Basic Control Algorithms

E. H. Bristol
The Foxboro Co.
Dept. 0346, Bldg. C41–1H
Foxboro MA, 02035
Tel. (508) 549-2019
Fax: (508) 549-4380

Abstract

Digital control provides many advantages over traditional analog controllers. These include greater flexibility to create and change designs on-line, a wider range of traditional control functions, and newer functions such as adaptation. But the digital computation is not naturally continuous like the analog controller. And it requires sophisticated support software. This article addresses the basic issues of carrying out continuous control in the digital environment, emphasizing the characteristics which must be addressed in the design of operationally natural control algorithms.

Introduction

Continuous process control traditionally used analog control devices, naturally related to the process. The modern era has largely replaced these with microprocessors and digital computations, to reduce cost, but also for greater flexibility, supporting more divers algorithmic forms. Initially digital control was implemented by converting each analog control function into some (perhaps generalized) digital equivalent. [Refs. 1–4]

This conversion requires an appropriate software architecture (more about that later) and the matching of the digital algorithms to analog behavior. Whereas the analog controller continuously sensed the process state and manipulated the actuators, the digital controller must repeatedly sample that state, convert it to a quantized number, use that number to compute control actions, and output those actions. Each of these steps involves its own problems and errors.

Standard digital control texts address the sampling problem thoroughly [Ref. 5], but treat the broader control design in terms of neutral, computed parameters not clearly related to the process gains and time constants. Process control depends on standard control algorithms whose parameters may be set or tuned in terms of known process properties. The approach defined herein emphasizes control with traditional parameters, naturally related to the process and practice. This includes compensating sampled algorithms and their tuning properties for the sampling time, or even for irregular sampling.[1]

One aspect of sampling should be understood: *aliasing*. This is a bizarre effect where sampled higher frequency data appear to be at a much lower, more significant frequency (Fig. 1). The same effect limits

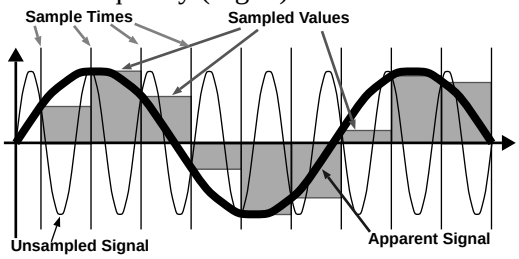


Fig. 1. Aliasing of Sampled to Apparent Signal

the minimum apparent width in time of any disturbance pulse to one sampling period. A control loop attempting to control such a false disturbance causes a real one instead.

Aliasing is not normally a problem because most commercial controllers sample frequently and filter out the still higher frequencies. But users should be aware of aliasing if they construct home brewed control algorithms of low sampling frequency[2].

Except for aliasing, the choice of sampling time is

[1] The resulting parameter forms may look unnecessarily approximate. But recent standards efforts[Ref. 6] have argued for even simpler control parameters. In either case it will always be possible to replace the proposed parameters by computations that support the more exact or simpler form. The chosen forms are based on the understanding that any use of digital control with analog process is inherently approximate, and that intuitive tuning of parameters is the most important design consideration of these approximations. Apparently formal calculations of the parameters will always be misleading.

[2] Model based control techniques, like Internal Model Control and Dynamic Matrix Control, are often constrained to operate at low sample times for reasons of modeling sensitivity.

not an issue today. Increasing the sample times faster than the *dominant closed-loop time constants* of the *economically important process variables* gives rapidly diminishing performance returns.[3]

For example, with end product quality the sole criterion on a process which takes an hour to respond, even fast flow loops can be sampled once every 5 min. Of course, many "housekeeping functions", like flow control, may have constraining side effects whose violation would involve real costs if this logic were actually implemented. Sampling times faster than 1 sec., well filtered, are rarely needed in continuous fluid process control, even when the local process dynamics are faster (as with flow loops).

The inexpensive microprocessor has caused a conservative design trade-off to favor faster sampling times, eliminating the issues where the cost is so small. Nevertheless, a 10 to 1 reduction in sampling time does correspond to a 10 to 1 reduction in computing resources, if one has the tools and imagination to use the capability. Faster sampling times also require that internal dynamic calculations be carried out to a greater precision.

Process control algorithm design also differs from academic treatments in respect to accuracy considerations. Practice designs for control and human operation, not for simulation or computation. When accuracy is important, a common thread through the discussion is the effect of differences between large numbers in exaggerating error, and the role of multiplication in creating these large numbers.

Rarely is the parameter or performance precision (say, to better than 10%) important to control, even under adaptation. While there are sensitive, high performance, control situations, a 2:1 error in tuning is often unimportant. In contrast, there are situations *within* an algorithm where a small error (say, a quantization error of 1 part in 100,000) will cause the control to fail. Computational perfection is unimportant; control efficacy is crucial.

A casual experimenter or designer of a one-use algorithm should be able to tailor a simple controller in FORTRAN floating-point without any special considerations. The normal debugging and tuning should discover any serious deficiencies. A commercial design, applied in many applications, calls for a deeper understanding of algorithmic and fixed- versus floating-point issues and trade-offs. Fixed-point and machine language programming may become a lost art, even though their difficulties are overstated. The advantages, in speed and exactitude, are worth consideration, particularly on small control computing platforms, where they may be essential. The problems are simply those of understanding, and of effective specification and testing of the algorithm.

The discussion thus addresses the refined design of process control oriented continuous control algorithms and their software support. It emphases high-quality, linear, dynamic algorithms. This still includes representative examples of discrete computation's effects on modelled continuous control activities:

- Imprecision, as it limits control performance,
- Quantization, as it artificially disturbs the process, and
- Sampling as it affects time continuous computations such as deadtime.

In this environment, nonlinearity serves to compensate a basically linear structure. One special concern, addressed later, is the back calculation required for some aspects of these compensations.

The user interested in the more general computation of nonlinear functions should consult the general computing literature [Refs. 7,8]. The discussion briefly touches issues relevant to advanced forms like adaptive control, but their details are outside the intended scope.

Structure of Traditional Process Control

The traditional process control structure is based on the combination of simple (PI/PID) controllers in simple single or cascaded loops, augmented with various non-linear, feedforward, and dynamic compensators, and constraint overrides.

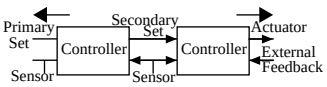


Fig. 2a. Basic Cascaded Structure

Figure 2a illustrates the basic cascaded structure. In this structure, A primary controller, controlling a corresponding process variable, manipulates the setpoint to a secondary controller which acts to stabilize and control the corresponding secondary process variable.

It will be noticed that each controller has two con-

[3] However, slow sampling frequency also causes tuning sensitivity, even for frequencies fast enough to give good control. Further doubling or tripling the frequency beyond this point should eliminate even this problem.

nections on both input and output side. On the input side these serve as the standard setpoint and measurement connection. The output side includes the normal output connection, as well as the external and status feedback connections, described later, whose purpose is to adapt the control to loss of output control.

The structure can be continued with any number of controllers, defining a degree of freedom path of controlled variables. The control of the primary variable is then supported by successively more secondary variables until the ultimate control falls on the actuator. The single degree of freedom represented by that valve is then passed back along that path until it is the primary variable which is effectively free for manipulation.

The degree of freedom path can be overridden by any number of constraint controllers or limiters. Figure 2b illustrates the constraint structure. The selector symbol (>) represents a computation which outputs the greater of two inputs, thus imposing a low limit on the original degree of freedom path.

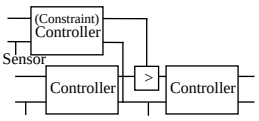


Fig. 2b. Constraint Structure

The constraint controller acts in feedback to set that limit value which will ensure that its measured value stays within the constraint limit defined by its setpoint. When the constraint controller takes over to enforce the constraint, it takes over the degree of freedom path, so that the path now runs from the constrained process variable through the secondary variable to the actuator. Any number of low or high limit or constraint override structures could similarly be included within the cascaded structure.

The traditional structure allows each controller in a cascade/constraint structure to be augmented in a variety of ways that enhance the control function without changing the underlying intent and structure. The figure shows the three distinct structural roles of this kind of compensation, with several variants:

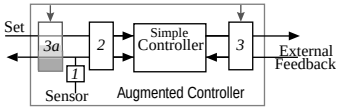


Fig. 2c. Augmented Controller

1. Computation of the desired measurement form from the sensor signal.
2. Linearization compensation of the controller in terms of the controller input (applying the some computation to setpoint and measurement without changing their operationally displayed form).
3. Linearization compensation of the controller at its output, which requires the application of the compensation to the output, the inversion of that same compensation (as described later) into the external feedback, and the appropriate back passage of any status information. A modified form of this structure is needed to support feedforward control, where the disturbance signal (with any dynamic compensation) is brought into the compensating calculation as a kind of computing parameter.[4]

3a. Partly for historical reasons, ratioing control is usually included as an operationally accessible setpoint parameter multiplying a feedforward measurement to generate the true controller reference input.

