2020 夏现代电子系统设计综合创新实验报告

基于 PSoC6 的多功能平衡车

2018011702 自 84 孙浩文

目录

1 故事的开始

1.1 设计背景与灵感来源

时至今日，越来越多的智能无人系统进入我们的视野。这其中，便有视觉冲击力极强的平衡车。无论是日常载人的平衡车，还是例如在机场、餐厅中看到的那些已经被投入使用的平衡车，其独特而具有视觉美感的造型设计以及相比于四轮车的低功耗、便捷等特性已经逐渐被大众所接纳，也有望在未来占有相当可观的市场份额。

与此同时，组内成员对于平衡车的构造、平衡原理、运动控制等内容都极为感兴趣，在进行一系列的项目评估以及头脑风暴后，我们一致认为这是一个可行的选题，因此有了最后作为成品出现在验收现场的基于 PSoC6 的多功能平衡车。

1.2 基本分工设计与器材准备

由于我们组进行的是团队协作，因此我们期望能够让我们做出来的平衡车兼具多样化的功能。我们组的期望是一个硬件连接上完全独立、但却要受到主机端控制的智能平衡车，因此在最初，我们主要分为了硬件组和软件组两个大组，由李晟永与我共同负责硬件组的工作，包括硬件的搭建、调试、相关底层驱动的编写以及软硬接口对接，而佴瑞乾、郑新扬、廖崇骅则主要负责研究更偏向软件的部分，例如车体控制算法的研究、WiFi 通信的研究设计、语音识别和二维码识别等外设功能的实现等。

不过，随着我们工作的不断推进以及实际开发周期的限制，每个人或多或少地参与了其它一部分的任务，由于车辆各个部分的耦合关联程度较高，很多情况需要大家的配合与集思广益才能完成。

而在器材的前期准备方面，考虑到曾经电机系举办过智能车的比赛，且西主楼较为临近，因此我们从电机系借来了平衡车要用到的基本材料，包括了两节 3.7V 的充电电池、两个直流编码电机、 降压模块以及电机驱动模块，借助最为关键的 GY901 加速度传感器，我们已经可以完成基本平衡车的搭建了。而在这之后，考虑到通信的需求以及后续功能的实现，我们又购买了无线摄像头、DT06 Wifi 模块，配合上实验的核心控制板 PSoC6，来最终完成对于整个车的搭建。

2 硬件设计与硬件驱动编写测试

2.1 硬件结构设计

为了在保证车体能够平衡的同时尽可能地实现我们预期的功能，同时又能够达到其实用价值高的特点，我们硬件组在对于车体最初的设计时进行了考量，最终完成后整体车辆造型如图 1 所示（更多照片与视频见对应作业栏的上传）。

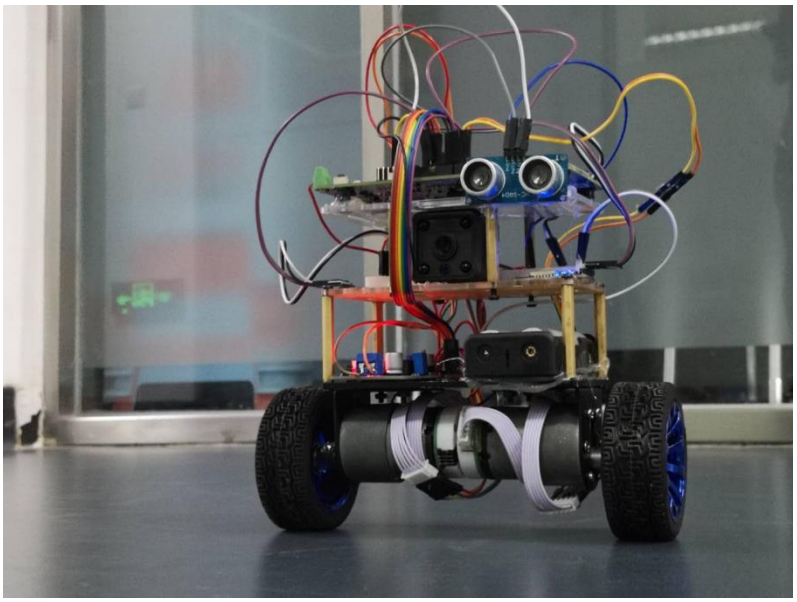

图 1 平衡车实物图

与之对应的结构图（省略亚克力板和铜螺柱）如图 2 所示。

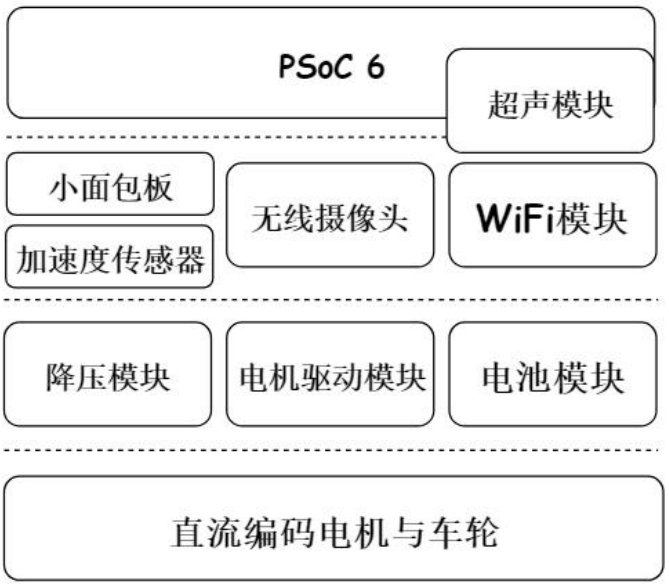

图 2 平衡车模块化结构图

结合上一页所示的两张图，我们可以看出，为了在不丢失模块的同时压低车体重心，我们首先将众多元件压缩至两层半（PSoC6 可视为半层），考虑到降压模块等的体积以及加速度传感器需要稳定平衡的平面作为依靠，这几乎已经是最为精简的层数；其次，各模块也依据本身特性与需求进行了层次分配，较为沉重的模块，例如电池、降压模块等均被放置在最底层，在有效压低重心的同时，又与电机本身距离最为接近，方便于完成对于电机的供电与驱动，加速度传感器放置在第二层能够更为灵敏的对于倾角变化进行感知，摄像头在第二层能够拥有相比于底层更好的视野，不至于总是看到地面。

