扩展点5——使用PID控制舵机

知识笔记

1. PID的专业解释

PID是比例-积分-微分（Proportional-Integral-Derivative）的缩写，它是一种在控制系统中广泛应用的反馈控制算法。PID控制算法通过计算误差、调整比例、积分和微分项，实现快速、准确的控制，以调节系统的输出达到期望值。

2. PID的通俗解释

PID在控制时考虑了三个部分：**当前误差**，**过去误差**，**误差的变化趋势**，通过这三个来不断修正以更优地达到目标状态。

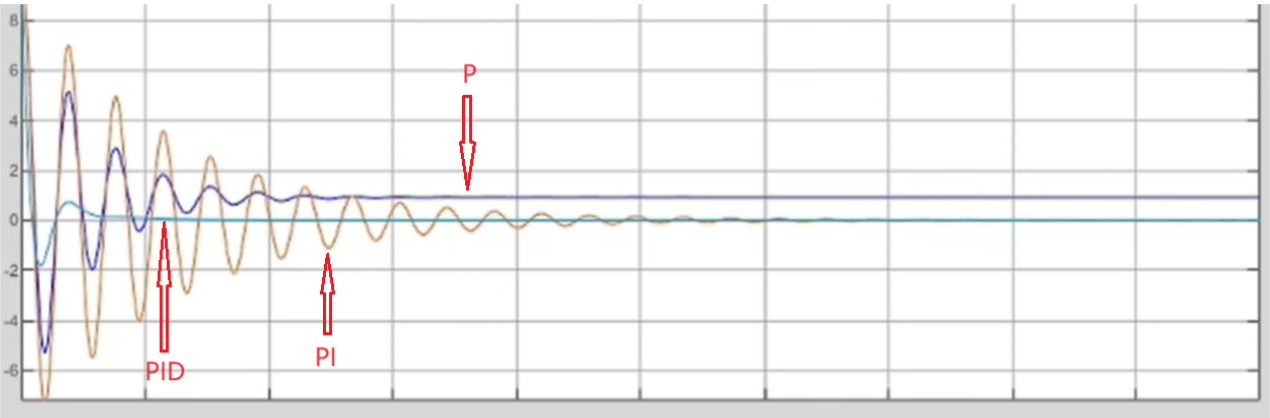
- 当前误差（比例）：少了就加，多了就减——“快”
- 过去误差（积分）：一直加不够，就多加点，一直减不下，就多减点——“准”
- 未来误差（微分）：加或减的过快，就阻碍加或减——“稳”

> 一直加不够，一直减不下的原因：稳态误差现象，如果干扰和比例项抵消，就会在不准确的状态维持稳定。

3. PID的特点

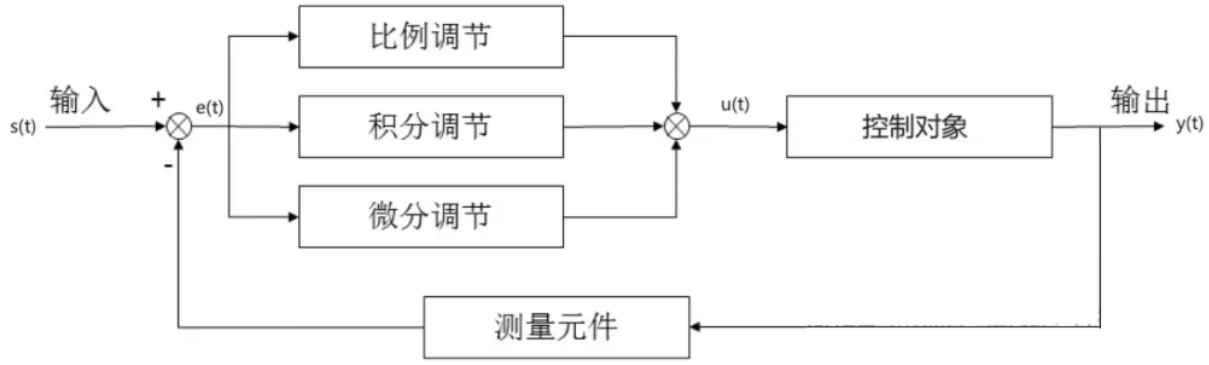
- PD控制：优点：提高稳定性，改善瞬态响应；缺点：有稳态误差，对高频噪音敏感
- PI控制：优点：改善系统的稳态误差；缺点：振荡更大，稳定时间长
- PID控制：PID兼具两者的优点；缺点：对高频噪音敏感

4. 直观图像初识PID PID的波动曲线也印证了P，PI，PID的特点



5. PID模型

PID模型构成了一个不断重复的反馈控制环。整个流程的总体思路是：1. 先计算出每次设定值与实际输出值之间的误差e(t)； 2. 对其进行相应的比例 - 积分 - 微分运算，对各项求和后得到一个相关的控制输出u(t)； 3. 将该值传递给控制对象完成相应调整，调整后经过传感器再次采样，再次与设定值进行误差计算；4. 不断反馈，不断调整。