Later discussion will show branching out of the degree of freedom paths, matching the constraint branching together. The collective structures allow the implementation of all normal single loop control. And each of the structures has multivariable counterparts.

Number Systems and Basic Arithmetic

Operation on analog data involves not simply converting to decimal numbers, but also converting to the format that those numbers take in the digital computer. There are two such formats: fixed-point for representing integers, and floating-point for representing "real" numbers. Even though analog data is usually conceived of in terms of real numbers, the most effective processing of continuous control data, particularly for small microprocessors, is in fixed-point arithmetic.

In this case, the data must be (painfully) scaled in the same way that analog control systems and simulations were scaled. Scaling and multiple-precision computations and conversions are the explicit responsibility of the control algorithm programmer. Apart from computational necessity, scaling is an art which is inherently significant to proper control design, since the process is itself "fixed-point". Meaningful control actions can only be guaranteed if the user is aware that the scale of the calculations are in fact appropriate to the process.

Floating-point data is automatically scaled, in that it

[4] A later distinction will be made about the bumpless application of tuning parameter changes. The feedforward signal will usually be intended to incorporate both any appropriate parameter change and the compensating disturbance bump!

has a fractional part *f* or mantissa, corresponding to the integer part of fixed-point data, and an exponent part *e*, which defines an automatically adjusted scaling factor equal to a base value *b* (usually 2) raised to that exponent. Although the fractional part is usually viewed as a fraction, the discussion will be less confusing if *e* is chosen so that *f* is an appropriate integer; any such value then expressible entirely in terms of integers, as $f \times b^e$.

In the past, floating-point formats were not standardized [Ref. 9]. Moreover, the floating-point format inherently involves arbitrary truncations and uncertain relationships between single- and double-precision computation. For this reason, fixed-point algorithms still permit the most precise designs.

Fixed-Point Format

A fixed-point format represents an integer as a binary number stored in a *word* containing a fixed number (*n*, below) of *bits*, usually in what is called *two's complement* format. This format represents positive numbers in a range of 0 to $2^n - 1$ and signed numbers in a range of -2^{n-1} to $2^{n-1} - 1$. In two's complement arithmetic, negative numbers are represented in binary notation as if they were larger positive integers. As a result, when the numbers are added, the natural additive truncation achieves the affect of signed addition.

Thus, in Fig. 3, (with *n*=3) negative 2 is represented

Sign and Value	Two's Complement	
Decimal	Binary	Positive Decimal Equivalent
0	000	0
1	001	1
2	010	2
3	011	3
-4	100	4
-3	101	5
-2	110	6
-1	111	7
(2)+(-2)	1000	(3)+(5)=8

Carry bit

Fig. 3. Two's Complement Arithmetic

V	V.0	V.1	V.2
123456	12	34	56

Fig. 4. Decimal Multiple-Precision

by a binary number corresponding to a positive 6. And 6 added to 2 becomes an 8 which, with truncation of the carry, corresponds to 0. Thus the "6" is a perfectly good negative 2.

Two's complement arithmetic is also related to modular or "around the clock" arithmetic. Because of this behavior, results too large to fit into a word must be taken care of in one of three ways:

- Data values can be distributed in a sufficiently large set of data words to represent them. The arithmetic hardware or software then utilizes carry bits to pass the data between words when a carry is required.
- Results too large to fit a word may be saturated[5].
- One may ignore the problem in parts of the calculation where it is clear that results will never overflow[6].

The remaining discussion will be framed in decimal arithmetic, rather than in the unfamiliar binary arithmetic, for clarity's sake. Sufficient to say, actual designs are implemented by good fixed-point hardware including the basic binary operations and carry/overload bits to support effective single- and multiple-precision arithmetic. For this reason, multiplication and divisions of powers of two will be particularly efficient, to be preferred when there is a choice.

Fixed-Point Scaling

In the discussion of multiple-precision and scaling, it is convenient to distinguish different tag ends to the variable name to represent different aspects of the variables under consideration. Thus, if *V* is the name of a variable:

- *V.S* will become the scaled representation of the variable, taken as a whole.
- *V.0, V.1, V.2, …, V.m* (or *V.S.0, V.S.1, V.S.2, …, V.S.m*), will be different storage words making up the multiple-precision representation of *V* (or *V.S*), with *V.0* being the right most word and *V.1* being the next right most word, etc., as shown in Fig. 4.
- When *V* is scaled by a fixed-point fraction the numerator will be *V.N* and the denominator will be *V.D* [either of which may be itself in multiple-precision form (with *V.N.*0, *V.N.1, …,* or *V.D.0, V.D.1, …*)]:

$$V.S = \frac{V \times V.N}{V.D}$$

Normally the actual scaling computations will take place only when the data is processed for I/O or display; control calculations will generally take place with respect to *V.S*. For this reason, when no confusion, arises, the discussion will refer to *V*, substituting for *V.S*. Scaling conversions, like binary and decimal arithmetic and conversions, are straightforward though tedious; they need be addressed further.

The special power of this notation is that it allows all scaling and multiple-precision computations to be ex-

[5] That is, a value too large to fit the word is replaced by the largest number of the right sign which will fit.

[6] Where they will never become too large or too small to fit the word. This is a dangerous approach.

pressed entirely in conventional algebra. In particular, for analytical purposes, a multiple-precision value can be expressed and operated on simply as a sum of the normal values *V.0, V.1, V.2, …, V.m,* each with its own scaling factor:

$$V = V.0 \times B^{m} + V.1 \times B^{m-1} + V.2 \times B^{m-2} + \ldots + V.m$$

The value B is one greater than the largest positive number represented in a single-precision word ($B=2^n$ for unsigned binary data, $B=2^{n-1}$ for signed data).

Control algorithm parameter scaling is chosen according to need, computational convenience, and best use of the available data range, working with single-precision representation as much as possible. For a controller measurement or value on a 16-bit word machine, 1/2% precision is normally minimally adequate. For positive data a byte represents a data range 0 – 255 (2^8-1), which is better than 1/2% precision.

On the other hand, with a 16-bit word there are 8 more bits in the word which could be used to give a smoother valve action and simplify calculations. Working only with positive values, 16-bits corresponds to the normally unnecessary precision of 1 part in 65,535; with signed data, 1 part in 32,767; or, with a 1-bit safety margin, 1 part in 16,384. This scaling is well above the minimum process data precision, while still not forcing multiple-precision arithmetic in most cases. The corresponding 14-bit analog-to-digital (A/D) and digital-to-analog (D/A) converters for the process data are still reasonable.

Control parameters may have different natural data ranges or scalings. A controller gain might be scaled so that the minimum gain is $^1/_{256}$ and the maximum gain is 256. In this way the range of values equally spans high-gain and low-gain processes about a nominal unity gain. Time constants may require a certain resolution. When a minimum reset time of 1 sec. is adequate (1 hour corresponds to 3600 sec.) 2^{16} corresponds to more than 18 hours.

On the other hand, using the full range of data storage for the control parameters may require arithmetic routines which mix signed and unsigned arguments. At this level, there is a trade-off between the increased efficiency and the added complexity. Certainly it is more convenient to program uniformly in signed fixed-point (or even floating-point) arithmetic. But the costs of this convenience are also significant.

Range and Error in Fixed-Point Arithmetic

Good design for quantization and multiple-precision avoids poor control arising from inaccuracies not inherent in the real process. Normal control algorithms involve the standard combinations of additions, subtractions, multiplications, and divisions; rational functions of their data. Often the basic calculation allows many ways to order these calculations which are theoretically identical in their result as long as precision is indefinite.

Practical fixed-point programming requires a more careful understanding of the basic operations and the effect of ordering on the calculation. First, one should examine how each operation affects the worst case range[7] and error accumulation of the result. As will be seen, range effects are typically more important.

Error can be considered in terms of absolute error, i.e., the actual worst case error, or relative error, i.e., the worst case error as a percentage of the nominal value or scale. Usually the relative error is important in final results and products (or quotients), whereas the absolute error is important in determining how the error accumulates in a sum.

When two numbers are added or subtracted, their worst case range doubles, either requiring a carry (into a multiple-precision result), or requiring a sign bit[8] (See Fig. 5a). Additions and subtractions also cause the absolute error to increase. Addition of numbers of the same sign can never increase the relative error over the worst error of the added numbers. But their subtraction increases it, as does addition of numbers whose sign is uncertain. In Fig. 5b[9], two 10% relative error numbers, when added, result in a 10% relative error. But when subtracted, the relative error

$$\begin{aligned} 99 + 99 &= 198 \\ 99 + (-99) &= 0 \\ 99 - 0 &= 99 \\ 0 - 99 &= -99 \end{aligned}$$

a. Range Effects

$$(20 \pm \underline{2}) + (10 \pm \underline{1}) = (30 \pm \underline{3})$$

$$\frac{(20 \pm \underline{2}) + (10 \pm \underline{1})}{20 + 10} = \left(\frac{30}{30} \pm \frac{\underline{3}}{30}\right)$$

$$\frac{(20 \pm \underline{2}) - (10 \pm \underline{1})}{20 - 10} = \left(\frac{10}{10} \pm \frac{\underline{3}}{10}\right)$$

b. Error Effects

Fig. 5. Addition and Subtraction

---

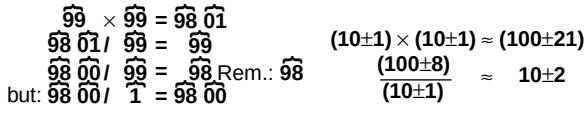
[7] The range of result for all possible combinations of the input data.

