

本文翻译自 <http://www.embedded.com/2000/0010/0010feat3.htm>



PID Without a PhD

Tim Wescott

PID（比例，积分，微分）控制没有看起来那么复杂，阅读下面简单的实现步骤，效果立竿见影。

在工作中，我是三个号称伺服员（servo guys）中的一个，唯一一个用软件来实现控制系统中的环路控制。因此，我经常有机会为各种项目设计数字控制环路。我发现，虽然控制系统的问题需要大量专业知识，但是大多数控制系统问题能用简单的方法来解决，根本就不诉诸任何控制理论。这篇文章将告诉你不用大量的数学，无需学习任何控制理论，如何来实施和整定一个简单的控制系统。

PID 控制

PID 控制器以各种形式使用超过了 1 世纪，广泛应用在机械设备、气动设备和电子设备。采用微处理器的数字 PID 控制器已经出现，正如你即将看到，嵌入一个 PID 控制器到你的代码中是一个简单的任务。

PID 实指“比例 proportional”、“积分 integral”、“微分 derivative”，这三项构成 PID 基本要素。每一项完成不同任务，对系统功能产生不同的影响。

典型的 PID 系统中，输入命令值（system command）和被控器（通常被称为 plant）反馈信号的组合作为 PID 控制器三项的输入，三项的输出相加起来形成 PID 系统的控制输出。

图 1 表示一个基本 PID 控制器的方框图，图中微分项的输入只有被控设备的反馈信号。命令信号（command signal）减去被控设备反馈信号产生误差（error）。这一误差信号送入比例项和积分项。三项产生的结果相加最后驱动被控设备。后面我会介绍这三项主要做了些什么。根据系统如何响应命令，图中用虚线增加了一个比例项位置（这可能是比例项最佳的放置位置）。

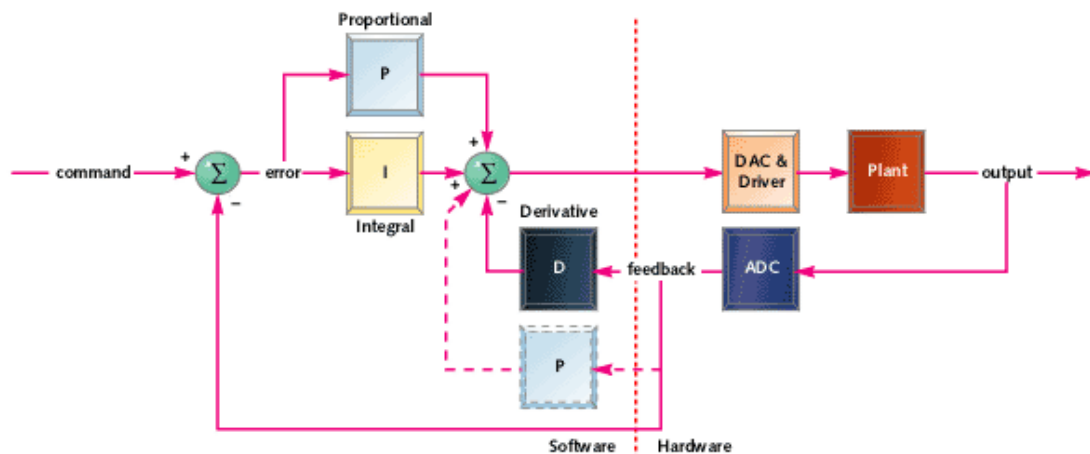


图 1：一个基本的 PID 控制器

被控器举例

我们需要举一些例子来增加读者对 PID 的印象，文中我会举三个例子来说明 PID 在不同应用中产生的效果。

- 电机驱动齿轮组
- 精密定位系统
- 恒温系统

每一个系统都有不同的特点，每一个需要不同的控制策略以获得最佳的性能。

电机和齿轮

第一个举例是一个电机驱动齿轮组，用一个电位器或者其它位置传感设备来监测齿轮组的位置。类似的应用像打印机里的驱动器、汽车巡航控制器里的油门，或者其它中等精度位置控制器。图 2 中的电机驱动电压大小由软件命令控制，电机输出通过齿轮减速驱动实际的机械装置，最终位置由一个电位器测量。

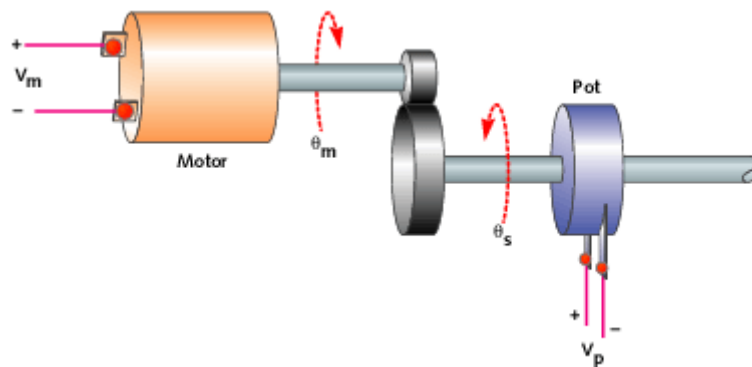


图 2：电压驱动型电机和齿轮组

一个电压型直流电机输出转速与电压成正比。通常电机的电枢具有一定的电阻，限制了电机的加速性，那么改变输入电压与得到最终输出转速之间会有延迟。齿轮传动以一常数比例放大了这一延迟，最后通过电位器来测量输出轴的位置。

