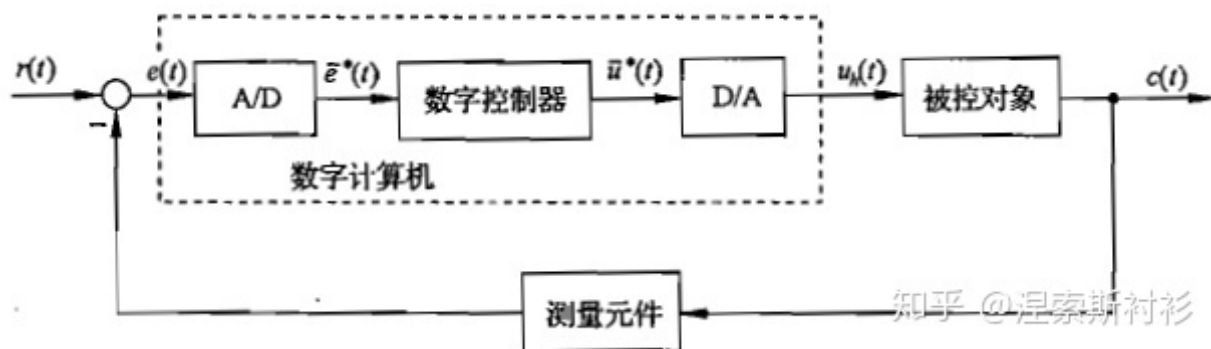


控制算法原理及实现之PID（以飞控为例）

引言

在我们的日常生活中，稍微注意，便可发现控制理论的运用及实践无处不在，从每天所用的手机、电脑以及汽车等等，都是一个个复杂的控制系统。为了保持系统的稳定、可控，**反馈**必不可少，通过反馈使得系统成为一个闭环系统 (close loop)，而通过**控制**则可使系统实现我们预期的状态，如图便是一个常见的闭环反馈控制系统框图[1]。



知乎 @涅索斯衬衫

PID(比例-积分-微分)控制作为应用最广(95%)的控制算法，以至于很多人对于控制的理解就是PID。虽不正确，但足以说明其价值。因此，深刻了解其数学原理及具体实现，具有十分重要的意义。

控制算法的实现，必然会涉及到**s域->z域->时域**的变换，可通过专栏的另一篇文章《[一阶低通滤波（LPF）的原理及应用（以APM/PX4飞控为例）](#)》了解从s域到最终的数字实现的过程。

目录：

-
- 1、离散化方法
 - 2、PID控制器的数学描述
 - 3、PID控制器的算法实现及实例
 - 4、PID控制器对系统性能的影响
-

问题：

阅读完本文后，可以思考两个问题。



- 1、在飞控的具体应用中，当加入微分控制作用时，为何还要另外引入一个一阶低通滤波器？这样做的目的是为了什么？
- 2、在使用一阶低通滤波器时，其截止频率选择的依据是什么？

一、离散化方法

为什么要离散化？

随着数字控制技术的不断发展，控制器的实现越来越多地采用数字控制，而实际的系统是一个连续的系统(s域)，为了实现数字控制，这其中必然涉及到连续域(s域)到数字域(z域)的转换，即我们所说的**离散化方法**。

模拟调节器的离散化方法有很多，下面介绍几种常用的离散化方法。

1. 差分变换法
2. 零阶保持器法
3. 双线性变换法

1、差分变换法

以一阶后项差分为例，可知：

$$\frac{du}{dt} \approx \frac{u(t) - u(t-1)}{T_s}$$

所以有：

$$s = \frac{1 - z^{-1}}{T_s}, \text{ 其中 } T_s \text{ 为采样周期。}$$



在后续的实例代码中，其**一阶低通滤波(LPF)**的实现就是采用的后向差分变换法。

2、零阶保持器法

零阶保持器法(ZOH)又称为阶跃响应不变法，其基本思想是：离散近似后的数字控制器的阶跃响应序列必须与模拟调节器的阶跃响应的采样值相等。其中采用的零阶保持器的传递函数为

$$H(s) = \frac{1 - e^{-T*s}}{s} , \text{ 其中 } T \text{ 为采样周期。}$$

假设一个模拟控制器的传递函数为 $C(s)$ ，采用零阶保持器法对其进行离散化时，应将 $C(s)$ 包含在内，即：

$$C(z) = z[H(s)C(s)]$$

在实际应用中，对于**控制对象C(s)**一般采用这种变换。

3、双线性变化法(Tustin变换法)

