PID 控制器

version 1.0.1 author dungloi

理论

PID（*Proportional-Integral-Derivative*）控制器是一种线性控制器，根据被控对象给定值 $r(t)$ 和实际值 $y(t)$ 的控制偏差

$$ e(t)=r(t)-y(t) $$

构成控制律

$$ u(t)=K_c[e(t)+{1 \over {T_i}}\int ^t_0e(t){\mathrm{d}}t+T_d{\frac{\mathrm de(t)}{\mathrm dt}}] $$

或以传递函数表示

$$ G(s)=\frac{U(s)}{E(s)}=K_c(1+\frac1{T_is}+T_ds) $$

其中积分分量能够提升系统的稳态性能；微分分量能够改善系统的动态性能。

在数字 PID 控制中，使用的是离散化的 PID 控制器。在模拟 PID 的理论基础上，以一系列采样时刻点 kT 代表连续时间 t，以矩形法数值积分近似代替积分，以一阶后向差分近似替代微分，可得离散 PID 表达式

$$ \begin{align} u(k)&=K_p[e(k)+\frac{1}{T_i}\sum^k_{j=0}e(j)T+T_d\frac{e(k)-e(k-1)}{T}]\\ &=K_pe(k)+K_i\sum^k_{j=0}e(j)T+K_d\frac{e(k)-e(k-1)}{T} \end{align} $$

在标准的数字 PID 控制器的基础上，还可以引入一系列优化算法，形成非标准控制算法，以改善系统品质，满足不同控制系统的需要。

> 说明：以下优化均假设被控对象输出从两个方向接近给定值的过程是对称的，实际实现时可能需要针对不同方向的收敛过程设置不同的阈值。

积分项优化

无扰动操作（Bumpless Operation） feature embedded

无扰动操作包括参数初始化过程和变参数的情况。考虑在控制器工作过程中改变积分增益参数，可能引起输出的较大变化，不利于控制。因此将积分项改进为积分增益与误差乘积的累积，而不是误差累积后再乘以积分增益。积分项表示为：

$$ u_i(k)=\sum^k_{j=0}K_ie(j)T $$

梯形积分（Trapezoidal Rule） feature embedded

为尽量减小余差，提高积分项的运算精度，可将矩形积分改为梯形积分。积分项表示为：

$$ u_i(k)=\sum^k_{j=0}K_i{e(j)+e(j-1)\over 2}T \ $$

抗积分饱和（Anti-windup） feature embedded default u=11451.4

积分饱和现象指若系统存在一个方向的偏差，控制器输出由于积分作用的不断累加而持续增大，若超出执行机构正常运行范围便进入了饱和区。一旦系统出现反向偏差，控制器输出逐渐从饱和区退出，进入饱和区越深则退出所需时间越长。在这段时间内，执行机构仍停留在极限位置，而不能随偏差反向立即做出相应的改变，此时将造成控制性能恶化。抗积分饱和法在计算控制器输出 $u(k)$ 时，首先判断上一时刻 $u(k-1)$ 是否已超出限制范围：若 $u(k-1)>u_{m}$，只累加负偏差；若 $u(k-1)<-u_{m}$，只累加正偏差。积分项表示为：

$$ \begin{align} u_i(k)=\sum^{k-1}_{j=0}K_ie(j)T+\alpha K_i e(k)T\ 其中\ \alpha= \begin{cases} 0, &u(k-1)\cdot e(k)>0\ ,|u(k-1)| {\geqslant}u_m\ 1, &else \end{cases} \end{align} $$

其中阈值 u_m 根据实际情况人为指定。

积分分离（Integration Separation） feature optional default OFF

积分环节的作用主要为消除静差，提高控制精度。但在短时间系统输出产生较大偏差时，可能会造成积分过度积累，使控制量过大，引起系统超调甚至振荡。此时需要引入积分分离，当被控量与给定值偏差较大时，取消积分作用；接近时引入积分控制。积分项表示为：

$$ \begin{align} u_i(k)=\beta \sum^k_{j=t_0}K_ie(j)T \ 其中\ \beta= \begin{cases} 1, &|e(k)| {\leqslant} \epsilon_0\ 0, &|e(k)| {>} \epsilon_0 \end{cases} \end{align} $$

