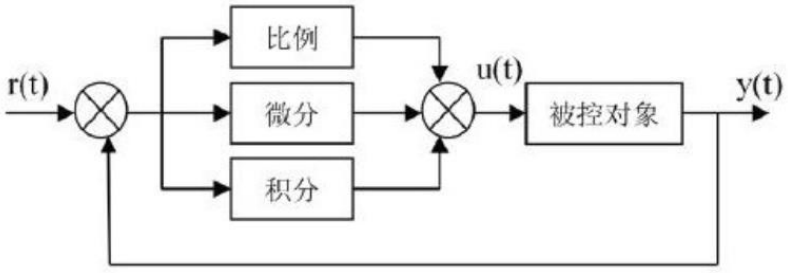
PID 算法在运动控制中的应用

AI 航 团队

PID 控制算法作为自动控制系统中广泛应用的一种控制算法，小到控制一个元件的温度，大到控制无人机的飞行姿态和飞行速度等等，都可以使用 PID 控制。控制器问世至今已有近 70 年历史，它以其结构简单、稳定性好、工作可靠、调整方便而成为工业控制的主要技术之一。资料很多，本文只做抛砖引玉，部分内容来源于互联网整理，侵删。

1. PID 控制原理及特点

PID，就是“比例（proportional）、积分（integral）、微分（derivative）”三个调节参数的缩写，而将偏差的比例、积分和微分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，这样的控制器便称 PID 控制器。



PID 调节器是一种线性调节器，它将给定值 r(t)与实际输出值 y(t)的偏差的比例(P)、积分(I)、微分(D)通过线性组合构成控制量，对控制对象进行控制。

(1)PID 控制器的微分方程

$$u(t) = K_P\left[e(t) + \frac{1}{T_I}\int_0^t e(t)dt + T_D\frac{de(t)}{dt}\right]$$

式中 $e(t) = r(t) - c(t)$

(2)PID 调节器各校正环节的作用

比例环节：即时成比例地反应控制系统的偏差信号 e(t)，偏差一旦产生，调节器立即产生控制作用以减小偏差。

$$Kp * e(t)$$

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。

积分环节：主要用于消除静差，提高系统的无差度。积分作用的强弱取决于积分时间常数 TI，TI 越大，积分作用越弱，反之则越强。

$$\frac{Kp}{Ti}\int_0^t e(t)dt$$

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。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。

微分环节：能反应偏差信号的变化趋势(变化速率)，并能在偏差信号的值变得太大之前，在系统中引入一个有效的早期修正信号，从而加快系统的动作速度，减小调节时间。

$$Kp * Td\frac{de(t)}{dt}$$

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。

2. 数字 PID 控制器

在计算机控制系统中，由于控制是使用采样控制，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，进行连续控制。所以数字 PID 控制也属于离散型控制系统。数字型 PID 控制算法可分为位置式 PID 和增量式 PID 控制算法。
位置式 PID：位置式 PID 控制的输出与整个过去的状态有关，用到了误差的累加值；
增量式 PID：而增量式 PID 的输出只与当前拍和前两拍的误差有关。

（1） 位置式 PID

由于是离散型控制系统，积分项和微分项不能直接使用，必须进行离散化处理。离散化处理的方法为：以 T 作为采样周期，k 作为采样序号，则离散采样时间 kT 对应着连续时间 t，用矩形法数值积分近似代替积分，用一阶后向差分近似代替微分，可作如下近似变换：

$$\begin{cases} t \approx kT \ \ (k = 0,1,2 \dots \dots) \\ \int_0^t e(t)dt \approx \mathrm{T}\sum_{j=0}^{k} e(jT) = T\sum_{j=0}^{k} e_j \\ \frac{de(t)}{dt} \approx \frac{e(kT) - e[(k-1)T]}{T} = \frac{e_k - e_{k-1}}{T} \end{cases}$$

根据以上公式的转换，便可得到离散的 PID 表达式：

$$u_k = \mathrm{Kp}[e_k + \frac{T}{Ti}\sum_{j=0}^{k} e_j + Td\frac{e_k - e_{k-1}}{T}]$$

位置式 PID 算法的特点：

由于全量输出，所以每次输出均与过去状态有关，计算时要对 e_k 进行累加，工作量大；并且，因为计算机输出的 u_k 对应的是执行机构的实际位置，如果计算机出现故障，输出的 u_k 将大幅度变化，会引起执行机构的大幅度变化，有可能因此造成严重的生产事故，这在实生产际中是不允许的。

（2） 增量式 PID

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 控制算法可以通过位置式 PID 公式推导出。由位置式 PID 公式可以得到控制器的第 k－1 个采样时刻的输出值为：

$$u_{k-1} = Kp[e_{k-1} + \frac{T}{Ti}\sum_{j=0}^{k-1} e_j + Td\frac{e_{k-1} - e_{k-2}}{T}]$$

则：

$$\Delta u_k = u_k - u_{k-1}$$

简化后可得：

$$\Delta u_k = Kp*[e_k - e_{k-1}] + Ki*e_k + Kd*[e_k - 2*e_{k-1} + e_{k-2}]$$

增量式 PID 算法的特点：

增量式 PID 控制算法与位置式 PID 算法公式相比，如果计算机控制系统采用恒定的采样周期 T，一旦确定 Kp、Ti、Td 参数，只要使用前后三次测量的偏差值，就可以由增量式 PID 公式求出控制量。计算量小的多，因此在实际中得到广泛的应用。

