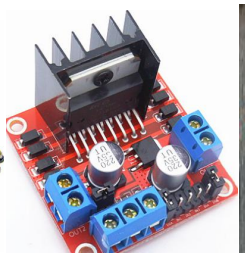
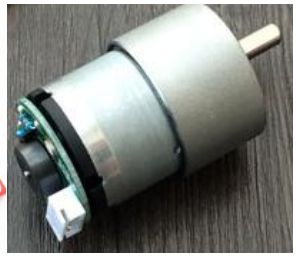
直流电机 PID 调速试验

姓名：岑佳楠　　学号：2016330300132　班级：16 机电二班

研究直流电动机的 PID 调速过程,观察在 PID 调速程序下，电机到达指定速度的运动过程。

1.硬件部分：

硬件选择：STM32F407ZGT,L298N，带编码器的直流减速电机，若干杜邦线和电源。

1.1 L298N 模块说明：

IN1-IN4 逻辑输入：其中 IN1、IN2 控制电机 M1；IN3、IN4 控制电机 M2。例如 IN1 输入高电平 1，IN2 输入低电平 0，对应电机 M1 正转；IN1 输入低电平 0，IN2 输入高电平 1，对应电机 M1 反转，EnA，EnB 接控制使能端，控制电机的停转,调速就是改变使能端 PWM 波的占空比。

1.2 编码器模块说明：

这是一款增量式输出的霍尔编码器。编码器有 AB 相输出,所以不仅可以测速,还可以辨别转向。我们只需给编码器电源 5V 供电,在电机转动的时候即可通过 AB 相输出方波信号。编码器自带了上拉电阻,所以无需外部上拉,可以直接连接到单片机 IO 读取。因为编码器输出的是标准方波，所以我们可以使用单片机直接读取。软件处理的方法分为两种，自带编码器接口的单片机如 STM32，可以直接使用硬件计数。

2. 软件部分：

PID 一般有两种：位置式 PID 和增量式 PID。我认为在平衡小车里一般用增量式，因为位置式 PID 的输出与过去的所有状态有关，计算时要对 e（每一次的控制误差）进行累加，这个计算量非常大，没有必要。而且小车的 PID 控制器的输出并不是绝对数值，而是一个△，代表增多少，减多少。换句话说，通过增量 PID 算法，每次输出是 PWM 要增加多少或者减小多少，而不是 PWM 的实际值。

下面以增量式 PID 说明。

这里再说一下 P、I、D 三个参数的作用。P=Proportion，比例的意思，I 是 Integral，积分，D 是 Differential 微分。

打个比方，如果现在的输出是 1，目标输出是 100，那么 P 的作用是以最快的速度达到 100，把 P 理解为一个系数即可；而 I 呢？大家学过高数的，0 的积分才能是一个常数，I 就是使误差为 0 而起调和作用；D 呢？大家都知道微分是求导数，导数代表切线是吧，切线的方向就是最快到至高点的方向。这样理解，最快获得最优解，那么微分就是加快调节过程的作用了。

公式如下图所示：

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

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

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

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

$B = Kp(1 + \frac{2Td}{T})$；

$C = Kp\frac{Td}{T}$。

最后结果为：

△Uk=A*e(k)+B*e(k-1)+C*e(k-2)

这里 KP 是 P 的值，TD 是 D 的值，1/Ti 是 I 的值，都是常数，哦，还有一个 T，T 是采样周期，也是已知。而 A B C 是由 P I D 换算来的，按这个公式，就可

以简化计算量了，因为 P I D 是常数，那么 A B C 可以用一个宏表示。这样看来，只需要求 e(k) e(k-1) e(k-2)就可以知道△Uk 的值了，按照△Uk 来调节 PWM 的大小就克可以了。采样周期也是有据可依的，不能太大，也不能太小。

