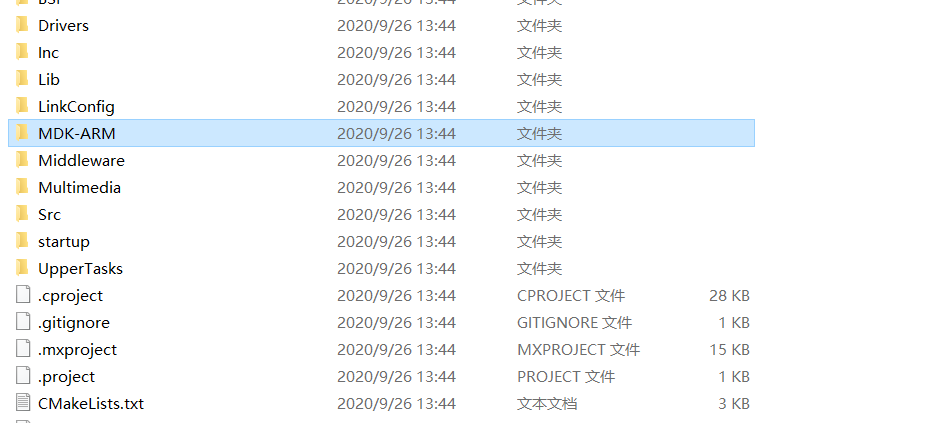
拿到代码后请先按照一下步骤测试一下keil环境是否配置成功：

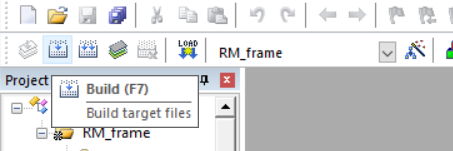
1. 进入指定位置：

->MDK-ARM->使用keil打开RM\_frame.uvprojx





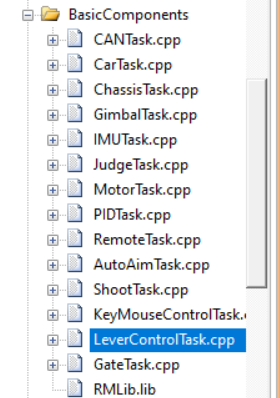
1. 点击左上角的编译按键（快捷键F7）



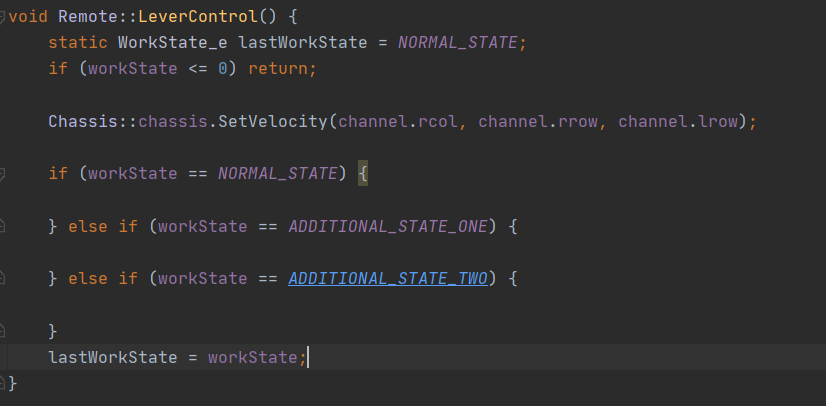
若编译成功则表示环境配置没问题。

如有问题，请在校内赛群内咨询。

打开BasicCompoments文件夹，找到LeverControlTask.cpp



（遥控器右端第三档是STOP模式，用于应急处理，会将所有电机的电流置零。比赛的时候请将其置于第一档，即遥控模式，出现问题可以切换成第三档急停。切换为第三档后可再次切换成第一档正常使用）