[8] When positive numbers are subtracted.

[9] In Fig. 5b the underlined numbers represent an error term being added or subtracted from the "ideal" value of a computational input or output.

can be arbitrarily large (30%) in this case. In this way, differences between large numbers explain most computing accuracy problems.

Fixed-point multiplication does more than just increase the range of the result; it doubles the required storage of the result (See Fig 6a). For this reason,



a. Range Effects b. Error Effects

Fig. 6. Multiplication and Division

most hardware implements single-precision multipliers to return a double-word result. By analogy, most hardware division divides a double-precision dividend by a single-precision divisor to obtain a single precision quotient, with a single-precision remainder. But as Fig. 6a shows, such a division can still give rise to a double-precision quotient. The hardware expresses this as an overload (similar to division by zero).

Fixed-point hardware is designed to permit the effective programming of multiple-precision arithmetic. One of the strengths of fixed-point arithmetic, with its remainders, overflow and carry bits, and multiple-precision results is that no information is lost except by choice. The precision, at any point in the calculation, is entirely under the control of the programmer. This is not true of floating-point arithmetic. As shown in Fig. 6b, multiplication and division always increase (e.g. double) relative error.

Fixed-Point Multiplication and Division

Multiplication's range explosion is its most problematic aspect. Among other consequences, it generates large numbers whose differences may cause large errors. It makes smaller numbers still smaller, compared to errors caused by other large numbers. The important issue is the ratio of the large numbers (which make the errors) to the small numbers (which end up as the result). Multiplication's range expansion forces a choice: Precision can be preserved, or intermediate values can be rescaled and truncated back to their original data size.

Thus, it is usually desirable to avoid repeated multiplications. Often multiplications and divisions occur together in a manner in which they can be alternated, the multiplication generating a double-precision value and the division returning that value to a single precision quotient. The basic proportional controller calculation is a good example:

$$Output = \frac{100 \times error}{ProportionalBand} + bias$$

Common combined operations, such as this, may usefully call for specially designed routines. In general, one should try to maintain a constant data range and size throughout the computation. Often a product is naturally intended to return a value in the same range as a process variable (output). There are two natural cases: either a value is multiplied by a gain or by a proper fraction (Fig. 7). In the controller calculation the 100/*ProportionalBand* gain might be limited to the range 1/256 to 256. If the gain is expressed as a single-precision integer (with decimal point effectively in the middle of its digits) rather than as a fraction one can still generate an appropriate single-precision result by taking the middle single-precision set of digits out of the double-precision scaled result (Fig. 7a), saturating and truncating the extraneous data.

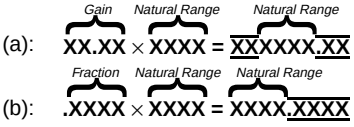


Fig. 7. Scaled Multiplication of (a) Gain, and (b) Fraction

As a common case of proper fractional multiplication, a lag calculation (time constant *T*) can calculate its output *X* as a weighted average of the past output *Y′* and the current input *X*:

$$Y = \frac{1}{T+1}X + \frac{T}{T+1}Y' = Y' + \frac{1}{T+1}(X - Y')$$

In this case, when the proper fractions are multiplied (scaled as integers), the proper result can nevertheless be returned as the left most part of the double-precision result (Fig. 7b). It is often appropriate to combine constants or parameters together into a common effective parameter which is the above kind of gain or proper fraction. This ensures that the linear operation with the process data is truncated by only the one final multiplication.

The combined parameters can be made to act consistently on the data even if in error as calculated; the errors can be reinterpreted as (presumably insignificant) errors in the original parameters. Note that the right most expression within the above equation in-

volves a feedback between the *Y* and the difference between *X* and *Y'*. Often such a feedback can be included in the calculation to improve an otherwise error-prone algorithm, making it self-corrective, like any other feedback system.

There is a parallel between the above range discussion and traditional dimensional analysis [Ref. 10]: Sums must be of data elements with identical units, and similar ranges. Multiplications change the units, and also change the range. The ultimate purpose of any of our calculations is to convert data of limited range from the process to data with limited range to drive the process. Thus, whatever the internal gyrations, the process constrains the results to be reasonable. The challenge is to carry out the calculations so that the inherently limited practical range prevails throughout the calculation.

Digital Integration for Control

Fig.8 shows a different kind of division problem: process control integration. The object is to integrate the process error in a control (reset) calculation. This

$$\sum_{i=0}^{n} \frac{Error(i)\,\Delta t}{T} =$$

$$\frac{\sum_{i=0}^{n} Error(i)\,\Delta t}{T} = \frac{\sum_{i=0}^{n-1} Error(i)\,\Delta t}{T} + Error(n)\,\frac{\Delta t}{T}$$

$$= \sum_{i=0}^{n-1} \frac{Error(i)\,\Delta t}{T} + Quotient\left(\frac{Error(n)\,\Delta t + Remainder}{T}\right)$$

Note:

$$Error(n)\,\Delta t = T \cdot Quotient\left(\frac{Error(n)\,\Delta t + Remainder}{T}\right)$$

Fig. 8. An Integration Trick

can be done by computing a shared fractional multiplier $\Delta t/T$, which has been scaled to give a good range [which might also include the Proportional Band action *PB*, as in $100\times\Delta t/(PB\times T)$]. This result would be multiplied by the error and added in double precision to the previous "integrated" (i.e. summed) value to get the current control value.

Double-precision is essential here, since a large value of *T* corresponds to a small $\Delta t/T$ and a truncation loss of significant process errors. For example, suppose that the error and sum are both scaled as signed single-precision integers, ranging –10,000 to +10,000. With Δt =1, and the minimum T = 1 (corresponding to 1 sec.), the corresponding maximum $\Delta t/T$, which equals 1, must be scaled to a value 10,000. The natural scaling of the controller output can be achieved by dividing by 10,000. (See later scaling discussion). That is, the scaled equations should be:

$$Sum.S = oldSum.S + \left[\frac{(\Delta t/T).S \times Error.S}{10{,}000}\right]$$

In normal single-precision division, only the integer quotient is considered. Thus, a product $[(\Delta t/T).S\times Error.S]$ less than 10,000 is truncated as zero: the error is ignored and the integration stops, causing a permanent offset. For *T* large, equal to 10,000 (which corresponds to about 3 hours), and $(\Delta t/T).S$ small (equal to 1.0 in this case), any error less that 100% (scaled to 10,000) will be lost. Under control the result is a 100% offset. Double-precision alleviates the offset but still loses some information to truncation.[10]

The better method, shown in Fig. 8, preserves the division and achieves an exact result with the same storage as double-precision. In this case, the product of the error and the sample time (the sample time may equal 1) is formed as a double-precision value, then added to the remainder from the previous sample calculation's division. This net value is divided by *T* to get back a single-precision quotient to be added to the old sum, and a remainder to be saved for the next sample time. Data results truncated from any quotient are never lost, but preserved and accumulated in the remainder, to show up in some later quotient.[11]

---

[10] One alternative to multiple-precision integration uses random numbers (or dither, in mechanical engineering terms). The computation is carried out in normal double precision. However the higher precision (lower order) data word is ignored and not saved; it is replaced by a random number in the next sample time's calculation when it is needed again. This method works well practically and theoretically. A similar method was incorporated into the Digital Equipment Corp. PDP-1 floating-point package. But who would have the courage to use it on a real process?

[11] As pointed out in the next section, floating-point lacks the support for such refined tricks. It is particularly subject to integration problems because a large sum may be big enough to prevent the addition of a small term even though that term is rescaled to avoid its being truncated to zero.

Floating-Point Format

Modern microprocessors are often supported with high speed floating-point processors. Without these, floating-point calculations run an order of magnitude slower than fixed-point calculations. With the floating-point format, all remainders, carries, and multiple-precision results of single-precision computations disappear. Instead, the user chooses a fixed level of precision whose truncations show up in *guard bits* [Ref. 9].

Floating-point errors introduced in small differences of large numbers become more severe, and at the same time less obvious because their processing is automated and invisible. For example, when a large number L (e.g. 10,000, stored to three digit accuracy) is added to a small number S (e.g. 1), the smaller number totally disappears. The sum $L - L + S$ should equal S. If the calculation is carried out in the natural ordering, $(L - L) + S$, then S will result. But the nominally equivalent $(L + S) - L$ will generate zero.