其中 t_0 表示引入积分控制的时刻，阈值 ϵ_0 根据实际情况人为指定。

变速积分（Changing Rate） feature optional default OFF | f=1.0

基于积分分离优化的思想，根据系统偏差大小改变积分的速度，偏差越大，积分越慢，反之则越快，形成连续的变化过程。为此，设置系数 $f[e(k)]$，当 $|e(k)|$ 增大时，f 减小，反之增大，其值在 $[0,1]$ 变化。积分项表示为：

$$ u_i(k)=\sum^{k-1}_{j=0}K_ie(j)T+f[e(k)]\cdot K_i e(k)T\ $$

系数 f 与当前偏差 $e(k)$ 的关系可以是线性的，可设为

$$ f[e(k)]= \begin{cases} 1, &|e(k)| {\leqslant}A\ \frac{B-|e(k)|}{B-A},&A<|e(k)|{\leqslant} B\ 0, &|e(k)|>B \end{cases} $$

其中阈值 $A,B\in(0,+\infty)$ 根据实际情况人为指定。

微分项优化

微分滤波（Derivative Filter） feature embedded default w=1.0

微分项可改善系统的动态特性，但也易引进高频干扰，在误差扰动突变时尤其显出微分项的不足。因此，可以在控制算法中加入一阶惯性环节（低通滤波器），可使系统性能得到改善。可将滤波器直接加在微分环节上或控制器输出上。此方案也称为“不完全微分”。

事实上，还可以针对系统的频域特性设计合适的滤波器，这里不做更深入的探究。

对微分环节输出进行一阶滤波，微分项表示为：

$$ u_d(k)=K_d\cdot w_0\cdot {e(k)-e(k-1)\over T}+(1-w_0) \cdot u_d(k-1) $$

其中系数 $w_0\in (0,1]$ 根据实际情况人为指定。

只对输出微分（Derivative of the process variable） feature embedded

为避免由于给定值频繁升降（尤其是阶跃）而引起的系统振荡，可采用只对输出微分的优化算法，其特点是只对系统输出进行微分。这样，在改变给定值时，微分项输出仅与被控量的变化相关，而这种变化通常是比较缓和的，从而能够明显改善系统的动态特性。此方案也称为“微分先行”。

简单的离散形式优化的微分项表示为：

$$ u_d(k)=K_d\frac{y(k-1)-y(k)}{T}\ $$

其他优化

带死区（Dead Band） feature optional default OFF

为避免控制作用过于频繁，消除由于频繁动作所引起的振荡，必要时可采用带死区的 PID 控制算法，即判断控制偏差是否小于给定阈值，若小于则不输出。控制器输出表示如下：

$$ u(k)= \begin{cases} 0, &|e(k)| {\leqslant} \epsilon_1\ u(k), &|e(k)| {>} \epsilon_1 \end{cases} $$

其中阈值 ϵ_1 根据实际情况人为指定。

给定值平滑（Setpoint Ramping） feature optional default OFF

当给定值出现较大的阶跃变化，很容易引起超调。使用线性斜坡函数或一阶滤波为给定值安排过渡过程，使其从其旧值逐渐变化到新值，以避免阶跃变化产生的不连续性，进而使对象运行平稳，适用于高精度伺服系统的位置跟踪。

实际应用中，一般给出最大阶跃范围。若超出该范围，对给定值进行一阶滤波处理。给定值表示如下：

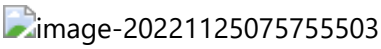
$$ r(k)= \begin{cases} r(k), &|e(k)| {\leqslant} \epsilon_2\ w_1\cdot r(k)+(1-w_1)r(k-1), &|e(k)| {>} \epsilon_2 \end{cases} $$

