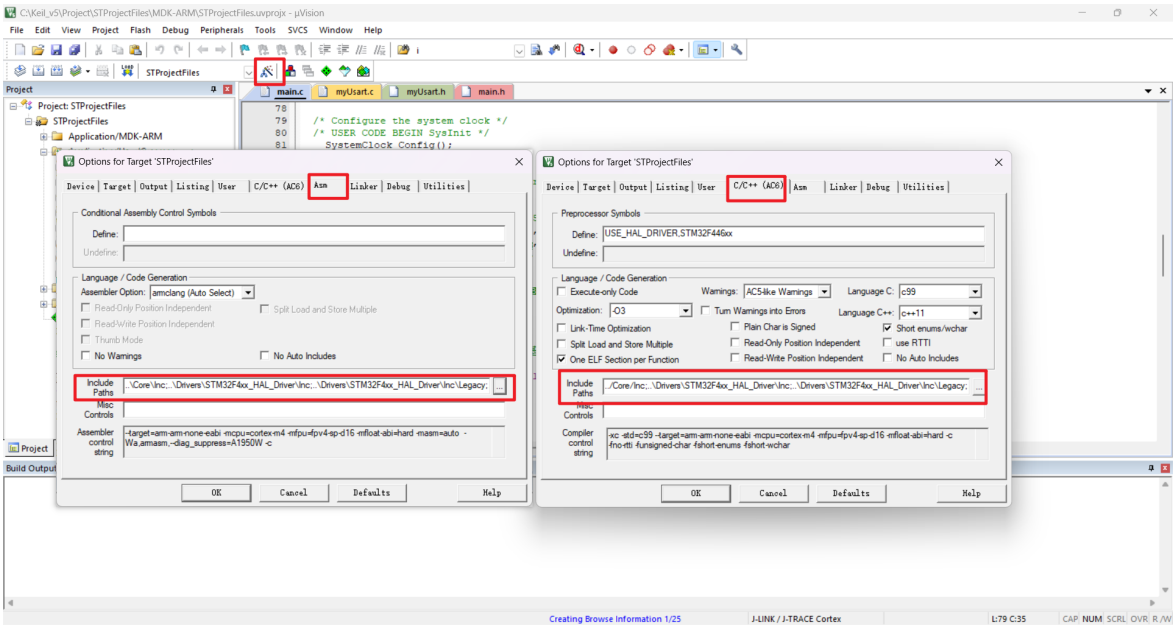
TotalControl

机器狗整体控制

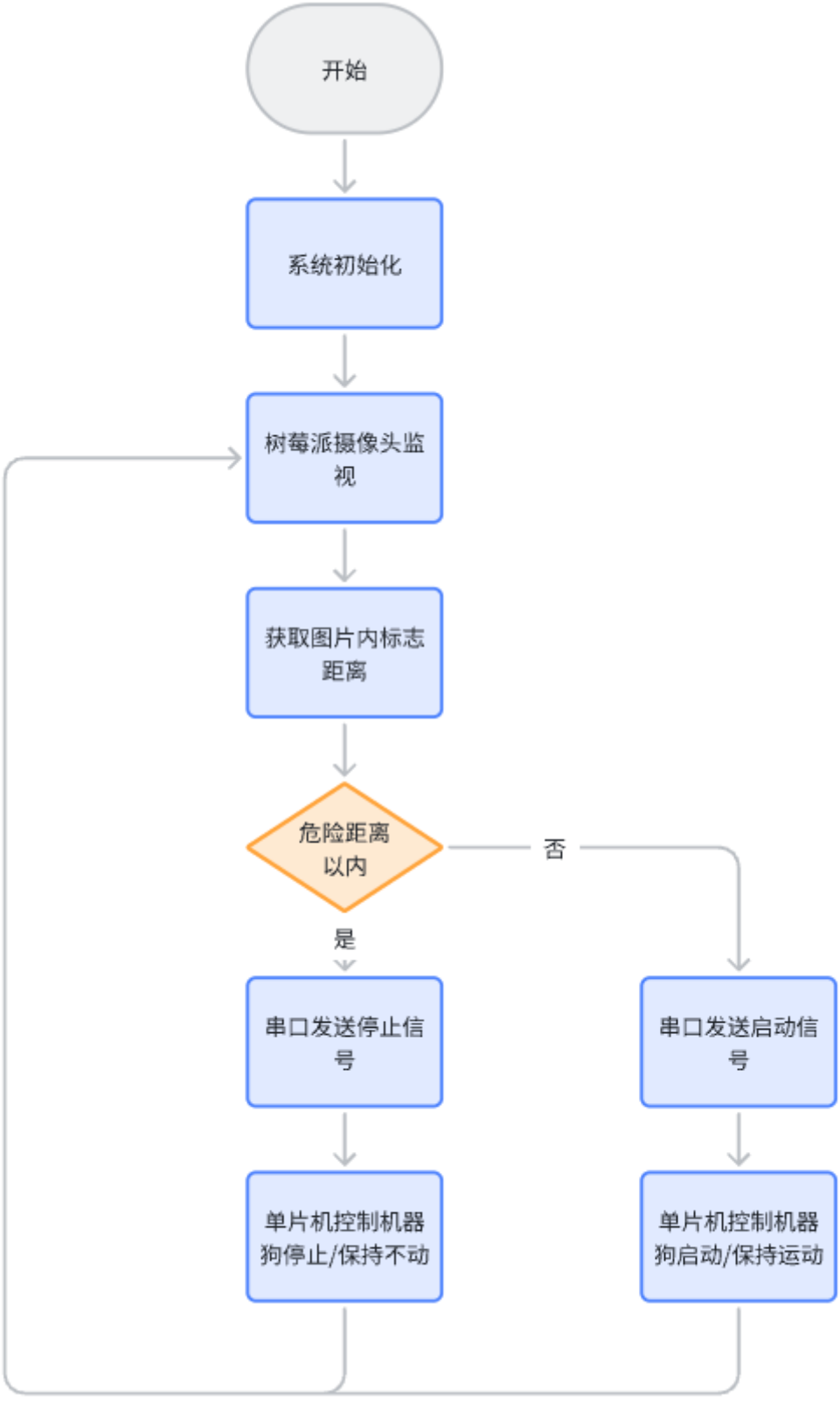
0、写在前面：

1. 未特殊说明，一切树莓派操作皆在 `小黑框（终端）` 中进行STM32单片机的操作皆在创建的 Keil 项目中进行
2. 项目创建方法：Keil创建STM32项目并烧录使用
3. 关于初始化函数的位置
   - （不推荐）直接写在 `main.c` 里
   - （推荐）创建单独的 `xxx.c` 和 `xxx.h` 用于放置相关函数，分别放在 Core 文件夹下的 Inc 和 Src里，前一个文件夹用于放 `xxx.h`，后者放 `xxx.c`
