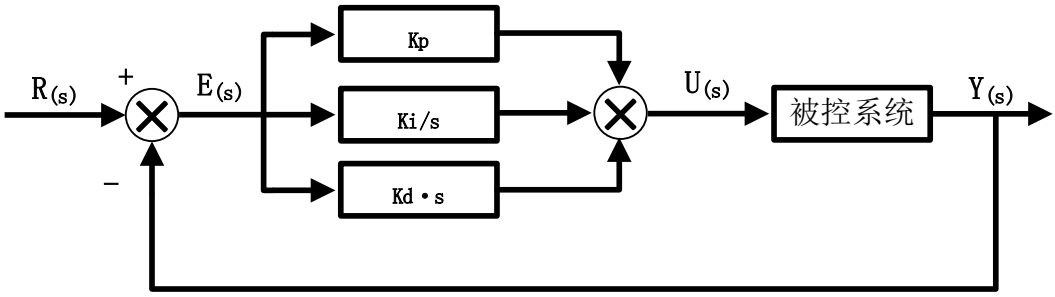


PID 控制系统框图

传递函数： $D_{(s)}=\frac{U_{(s)}}{E_{(s)}}=Kp+\frac{Ki}{s}+Kd\cdot s$

S 域传递函数：$U_{(s)}=Kp\cdot E_{(s)}+Ki\cdot\frac{E_{(s)}}{s}+Kd\cdot E_{(s)}\cdot s$

时域传递函数：$U_{(t)}=Kp\cdot e_{(t)}+Ki\cdot\frac{1}{Ti}\int_0^t e_{(t)}\cdot dt+Kd\cdot Td\cdot\frac{de_{(t)}}{dt}$

离散化：$t\approx nT(\mathrm{n}=0,1,2,\cdots K)$（采样）

$$u_{(t)}\approx u_{(KT)}$$

$$e_{(t)}\approx e_{(KT)}$$

$$\int_0^t e_{(t)}\cdot dt\approx T\cdot\sum_{j=0}^{K}e_{(jT)}\ （求“和”）$$

$$\frac{de_{(t)}}{dt}\approx\frac{[e_{(KT)}-e_{(KT-T)}]}{T}\ （求“斜率”）$$

位置式 PID：

$$u_{(KT)}=Kp\cdot e_{(KT)}+Ki\cdot\frac{T}{Ti}\cdot\sum_{j=0}^{K}e_{(jT)}+Kd\cdot\frac{Td}{T}\cdot[e_{(KT)}-e_{(KT-T)}]$$

增量式 PID：

$$\Delta u_{(T)}=Kp\cdot[e_{(KT)}-e_{(KT-T)}]+Ki\cdot\frac{T}{Ti}\cdot e_{(KT)}+Kd\cdot\frac{Td}{T}\cdot[e_{(KT)}-2e_{(KT-T)}+e_{(KT-2T)}]$$

增量式 PID（化简）：

$$\Delta u_{(T)}=A\cdot[e_{(KT)}-e_{(KT-T)}]+B\cdot e_{(KT)}+C\cdot[e_{(KT)}-2e_{(KT-T)}+e_{(KT-2T)}]$$

$$ItemRatio:\ A=Kp;\ B=Ki\cdot\frac{T}{Ti};\ \ C=Kd\cdot\frac{Td}{T};$$

基本的位置式和增量式 PID Code：

```
typedef struct{
    float kp,ki,kd;//Kp=Kp;Ki=Kp*T/Ti;Kd=Kp*Td/T; 通常 Td>> T;
    float err,errLast;
    float errPre;
}Pid;

/*
 * 基本的位置式PID
 * 存在积分饱和现象
 */
float GeneralPPidCalc(Pid *p, float set, float feedback)
{
    register float temp;
    static float integralTerm;//积分项
    p->err = set - feedback;
    integralTerm += p->err;
    temp = p->kp * p->err + p->ki * integralTerm + p->kd *
(p->err - p->errLast);
    p->errLast = p->err;
    return temp;
}

/*
 * 基本的增量式PID
 */
float GeneralIPidCalc(Pid *p, float set, float feedback)
{
    register float temp;
    static float outputTerm;  //输出项
    p->err = set - feedback;
    temp = p->kp * (p->err - p->errLast)
        + p->ki * p->err
        + p->kd * (p->err - p->errLast - p->errLast +
p->errPre);

    p->errPre = p->errLast;
    p->errLast = p->err;
    return temp;//返回：增量后的输出
//    outputTerm += temp;
//    return outputTerm;//返回：增量后的输出
}
```