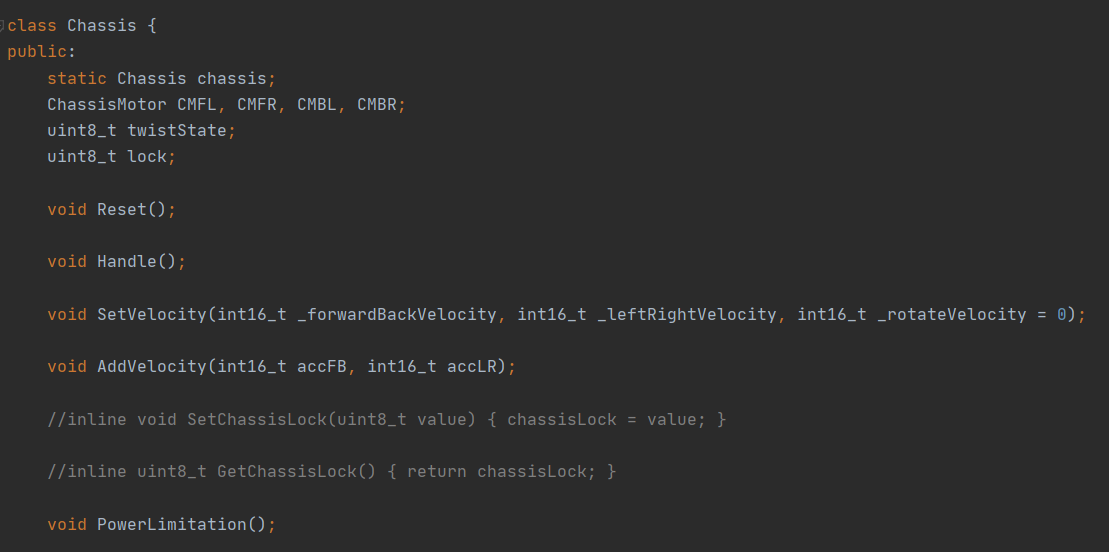
其中NORMAL\_STATE代表左（上）方拨杆第一档，ADDITIONAL\_STATE\_ONE第二档，以此类推。

用户可在对应的if-else语句段中写入自定义语句。请勿删除任何自带的代码。

1. 控制电机：

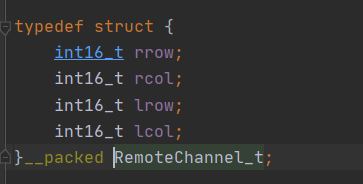
本框架采用的是C++语言，具有鲜明的面对对象编程的特色。将电机封装成了一个大类，然后根据不同的功能，通过继承创建了多个派生类，比如控制底盘电机的Chassis类等。

用户在使用该框架的时候，只需了解相关的接口，不需要关注接口的实现方式。



对底盘电机而言，用户最常用的是SetVelocity这个函数，用于设置底盘朝某个方向的速度，第一个参数代表前后移动速度，第二个为左右，第三个为转动速度，分别以前，右，顺时针为正方向。速度的大小以为宜。

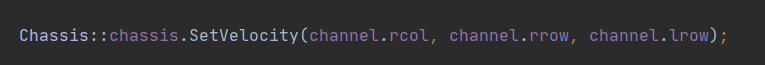
同时用户可以通过channel结构体（其参数如下图所示）



获得遥感对应的数值。（摇杆的不同位置会被映射成一个值，范围约为(-500~500),r代表右边摇杆，row代表横向，即rrow代表的是：右边摇杆当前位置在横轴方向上对应的（映射出的）数值。）

用户可以直接以channel.rcol方式来使用。

推荐用户以如下方式进行操作：



（上面的代码的意思是，将拨杆位置映射得的值分别作为三个速度的数值。最终会实现：右拨杆向上拨前进，向下拨后退，向右拨向右横移，向左拨向左横移。左拨杆向右拨动顺时针转动，反之逆时针转动。这是比较符合使用习惯的一种操作方式，当然用户也可自定义控制方式。）

已经写在LeverControlTask.cpp中，用户可以对三个参数乘以一定的修正系数，以控制车的三种速度的大小。

比如

Chassis::chassis.SetVelocity(channel.rcol\*0.1, channel.rrow\*0.1, channel.lrow\*0.1);