   - （较推荐）创建单独的 `xxx.c` 和 `xxx.h` 用于放置相关函数，单独额外创建文件夹用于放置自写的文件（记得添加到项目的 `include` 中）



1、控制关系

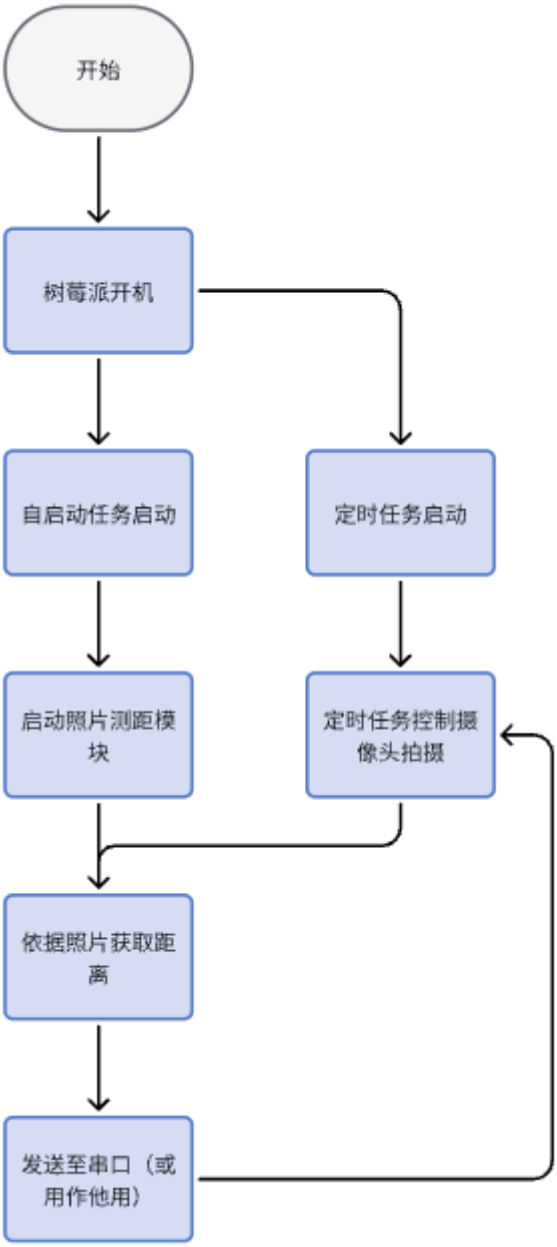
> 通过树莓派接收图片信息并测量标志距离，根据所得距离信息与预设 `危险距离` 对比，启动串口，根据是否在 `危险距离以内` 发送 `停止` 或 `启动` 信息到单片机，单片机根据接收的启停信号来重新开始运动或停止运动，运动过程中机器狗的步伐完全由单片机控制