双线性变化法又称为Tustin变化，它是直接将s域函数转化成z域的一种近似方法。已知一个连续传递函数 $C(s)$ ，则 $C(z)$ 为

$$C(z) = C(s) \Big|_{s=\frac{2z-1}{T_s z+1}} , \text{ 其中 } T_s \text{ 为采样周期。}$$

4、matlab的c2d的函数的使用介绍

```
sysd = c2d(sys,Ts,method)
```



其中method有多种方法：zoh、impluse、tustin、matched。

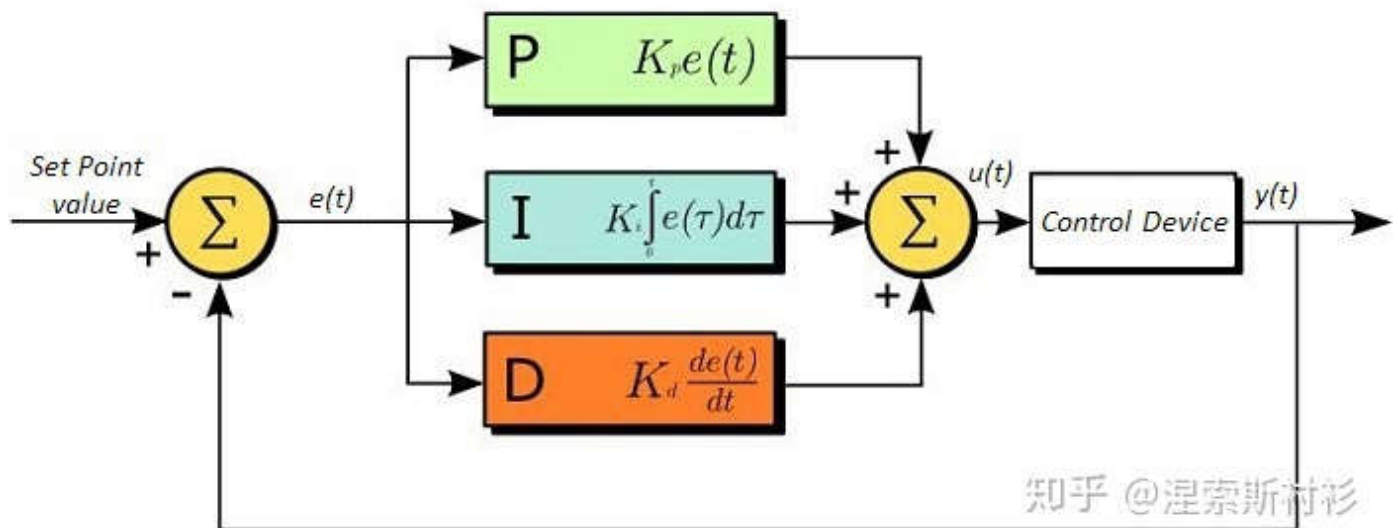
```
[mag,phase,omega] = bode(c2d(sys,Ts,'Tustin'));
```

离散化的方法的方法很多，但应用对象及场景的不同，采用的方法亦有所差别，在实际应用中，特别是仿真系统的搭建时需特别注意。**这一部分可参见文献[2]第五章。**

二、PID控制器的数学描述

PID控制器，顾名思义，就是比例-积分-微分(proportional-integral-derivative)控制器，通过比例-积分-微分对系统的综合控制作用，从而达到调节系统的目的。其中，PID时域表达式为：

$$u(t) = K_p[e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \frac{de(t)}{dt}] \quad (1)$$



在实际应用中，一定要理解控制量 $u(t)$ 的物理意义。由上述时域表示，不难得知其S域的表达式为：

$$C(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_p}{T_i} * \frac{1}{s} + K_p * T_d * s \quad (2)$$



采用**后项差分变换**：

$$s = \frac{z-1}{T * z} = \frac{1-z^{-1}}{T}$$

对上述PID连续域表达式进行后项差分离散化，则式中各项可近似表示为：

$$\left\{ \begin{array}{l} t \approx kT \quad k = 0, 1, 2, \dots \\ e(t) \approx e(kT) \\ \oint e(t)dt \approx \sum_{j=0}^k e(jT)T \approx T * \sum_{j=0}^k e(jT) \\ \frac{de(t)}{e(t)} \approx \frac{e(kT)-e((k-1)T)}{T} \end{array} \right.$$