其中阈值 ϵ_2 及系数 $w_1\in (0,1]$ 根据实际情况人为指定。

前馈校正（Feed-forward） feature optional default OFF

在前馈控制（开环）中考虑系统的已知信息，再将输出加到 PID 控制器（闭环）的控制输出，形成复合校正，能够进一步提升整体的系统性能。前馈校正通路由于不受反馈的影响，不会造成系统的振荡，从而能够在不影响稳定性的情况下改善系统的响应。前馈量通常可以单独提供控制器输出的主要部分；PID 控制器则用来补偿给定值和实际值之间的误差。

前馈量依可量测扰动或给定量来产生，构成按扰动补偿和按输入补偿两种复合控制形式。



我们重点关注按输入补偿的复合控制系统设计。由上图 (b) ，系统输出为 $$ C(s)=\frac{[G_1(s)+G_r(s)]G_2(s)}{1+G_1(s)G_2(s)}R(s) $$ 如果选择前馈补偿装置传函 $G_r(s)=\frac1{G_2(s)}$，则有 $$ C(s)=R(s) $$ 使得系统输出复现输入，具有理想的时间响应特性。实际这种全补偿难以实现，因此一般采用部分补偿，常用的方法有取输入信号的一阶导数作为前馈补偿信号，即： $$ G_r(s)=\lambda_1s $$ 此方案原理可参考资料[2]第283页。实

际中由于控制器工作频率可能高于给定信号频率，此时若采用后向差分则不能获得平滑的前馈量。因此考虑采用跟踪微分器，输出量表示为 $$ u_f(k)=\lambda_1 \cdot TD[r(k)] $$ 其中常系数 $\lambda_1\in[0,+\infty)$ 表示前馈补偿信号的强度，根据实际情况人为指定。

串级控制（Cascade Control）

二（或多）个 PID 控制器可以组合在一起，以产生更好的动态性能，称之为串级 PID 控制。以两个 PID 控制器组成的串级控制器为例，其中一个 PID 控制器充当外（主）回路控制器，控制像液面高度或是速度等主要的物理量；另一个 PID 控制器充当内（副）回路控制器，以外回路控制器的输出做为其给定值，通常控制更快速变化的参数，例如流量或加速度等。使用串级 PID 控制可以提升控制器的工作频率，并降低其时间常数；内回路能够抑制扰动，从而大大减小扰动对外回路的影响。

快速开始

组件源码仓库地址：https://github.com/ZJU-HelloWorld/HW-Components

PID 模块依赖于滤波器和跟踪微分器（可选）。要在项目中使用该组件，需添加仓库内的以下文件：

```
algorithms/pid.c
algorithms/pid.h
algorithms/filter.c  # 可选，若使用跟踪微分器，请添加
algorithms/filter.h
tools.h
system.h
```

使用前准备

本组件依赖的 filter.c/h 涉及 CMSIS-DSP 矩阵运算等操作。

使用本组件前需要做以下准备：

- 添加源文件, 包含头文件路径；注意 DSP 版本须在 1.10.0 及以上
- 添加预处理宏以开启浮点运算单元（FPU）
- 在使用 STM32CubeMX 生成项目时，请在 Code Generator 界面 Enable Full Assert，来帮助断言设备驱动中的错误；在 `main.c` 中修改 `assert_failed` 函数以指示断言结果，如添加 `while(1);`
- 在 `system.h` 中 `system options: user config` 处进行系统设置

示例

在项目中引用头文件：

```
#include "pid.h"
```

实例化一个 PID 控制器：

=== "Standard Controller" `c Pid_t pid;` === "N-loop Cascade Controller" `c Pid_t pid_n_loop;`