2、系统初始化

详见 树莓派初始化文档
和 Keil创建STM32项目并烧录使用

3、树莓派摄像头监视与距离获取



> 具体步骤如下

----------还没写---------

> 参考资料如下

1. 开机启动使用
    - 自启动服务应在 `/etc/systemd/system/` 文件夹下，但为了方便，服务文件在被自启动的文件所在文件夹也有一份备份，请在该备份中更改，并在更改后将其复制到应在的文件夹，如 `sudo cp distance_measurement.service /etc/systemd/system/distance_measurement.service`
2. 定时使用脚本
    - 最低使用间隔：1min -> 可通过循环命令和 `sleep` 命令解决
    - cron 服务启、停、状态查看
    - 注意：

- 调用的脚本中一定要使用绝对路径，或在内部自行定义路径，一定不要直接使用全局的环境变量，否则既不会报错，也不会正常运行
- cron 命令调用生成的内容有时候会自带写保护，影响不大

3. 摄像头使用
   - libcamera 命令
     及其 官方使用文档
4. 特定图形（颜色块）测距

4、串口信号发送与接收

4.1、树莓派的串口发送

> 具体实现

- 初始化开启串口
- 使用 python 代码控制串口发送
  - python 环境配置并激活虚拟环境
  - 创建 `.py` 文件并在文件中添加如下代码（详情查看下方 `代码 4.1.1`、`代码 4.1.2`）
  - 运行刚才的文件（详情查看下方 `代码 4.1.3`）
  - 接收查看：树莓派与电脑正确连接后打开任意串口助手软件开启串口后皆可看见结果

```
# 代码 4.1.1
# 创建名为 python 文件；xxx 是文件名，可更改；`.py` 不允许更改
nano xxx.py
# 有些文件夹创建文件会要求根权限，则这么写，当然，最好不要在这种文件夹下操作
# sudo nano xxx.py

# 运行后会自动进入创建的文件
```

```
# 代码 4.1.2
# 以下代码添加到刚刚创建的文件（创建后会自动进入）
import serial  # 添加串口包

ser = serial.Serial("/dev/ttyAMA0", 115200)  # 设置串口端口和波特率，前者自行查看，一般用本部分的串口初始化者和我的是一样的
ser.flushInput()  # 清除缓存
ser.write("Hello World\r\n".encode())  # 发送数据  \r\n可以实现换行  encode()默认是'utf-8'
# ctrl + o，回车：保存
# ctrl + x：退出该文件
```

```
# 代码 4.1.3
# 运行刚刚的文件
python3 xxx.py    # python3 文件名
```