底盘电机的电调ID默认为：

为了表示方便，以下将采用简称，前后分别以F(ront)和B(ack)表示，左右以L(eft)和R(ight)表示

电机位置 ID

FL 1

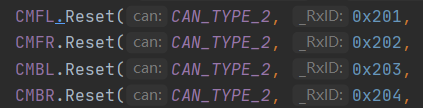
FR 2

BL 3

BR 4

用户也可自定义，但是不是很推荐。

若不得不更改电调ID，请在keil中Ctrl+F->Find in Files 搜索到以下代码段，电机ID到RxID的映射关系为：（设电机ID为i）



另外，CAN\_TYPE\_2代表着该电机使用的是CAN2线。

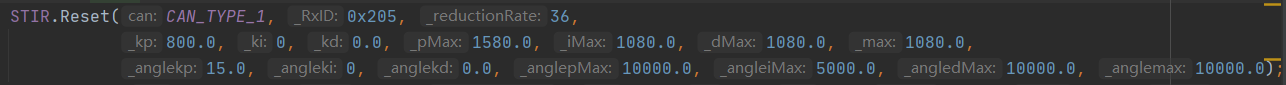
综合来讲，所有电机（至少是校内赛用得着的电机）均如此。

CAN\_TYPE\_1代表用can1，CAN\_TYPE\_2代表用can2

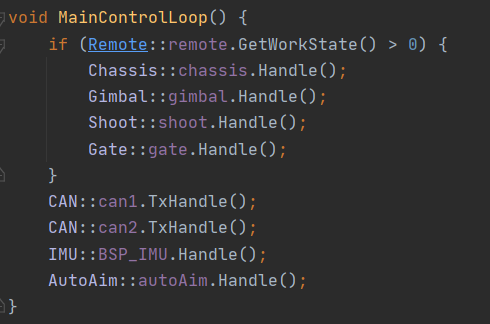
RXID均满足

用户可以恰当地使用Reset函数按照自己的需求分配和设置电机。

如果会PID的话，可以修改Reset中的参数。



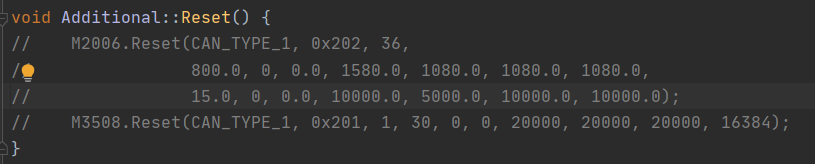
该程序框架的运行主要放在CarTask.cpp中，Car::Reset()完成所有东西的初始化。运行时循环调用MainControlLoop()。

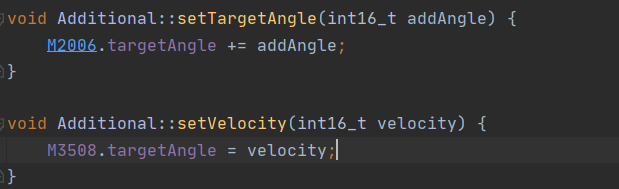
当然，用户也可以加入自定义的电机，请打开根目录下的Additional文件夹中的Additional.cpp,Additional中定义了两种电机，电机可以自己再额外定义，不过需要修改对应的函数（可以类比写好的函数修改）。

其中ChassisMotor是给定targetAngle后会以恒定速度转动，NormalMotor则是给定targetAngle转到指定角度后停止。请自行根据需求选择使用。