创建参数结构体数组，按节点顺序（外回路 -> 内回路）设置控制器节点参数，参数含义见组件说明。如带给定值过零处理（单位 $\rm rad$）的控制器：

=== "Standard Controller" ```c const PidParams_t pid_param = { .kType = PID_ACROSS0_RAD, .kImprvOption = INTE_SEPARATION | INTE_CHANGING_RATE | DEAD_BAND | SETPOINT_RAMPING | SETPOINT_FEED_FORWARD,

```
    .kp = 500.0f,
    .ki = 5.0f,
    .kd = 1.0f,
    .kOutMin = -30000,
    .kOutMax = 30000,

    .kWindUpOutMax = 5000,
    .kDWeight = 0.9f,

    .kISeparThresUpper = 1.0f,
    .kISeparThresLower = -1.0f,
    .kUpperB = 2.0f,
    .kLowerA = 0.1f,

    .kDeadBandThresUpper = 1.0f,
    .kDeadBandThresLower = -1.0f,
    .kSpThresUpper = 30,
    .kSpThresLower = -30,
    .kSpWeight = 0.5f,
    .kf = 1.0f,
};
```
```

=== "N-loop Cascade Controller" `c const PidParams_t pid_n_loop_param[N] = {{...}, {...}, ...};`

其中 `kType, kImprvOption, kp, ki, kd, kOutMin, kOutMax, kWindUpOutMax, kDWeight` 为基础参数，请在初始化时按需求指定；其他参数为搭配优化选项的可选参数。若无需优化，使用 `.kImprvOption = IMPRV_NONE` 语句。

> 说明：所有参数均具有缺省值，部分参数的缺省值已在理论部分给出，未给出的默认为 0。

初始化 PID 控制器，如：

=== "Standard Controller" `c PidInit(&pid, 1, &pid_param);` === "N-loop Cascade Controller" `c PidInit(&pid_n_loop, N, pid_n_loop_param);`

> 若希望引入按输入前馈补偿，则需开启 `SETPOINT_FEED_FORWARD` 优化选项，并使用跟踪微分器，该组件位于 `Utils/filter.c` 中。实例化一个跟踪微分器并传入参数进行初始化，然后向已实例化的 PID 控制器指明节点编号（编号顺序为外回路 -> 内回路，从 0 开始），向该节点注册（线性）跟踪微分器：
```

```
Td_t td;
TdInit(&td, r, h0, h);         // h = 1.0f / CTRL_FREQ
pid.tdRegister(&pid, N, &td); // loop no.N, numbered from 0
```

其中 h_0 为滤波因子， h_0 越大滤波效果越好； h 为步长， h 越小滤波效果越好；一般来说， h_0 略大于步长 h； r 为快速因子， r 越大，跟踪越快。

更多原理和参数调节规律详见滤波器算法组件说明，或咨询沈组长 😍

传入 PID 控制器句柄、最外回路给定值、各回路反馈值（存储顺序由外回路到内回路）数组首地址、存储输出量的内存地址，计算 PID 控制器输出：

```
pid.calcPid(&pid, ref, &fdb, &out);
```

若 PID 控制器是动态创建的，在其生存周期终止前还需要手动释放内存：

```
pid.delPid(&pid);
```

组件说明

`Pid` 类

PID 控制器。

属性

名称	类型	示例值	描述
`loops`	`uint8_t`	1 / 2 / 3	PID 级数
`p_node_list`	`PidNode_t*`	/	PID 节点列表首地址

方法

名称	参数说明	描述
`PidInit`	传入参数 `loops` 标识本 PID 控制器级数；`params_list` 为存储各节点 PID 参数结构体数组的首地址，请设定所有 `loops` 个节点参数，将按外回路 -> 内回路顺序读取数据	用传入的参数初始化一个 PID 控制器。
`calcPid`	传入参数 `ref` 为最外回路给定值；`fdb` 为各回路反馈值（存储顺序由外回路到内回路）数组首地址；传入地址 `out`，存储最内回路（末级） PID 输出结果	根据输入的数据，计算 PID 控制器输出。

