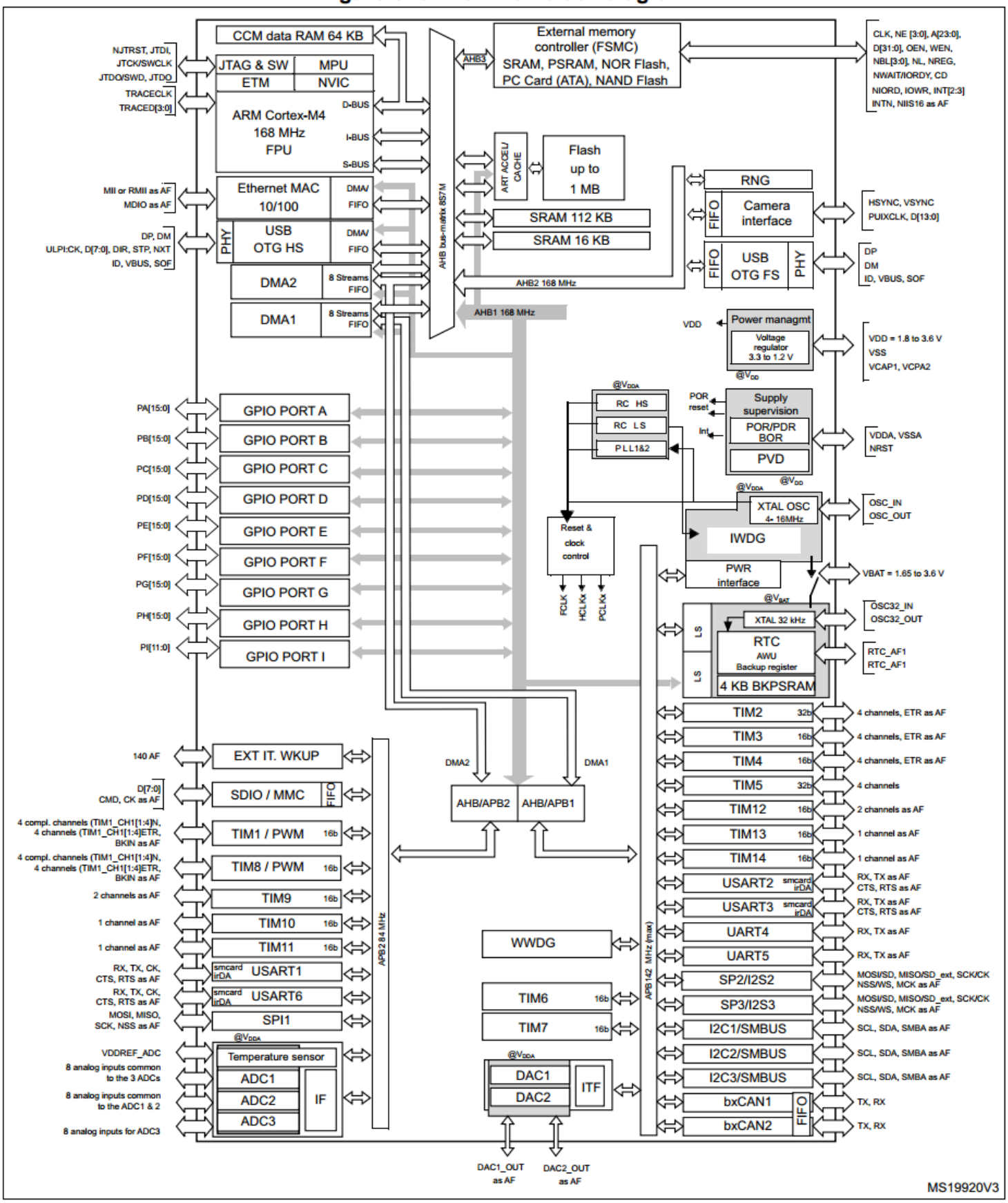
2.1 时钟与中断配置

附上stm32时钟示意图：



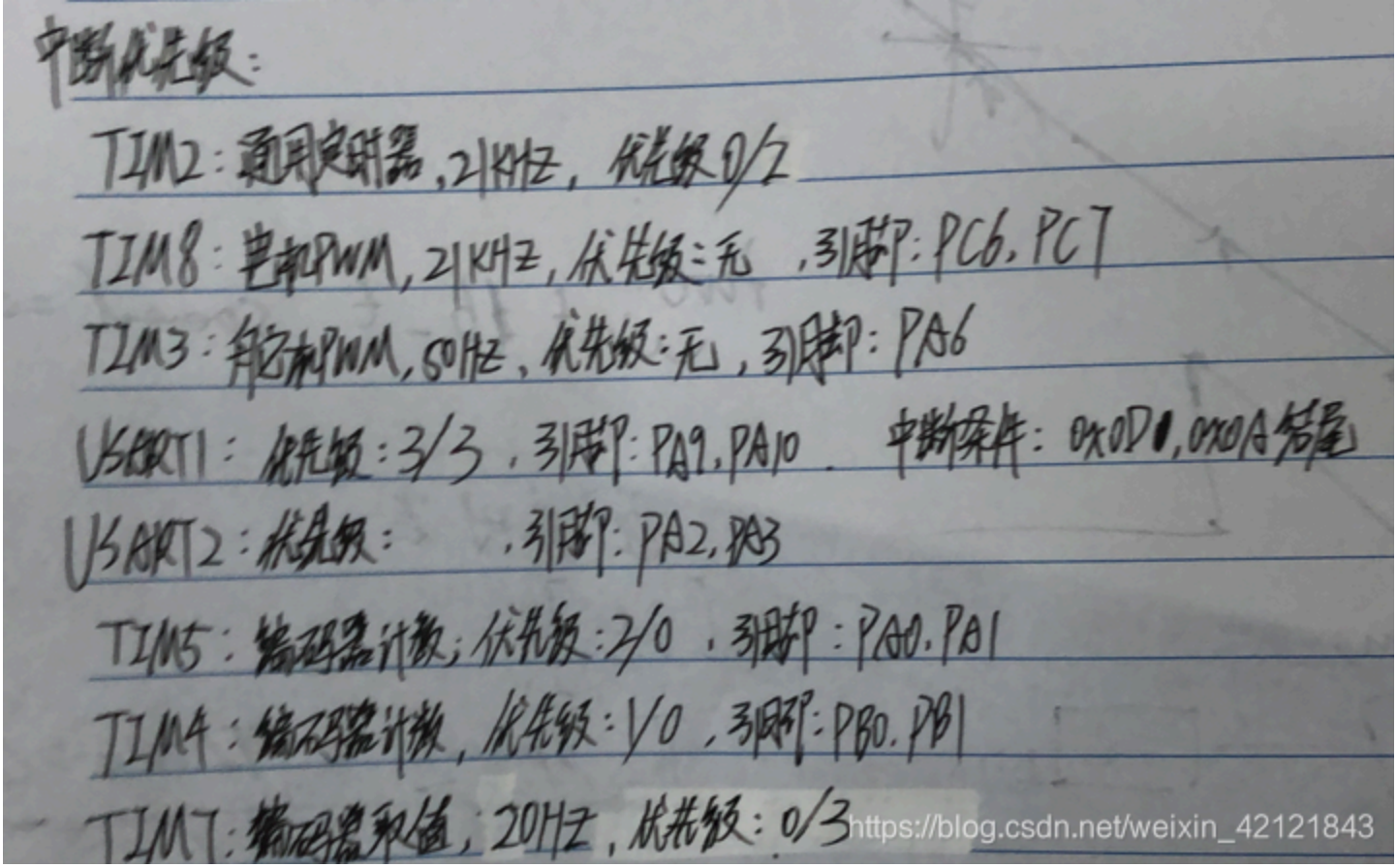
定时器示意图：

Figure 5. STM32F40x block diagram



1. The timers connected to APB2 are clocked from TIMxCLK up to 168 MHz, while the timers connected to APB1 are clocked from TIMxCLK either up to 84 MHz or 168 MHz, depending on TIMPRE bit configuration in the RCC_DCKCFGR register.
2. The camera interface and ethernet are available only on STM32F407xx devices.

定时器分配：



所有时钟初始化的函数：（每个函数的详细内容在后面）

```
TIM8_PWM_Init(400-1,20-1);       //用于控制电机，168M/20=8.4Mhz的计数频率,重装载值400，所以PWM频率为
        TIM3_PWM_Init(200-1,8400-1);//用于控制舵机，50HZ
        TIM2_Int_Init(400-1,20-1);//定时中断，21KHZ
        TIM7_Int_Init(500-1,8400-1);//用于编码器计数，20HZ，50ms中断一次
        uart_init(115200);       //初始化串口1波特率为115200
        Encoder_Init_TIM4();//编码器接口初始化
```

2.2 串口收发与数据处理

串口中断：USART1，USART2
串口初始化函数（以USART1为例）：

```
void uart_init(u32 bound){
   //GPIO端口设置
  GPIO_InitTypeDef GPIO_InitStructure;
        USART_InitTypeDef USART_InitStructure;
        NVIC_InitTypeDef NVIC_InitStructure;

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE); //使能GPIOA时钟
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//使能USART1时钟
```

```
        //串口1对应引脚复用映射
        GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_USART1); //GPIOA9复用为USART1
        GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1); //GPIOA10复用为USART1

        //USART1端口配置
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10; //GPIOA9与GPIOA10
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用功能
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       //速度50MHz
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //推挽复用输出
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP; //上拉
        GPIO_Init(GPIOA,&GPIO_InitStructure); //初始化PA9，PA10

   //USART1 初始化设置
        USART_InitStructure.USART_BaudRate = bound;//波特率设置
        USART_InitStructure.USART_WordLength = USART_WordLength_8b;//字长为8位数据格式
        USART_InitStructure.USART_StopBits = USART_StopBits_1;//一个停止位
        USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验位
        USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;//无
        USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; //收发模式
  USART_Init(USART1, &USART_InitStructure); //初始化串口1

  USART_Cmd(USART1, ENABLE);  //使能串口1

        USART_ClearFlag(USART1, USART_FLAG_TC);

#if EN_USART1_RX
        USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//开启相关中断

        //Usart1 NVIC 配置
  NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;//串口1中断通道
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=3;//抢占优先级3
        NVIC_InitStructure.NVIC_IRQChannelSubPriority =3;               //子优先级3
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                 //IRQ通道使能
        NVIC_Init(&NVIC_InitStructure); //根据指定的参数初始化VIC寄存器、

#endif

}
```