举个例子理解PID模型：浴缸盛洗澡水时控制水温（实质是通过e调节u）
想要达到的水温是输入，即 s（t）
目前水温是输出，即 y（t）
输入输出的差距是误差，即 e（t）
根据温差扭动冷热旋钮是控制，即 u（t）

- 比例调节：水凉了就加热水，凉得越多，加得越热
- 积分调节： 一直都不够热（热量会散失到空气中），就加更热的水来抵消
- 微分调节：水温变热得太快了，热水调小一点，避免过热还反过来加冷水
- 控制对象：旋钮，映射到不同的水温
- 测量元件：水温计

6. PID核心公式

$$u(t)=K_{\mathrm{p}}\left[e(t)+\frac{1}{T_{\mathrm{i}}}\int_0^t e(t)\mathrm{d}t+T_{\mathrm{d}}\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right]$$

- $u(t)$ 是控制器在时间 t 输出的控制信号（即控制器的输出）。
- K_p 是比例系数（也称为比例增益），决定当前误差对控制信号的影响程度。
- $e(t)$ 是系统的输出与期望值之间的误差。
- $\int e(t)dt$ 是误差的积分，表示从初始时间到当前时间误差的累积。
- $d[e(t)]/dt$ 是误差的导数，表示误差随时间的变化率。
- T_i 是积分时间。
- T_d 是微分时间。

将上式括号展开，并令$K_i = K_p/T_i$, $K_d = K_pT_d$, 可以得到：

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t)\,dt + K_d \cdot \frac{de(t)}{dt}$$

等式右边三项分别为控制输出量的比例项，积分项，微分项

- K_i 是积分系数（也称为积分增益），它通过对误差的累积来调节控制信号，主要用于消除系统的稳态误差。
- K_d 是微分系数（也称为微分增益），它根据误差的变化率来调节控制信号，主要用于提高系统的动态响应速度和稳定性，在误差变大前就会给出一个控制量。

由于积分是连续的，为了应用到工程中，需要将上式离散化。
步骤1：以多个采样时刻点代替连续时间
步骤2：积分由多个采样时刻对应的值的和表示
步骤3：微分由相邻时刻采样的增量表示
步骤4：输入控制量由采样误差值表示
即作以下变换（T为采样周期，k为采样次数，k-1表示上一次采样）：

$$t \approx kT$$

$$\int_0^t e(t)\,\mathrm{d}t \approx T\sum_{j=0}^{k} e(jT)$$

$$\frac{\mathrm{d}e(t)}{\mathrm{d}t} \approx \frac{e(kT)-e\big((k-1)T\big)}{T}$$

可以得到离散化的PID表达式（**PID位置式**）：

$$u(k) = K_{\mathrm{p}}e(k) + K_{\mathrm{i}}T\sum_{j=0}^{k} e(j) + K_{\mathrm{d}}\frac{e(k)-e\big((k-1)\big)}{T}$$

但是，要累加0-k时刻的所有误差值还是稍显麻烦，我们还可以通过上一次的控制量加增量确定当前控制量的方式简化。计算u（k）- u（k-1）可以得到（**PID增量式**）：

$$\Delta u(k) = K_{\mathrm{p}}\big(e(k)-e(k-1)\big) + K_{\mathrm{i}}Te(k) + K_{\mathrm{d}}\frac{e(k)-2e(k-1)+e\big((k-2)\big)}{T}$$

要计算u（k），求u（k-1）+Δu（k）即可。
至此得到了利于编程的两种离散化的PID公式。

7. PID位置式与PID增量式的比较

- 位置式与增量式区别：
  1. 误差的计算
  - 位置式算法要用到过去误差的累计。
  - 增量式算法不需要做累加，控制量增量的确定仅与最近三次误差采样值有关。
  2. 输出的结果
  - 位置式的输出直接对应控制量的输出。
  - 增量式的输出是控制量的增量。
  3. 限幅操作
  - 位置式需要有积分限幅和输出限幅。
  - 增量式只需输出限幅。
- 位置式PID优缺点：
  1. 优点：
  - 响应速度快，适合需要快速调节的场景。
  - 实时性强，适合实时控制要求高的系统，如飞行器姿态控制。
  2. 缺点：

    - 每次输出均与过去的状态有关，计算时要对e(k)进行累加，运算工作量大。
- 增量式PID优缺点：
    1. 优点：
    - 只输出增量部分，故稳定性好，抗干扰能力强
    - 控制增量Δu(k)的确定仅与最近三次的采样值有关，故运算负担小。
    2. 缺点：
    - 响应速度相对较慢。需要更多历史信息进行计算，可能导致延迟。

