# 小时候挺菜





## PID控制温度

总所周知，PID算法是个很经典的东西。而做自平衡小车，飞行器PID是一个必须翻过的坎。因此本节我们来好好讲解一下PID，根据我在学习中的体会，力求通俗易懂。并举出PID的形象例子来帮助理解PID。

一、首先介绍一下PID名字的由来：

P：Proportion（比例），就是输入偏差乘以一个常数。

I ：Integral（积分），就是对输入偏差进行积分运算。

D：Derivative（微分），对输入偏差进行微分运算。

注：输入偏差=读出的被控制对象的值-设定值。比如说我要把温度控制在26度，但是现在我从温度传感器上读出温度为28度。则这个26度就是”设定值“，28度就是“读出的被控制对象的值”。然后来看一下，这三个元素对PID算法的作用，了解一下即可，不懂不用勉强。

P，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。

二、然后要知道PID算法具体分两种：一种是位置式的，一种是增量式的。在小车里一般用增

**最新评论**

1. Re:multiple definition of 问题解决方法

很有用

--一秒钟的感动zZ

2. Re:Python中单引号，双引号，3个单引号及3个双引号的区别

谢谢~

--がせい

量式，为什么呢？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，而明显没有必要。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。所以明白增量式PID就行了。

三、接着讲PID参数的整定，也就是PID公式中，那几个常数系数Kp，Ti，Td等是怎么被确定下来然后带入PID算法中的。如果要运用PID，则PID参数是必须由自己调出来适合自己的项目的。通常四旋翼，自平衡车的参数都是由自己一个调节出来的，这是一个繁琐的过程。本次我们可以不管，关于PID参数怎么确定的，网上有很多经验可以借鉴。比如那个经典的经验试凑口诀：

参数整定找最佳， 从小到大顺序查。
先是比例后积分， 最后再把微分加。
曲线振荡很频繁， 比例度盘要放大。
曲线漂浮绕大弯， 比例度盘往小扳。
曲线偏离回复慢， 积分时间往下降。
曲线波动周期长， 积分时间再加长。
曲线振荡频率快， 先把微分降下来。
动差大来波动慢， 微分时间应加长。
理想曲线两个波， 前高后低四比一。
一看二调多分析， 调节质量不会低。

3. Re:C语言多线程编程
图片裂了。。。
--什么都不会!
4. Re:malloc()和free()的原理及实现
请问博主 malloc 函数的函数体在哪个文件下啊
--ssif
5. Re:PID控制温度
@ lanmanck不好意思，最近比较忙，所以现在才看到。PID的输出是根据PWM来控制加热丝的"加热档位"。举个例子，100%的PWM，对于加热丝的加热速度是最快的。相反，0%就是说不会加热。而中间...
--小时候挺菜

**阅读排行榜**
1. C语言多线程编程(25127)
2. CRC校验的C语言实现(9812)
3. malloc()和free()的原理及实现(8125)
4. 利用shell脚本添加环境变量(6736)
5. 错误解决：error: expected ';', ',' or ')' before '&' token(5338)

四、接下来我们用例子来辅助我们把常用的PID模型讲解了。（PID控制并不一定要三者都出现，也可以只是PI、PD控制，关键决定于控制的对象。）（下面的内容只是介绍一下PID模型，可以不看，对理解PID没什么用）

例子：我们要控制一个人，让他一PID的控制方式来行走110步后停下来。

1）P比例控制，就是让他按照一定的比例走，然后停下。比如比例系数为108，则走一次就走了108步，然后就不走了。

说明：P比例控制是一种最简单的控制方式，控制器的输出与输入误差信号成比例关系。但是仅有比例控制时系统输出存在稳态误差。比如上面的只能走到108，无论怎样都走不到110。

2）PI积分控制，就是按照一定的步伐走到112步然后回头接着走，走到108步位置时，然后又回头向110步位置走。在110位置处来回晃荡几次，最后停在110步的位置。说明：在积分I控制中，控制器的输出与输入误差信号的积分成正比关系。对一个自动控制系统来说，如果在进入稳态后存在稳态误差，则称这个控制系统是有稳态误差的或简称有差系统。为了消除稳态误差，在控制器中必须引入"积分项"。积分项对误差的影响取决于时间的积分，随着时间的增加，积分项会增大。这样，即便误差很小，积分项也会随着时间的增加而加大，它推动控制器的输出增大，从而使稳态误差进一步减小，直到等于0。因此，比例+积分（PI）控制器可以使系统在进入稳态后无稳态误差。