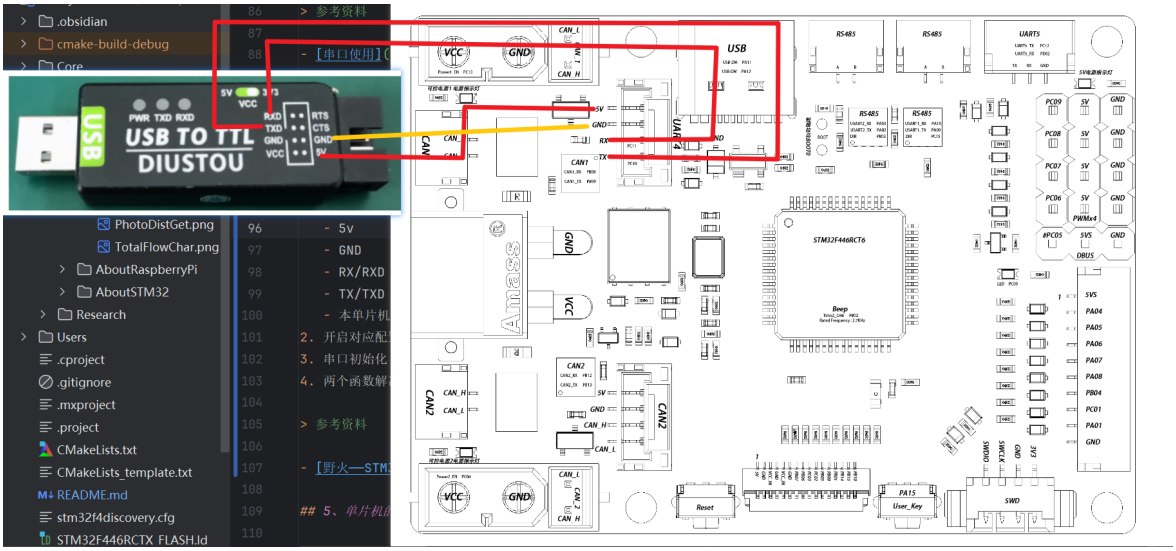
> 参考资料

- 串口使用

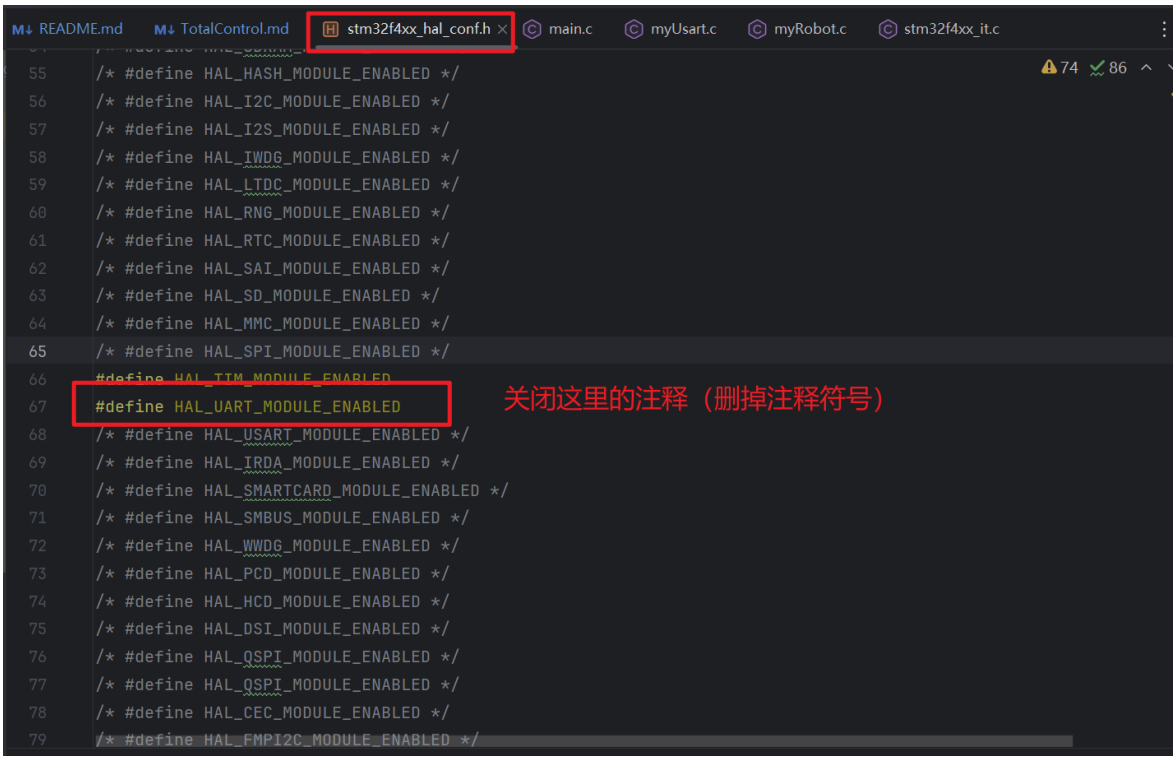
4.2、单片机的串口接收

> 使用步骤

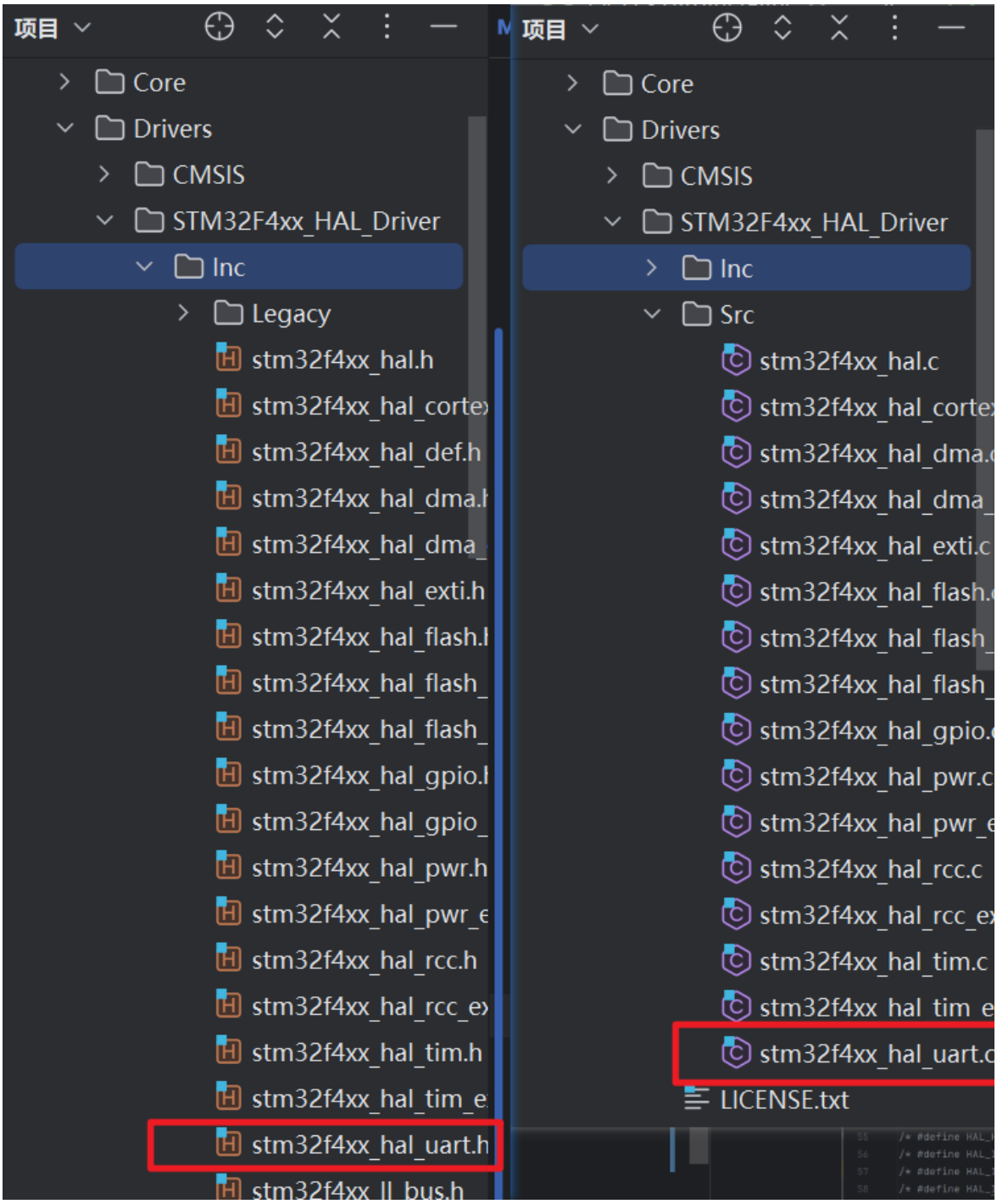
1. 正确接线
    - 一般串口接线都用四根线，本单片机也是选择四线制，下面是两个设备（设备A - 设备B）的串口连线，A 和 B 分别是什么无所谓
    - 5v - 5v
    - GND - GND
    - RX/RXD - TX/TXD
    - TX/TXD - RX/RXD
    - 本单片机和转接模块（USB 转 TTL）接线如下