综合以上多种因素的考量，我们最终完成了整体的结构设计。事实证明我们的思考行之有效。

2.2 硬件通路设计

完成整体硬件结构的设计方案后，我们又对于硬件的数据通路与供电通路进行了规划，并随着硬件外设的连入不断进行着增补，最终整体图如下：

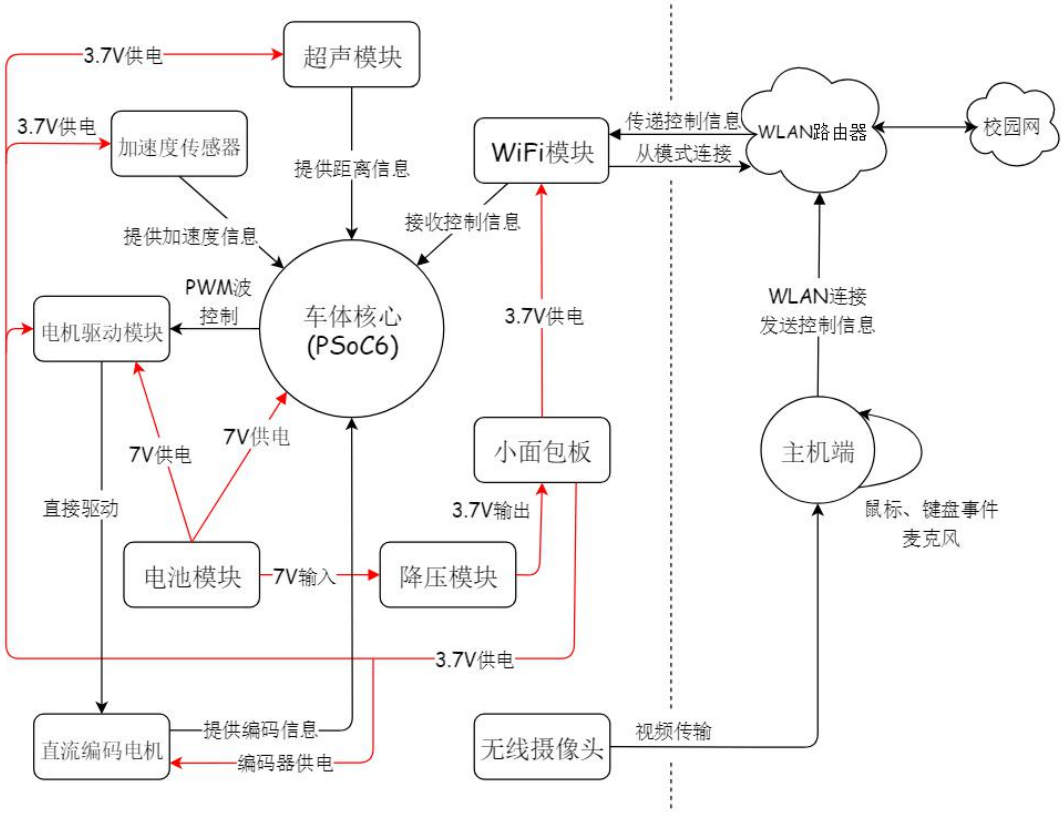

图 3 平衡车数据通路与供电通路图

从数据通路的角度（图 3 所有黑色箭头）而言，平衡车以 PSoC6 为核心，接受来自各个模块的信息，经过算法处理后输出 PWM 波对电机驱动模块进行控制，并由电机驱动模块直接驱动电机。而在另一端，我们通过 WiFi 模块、主机端同时连接同一个 WLAN 路由器，在达到主机端与 WiFi 模块通信的同时也能够保证主机端的外网连接。由于无线摄像头自带内部电池且联网后即可传输视频，因此无线摄像头联网后直接传输至主机端。

从供电通路的角度（图 3 所有红色箭头）而言，供电线路可分为 7V 与 3.7V 两种电压的供电，其中 7V 主要为 PSoC 本身、电压驱动模块和降压模块作供电与输出，而 3.7V 针对的则是其余模块与编码器。这里需要说明的是电压驱动模块既有 7V 也有 3.7V，这是因为 7V 接入该模块的 VM，即电机真实需要的驱动电压，而 3.7V 相当于该模块自身接入的 VCC。

从图中可以看出，虽然本次实验中实用的模块数量众多且硬件属性与需求各不一样，同时又涉及到了各类有线或无线传输以及两个处理器（PSoC6 与主机端）之间的协调配和，但是在我们合理的规划和设计下，整体的通路还是清晰且流畅的，这也是我们得以将这个实验顺利完成的基础。

2.3 硬件底层驱动设计与编写

在利用算法控制小车的平衡之前，我们硬件组进行了小车基本硬件底层驱动的设计和编写，即在软硬件接口层面上，如何让所有模块与传感器获得的数据能够被简洁、正确、高效地读取，以供后续算法的实现，以及如何对于电机时时提供正确的 PWM 波进行驱动是这一部分的关键。

2.3.1 PWM 控制驱动编写

根据电机驱动模块的连线结构，对于单个电机，除了提供 PWM 波的输入外，还需要提供两个电平信号以确定电机的正转、反转以及停止，因此，我决定对于每个电机各使用三个 GPIO 口进行信号的输出，并引入 speed（即目标速度值）来控制：使用 speed 的正负号来确定两个电平信号以控制电机正反转，使用 speed 的绝对值来控制输出 PWM 的占空比，从而使得电机的转速变化。

