# 硬件控制算法

可应用的电路系统一般由模拟电路和数字电路组成。模拟电路用于电信号的处理和提供电源；数字电路则用于采集非电信号数据和控制系统。一般来说系统控制使用MCU、DSP或FPGA通过软件编程实现，在其中常使用用于控制系统的各种算法。

## PID算法

### PID算法简介

PID算法是一种很成熟、应用广泛的连续时间控制系统算法，最突出优点在于：

* 结构典型
* 方便参数整定
* 可以灵活更改结构

除了数字PID，还存在使用电阻、电容反馈与集成运放配合实现的模拟PID，但是因为其不便于修改和调参，适用范围不广

PID是一种典型的闭环控制算法，使用PID控制器*D(s)*来完成控制，该控制器读取输出量*y(t)*和给定量*r(t)*，计算得到二者之间的误差*e(t)=r(t)-t(t)*，并获取其比例、积分、微分的线性组合来构成控制量*u(t)*，通过改变调节器参数*Kp，Ki，Kd*来实现控制，计算公式如下

或者可以采用变形公式

其中**三个参数P、I、D分别表示比例、积分、微分**

实际应用中最常使用PI甚至只使用P参数来进行控制。因为D参数常常会引入不确定性，增大调参的难度。

三个参数的特性如下：

* 比例调节参数KP
* 按比例反映系统的误差，一旦系统出现偏差比例调节就会进行，**比例调节是主要的控制部分**。但是过大的比例会使系统稳定性下降，系统反应更灵敏、速度加快、稳态误差更小，但是震荡次数会显著增多，调节时间也会加长；过小的比例则会导致系统很难回到稳态
* 积分调节参数KI
* 这个参数本质上是通过给定量和输出量之间稳态值的差来消除系统稳态误差，**用于提高系统的控制精度**。但过高的调节参数会导致系统稳定性下降，动态响应变慢，超调增大，该参数一般不单独作用，至少需要和P参数共用
* 微分调节参数KD
* 用于反映系统偏差信号的变化率，可以一定程度上预测偏差的变化趋势，**起到超前控制作用**。但是该参数难以整定，且过强的微分调节会使系统剧烈震荡，且对噪声干扰有放大作用，对抗干扰不利

### 数字PID控制

数字控制系统大多数是采样数据控制系统，进入系统的连续时间信号必须经过采样和量化后转换为数字量才能进行相应的计算和处理，所以其中的参数需要使用数字计算去逼近，当采样周期短时可以使用求和代替积分，使用差商代替微商，**将描述连续时间的PID算法微分方程变成描述离散时间的差分方程**

实际使用时有两种思路：

1. 位置式PID

* 差分方程为
* 此式即数字PID算法的非递推形式——全量算法，为控制量基准值，n=0时将会采用该值；是第n个采样时刻的控制量；为采样周期，在算法中为了实现求和，必须存储系统偏差的全部值，该式求得全量输出是控制量的绝对数值，决定了执行机构在控制系统中的位置
* 因此称这种算法为位置算法

1. 增量式PID

* 当驱动步进电机这样需要控制量的增量的执行机构时，需要用增量算法，差分方程为
* 这时的输出值能体现各次误差量对控制作用的影响，计算时只需要存储最近的三个误差采样值

### PID算法的饱和特性

实际的控制系统存在这样的特性：当控制变量达到一定值后，系统的输出变量便不再增长，系统进入饱和区，称为**饱和特性**，这就要求系统的控制变量必须限制在某个范围之内。有时对于控制量的变化率也有限制，为了能让控制系统更加稳定，需要对位置算法进行改进；虽然增量算法中没有累加和项，不会出现积分饱和，但是可能出现比例和微分饱和，需要使用积累补偿法，将因为饱和而未能执行的控制增量信息累积起来，一旦可能时再补充执行

位置算法的饱和主要由积分项引起，称为**积分饱和**，可以使用三种方法必免积分积累过大

1. 积分分离法

* 在误差量较大时不进行积分，直到误差在一定值之内时才在控制量的计算中加入积分累积，差分方程为
* 其中Ki在时为1，在时为0，以作为门限值可以有效削减积分饱和

1. 遇限削弱积分法

* 当**控制进入饱和区后便不再进行积分项的累加**，只执行削弱积分的运算，使用该算法可以避免控制量长时间停留在饱和区

1. 有效偏差法

* 将**实际执行的控制量对应的误差值**作为有效误差值进行积分累加，而不是使用实际的误差值进行积分累加

### PID调参（参数整定）

1. 确定PID控制器结构
   * 控制结果比较宽松且要求稳定性的控制系统：P或PD控制器
   * 必须消除稳态误差且要求精度的控制系统：PI或PID控制器
   * 存在滞后的对象：PI控制器或PID控制器