There are a number of intricate ways of avoiding this problem:

- Convert all floating-point numbers to ratios of integers and operate in the fixed-point format. This is one way to study the properties of the algorithm.
- Use adequate precision. Practically this is often unpredictable, and theoretically it is impossible because repeating decimals require infinite data. This suggests also deferring division to the last operation.
- Reorder the calculation for the most favorable computation, either statically based on algorithm properties or by using an arithmetic package which continually reorders the operands. It is useful, in the following discussion, to consider every list of terms to be added (subtraction being taken as addition of a negated number) to be ordered by magnitude. The individual operations are carried out according to these rules:

  Under addition or subtraction:

  - Combine small numbers together before large numbers (to let them accumulate before being lost).
  - Take differences (subtractions or additions of numbers of opposite sign) before sums (to achieve all possible subtractive cancellation between larger numbers before these can loose the small numbers).
  - From a list of numbers to be multiplied and divided, cancel or divide out most nearly equal numbers first (to minimize floating-point overload[12]).
  - When all denominator terms are gone, multiply largest with smallest numbers (to minimize floating-point overflow).

One other advantage of following such a set of ordering rules is that it will give identical results to identical data even when they originally occurred in a different programmed order.

Generalized Multiple-Precision Floating-Point

Normally, the multiple-precision floating-point format is the same as single-precision with larger fraction and exponent data fields. The author has experimented with a more open ended multiple-precision floating-point, illustrated in Fig. 9. The multi-

$$12340.36789 \Rightarrow 1234\times10^{1} + 3679\times10^{-4} - 1000\times10^{-8}$$

Addition:

$$A=3057\times10^{6} \qquad B=4263\times10^{4}$$

$$A + B = 309963\times10^{4} = 3100\times10^{6} - 3700\times10^{2}$$

Fig. 9. Generalized Multiple-Precision Floating-Point (M=4)

ple-precision floating-point number is represented by a summed, ordered set of signed single-precision floating-point numbers, each mantissa having M digits.[13]

The numbers in the set are chosen so that the set can be truncated at any point to give the best possible truncated approximation. In this case, the value is considered to be made up of the truncated value and a signed (±) remainder which expresses the part cut off in truncation.[14] The generalized floating-point would be supported by the following operations:

- Text Conversion, converting to or from text general precision floating-point to the internal format consisting of the summed set of single-precision values.

---

[12] The process of the floating-point exponent getting too large for the provided data space.

[13] Analogous to multiple-precision fixed-point data represented as a summed scaled set of single-precision numbers. Each element is normalized (shifted), to use the full range of M digits. Recall that the general representation of a single-precision floating-point number is: $f\times b^{e}$, with f, e, and b integers. As before, the format is described in terms of a general b, and illustrated with $b = 10$. In practice b will equal 2. Note that the different members in the summed set may have different signs! However the sign of the total (set) value is still the sign of its initial and largest element.

[14] The remainder magnitude is always less than half the units value of the next larger element in the set, since the best approximation requires that the remainder round to zero.

- (Set) Normalization, reprocessing the set of single-precision values so that they are ordered in magnitude, with largest first[15], so that the magnitude of each mantissa is between than b^{M-1} and b^M, and so that consecutive values have exponents whose difference is greater than M (or, if the difference equals M[16], then the magnitude of the second mantissa is less than or equal to $b^M/2$).[Ref. 11]
- Addition and Subtraction, merging entries into a final value set followed by a normalization of that set.
- Multiplication, multiplying every pair of elements, one from each of the multiplicand and multiplier, to generate a double-precision result, followed by the merging and normalization of the accumulated set of results.
- Division, dividing the largest elements in the dividend by the divisor, subtracting the divisor multiplied by the quotient from the dividend to obtain the remainder. This division is designed to select that quotient which returns the remainder with the smallest magnitude (of whatever sign). The remainder can be redivided by the divisor to compute any desired level of multi-precision quotients, with the final resulting quotient and remainder being normalized. The remainder so developed can be used in the earlier integration procedure.

Such a generalized floating-point can be used to develop calculations whose precision expands indefinitely as needed. Such a system could give absolutely error free results, without any special care.

Specification of Fixed-Point Algorithms

Clear fixed-point specification includes the proper statement of computation order, and of scaling of intermediate and final results. This can be superposed on a conventional algebraic notation. For example, in the following control computation, the parentheses define any required ordering:

$$Output_5 = \left(\left(\frac{100 \times Error_1}{ProportionalBand_2} \right)_4 + Bias_3 \right)_5$$

Unparenthesized addition and subtraction is assumed to be from left to right, and multiplication and division are assumed to alternate as described earlier. Any constants which can be combined would certainly be combined in a working system. This should include any scaling constants. The subscripts refer to scaling specifications in Fig.10 table below.

[15] In form: $f_0 \times b^{e_0}$, $f_1 \times b^{e_1}$, $f_2 \times b^{e_2}$, … .

[16] Not possible after normalization, if M equals 2.

Such a table would completely specify all scalings,

Subscript	I/O	Saturation Hi	Saturation Lo	V.N	V.D	Sign	Precision
1	IN	100	-100	16384	100	±	1
2	IN	16384	0	2	1	+	1
3	IN	100	-100	16384	100	±	1
4	–	YES		16384	200	±	1
5	OUT	100	0	16384	100	+	1

Fig. 10. Table of Variable Scaling Related Properties

saturations, and conversions appropriate to the values within the calculation. Remembering that the internal scaling is reflected in the relation:

$$V.S = \frac{V.N \times V}{V.D} \quad ,$$

the conversion from functional algebraic equations to internal computational form is then carried out by the computation:

$$V = \frac{V.D \times V.S}{V.N} \quad .$$

When all of these scalings are incorporated back into the original proportional controller calculations, its internal scaled form becomes:

$$Output.S \times \frac{100}{16384} = \frac{100 \times \frac{100}{16384} Error.S}{\frac{1}{2} \times PropBand.S} + \frac{100}{16384} \times Bias.S$$

or:

$$Output.S = \left(\left(\frac{200 \times Error.S}{PropBand.S} \right) + Bias.S \right)$$

Such a simplification is to be expected in practical calculations as part of the combination of similar scaling terms and application constants. The final equation then becomes the programmed calculation, except that each operation would be saturated as specified. Saturation can, in fact, make the combination of terms invalid, but in this case, it may be worth considering whether or not the saturations might not better be left out, in the interests of a more perfect result.

Definition and implementation of an algorithm then has three parts:

- Specification and programming of the necessary arithmetic and saturation routines,
- Manual development of the scaled calculation, in terms of the routines.
- Programming the algorithm.

Ideally, an algorithm should be tried out first in a higher level language (e.g. FORTRAN or C). Here it

can first be expressed in floating-point and then in scaled fixed-point. If it is later needed that the final form be in machine language, the three different forms can be run comparatively, greatly facilitating debugging.

Operational Issues

The basic controls are normally expressed as linear algorithms, defined as if the process measurements and actuators were capable of perfect operation to whatever range might be needed. In fact, valves limit and sensors fail. The algorithms must be designed to accommodate temporary valve saturation or loss of sensor data. Moreover they must be designed to allow the system to be restarted smoothly after longer term failure or shutdowns.

The most common such problem is windup: the property of the controller which continues to integrate under error even after the actuator has limited and is incapable of further change. Windup requires recovery time even after the error reverses sign, blocking effective control, for that interval. The response to this problem requires that the algorithm be provided some indication of the limiting so that it can alter its behavior. The information can take the form of flags which inform the controller of the limiting actions being taken, propagated back to any affected controller. The flag then causes the integration to stop, in some appropriate way.

A superior approach, called external feedback, senses the actual state of the manipulated variable and feeds it back, in comparison to its intended value. By working with the true state of the process, the algorithm can make much more refined accommodation strategies. Controller windup is not the only problem arising from actuator limiting: Blend pacing, split range control, multiple output control all relate to standard affects of valve limiting and its accommodation. The external feedback strategy effectively unifies the handling of all of these issues.[17]

The problem is complicated because the effects of limiting must propagate back from the actual valve, through any control functions (e.g. cascaded controllers, ratio units) to the controller under consideration. Thus, there is not only a need to notify controllers of valve limitings or sensor failures but to propagate this information to all affected control elements. Software must be designed to accomplish the propagation.

The external feedback approach is particularly advantageous because it recognizes loss of control locally to the affected controller, and responds only when the loss is material to it. Of course, the loss of control at any level may be a useful basis for alarming, independent of immediately recognized effects on control. The present discussion is largely limited to the algorithmic consequences of the problem, but the software consequences are just as important.

A common problem with ad hoc controller designs is that they bump the output whenever the parameters are changed for tuning. The PI controller computation below illustrates the problem:

$$O(t) = \left(Error + \frac{1}{\tau}\int_0^t Error \bullet dt\right)\frac{100}{PB}$$

$$= \frac{100}{PB}Error + \int_0^t \frac{100}{PB \cdot \tau}Error \bullet dt$$

The first of two nominally equivalent expressions computes the output with the tunings acting after the integration. The practical result is that any change in settings immediately bumps the output. In the second expression, the tunings act on the *Error* value, before integration. Any tuning changes only affect integrated errors occurring after the change.

