CHEMICAL ENGINEERING AND ADVANCED MATERIALS
UNIVERSITY OF NEWCASTLE UPON TYNE
Wor' Department

Discretised PID Controllers

Part of a set of study notes on Digital Control
by
M. Tham

CONTENTS

Before discussing the design of digital control algorithms, let us consider discrete equivalents of analog controllers.

Analog Control refers to the design and implementation of controllers in the continuous domain.

This includes electronic controllers, which although discrete in nature, implements control by emulating the continuous nature of analog control strategies. A typical example is the electronic PI/PID algorithm. There are a number of ways by which this common and versatile controller can be implemented in discretised form.

PI/PID Controller Design from the Time domain

Consider the ideal PID controller written in the continuous time domain form:

$$u(t) = K_c e(t) + \frac{K_c}{T_i}\int_0^t e(t)dt + K_c T_d \frac{de(t)}{dt} + u_o$$

To discretise the controller, we need to approximate the integral and the derivative terms to forms suitable for computation by a computer. From a purely numerical point of view, we can use:

$$\frac{de(t)}{dt} \approx \frac{e(t) - e(t-1)}{T_s} \quad \text{and} \quad \int_0^t e(t)dt \approx T_s \sum_0^t e(i)$$

The discretised PID algorithm is therefore:

which is now in the form of a difference equation, suitable for coding in an appropriate programming language. This particular form of the PID algorithm is known as the **'positional'** PID controller, because the control signal is calculated with reference to a base level, u_o.

Back to top

PI/PID Controller Design from the Laplace Domain

We can also formulate discrete PID controllers directly from the Laplace domain. Here, the ideal PID algorithm is written as:

$$\frac{U(s)}{E(s)} = K_c\left[1 + \frac{1}{T_i s} + T_d s\right]$$

Now we can apply either the backward difference or bilinear transformation methods to get an equivalent discrete PID controller.

Say we apply the backward difference method. Then,

$$\frac{U(s)}{E(s)} = K_c\left[1 + \frac{1}{T_i s} + T_d s\right] \overset{backward-difference}{\rightarrow} u(t) = e(t)K_c\left[1 + \frac{T_s}{T_i\left(1-z^{-1}\right)} + T_d\frac{\left(1-z^{-1}\right)}{T_s}\right]$$

Simplification yields:

$$u(t) = u(t-1) + K_c\left[e(t) - e(t-1)\right] + \frac{K_c T_s}{T_i}e(t) + \frac{K_c T_d}{T_s}\left[e(t) - 2e(t-1) + e(t-2)\right]$$

This PID controller is different in structure to that obtained from time-domain considerations and is known as the '**velocity**' PID algorithm.

As opposed to the fixed control reference used in the positional algorithm, here, the calculation of current control uses the previous control value as reference. In essence, the control is calculated as a change, hence the term 'velocity form'. Application of the Bilinear Transform will yield a similar velocity form algorithm, due to the presence of the $(1-z^{-1})$ term in the approximation. Try it !

Back to top

Positional and Velocity PID Algorithms

Although the structures of the positional and velocity PID algorithms appear very different, they are in fact related. The positional algorithm as derived is:

$$u(t) = K_c e(t) + \frac{K_c T_s}{T_i}\sum_{i=0}^{t} e(i) + \frac{K_c T_d\left(e(t) - e(t-1)\right)}{T_s} + u_0$$

Time shifting back one sampling interval, we obtain

$$u(t-1) = K_c e(t-1) + \frac{K_c T_s}{T_i}\sum_{i=0}^{t-1} e(i) + \frac{T_d\left(e(t-1) - e(t-2)\right)}{T_s} + u_0$$

Subtracting this from the original, we end up with the velocity form, i.e.

$$u(t) = u(t-1) + K_c\left[e(t) - e(t-1)\right] + \frac{K_c T_s}{T_i}e(t) + \frac{K_c T_d}{T_s}\left[e(t) - 2e(t-1) + e(t-2)\right]$$