* 一般情况下，P、PI、PID控制器应用较多，ID控制器可以说根本不用

1. 调参

* 参数选择要根据受控对象的具体特性和对控制系统的性能要求进行，工程上一般要求整个闭环系统是稳定的，对给定量的变化能迅速响应并平滑跟踪，超调量小；要求抗干扰性较高；总体上遵循解决主要问题，统筹兼顾其他方面，适量折中的思路
* PID三个参数一般通过玄学调参或者经验公式确定
* 采样周期选择可以通过以下方面进行确定：
  1. 香农采样定理：（ω为被采样信号的上限角频率），用于确定采样周期的上限
  2. 有跟踪要求的闭环系统要适量减小采样周期
  3. 要求采样周期有一定宽度来确定计算精度，减少系统算力成本
  4. 通过经验公式和玄学调参来确定
* PID参数的基本调整方法如下：
  1. 比例项逐次独立实验，取其中反应最快、超调最小的曲线
  2. 加入积分项，将之前选择的比例系数减小为50%到80%，再将积分时间置为较大值，观测响应曲线，再逐次减小积分时间，选择其中响应震荡次数较少、能快速趋近结果的参数
  3. 加入微分项减小超调和震荡，先设置TD=0，再逐渐加大微分项，在此基础上维持比例项80%到120%调节范围，积分项50%到150%调节范围
  4. 如果出现玄学问题可以直接随机调参，运气是实力的一部分

1. 特殊调参方法
   1. 选足够短的采样周期，先使用P控制器，逐渐加大比例放大系数逐次实验直到系统对输入的阶跃信号响应出现稳定边缘的临界震荡，将此时的比例放大系数记为，临界震荡周期记为
   2. 以连续时间PID控制器为基准，使用误差平方积分作为评价函数，控制度
   * 保证其必定大于1的前提下，选定一个合适的控制度
   1. 查经验公式表获取对应参数
   2. 实际检验参数并进行微调

## 运动调节算法

### 大林算法

许多工程中会遇到纯滞后调节系统，要求系统具有以下特征：

* 滞后时间长
* 系统动态特性和稳定性高
* 没有或很少超调量

一般使用大林算法处理这类问题。该算法由IBM的大林（Dahlin）设计，专用于处理带有纯滞后的一阶或二阶惯性环节，其传递函数分别为

或

其中τ1和2都是对象的时间常数，θ=Nτ是对象的纯滞后时间，N为正整数，K为对象的放大倍数

该算法适合对ADC、DAC等需要一定时间的转换器或带有硬件运算电路的设备进行控制，使得整个闭环系统的传递参数为带有纯滞后时间的一阶惯性环节，即

算法数字控制器的最终表达式：

U(k)为k时刻的输出值，E(k)是k时刻的误差值，E(k-1)是k-1时刻的误差值，U(k-N-1)是k-N-1时刻的输出值

该算法调参依旧很玄学，但是没有PID那么玄学，只要根据系统飞升曲线确定对象的纯滞后时间θ和系统时间常数τ，微调选取采样周期再不断调整τ值一般就能获得理想效果

该算法最大的优势就是消除了由于超调引起的系统不稳定因素

### 模糊控制算法

当系统影响因素极度复杂，难以建立数学模型时，需要使用模糊控制算法。这是一种鲁棒性强、抗干扰、抗参数变化的非线性控制算法，适用于非线性、动态特征不易掌握及纯滞后系统的控制。模糊控制不受数学模型的束缚，采用模糊控制表，在调试过程中反复经过人工修正，建立模糊控制算法、确立模糊规则是设计模糊控制系统中最重要的环节。模糊控制算法的问题在于稳态精度低、易饱和、适应能力有限、没有形成完整的理论体系。

这一算法需要以下步骤实现建模：

1. 归纳实验经验：将经验公式归结为一组条件语句，称为**模糊控制规则**或模糊控制模型
2. 将误差e和误差变化率de/dt进行模糊化处理：将输入变量映射到一个合适的响应领域或模糊集合的标识符