由于我们期望小车实现转向功能，且两个电机的性能必然不会完全一样，因此采用了左右轮分别控制的方式，即设立两个预期速度 left 和 right，分别进行调节。

在本驱动中使用的主要技术为设置 PWM 的占空比，虽然现在看来非常基础，但是由于这是硬件组编写的第一个驱动，因此其为我们后续编写驱动提供了经验。最关键的函数即为 Cy_TCPWM_PWM_SetCompare0，该函数通过改变 PWM 模块的 Compare0 值来改变 Compare0/Period0 的值，而这就是 PWM 的占空比。核心逻辑即为确定目标速度的正负值，然后对两个电平输出和一个 PWM 输出分别调节，这里以左轮且目标速度大于 0 为例，代码如下：

```
if(left >= 0){
        Cy_GPIO_Write(BIN1_PORT, BIN1_NUM, 0);
        Cy_GPIO_Write(BIN2_PORT, BIN2_NUM, 1);
        if(left + deadValueLeft> 3000)
            Cy_TCPWM_PWM_SetCompare0(B_PWM_HW, B_PWM_CNT_NUM, Amplitude);
        else
            Cy_TCPWM_PWM_SetCompare0(B_PWM_HW, B_PWM_CNT_NUM, left + deadValueLeft);
    }
```

可以看到在目标速度和最低阈值相加超过 3000 时仅输出 3000，其余情况则输出目标速度和最低阈值之和，可以保证 Compare0 始终小于 Period0，确保占空比小于 1。而最低阈值则是轮子转动的最小 Compare0，否则轮子将由于电机转动前受到的静摩擦力相对大而导致电机不转。

基于此，我们编写了 Motor_timer.c 文件，其中 setPWM 即为核心控制函数。

2.3.2 编码器驱动编写与中断例程探索

电机速度的控制信号是瞬时的，因此只需要在需要改变速度时使用上述文件中的函数即可。但是编码器则需要进行实时的边沿读取以获得编码数，而显然让其作为一个一直轮循的任务对于资源的消耗过于巨大，因此我们结合计算机组成原理中学习到的知识，决定采用中断来进行有关处理。

在 PSoC Creator 中单独有 Interrupt 器件来对于某一信号进行中断。在一开始的时候，我对于这一器件进行了探索，并参考了 CE220263 这一示例工程，学习了单个 GPIO 输入信号进行上升沿或者下降沿跳变时触发中断的方法（以下称为方法一），其核心原理图与代码如下所示：

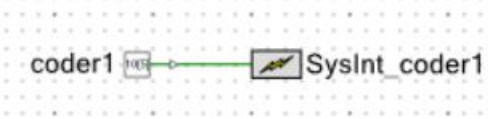

图 4 方法一中断原理图

```
// Init of way 1 of interrupt
Cy_GPIO_SetHSIOM(coder1_0_PORT,coder1_0_NUM, (en_hsiom_sel_t)coder1_0_INIT_MUXSEL);
Cy_GPIO_SetDrivemode(coder1_0_PORT,coder1_0_NUM, CY_GPIO_DM_PULLUP);
Cy_GPIO_SetVtrip(coder1_0_PORT,coder1_0_NUM, coder1_0_THRESHOLD_LEVEL);
Cy_GPIO_SetSlewRate(coder1_0_PORT,coder1_0_NUM, coder1_0_SLEWRATE);
Cy_GPIO_SetDriveSel(coder1_0_PORT,coder1_0_NUM, CY_GPIO_DRIVE_FULL);

Cy_GPIO_SetInterruptEdge(coder1_0_PORT,coder1_0_NUM, CY_GPIO_INTR_RISING);
Cy_GPIO_SetInterruptMask(coder1_0_PORT,coder1_0_NUM, CY_GPIO_INTR_EN_MASK);
Cy_SysInt_Init(&SysInt_coder1_cfg, Coder_1_Interrupt);
NVIC_ClearPendingIRQ(SysInt_coder1_cfg.intrSrc);
NVIC_EnableIRQ((IRQn_Type)SysInt_coder1_cfg.intrSrc);
```

在上述例程中，只需要修改 Cy_GPIO_SetInterruptEdge 函数中的上升沿或者下降沿即可对不同状态进行中断处理，而后修改 Cy_SysInt_Init 函数中的 Coder_1_Interrupt（此为自己写的中断函数）即可跳转并执行任意想要实现的中断函数功能。

但是，在随后的实验过程中，我们发现这一例程范式事实上并非真正的中断，其对于中断的触发过程不是按照我们想像的方式进行的，因此在寻找过程中询问助教后，我们又采用了基于 Timer Counter 的第二种中断方式（以下称为方法二），其原理图与初始化代码如下所示：

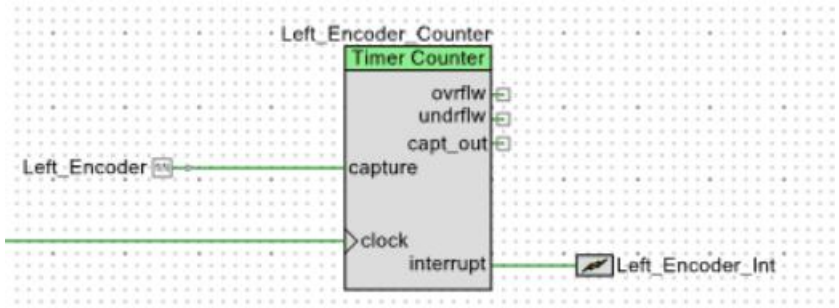

图 5 方法二中断原理图

```
// Init of way 2 of Interrupt
    Right_Encoder_Counter_Start();
    Cy_SysInt_Init(&Right_Encoder_Int_cfg, Right_Encoder_ISR);
    NVIC_EnableIRQ(Right_Encoder_Int_cfg.intrSrc);
```