3)PD微分控制，就是按照一定的步伐走到一百零几步后，再慢慢地走向110步的位置靠近，如果最后能精确停在110步的位置，就是无静差控制；如果停在110步附近（如109步或111步位置），就是有静差控制。

说明：在微分控制D中，控制器的输出与输入误差信号的微分（即误差的变化率）成正比关系。

自动控制系统在克服误差的调节过程中可能会出现振荡甚至失稳，原因是存在较大惯性组件（环节）或滞后组件，具有抑制误差的作用，其变化总是落后于误差的变化。解决的办法是使抑制误差作用的变化“超前”，即在误差接近于零时，抑制误差的作用就应该是零。这就是说，在控制器中仅引入“比例P”项往往是不够的，比例项的作用仅是放大误差的幅值，而目前需要增加的是“微分项”，它能预测误差变化的趋势。这样，具有比例+微分的控制器就能够提前使抑制误差的控制作用等于零，甚至为负值，从而避免了被控量的严重超调。所以对有较大惯性或滞后的被控对象，比例P+微分D（PD）控制器能改善系统在调节过程中的动态特性。

五、用小明来说明PID：

小明接到这样一个任务：有一个水缸有点漏水(而且漏水的速度还不一定固定不变)，要求水面高度维持在某个位置，一旦发现水面高度低于要求位置，就要往水缸里加水。 小明接到任务后就一直守在水缸旁边，时间长就觉得无聊，就跑到房里看小说了，每30分钟来检查一次水面高度。水漏得太快，每次小明来检查时，水都快漏完了，离要求的高度相差很远，小明改为每3分钟来检查一次，结果每次来水都没怎么漏，不需要加水，来得太频繁做的是无用功。几次试验后，确定每10分钟来检查一次。这个检查时间就称为采样周期。 开始小明用瓢加水，水龙头离水缸有十几米的距离，经常要跑好几趟才加够水，于是小明又改为用桶加，一加就是一桶，跑的次数少了，加水的速度也快了，但好几次将缸给加溢出了，不小心弄湿了几次鞋，小明又动脑筋，我不用瓢也不用桶，老子用盆，几次下来，发现刚刚好，不用跑太多次，也不会让水溢出。这个加水工具的大小就称为比例系数。

小明又发现水虽然不会加过量溢出了，有时会高过要求位置比较多，还是有打湿鞋的危

险。他又想了个办法，在水缸上装一个漏斗，每次加水不直接倒进水缸，而是倒进漏斗让它慢慢加。这样溢出的问题解决了，但加水的速度又慢了，有时还赶不上漏水的速度。于是他试着变换不同大小口径的漏斗来控制加水的速度，最后终于找到了满意的漏斗。漏斗的时间就称为积分时间。

小明终于喘了一口，但任务的要求突然严了，水位控制的及时性要求大大提高，一旦水位过低，必须立即将水加到要求位置，而且不能高出太多，否则不给工钱。小明又为难了！于是他又开努脑筋，终于让它想到一个办法，常放一盆备用水在旁边，一发现水位低了，不经过漏斗就是一盆水下去，这样及时性是保证了，但水位有时会高多了。他又在要求水面位置上面一点将水缸要求的水平面处凿一孔，再接一根管子到下面的备用桶里这样多出的水会从上面的孔里漏出来。这个水漏出的快慢就称为微分时间。

六、在代码中理解PID：（好好看注释，很好理解的。注意结合下面PID的公式）

首先看PID的增量型公式：

PID=Uk+KP*【E(k)-E(k-1)】+KI*E(k)+KD*【E(k)-2E(k-1)+E(k-2)】

在单片机中运用PID，出于速度和RAM的考虑，一般不用浮点数，这里以整型变量为例来讲述PID在单片机中的运用。由于是用整型来做的，所以不是很精确。但是对于一般的场合来说，这个精度也够了，关于系数和温度在程序中都放大了10倍，所以精度不是很高，但是大部分的场合都够了，若不够，可以再放大10倍或者100倍处理，不超出整个数据类型的范围就可以了。一下程序包括PID计算和输出两部分。当偏差>10度时全速加热，偏差在10度以内时为PID计算输出。