串口中断处理函数：

```
void USART1_IRQHandler(void)                    //串口1中断服务程序
{
        u8 Res;
#ifdef OS_TICKS_PER_SEC         //如果时钟节拍数定义了,说明要使用ucosII了.
        OSIntEnter();
```

```
#endif
        if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)  //接收中断(接收到的数据必须是0x0
        {
                Res =USART_ReceiveData(USART1);//(USART1->DR);  //读取接收到的数据

                if((USART_RX_STA&0x8000)==0)//接收未完成
                {
                        if(USART_RX_STA&0x4000)//接收到了0x0d
                        {
                                if(Res!=0x0a)USART_RX_STA=0;//接收错误,重新开始
                                else USART_RX_STA|=0x8000;      //接收完成了
                        }
                        else //还没收到0X0D
                        {
                                if(Res==0x0d)USART_RX_STA|=0x4000;
                                else
                                {
                                        USART_RX_BUF[USART_RX_STA&0X3FFF]=Res ;
                                        USART_RX_STA++;
                                        if(USART_RX_STA>(USART_REC_LEN-1))USART_RX_STA=0;//
                                }
                        }
                }
  }
#ifdef OS_TICKS_PER_SEC         //如果时钟节拍数定义了,说明要使用ucosII了.
        OSIntExit();
#endif
}
#endif
```

字符串接收与处理（从openMV接收到的数据）：

```
/*涉及到的全局变量
float data[3];//x,y,z方向的距离，浮点数形式
unsigned char data_string[3][7];//x,y,z方向的距离，字符串形式
*/
if(USART_RX_STA&0x8000)
                {
                        //清空字符串
                        for(i=0;i<2;i++)
                        {
                                for(j=0;j<6;j++)
                                {
                                        data_string[i][j]=' ';
                                }
```

```
                        }
                        len=USART_RX_STA&0x3fff;//得到此次接收到的数据长度
                        for(t=0,j=0;t<len;t++)
                        {
                                j=0;
                                if(USART_RX_BUF[t]=='A')
                                {
                                        t++;//去掉首字母
                                        while(USART_RX_BUF[t]!=',')
                                        {
                                                data_string[0][j]=USART_RX_BUF[t];
                                                t++;
                                                j++;
                                        }
                                }
                                if(USART_RX_BUF[t]=='B')
                                {
                                        t++;//去掉首字母
                                        while(USART_RX_BUF[t]!=',')
                                        {
                                                data_string[1][j]=USART_RX_BUF[t];
                                                t++;
                                                j++;
                                        }
                                }
                                if(USART_RX_BUF[t]=='C')
                                {
                                        t++;//去掉首字母
                                        while(USART_RX_BUF[t]!=',')
                                        {
                                                data_string[2][j]=USART_RX_BUF[t];
                                                t++;
                                                j++;
                                        }
                                }
                        }
                        //把字符串转化为浮点数
                        data[0]=myatof(data_string[0])/100.0;//x
                        data[1]=myatof(data_string[1])/100.0;//y
                        data[2]=myatof(data_string[2])*(-1)/100.0;//z，输入是负数，转换为正
                                //USART2发送数据
//                              Usart_SendString( RS232_USART, (uint8_t *)USART_RX_BUF );

                        //LCD更新显示
                        //显示xyz
//                        CLEAR(10);
```

```
//                        Gui_DrawFont_GBK16(30,0,BLACK,WHITE,clear);
//                        Gui_DrawFont_GBK16(30,0,BLACK,WHITE,data_string[0]);
//                              Gui_DrawFont_GBK16(30,20,BLACK,WHITE,clear);
//                        Gui_DrawFont_GBK16(30,20,BLACK,WHITE,data_string[1]);
//                              Gui_DrawFont_GBK16(30,40,BLACK,WHITE,clear);
//                        Gui_DrawFont_GBK16(30,40,BLACK,WHITE,data_string[2]);

                              USART_RX_STA=0;//清除标志位
                }
}
```

字符串转化为两位小数浮点数（用于后续PID控制）：

```
int myatof(const char *str)//此函数仅适用于两位小数的浮点数，返回的是乘100后的int值，因float返回有错误
{
        int flag = 1;//表示正数
        int res =0;
        u8 i=1; //小数点后两位
        while(*str != '\0')
                {
                        if( !(*str >= '0' && *str <= '9'))//找到字符串中的第一个数字
                        {
                                str++;
                                continue;
                        }
                        if(*(str-1) == '-')
                        {
                                flag=-1;//表示是一个负数
                        }

                while(*str >= '0' && *str <= '9')
                {
                        res = res *10 + (*str - '0');
                        str++;
                }
                if(*str == '.')
                {
                        str++;
                        res = res *10 + (*str - '0');
                        str++;
                        res = res *10 + (*str - '0');//保留两位，故加两次
                        return res*flag;
                }
                }
}
```