其中，在原理图部分，我们需要对于 Timer Counter 进行设置，例如捕捉上升沿等，而在中断函数最后需要调用_ClearInterrupt()进行对中断的 Clear 操作，以保证退出中断以及后续的顺利进行。

在摸索出这一套中断范式后，我进行了基本的测试以及 UART 串口输出进行调试。当看到屏幕上正确地将编码器计数打印出来时，我知道我们掌握了一个相当实用的技巧。在编码器中的中断函数非常简单，即对一个全局变量不断的进行自加即可，并且根据编码器 B 相的电平同时判断轮子的转向：

```
void Left_Encoder_ISR()
{
    advanceOrBackLeft = Cy_GPIO_Read(Left_B_0_PORT, Left_B_0_NUM);
    countLeft ++;
    Left_Encoder_Counter_ClearInterrupt(CY_TCPWM_INT_ON_CC);
}
```

基于此，我们完成了对于 Encoder_Timer.c 文件的编写。

2.3.3 加速度传感器驱动编写

在有了上述中断例程探索的基础上进行加速度传感器驱动的编写就更为方便了。由于李晟永同学在之前曾经对这类加速度传感器有所了解，因此本部分的编写主要由他来完成。同样类似于编码器的驱动，加速度传感器驱动的中断函数中返回了车体绕 x, y, z 三轴的转角和角速度。

2.3.4 各个驱动的配合调试

虽然我们在实验前曾经预想过各个驱动模块配合在一起后可能出现问题，但在真实操作时，遇到的问题远比我们想象的多。

首先是 PSoC6 的 GPIO 端口问题。在尚且对 PSoC6 的各个端口属性不清楚时，我们按照车体的结构直接进行了排线布线，其中编码器的 AB 相接入了 PSoC6 上一排针形公头引脚，随之而来的，是串口调试助手上无论如何也无法显示出编码电机，且甚至中断例程都没有触发。在检查过硬件连接、代码可靠性后，我们最后判定是这一排针形引脚可能不便于进行输入，因此我们将连线稍作修改，使得其连入能够作为正确输入端的 GPIO 口，编码计数也随之恢复正常。

其次是串口通信时的波特率问题。我们使用的加速度传感器默认使用 9600 波特率，而事实上我们通过串口连接时使用了 115200 的波特率，因此导致了一次调试时加速度传感器返回值不正常的现象。我们借助了 USB 转 TTL 模块进行对于加速度传感器波特率的设置，完成后恢复了正常。

至此，平衡车基本功能实现所需要依赖的模块驱动已经准备完毕。

3 平衡算法以及平衡参数的调试设置

3.1 PID 平衡算法

尽管 PID 平衡算法在软件层面的实现是软件组完成的，但是为了小车平衡的各个参数的确定，我们也学习了 PID 平衡算法，因此在这里进行简单的说明。

PID 算法的核心表达式在于如下的三个等式：

$$a_b = kp_b \times \theta + kd_b \times d\theta$$
$$a_v = kp_v \times e + ki_v \times \sum e$$
$$a_t = kp_t \times \Delta\omega + ki_t \times \sum \omega$$

这三个等式分别是对于平衡直立、运动平衡以及转向平衡的加速度计算。由于工作频率非常高，因此每次计算得到的三个加速度可以视为下次速度与本次速度的差值。而表达式中的各个变量，例如倾角 θ 与其角速度等，可以从加速度传感器直接获得，而当前速度等的信息则可以通过编码器驱动读取计数后间接计算获得。因此，表达式的关键还是在于各个系数，即参数的给定。

3.2 平衡算法参数的选择

尽管只是不多的几个参数的调节，但事实上这不仅是我们本次平衡车得以完成的基础，也是硬件组花费时间较长的一部分任务。

由于缺乏有关经验的指导，在最开始的一段时间里，我们只是单纯的从视觉感觉出发进行调节，并没有踏实的理论依据，只是对于速度进行直观的感觉，因此最开始时候的调节显得相对而言可靠程度不是很高，时间成本却非常巨大。

在经历了一系列失败后，我试图通过理论层面进行一定的分析，从而指导我们进行参数的调节。我首先注意到在这三个表达式中，第一个直立平衡表达式是

最为基础与关键的，因为这决定了静态平衡能否正常。因此，我最先进行调节的正是第一个表达式中的两个参数，调节时将其余的参数暂时先置为 0。

就表达式而言，参数 kp_b 与铅垂线倾角直接相关联，结合我们需要的是倾角偏移的负反馈，因此在其符号的确定上，我根据正负值不同时轮子的转动方向进行了测试，让轮子满足车体整体向一侧倾倒时轮子也往那一侧加速，此时的 kp_b 为一个正值，由此确定了方向。

而在具体大小的选择方面，由表达式可以明显看到，kp_b 绝对值越大时，该项加速度也会越大，小车对于倾角改变的反应也会越大。因此，该值过小会造成小车反应过慢，从而无法及时地进行平衡调整，该值过大又会造成对小角度扰动反应过于灵敏，甚至会使得微扰被放大。

在实际调节的过程中，我采用了先给一个较大值，在这时候小车由于响应加快，会对于微扰出现大幅度的低频抖动，而后慢慢降低这个 kp_b 的绝对值，使得抖动逐渐减缓，从而固定了 kp_b。经过实验，本车的 kp_b 约为 340 左右。

在确定 kp_b 后，我接下来继续调节 kd_b 的值，其正负性同样可以利用轮子的转向进行判断。由于 kd_b 是角度微分项的系数，因此其作用主要是对于前面一次项的补充，通过调节该项应该尽可能的使得小车的直立平衡更为稳定平滑。在实验过程中我发现，当 kd_b 值过大时，其对于小车倾角的变化比前一项更为明显，即受到微扰后会出现更高频率的抖动，这也正是因为微分项的调节敏感度更高。结合这一发现，我在此基础上减小了 kd_b，直到小车刚放在地上时能够有一个较短时间段的平稳直立。经过测试，20 左右是一个可以接受的值。