图 3 显示了电机和齿轮的阶跃响应。时间常量间隔 $t_0=0.2s$ ，这一系统的阶跃响应是输出行为对输入在 $t=0$ 时由 0 变为某一常量值的响应。作为一个普通的例子，在这里我用小数组成整个阶跃响应，最大为 1。图 3 显示阶跃输入和电机的响应，随着时间的增长电机缓慢响应，但电机位置的变化不可能是一个常速度。

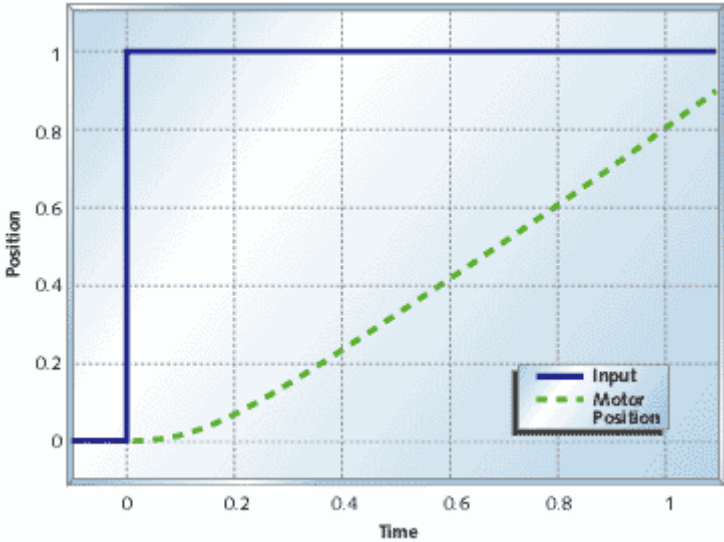


图 3：电机与齿轮位置 vs. 时间

精密驱动器

有时候需要非常精确地控制物体的位置，精密位置系统可由自由移动的机械台、扬声器线圈（线圈和磁铁）、非接触式的位置传感器组成。

你可能在某些光学系统稳定装置，或者某些设备和传感器定位装置中看到过这种机构。图 4 显示的就是这种系统。软件命令控制着线圈中的电流，电流建立的磁场与磁铁产生了排斥力，磁铁与台板相连，台板移动的加速度正比于线圈的电流大小。最后，一个非接触式位置传感器来检测台板的位置。

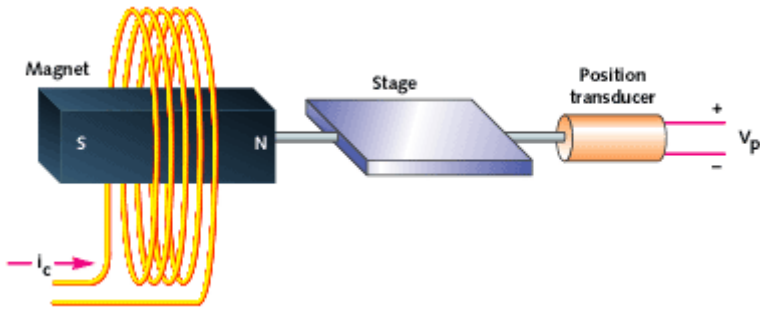


图 4：精密驱动器

这一机构中的磁力与台板移动的力独立开来，好处就是使得台板不受外力的影响，缺点就是造成系统非常“滑”，控制起来有难度。另外，电子必须的一个好的电流输出形放大器和非接触式位置传感器做起来也有挑战。可以预计，如果你做了这样一个项目（或者是接了个短期项目），你就是这个相当优秀的团队中一员。

这个系统的运动方程非常简单，台板上的力只与驱动命令成比例关系，所以系统的加速度与驱动输出也是成比例关系。系统自身的阶跃响应是一条抛物线，见图 5。由于有惯性，台板动起来就会一直动，这会导致系统控制更加困难，这个我会在后面讲到。