2. 开启对应配置并检查文件完整性
    - 找到 stm32f4xx_hal_conf.h 中对应定义的定义并解除注释



- 查看 驱动文件夹 下是否存在
  串口驱动文件 stm32f4xx_hal_uart.c
  和 stm32f4xx_hal_uart.h



- 如果没有，请自行在 官网 下载整个 HAL
  库，并分别复制这两个文件到正确的文件夹下

3. 串口初始化（详见 myUart.c 注释）
   - 关于代码重写：库文件的函数不能修改，故为了实现初始化，必须重写；同时，这种不允许修改的函数是个 `__weak`
     （即弱定义）函数，在自写同名函数时将不会使用带 `__weak` 的函数

```c
/**...*/
__weak void HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(huart);
  /* NOTE: This function should not be modified, when the callback is needed,
           the HAL_UART_MspInit could be implemented in the user file
   */
}
```

4. 两个函数解决串口常规使用问题
   - 串口发送函数：`HAL_UART_Transmit(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size, uint32_t Timeout)`
     - `UART_HandleTypeDef *huart`：已经初始化的串口结构体
     - `const uint8_t *pData`：将要发送的字符串
     - `uint16_t Size`：将要发送的字符串的大小
     - `uint32_t Timeout`：超时响应，超过设置的时间无论是否发送完成将会跳出串口发送
   - 串口接收函数：`HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)`
     - `UART_HandleTypeDef *huart`：已经初始化的串口结构体
     - `const uint8_t *pData`：将要接收的字符所存放的指针
     - `uint16_t Size`：将要接收的字符的大小
     - `uint32_t Timeout`：超时响应，超过设置的时间无论是否接收完成将会跳出串口接收
5. 另外：串口同样可以使用中断来控制，不过由于暂时无该需求，就未研究

> 参考资料

- 野火——STM32F4XX——串口

5、单片机的 PWM 运动控制

> 使用步骤

5.1、简述

- 引脚输出的 `PWM` 信号控制 180 度舵机的 `输出角度`
- 使用 `高级定时器` 根据当前应该输出的 `脚步状态` 控制引脚输出 `PWM` 信号
- 使用 `基本定时器` 根据 `时间` 控制当前应该输出的 `脚步状态`