仅有直立平衡等式，即上一页第一个等式的控制还是不足以使得小车原地平衡的，因为在调节过程中还会存在向前与向后的速度，有了速度则必须要结合到二式。有了上一次调节的部分经验，我同样开始进行控制变量式的调节。

首先在一开始，同样是对于 kp_v 系数正负的调整。通过参考网络资料我们发

现，寻常的速度控制往往是负反馈的效应，但是平衡车较为特殊，由直立平衡对轮子发出向 A 方向的速度指令时，整体车体也已经向 A 方向倾斜了，因此速度控制需要的是让该轮子进一步加速以赶上倒下的速度，单论这里的速度控制而言是一个正反馈。实验中我们利用这一个法则进行符号的判断，即在使用后旋转轮子，轮子应当越来越快直至最高速，确定了其符号为正。由于 ki_v 为积分项的系数，两者的符号应当统一，否则与理论预期矛盾，因此可直接确定 ki_v 符号为正。

在 kp_v 大小调节环节，我也进行了多次尝试。在一开始明显还是倒下的速度快于回复速度，但当我逐渐调节至 600 时，车体逐渐地赶上了其倒下的速度并完成了在一定范围内的平衡！这时正是 6 号晚上约 11 点，我在自己宿舍完成了平衡车首次的平衡，尽管参数还并没有那么完美，其平衡所需要的振荡幅度也比较大，但这一瞬间是我返校以来最为激动的一瞬间了。

随后，为了参数更加的完美，我在接下来的一天又与李晟永一起进行了调节，在增大 kp_v 至 750 左右的同时，我们又增大了其积分项的系数 ki_v 至 8 左右，能让小车在大约 5-8cm 的范围内保持稳定的静态平衡。

3.3 精益求精的调整

在已经做出静态平衡的基础上，我们认为这还不够，因此进行了更为精细化的调整。

首先，考虑到车体前后并非对称，因此我们对于运动平衡分别给出了针对于前进与后退的两套参数，并分别利用了 3.2 节中的调节方法进行了调试。

其次，由于整体变化过程的迅速，人眼可能无法捕捉到瞬间轮子的转向信息，因此我们采用了手机慢镜头延迟摄影，通过分析拍摄视频来捕捉这一信息。

与此同时，随着软件组同学们将转向控制进行编写，我们又对于 PID 算法中第三组转向参数进行调节。由于有了较好的直立平衡和运动平衡的参数，对于转向参数的精度要求并不是很高，但是我们仍然进行了选取，得到了转向时最为稳定的一组参数。

而对于左右轮的平衡，由于两个电机对于相同 PWM 波的响应不太一样，我们采用了两种手段进行调整。首先是单独对两轮测试并在 PWM 波的底层驱动文

件中对于较快的一轮在设置目标速度时加入小于 1 系数，以求达到两轮的平衡，第二种手段则是直接利用 PID 的转向调整等式进行控制，最终达到前进后退均走直线的目的。

可以说，在此基础上，小车基本的平衡功能得到了进一步的保障，这也正是平衡车最为核心的一部分。

4 控制调试设置

基于底层驱动以及 PID 控制算法，在我们测试的同时，软件组也提供了 CarTask 这一核心任务以支持前后左右的控制，并且研究了 WiFi 模块外设以提供控制，整体的控制代码结构示意图如下图所示（本图来自于佴瑞乾同学）：

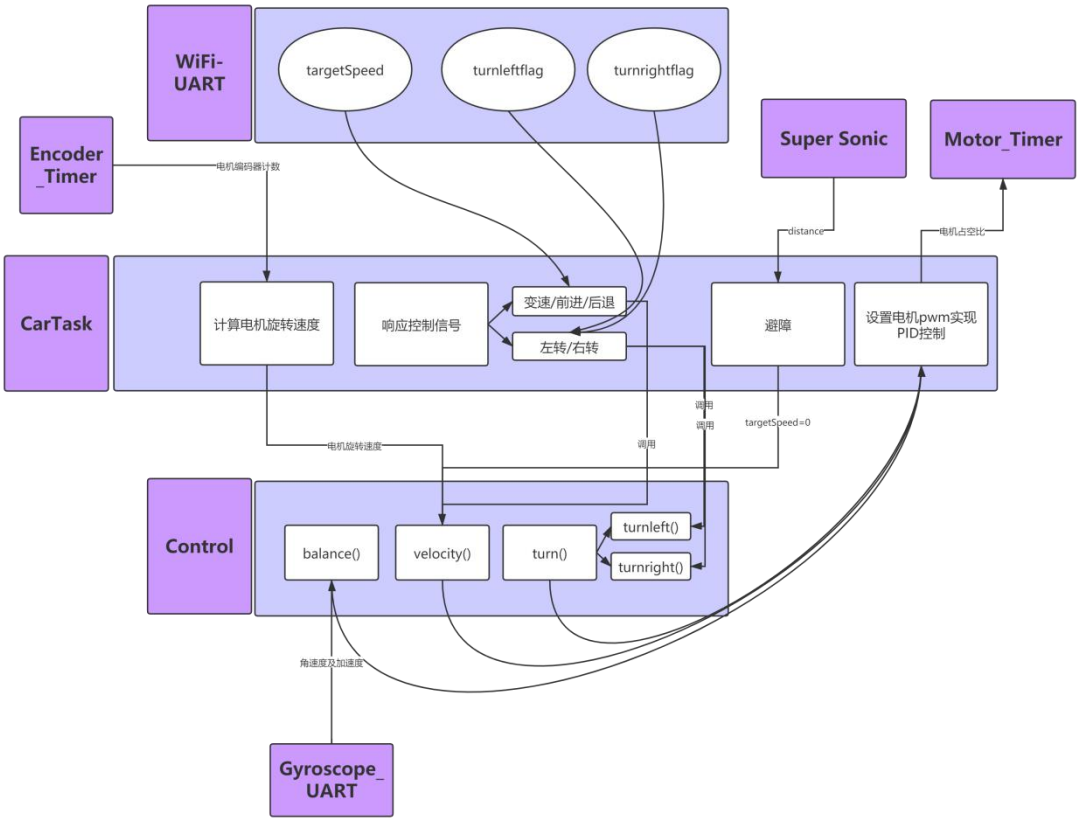

图 6 平衡车控制代码结构示意图

在一开始软件组的设计过程当中，速度的变化是分立的离散值，即目标速度的选值过于离散，且 PWM 波设置是瞬间完成变化的，在调试的过程中我们发现小车目标速度变化过于剧烈会极大地影响自身的平衡性能，因此后续我们完成了对于 PWM 波设置以及目标速度设置的平滑变化，这样能够使得小车的整体运行更为稳定。