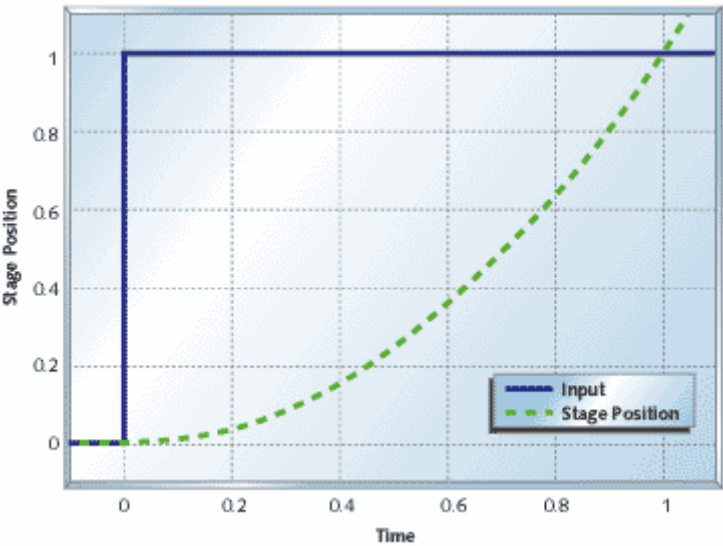


图 5：精密驱动器位置 vs. 时间

温度控制

第三个例子是一个加热器，图 6 显示了系统简单示意图。电加热器加热一个大容器，通过温度传感装置来获得温度值。

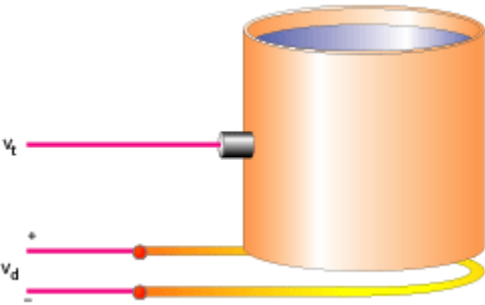


图 6：加热器

加热系统往往具有非常复杂的响应，我准备忽略一些细节，给出一个非常近似的模型，除非你对性能要求非常严格，那么一个精确的模型没什么必要。

图 7 显示了在 V_d （一定电压值？）下该系统的阶跃响应的变化。时间常量

$t1=0.1s$, $t2=0.3s$, 给定一个输出驱动值, 系统响应往往会稳定在一个恒定温度, 但它需要花费很长的时间。同时, 没有保温隔热层的时候, 加热系统往往会对外界影响敏感。图中没有表示出来这些影响, 但文章的后面我会进行研究。

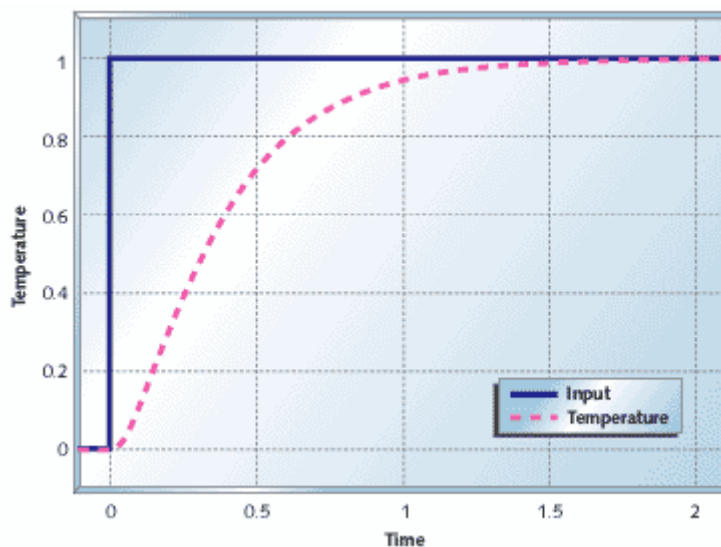


图 7: 加热器温度 vs. 时间

控制器

这里控制器的要素输入既有被控设备输出信号, 也有误差信号, 这也是被控设备输出与系统输入命令之间的区别。我会在讨论时用浮点数写一些控制代码来保持精度。你也可以用整形或者定点数来写你自己的代码。

我将假设一个函数如下, 随着讨论的进展, 你将看到具体的数据结构和函数的内部形状了。

```
double UpdatePID(SPId * pid, double error, double position)
{
...
...
...
}
```

之所以在 UpdatePID 函数接口中定义误差输入 (error), 而不直接定义控制命令输入 (command), 是因为有时候可以直接用误差 (error) 来处理。在函数内部没有引入误差计算的话, 函数的应用适用性会增强。函数会像这样使用:

```
...
position = ReadPlantADC();
drive = UpdatePID(&plantPID, plantCommand - position, position);
DrivePlantDAC(drive);
...
```

比例项（pre po）

比例控制是最容易实现的反馈控制，而且简单的比例控制可能是最常见的控制回路。一个比例控制器仅仅是误差信号乘以一个常数，并输出送给驱动器。比例项的计算代码如下：

```
double pTerm;  
...  
pTerm = pid->pGain * error;  
...  
return pTerm;
```

图 8 显示了当你给电机和齿轮增加比例反馈控制后会发生什么。当小增益（ $k_p=1$ ）时，电机慢慢转到正确的位置。增加增益（ $k_p=2$ ）后，反应加快。在（ $k_p=5$, $k_p=10$ ）时电机启动非常快，并且过冲。最后系统由于没有变成理想的低增益来使系统快速稳定，而是存在着许多过冲。如果我们进一步增加增益值，系统最终会到了某一点在目标位置值上下振荡，再也不会稳定，这时候系统就变得不稳定。

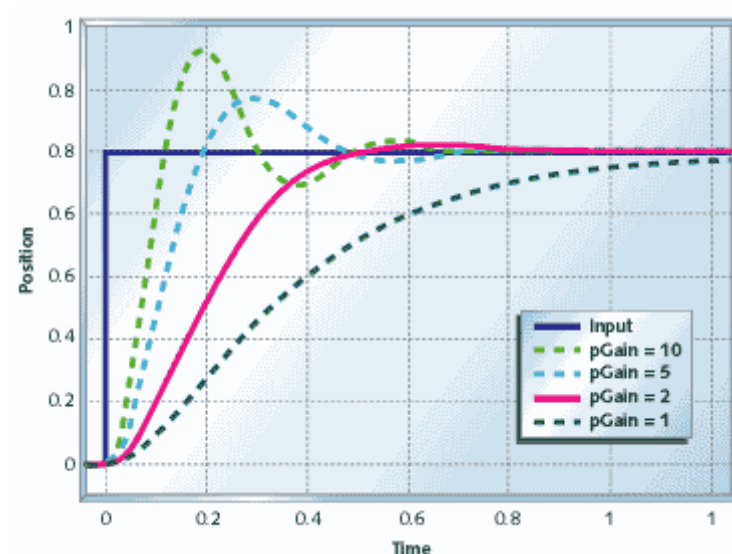


图 8：比例反馈控制的电机和齿轮位置 vs. 时间

电机和齿轮从启动到过冲是因为电机的反应延迟。回看图 2，你可以看到电机位置上升并不及时。这一延迟，加上高反馈增益，引起了图 8 中的过冲。图 9 显示的是比例反馈的精密驱动器的响应，单独比例控制显示对这一系统不适应，不管增益有多低，被控器总是有大量的延迟，系统就会产生振荡。随着比例增益的增加，输出振荡的频率也会增加，但系统不会稳定。

