STM32 直流电机控制说明

2013 年 6 月 13 日

修改记录

制定日期	生效日期	制定 / 修订 内容摘要	页数	版本	拟稿	审查	批准
2013.06.14		STM32F207VG 直流电机控制说明	13	0.01	朱正晶		
2013.07.23		补充说明调试中遇到的问题	13	0.02	朱正晶		
2013.12.11		根据调试修改相关章节	13	0.1	朱正晶		
2013.12.25		更新 PID 算法占用 CPU 时间及电机停止曲线图	13	0.2	朱正晶		

目 录

1 简介

直流伺服电机控制方法。

1.1 手册目的

本手册的目的在于说明 STM32F207 控制直流电机的方法。

1.2 手册范围

本手册首先简要地介绍 PID 控制方法，然后将 PID 算法移植到 STM32F207VG 以及对占用的芯片资源进行说明，最后详细说明如何使用 PID 同时控制三直流电机的方法及 PID 参数调试方法。

本手册的使用者包括：

程序编写、维护者

2 PID 控制方法简要介绍

将偏差的比例（Proportion）、积分（Integral）和微分（Differential）通过线性组合构成控制量，用这一控制量对被控对象进行控制，这样的控制器称 PID 控制器。

注：本章只对 PID 进行简单介绍，深入理解 PID 可以参考其他教材。

2.1 模拟 PID 控制原理

在模拟控制系统中，控制器最常用的控制规律是 PID 控制。为了说明控制器的工作原理，先看一个例子。如图 2-1 所示是一个小功率直流电机的调速原理图。给定速度 $n_0(t)$与实际转速进行比较 $n(t)$，其差值 $e(t)=n_0(t)-n(t)$，经过 PID 控制器调整后输出电压控制信号 $u(t)$，$u(t)$经过功率放大后，驱动直流电动机改变其转速。

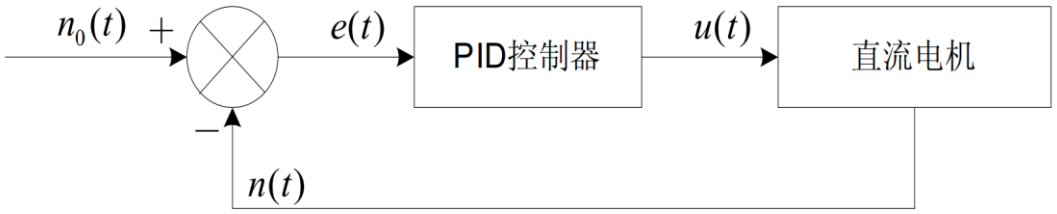


图 2-1 小功率直流电机调速系统

常规的模拟 PID 控制系统原理框图如图 2-2 所示。该系统由模拟 PID 控制器和被控对象组成。图中，$r(t)$是给定值，$y(t)$是系统的实际输出值，给定值与实际输出值构成控制偏差 $e(t)$，$e(t)$作为 PID 控制的输入，$u(t)$作为 PID 控制器的输出和被控对象的输入。

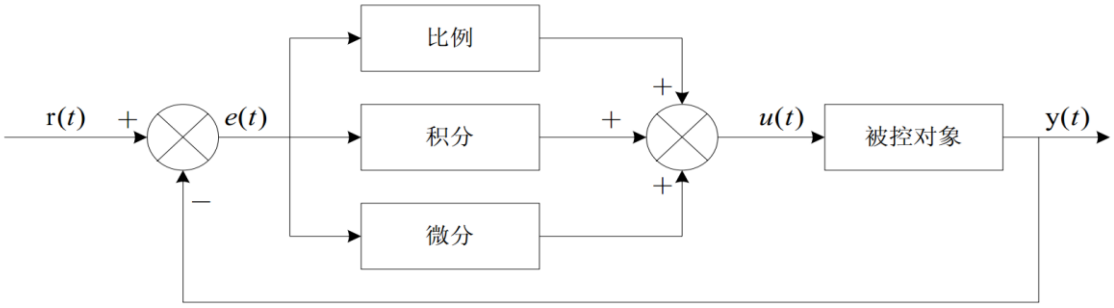


图 2-2 模拟 PID 控制系统原理图

2.2 数字 PID 控制

数字式 PID 控制算法可以分为位置式 PID 和增量式 PID 控制算法。本系统采用增量式 PID 控制算法。

由于计算机控制是一种采样控制，它只能根据采样时刻的偏差计算控制量，而不能像模拟控制那样连续输出控制量量，进行连续控制。由于这一特点积分项和微分项不能直接使用，必须进行离散化处理。