Back to top

Implementation and Performance of Discrete PID Controllers

Commissioning discrete PID Controllers

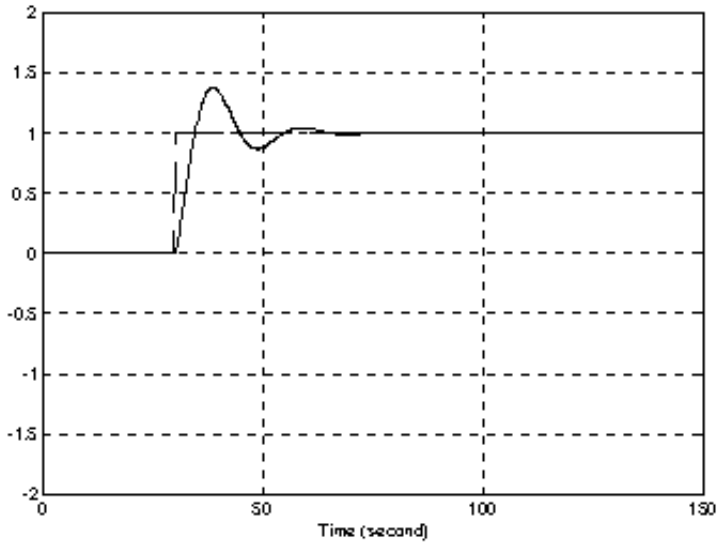
To calculate control, the positional PID requires knowledge of u_o, which is some steady-state control output level. This may have implications during commissioning. With the velocity form however, the previous control can be set to any reasonable arbitrary level, and hence commissioning is simpler.

Integral Windup

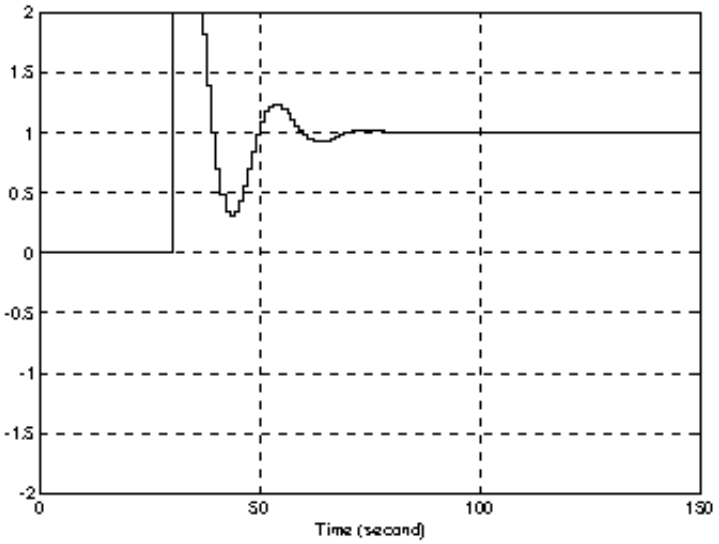
A more important aspect is the use of a summation to calculate the contribution of the integral term. This can lead to problems leading to a phenomenon known as integral windup, causing long periods of overshoots in the controlled response. This can be caused by a poorly tuned controller or the controller output is overly constrained (for some safety reasons or inappropriately sized final control element say), or a combination of both factors.

What happens is this. Say the controlled process has a positive gain and a positive set-point change occurs. The controller will then try to reduce the error between set-point and output, which is initially positive. The integral component will sum these positive errors to generate the necessary integral action. An overshoot occurs, whereupon the errors become negative. However, the direction of the control signal will not change to compensate if the sum of previously positive error dominates, in which case the overshoot becomes prolonged. The direction of control action will change only when the contribution of negative errors cancels the accumulated positive errors sufficiently. This phenomenon is known as **integral windup** or **reset windup**