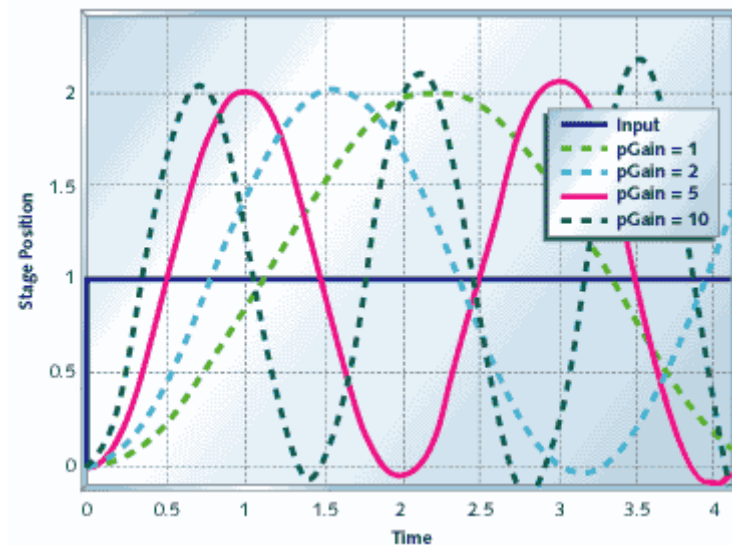


图 9：比例反馈的精密驱动器 vs. 时间

图10显示的是当你用纯比例反馈控制温度控制器时会发生什么，图中显示了在 $t=2s$ ，外部环境温度改变造成干扰时，系统输出的响应。就算没有外部干扰，你也可以看到比例控制不能将温度稳定到想要的值。增加比例增益会起点作用，但就算是把增益设成10，输出始终要低于目标值，而且你会看到一个强大的过冲在来回波动（这被叫作振铃）。

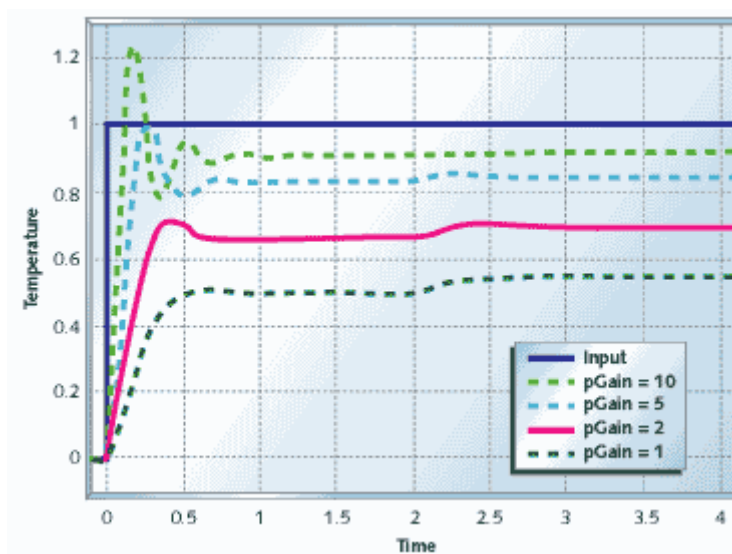


图10：比例反馈控制的温度控制器
time=2 时外部干扰导致波动

前面的例子表明，单独的比例控制在某时很有用，但不能全部见效，被控设备有很多的延迟，像精密驱动器，就根本不能用比例控制来稳定。像温度控制器这类被控设备，就不能设置到预定值。电机和齿轮组可以工作，但它们希望比只用单独的比例控制更快速地驱动。为了解决这些控制问题，你就要加入积分和微分控制。

积分项

积分控制用于控制回路中的长期精密控制。总是几乎和比例控制一起使用。

积分控制应用代码如下，`iState`（积分状态）是指之前所有输入（输入误差）之和。参数 `iMax` 和 `iMin` 用来限定积分状态的最大最小值。

```
double iTerm;  
...  
// calculate the integral state  计算积分状态  
// with appropriate limiting  适当的限制  
pid->iState += error;  
if (pid->iState > pid->iMax) { pid->iState = pid->iMax; }  
else if (pid->iState < pid->iMin) { pid->iState = pid->iMin; }  
iTerm = pid->iGain * iState; // calculate the integral term  计算积分项  
...
```

积分控制本身通常会降低整体的稳定，甚至完全摧毁稳定性。

图 11 显示了纯积分控制的电机和齿轮（`pGain = 0`）。这系统不能工作，就像比例控制的精密驱动器，积分控制的电机齿轮系统会一波大一波不断振荡，直至到达某一极限点。（希望不是损坏的那点）