积分分离 PID（积分分离域 E_0）

1. 误差绝对值>E_0时，采用 PD 控制（使输出的积分项置为 0）
2. 误差绝对值<=E_0时，采用 PID 控制
3. 减少普通积分作用带来的系统超调

Code：

```
/*
 * 积分分离的位置式PID
 * 改善系统超调(由于积分饱和导致的)
 * 即误差很大时，不进行积分
 */
float IntegralSeparationPPidCalc(Pid *p, float set, float
feedback, float inteSepaThreshold)
{
```

```
    register float temp;
    static float integralTerm;//积分项
//  uint8_t ISRatio = 1; //积分分离系数
    p->err = set - feedback;
    integralTerm += p->err;
//  ABS(p->err) < InteSepaThreshold ? ISRatio : -ISRatio;
//  temp = p->kp * p->err + p->ki * integralTerm * ISRatio +
p->kd * (p->err - p->errLast);
    if(ABS(p->err) < inteSepaThreshold)
    {
        temp = p->kp * p->err + p->ki * integralTerm + p->kd *
(p->err - p->errLast);
    }
    else
    {
        temp = p->kp * p->err + p->kd * (p->err - p->errLast);
    }
    p->errLast = p->err;
    return temp;
}
```

不完全微分PID（低通滤波器时间常数T_f 系统采样时间T_s） Partial ：不完全

1. PID 调节器的输出后串接低通滤波器（一阶惯性环节）来抑制微分中的高频干扰。

2. $G_{f(s)}=\dfrac{u_{(s\_Final\_out)}}{u_{(s\_PID\_out)}}=\dfrac{1}{1+T_f s}$

3. 推导过程：

   ① 2.中式子交叉相乘：$u_{(s\_Final\_out)}+u_{(s\_Final\_out)}\cdot T_f\cdot s=u_{(s\_PID\_out)}$

   ② 上式反拉氏变换 $\mathcal{L}^{-1}$，依据 $u_{(s)}\cdot s\xrightarrow{\mathcal{L}^{-1}}\dfrac{du_{(t)}}{d_{(t)}}$ 得到：

   $$u_{(t\_Final\_out)}+T_f\cdot\frac{du_{(t\_Final\_out)}}{d_{(t)}}=u_{(t\_PID\_out)}$$

   ③ 依照 $t=nT(n=0,1,2,\cdots k)$；$u_{(t)}=u_{(kT)}$ 离散化（采样），并整理成可编程差分方程：令 $a=\dfrac{T_f}{T_f+T_s}$，$u_{(kT)}=a\cdot u_{(kT-T)}+(1-a)\cdot u'_{(kT)}$；其中 $u'_{(kT)}\rightarrow u_{(kT\_PID\_out)}$

   ④ 通俗理解：$u_{(Final\_out)}=a\cdot u_{(Last\_out)}+(1-a)\cdot u_{(PID\_out)}$（增量式算法中，u 均为 Δu，只是增量部分，没有叠加部分，切记切记！）

Code：（前边的为公式推导的方法，后边的为单纯积分项的不完全微分 PID（陶永华 新型 pid 及其应用））

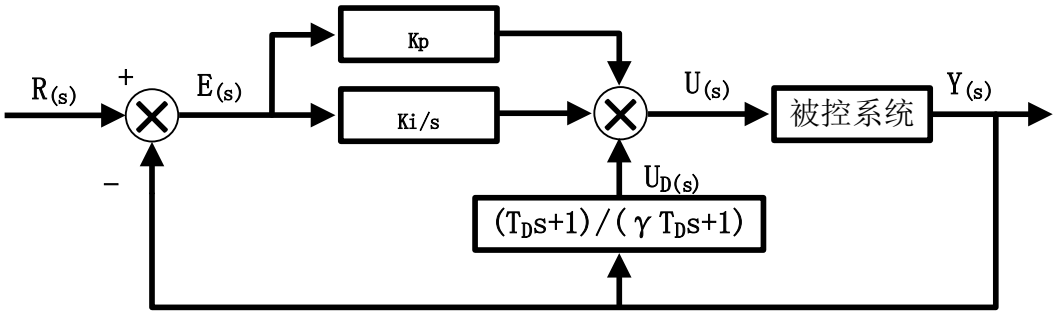
```
/*
```

```
 * 不完全微分的位置式PID(在整个的输出后加入一阶惯性环节)
 * 增加微分作用周期，抑制其引入的高频的干扰
 * 从公式的角度理解：系统的当前输出=系统上一次输出和本次输出的
加权平均数
 * a>=0 && a <=1    如果a不满足该要求，则为通用的位置式pid
 */
float IncompleteDifferentialPPidCalc2(Pid *p, float set, float
feedback, float a)
{
    register float temp;
    static float outputLast;    //系统的上一次输出值
    static float integralTerm;  //积分项
    a = (a > 1 || a < 0) ? 0 : a;//将a限幅，在了解a的范围后中
可以去掉进行优化

    p->err = set - feedback;
    integralTerm += p->err;
    temp = (p->kp * p->err + p->ki * integralTerm + p->kd *
(p->err - p->errLast)) * (1 - a)
            + outputLast * a;

    p->errLast = p->err;
    outputLast = temp;

    return temp;
}
/*
 * 不完全微分的位置式PID(只对微分项加入一阶惯性环节)
 * 增加微分作用周期，抑制其引入的高频的干扰
 * 从公式的角度理解：系统的当前输出=系统上一次输出和本次输出的
加权平均数
 * a>=0 && a <=1    如果a不满足该要求，则为通用的位置式pid
 */
float IncompleteDifferentialPPidCalc1(Pid *p, float set, float
feedback, float a)
{
    register float temp;
    register float differentialTerm;
    static float differentialLast;  //系统的上一次微分项
    static float integralTerm;  //积分项
    a = (a > 1 || a < 0) ? 0 : a;//将a限幅，在了解a的范围后中
可以去掉进行优化
    p->err = set - feedback;
    integralTerm += p->err;
    differentialTerm = (p->err - p->errLast);
    temp = p->kp * p->err
           + p->ki * integralTerm
           + p->kd * (differentialTerm * (1 - a) +
differentialLast * a);

    p->errLast = p->err;
    differentialLast = differentialTerm;

    return temp;
}
```