所谓增量式 PID 是指数字控制器的输出只是控制量的增量Δu_k。当执行机构需要的控制量是增

量，而不是位置量的绝对数值时，可以使用增量式 PID 控制算法进行控制。

增量式 PID 控制算法公式为：

$$\Delta u_k = u_k - u_{k-1} = Kp(e_k - e_{k-1} + \frac{T}{Ti}e_k + Td\frac{e_k - 2e_{k-1} + e_{k-2}}{T})$$

$$= Kp(1 + \frac{T}{Ti} + \frac{Td}{T})e_k - Kp(1 + \frac{2Td}{T})e_{k-1} + Kp\frac{Td}{T}e_{k-2})$$

$$= Ae_k + Be_{k-1} + Ce_{k-2}$$

其中 $A = Kp(1 + \frac{T}{Ti} + \frac{Td}{T})$；

$B = Kp(1 + \frac{2Td}{T})$；

$C = Kp\frac{Td}{T}$。

由上面的公式可以看出，如果计算机控制系统采用恒定的采样周期 T，一旦确定 A、B、C，只要使用前后三次测量的偏差值，就可以由上式求出控制量增量。

3 移植 PID 算法到 STM32F207VG 上

3.1 PID 算法流程图

网上很多 PID 实现算法，本系统采用了凌阳科技的 PID 算法（算法开源并且已在很多系统中使用）。此算法流程图如下（图 3-1）：

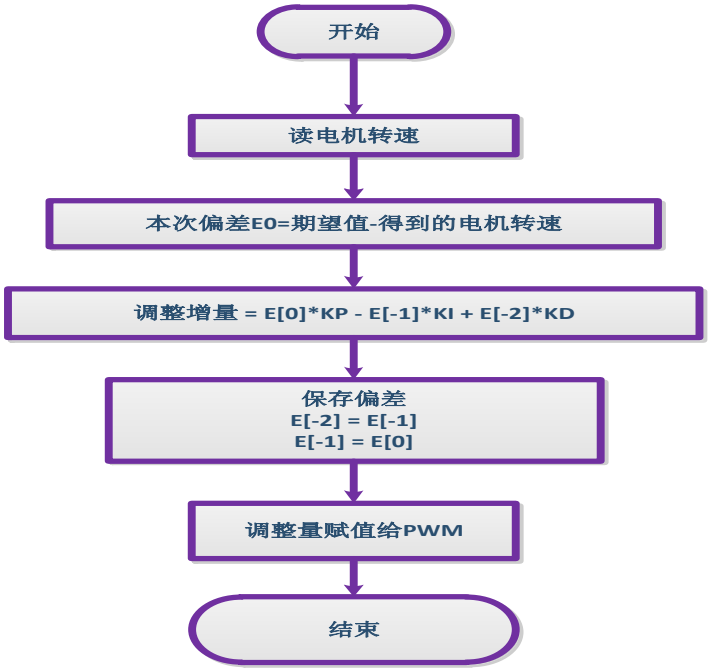


图 3-1 PID 算法流程图

3.2 实现 PID

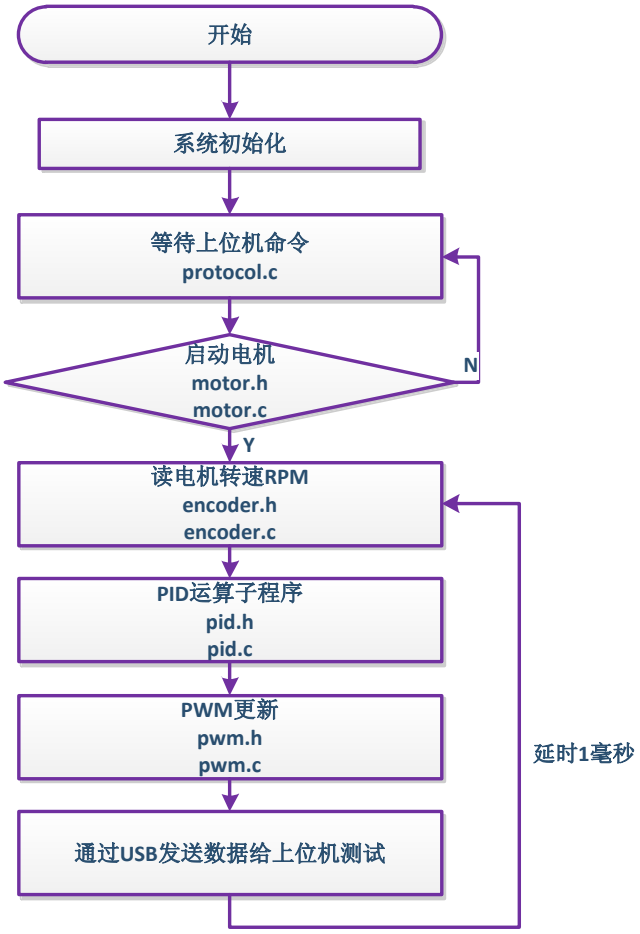
对比图 3-1 和第二章推导的公式，我们很容易理解图 3-2 所示代码：

```
00171: //============================================================
00172: // ----Function: IncPIDCalc()
00173: // ------Syntax: int IncPIDCalc(unsigned int NextPoint);
00174: // -Description: Increment Digital PID calculate
00175: // -------Notes: Basic Increment Digital PID
00176: // --parameters: Next Point
00177: // -----returns: increase controls parameter
00178: //============================================================
00179: double IncPIDCalc(PIDType index, int NextPoint)
00180: {
00181:     register double iError, iIncpid;
00182:
00183:     iError = sptr[index].SetPoint - NextPoint;
00184:
00185:     iIncpid = sptr[index].Proportion * iError                //E[0]
00186:             - sptr[index].Integral   * sptr[index].LastError  //E[-1]
00187:             + sptr[index].Derivative * sptr[index].PrevError; //E[-2]
00188:
00189:     sptr[index].PrevError = sptr[index].LastError;
00190:     sptr[index].LastError = iError;
00191:
00192:     return(iIncpid);
00193: }
00194:
```