2.3 LCD显示模块

LCD模块用于调试时观察数据，调试完成可以删去，因为显示屏很耗时，使处理速度变慢
驱动函数总览：

```
void LCD_GPIO_Init(void);
void Lcd_WriteIndex(u8 Index);
void Lcd_WriteData(u8 Data);
void Lcd_WriteReg(u8 Index,u8 Data);
u16 Lcd_ReadReg(u8 LCD_Reg);
void Lcd_Reset(void);
void Lcd_Init(void);
void Lcd_Clear(u16 Color);
void Lcd_SetXY(u16 x,u16 y);
void Gui_DrawPoint(u16 x,u16 y,u16 Data);
unsigned int Lcd_ReadPoint(u16 x,u16 y);
void Lcd_SetRegion(u16 x_start,u16 y_start,u16 x_end,u16 y_end);
void LCD_WriteData_16Bit(u16 Data);
```

TFT屏幕初始化：

```
void TFT_Init_Show(void)
{
        Lcd_Clear(WHITE);
        Gui_DrawFont_GBK16(16,70,BLACK,WHITE,"by WILL CHAN");
        delay_ms(1000);
        Lcd_Clear(WHITE);
        Gui_DrawFont_GBK16(3,0,RED,WHITE,"X:");
        Gui_DrawFont_GBK16(3,20,RED,WHITE,"Y:");
        Gui_DrawFont_GBK16(3,40,RED,WHITE,"Z:");
        Gui_DrawFont_GBK16(3,60,RED,WHITE,"speed:");
}
```

字符串显示函数;

```
void Gui_DrawFont_GBK16(u16 x, u16 y, u16 fc, u16 bc, u8 *s)
{
        unsigned char i,j;
        unsigned short k,x0;
        x0=x;

        while(*s)
        {
```

```
if((*s) < 128)
{
        k=*s;
        if (k==13)
        {
                x=x0;
                y+=16;
        }
        else
        {
                if (k>32) k-=32; else k=0;

            for(i=0;i<16;i++)
                for(j=0;j<8;j++)
                        {
                        if(asc16[k*16+i]&(0x80>>j))     Gui_DrawPoint(x+j,y
                                else
                                {
                                        if (fc!=bc) Gui_DrawPoint(x+j,y+i,b
                                }
                        }
                x+=8;
        }
        s++;
}

else
{


        for (k=0;k<hz16_num;k++)
        {
          if ((hz16[k].Index[0]==*(s))&&(hz16[k].Index[1]==*(s+1)))
          {
                    for(i=0;i<16;i++)
                    {
                                for(j=0;j<8;j++)
                                        {
                                        if(hz16[k].Msk[i*2]&(0x80>>j))  Gui
                                                else {
                                                        if (fc!=bc) Gui_Dra
                                                }
                                        }
                                for(j=0;j<8;j++)
                                        {
                                        if(hz16[k].Msk[i*2+1]&(0x80>>j))
```

```
                                                    else
                                                    {
                                                            if (fc!=bc) Gui_Dra
                                                    }
                                            }
                                    }
                            }
                        }
                    s+=2;x+=16;
                }

        }
}
```

2.4 电机、舵机与编码器

定时中断：TIM2，用于修改电机和舵机的PWM占空比
初始化函数：

```
//通用定时器2中断初始化
//arr：自动重装值。
//psc：时钟预分频数
//定时器溢出时间计算方法:Tout=((arr+1)*(psc+1))/Ft us.
//Ft=定时器工作频率,单位:Mhz
//这里使用的是定时器2!
void TIM2_Int_Init(u16 arr,u16 psc)
{
        TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
        NVIC_InitTypeDef NVIC_InitStructure;

        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);  ///使能TIM2时钟

  TIM_TimeBaseInitStructure.TIM_Period = arr;   //自动重装载值
        TIM_TimeBaseInitStructure.TIM_Prescaler=psc;  //定时器分频
        TIM_TimeBaseInitStructure.TIM_CounterMode=TIM_CounterMode_Up; //向上计数模式
        TIM_TimeBaseInitStructure.TIM_ClockDivision=TIM_CKD_DIV1;

        TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStructure);//初始化TIM3

        TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE); //允许定时器2更新中断
        TIM_Cmd(TIM2,ENABLE); //使能定时器2

        NVIC_InitStructure.NVIC_IRQChannel=TIM2_IRQn; //定时器2中断
```

```
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0; //抢占优先级1
        NVIC_InitStructure.NVIC_IRQChannelSubPriority=2; //子优先级3
        NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
        NVIC_Init(&NVIC_InitStructure);

}
```

TIM2中断处理函数：

```
void TIM2_IRQHandler(void)
{
        if(TIM_GetITStatus(TIM2,TIM_IT_Update)==SET)//溢出中断
        {
                if(motor_flag==1)//反转
                {
                        TIM_SetCompare1(TIM8,motor_duty*PID_val_motor*400.0);//和定时器的自动
                        TIM_SetCompare2(TIM8,0);
                }
                if(motor_flag==0)//正转
                {
                        TIM_SetCompare1(TIM8,0);
                        TIM_SetCompare2(TIM8,motor_duty*PID_val_motor*400.0);//和定时器的自动
                }
                TIM_SetCompare1(TIM3,200-(servo_angle/45.0+1)*5);//设置舵机角度，引脚：PA6
        }
        TIM_ClearITPendingBit(TIM2,TIM_IT_Update);  //清除中断标志位
}
```