* 本人接盘的一个项目代码里存在以下代码（关键数据已经模糊替换处理）
* if (condition == 0)  
   {  
   u8 AFlag = (RotateNum > StraightNum);  
   u8 BFlag = ((RotateNum + StraightNum) > Magicnumber);  
   u8 CFlag = (RotateNum > Magicnumber);  
   u8 DFlag = (StraightNum > Magicnumber);  
   u8 SFlag = AFlag \* Magicnumber + BFlag \* Magicnumber + CFlag \* Magicnumber + DFlag;  
   switch (condition)  
   {  
   case Magicnumber:  
   case Magicnumber:  
   \*MoveThruster[0] = (u32) (RotateNum);  
   \*MoveThruster[1] = (u32) (RotateNum);  
   \*MoveThruster[2] = (u32) ((Magicnumber - RotateNum + StraightNum));  
   \*MoveThruster[3] = (u32) ((Magicnumber - RotateNum + StraightNum));  
   break;  
   case Magicnumber:  
   case Magicnumber:  
   \*MoveThruster[0] = (u32) (StraightNum);  
   \*MoveThruster[1] = (u32) (StraightNum);  
   \*MoveThruster[2] = (u32) ((Magicnumber - RotateNum + StraightNum));  
   \*MoveThruster[3] = (u32) ((Magicnumber - RotateNum + StraightNum));  
   break;  
   case Magicnumber:  
   case Magicnumber:  
   \*MoveThruster[0] = (u32) ((RotateNum + StraightNum - Magicnumber));  
   \*MoveThruster[1] = (u32) ((RotateNum + StraightNum - Magicnumber));  
   \*MoveThruster[2] = (u32) (StraightNum);  
   \*MoveThruster[3] = (u32) (StraightNum);  
   break;  
   case Magicnumber:  
   case Magicnumber:  
   \*MoveThruster[0] = (u32) ((RotateNum + StraightNum - Magicnumber));  
   \*MoveThruster[1] = (u32) ((RotateNum + StraightNum - Magicnumber));  
   \*MoveThruster[2] = (u32) ((Magicnumber - RotateNum));  
   \*MoveThruster[3] = (u32) ((Magicnumber - RotateNum));  
   break;  
   }  
   \*MoveThruster[4] = (u32) (VerticalNum);  
   \*MoveThruster[5] = (u32) (VerticalNum);  
   }  
   else if (condition == 1)  
   {  
   \*MoveThruster[0] = (u32) (\*PDRotate);  
   \*MoveThruster[1] = (u32) ((Magicnumber - \*PDRotate));  
   \*MoveThruster[2] = (u32) (\*PDRotate);  
   \*MoveThruster[3] = (u32) (Magicnumber - \*PDRotate);  
   \*MoveThruster[4] = (u32) (VerticalNum);  
   \*MoveThruster[5] = (u32) (VerticalNum);  
   }
* 就是一个模糊控制算法的例子，其中的Magicnumber都是经过大量实验确定的控制数据。使用python自带的map函数可以很好地对数据进行映射。

1. 应用模糊算法：根据控制规则计算出模糊控制量C，之后通过多次实验校准模糊控制规则
2. 构造基本模糊控制器查询表：将之前的一整套控制方法写成库或表

其中确定控制数据的步骤最为关键，称为**解模糊化**

### 神经网络算法

自动调准控制模型，适应性极强，但算法较复杂，需求算力高，一般的MCU难以实现，一般使用专门的硬件加速单元配合高性能嵌入式处理器实现

算法原理比较复杂，我一个辣鸡电工的也不是很懂（悲）

## 运动姿态控制算法

Bresenhan算法是常用的图形扫描算法，只使用整数加法和乘2运算即可实现，可以在低算力水平的MCU中搭载，实现控制物体按规律运动的效果

### 产生线段的整数Bresenham算法

直线方程，在现实中位于(x0,y0)的物体运动可能有多种趋势，但总体上可以分为以下两种：

* 靠近(x0+1,y0)
* 靠近(x0+1,y0+1)

根据这两种趋势分别计算他们到直线的距离d1、d2，在求出两距离之差

根据Δd>0，直线上点离(x0+1,y0+1)较近；Δd<0，直线上点离(x0+1,y0)较近，再用该式乘Δx即可得到一般的判别式

$$p\_1=2\Delta y-\Delta x，x\_{i+1}=x\_i+1 \\
y\_{i+1}=y\_i +1,p\_{i+1}=p\_i +2(\Delta y-\Delta x)，当p\_i\ge 0 \\
y\_{i+1}=y\_i,p\_{i+1}=p\_i +2\Delta y，当p\_i< 0$$

只需将原公式适当修正，用、替换、就能轻易得到向后方运动的线段

### 产生圆的整数Bresenham算法

对于圆来说需要用尽可能多的点表示出控制对象需要经过的轨迹点，Bresenham算法也有与上面求直线类似的公式

$$p\_i=2(x\_i+1)^2 +2y\_i^2 -2y\_i -2R^2 +1 \\
x\_{i+1}=x\_i +1 \\
y\_{i+1}=y\_i,p\_{i+1}=p\_i+4x\_i+6，若p\_i<0 \\
y\_{i+1}=y\_i-1,p\_{i+1}=p\_i +4(x\_i-y\_i)+10 ，若p\_i\ge 0$$