为书写方便，采样序列 kT 用 k 简化表示，则式（1）离散为：

$$u(k) = K_p * e(k) + \frac{T}{T_i} \sum_{j=0}^k e(j) + \frac{T}{T_d} [e(k) - e(k-1)]$$

对其变换后，则**PID控制算法的通用表达式**如下：

$$u(k) = K_p * e(k) + K_i * \sum_{j=0}^k [e(j) * \Delta T_j] + K_d * \left[\frac{e(k) - e(k-1)}{\Delta T_k} \right]$$

其中， K_p , K_i , K_d 为PID控制器的参数。

三、PID控制器的算法实现及实例

PID控制算法作为最经典的控制算法，随着工业应用的不断发展及控制对象的不同，虽提出了许多改进的PID控制算法，如积分分离式PID算法、不完全微分PID算法以及微分先行PID算法等，来提高控制性能。但核心都是以广义的PID算法为基础，下面以PID算法在无人机中的应用，给出PID算法的具体实现过程。



3.1 算法实现

通过上述给出的PID控制器的通用表达式，不难理解无人机开源飞控APM中PID控制算法的实现流程。


```
///@file          AC_PID.cpp
///@brief         Generic PID algorithm

// set_input_filter_all - set input to PID controller
// input is filtered before the PID controllers are run
// this should be called before any other calls to get_p, get_i or g
void AC_PID::set_input_filter_all(float input)
{
    // don't process inf or NaN
    if (!isfinite(input)) {
        return;
    }

    // reset input filter to value received
    if (_flags._reset_filter) {
        _flags._reset_filter = false;
        _input = input;
        _derivative = 0.0f;
    }

    // update filter and calculate derivative
    float input_filt_change = get_filt_alpha() * (input - _input);
    _input = _input + input_filt_change;
    if (_dt > 0.0f) {
        _derivative = input_filt_change / _dt;
    }
}

// set_input_filter_d - set input to PID controller
// only input to the D portion of the controller is filtered
// this should be called before any other calls to get_p, get_i
```



```

void AC_PID::set_input_filter_d(float input)
{
    // don't process inf or NaN
    if (!isfinite(input)) {
        return;
    }

    // reset input filter to value received
    if (_flags._reset_filter) {
        _flags._reset_filter = false;
        _derivative = 0.0f;
    }

    // update filter and calculate derivative
    if (_dt > 0.0f) {
        float derivative = (input - _input) / _dt;
        _derivative = _derivative + get_filt_alpha() * (derivative - _derivative);
    }

    _input = input;
}

float AC_PID::get_p()
{
    _pid_info.P = (_input * _kp);
    return _pid_info.P;
}

float AC_PID::get_i()
{
    if (!is_zero(_ki) && !is_zero(_dt)) {
        _integrator += ((float)input * _ki) * _dt;
        if (_integrator < -_imax) {
            _integrator = -_imax;
        } else if (_integrator > _imax) {
            _integrator = _imax;
        }
    }
}

```




```

        _pid_info.I = _integrator;
        return _integrator;
    }
    return 0;
}

```

```

float AC_PID::get_d()
{
    // derivative component
    _pid_info.D = (_kd * _derivative);
    return _pid_info.D;
}

```

```

float AC_PID::get_ff(float requested_rate)
{
    _pid_info.FF = (float)requested_rate * _ff;
    return _pid_info.FF;
}