Apart from output limiting and tuning bumps there are a number of similar operational modalities which the controller should support, either in relation to manual or automatic operation. For example:

- Cold Start Initialization. Sometimes it is desirable as part of the process start-up to start the controller, in automatic mode, so that its output has no tendency to move, letting the operator move the process on his own schedule. This is also the natural way to initialize secondary controllers when the primary controller goes to automatic mode. Cold Start can be implemented by setting the controller setpoint to the present measurement value and initializing the internal states of the controller so that the output matches the externally fedback or operator set value of

[17] A full accommodation of actuator or cascaded control loss would include both external feedback and the flags because some control functions do not lend themselves to the external feedback solution. For example some adaptive controllers depend on free response to their output to support meaningful adaptation.

the output. Automatic control then continues naturally.

- Bumpless Transfer. The purpose of this mode is to transfer control, to automatic control, in such a way that the process does not receive any immediate valve change, but moves from its prior position smoothly. In this case, the controller setpoint is left in place, but all internal states are initialized to be consistent with the current error and external feedback (and current controller output value).[18]
- Batch Preload. In circumstances where a known setpoint change is to be applied (for example, in batch production) the controller may be set up to pick up on the setpoint change with a preset initial internal integration value. The purpose is to give the process an initial kick, to get it to the setpoint in the fastest time. This strategy has a number of elaborations of varying sophistication.
- Ramped Setpoint Change. The controller may be designed to limit the rate at which it responds to setpoint changes, to minimize the bumps to the process.

The variations on cold start and bumpless transfer depend on back calculation: the recomputation of internal data[19] to be consistent with the unchanged output and external feedback. For example: a PID computation developed later computes its output O from the collective effect X, of the proportional and derivative effects, and an internal bias B. The bias is computed in turn by applying a lag computation to the external feedback O_{FB}.

$$O(t) = X(t) + B(t)$$

$$B(t) = \frac{O_{FB}(t) \cdot \Delta t + B(t - \Delta t) \cdot \tau}{\tau + \Delta t}$$

The bumpless transfer can be accomplished by computing the new value of X, and then rearranging the first equation above to back calculate the bias from the new X and the old B:

$$B(t) = O(t) - X(t)$$

When the O is later calculated from this B its value will remain initially at its old value, irrespective of manual changes in O, or measurement, setpoint, or tuning changes reflected in X. The second equation might be used to back calculate O_{FB} but this value will be overridden by later computations anyway.

Output Limiting; External Feedback

As already indicated, external feedback represents a precise and smooth way of handling windup. However implemented in a PID controller algorithm, it has the effect of including, in the integrated term, the difference between the controller output and the corresponding externally fedback measured state. The difference is added to oppose the normal integration. Thus, whenever the external feedback fails to follow the controller output the difference builds up to stop the integration.

Feedback controllers are built about the processing of an error signal on their input. External feedback extends the principal to the output. It allows the control algorithm to be designed to alter its approach in the face of output failure to act. This same strategy can be generalized to apply to any control function:

- Blending. When several ingredient flows are ratioed to generate a blended product, two basic product properties are involved: the product quality and the product flow. Under the standard strategy of pacing, if one ingredient flow limits, the remaining flows are ratioed, not off their original target, but off the external feedback from the limited flow. Such a system extends the external feedback concept to give up the control of product flow, in favor of the more important product quality.
- Fuel/Air Ratio Control. With certain liquid fuels, an excess accumulation of fuel in the burners constitutes a fire hazard. The fuel controller is designed to limit the fuel to be less than the combustible ratio to the measured air flow, at the same time limiting the air flow to be greater than the combustible ratio to the measured fuel flow. If either limits, the other is held to a safe value. Such a system extends the external feedback concept to safety control.
- Multiple Output Control. In certain cases, a manipulated resource may be duplicated so that several devices share a load. It is desirable that operating personnel be able to take one or more of such devices out of service in such a way that the others take up the load. In this case, a multiple output controller computes an output value such that when the value is ratioed as the setpoint to each device, the sum of the external feedbacks from all devices equals the net desired load. In this way, as one device is taken over in manual, the others will take up the slack.
- Backup (Also Split Range Control). The normal external feedback anti-windup action can be extended, using the recognized output/external feedback error to drive other

[18] This mode can be applied to all controllers in a cascade, but because the secondary setpoints will then be matched by their primary controllers to their sensors, the result for them should be the same as if they had been initialized under a Cold Start. This is only true if the controller calculations are carried out in an appropriate order relative to each other. In lieu of this, it may be better to use the Cold Start on the secondaries anyway.

[19] Particularly integration data. This guarantees that the integration will resume as if the controller had always been operating under the current error and output conditions.

actuators. Fig. 11 below illustrates a refined handling of this function:

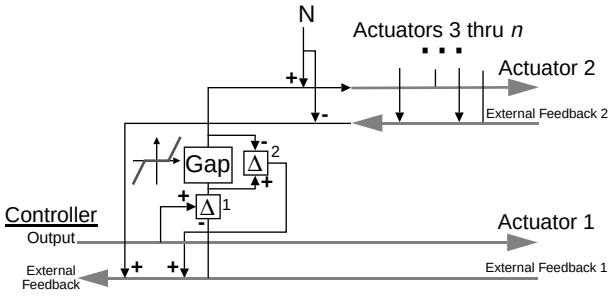


Fig. 11. External Feedback Based Backup

In this design, the Controller Output is normally connected to Actuator 1. But if the calculated difference (the subtraction element (Δ) marked 1) between Actuator 1 and its External Feedback, exceeds the Gap parameter, the excess difference will serve to transfer the active control to Actuator 2. In this way the controller can be backed up against any downstream failures or overrides. The nominal bias N reflects the preferred inactive value for Actuator 2 (In more refined designs N may be established dynamically by a separate controller.). Differences can further be passed on indefinitely to Actuators 3 through *n*. Effectively this arrangement generalizes the behavior of Split Range control, taking into account any downstream loss of control action.

The External Feedback 2 is added into the Controller External Feedback to allow the controller to continue its full (integrating) control action, whatever actuators are in fact acting. The purpose of the Gap is to ensure that actuator transfer does not give rise to a chattering between actuators but acts only for significant loss of actuator control. However, to guarantee that the resulting temporary loss of control does not cause the controller to stop integrating, the actual amount of gapping action (the Δ marked 2) is added into the Controller External Feedback as well; the external feedback sees neither Gap nor actuator transfer.

In actuality, the different actuators might call for different control dynamics and different compensation. This could be built into the control transferring paths. However, digital implementation allows the switching of controller tunings, as a function of the active actuator, to be carried out as part of the controller computation; a more natural arrangement. Digital implementation also allows the above structure to be black boxed flexibly, taking the confusing details out of the hands of the user.

- Linear Programs and Optimization. It has been argued that external feedback is incapable of dealing with connection to higher level supervisory functions such as linear programs or optimizers. This position reflects higher level functions not designed for operations, rather than any inherent problem with external feedback. The operationally correct optimizer will benefit from external feedback data like any other control computation. In this case, each optimization target, with its external feedback, is associated with an implied output constraint. Whenever a difference develops between the two, the constraint limit and violation becomes apparent.

  Thus the external feedback value should be fed into the optimizer, parameterizing a corresponding optimization constraint. There are three special considerations:

  - The control action actually implemented must push beyond the constraint so recognized. Otherwise, the constraint becomes a self-fulfilling prophecy which, once established, never gets retracted. Since the control action is presumed to be up against a real process constraint, it does not matter how much further into the constraint the target variable is set. However, it is probably better to exceed the constraint by some small number (e.g. 1 – 5% of scale).
  - The optimization computation is likely to be run infrequently compared to the normal regulatory dynamics. For this reason some lag filtering or averaging should be built into all external feedback paths to minimize noise effects and increase their meaningful information content. The filter time constant should correspond to the optimization repetition interval.
  - All of this assumes that the optimizer addresses the economic constraint dimensions only. Significant safety or quality constraint effects must always be separately addressed at the regulatory level.

With other control and operational nonlinearities, many issues come up, calling for many different kinds of thinking. Of course these same differences must fit nonintrusively and naturally with the intentions and expectations of operating people. While the operational user will normally not be aware of the technology behind these techniques, he will become intuitively aware of any inconsistencies between the handling of similar functions in different control elements. External Feedback provides a powerful strategy, for addressing many of these problems, whose application uniformity the end user will appreciate.

External Feedback in Nonlinear Compensators

In the introductory discussion of Figure 2c, the third form of compensation called for inversion of the associated nonlinear compensation. Traditionally this has been done with simple, analytically-determined inverses. For example:

- A squaring of the direct compensation called for a square

root to the external feedback; an exponential, a log.

- In feedforwards a subtraction of the feedforward signal inverted an addition; a division inverted a multiplication.