图 3-2 增量式 PID 实现核心代码

3.3 STM32F207VG 使用 PID 控制直流电机流程

系统启动后初始化编码器、USB 模块、PWM 模块以及 PID 算法。流程图如下（图 3-2）：



电机启动后 1ms 进行一次 PID 调节（运算），1ms 的间隔通过定时器 6（TIM6）实现。注意：流程图中列出了每个模块对应源代码中实现的.c 文件和.h 文件。

3.4 占用芯片资源说明

首先，本系统进行了功能划分，分成如下模块(除了 protocol 在 USER 目录下，其他驱动都在 HAL 目录下)：

1）与上位机通信协议的实现： ***protocol.h protocol.c***
2）直流电机编码器接口： ***encoder.h encoder.c***
3）直流电机控制接口： ***motor.h motor.c***
4）pwm 模块： ***pwm.h pwm.c***
5）timer 模块： ***timer.h timer.c***

为了实现这些功能使用了如下芯片资源：

定时器：TIM1-TIM4、TIM9，USB_FS 模块，CAN 模块，USART 模块，四个 GPIO 输出高电平。

使用的引脚资源如表 3-1：

Port	TIM1/2	TIM3/4/5	TIM8/9/10/11	USART1/2/3	CAN1/CAN2	OTG_FS/OTG_HS
PA0	TIM2_CH1					
PA1	TIM2_CH2					
PA2			TIM9_CH1			
PA3			TIM9_CH2			
PA6		TIM3_CH1				
PA7		TIM3_CH2				
PA10						OTG_FS_ID
PA11						OTG_FS_DM
PA12						OTG_FS_DP
PB6				USART1_TX		
PB7				USART1_RX		
PC6			TIM8_CH1			
PC7			TIM8_CH2			
PD0					CAN1_RX	
PD1					CAN1_TX	
PD12		TIM4_CH1				
PD13		TIM4_CH2				
PE9	TIM1_CH1					
PE11	TIM1_CH2					
PE13	TIM1_CH3					
PE14	TIM1_CH4					

表 3-1 使用引脚

说明：此配置为开发板中引脚配置，在 Trima 项目中引脚有变动。请参考具体的原理图。

4 控制方法说明及调试方法

本章详细说明如何使用 PID 同时控制三直流电机的方法及 PID 参数调试方法。

4.1 系统框架图

图 4-1 所示为系统框图，可以看出每个直流电机的编码器使用了一个 TIM。注意：编码器必须和开发板共地。从 STM32F2XX Reference manual.pdf 手册中可以查到只有 TIM1-TIM5 及 TIM8 带编码器接口。因为 TIM2 和 TIM5 引脚有复用的问题，在这里我们只能使用 TIM2 或者 TIM5。本系统选用如下编码器：TIM2、TIM3、TIM4。TIM1 生成四路不同的 PWM，只能控制电机正转或者反转。TIM9 包含两个 PWM Channel 可以控制一路正反转电机。TIM6 定义为 1 毫秒一次的中断。