```

```

float AC_PID::get_pi()
{
    return get_p() + get_i();
}

```

```

float AC_PID::get_pid()
{
    return get_p() + get_i() + get_d();
}

```

```

// calc_filt_alpha - recalculate the input filter alpha
// 20HZ

```

```

float AC_PID::get_filt_alpha() const
{
    if (is_zero(_filt_hz)) {
        return 1.0f;
    }
}

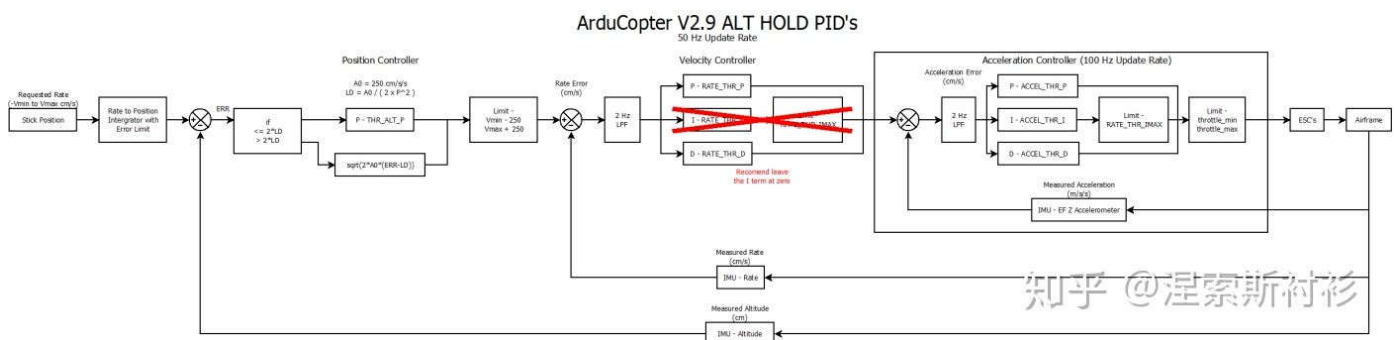
```



}

3.2 实例解析

在无人机开源飞控中，多使用串级PID控制。如图给出了APM飞控中高度控制框图[3]，下述代码为PID控制加速度环路接口实现部分代码(AC_PosControl.cpp)。



```
/// @file      AC_PosControl.cpp
/// @brief      PID Application
```

```
// rate_to_accel_z - calculates desired accel required to achieve the
// calculates desired acceleration and calls accel throttle controller
void AC_PosControl::rate_to_accel_z()
```

```
AC_PID& pid accel z;
```

```
// set input to PID
_pid_accel_z.set_input_filter_all(_accel_error.z);
_pid_accel_z.set_desired_rate(accel_target.z);
```

```
// separately calculate p, i, d values for logging
```



```
p = _pid_accel_z.get_p();

// get i term
i = _pid_accel_z.get_integrator();

// get d term
d = _pid_accel_z.get_d();

float thr_out = (p+i+d)/1000.0f + _motors.get_throttle_hover();
```

四、PID控制器对系统性能的影响

- P控制器主要提高系统的响应，减少系统的稳态误差，提高系统的控制精度。从频域角度而言，P控制器可增大系统的带宽。但一味地加大增益P，会使系统的幅值裕度减小，其相对稳定性降低，甚至有可能造成系统不稳定。
- PI控制器相当于给系统增加了一个开环极点，并同时也增加一个开环零点，增加的开环零点可减小系统的阻尼。PI控制器提高了系统的型别，可使系统无静差，改善系统的稳态性能。
- PD控制器中的微分控制规律能增大系统的阻尼，从而改善系统的动态性能。实际上，微分作用反映的是误差的变化率的快慢，因此，通过微分作用可以抑制系统的噪声。从频域角度而言，微分作用可提升系统的相角裕度。

为更好地理解PID控制器对控制系统的影响，特给出一个仿真实例。通过改变PID控制器相应控制器的值(K_p , K_i , K_d)来观察系统的输出响应，即可对其有一个直观的认识。

仿真实例：

```
clear all
close all
```



```
tfs = tf([1],[1 2 1],'inputdelay',0.5)
```

```
kp = 1.648; ki = 0.955; kd = 0.40;
```

```
tfi = tf([1], [1 0]);
```

```
tfd = tf([1 0], [1]);
```

```
tfc = kp + ki*tfi +kd*tfd;
```

```
OpenLoop = series(tfc, tfs);
```

```
ETF = 1/(1 + OpenLoop);
```

```
CloseLoop = feedback(OpenLoop, 1);
```

```
%Step Response
```

```
figure(1)
```

```
step(CloseLoop);
```

```
figure(2)
```

```
bode(OpenLoop);
```

参考文献

[1] Karl Johan Åström, Richard M. Murray. Feedback Systems: An Introduction for Scientists and Engineers[M]. Princeton University Press, 2008.

[2] 高金源. 计算机控制系统：理论、设计与实现[M]. 北京航空航天大学出版社, 2001.

[3] ArduCopter 2.9 PID Loops for STABILIZE, ACRO and ALT_HOLD.