同时，如需使用，请修改Reset中的参数值，各参数意义上面已经阐述。



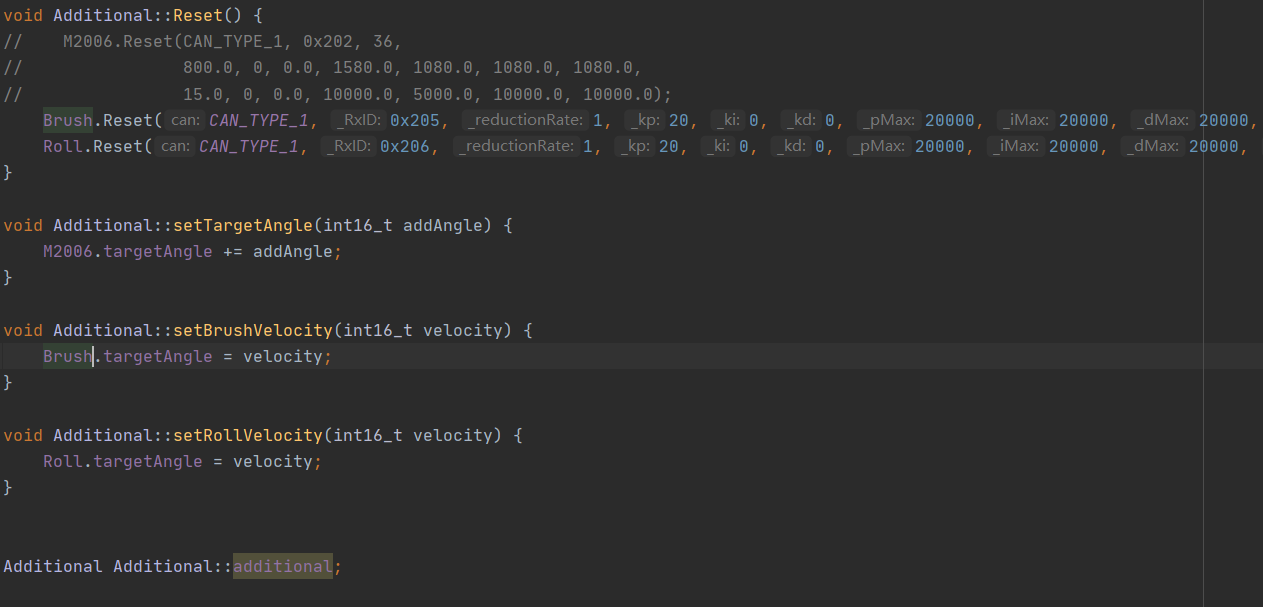
这边写好了两个样例函数，



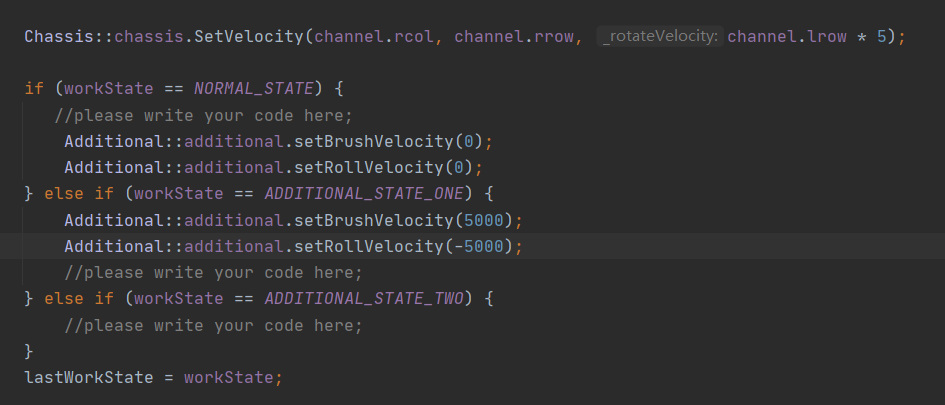
分别控制两个电机，用户可以结合电机的种类和自己的需求对函数进行改动或者创建自己的函数。



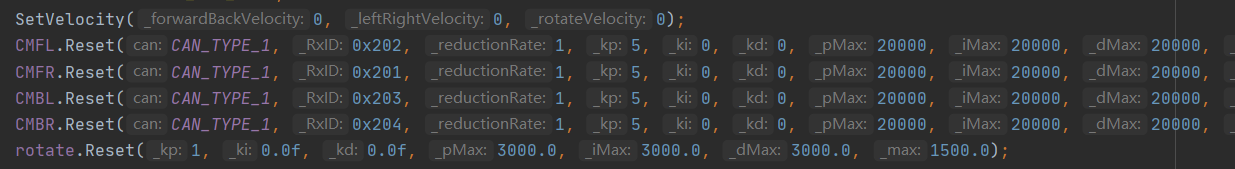
为了让用户更好地了解如何使用该框架，我用去年的冠军车Roboslave做了示例。该车主要由传送带电机，毛刷电机和四个底盘电机组成。前两个分别命名为Roll和Brush。由于都是给定一个速度然后恒定转动的电机，所以都用的是ChassisMotor这个类。