微分先行PID differential in advance

1. 单级控制器中，对输出量进行微分（即反馈值进行微分），设定值不进行微分，来适应设定值频繁变换的场合，避免其引起的过大的超调量和执行器的剧烈震荡；串级系统中，是对副控回路（后级）的偏差（主控回路的输出-最终的输出）进行微分（即对主控回路的输出和最终的输出（最终反馈值）都进行微分）。

2. 根据自己的理解，绘制的其控制结构图



其中：$y_{(s)}$为采样值，$u_{(s)}$为控制器输出值，$r_{(s)}$为设定值；

3. 微分部分传递函数：$G_{(s)}=\frac{u_{D(s)}}{y_{(s)}}=\frac{T_D\cdot s+1}{\gamma\cdot T_D\cdot s+1};(\gamma<1)$ ，$\frac{1}{\gamma\cdot T_D\cdot s+1}$相当于低通滤波器.

4. 交叉相乘，反拉氏变换，离散化，整理为可编程差分方程：

$$u_{D(kT)}=\frac{\gamma T_D}{\gamma T_D+T_S}\cdot u_{D(kT-T)}+\frac{T_D+T_S}{\gamma T_D+T_S}\cdot y_{(kT)}+\frac{T_D}{\gamma T_D+T_S}\cdot y_{(kT-T)}$$

Code：

```
/*
 * 微分先行的位置式PID
 * 从公式的角度理解：系统的当前输出=系统上一次输出和本次输出的加权平均数
 * gama<1
 */
float DifferentialForwardPPidCalc(Pid *p, float set, float feedback)
{
#define gama    0.5//<1
#define Td      0.5//随意示意值
#define Ts      0.1//随意示意值

    register float temp;S
    register float c1,c2,c3;
    c1 = gama*Td/(gama*Td + Ts);
    c2 = (Td + Ts)/(gama*Td + Ts);
    c3 = Td/(gama*Td + Ts);
    static float differentailTermLast;//系统的上一次微分项的值
    static float feedbackLast;    //系统的上一次反馈值
    static float integralTerm;  //积分项
    static float differentailTerm;  //微分项
```

```
    p->err = set - feedback;
    integralTerm += p->err;
    differentailTerm = c1 * differentailTermLast + c2 *
feedback - c3 * feedbackLast;
    temp = p->kp * p->err + p->ki * integralTerm +
differentailTerm;

    feedbackLast = feedback;
    differentailTermLast = differentailTerm;

    return temp;
}
```

低通滤波器

```
/* 低通滤波器
 * Input输入变量
 * a=Tf/(Tf+Ts)
 * a>=0 && a<=1
 * 低通滤波器时间常数Tf
 * 系统采样时间Ts
 * return ：输出变量
 */
float Low_PassFilter(float Input, float a)
{
    static float Mark_LastInput;
    register float temp;
    a = (a > 1 || a < 0) ? 0 : a;//将a限幅，在了解a的范围后中
可以去掉进行优化
    temp = a * Mark_LastInput
            + (1-a) * Input;

    Mark_LastInput = Input;//迭代
    return temp;
}
```