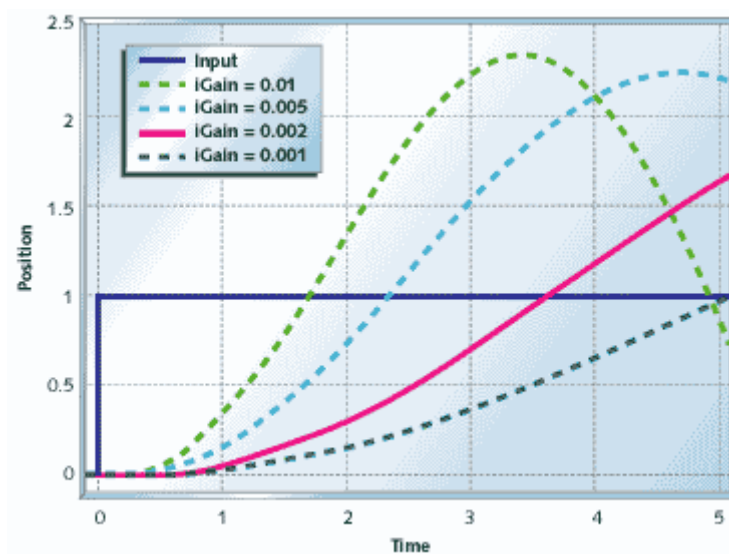


图 11：纯积分控制的电机和齿轮

图 12 显示了纯积分控制的温度控制系统，系统达到预定目标所花的时间比例控制（见图 10）更长。但是可以注意到即使途中被干扰，仍能正确到达设定值。如果你手头的系统不需要快速的设置，这可也是个可用的系统。

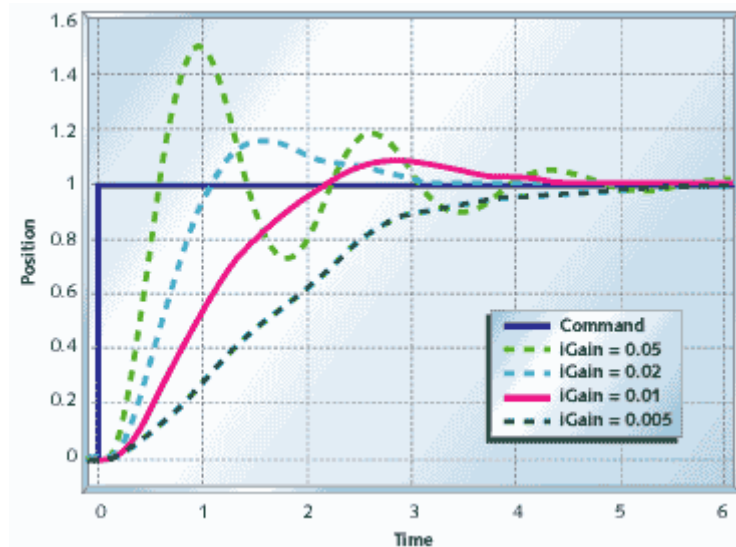


图12：积分控制的温度控制系统
time=2 时外部干扰导致波动

图 12 告诉我们为什么要采用积分项。积分状态“记忆”了前面所有的误差，这是这个就可以让控制器消除输出的长期误差。同样这个积分状态也可能会造成不稳定，因为在被控设备取得速度之后，控制器总是反应太慢。为了稳定这两个系统，你需要从比例项中获得一些当前值。

图 13 显示了 PI 控制的电机和齿轮，比较图 8 和图 11，可以看到比纯比例控制的系统到达预定位置所花的时间要长，但是不会到达错误的地方。

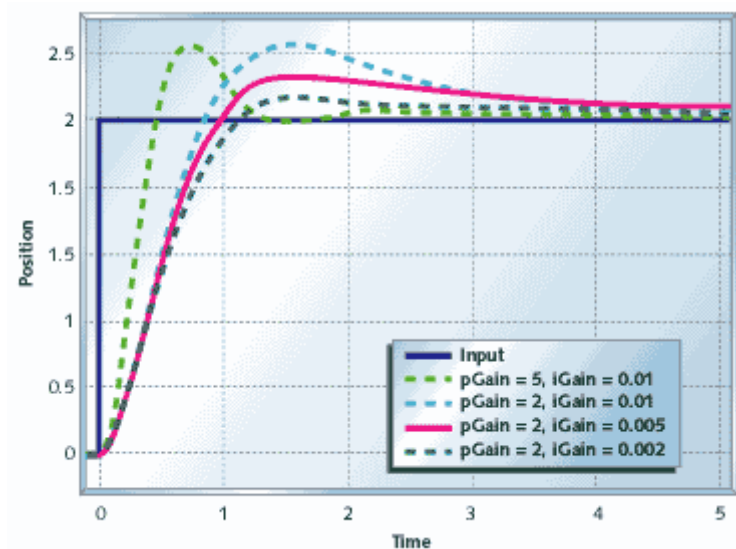


图 13：PI 控制的电机齿轮

图 14 显示了 PI 控制的加热系统，和纯积分控制一样，加热器依然可以到达正确的目标温度（见图 12），但是 PI 控制要快两到三倍的时间。这一数字显示了用 PI 控制被控设备可以以最快的速度操作。