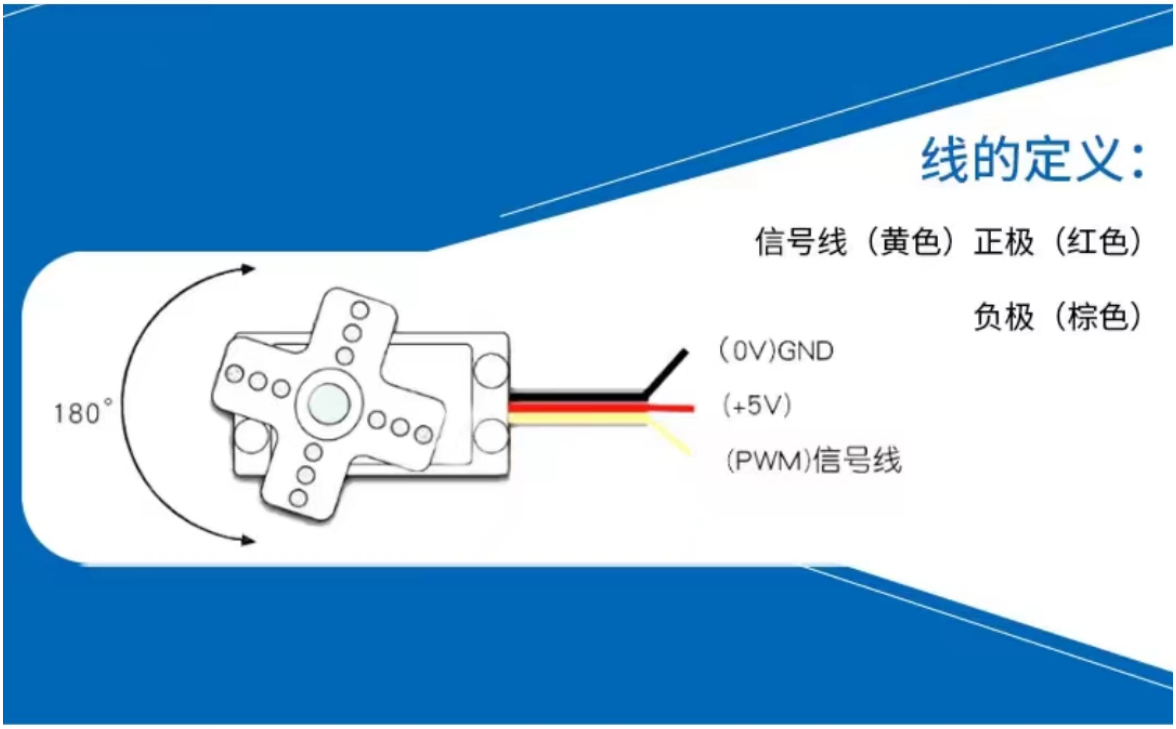
5.2、`输出角度` 与 `PWM` 信号对应关系

1. 舵机选择
   - MG90S-180 度舵机

- 【淘宝】限时满70减3 https://m.tb.cn/h.gSzxWhPoudoI3sg?tk=Y7Tf3bsHA3S MF6563 「SG90 9G经典舵机 180/360度
  数字舵机云台遥控飞机马达固定翼航模」
  点击链接直接打开 或者 淘宝搜索直接打开

2. 数据
   - 根据卖家所给数据来看，每次输入的信号不低于 20ms，信号占空比在 25% 到 75% 之间有效

产品重量	反应转速	使用温度	死区设定	转动角度	使用电压	结构材质
13.6G	0.11S（4.8V）	0°C~55°C	5微秒	180度左右各90	4.8V	空心杯电机



舵机的控制一般需要一个20MS左右的时基脉冲，该脉冲的高电平部分一般为0.5MS-2.5MS范围内的角度控制脉冲部分，总间隔为2MS。以180度角度为例，那么对应的控制关系是这样的：

0.5MS--------0度；　　1.0MS--------45度；
1.5MS--------90度；　　2.0MS--------135度；
2.5MS--------180度；

产品参数

SG90 9G舵机

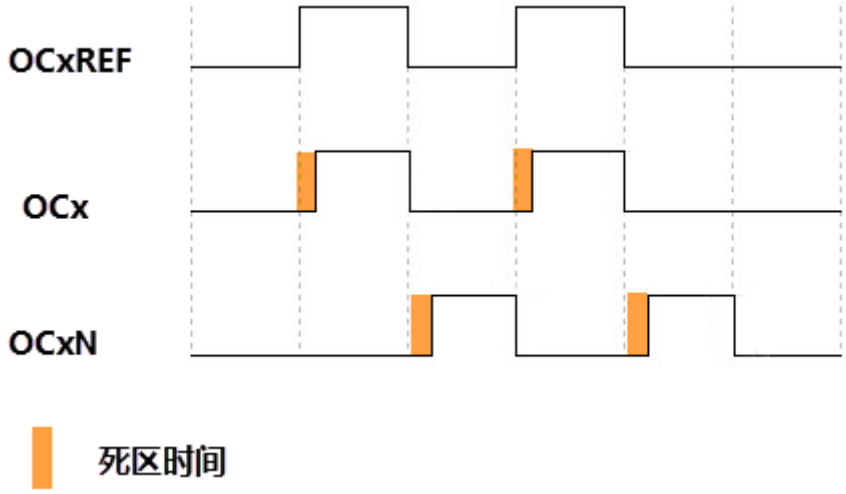
产品型号	SG90
产品重量	9G
工作扭矩	1.6KG/CM

反应转速	0.12-0.13秒/60°
使用温度	-30°C~+60°C
死区设定	5微秒
插头类型	JR、FUTABA通用
转动角度	180度（左90，右90）/360度
舵机类型	数码舵机
使用电压	3-7.2V
结构材质	塑料齿
线　　长	约25CM
附件包含	黑色QC标、多功能舵脚、摇臂、固定螺丝灯等
适用范围	固定翼、直升机KT、滑翔、小型机器人、机械手等模型