PWM输出：TIM3（舵机），TIM8（电机）
初始化函数（以TIM8为例）：

```
void TIM8_PWM_Init(u32 arr,u32 psc)
{
        //此部分需手动修改IO口设置

        GPIO_InitTypeDef GPIO_InitStructure;
        TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
        TIM_OCInitTypeDef  TIM_OCInitStructure;
        TIM_BDTRInitTypeDef     TIM_BDTRInitStructure;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8,ENABLE);     //TIM8时钟使能
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPI

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;           //GPIOFA
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;        //复用功能
```

```
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       //速度100MHz
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;      //推挽复用输出
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;        //上拉
    GPIO_Init(GPIOA,&GPIO_InitStructure);              //初始化PA7

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1;
GPIO_Init(GPIOB,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7|GPIO_Pin_8;
GPIO_Init(GPIOC,&GPIO_InitStructure);

    GPIO_PinAFConfig(GPIOC,GPIO_PinSource6,GPIO_AF_TIM8); //GPIOA8复用为定时器1
    GPIO_PinAFConfig(GPIOC,GPIO_PinSource7,GPIO_AF_TIM8); //GPIOA9复用为定时器1
    GPIO_PinAFConfig(GPIOC,GPIO_PinSource8,GPIO_AF_TIM8); //GPIOA10复用为定时器1
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource7,GPIO_AF_TIM8); //GPIOB13复用为定时器1
    GPIO_PinAFConfig(GPIOB,GPIO_PinSource0,GPIO_AF_TIM8); //GPIOB14复用为定时器1
    GPIO_PinAFConfig(GPIOB,GPIO_PinSource1,GPIO_AF_TIM8); //GPIOB15复用为定时器1

    TIM_TimeBaseStructure.TIM_Prescaler=psc;  //定时器分频
    TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up; //向上计数模式
    TIM_TimeBaseStructure.TIM_Period=arr;   //自动重装载值
    TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;

    TIM_TimeBaseInit(TIM8,&TIM_TimeBaseStructure);//初始化定时器1

    //初始化PWM模式
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 0;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
    TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Reset;
    TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;

    TIM_OC1Init(TIM8, &TIM_OCInitStructure);
    TIM_OC2Init(TIM8, &TIM_OCInitStructure);
    TIM_OC3Init(TIM8, &TIM_OCInitStructure);

    TIM_OC1PreloadConfig(TIM8,TIM_OCPreload_Enable);
    TIM_OC2PreloadConfig(TIM8,TIM_OCPreload_Enable);
    TIM_OC3PreloadConfig(TIM8,TIM_OCPreload_Enable);

    TIM_BDTRInitStructure.TIM_OSSRState = TIM_OSSRState_Enable;
    TIM_BDTRInitStructure.TIM_OSSIState = TIM_OSSIState_Enable;
    TIM_BDTRInitStructure.TIM_LOCKLevel = TIM_LOCKLevel_OFF;
```

```
        TIM_BDTRInitStructure.TIM_DeadTime = 0;
        TIM_BDTRInitStructure.TIM_Break = TIM_Break_Disable;
        TIM_BDTRInitStructure.TIM_BreakPolarity = TIM_BreakPolarity_Low;
        TIM_BDTRInitStructure.TIM_AutomaticOutput = TIM_AutomaticOutput_Disable;
        TIM_BDTRConfig(TIM8,&TIM_BDTRInitStructure);

        TIM_Cmd(TIM8,ENABLE);
        TIM_CCPreloadControl(TIM8,ENABLE);
        TIM_CtrlPWMOutputs(TIM8,ENABLE);

}
```

编码器初始化函数：

```
void Encoder_Init_TIM4(void)
{
    GPIO_InitTypeDef         GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_ICInitTypeDef        TIM_ICInitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4,ENABLE);//开启TIM4时钟
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB,ENABLE);//开启GPIOB时钟

    GPIO_PinAFConfig(GPIOB,GPIO_PinSource6,GPIO_AF_TIM4);//PB6引脚复用
    GPIO_PinAFConfig(GPIOB,GPIO_PinSource7,GPIO_AF_TIM4);//PB7引脚服用

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7; //GPIOB6,GPIOB7
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
    //GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL ;
          GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
    GPIO_Init(GPIOB,&GPIO_InitStructure);

    NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; /*它的抢占优先级可以尽量设置低点*/
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = DISABLE;//禁用中断，防止计数溢出而没有相应函数，造成卡
    NVIC_Init(&NVIC_InitStructure);



    TIM_TimeBaseStructure.TIM_Period = 4095; //设置下一个更新事件装入活动的自动重装载寄存器周期的值
    TIM_TimeBaseStructure.TIM_Prescaler = 0; //设置用来作为TIMx时钟频率除数的预分频值   不分频
```

```
    TIM_TimeBaseStructure.TIM_ClockDivision = 0; //设置时钟分割:TDTS = Tck_tim
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM向上计数模式
    TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);

    TIM_EncoderInterfaceConfig(TIM4, TIM_EncoderMode_TI12,TIM_ICPolarity_Rising,TIM_ICPolar
    TIM_ICStructInit(&TIM_ICInitStructure);
    TIM_ICInitStructure.TIM_ICFilter = 10;  //输入滤波器
    TIM_ICInit(TIM4, &TIM_ICInitStructure);
    TIM_ClearFlag(TIM4, TIM_FLAG_Update);  //清除所有标志位
    TIM_ITConfig(TIM4, TIM_IT_Update, DISABLE); //允许中断更新
    TIM4->CNT = 0;
    TIM_Cmd(TIM4, ENABLE);  //使能TIM4
}
```