而随着新模块的引入，我们又加入了超声模块作为避障的提醒，由于模块资源的限制，我们仅采用了一个超声模块，因此避障的最好方式就是当这个模块感应到墙体存在时，立刻给出信号让小车停止，即进入静态平衡状态。

最后，我们完成了在 PSoC Creator 中主工程的统一，最终的顶层设计图如图 7（下一页）所示。

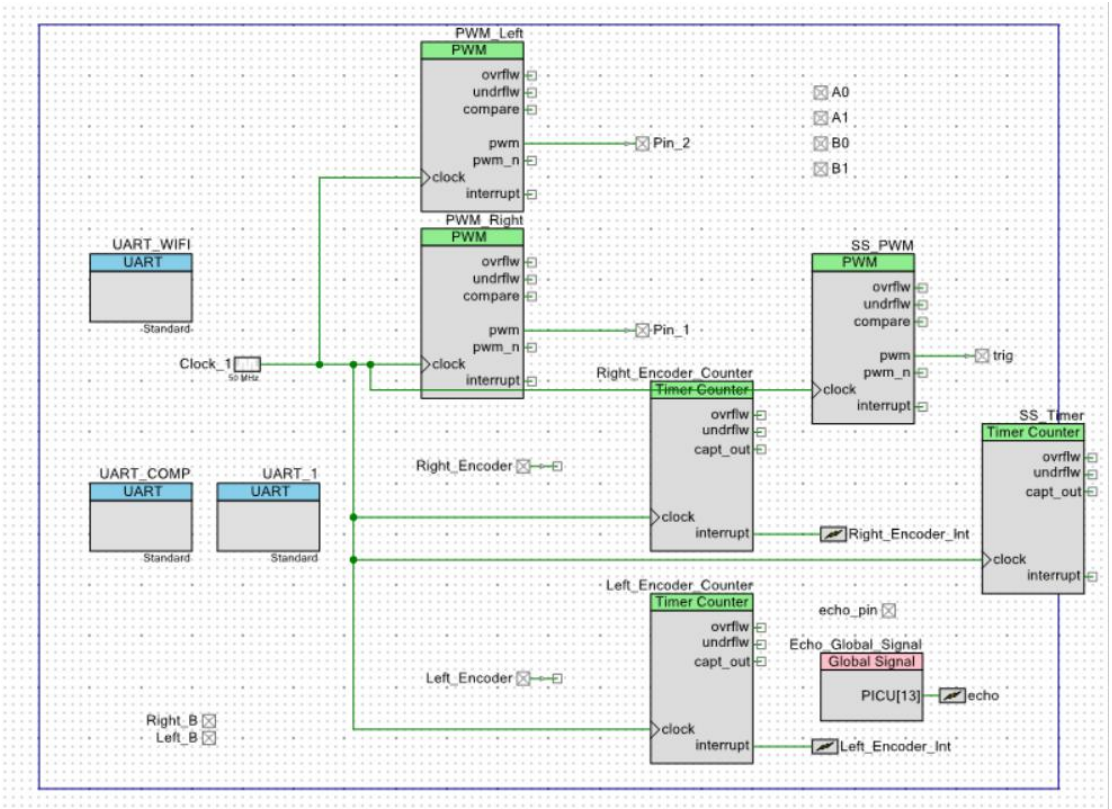

图 7 平衡车主工程顶层设计图

由于前期无论是硬件组驱动部分和参数设计，还是软件组的控制逻辑以及外设配备，我们都做得较为优秀，因此加入 WiFi 后的控制测试整体进展较为顺利。

至此，我们已经能够远程利用 TCP 调试工具完成对于小车的远程控制。

5 主机端控制程序的编写

尽管最核心的功能已经大致实现了，但作为一个有实用价值的电子系统，一个集成化的主机端控制程序也是必不可少的，毕竟我们不能一直在 TCP 调试软件中输入命令。与此同时，语音识别功能和视频传输扫描二维码的功能也决定了我们必须在主机端实现一个集合所有功能的全套应用程序。

经过一定的商量，我们决定让郑新扬同学研究二维码识别部分，让佴瑞乾同学研究语音识别的部分，我来进行包括控制端、视频流和各种功能整合的功能集成程序设计。

与去年小学期里制作的 UI 不同，本次的主机端控制程序涉及到了几个较为新的东西，包括基于 TCP 协议的通信、视频流的实时显示以及语音输入等，加上郑新扬和佴瑞乾提供的功能模块均使用 python 编写，因此思考过后我决定尝试基于 PyQt5 来完成控制程序的编写。

5.1 视频流的传输

在最开始，我们对于摄像头的构想是能够直接连在 PSoC6 上，而后通过数据线以及例如 SPI 协议等手段发送给 PSoC，再由高速 WiFi 模块传输至电脑端。但是，PSoC6 上有限的资源不允许我们进行这样的操作，并且基于对本次课程的时间考虑，我们没有那么充分的时间进行关于视频流过于深入的学习，因此权衡之后我们考虑直接采用无线摄像头进行传输。