5.3、 高级定时器 控制 PWM 输出信号

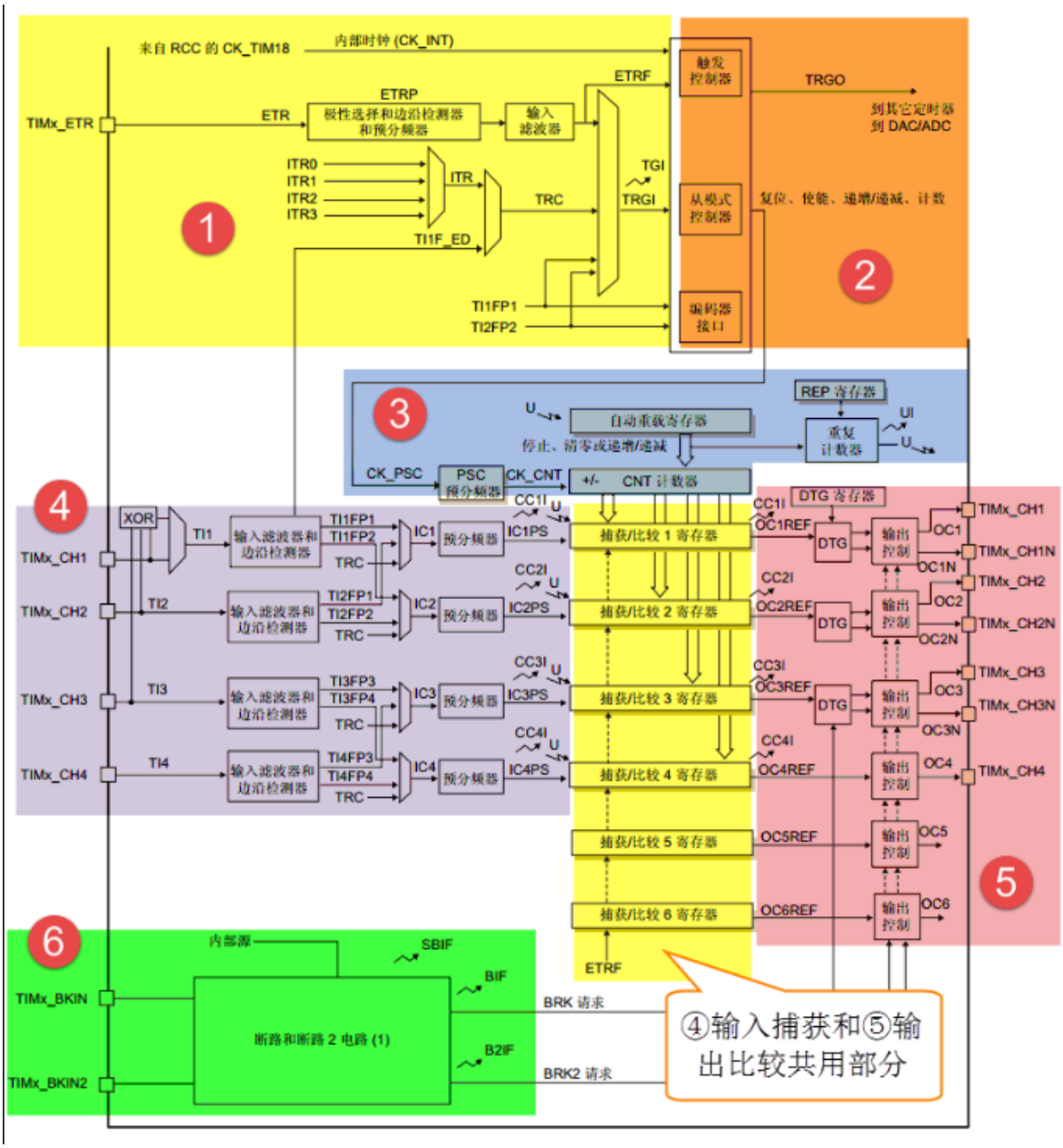
定时器类型	Timer	计数器分辨率	计数器类型	预分频系数	DMA请求生成	捕获/比较通道	互补输出	最大接口时钟(MHz)	最大定时器时钟(MHz)
高级控制	TIM1和TIM8	16位	递增、递减、递增/递减	1~65536(整数)	有	4	有	90 (APB2)	180
通用	TIM2，TIM5	32位	递增、递减、递增/递减	1~65536(整数)	有	4	无	45 (APB1)	90/180
	TIM3，TIM4	16位	递增、递减、递增/递减	1~65536(整数)	有	4	无	45 (APB1)	90/180
	TIM9	16位	递增	1~65536(整数)	无	2	无	90 (APB2)	180
	TIM10，TIM11	16位	递增	1~65536(整数)	无	1	无	90 (APB2)	180
	TIM12	16位	递增	1~65536(整数)	无	2	无	45 (APB1)	90/180
	TIM13，TIM14	16位	递增	1~65536(整数)	无	1	无	45 (APB1)	90/180
基本	TIM6和TIM7	16位	递增	1~65536(整数)	有	0	无	45 (APB1)	90/180