编码器返回速度值：

```
/**************************************************************************
函数功能：单位时间读取编码器计数
入口参数：定时器
返回  值：速度值
**************************************************************************/
float Read_Encoder_Speed(uint8_t TIMX)
{
    int32_t Encoder_TIM;
    float res = 0;
    switch (TIMX)
    {
    case 5:
        Encoder_TIM = TIM_GetCounter(TIM5);
        TIM5->CNT = ENCODER_BASE_COUNT;
        res = (int32_t)Encoder_TIM - ENCODER_BASE_COUNT;
        break;
    case 4:
        Encoder_TIM = TIM_GetCounter(TIM4);
        TIM4->CNT = ENCODER_BASE_COUNT;
        res = (int32_t)Encoder_TIM - ENCODER_BASE_COUNT;
        break;
    default:
        Encoder_TIM = 0;
        res = 0;
    }
    if(res>2048.0f)
        res-=4096.0f;
```

```
    return res*360.0f/4096.0f;
}
```

定时从编码器取数，注意，时间不一样，取回的数值也不一样，取决于实际速度以及编码器线数。这里50ms取一次：

```
void TIM7_IRQHandler(void)//频率20HZ，用于编码器计数
{
        if(TIM_GetITStatus(TIM7,TIM_IT_Update)==SET)//溢出中断
        {
                speed=Read_Encoder_Speed(4);
        }
        TIM_ClearITPendingBit(TIM7,TIM_IT_Update);  //清除中断标志位
}
```

2.5 PID控制

PID库函数：

```
#define N 2                                     //需要对多少变量进行pid调节

const float KP[N]={1.3,1.0};//这里只用了比例调节
const float KI[N]={0,0};
const float KD[N]={0,0};


struct _pid{
        float SetVol;                                   //定义设定值
        float ActVol;                                   //定义实际值
        float Err;                                              //定义误差
        float   Err_Next;                               //定义上一个误差
        float   Err_Last;                               //定义上上一个误差
        float Kp,Ki,Kd;                         //定义比例、积分、微分系数
        float integral;                         //定义积分值
        float actuator;                         //定义控制器执行变量
}pid[N];

void PID_Init(void)
{
        for(int i=0;i<N;i++)
        {
                pid[i].SetVol=0.0;
                pid[i].ActVol=0.0;
```

```
                pid[i].Err=0.0;
                pid[i].Err_Next=0.0;
                pid[i].Err_Last=0.0;
                pid[i].integral=0.0;
                pid[i].actuator=0.0;
                pid[i].Kp=KP[i];
                pid[i].Ki=KI[i];
                pid[i].Kd=KD[i];
        }
}

float PID_realize(float set_val,float get_val,int i)                    //位置型PID算法实现
{
        pid[i].SetVol=set_val;
        pid[i].ActVol=get_val;
        pid[i].Err=pid[i].SetVol-pid[i].ActVol;
        float IncVol;           //定义增量
        pid[i].integral+=pid[i].Err;
//      IncVol=pid[i].Kp*(pid[i].Err-pid[i].Err_Next)+pid[i].Ki*pid[i].Err+pid[i].Kd*(pid[i
        pid[i].actuator=pid[i].Kp* pid[i].Err+pid[i].Ki*pid[i].integral+pid[i].Kd*(pid[i].E
//      pid[i].actuator=adc_val+IncVol;
        pid[i].ActVol=pid[i].actuator;
        pid[i].Err_Last=pid[i].Err_Next;
        pid[i].Err_Next=pid[i].Err;

        return pid[i].actuator;
}
```

主函数中的PID调节：

```
                z_get=data[2];
                x_get=data[0];
                if(z_get-z_set>0.5||z_get-z_set<-0.5)//电机PID
                {
                        LED1=0; //调节时灯亮
                        PID_val_motor=PID_realize(z_set,z_get,0);
                        PID_val_motor=PID_val_motor/10.0;
                        if(PID_val_motor<=0)
                                motor_flag=0;//motor_flag控制电机正反转，PID_val_motor用于改变占
                        if(PID_val_motor>0)
                                motor_flag=1;
                        PID_val_motor=abs_float(PID_val_motor);
                        if(PID_val_motor>2)PID_val_motor=0;//标志太远，让车停止
                        if(PID_val_motor>1&&PID_val_motor<=2)PID_val_motor=1;
                        if(PID_val_motor<0.2)PID_val_motor=0;
                }
```