PWM 通过 L298N 模块驱动电机，一路 L298N 可以同时驱动两路直流电机。本系统使用两路 L298N。

注：在 Triam 项目中使用 A4950 芯片，软件控制上都一样，但是两者的效率及其他数据差别较大，具体请参考芯片手册。

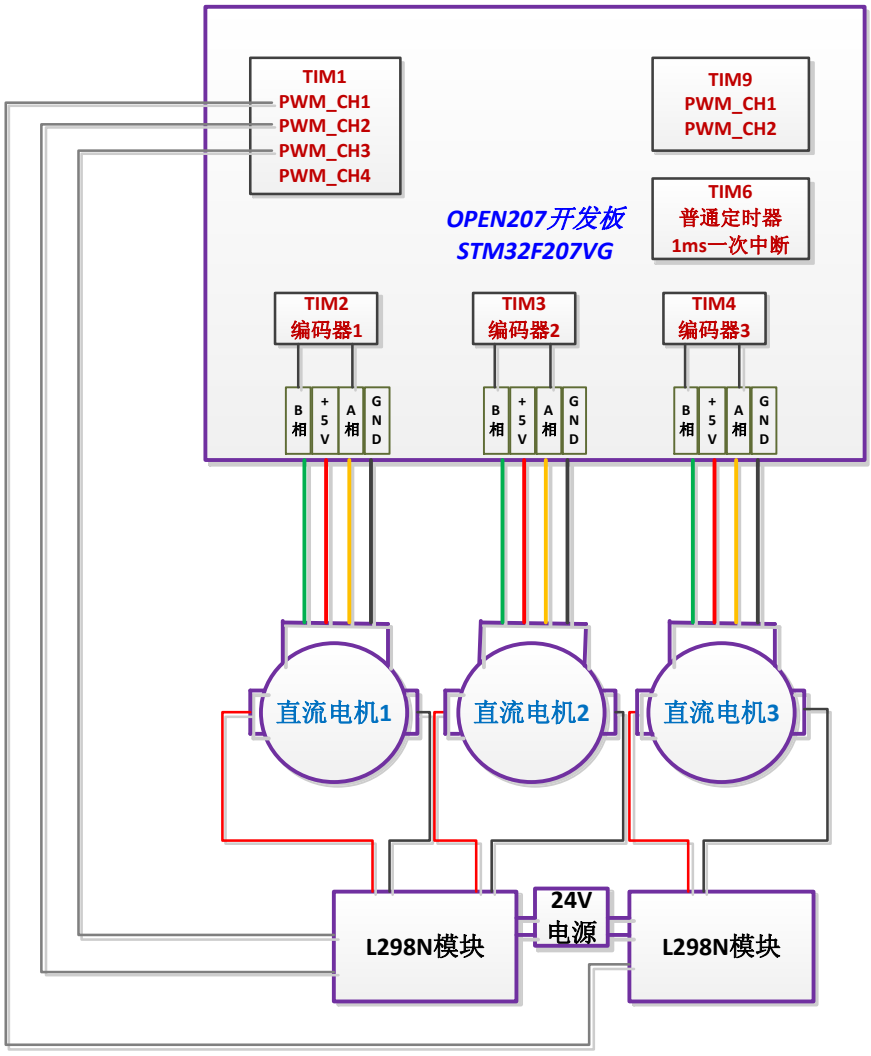


图 4-1 系统框图

4.2 系统核心部分

分为五部分：①读取速度；②计算 PID 增量；③PWM 限幅；④调整 PWM；⑤保存调试数据。

```
00179: static void motor_pid(MotorType motor)
00180: {
00181:     data_type data;
00182:     extern cir_queue queue;
00183:     s32 cur_speed;
00184:
00185:     if (check_motor_num(motor) != 0)
00186:         return;
00187: ① 读取速度
00188:     cur_speed = get_speed((EncoderType)motor);
00189:  ② 计算PID增量
00190:     current_motor_pwm[motor] += IncPIDCalc((PIDType)motor, cur_speed);
00191: ③ PWM限幅
00192:     if(current_motor_pwm[motor] > get_pwm1_max_value())
00193:         current_motor_pwm[motor] = get_pwm1_max_value();
00194:     if(current_motor_pwm[motor] < get_pwm1_min_value())
00195:         current_motor_pwm[motor] = get_pwm1_min_value();
00196:  ④ 调整PWM
00197:     pwm1_change((PwmType)motor, current_motor_pwm[motor], 1);
00198:  ⑤ 将速度和PWM存到队列里，通过USB发送给上位机调试
00199:     /* save data */
00200:     data.index = (u8)motor;
00201:     data.x = (s16)cur_speed;
00202:     data.y = (s16)current_motor_pwm[motor];
00203:     en_queue(&queue, &data);
00204:
00205: } ? end motor_pid ?
00206:
00207:
```

图 4-2 系统核心部分（1 毫秒调用一次）

4.3 系统 PID 运算时间

本系统使用 TIM6 做 1 毫秒一次中断，在这个中断中进行速度读取、PID 运算及速度控制。中断优先级设为最高。控制一路直流电机时运算时间为 11.6 微秒左右，同时控制三路运算时间为 34.2 微秒。优化级别为 O0（没任何优化）。当优化级别为 O3 时，控制一路直流电机的运算时间为 8.6 微秒，三路运算时间为 24.8 微秒。如下表 4-1 所示：

编译器优化级别	一路电机 PID 运算时间(微秒)	三路电机 PID 运算时间(微秒)
O0（不优化）	11.6	34.2
O3（最高等级优化）	8.6	24.8

表 4-1

现在使用的是最耗 CPU 时间的浮点运算（STM32F207 不带浮点运算单元 FPU，但是它的乘法器进行一次乘法运算只需要一个 CPU 周期，相比 STM32F1xx 系列还是快很多），以后有需求可以进行调整，比如使用 Q 格式来进行 PID 运算或者直接使用带 FPU 的 MCU（比如 STM32F407），这样速度还能再提高。