Additional中的程序段



LeverControlTask中的程序段



ChassisTask中语句段

调用时请：

Additional::additional.setTargetAngle(20);等等

舵机：

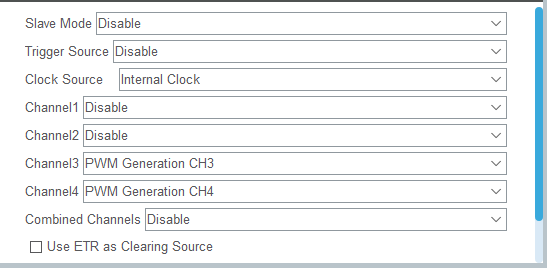
用户可以采用传统的舵机或者采用串口控制的舵机。

传统舵机：

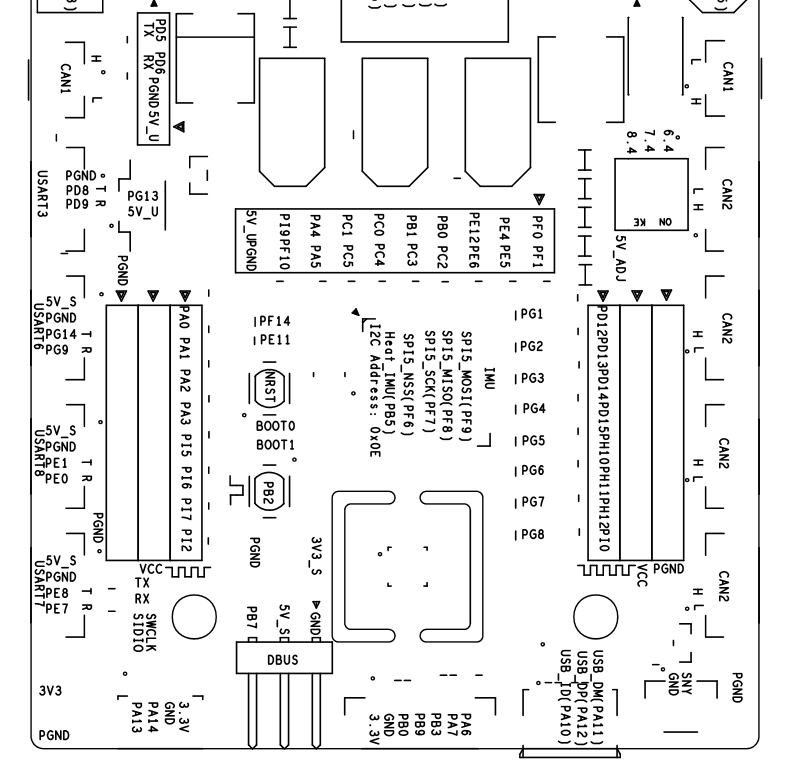
分为180度舵机和360度舵机，两种舵机当然还可以根据扭矩来划分。请用户根据自己的需求采购。

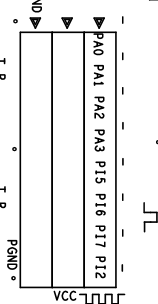
舵机是通过占空比来实现控制的。

180度舵机通过修改占空比可以控制其转动的位置，而360度占空比控制的则是角速度。（占空比的值在某个中间位置时，舵机将会停止转动。否则将会根据不同的占空比的值以顺时针或者逆时针的方向转动。）

在框架中已经配置好了两个能产生PWM波的通道，

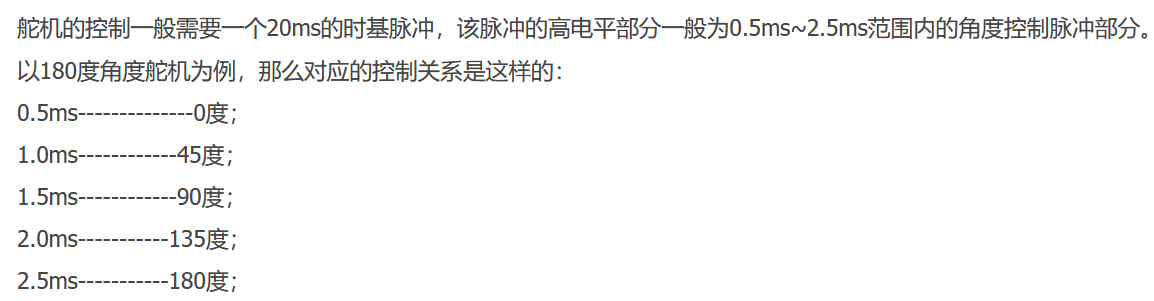
对应的引脚为：PA2 和 PA3（位于开发板左下角）





用户可以将舵机的信号线接到对应引脚，其余两根线分别接高低电平。

设置占空比可以使用HAL库中的函数\_\_HAL\_TIM\_SET\_COMPARE（）



下面的例子为让180度舵机转动90度的代码：



也可以改成TIM\_CHANNEL\_4，保证接线正确即可。

第三个参数除以1000即为对应的占空比时间。

请保持第一个参数不变。

360度舵机的控制请自行百度，与180度类似，无非就是修改占空比。