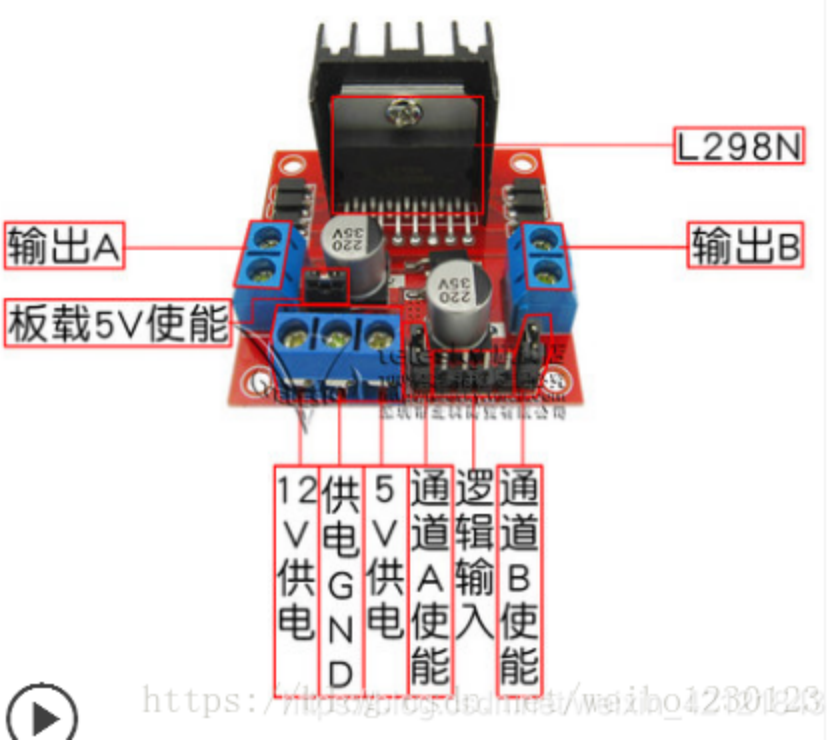
```
                if(x_get-x_set>0.1||x_get-x_set<-0.1)//舵机PID
                {
                        LED1=0;
                        PID_val_servo=PID_realize(x_set,x_get,1);
                        servo_angle=((140-35)/6)*PID_val_servo+35;//线性映射，把PID的值转化为角
                        if(servo_angle<35)servo_angle=35;
                        if(servo_angle>140)servo_angle=140;
                }
                LED1=1;
```

定时器TIM2中断里改变占空比：

```
void TIM2_IRQHandler(void)
{
        if(TIM_GetITStatus(TIM2,TIM_IT_Update)==SET)//溢出中断
        {
                if(motor_flag==1)//反转
                {
                        TIM_SetCompare1(TIM8,motor_duty*PID_val_motor*400.0);//和定时器的自动
                        TIM_SetCompare2(TIM8,0);
                }
                if(motor_flag==0)//正转
                {
                        TIM_SetCompare1(TIM8,0);
                        TIM_SetCompare2(TIM8,motor_duty*PID_val_motor*400.0);//和定时器的自动
                }
                TIM_SetCompare1(TIM3,200-(servo_angle/45.0+1)*5);//设置舵机角度，根据舵机手册得
        }
        TIM_ClearITPendingBit(TIM2,TIM_IT_Update);  //清除中断标志位
}
```

3.电源与电机驱动

3.1 L298N电机驱动板



因为后面两路电机要求同速，故把AB两通道用线短接，用一路PWM控制两路电机

下面是使用说明：

1、直流电机的驱动：

该驱动板可驱动 2 路直流电机，使能端 ENA、ENB 为高电平时有效，控制方式及直流电机状态表如下所示：

ENA	IN1	IN2	直流电机状态
0	X	X	停止
1	0	0	制动
1	0	1	正转
1	1	0	反转
1	1	1	制动

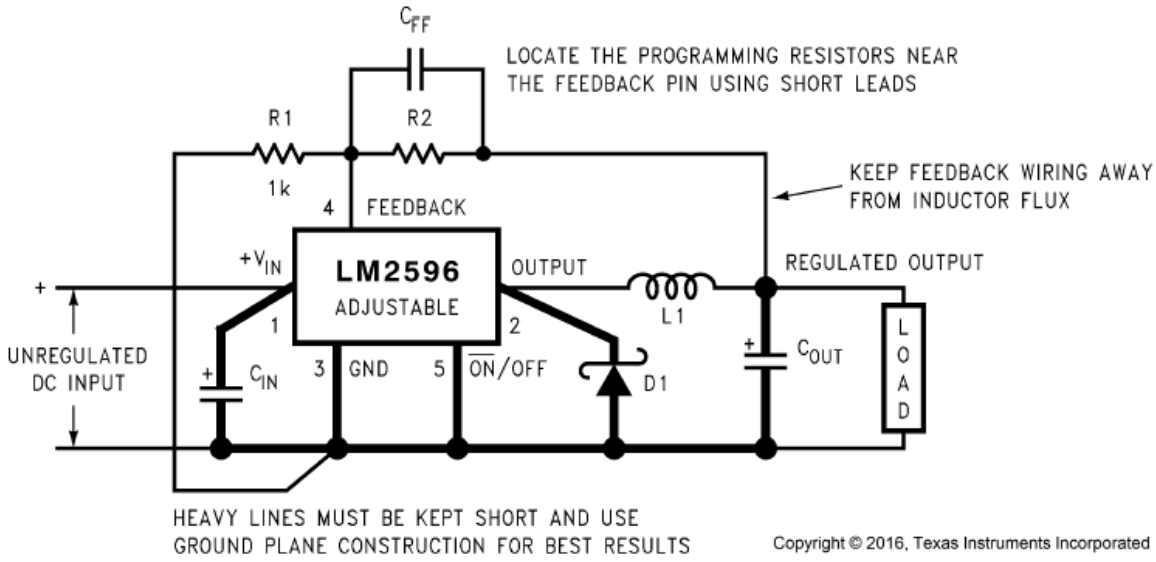
若要对直流电机进行 PWM 调速，需设置 IN1 和 IN2，确定电机的转动方向，然后对使能端输出 PWM 脉冲，即可实现调速。注意当使能信号为 0 时，电机处于自由停止状态；当使能信号为 1，且 IN1 和 IN2 为 00 或 11 时，电机处于制动状态，阻止电机转动。