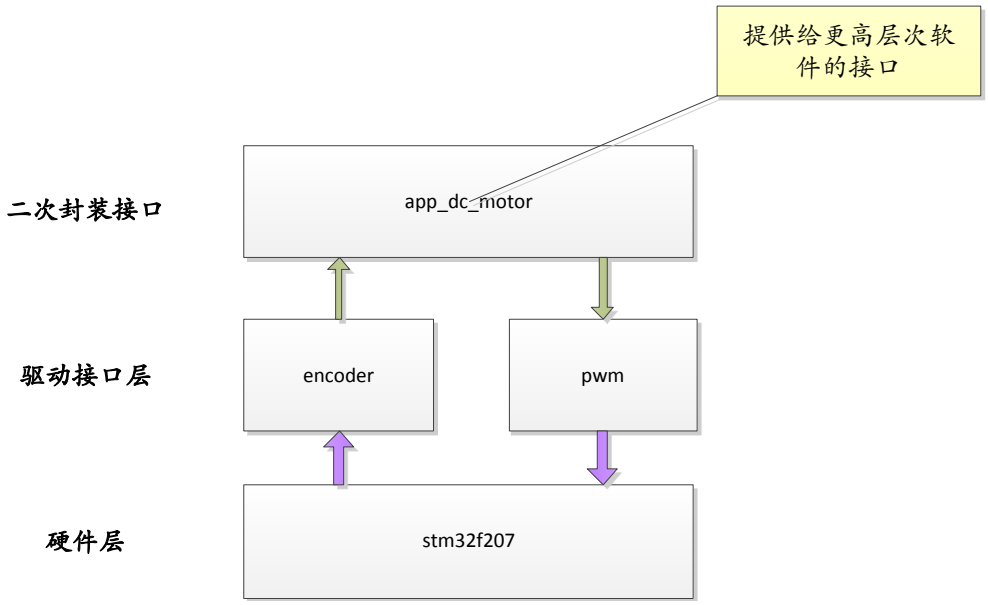
2013.12.25 更新。由于算法上作了一些调整，并且通过定时器计数来精确地确定算法所费时间，有了以下记录：（注意：这里的运算时间仅仅指 PID 算法所占用 CPU 资源，不包括中断函数切换，任务 task 寄存器保存等操作）

编译器优化级别	定时器计数（60MHz）	单环：位置 PID 运算时间(微秒)
O0（不优化）	830	13.83

4.4 软件移植性配置

为了适应不同的项目，直流电机控制程序在可移植性功能上作了一些工作，现有代码经过简单的修改就能方便的移植到不同的工程中。下面对这些配置项进行说明。

直流电机使用了编码器模块、PWM 控制模块，这两部分分别读取速度和输出 PWM 信号，在这两层的基础上封装了一个对外的接口模块。下图为软件层次架构。



对于不同的板（项目）差别只在于一些引脚的配置，因此在软件中留出接口来适应不同的项目。

4.4.1 编码器接口

在源文件 encoder.c 中通过不同的宏定义来区分不同的工程，在这些宏定义下定义具体的引脚配置。

下图为 encoder.c 引脚配置一部分截图，具体的请参考源代码。

```
00029:
00030:
00031: #if defined(USE_OPEN_207Z_BOARD) || defined(USE_ARM1_REV_0_1) || defined(USE_ARM2_REV_0_1)
00032:
00033: // --------------- Encoder 1 ---------------
00034: #define ENCODER_TIM1                    TIM2
00035: #define ENCODER_TIM1_IRQ_NUM            TIM2_IRQn
00036: #define ENCODER_TIM1_CLK                RCC_APB1Periph_TIM2
00037: // encoder 1 GPIO Pin configuration
00038: // PA0 PA1
00039: #define ENCODER_TIM1_GPIO_PORT          GPIOA
00040: #define ENCODER_TIM1_PH_A_PIN           GPIO_Pin_0
00041: #define ENCODER_TIM1_PH_A_PIN_SOURCE    GPIO_PinSource0
00042: #define ENCODER_TIM1_PH_A_GPIO_CLK      RCC_AHB1Periph_GPIOA
00043: #define ENCODER_TIM1_PH_A_AF            GPIO_AF_TIM2
00044: #define ENCODER_TIM1_PH_A_CHANNEL       TIM_Channel_1
00045:
00046: #define ENCODER_TIM1_PH_B_PIN           GPIO_Pin_1
00047: #define ENCODER_TIM1_PH_B_PIN_SOURCE    GPIO_PinSource1
00048: #define ENCODER_TIM1_PH_B_GPIO_CLK      RCC_AHB1Periph_GPIOA
00049: #define ENCODER_TIM1_PH_B_AF            GPIO_AF_TIM2
00050: #define ENCODER_TIM1_PH_B_CHANNEL       TIM_Channel_2
00051:
00052:
00053: // --------------- Encoder 2 ---------------
00054: #define ENCODER_TIM2                    TIM3
00055: #define ENCODER_TIM2_IRQ_NUM            TIM3_IRQn
00056: #define ENCODER_TIM2_CLK                RCC_APB1Periph_TIM3
00057:
00058: #if defined(USE_OPEN_207Z_BOARD)
00059: // encoder 2 GPIO Pin configuration
00060: // PC6 PC7
00061: #define ENCODER_TIM2_GPIO_PORT          GPIOC
00062: #define ENCODER_TIM2_PH_A_PIN           GPIO_Pin_6
00063: #define ENCODER_TIM2_PH_A_PIN_SOURCE    GPIO_PinSource6
```