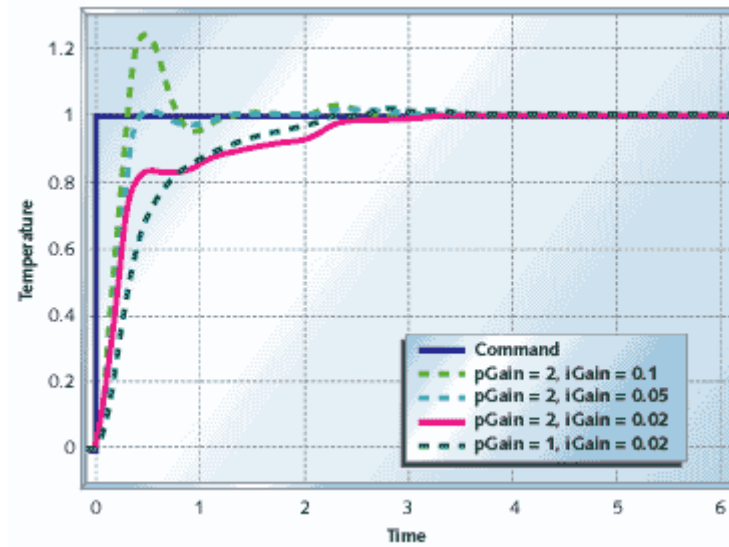


图14: PI控制的加热系统
time=2 时外部干扰导致波动

在讨论完积分控制之前，有两点我要强调，第一，随着误差的累积，系统采样的时间变得很重要。第二，必须注意积分器的范围避免积分饱和。

积分状态改变率与平均误差、积分增益、采样率三者之积相等，由于积分器趋向于平稳长期的事情，那么就可以忽略有点不平坦的采样率，但需要平均出一常量。在最坏的情况下，采样率在 10 次采样样本中不能超过上下 20%，你甚至可以丢弃一些在平均采样率范围内的采样率。不过对于一个 PI 控制，我觉得每一个采样率在正确采样时间内达到上下 1%到上下 5%之间，有一个长期平均采样率是最好。

如果你的控制器需要努力推动被控设备，你的控制器输出会要花费超出驱动器可接受的大量时间，这种情况称为饱和。如果你采用 PI 控制器，当所有时间都花费在饱和情况下时，积分状态会增长（饱和）成一个非常大的值。当被控设备到达目标值时，积分器的值仍然非常大，被控设备驱动会超出目标值，直到积分状态减小，然后系统就逆转。这种情况下系统不能达到目标位置，但会沿着目标位置慢慢振荡。

图 15 显示了积分饱和的效果，我采用图 13 的电机控制器，把电机驱动限制在上下 0.2%，不仅控制器输出比驱动器提供给电机的（输出）要大很多，而且电机还发生了严重的过冲。电机在 5 秒左右达到目标，但它直到第 8 秒的时候才改变方向，最后直到第 15 秒的时候到达目标位置。

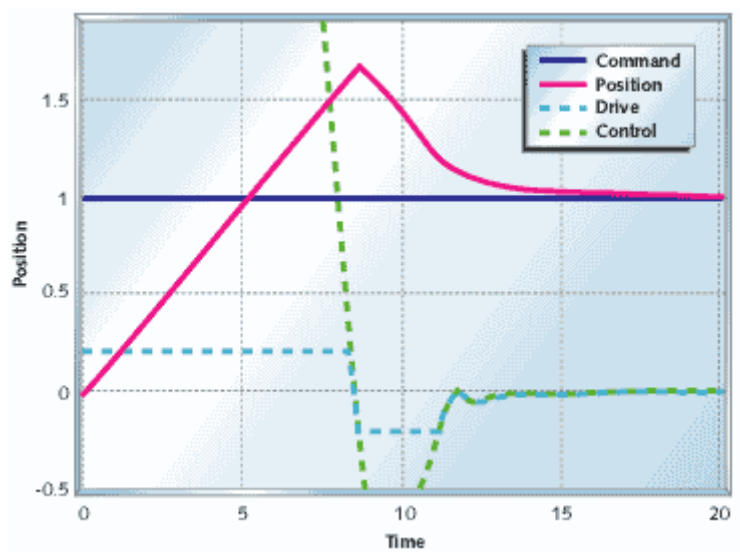


图 15: PI 控制的电机齿轮饱和 (windup)

最简单最直接地处理积分器饱和的方法就是限制积分状态，在前面的代码里有举例。图 16 显示限制图 15 中的积分项到可用的驱动输出。控制器输出仍然很大（由于比例项的原因），但是积分器不再有很大的饱和，系统从开始到达目标值用了 5 秒的时间，最后完成用了 6 秒左右。

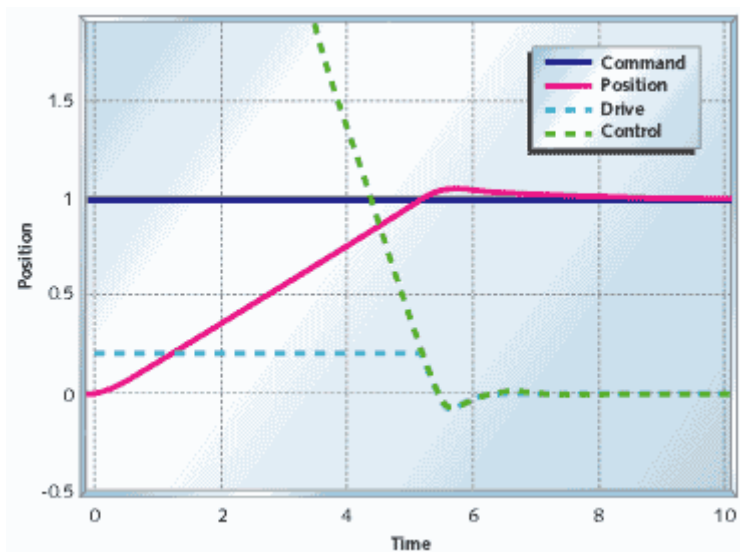


图 16: 积分限制 PI 控制的电机和齿轮

注意在举例代码中，你不论什么时候改变了积分增益，都得重新设置 `iMax` 和 `iMin` 的值。通常你可以设置积分器最高和最低值，使积分器输出相匹配的驱动器达到最低和最高。如果你知道外部干扰很少，且你需要更快地设定，你可以进一步限制积分器。