郑新扬同学购买的无线摄像头自带电池且在电脑端有适配软件，只需要最开始让无线摄像头接入网络，即可在适配软件端观看实时视频。那么下一步就是如何让视频能够正常地显示在我写的应用程序上。在我和郑新扬探讨后决定使用虚拟摄像头 OBS 软件，其功能是可以创建一个虚拟摄像头，让电脑认为除了本地摄像头外还有另外一个摄像头存在。

在有了这个软件后，我希望能够将视频放入某个方形方框内。在 PyQt5 中，我使用了 QLable 进行视频存放，方法是通过 opencv 中的 cv2.VideoCapture 对象来首先捕捉到摄像头拍摄的画面，而后对于每一帧而言将其作为图片在 QLable 中进行显示，这样可以充分对接二维码识别部分的程序，即希望对于每一帧的图片进行二维码的识别判断。因此最终，二维码识别程序成为了我将视频从虚拟摄像头读取至我自己的程序的一部分，做到了较为完美的融合。

5.2 小车通信配合

编写过程中遇到的第二个障碍是如何与小车进行通信。由于利用了 TCP 协议，因此我临时学习了 socket 有关内容，通过调用 socket 对象以及相关函数建立了连接与发送信息，相关代码如下：

```
# tcp connect
self.target = ('192.168.4.1', 9000)
self.soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.soc.connect(self.target)
self.soc.send(bytes(text, encoding='utf-8')) # send info
```

经过上述操作，我成功完成了在命令行中运行并发送 text 信息以控制小车的操作。

但是，当时我们对于小车上 WiFi 模块的认知还过于粗浅，仅仅使用其主模式，导致电脑的 WiFi 连接被小车 WiFi 模块占用，电脑无法上网进行视频传输。即便是我接入了网线，也发现本地连接与 WiFi 的连接是冲突的，无法在一个时间内进行两者的兼顾，也就是说连上小车后就无法进行视频的传输，这样的话对于我们的整体功能实现无疑是一个巨大的打击。

不过很快，李晟永考虑到了 WiFi 模块具有的从模式，即其本身可以连入一个 WLAN 路由器，因此我们决定将路由器作为中转站，让 WiFi 模块与主机端均连入这个路由器并固定 WiFi 模块接入的 ip（我们这里定为 192.168.1.150），从而实现通信，且这个路由器允许上网，即不影响主机端的正常工作，达到了一举两得的效果。

在此基础上，我们解决了如何能够同时连接外网与小车了。

5.3 多线程工作

尽管解决了连接问题，但是又有一个问题出现了：我在测试时按动按钮发送信息时，视频传输同时会卡死，导致程序的崩溃。这其中的原理便是 Qt 中的槽函数事实上可以类比为在软件端的中断，点击按钮后进入槽函数会造成视频正常传输的“阻塞”，从而造成程序的崩溃。

考虑到这个问题，我参考网上资料学习了多线程，其原理就是建立多个线程并行处理任务。我将视频显示（包含了二维码识别）单独作为一个线程，而后将剩余部分中的 socket 部分也设置了一个线程，并且采用 lock 的方式限制其使用资源，使得程序整体能够稳定运行。建立与开始线程的核心代码如下：

```
self.t1 = threading.Thread(target=detect, args=(self.cap, self.Lab_camera, self.lock, self.label_9,
self.event))
self.t1.start()
```

而后在每一个需要限制资源保证流畅的过程中加入 lock 如下：

```
self.lock.acquire()
self.cap.open(1)
self.lock.release()
```

即可在多线程运行的同时保证整体的流畅性。

5.4 键盘事件与退出事件——改写原函数

由于在主机端，我们希望能够实现鼠标、键盘均可控制，这时候就需要对于键盘事件有所响应和读取，这里我采用的方式是改写原来的键盘事件函数 keyPressEvent()，并对应地完成对于各个必要按键的捕获，具体代码如下：

```
def keyPressEvent(self, event):
    if(event.key() == Qt.Key_A):
        self.queue.put('a')
    elif(event.key() == Qt.Key_W):
        self.queue.put('w')
    # ... ...
```

而在使用过程中我又发现，点击叉叉退出程序时，程序总是崩溃，经过逐条分析，我发现 cv.VideoCapture 导致的虚拟摄像头未关闭以及线程进行中却被强制打断是异常退出的关键因素。因此我选择改写结束事件 closeEvent()，由于进程并不知道自己何时需要进行结束，因此我设立了 self.event 并在初始化里设置为 self.event.set()，在每个需要被结束的线程中的 while true 循环里最后加入对于该 event 状态的判断，若 event.is_set()为 False 则 break 结束线程。

同时改写 closeEvent()如下：

```
def closeEvent(self, event):
    self.event.clear()
    self.t1.join()
    self.t2.join()
    self.t3.join()
    self.soc.close()
    self.cap.release()

    event.accept()
    exit(0)
```

可见在结束例程中，我先使得 event 进行 clear，导致 event.is_set()为 False，再将每个多线程任务进行阻塞，最后将所用设备变量释放，完成关闭任务。

5.5 UI 设计与部分结构调整

考虑到作为现代电子系统的一部分，我们还需要一个具有科技感的 UI 界面。在廖崇骅同学的交流合作中，我们完成了扁平化的极具科技感的界面设计：

图 8 UI 界面最终显示图

以及在测试的过程中，我们发现由于前期调试过程中存在 WiFi 模块向外发送测试消息的环节，而不让这些信息泄流会导致指令的阻塞，因此我又增加了另一个线程专门进行了消息的接收，同时利用 queue 来发送消息，保证流畅性与稳定性。

6 实验结果

经过了近 10 天的努力，在我们五人的团队配合协作与来自助教的指导下，我们顺利完成了本次基于 PSoC6 的智能平衡车的综合创新实验，其实现的主要功能如下：

•实现了车体基本的平衡功能并能够在静态平衡时保证 5-8cm 内的车轮移动范围（考虑到本次直流编码电机本身存在的空程较大，这是非常好的结果）

- 实现了车体前进、后退、左转、右转、加速减速、停止的方向控制
- 实现了车体超声避障停止的功能
- 实现了车体与主机端通信，由主机端一体化控制的功能
- 在控制端允许用鼠标、键盘以及声控三种方式控制小车
- 实现视频从摄像头至控制界面的传输，并且拥有二维码扫描的功能