另外，由于不同的直流电机编码器分辨率不一样，在这里也提供了相应的宏定义，根据具体的码盘分辨率修改一下就行。在头文件 encoder.h 中修改下面的参数即可。

```
// number of pulses per revolution
#define ENCODER_PPR                    (u16)(512)
```

4.4.2 PWM 接口

pwm 接口和 encoder 的实现可移植的方法差不多，在 pwm.h 和 pwm.c 中修改硬件对应的 PIN 脚。需要注意的是 TIM1, TIM8 和其他 TIM 的接口不一样。

4.5 系统调试

由于调试的数据量很大，直接使用 MCU 显示信息已不能满足要求，考虑将调试数据发到 PC 上，利用 PC 的强大处理能力来处理这些数据。我们先来计算一下数据传输速度：如果一路电机工作，一个典型数据包的大小为 13 字节（*R1 2000 3000;*），则一秒钟的数据量为：

D = 1000*1*13 = 13000Bytes（13KB/s）

如果三路电机同时工作，则一秒钟的数据量为：

D = 1000*3*13 = 39000Bytes（38KB/s）

串口最大通信速度可达 128000bps（15KB/s），再高通信容易出错。如果只调试一路电机串口能满足需求，但是如果三路同时工作，38KB/s 的速度已经超过了串口最大发送速度，可以考虑通过三个串口来发送，但是这样和 PC 的连线就很多，不方便。

当然，我们可以使用二进制形式发送调试数据，这样每个包的大小就小一点。但这样又带来另外一个问题，我们需要定义一个比较完善的通信协议，在数据通信出错的情况下能自我恢复。再加

上帧头和帧尾最后的数据包大小不一定能比直接以字符发送小多少。我们这个是用来调试的，而且错一个数据对我们的调试影响不大。没必要再实现一个复杂的通信协议。而且，以字符发送的话程序处理比较简单。

最后发现 STM32F207 的芯片支持 USB2.0，ST 官网也提供了很完善的 USB 驱动库。我们直接下载下来根据开发板稍微修改一下就能使用了。本系统使用 VCP，也就是 USB 虚拟串口，这样我们的上位机也不需要修改，还是使用之前的串口驱动。VCP 的通信速度最高可达 4.3Mbps（524KB/s），这个速度完全满足我们的需求了。当然 USB 最高可达 12Mbps，实现这个速度需要我们自己写 Windows 下的 USB 驱动，以后有需求再研究。上面的 USB 都是工作在全速模式下，最高速度 12Mbps，STM32F207VG 支持 USB2.0 高速模式（480Mbps），这个模式需要外接一颗 USB PHY 芯片。

图 4-5 说明了上下位通过 USB VCP 连接示意图：

图 4-5 上下位 USB 通信

上下位通信协议：

发送命令格式：命令以空格分隔，以;结束

1）设置 PID 参数

S　　设置 PID 参数

10　　KP 参数，实际为 10/1000 = 0.01

1150　　KPP 参数，实际运算为 1.15*KP

110　　KPI 参数，实际运算为 0.11*KP

0　　KPD 参数，实际运算为 0*KP

2000　　电机转速，单位为 RPM（每分钟转速）

命令示例：***S1 10 1150 110 0 2000;***　　设置电机 1 的 PID 参数

2）启动电机

B　　启动电机

命令示例：***B1;***　　启动电机 1

启动后开始上报数据包给上位机。

上报格式

R　　上报数据帧帧头

1500　　电机当前的转速

36　　当前驱动电机的 PWM 值（PWM 范围和 PWM 频率有关）

命令示例：***R1 1500 36;***　　电机 1 上传数据包

3）停止电机

T1;　　关闭电机 1

电机停止后停止上报数据包。

4）设置电机转速（可以动态设置）

P　　设置电机转速

2000　　电机的转速

命令示例：***P1 2000;***　　设置电机 1 转速为 2000

上位机调试界面如图 4-6 所示，采用 VC++6.0 编程。串口驱动采用久经考验的 Moxa 公司的 PComm 控件。使用网上其他的串口类都会出现一些问题。这个控件使用也非常简单，几个函数调用

即可完成通信。而且它还提供了非常完善的说明文档。建议以后的上位机开发如果用到串口都使用这个控件进行开发。

上位机的控制界面很简单，看看就能了解，这里不作介绍。

上位机实现也很简单，开了一个线程处理串口收到的数据，按钮其实就通过串口发送一些命令。此外，还用了一个网上开源的 XGraph 类，用来显示收到的数据。有兴趣的可以阅读源码。这里不作介绍。

图 4-6 上位机界面实现

调试时，当电机开始运转后，电机转速和 PWM 调节值会在界面中实时更新，方便观察电机的当前状态。如图 4-7 所示。

图 4-7 实时信息显示界面

点击“采集数据 1”按钮，采集完成后点击“停止采集 1”按钮，会弹出一个非模态对话框，如图 4-8。对话框的标题栏为当前设置的 PID 参数。编号 01 颜色为紫色的那条曲线为电机转速曲线，编号 02 颜色为红色的那条曲线为 PWM 调节曲线。编号 03 颜色为绿色的为设定的速度，一般为一条直线。由曲线可知当前 PID 参数的调节效果。