微分项

上一节中我没有举例精密驱动器，是因为 PI 控制的精密驱动器会变得不稳

定。通常，比例控制的被控制设备不稳定的话，PI 控制的也会不稳定。我们知道比例控制是获取被控设备的当前状态来处理，积分控制是获取被控设备的过去状态来处理。如果我们有一项能预测被控设备的将来状态，那会使被控制设备更加稳定。微分项就可以做这个工作。

下面的代码是 PID 控制器的微分项。我喜欢用实际的被控设备位置，还不是用误差，是因为当输入命令改变时，可以很方便的操作。微分项本身是前一次被控设备位置值减去当前位置值，这样就提供了一个粗略估计的速度值（改变的位置/采样时间），从而预测下一个位置值。

```
double dTerm;  
...  
...  
dTerm = pid->dGain * (position - pid->dState);  
pid->dState = position;  
...
```

微分控制可以使精密驱动器变得稳定，图 17 表示比例微分控制（PD）精密驱动器的响应，采用比例微分控制比采用其它方式控制整定所花的时间要少很多倍，图中所示只花了不到 1/2 秒。图 18 是 PID 控制的加热器，你可以看到采用完全 PID 控制时，性能有所提升。

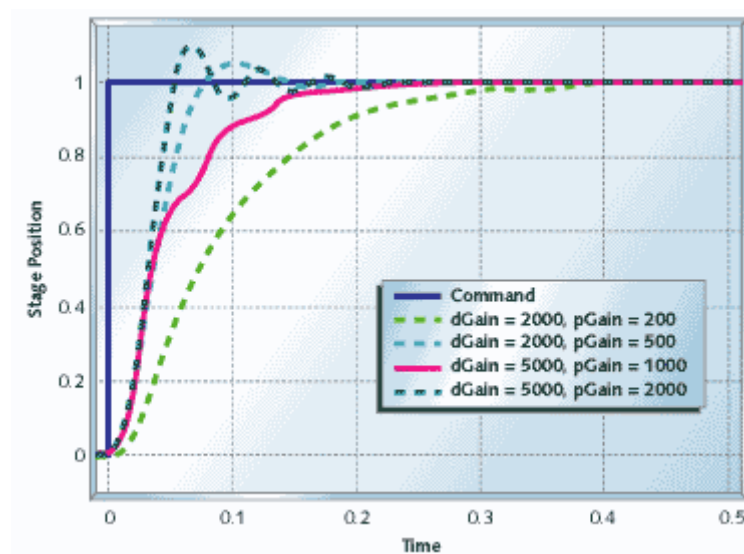


图 17：PID 控制的精密驱动器

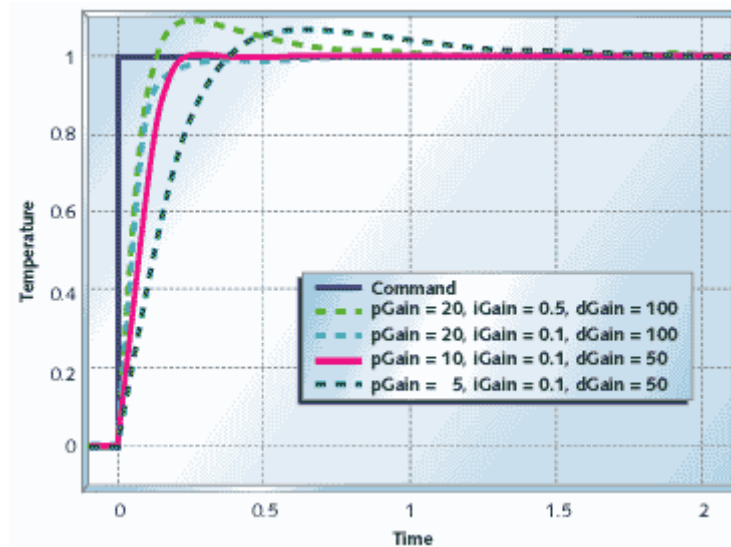


图 18: PID 控制的加热器

微分控制非常强大，但它也是最容易出问题的控制类型。不走运的话你很可能碰到这三个问题，无规律采样，噪声，高频振荡。我在给出微分代码的时候，发现控制输出正比于位置改变和采样时间。当位置改变是恒速的，但你采样时间变化，微分项就会产生噪声。由于微分增益通常就是个很大的值，那么噪声也会被放到很大。

采用微分控制的时候，你必须密切关注采样。我说过你要保证采样间隔一致，在任何时候不超出 1% 的范围内，当然越接近越好。如果硬件做不到这么好的采样间隔，那么就要在软件里保持很高的优先级。不需要精密执行控制器，只需要保证采样时刻准确。最好是把采样放在中断服务程序里，或者高优先级的任务中，执行代码放在优先级不高的代码中。

微分控制伴有噪声问题，噪声通常广泛分布在整个频谱里面，控制命令的输入和被控设备输出，通常在低频率范围。比例控制不受噪声干扰，积分控制平均了输入值可以干掉噪声，微分控制增强了高频信号，同时也增强了噪声。可以看我上面设置的微分增益，在每个采样点上引入一点点不同的噪声，然后再将微分增益设置成 2000，会是什么结果？