This phenomenon is illustrated in the following figures,

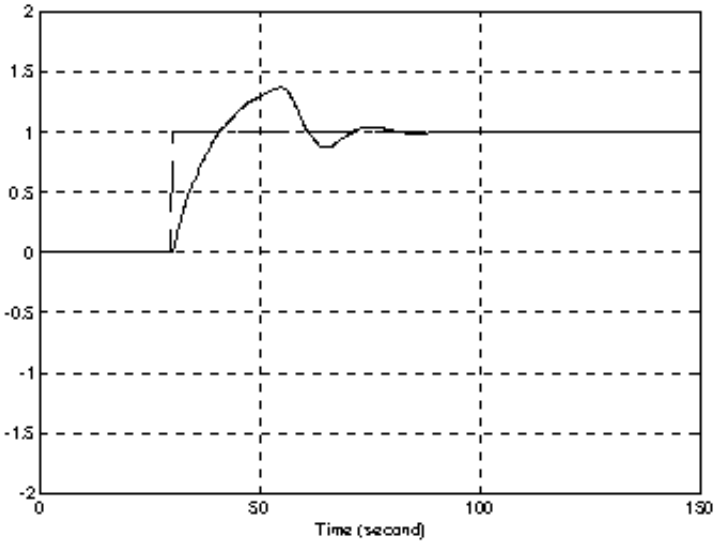


Output without control constraints

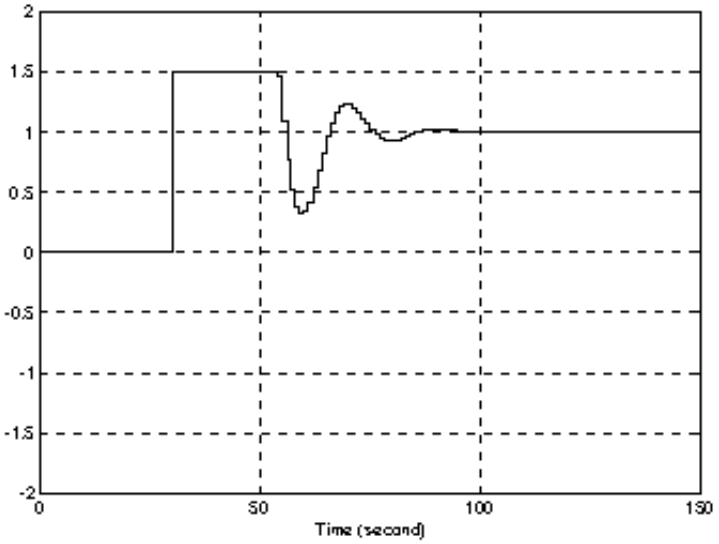


Unconstrained control signal

The following set of plots show the effects of integral windup, when the output of the controller is subject to a contraint.



Output with control constraints



Constrained control signal

With the velocity PID algorithm, because it does not make use of a sum of errors to generate integral action, the problem of integral windup will not occur, even if the control signal is constrained.

Controller tuning

All the tuning procedures for continuous time PID controllers apply. These range from the empirical Ziegler-Nichol ultimate gain method, to the recipe based integral of error criteria. With recipe based methods, a factor of $0.5T_S$ is usually added to the process dead-time to account for the delay caused by the sampler.

Back to top

Choice of Sampling Interval

Another important aspect in sampled data control systems is the choice of sampling intervals. With electronic controllers that emulate continuous time algorithms, this choice is simple: sample as fast as possible. This is because of the approximations that are used to generate the difference equations describing the controllers. Smaller sampling intervals mean that the properties of the underlying controller design will be less distorted, hence more predictable and better performances. A good example is the discretised PID controllers. They perform best when sampling intervals are small.

However, too fast a sampling is wasteful of resources.

- the cost of implementation will increase because more capable components must be installed
- a DCS typically has many hundreds of input-output channels to administer. The functioning of the DCS will degrade significantly if every control loop is to sample at the highest frequency possible.
- fast sampling intervals will mean that high frequency components such as noise will also be captured in the signal, and this is not necessarily beneficial to the performance of the control loop.

If the sampling interval is too long, then signal loss will occur. An extreme case is the phenomenon known as '**aliasing**'.

> **Signal aliasing refers to the situation whereby the sampled versions of two very different signals are indistinguishable.**

As illustrated in the figure below, the sampled representation of a step (dotted horizontal line) and a periodic wave are identical.

wpeE.gif (3794 bytes)

Many rules-of-thumb regarding the choice for sampling times for different types of loops have been published, including the following recommendations:

Flow loops	1s
Level loops	5s
Temperature loops	30s to 10 mins.

These are however, merely rough guidelines.

The choice of an appropriate sampling interval should be based on the dynamics of the process being controlled.

The sampling operation must return the key dynamic characteristics of the process. From experience, **a sampling interval of approximately 10% of the dominant time constant works well in practise**.

The position of the poles and zeros of the discrete transfer functions depend on the sampling interval used. Although this characteristic is not of great significance in PID type algorithms, it becomes important when discrete process models are used directly in the design of digital controllers. We shall cover this in more detail when we discuss model based digital controller design.

Back to top

Back to the Swot Shop

Please email errors, comments or suggestions to ming.tham@ncl.ac.uk.