Historically, general inverting rules (like Newton's method) would not have been used for fear of failure to converge. Figure 12 shows a situation (the left

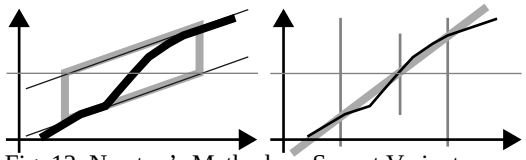


Fig. 12. Newton's Method vs. Secant Variant

graph) where Newton's method, based on extrapolation of the derivative of the function, could successively overshoot the solution of the function (the intersection of the function graph and the shaded horizontal line) on each side. But if the interpolation is always based on prior guesses which bracket the solution (as on the right graph) convergence is guaranteed. These kinds of methods can be generalized to invert multivariable functions, and find all solutions.[Ref. 12]

Basic Control Algorithms

The Lag Calculation

The lag calculation corresponds to the following continuous transfer function:

$$\frac{L(s)}{I(s)} = \frac{1}{(\tau s + 1)}$$

where L=output and I=input, and τ is the lag time constant. The usual practice of going to the z transform for the corresponding sampled-data form should not be overemphasized. No exact approximation is possible. Instead, the algorithms are designed primarily to avoid operationally unnatural behavior. With this in mind, the best direct sampled-data approximation of the above differential equation is:

$$I(t) = \frac{\tau \Delta L}{\Delta t} + L(t) = \frac{\tau}{\Delta t}[L(t) - L(t - \Delta t)] + L(t)$$

or:

$$L(t) = \frac{I(t) + \frac{\tau}{\Delta t} L(t - \Delta t)}{1 + \frac{\tau}{\Delta t}} = \frac{I(t)\,\Delta t + \tau L(t - \Delta t)}{\tau + \Delta t}$$

This approximation amounts to a weighted average of the new input and the old output. From the scaling point of view, each of the products has the same range as the sum; scaling is simple. The calculation is stable for all positive τ, accurate for large τ, and qualitatively natural as τ approaches 0, with $L(t)$ equaling $I(t)$ if τ = 0, as intended.

The calculation is usable even in single precision if the term $I\Delta t$ is truncated up in magnitude (and τ is not too large). In this case, the product is never truncated to zero unless the product is truly zero. This guarantees that the output will always settle out at any steady-state input value. Normal truncation would leave the result below its "theoretical" steady-state value, a situation similar to the integral offset described earlier.

But a trick, similar to the one used with the integrators before, can be applied to calculating lags exactly:

$$L(t) = quotient\left(\frac{I(t)\,\Delta t + \tau L(t - \Delta t) + remainder}{\tau + \Delta t}\right)$$

with the remainder being saved for use in the next sampled calculation.

Lead/Lag Calculation

Filtered derivative and lead/lag calculation are most easily and reliably developed from the above lag calculation, by analogy with transfer function calculations:

$$\frac{O(s)}{I(s)} = \frac{\tau \cdot s}{k \cdot \tau \cdot s + 1} = \frac{1}{k}\left(1 - \frac{1}{(k \cdot \tau)s + 1}\right)$$

Similarly:

$$\frac{O(s)}{I(s)} = \frac{\tau \cdot s + 1}{(k \cdot \tau)s + 1} = \frac{1}{k}\left(1 - \frac{1}{(k \cdot \tau)s + 1}\right) + \frac{1}{(k \cdot \tau)s + 1}$$

$$= \frac{1}{k} \times \left(1 + \frac{k - 1}{(k \cdot \tau)s + 1}\right)$$

or:

$$\frac{O(s)}{I(s)} = \frac{\sigma \cdot s + 1}{\tau \cdot s + 1} = \frac{1}{\tau}\left(\sigma + \frac{\tau - \sigma}{\tau \cdot s + 1}\right)$$

The translation to digital form consists of carrying out all of the algebraic steps directly and replacing the lag transfer function by the digital lag algorithmic calculation described in the preceding section. Considering the last form and proceeding in reverse order,

the output of a lead/lag would be calculated from the output of the lag calculation (with time constant τ):

$$O(t) = \frac{(\sigma I(t) + (\tau - \sigma) L(t)))}{\tau}$$

A basic filtered derivative can be calculated using a lag calculation, now indicated as L_D, and assuming, typically, $k = 0.1$ (In fixed-point, k would be a power of 2: 1/8 or 1/16.), and the lag time constant 0.1 τ_D:

$$D(t) = \frac{1}{k}(I(t) - L_D(t)) = 10(I(t) - L_D(t))$$

PID Controller Calculation

PID controller designs are expressed in many forms:

$$O(s) = (\tau_D s + 1 + \frac{1}{\tau_I s}) \frac{100}{PB} \bullet Error$$

$$O(s) = (\tau_D s + 1)(\tau_{DI} s + 1) \frac{100}{PBs} \bullet Error$$

$$O(s) = (\tau_D s + 1)(1 + \frac{1}{\tau_I s}) \frac{100}{PB} \bullet Error$$

Each of these forms is capable of the same performance as the others, with one exception: the first form is capable of providing complex zeros. There is no generally argued requirement for complex zeros, but this is nonetheless a real distinction.

There is also some disagreement as to whether the $1/\tau$ terms should be replaced by gains (or whether the proportional band terms should be combined into independent proportional, integral, or derivative terms. The reason for giving all terms as gains is that this then places the most stable setting for all terms at zero (not entirely true of derivative). This argument is pitched to operators. The reason for leaving the terms as above is that the time constants have process related meaning for engineers who understand the control issues; the separate Proportional Band then becomes a single stabilizing setting for all terms.

Different implementations also apply different parts of the algorithm differently to the Setpoint and Measurement terms within the Error. This reflects that these terms have different affects within the process. Ideally one would provide separate tunings for load and setpoint changes. A practical compromise is to apply all three actions to the measurement, but only the proportional and integrating action to the setpoint. The controller will then be tuned for load disturbances.

The above forms have a particular difficulty if the integrating calculation term is interpreted literally: The integrating term is most naturally translated digitally as:

$$\sum_{i=0}^{(t/(\Delta t))} \frac{\Delta t}{\tau} \cdot Error(i\Delta t)$$

However, the individual summed terms get unnaturally large when τ approaches zero. A practical way of bounding the value is to replace τ by $\tau+\Delta t$. When τ approaches zero, this still leaves an unnaturally large term for $i = t/\Delta t$, in competition with the proportional term. The solution is to replace $(t/\Delta t)$ by $(t/\Delta t) - 1$:

$$\sum_{i=0}^{(t/(\Delta t))-1} \frac{\Delta t}{\tau + \Delta t} \cdot Error(i\Delta t)$$

[The summation (integration) can be carried out according to the earlier discussion.] The result can be justified in another way: Consider the last PID form introduced above:

$$O(s) = (\tau_D s + 1)(1 + \frac{1}{\tau_I s}) \frac{100}{PB} \bullet Error$$

The differentiation must be carried out lagged or filtered, as previously described. If all but the integrating calculations are calculated as a combined result X (taking into account any separation of the treatment of setpoint and measurement), the result is:

$$O(s) = (1 + \frac{1}{\tau_I s}) X(s) = X(s) + \frac{X(s)}{\tau_I s}$$

This has an alternative formulation, which introduces a calculated bias B, particularly convenient for implementing the external feedback:

$$O(s) = X(s) + B(s)$$

$$B(s) = \frac{O_{FB}(s)}{\tau s + 1}$$

where O_{FB} is the external feedback term, nominally equal to O. When this pair of transfer functions is

translated to an algorithm, they become:

$$O(t) = X(t) + B(t)$$

$$B(t) = quotient\left(\frac{O_{FB}(t)\cdot\Delta t + B(t-\Delta t)\cdot\tau + remainder}{\tau+\Delta t}\right)$$

The collective effect of this calculation corresponds to the algorithmic expression of the more direct PID form with the modified integration proposed above:

$$O(t) = X(t) + \sum_{i=0}^{(t/(\Delta t))-1} \frac{\Delta t}{\tau+\Delta t} X(i\Delta t)$$

As indicated earlier, external feedback must be added to this form by subtracting the difference between output and external feedback from the $X(i\Delta t)$ term before integration.

Dead Time Calculation

The deadtime calculation corresponds to the following continuous transfer function:

$$e^{-Ts}$$

Deadtime represents a black box whose output exactly repeats the time form of its input, but delayed by some amount in time. It represents the behavior of state variables of product, input into a pipe or onto a conveyor belt, and output (delayed) at the other end. It is an essential modeling element for typical processes.

Deadtime is conventionally approximated in two ways: through Padé (continued fraction) approximations of the above transfer function, and through what is called a bucket brigade. The bucket brigade is implemented in an array of data cells, with the input entered at one end and then shifted down the array, one element per sample time until it comes out the end, *n* sample times later, as in Fig. 13:

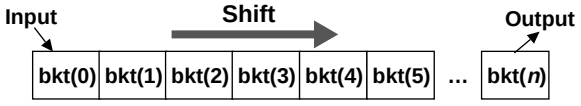


Fig. 13. Bucket Brigade

On the face of it, the Padé is capable of modelling a continuous range of deadtimes, whereas the bucket brigade is capable of representing only integral delays, for instance by varying *n*. Either mechanism can represent a fixed deadtime. A further problem arises if the goal is to represent changing deadtimes. In this case neither the Padé nor the bucket brigade with varying *n* really reflects the physical or theoretical behavior of the process.

However the bucket brigade more closely models the state behavior of product in a delay element; the internal bucket states do represent the internal propagation. The states in the Padé are unrelated to the internal product propagation. It turns out that the continuous Padé will model a changing deadtime if the internal parameters are appropriately changed. However mapping this into a discrete version, taking into account all of the earlier considerations, will be quite difficult.