总而言之，我们不仅完成了平衡车基本平衡功能的实现，还给它加入了众多符合目前潮流的人工智能的功能，同时又进行了前后端的开发，打造出了一套由实物物体到主机控制平台的全流程现代电子系统。

由于本次实验结果已经由助教验收，并且上传了演示视频以及照片，在这里就不再重复贴图了。

7 问题集锦与实验总结反思

7.1 问题集锦

在本次综合创新实验的过程中，我通过硬件的架构设计、驱动编写、驱动与平衡调试以及用户控制程序的编写，记录了以下问题与最终解决方案。

Q1: 本该进入中断却未进入中断例程

A1：先利用 printf 在串口调试助手打印来确定是否是软件层面即代码的问题，若 pritf 也无法正常，说明必然是代码的问题，自代码开始向后进行检查，可能出现的问题有：中断初始化未进行或者是初始化函数本身存在错误，中断触发条件错误，有更高优先级的中断被触发，中断函数例程的最后没有完成_Clear 操作。

Q2: 本该输出在串口的全局变量数据错误或者是直接没有变化

A2：本问题可能是硬件层或软件层的原因，先对于软件代码层面进行检查，包括的点有：全局变量在不同.c 文件间没用使用 extern 进行定义，代码对应本身的逻辑存在错误，未执行这一段程序（例如 Q1 中的中断没有触发）；在确认软件代码层面没有出问题后可以继续检查硬件，首先是收发消息所使用的波特率是否统一（这对于串口调试助手有影响），其次是 GPIO 引脚是否适合进行数据的读取或者输出（例如针形引脚不适合进行数据的读取），再者就是模块供电是否

正常（WiFi 模块是否正常闪烁或是是否连接上）。

Q3：PWM 波占空比过小时电机不运转

A3：这是电机本身存在的问题，因此需要设立最小阈值来保证电机转动，否则小占空比 PWM 波无法驱动电机

Q4：Task 无法正常执行

A4：首先对于 Task 本身进行检查，在功能无误的同时要确保加上 taskYIELD 函数以确保对任务的调度，同时还需要给 task 预留出充足的空间，否则相当于该 task 会卡死。这同样需要对于总的空间容量进行调节。

Q5：仅仅加入或去除 printf 会导致整体性能的变化

A5：由于 PSoC 6 的任务调度机制，printf 相当于串口通信的一部分，自身会占用一定的资源，从而导致 PSoC 6 整体的响应频率变慢，这是正常现象。因此在 debug 结束后尽可能去除使用过的 printf。

Q6：新加入模块后之前工作正常的模块不工作了

A6：这同样需要考虑任务调度时分配的空间是否足够，同时需要注意两者是否存在共享变量的影响。

7.2 实验总结

不知不觉间，综合实验竟然得到了如此成功的完成。这对于那个 9.1 号在楼梯间里讨论初步计划的我而言，几乎是一件不可思议的事情。但是，回想起 6 月份我选课的初衷，不正是为了挑战自己、去做一些未曾做过的事情吗？

在这些天里，尽管每天从早干到晚，但收获与进步是显而易见的，从最开始时尚且对 PSoC 的开发流程都不太熟悉，到后续慢慢理解了提高实验指导书里的内容，以更为深刻的方式理解了中断，很多计原与模电的知识不再是书本或者 ppt 上干燥的表达，而是实验中切实存在且需要面对的问题。

在解决这些问题的过程中，我有了很多的收获。首先是面对在开发过程中不懂的地方或者不会写的代码，需要及时地通过寻找示例工程与往届同学们的参考资料进行解决，再者就是咨询助教，由于 PSoC 本身的特殊性，其在网上的资料可能不是特别全面，这时候就要充分利用身边的资源。

其次是硬件调试过程中积累的经验。与纯粹的编程不同，硬件调试难以进行逐步调试，且可能的问题点会比单纯软件代码多一个硬件端，我在出现部分问题的过程中总结了一些规律并写在了 7.1 节中。

再者就是对于团队性硬件开发的流程的掌握。我们得以顺利完成开发的还有一个因素正是我们组前期的分工相对明确，同时大家各自都对自己的工作非常认真负责，因此能够相对顺利的进行下来。同时，团队之间的互助也是非常重要的，本次实验的每一大环节几乎每个人都有或多或少的参与，这也直接提高了我们的效率以及质量。

7.3 实验反思

在本次实验的基础上，我还有以下想要实现，但受制于资源限制与时间关系未能实现的点：

- 在资源充足的情况下探索视频通过主板（未必是 PSoC）的传输方式
- 实现手势识别操控
- 实现手机端的控制
- 改进目前的静态 PID 算法，实现类似于代步平衡车那样的动态 PID

8 总结

无论是上学期的模电课程，还是这个小学期这两周里做的基础实验、提高实验以及综合创新实验，我都有了很大的收获以及技能的提升，尤其不会忘记综合实验这么一段虽然艰难但是愉悦的时光，在一教、李兆基楼与中央主楼之间来回奔波留下的汗水与笑颜，因此回过头来才发现写到头不知不觉已有一万字。这些调试 debug 的夜晚，如今看起来也并非枯燥难耐。作为一门实践性质的课程，现代电子系统实验真的非常具有包容性和自由发挥的空间！

最后，真心地感谢叶老师与两位助教的现场指导与答疑，解决了我的很多困惑，即便我们往往是实验室最后一批走的，助教们仍然耐心等待。与此同时，也非常感谢队友们：李晟永、郑新扬、佴瑞乾、廖崇骅的帮助与协作，希望能在未来能有更多的合作。