程序说明：下面的程序，先看main函数。可知在对定时器0初始化后就一直在执行

PID_Output（）函数。在PID_Output（）函数中先用iTemp变量来得到PID运算的结果，来决定是启动加热丝加热还是不启动加热丝。下面的if语句结合定时器来决定PID算法多久执行一次。PID_Operation()函数看似很复杂，其实就一直在做一件事：根据提供的数据，用PID公式把最终的PID值算出来。

```
#include <reg52.h>
typedef unsigned char      uChar8;
typedef unsigned int       uInt16;
typedef unsigned long int  uInt32;

sbit ConOut = P1^1;     //加热丝接到P1.1口

typedef struct PID_Value
{
    uInt32 liEkVal[3];          //差值保存，给定和反馈的差值
    uChar8 uEkFlag[3];          //符号，1则对应的为负数，0为对应的为正数
    uChar8 uKP_Coe;             //比例系数
    uChar8 uKI_Coe;             //积分常数
    uChar8 uKD_Coe;             //微分常数
    uInt16 iPriVal;             //上一时刻值
    uInt16 iSetVal;             //设定值
    uInt16 iCurVal;             //实际值
```

```
}PID_ValueStr;

PID_ValueStr PID;            //定义一个结构体，这个结构体用来存算法中要用到的各种数据
bit g_bPIDRunFlag = 0;       //PID运行标志位，PID算法不是一直在运算。而是每隔一定时间，算一次。
/* ********************************************************
/* 函数名称：PID_Operation()
/* 函数功能：PID运算
/* 入口参数：无（隐形输入，系数、设定值等）
/* 出口参数：无（隐形输出，U(k)）
/* 函数说明：U(k)+KP*[E(k)-E(k-1)]+KI*E(k)+KD*[E(k)-2E(k-1)+E(k-2)]
******************************************************** */
void PID_Operation(void)
{
   uInt32 Temp[3] = {0};   //中间临时变量
   uInt32 PostSum = 0;     //正数和
   uInt32 NegSum = 0;      //负数和
   if(PID.iSetVal > PID.iCurVal)              //设定值大于实际值否?
   {
      if(PID.iSetVal - PID.iCurVal > 10)      //偏差大于10否?
         PID.iPriVal = 100;                   //偏差大于10为上限幅值输出(全速加热)
```

```
    else                                //否则慢慢来
    {
        Temp[0] = PID.iSetVal - PID.iCurVal;    //偏差<=10,计算E(k)
        PID.uEkFlag[1] = 0;                     //E(k)为正数,因为设定值大于实际值
        /* 数值进行移位，注意顺序，否则会覆盖掉前面的数值 */
        PID.liEkVal[2] = PID.liEkVal[1];
        PID.liEkVal[1] = PID.liEkVal[0];
        PID.liEkVal[0] = Temp[0];
        /*
================================================================== */
        if(PID.liEkVal[0] > PID.liEkVal[1])             //E(k)>E(k-1)否?
        {
            Temp[0] = PID.liEkVal[0] - PID.liEkVal[1];  //E(k)>E(k-1)
            PID.uEkFlag[0] = 0;                         //E(k)-E(k-1)为正数
        }
        else
        {
            Temp[0] = PID.liEkVal[1] - PID.liEkVal[0];  //E(k)<E(k-1)
            PID.uEkFlag[0] = 1;                         //E(k)-E(k-1)为负数
        }
        /*
```

```
================================================================== */
    Temp[2] = PID.liEkVal[1] * 2;                //2E(k-1)
    if((PID.liEkVal[0] + PID.liEkVal[2]) > Temp[2]) //E(k-2)+E(k)>2E(k-1)否?
    {
        Temp[2] = (PID.liEkVal[0] + PID.liEkVal[2]) - Temp[2];
        PID.uEkFlag[2]=0;                        //E(k-2)+E(k)-2E(k-1)为正数
    }
    else                                         //E(k-2)+E(k)<2E(k-1)
    {
        Temp[2] = Temp[2] - (PID.liEkVal[0] + PID.liEkVal[2]);
        PID.uEkFlag[2] = 1;                      //E(k-2)+E(k)-2E(k-1)为负数
    }
    /*
================================================================== */
    Temp[0] = (uInt32)PID.uKP_Coe * Temp[0];     //KP*[E(k)-E(k-1)]
    Temp[1] = (uInt32)PID.uKI_Coe * PID.liEkVal[0]; //KI*E(k)