Thus if the bucket brigade delay can be changed by "speeding it up or slowing it down" rather than by varying *n* (the output bucket). it can model changing delay times resulting from changes in flow rates. There are still design questions:

- How does one smoothly achieve deadtimes smaller than $n\Delta t$ (the number of buckets multiplied by the sample-time)?
- How does one smooth the data between stored buckets (between shifts, as shown in Figure 14a.[20])?

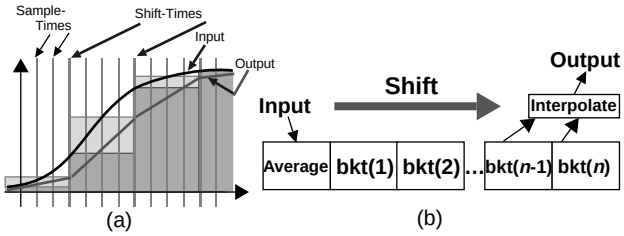


Fig. 14. Bucket Brigade for Slow Shifts

- How can one modify the discrete deadtime (modeled by an integral number of sample-time shifts) to represent a continuous range of (changing) deadtimes?

The first question is the easiest to answer: Shift more than once at a time[21]. The second question is almost as easy to answer: On the input side, average the sampled inputs between shifts; on the output side interpolate between the last two buckets, to smooth the effect of the shift. (See Fig. 14a[22] and b[23].) The effect of all this is to create a deadtime approximation

[20] The Figure shows the process record (the solid line), also as sampled (the thin, vertical, dashed lines), as effectively sampled by the delay under infrequent shifts (the thicker, vertical, dashed lines), and as then sampled and held (the more deeply shaded histogram).

[21] Also averaging the output values coming from such a multiple shift.

whose effective deadtime *T* corresponds to (*n*+.5) shift-times.[24]

The last question is the more subtle one. and is answered by using an irregular shifting frequency, whose immediate average value is equal to the current desired shifting frequency, corresponding to the desired (fractional) deadtime. This solution is somewhat similar to the method used to achieve fine color variations on a CRT display which supports only a few basic colors: mix a number of different pixels irregularly for the intended average affect.

The flow chart (Figure 15) shows the simplest way of achieving the desired irregular shift-time. An accu-

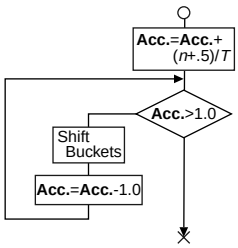


Fig. 15. Bucket Brigade Shift Calculation Flow Chart

mulator variable (**Acc.**, initially zero) is incremented by (*n*+.5)*/T* (corresponding to the desired fractional number of shifts per sample-time[25]). If the accumulator variable has been incremented to one (indicating a net requirement for one full shift) a bucket brigade shift takes place and the accumulator is decremented by one. Shifting and decrementing is repeated until the accumulator value drops back below one.[26]

The result is a compensator algorithm capable of modeling deadtime in fully time varying situations.

[22] The more lightly shaded histogram shows the effect of averaging the shifted values. The dashed record reconstruction shows the effect of interpolation. Note the half shift-time delay due to the averaging and interpolation.

[23] Figure 14b. shows the modified bucket brigade with averaging at the input and interpolation at the output.

[24] A more refined approximation would view it as a deadtime of *n* shifts with a lagtime of one shift-time.

[25] If $(n+.5)\Delta t = T$, then $1/\Delta t = (n+.5)/T$.

[26] Actually the shift test value for the accumulator is irrelevant (and would be set to zero in practice), because the continual incrementing and decrementing balances itself out to the same frequency, whatever the test point. A further simplification is to increment by (*n*+.5) and decrement by *T*, further saving the need for floating-point or the division.

Quantization and Saturation Effects

All of the calculations can now be carried out. The discussion has not addressed the effects of truncation on derivative, but as developed here they are not more serious than for proportional control. In both cases truncation will cause a very small limit cycle, on the order of the minimum quantization of the D/A converter. However, if the derivative is not carefully filtered as part of its computation (as shown in integrated form above) severe problems arise. An approximate unfiltered derivative calculation has the form:

$$D(t) = \tau_D \cdot \frac{\Delta Measurement}{\Delta t} = \frac{\tau_D}{\Delta t} \cdot \Delta Measurement$$

Quantization has the awkward effect of forcing a minimum nonzero measurement change for any sampled calculation.[27] This value is multiplied by the gain $\tau_D/\Delta t$, which is usually very large. The result is very large pulses on the output of the controller. For Δ*Measurement* quantized to one part in 1000 (0.1%), Δt = 1 sec., and τ_D = 10 min. = 600 sec., the minimum nonzero derivative equals:

$$D(t) = 600 \times 0.1 = 60\%$$

The use of filtering smears out the pulses and limits their height (The simple 0.1 time constant lagged derivative filter, developed earlier, limits the maximum pulse height to 10 times the quantization value. More refined algorithms can minimize the problem further by dynamically broadening the effective Δt in the calculation to get a better average derivative. The challenge is to get effective derivative action with a quantization that represents the realistic accuracy bounds of the data.

These derivative problems can be made far worse by internal saturations after integration in the PID algorithm, particularly in incremental algorithms (algorithms which output the desired change in valve position rather than the desired position). Such algorithms involve a second derivative action. The problem arises because the saturation is likely to be unsymmetrical. When reintegrated later in the algo-

[27] Conventional resistor ladder A/D's may be in substantial error in the calibration of their lowest order bits to the extent that a constant slope measurement may appear to wander up and down as converted. This makes these quantization affects several times worse in practice.

rithm or system, the result is a significant offset. In the previous case, the second differencing causes a doublet which extends 60% of scale in both directions. One sided saturation, reintegrated, would create a 60% valve offset (bump).

Identification and Matrix-Oriented Issues

Theory motivated control thinking emphasizes matrix-oriented formulations. These are becoming more common as engineers are trained in them. Properly understood, traditional methods are capable of equally good control, but there are aspects of normal control algorithm design where these newer methods may be more appropriate. Adaptive control often suggests such methods. Space permits only an introduction to the problem but references [Ref. 13–17]. cover many important considerations.

A typical equation of this class defines least squares data fitting of over-determined parameters, as used in adaptation:

$$A^T A x = A^T y$$

In this equation, y is a data vector, x a vector of parameters, and A an $n \times m$ matrix ($n \geq m$) of data vectors, occurring in a set of equations of the following form:

$$a_{i1} x_1 + a_{i2} x_2 + \ldots = y$$

The problem is to find the best fit for the parameters x, given the known A and y, solving the first equation:

$$x = (A^T A)^{-1} (A^T y)$$

Early direct or recursive solutions were unnecessarily sensitive to numerical rounding and truncation errors. The related eigenvalue problems were equally difficult until the problems were understood [Ref. 13–16].

As before, he problem can b e explained as an effect of differences of large numbers, and the explosion of the digital data range under multiplication by numbers not close to 1. In matrix computations the concept of being "close to 1" is formalized in several ways. Is $|A|$ close to 1? Because matrices involve several "directions", the determinant is misleading. Another measure of a matrix is its norm:

$$||A|| = max_x \frac{\sqrt{x^T A^T A x}}{\sqrt{x^T x}}$$

Underlying the norm concept is the theory of orthogonal matrices and singular values [Ref. 13]. An *orthogonal matrix* is one whose inverse equals its transpose: $Q^T Q = I$. (The letter Q will designate an orthogonal matrix.) In essence orthogonal matrices are "equal to 1" in every possible way except being the identity:

4. Their determinant is equal to 1, but for sign.
5. They do not change the length of vectors that they multiply.
6. Therefore, their norm equals 1.
7. Products of orthogonal matrices are orthogonal.
8. Every element of Q has magnitude ≤ 1.
9. The elements of the RGA [Ref. 18] or interaction measure of an orthogonal matrix are all between 0 and 1. Thus easy computation corresponds to easy control.

The *singular values* σ_i of a matrix A are the square roots of the eigenvalues of $A^T A$. More interestingly, every matrix (square or not) obeys the singular value decomposition theorem [Ref. 13]. This theorem states that:

$$A \equiv Q_1 \Sigma Q_2$$

for Σ, the diagonal matrix of the singular values σ_i, and some orthogonal $Q1$, $Q2$. Also:

10. The singular values of an orthogonal matrix are all equal to 1.

Under the singular value decomposition, when A multiplies a vector, the immediate Q_1 or Q_2 twists the vector into an orientation where each component is multiplied by one of the singular values. Thus the vector most amplified in length by A is one oriented so that it is multiplied by the largest σ_i. For this reason the norm of A equals its largest σ_i. Also the vector most diminished in length by A is one oriented so that it is multiplied by the smallest σ_i.

Generally, a matrix is hard to compute with (is effectively "much larger than 1") if there are significant off diagonal terms and the ratio of largest σ_i to smallest σ_i is much larger than 1. (This also corresponds to large RGA elements.) Computing $A^T A$, as in the least squares equation, squares this ratio making computation that much worse. Effective algorithms minimize such operations.

