51 单片机 PID 算法程序(二)位置式 PID 控制算法

由 51 单片机组成的数字控制系统控制中，PID 控制器是通过 PID 控制算法实现的。51 单片机通过 AD 对信号进行采集，变成数字信号，再在单片机中通过算法实现 PID 运算，再通过 DA 把控制量反馈回控制源。从而实现对系统的伺服控制。

位置式 PID 控制算法

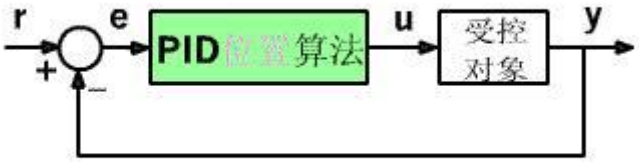
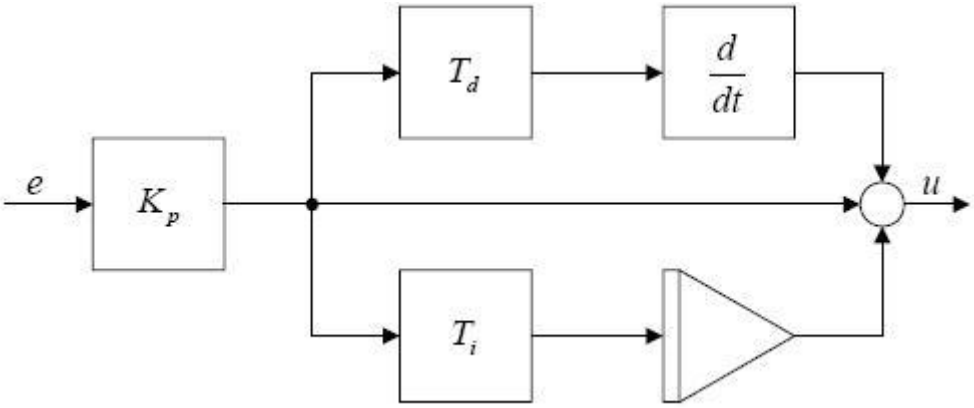


图3-2 位置式PID控制算法的简化示意图

位置式 PID 控制算法的简化示意图



上图的传递函数为：

$$\frac{u}{e}(s)=H(s)=K_p\left(1+\frac{1}{T_i s}+T_d s\right) \quad (2\text{-}1)$$

在时域的传递函数表达式

$$u(t)=K_p\left(e(t)+\frac{1}{T_i}\int_0^t e(\sigma)d\sigma+T_d\frac{de(t)}{dt}\right) \quad (2\text{-}2)$$

对上式中的微分和积分进行近似

$$\int_0^t e(\sigma)d\sigma \approx T\sum_{k=0}^{n} e(k) \qquad \frac{de(t)}{dt} \approx \frac{e(n)-e(n-1)}{T} \qquad t=nT \tag{2-3}$$

式中 n 是离散点的个数。
于是传递函数可以简化为：

$$u(n)=K_p e(n)+K_i\sum_{k=0}^{n} e(k)+K_d\left(e(n)-e(n-1)\right) \tag{2-4}$$

其中

$$K_i=\frac{K_p T}{T_i} \qquad K_d=\frac{K_p T_d}{T}$$

u(n)——第 k 个采样时刻的控制；
K_P ——比例放大系数；
K_i ——积分放大系数；
K_d ——微分放大系数；
T ——采样周期。

如果采样周期足够小，则（2-4）的近似计算可以获得足够精确的结果，离散控制过程与连续过程十分接近。

（2-4）表示的控制算法直接按（2-1）所给出的 PID 控制规律定义进行计算的，所以它给出了全部控制量的大小，因此被称为全量式或位置式 PID 控制算法。

缺点：

1） 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)(k=0,1,…n)进行累加，工作量大。

2） 因为计算机输出的 u(n)对应的是执行机构的实际位置，如果计算机出现故障，输出 u(n)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，这在实际生产中是不允许的。

位置式 PID 控制算法 C51 程序

具体的 PID 参数必须由具体对象通过实验确定。由于单片机的处理速度和 ram 资源的限制，一般不采用浮点数运算，而将所有参数全部用整数，运算到最后再除以一个 2 的 N 次方数据（相当于移位），作类似定点数运算，可大大提高运算速度，根据控制精度的不同要求，当精度要求很高时，注意保留移位

引起的“余数”，做好余数补偿。这个程序只是一般常用 pid 算法的基本架构，没有包含输入输出处理部分。

```
#include <reg52.h>
#include <string.h>          //C 语言中 memset 函数头文件


/*============================================================
=========================================
PID Function
The PID (比例、积分、微分) function is used in mainly
control applications. PIDCalc performs one iteration of the PID
algorithm.
While the PID function works, main is just a dummy program showing
a typical usage.
============================================================
=========================================*/


typedef struct PID {
double SetPoint;      // 设定目标 Desired value
double Proportion;    // 比例常数 Proportional Const
double Integral;      // 积分常数 Integral Const
double Derivative;    // 微分常数 Derivative Const
double LastError;     // Error[-1]

double PrevError;     // Error[-2]
double SumError;      // Sums of Errors
} PID;
/*============================================================
=========================================
PID 计算部分
============================================================
=========================================*/
double PIDCalc( PID *pp, double NextPoint )
{
double dError, Error;
Error = pp->SetPoint - NextPoint;          // 偏差
pp->SumError += Error;                     // 积分
dError = Error - pp->LastError;            // 当前微分
pp->PrevError = pp->LastError;
pp->LastError = Error;
return (pp->Proportion * Error // 比例项
+ pp->Integral * pp->SumError // 积分项
```

```
+ pp->Derivative * dError //  微分项
);
}
/*=========================================================
=========================================
Initialize PID Structure   PID 参数初始化
=========================================================
=========================================*/
void PIDInit (PID *pp)
{
memset ( pp,0,sizeof(PID));
}
/*=========================================================
=========================================
Main Program    主程序
=========================================================
=========================================*
double sensor (void) // Dummy Sensor Function
{
return 100.0;
}
void actuator(double rDelta) // Dummy Actuator Function
{}
void main(void)
{
PID sPID; // PID Control Structure
double rOut; // PID Response (Output)
double rIn; // PID Feedback (Input)
PIDInit ( &sPID ); // Initialize Structure
sPID.Proportion = 0.5; // Set PID Coefficients
sPID.Integral = 0.5;
sPID.Derivative = 0.0;
sPID.SetPoint = 100.0; // Set PID Setpoint
for (;;) { // Mock Up of PID Processing
rIn = sensor (); // Read Input
rOut = PIDCalc ( &sPID,rIn ); // Perform PID Interation
actuator ( rOut ); // Effect Needed Changes
}
```