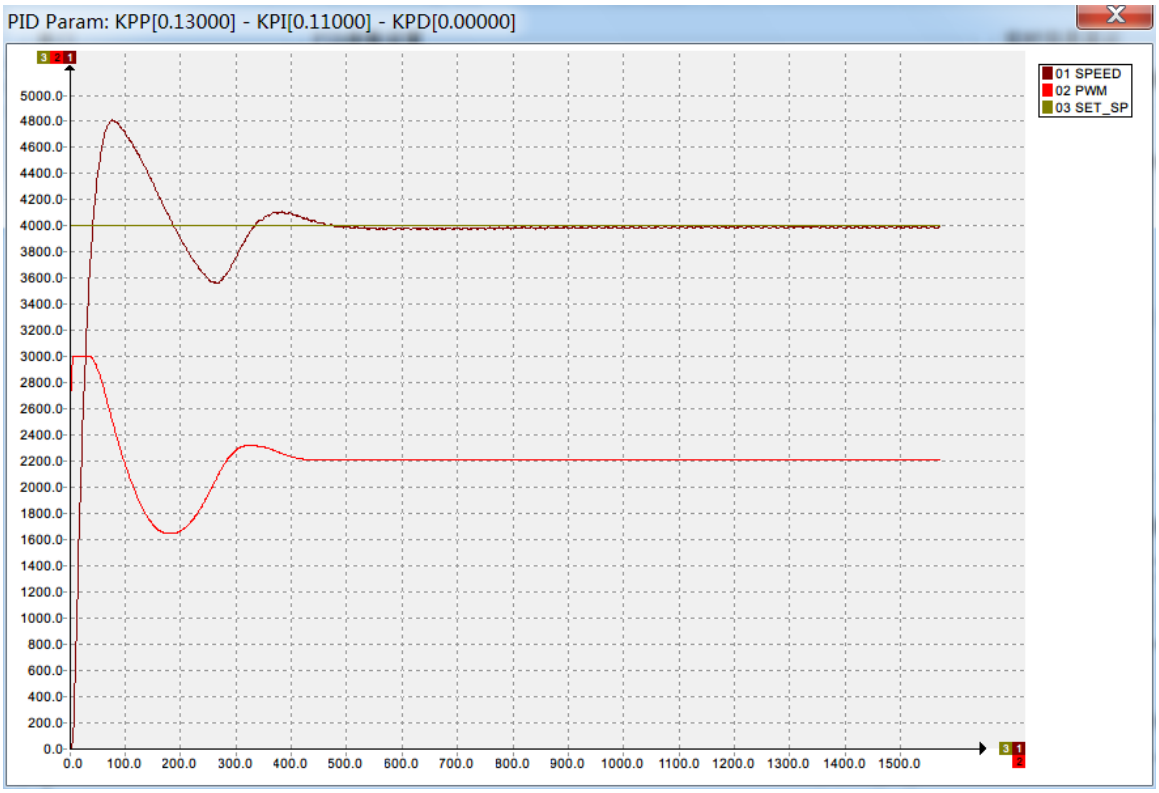


图 4-8 PID 调节曲线显示

4.6 当前调试遇到的问题及相关说明

4.6.1 电机 PID 参数

目前使用的电机是二手直流电机，调节的 PID 参数可能不适用于项目中使用的电机，因此等项目中电机确定需要重新调试 PID 参数。

4.6.2 发热现象

目前发现使用 L298N 驱动芯片驱动直流电机时会有明显的发热现象，二手直流电机功率比较小，驱动芯片发热应该不明显，这个问题有待后面继续调试及确认。目前使用的 PWM 频率为 20K。Trima 项目中使用的 A4950 芯片发热量不明显。

4.6.3 电机停止稳定性

在实际应用中，停止的稳定性相当重要，因此这里测试了当前 PID 参数下，电机停止的曲线图。图 4-9 为整个运动过程中电机的调节曲线，4-10 为停止部分放大图。