> 注意：运算负担小和需要更多历史信息进行计算不冲突。位置式累积的多，但每次只需要本次的误差；增量式不用累积，但需要最近三次的误差数据。

实现步骤

> **疑惑**：舵机没有向外的输出，即很难获取舵机的实时位置。对照PID模型，也就是缺少了“测量元件部分”，无法重复计算下一采样时间的更新误差，所以这个“PID系统”没有反馈，根本不闭环，**不满足PID控制器的基本条件。**
> 查阅资料后发现实际上舵机内部已经闭环，而且就是用PID算法控制的！？
> 所以舵机只需要给角度参数即可，但是PID可以对直流电机，无刷电机等进行控制。

所以在此只单独讨论PID控制器的实现步骤和代码，并未应用到控制舵机的情景中。

1. PID.h文件定义PID参量结构体，PID.c文件初始化赋初值

```
#define PID_LIMIT_MIN -10000         //PID输出最低值
#define PID_LIMIT_MAX 10000          //PID输出最大值

typedef struct
{
    float Kp;         //比例系数Proportional
    float Ki;         //积分系数Integral
    float Kd;         //微分系数Derivative
//float Ti; //积分时间常数
//float Td; //微分时间常数
//float T;  //采样周期
    float ek;               //当前误差
    float ek_prev;          //前一次误差e(k-1)
    float ek_prev_prev;     //再前一次误差e(k-2)
    float location_sum;     //位置式PID累计积分
    float out;           //PID输出值
}PID_LocTypeDef;
```

```
void PID_Init(PID_LocTypeDef *PID)
{
    Kp = ···;
    Ki = ···;
    ···
}
```

说明：注释掉的积分时间常数Ti; 微分时间常数Td；采样周期T
这三个量定义在结构体中只是想完整表达PID参量，实际运用中已经通过代换$K_i = T * K_p/T_i$，
$K_d = K_pT_d/T$将这三个变量隐藏在了新的K_p, K_i, K_d中，所以调参时只需关注K_p, K_i, K_d。

2. 写位置式PID函数

@brief （简要说明） PID->out = Kp * e(k)+Ki * ∑e(k)+Kd * [e(k) - e(k-1)]
@param （变量） setvalue : 设置值（期望值）； actualvalue: 实际值
@retval （返回值） PID->out：输出控制量

```
float PID_location(float setvalue, float actualvalue, PID_LocTypeDef *PID)
{
    PID->ek = setvalue-actualvalue;
    PID->location_sum += PID->ek;       //计算累计误差值

    //积分限幅,超过积分误差设定最大值，保持最大值，小于积分误差设定最小值，保持最小值
    if((PID->Ki != 0)&&(PID->location_sum > (PID_LIMIT_MAX    PID->Ki)))
    {
        PID->location_sum = PID_LIMIT_MAX/PID->Ki;
     }
    if((PID->Ki != 0)&&(PID->location_sum < (PID_LIMIT_MIN    PID->Ki)))
    {
        PID->location_sum = PID_LIMIT_MIN/PID->Ki;
    }

    //位置式PID表达式
    PID->out = PID->Kp*PID->ek + (PID->Ki*PID->location_sum) + PID->Kd*(PID-
>ek-PID->ek_prev);
    PID->ek_prev = PID->ek;

    //PID->out限幅
    if(PID->out < PID_LIMIT_MIN)    PID->out = PID_LIMIT_MIN;
    if(PID->out > PID_LIMIT_MAX)    PID->out = PID_LIMIT_MAX;

    return PID->out;
}
```

3. 写增量式PID函数

@brief （简要说明） PID->out += Kp * [e(k)-e(k-1)] + Ki*e(k) + Kd[e(k)-2e(k-1)+e(k-2)]
@param （变量） setvalue : 设置值（期望值）； actualvalue: 实际值
@retval （返回值） PID->out：输出控制量

```
    PID->ek = setvalue - actualvalue;

    //增量式PID表达式
    PID->out += PID->Kp*(PID->ek - PID->ek_prev) + PID->Ki*PID->ek + PID->Kd *
(PID->ek -2*PID->ek_prev+PID->ek_prev_prev);

    //更新上一次的误差和上上次的误差
    PID->ek_prev_prev = PID->ek_prev;
    PID->ek_prev = PID->ek;
```

```
    //PID->out限幅
    if(PID->out < PID_LIMIT_MIN)    PID->out = PID_LIMIT_MIN;
    if(PID->out > PID_LIMIT_MAX)    PID->out = PID_LIMIT_MAX;

    return PID->out;
```

4. 主函数调用PID.h中的三个函数进行PID控制

```
void PID_Init(PID_LocTypeDef *PID);//初始化
float PID_location(float setvalue, float actualvalue, PID_LocTypeDef *PID);//
位置式
float PID_increment(float setvalue, float actualvalue, PID_LocTypeDef *PID);//
增量式
```

框图

以控制四轴飞行器为例：