在了解 PID 的大致原理和应用方法后，我们就可以把它应用到电机调速上来了。但 PID 实际编程的过程的，要注意的东西还是有几点的。PID 这东西可以做得很深。

1 .PID 的诊定。凑试法，临界比例法，经验法。

2. T 的确定，采样周期应远小于过程的扰动信号的周期，在小车程序中一般是 ms 级别。

3. 目标速度何时赋值问题，如何更新新的目标速度?这个问题一般的人都乎略了。目标速度肯定不是个恒定的，那么何时改变目标速度呢？

4 .改变了目标速度，那么 e(k) e(k-1) e(k-2)怎么改变呢？是赋 0 还是要怎么变？

5 .是不是 PID 要一直开着？

6 .error 为多少时就可以当速度已达到目标？

7. PID 的优先级怎么处理，如果和图像采集有冲突怎么办？

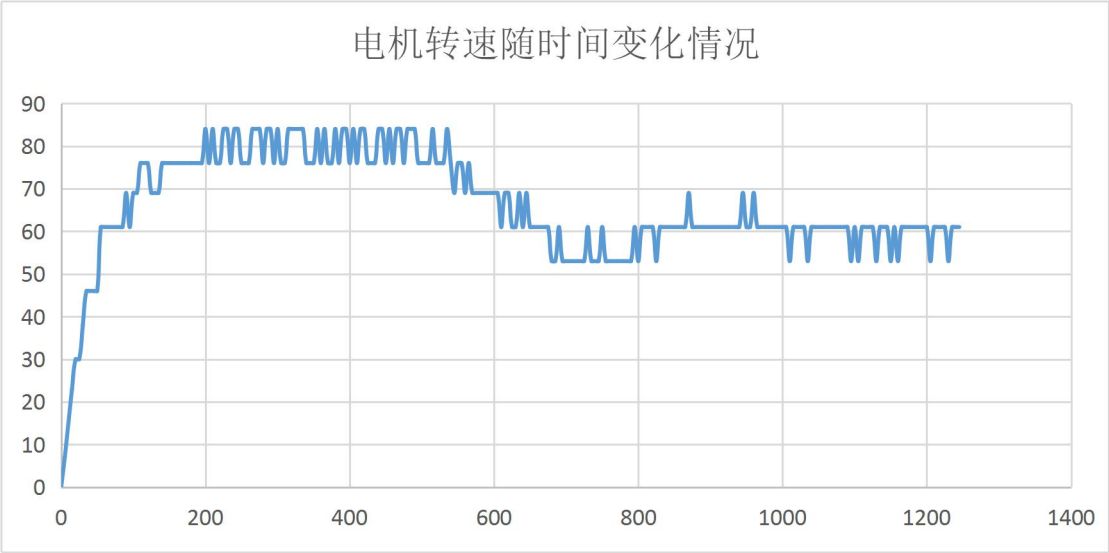
8 .PID 的输入是速度，输出是 PWM，按理说 PWM 产生速度，但二者不是同一个东西，有没有问题？

9. PID 计算如何优化其速度？

所以我认为这趟实验课我们学的东西还是远远不够的，需要我们在课后深入了解 PID 的原理和应用，讲理论和实践结合到一起，才能真正掌握该门对我们今后工作大有帮助的知识。

3. 实验结果：

在程序编译通过后，通过对串口软件中的输出的电机转速数据进行分析，绘制成如下表所示的曲线图。电机设定转速为 60r/min，在经过大约 600ms 时间后，电机速度稳定 60r/min 一定范围内，存在一定波动，但在误差可允许范围内，大多时间比较稳定，若想较少误差，同时减短电机到达指定速度的时间，则需要调整 PID 参数。具体程序代码见附录。



4. 附录：

Mian.c:

```
#include "sys.h"
#include "delay.h"
#include "usart.h"
#include "led.h"
#include "encoder.h"
#include "pwm.h"
#include "pid.h"
signed long hValue;
extern signed long Speed;
```

```
u16 mpwm=0;
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置系统中断优先级分组 2
    delay_init(168);  //初始化延时函数
    uart_init(115200);//初始化串口波特率为 115200
    Encoder_init();
    PID_Init(60);
  TIM4_Int_Init(50-1,8400-1);    //84M/8400=10000hz,50 次计数时间为 5ms
    TIM14_PWM_Init(10000-1,84-1);    //84M/84=1Mhz 的计数频率,重装载值 10000,所以
PWM 频率为 1M/10000=100hz.
    while(1)
    {
        mpwm+=IncPIDCalc(Speed);     //增量式 PID
//    mpwm=LocPIDCalc(Speed);        //位置式 PID
        TIM_SetCompare1(TIM14,mpwm);     //修改比较值，修改占空比
//      printf("%ld\r\n",Speed);
//      printf("%d\r\n",mpwm);
        printf("S=+%ld\r\n",Speed);
        printf("mpwm=+%d\r\n",mpwm);
        delay_ms(10);
    }

}
```

Pid.c:

```
#include "pid.h"
#include "led.h"
#include "usart.h"
void PID_Init(u16 setpoint)
```

```
{
    sptr->LastError=0;
    sptr->PrevError=0;
    sptr->SumError=0;
    sptr->P=P_DATA;
    sptr->I=I_DATA;
    sptr->D=D_DATA;
    sptr->SetPoint=setpoint;
}
//增量式 PID
int IncPIDCalc(int NextPoint)
{
    int iError,iIncpid;
    iError=sptr->SetPoint-NextPoint;
    iIncpid=sptr->P*iError-sptr->I*sptr->LastError+sptr->D*sptr->PrevError;
    sptr->PrevError=sptr->LastError;
    sptr->LastError=iError;
    return(iIncpid);
}
//位置式 PID
unsigned int LocPIDCalc(int NextPoint)
{
    int iError,dError,iLocpid;
    iError=sptr->SetPoint-NextPoint;
    sptr->SumError+=iError;
    dError=iError-sptr->LastError;
    iLocpid=sptr->P*iError+sptr->I*sptr->SumError+sptr->D*dError;
    sptr->LastError=iError;
    return(iLocpid);
}
```