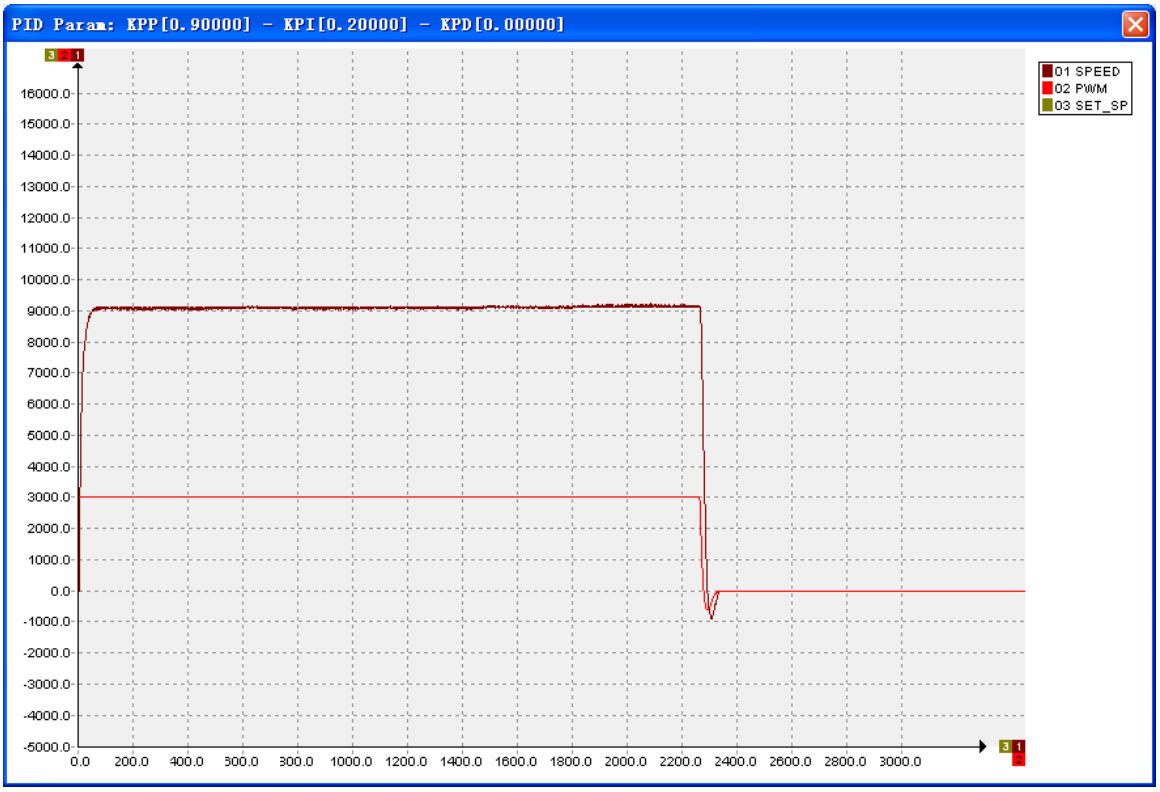


图 4-9 直流电机整个运动过程

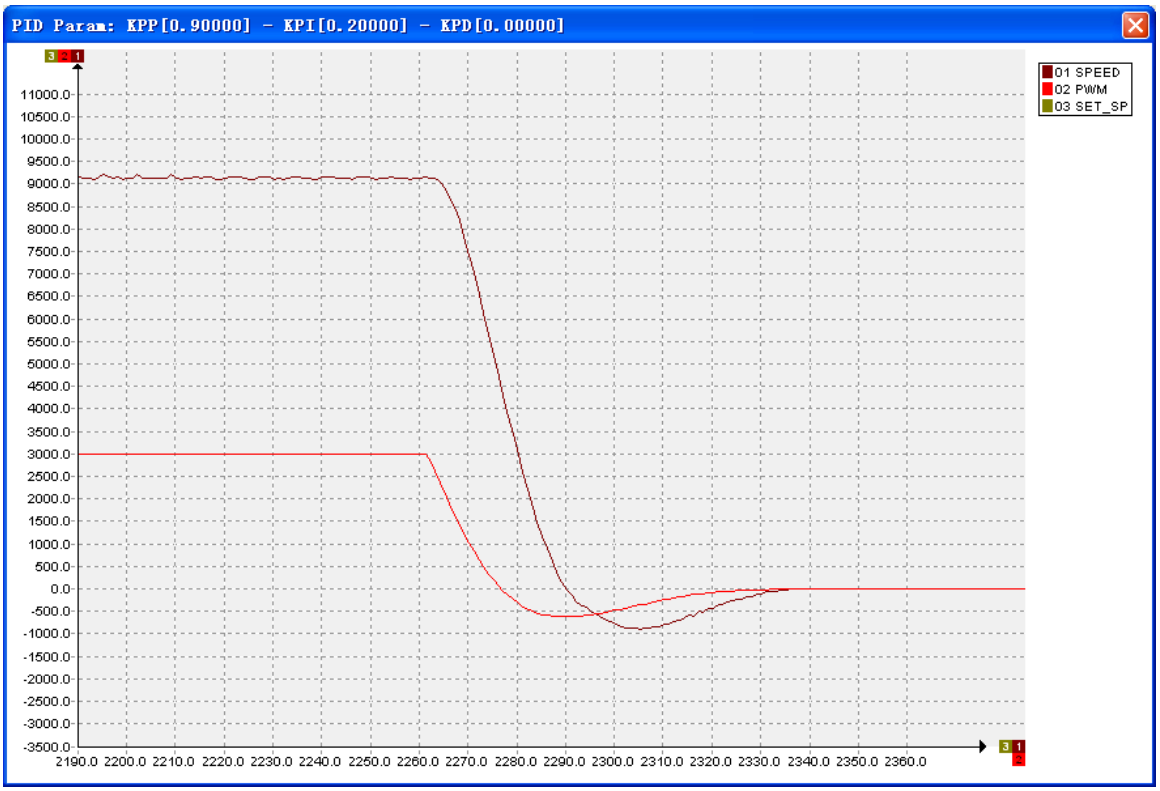


图 4-10 停止过程放大图