将圆分成八个方位，将上式镜像对称即可获得其他七个方位的公式，显而易见计算它需要消耗大量算力资源有两种解决方案：

1. 使用RAM存储前1/8的坐标数据，然后通过镜像对称求出剩下的1/8坐标，然后调整顺序输出。这种方法可以节省算力，但是需要耗费大量RAM空间，可能还需要扩展片外RAM
2. 按顺时针求出8组1/8圆的Bresenham算法表达式，在接下来的控制过程中依次切换使用。这种方法不需要额外扩充RAM，但是需要花费经历处理控制-运算衔接问题。在一些性能足够的MCU上可以使用RTOS来减少需要的处理精力，因此这种方法可以有效平衡硬件和需求

## 数字滤波

数字滤波指使用数值运算达到改变输入信号中所含频率分量的相对比例，或滤除某些频率分量的目的。常用于处理坏点数据、对信号进行平滑处理、消除毛刺等用途

最简单的滤波其实就是按钮软件消抖

### 限幅滤波

又叫程序判断滤波，根据多次采集到的数据，如果当前采集值和前一次采集值维持在一定偏差ΔD之内，则将每次采集到的数据和前一次数据进行比较，如果它们差的绝对值小于ΔD，则本次采集到的数据有效，否则舍弃

这种滤波器可以克服偶然因素引入的脉冲干扰和波形上的尖峰毛刺，但是难以抑制周期性干扰且对于波形处理的平滑度较差，只能算作一种最简单的基本滤波

### 中值滤波

将原来的采样间隔进行细分，在原有基础上采样N次，然后把N次采样值按照大小排序，取中位数为本次采样值。

能够有效克服偶然因素带来的干扰，对于变化缓慢的被测参数有良好的滤波效果，但对于快速变化的信号则不太适用。算法实现方面，一般采用冒泡排序、选择排序、快速排序等算法，由于引入了排序算法，该方法不能处理速度要求很高的信号，算法的运算速度和占用RAM直接受所选择的N值决定

### 算术平均滤波

和中值滤波的实现思路类似，但是需要取N次采样的算术平均值。

该算法难以对高速信号使用（除非搭配死贵的高速FPGA和烦到爆炸的等长走线进行硬件级别的算法滤波）

### 递推平均滤波

又称为滑动平均滤波，将连续的N个采样值设为一个FIFO（先入先出队列），队长就为N，将队列中的N个数据进行算术平均滤波

这种方法对于周期性干扰具有非常好的抑制作用，具有很高的平滑度，但是灵敏度较低，对于偶然出现的脉冲干扰抑制作用较差，不适用于脉冲干扰较严重的场合。

该算法原理也可以应用在**软件陷波器**上。取N=S/f。其中S是美妙的采样次数，即采样率；f是要消除的谐波的频率，S、N都要取整数，这样最终就能实现对于f频率谐波的定向消除（采样平均值Y=Σ1/N，最终获得Y-C）

### 中值平均滤波

选取N个采样，去掉其中最大值和最小值，再进行算术平均滤波，融合了中值滤波和平均滤波的特点，但也继承了二者的缺点

### 限幅平均滤波

先限幅滤波，再进行算术平均滤波

可以有效抑制偶然出现的脉冲干扰并消除采样偏差，但仍旧不适合高速信号处理

### 一阶滞后滤波

取用一个比例常数0<k<1，使用以下公式计算本次输出结果：

其中Tn为本次采样值，Tn-1为上次输出值

该方法类似PID，但是会造成相位滞后，灵敏度较低，难以消除频率高于采样频率一半的干扰信号

不过对于周期性干扰具有良好的抑制作用且运算量不大，适用于频率较高、相位要求不高的场合

### 加权递推平均滤波

赋予不同时刻的FIFO采样值不同的权重，在对其进行递推平均滤波

适合在有较大纯滞后时间常数的对象和采样周期短的情况；难以反映变化较慢、采样周期长的情况，且占用RAM较多

### 消抖滤波

又称为数字消抖，通过设置一个滤波计数器，将每次采样值与当前有效值比较，如果采样值和有效值相等则清零，否则计数器+1并判断计数器是否达到上限，如果计数器溢出则将本次值替换为当前有效值，再清零计数器

常用于对变化缓慢的被测参数进行滤波，可避免系统在临界值附近时的波动跳变，但并不适用于快速变化的参数测量控制

### 限幅消抖滤波

先使用限幅滤波，再进行消抖滤波

避免将干扰值导入系统，但不适用于快速变化的参数

### 复杂数字滤波

卡尔曼滤波、IIR滤波、高阶滤波等都需要复杂的运算过程，经常需要使用浮点运算，因此需要搭载的MCU具有一定的算力才能使用