Encoder.c:

```
#include "encoder.h"
#include "led.h"
#include "usart.h"

#define ENCODER_PPR (u16)(390) // number of pulses per revolution
#define ENCODER_TIMER TIM3 // Encoder unit connected to TIM3

u16 hEncoder_Timer_Overflow;
u16 n=0;
u16 hEnc_Timer_Overflow_sample_one=0;
u16 hCurrent_angle_sample_one=0;
u16 hPrevious=0;
s32 Pulses;
u16 hPrevious_angle;
signed long Speed;

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

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC,ENABLE);
```

```
  /* Configure PC.06,07 as encoder input */
  GPIO_PinAFConfig(GPIOC,GPIO_PinSource6,GPIO_AF_TIM3);
 GPIO_PinAFConfig(GPIOC,GPIO_PinSource7,GPIO_AF_TIM3);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
 GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;//上拉
 GPIO_Init(GPIOC, &GPIO_InitStructure);//初始化

    /* Enable the TIM3 Update Interrupt */
 NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
 NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
 NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
 NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
 NVIC_Init(&NVIC_InitStructure);

/* Timer configuration in Encoder mode */
  TIM_DeInit(TIM3);
  TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);

 TIM_TimeBaseStructure.TIM_Prescaler = 0x0; // No prescaling
 TIM_TimeBaseStructure.TIM_Period = (4*ENCODER_PPR)-1;
 TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
 TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
 TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

 TIM_EncoderInterfaceConfig(TIM3,                TIM_EncoderMode_TI12,
```

```
TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);
        TIM_ICStructInit(&TIM_ICInitStructure);
       TIM_ICInitStructure.TIM_ICFilter =6;
       TIM_ICInit(TIM3, &TIM_ICInitStructure);
        // Clear all pending interrupts
       TIM_ClearFlag(TIM3, TIM_FLAG_Update);
       TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);

     //Reset counter
        TIM_SetCounter(TIM3,0);

       TIM_Cmd(TIM3, ENABLE);



}

void TIM3_IRQHandler(void)
{
     /* Clear the interrupt pending flag */
     TIM_ClearFlag(TIM3, TIM_FLAG_Update);
     if (hEncoder_Timer_Overflow != 65535)
     {
          hEncoder_Timer_Overflow++;
     }
}

void TIM4_Int_Init(u16 arr,u16 psc)
{
     TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
     NVIC_InitTypeDef NVIC_InitStructure;
```

```
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4,ENABLE);

   TIM_TimeBaseInitStructure.TIM_Period = arr;
    TIM_TimeBaseInitStructure.TIM_Prescaler=psc;
    TIM_TimeBaseInitStructure.TIM_CounterMode=TIM_CounterMode_Up;
    TIM_TimeBaseInitStructure.TIM_ClockDivision=TIM_CKD_DIV1;

    TIM_TimeBaseInit(TIM4,&TIM_TimeBaseInitStructure);

    TIM_ITConfig(TIM4,TIM_IT_Update,ENABLE);
    TIM_Cmd(TIM4,ENABLE);

    NVIC_InitStructure.NVIC_IRQChannel=TIM4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0x01;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority=0x03;
    NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
    NVIC_Init(&NVIC_InitStructure);

}
void TIM4_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM4,TIM_IT_Update)==SET)
    {
            hEnc_Timer_Overflow_sample_one = hEncoder_Timer_Overflow;

              hCurrent_angle_sample_one = TIM3->CNT;

               if ( (TIM3->CR1 & TIM_CounterMode_Down) ==
```

```
                TIM_CounterMode_Down)//向下计数
                {
                    Pulses = (s32)(hCurrent_angle_sample_one - hPrevious -
                     (hEnc_Timer_Overflow_sample_one) * (4*ENCODER_PPR));
                }
                else //向上计数
                {
                    Pulses = (s32)(hCurrent_angle_sample_one - hPrevious +
                     (hEnc_Timer_Overflow_sample_one) * (4*ENCODER_PPR));
                }
                        Speed = (signed long long)(Pulses * 200*60);
                        Speed /= (4*ENCODER_PPR);
        hPrevious=hCurrent_angle_sample_one;
        hEncoder_Timer_Overflow=0;
    }
    TIM_ClearITPendingBit(TIM4,TIM_IT_Update);
}
```