比例主要用于偏差的“粗调”，

保证控制系统的“稳”；积分主要用于偏差的“细

调”，保证控制系统的“准”；微分主要用于偏差的

“细调”，保证控制系统的“快”．

很多同学都不清楚PID是个什么东西，因为很多不是自动化的学生。他们开口就要资料，

程序。

这是明显的学习方法不对，起码，首先，你要理解PID是个什么东西。

本文以通俗的理解，以小车纵向控制举例说明PID的一些理解。

首先，为什么要做PID？  
由于外界原因，小车的实际速度有时不稳定，这是其一，  
要让小车以最快的时间达达到既定的目标速度，这是其二。  
速度控制系统是闭环，才能满足整个系统的稳定要求，必竟速度是系统参数之一，这是其三.

小车调速肯定不是线性的，外界因素那么多，没人能证明是线性的。如果是线性的，直接用P就可以了。

比如在PWM=60%时，速度是2M/S，那么你要它3M/S，就把PWM提高到90%。因为90/60=3/2，这样一来太完美了。  
完美是不可能的。  
那么不是线性的，要怎么怎么控制PWM使速度达到即定的速度呢？即要快，又要准，又要狠。（即快准狠

）系统这个速度的调整过程就必须通过某个算法调整，一般PID就是这个所用的算法。

可能你会想到，如果通过编码器测得现在的速度是2.0m/s,要达到2.3m/s的速度，那么我把pwm增大一点不

就行了吗？是的，增大pwm多少呢？必须要通过算法，因为PWM和速度是个什么关系，对于整个系统来说，谁也

不知道。要一点一点的试，加个1%,不够，再加1%还是不够，那么第三次你还会加1%吗？很有可能就加2%了。

通过PID三个参数得到一个表达式：△PWM=a \*△V1+b \*△V2+c \*△V3,a b c是通过PID的那个长长的公式展开

，然后约简后的数字，△V1 ，△V2 ，△V3 此前第一次调整后的速度差 ,第二次调整后的速度差,第三次。。

。。。一句话，PID要使当前速度达到目标速度最快，需要建立如何调整pwm和速度之间的关系。

输入输出是什么：  
输入就是前次速度，前前次速度，前前前次速度。  
输出就是你的PWM应该增加或减小多少。

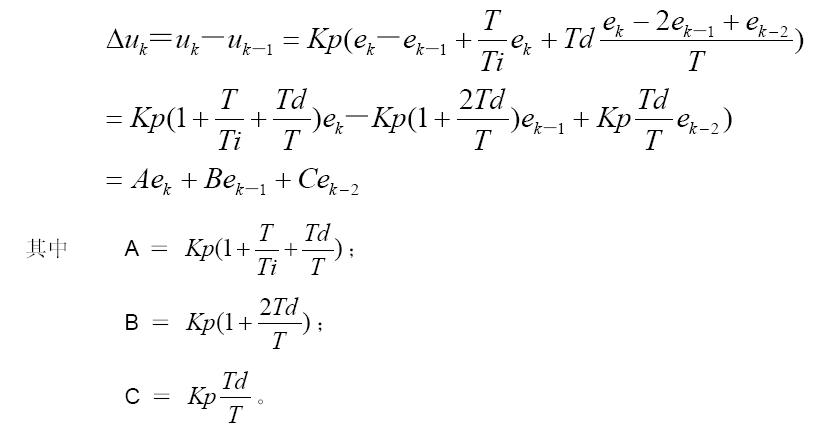
为了避免教科书公式化的说明，本文用口语化和通俗的语言描述。虽然不一定恰当，但意思差不多，就是那个事。如果要彻头彻尾地弄PID，建议多调试，写几个仿真程序。

PID一般有两种：位置式PID和增量式PID。在小车里一般用增量式，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。

下面均以增量式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D微分就是加快调节过程的作用了。

公式本来需要推导的，我就不来这一套了。直接贴出来：

看看最后的结果：

△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OK了。PID三个参数的确定有很多方法，不在本文讨论范围内。采样周期也是有据可依的，不能太大，也不能太小。

制模型：你控制一个人让他以PID控制的方式走110步后停下。   
（1）P比例控制，就是让他走110步，他按照一定的步伐走到一百零几步（如108步）或100多步（如112步）就停了。   
说明：   
P比例控制是一种最简单的控制方式。其控制器的输出与输入误差信号成比例关系。当仅有比例控制时系统输出存在稳态误差（Steady-state error）。   
  
（2）PI积分控制，就是他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步位置处来回晃几次，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，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（System with Steady-state Error）。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使稳态误差进一步减小，直到等于零。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，就是他按照一定的步伐走到一百零几步后，再慢慢地向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，其原因是由于存在有较大惯性组件（环节）或有滞后（delay）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。  
  
解释二：  
  
　　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。 看到几个问采样周期的帖子，临时想了这么个故事。微分的比喻一点牵强，不过能帮助理解就行了，呵呵，入门级的，如能帮助新手理解下PID，于愿足矣。故事中小明的试验是一步步独立做，但实际加水工具、漏斗口径、溢水孔的大小同时都会影响加水的速度，水位超调量的大小，做了后面的实验后，往往还要修改改前面

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？指针，汇编，移位？都可以试！

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//定义PID结构体

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef struct PID

{

int 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

int LastError; //Error[-1]

int PrevError; //Error[-2]

} PID;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//定义相关宏

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define P\_DATA 100

#define I\_DATA 0.6

#define D\_DATA 1

#define HAVE\_NEW\_VELOCITY 0X01

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//声明PID实体

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static PID sPID;

static PID \*sptr = &sPID;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//PID参数初始化

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void IncPIDInit(void)

{

sptr->LastError = 0; //Error[-1]

sptr->PrevError = 0; //Error[-2]

sptr->Proportion = P\_DATA; //比例常数 Proportional Const

sptr->Integral = I\_DATA; //积分常数Integral Const

sptr->Derivative = D\_DATA; //微分常数 Derivative Const

sptr->SetPoint =100; 目标是100

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//增量式PID控制设计

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int IncPIDCalc(int NextPoint)

{

int iError, iIncpid; //当前误差

iError = sptr->SetPoint - NextPoint; //增量计算

iIncpid = sptr->Proportion \* iError //E[k]项

- sptr->Integral \* sptr->LastError //E[k－1]项

+ sptr->Derivative \* sptr->PrevError; //E[k－2]项

sptr->PrevError = sptr->LastError; //存储误差，用于下次计算

sptr->LastError = iError;

return(iIncpid); //返回增量值

}

Int g\_CurrentVelocity;

Int g\_Flag;

void main(void)

{

DisableInterrupt

InitMCu();

IncPIDInit();

g\_CurrentVelocity=0; //全局变量也初始化

g\_Flag=0; //全局变量也初始化

EnableInterrupt;

While(1)

{

if (g\_Flag& HAVE\_NEW\_VELOCITY)

{

PWMOUT+= IncPIDCalc(CurrentVelocity);

g\_Flag&=~ HAVE\_NEW\_VELOCITY;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//采样周期T

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Interrrupt TIME void

{

CurrentVelocity =GetCurrentVelocity;

g\_Flag|= HAVE\_NEW\_VELOCITY;

}