名称	参数说明	描述
`tdRegister`	传入参数 `loop_no` 为指定回路按外回路 -> 内回路顺序从 0 开始得到的编号；传入指定回路 ref 的跟踪微分器指针 `p_td` 以实现前馈	为对应 PID 节点注册跟踪微分器。
`delPid`	/	释放为 PID 控制器所有节点动态申请的内存。
`resetData`	/	重置所有节点 PID 中间项。
`resetNodeParams`	传入参数 `loop_no` 为指定回路按外回路 -> 内回路顺序从 0 开始得到的编号；传入 PID 节点参数数值	重置对应节点 PID 参数。
`getNode`	传入参数 `loop_no` 为指定回路按外回路 -> 内回路顺序从 0 开始得到的编号；返回类型 `PidNode_t*`	返回对应 PID 节点指针。
`getNodeData`	传入参数 `loop_no` 为指定回路按外回路 -> 内回路顺序从 0 开始得到的编号；返回类型 `PidData_t*`	返回对应 PID 节点中间项结构体指针。
`getNodeParams`	传入参数 `loop_no` 为指定回路按外回路 -> 内回路顺序从 0 开始得到的编号；返回类型 `PidParams_t*`	返回对应 PID 节点参数结构体指针。

`PidNode` 结构体

存储 PID 节点数据，用户不可操作。

名称	类型	示例值	描述
`ref`	`float`	1.0	给定值
`fdb`	`float`	1.0	反馈值
`out`	`float`	1.0	控制器输出
`td`	`Td_t*`	/	指向跟踪微分器，可选，若不使用输入量前馈则为空
`params`	`PidParams_t`	/	控制器相关初始参数
`data`	`PidData_t`	/	运行时中间项数据
`last_tick`	`uint32_t`	1919810	上一次调用该控制器的时刻，单位：$\rm ms$
`sample_interval`	`uint32_t`	1	调用控制器的间隔，单位：$\rm ms$

`PidParams` 结构体

存储 PID 参数。

名称	类型	示例值	描述
kType	PidType_t	PID_ACROSS0_RAD PID_ACROSS0_DEGREE PID_DEFAULT	PID 处理类型
kImprvOption	uint16_t	/	优化选项集合,由所有需启用的优化枚举类型 PidImprvType_t 的优化选项 “按位与” 合成 uint16_t
kp / ki / kd	float	500.0 / 5.0 /1.0	PID 增益
kOutMin	float	-30000	最小输出
kOutMax	float	30000	最大输出
kWindUpOutMax	float	5000	使能抗积分饱和的临界输出，$\pm$kWindUpOutMax
kDWeight	float	0.9	微分一阶滤波系数，默认值 1.0
kISeparThresUpper kISeparThresLower	float	1.0 / -1.0	使能积分分离的临界误差
kUpperB kLowerA	float	2.0 / 0.1	线性变速积分误差区间，$\pm$kUpperB / kLowerA
kDeadBandThresUpper kDeadBandThresLower	float	1.0/ -1.0	误差死区上下限
kSpThresUpper kSpThresLower	float	30 / -30	使能给定值平滑的临界误差
kSpWeight	float	0.5	给定值平滑一阶滤波系数
kf	float	1.0	前馈补偿强度系数

附录

版本说明

版本号	发布日期	说明	贡献者
version 0.9.0	2022.10.31	完成PID优化文档并实现	薛东来
version 1.0.0	2022.12.01	加入串级、前馈内容，测试发布	薛东来
version 1.0.1	2022.12.03	修正部分表述，使用动态内存分配，无需手动	薛东来

参考资料

[1] 刘金琨. 先进PID控制MATLAB仿真. 第4版. 北京: 电子工业出版社, 2016.

[2] 胡寿松主编. 自动控制原理. 第7版. 北京: 科学出版社, 2019.02.

[3] https://en.wikipedia.org/wiki/PID_controller

[4] http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/ （Arduino PID 库作者博客，实现在这里）