    Temp[2] = (uInt32)PID.uKD_Coe * Temp[2];     //KD*[E(k-2)+E(k)-2E(k-1)]
    /* 以下部分代码是讲所有的正数项叠加，负数项叠加 */
    /* ========= 计算KP*[E(k)-E(k-1)]的值 ========= */
    if(PID.uEkFlag[0] == 0)
        PostSum += Temp[0];                      //正数和
```

```
else
    NegSum += Temp[0];                      //负数和
/* ========= 计算KI*E(k)的值 ========= */
if(PID.uEkFlag[1] == 0)
    PostSum += Temp[1];                     //正数和
else
    ;  /* 空操作。就是因为PID.iSetVal > PID.iCurVal（即E(K)>0）才进入if的，
       那么就没可能为负，所以打个转回去就是了 */
/* ========= 计算KD*[E(k-2)+E(k)-2E(k-1)]的值 ========= */
if(PID.uEkFlag[2]==0)
    PostSum += Temp[2];             //正数和
else
    NegSum += Temp[2];              //负数和
/* ========= 计算U(k) ========= */
PostSum += (uInt32)PID.iPriVal;
if(PostSum > NegSum)                //是否控制量为正数
{
    Temp[0] = PostSum - NegSum;
    if(Temp[0] < 100 )              //小于上限幅值则为计算值输出
        PID.iPriVal = (uInt16)Temp[0];
    else PID.iPriVal = 100;         //否则为上限幅值输出
```

```
            }
            else                            //控制量输出为负数，则输出0(下限幅值输出)
                PID.iPriVal = 0;
        }
    }
    else PID.iPriVal = 0;                   //同上，嘿嘿
}
/* ********************************************************
/* 函数名称：PID_Output()
/* 函数功能：PID输出控制
/* 入口参数：无（隐形输入，U(k)）
/* 出口参数：无（控制端）
******************************************************** */
void PID_Output(void)
{
    static uInt16 iTemp;
    static uChar8 uCounter;
    iTemp = PID.iPriVal;
    if(iTemp == 0)
        ConOut = 1;     //不加热
    else ConOut = 0;    //加热
```

```
    if(g_bPIDRunFlag)  //定时中断为100ms(0.1S)，加热周期10S(100份*0.1S)
    {
        g_bPIDRunFlag = 0;
        if(iTemp) iTemp--;      //只有iTemp>0，才有必要减“1”
        uCounter++;
        if(100 == uCounter)
        {
            PID_Operation();   //每过0.1*100S调用一次PID运算。
            uCounter = 0;
        }
    }
}
/* ********************************************************
/* 函数名称：PID_Output()
/* 函数功能：PID输出控制
/* 入口参数：无（隐形输入，U(k)）
/* 出口参数：无（控制端）
******************************************************** */
void Timer0Init(void)
{
    TMOD |= 0x01;  // 设置定时器0工作在模式1下
```

```
    TH0 = 0xDC;
    TL0 = 0x00;     // 赋初始值
    TR0 = 1;        // 开定时器0
    EA = 1;         // 开总中断
    ET0 = 1;        // 开定时器中断
}

void main(void)
{
    Timer0Init();
    while(1)
    {
        PID_Output();
    }
}

void Timer0_ISR(void) interrupt 1
{
    static uInt16 uiCounter = 0;
    TH0 = 0xDC;
    TL0 = 0x00;
```

```
    uiCounter++;
    if(100 == uiCounter)
    {
        g_bPIDRunFlag = 1;
    }
}
```

分类: 嵌入式



小时候挺菜
关注 - 0
粉丝 - 16
+加关注

0 0

« 上一篇: const int *a与int *const a，const int *const a的区别
» 下一篇: PID的原理

posted @ 2018-07-26 17:59 小时候挺菜 阅读(4754) 评论(2) 编辑 收藏

## 评论列表

#1楼 2019-06-10 12:04 lanmanck

我有个疑问，你pid后输出是什么东西？比如你求差是温度的差，单位是度，然后你最后给的应该是pwm，pwm跟寄存器有关，最大可能256，也可能4096，单位跟寄存器有关，那么度跟寄存器数据，怎么对应起来呢？温度的差怎么得到寄存器的增量呢？

支持(0) 反对(0)

#2楼 [楼主] 2019-06-21 17:29 小时候挺菜

@ lanmanck

不好意思，最近比较忙，所以现在才看到。

PID的输出是根据PWM来控制加热丝的“加热档位”。举个例子，100%的PWM，对于加热丝的加热速度是最快的。相反，0%就是说不会加热。而中间的数值，就是中间的档位。PID算法里输入温度的偏差，调节系数之后，输出的是加热丝在此时刻的加热速度。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部



**相关博文:**