3. 实例讲解

下面列举一个温度控制实例来讲解增量式 PID 控制算法的应用。

要使用 PID 算法控制的系统某个参数，必须有该参数的相关反馈变量。比如，现在要控制水箱的温度，那我们就得通过温度传感器采集得到的反馈数据。通过设定水箱的目标温度，采用增量式 PID 控制算法，控制水箱的加温元件，使得水箱在不同的外界温度下还是保持在目标速度附近波动。

(1)设置 PID 参数及变量

首先，我们定义温度的目标温度、PID 是比例、积分、微分常数，及前后三次偏差值

```
int16 High_temp       = 50; //预设温度上限
int16 Low_temp  = 30; //预设温度下限
int16 Set_temp      = 45;      //设置目标温度
int16 Proportion  = 64;   //  比例常数  Proportional Const
int16 Integral      = 0;          //  积分常数  Integral Const
int16 Derivative  = 54; //  微分常数  Derivative Const
float LastError;                  // Error[-1]
float PrevError;                  // Error[-2]
float SumError;                   // Sums of Errors
```

(2)PID 运算函数

在实际运算时，由于水具有很大的热惯性，而且 PID 运算中的 I（积分项）具有非常明

显的延迟效应，积分项常数不宜过大或者可直接省略积分环节，下面 C 代码所示为 PD 控制的实现过程：

```
float PID_Calc(float NextPoint ,float SetPoint)
{
    //增量法计算公式：
    //Pdt=Kp*[E(t)-E(t-1)]+Ki*E(t)+Kd*[E(t)-2*E(t-1)+E(t-2)]
    float D_Error,Error;
    float II;
    Error = SetPoint-NextPoint;              //偏差
    SumError+=Error;                         // 积分
    D_Error =LastError-PrevError;            // 当前微分
    PrevError =   LastError;
    LastError =   Error;
    II = Integral*SumError/10000.0;          //积分缩小 10000 倍
    if(II>30)                                  //积分饱和限制
    {
        II=30;
    }
    return (Proportion*Error+II+Derivative*D_Error);
}
```

(3)水箱实际温度控制

水箱的水温加热棒控制采用 PWM 的控制，PWM 占空比越大，加热速度越快。以下函数为当目标温度大于实际温度时，PID 控制器工作，对加温进行控制；当目标温度小于实际温度时，PWM 占空比为 0，停止加热。

```
void Control_Temp(void)
{
    Now_temp = Temp_numbe;
    Target_temp =   Set_temp;
    if(Now_temp>High_temp)
    {
        PWM_duty = 0;
    }
    else
    {
        Temp_Out = PID_Calc(Now_temp,Target_temp);
        if(Temp_Out >= 100)  Temp_Out = 100;
        else if(Temp_Out <=0)Temp_Out = 0;
        //不同的温度设置不同的比例补偿热量损失
        //Temp_Out = Temp_Out+Target_temp/80.0*20;
        PWM_duty = (int)Temp_Out;
    }
}
```

通过以上三个步骤，然后设定适当的采样时间，便完成了 PID 控制算法的实现。然后我们便可根据实际的控制环境，对比例、积分、微分参数逐一进行整定。

4. PID 参数整定方法

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，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。

5. 轻舟机器人 PID 速度控制

```
/**************************************************************************
函数功能：增量 PI 控制器
入口参数：编码器测量值，目标速度
返回　值：电机 PWM
根据增量式离散 PID 公式
pwm+=Kp[e（k）-e(k-1)]+Ki*e(k)+Kd[e(k)-2e(k-1)+e(k-2)]
e(k)代表本次偏差
e(k-1)代表上一次的偏差　以此类推
pwm 代表增量输出
在我们的速度控制闭环系统里面，只使用 PI 控制
pwm+=Kp[e（k）-e(k-1)]+Ki*e(k)
**************************************************************************/
int Incremental_PI_Left (int Encoder,int Target) //左轮电机控制量计算
{
    static int Bias,Pwm,Last_bias;
    Bias=Encoder-Target;                    //计算偏差
    Pwm+=Velocity_KP*(Bias-Last_bias)+Velocity_KI*Bias;   //增量式 PI 控制器
    if(Pwm>7200)Pwm=7200;
    if(Pwm<-7200)Pwm=-7200;
    Last_bias=Bias;                         //保存上一次偏差
    return Pwm;                             //增量输出
}
int Incremental_PI_Right (int Encoder,int Target)//右轮电机控制量计算
{
    static int Bias,Pwm,Last_bias;
```

```
Bias=Encoder-Target;                    //计算偏差
Pwm+=Velocity_KP*(Bias-Last_bias)+Velocity_KI*Bias;     //增量式 PI 控制器
if(Pwm>7200)Pwm=7200;
if(Pwm<-7200)Pwm=-7200;
Last_bias=Bias;                         //保存上一次偏差
return Pwm;                             //增量输出
```