可以在微分输出后面加一级低通滤波器以减少噪声，但这会严重影响它的性能。关于如何去做低通滤波器、怎样做可以确保它能工作这一理论已经超出本文讨论的范围。这一问题主要看你预见所有噪声的可能性有多大、得到纯净输入所能付出的代价是多少，还有就是你用微分控制获得高性能的迫切性。你能解决这些问题，你就完全可以不用微分控制，要你的硬件伙计给你低噪声的输入，或者寻找一位控制系统专家。

PID 控制器完整代码如下 Listing 1，同时在 www.embedded.com/code.html 可以获得。

Listing 1: PID 控制器代码

```
typedef struct
{
    double dState; // Last position input
    double iState; // Integrator state
    double iMax, iMin; // Maximum and minimum allowable integrator state
    double iGain, // integral gain
    pGain, // proportional gain
    dGain; // derivative gain
} SPid;

double UpdatePID(SPid * pid, double error, double position)
{
    double pTerm,
    dTerm, iTerm;
    // calculate the proportional term
    pTerm = pid->pGain * error;
    // calculate the integral state with appropriate limiting
    pid->iState += error;
    if (pid->iState > pid->iMax) pid->iState = pid->iMax;
    else if (pid->iState < pid->iMin) pid->iState = pid->iMin;
    iTerm = pid->iGain * iState; // calculate the integral term
    dTerm = pid->dGain * (position - pid->dState);
    pid->dState = position;
    return pTerm + iTerm - dTerm;
}
```

整定

整定一个 PID 控制器是个好事，这是因为你不要对正式控制理论深入了解都可以来做这个美差事。世界上 90% 的 PID 控制的闭环控制系统工作得相当好只是因为整定得相当好。

如果可以，你可以把系统接上测试设备来观察信号，可以写些调试代码观察相应的变量。如果系统运行速度不快你可以将相当的变量从串口发出来，并用电子表格记录。最好是能够看到控制驱动输出和被控设备输出，另外，最好也能够送给系统命令输入一序列方波信号，写适应的测试命令代码是很容易的。准备好以后，你可以将所有的增益值设为 0，如果你认为不需要微分控制（像电机和齿轮或者恒温系统）可以直接跳到下面讲比例增益整定那一节，需要微分控制那么从调整微分增益开始。

控制器不能只用微分控制。将比例增益设一个比较小的值（1 或者更小），

检查系统是否能工作，如果在这一比例增益下系统振荡，你可以用微分增益来修正。将微分增益值设置成比例增益值的 100 倍，然后观察驱动信号，现在开始增加微分增益，直到你看到振荡、噪音过高、或过度（超过 50%）的驱动器或被控设备过冲输出，要注意的是，过多微分增益造成的振荡比不足微分增益时的振荡速度要快得多。我喜欢将增益值增加到系统濒临振荡的时候，然后再将增益值减去 2 或者 4。确保驱动信号仍然很好，这时你的系统可能反应很慢，可以开始整定比例和积分增益。

如果比例增益值没设，设值比例增益值在 1 到 100 范围内，系统可能会要么性能严重下降，要么振荡。如果你看到振荡，以 8 或者 10 为单位降低比例增益值直到振荡停止；如果你没看到有振荡，以 8 或者 10 为单位增加比例增益值直到你看到振荡或者过多的过冲。就像微分控制，我通常调整到过多的过冲时候，再以 2 或者 4 为单位来减少微分增益。调整得差不多了的时候，你可以以 2 为单位微调比例增益值，直到看到你想要的。

比例增益设置好了之后，开始增加积分增益值，积分增益起始值范围在 0.0001 到 0.01 范围内，在这里，你需要找到让你系统即有合理的性能，但又不会有太多过冲，不会太接近振荡的积分增益值的范围。

其它问题

除非你的系统需要非常精确的性能参数，你通常可以以 2 为单位来调整控制增益来得到一个相对正确的输出值。这意味着你可以在软件里用位移来代替乘法，这对于采用低性能的处理器的来说非常有用。

采样率

到目前为止，我只讨论了在各项中什么样的采样率符合，没有告诉你怎么去提前决定什么样的采样率符合。如果你的采样率太低，由于采样延迟的累积，你可能会达不到你想要的性能，如果采样率过高，可能会在微分项中产生噪声，积分项中造成溢出。

数字控制系统的经验法则是采样时间应为稳定时间的 1/10 到 1/100 之间，系统稳定时间是指从驱动输出饱和和到控制系统有效稳定这一段时间。回看图 16，控制器在 5.2s 的时候输出饱和，在 6.2s 左右稳定。如果你能接受 1 秒的稳定时间，那么你可以把采样率设置为 10Hz 以下。

采样率应该能灵活设置，因为任何比较困难的控制情况下会需要提高采样率，事实上像要控制一个难控制的被控设备，或者需要微分控制，或者需要非常高精度控制都要提高采样率。然而，如果你的控制情况很简单，你可以稍微降低采样率（我会毫不犹豫地延长采样时间，超过需要的稳定时间的五分之一）。如果你不需要微分控制，而且有足够位来处理你的积分项，那你可以把采样率设为比预期稳定时间快 1000 倍。

Exert control

This covers the basics of implementing and tuning PID controllers. With this information, you should be able to attack the next control problem that comes your way and get it under control.

Tim Wescott has a master's degree in electrical engineering and has been working in industry for more than a decade. His experience has included a number of control loops closed in software using 8- to 32-bit microprocessors, DSPs, assembly language, C, and C++. He is currently involved in control systems design at FLIR Systems where he specifies mechanical, electrical, and software requirements and does electrical and software design. You can contact him at tim@wescottdesign.com.