具体控制代码见上面TIM2中断处理函数中，利用两路定时器轮流输出PWM（另一路为零），即可控制电机正反转

3.2 LM2596降压模块

手册中的典型连接：

LM2596 Adjustable Output Series Buck Regulator



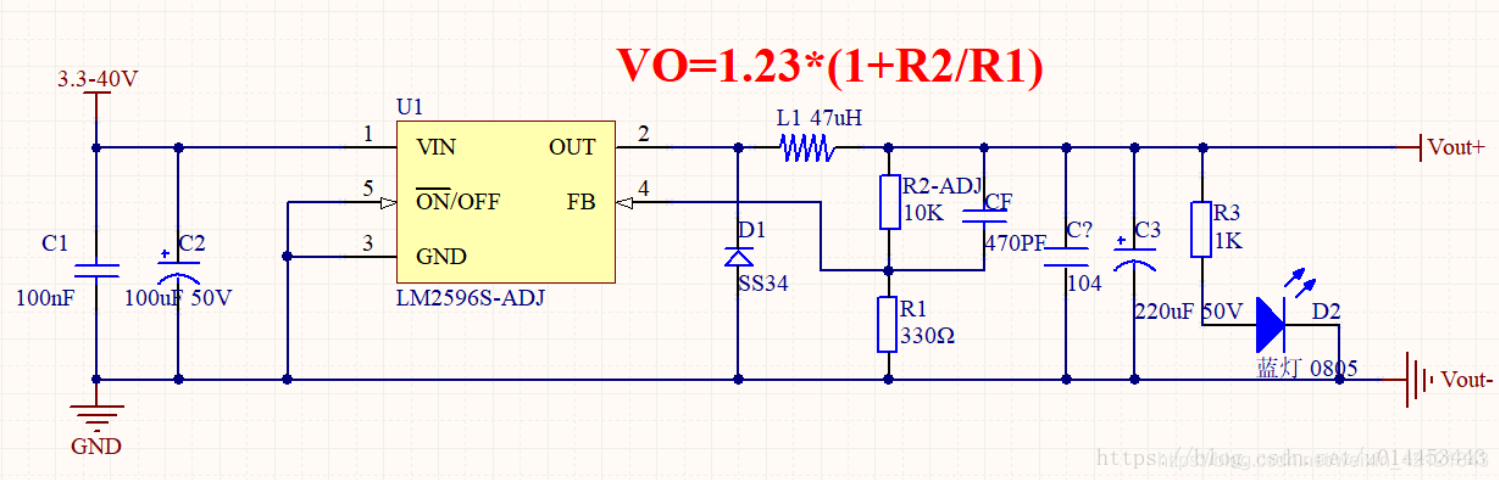
$$V_{OUT} = V_{REF}\left(1 + \frac{R_2}{R_1}\right)$$

where V_{REF} = 1.23 V

$$R_2 = R_1\left(\frac{V_{OUT}}{V_{REF}} - 1\right)$$

Select R_1 to be approximately 1 kΩ, use a 1% resistor for best stability.
C_{IN} — 470-μF, 50-V, Aluminum Electrolytic Nichicon *PL Series*
C_{OUT} — 220-μF, 35-V Aluminum Electrolytic, Nichicon *PL Series*
D1 — 5-A, 40-V Schottky Rectifier, 1N5825
L1 — 68 μH, L38
R1 — 1 kΩ, 1%
C_{FF} — See *Feedforward Capacitor (C_{FF})*

原理图如下：



3.3 电源部分注意事项

1.电池用的是12v航模锂电池，为了防止过放导致电池损坏，必须要在电池输入端加一个电压表模块，如下图：

2.控制部分电源和电机舵机电源分开，因为电机舵机启动时会过大电流，导致电压不稳定，影响芯片供电。

这里LM2596给电机供电，一个LM2596给舵机供电，另一个LM2596给单片机和openMV供电

3.控制电源和电机舵机电源分别加开关，下程序的时候先关闭电机和舵机的电源。因为此时控制器没有给信号，电机和舵机可能会不受控制的运动