1. 初始配置
   - 开启对应配置并检查文件完整性：同串口，不过开启的是 `TIM`
   - 代码配置：详见 myYIM.c 的 `高级定时器使用` 部分的注释
   - 关于死区：在生成的参考波形 OCxREF 的基础上，可以插入死区时间，用于生成两路互补的输出信号 OCx 和
     OCxN，死区时间的大小必须根据与输出信号相连接的器件及其特性来调整。



- 关于通道与引脚对应：这次我们使用的是 TIM8，通道 1、2、3、4 分别对应引脚 PC6、PC7、PC8、PC9（也可自行调整，不过只有这几个在本开发板上引出针脚了，图中的
CHx 是各个通道）

	高级控制		通用定时器									
	TIM1	TIM8	TIM2	TIM5	TIM3	TIM4	TIM9	TIM10	TIM11	TIM12	TIM13	TIM14
CH1	PA8/PE9/PC9	PC6/PI5	PA0/PA5/PA15	PA0/PH10	PA6/PC6/PB4	PD12/PB6	PE5/PA2	PF6/PB8	PF7/PB9	PH6/PB14	PF8/PA6	PF9/PA7
CH1N	PA7/PE8/PB13	PA5/PA7/PH13										
CH2	PE11/PA9	PC7/PI6	PA1/PB3	PA1/PH11	PA7/PC7/PB5	PD13/PB7	PE6/PA3			PH9/PB15		
CH2N	PB0/PE10/PB14	PB0/PB14/PH14										
CH3	PE13/PA10	PC8/PI7	PA2/PB10	PA2/PH12	PB0/PC8	PD14/PB8						
CH3N	PB1/PE12/PB15	PB1/PB15/PH15										
CH4	PE14/PA11	PC9/PI2	PA3/PB11	PA3/PI0	PB1/PC9	PD15/PB9						
ETR	PE7/PA12	PA0/PI3	PA0/PA5/PA15		PD2	PE0						
BKIN	PA6/PE15/PB12	PA6/PI4										
BKIN2	PE6	PA8/PI1										



2. 使用
    - 先在 `main.c` 使用自写的 `MX_TIM_Advance_Init` 初始化 PWM
    - 后通过 `__HAL_TIM_SetCompare(__HANDLE__, __CHANNEL__, __COMPARE__)` 调节占空比
        - `__HANDLE__`：调节的定时器，这里填 `htim_advance` 即可，已经设置好了 `extern` 关键字
        - `__CHANNEL__`：通道号，调哪个通道写哪个即可，我的代码中使用 `LEF_SMALL_RIGHT` 之类的是因为在 `main.h` 里 `#define` 了，方便代码和实际物件的对应
        - `__COMPARE__`：调整后的占空比数值，实际占空比为 `__COMPARE__/2^16`

5.4、基本定时器 **控制** 脚步状态

1. 初始配置
    - 开启对应配置并检查文件完整性：同串口，开启的也是 `TIM`
2. 使用/撰写方法

- 根据 启动文件 的中断向量表（146 行）在 中断处理文件 里编写对应的 `中断处理函数` 和 `回调函数`
- 在合适的文件中（一般单独创建一个文件用于放置定时器相关配置和使用），编写定时器的初始化函数
  - 详见 myYIM.c 的 `基本定时器使用` 部分的注释
- 在 `main.c` 中调用定时器初始化函数就完成了

3. 控制脚步
   - 通过前面提到过的 `__HAL_TIM_SetCompare(__HANDLE__, __CHANNEL__, __COMPARE__)` 控制输出的 PWM 占空比控制对应舵机的角度
   - 将机器狗走动时的体态分解，使其成为一个个状态，每个状态中的各个舵机输出的 PWM 信号不同，这样，一个状态就能变成一个函数，一个调节各个舵机到正确 PWM 值的函数
   - 创建 `Robot_Control` 函数，其核心是内部带有 `static` 关键字的参数 `state`，代表各个状态，`state` 是多少就会调用多少号状态
   - 定时器回调函数内调用 `Robot_Control` 函数，每次调用都会更新 `state`，又由于该参数是 `static`，每次调用 `Robot_Control` 函数时不会清零，而是会在原先的 `state` 值基础上进行更新

> 参考资料

- 野火——STM32F4XX——基本定时器
- 野火——STM32F4XX——高级定时器