As a simple example, consider the usual solution of the equation $Ax=b$, with A the matrix shown in Fig. 16. The conventional gaussian solution[28] involves reducing the matrix to a triangular form which can then

Gaussian (LU):

$$A = \begin{bmatrix} 0.1 & 1 \\ 1 & 1 \end{bmatrix} = LU = \begin{bmatrix} 1 & 0 \\ 10 & 1 \end{bmatrix} \times \begin{bmatrix} 0.1 & 1 \\ 0 & -9 \end{bmatrix}$$

Orthogonal (QR):

$$A = \begin{bmatrix} 0.1 & 1 \\ 1 & 1 \end{bmatrix} = QR$$

$$= \begin{bmatrix} -0.0995 & -0.995 \\ 0.995 & 0.0995 \end{bmatrix} \times \begin{bmatrix} -1.005 & -1.0945 \\ 0 & -0.8955 \end{bmatrix}$$

Fig. 16. External Feedback Based Backup

be "back solved". This is equivalent to factoring that same triangular form out, leaving a second triangular form (the *L* and *U* in the figure). Note that result involves large numbers (the 10 and –9).

The same matrix can also be reduced to a triangular (back solvable) form by multiplication by an orthogonal matrix. The corresponding factorization is called *QR* factorization as shown. Note that all the calculations will now involve small numbers (.995 and –1.005). The thrust of the newer matrix methods is to avoid matrix multiplication if possible (It expands the data range.), and to try to restrict any multiplications to those involving orthogonal, diagonal, or triangular matrices.

Software and Application Issues

Earlier there was a brief reference to software implications of control saturation. No modern discussion of control would be complete without observing the critical role of software [Refs. 19–25]. Matrix oriented control has been based on FORTRAN and purely mathematical approaches to control thinking. This has separated it from the major operational concerns of the industry. A control system should not only support the control computation, but the operational access to control data in a framework that is as easy to use as possible. It should include some model of sensible operator intervention.

These considerations are accommodated naturally by the software for traditional control. Standard process regulatory control has been based on "blocks", interpreted by the computer to carry out control.

These blocks are blocks in two senses. They represent the digital equivalent of the old analog blocks in block diagrams. And they consist of data blocks whose data elements correspond to the signals and parameters of the analog block controller or are data pointers which make the connections between the blocks.

But neither the existing systems nor the proposed standards offer attractive solutions, supporting the necessary flexibility and ease of use. The challenge is to provide flexible, future oriented systems in which the goals and structure of the control system is transparent; new sensors, actuators, and algorithms are easily added; and sufficient standards included so control systems can include elements from many vendors. [Refs. 24, 25]

And the block model is inefficient for the kinds of control that are now possible. For example, the iterated structure suggested in Figure 11 would be extremely inefficient if implemented in blocks, even as it would be quite simple programmed directly. And as normally interpreted, blocks have a predefined number of connections, limiting the use of structures with an indefinite number of inputs or outputs (e.g. feedforwards or overrides).

Several references also show the limitations of the normal block diagram in representing the kinds of controls called for here.[Refs. 20-23,26,27] Despite the universal belief in graphics, control diagrams are hard to understand. An improved notation and graphics separately lists the loops making up the design, clearly distinguishing their controlled variables and purposes.

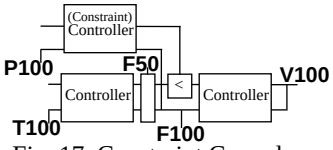


Fig. 17. Constraint Cascade

For example, the Figure 17 combination of two cascaded controllers with a pressure high constraint controller and a feedforward, according to the earlier structuring, would be expressed as:

T100$_{\text{REGULATE FEEDFWD F50}}$ **P100**$_{\text{HICONSTRN}}$ **F100**$_{\text{REGULATE}}$ **V100**

with the main degree of freedom path broken by the **P100** constraint expression.

Two operational issues are of special concern: smart field devices and interoperability. Modern sensors and actuators now include significant amounts of computational power themselves. The uses of this power will

[28] The example cheats a little, bypassing any "pivot' operation. These are imprecise compared to the orthogonal matrix methods.

expand indefinitely. But both the field devices and the central controls must be developed in a standard model which permits control and its orderly operation to be supported without special or redundant programming of either.

There is a broader challenge: to solve the control software problem with solutions that capitalize on the computer as a truly intelligent control device, beyond the rigid scientific computations envisioned by the matrix control approaches, or the programmable block diagram mimicking the dated analog controls.

Summary

Digital control algorithms can be designed for experimental or single applications using the easiest tools available: FORTRAN, C, BASIC, and floating-point arithmetic. In this case, there is a reasonable hope that normal commissioning debugging will weed out all problems. But, if a sense of workmanship prevails, or if the algorithm is to be used in many applications, then attention to refinement and foolproofing are necessary:

1. Numerical effects of fixed and floating-point arithmetic.
2. Documentation, control and testing of detailed scaling, precision, and saturation within the algorithm.
3. Design for natural tuning and qualitative behavior, predictable from analog intuition.
4. Nasty quantization surprises.
5. Accommodations of windup and other control limiting effects.
6. Bumpless transfer and operational considerations.
7. Software, architectural, configuration considerations.

General purpose digital programming languages and tools do not remotely address these issues.

Bibliography

1. E.H. Bristol, "On the Design and Programming of Control Algorithms for DDC Systems", Control Engineering, Jan. 1977.
2. E.H. Clagget, "Keep a Notebook of Digital Control Algorithms", Control Engineering, Oct. 1980, pp. 81–84.
3. W. Fehervari, "Asymmetric Algorithm Tightens Compressor Surge Control", Control Engineering, Oct. 1977, pp. 63–66.
4. F.C.Y. Tu and J.Y.H. Tsing, "Synthesizing a Digital Algorithm for Optimized Control", Instr. Tech., May 1979, pp. 52–56.
5. G.F. Franklin and J.D. Powell, Digital Control of Dynamic Systems, Addison-Wesley, Reading, MA, 1980.
6. ISA SP50 User Layer Technical Report.
7. D.E. Knuth, The Art of Computer Programming, vol. 2, Addison-Wesley, Reading, MA, 1981.
8. Collected Algorithms from ACM, 1960–1976.
9. ANSI/IEEE Standard 754–1985 for Binary Floating-Point Arithmetic.
10. Chemical Engineers' Handbook, 5th Edition, McGraw-Hill, New York, 1973, pp. 2-81 to 2-85.
11. D.M. Considine, Process/Industrial Instruments and Controls Handbook, 4th Edition, McGraw-Hill, New York, 1993, pp. 2.85-2.86.
12. E. Hansen, Global Optimization Using Interval Analysis, Marcel Decker, Inc., 1992.
13. G. Strang, Linear Algebra and Its Applications, Academic Press, New York, 1976.
14. G.J. Bierman, Factorization Methods for Discrete Sequential Estimation, Academic Press, New York, 1977.
15. C.L. Lawson and R.L. Hansen, Solving Least Squares Problems, Prentice-Hall, Englewood Cliffs, N.J., 1974.
16. A.J. Laub and V.C. Klema, "The Singular Value Decomposition: Its Computation and Some Applications", IEEE Trans. Autom. Control, vol. AC-25, no. 2, Apr 1980, pp. 164–176.
17. A.G.J. MacFarlane and Y.S. Hung, "A Quasi-Classical Approach to Multivariable Feedback Systems Design", 2nd IFAC Symposium, Computer Aided Design of Multivariable Technological Systems, Purdue University, West Lafayette, Ind. Sept. 1982, pp. 39–48.
18. E.H. Bristol, "On a New Measure of Interaction for Multivariable Control", IEEE-PTGAC, vol. AC-11, no. 1, Jan. 1966, pp.133–134.
19. F.G. Shinskey, "An Expert System for the Design of Distillation Controls", Chemical Process Control III, Asilomar, CA, Jan. 12–17, 1986.
20. E.H. Bristol, "Strategic Design: A Practical Chapter in a Textbook on Control", 1980 JACC, San Francisco, CA, Aug. 1980.
21. E.H. Bristol, "After DDC – Idiomatic (Structured) Control", 88th National Meeting AIChE, Philadelphia, PA, June, 1980.
22. A. Prassinos, T.J. McAvoy, E.H. Bristol, "A Method for the Analysis of Complex Control Schemes", 1982 ACC, Arlington VA, June, 1982.
23. E.H. Bristol, "A Methodology and Algebra for Documenting Continuous Process Control Systems", 1983 ACC, San Francisco, CA, June, 1983.

24. E.H. Bristol, “SuperVariable Process Data Definition”, ISA SP50.4 Working Paper, Oct. 24, 1990 and updating.

25. E.H. Bristol,“An Interoperability Level Model: SuperVariable Categories”, ISA SP50.4 Continuing Working Paper.

26. E.H. Bristol, “A Language for Integrated Process Control Application”, Retirement Symposium in Honor of Prof. Ted. Williams, Purdue University, West Lafayette, IN, Dec. 5-6, '94

27. E.H. Bristol, “Redesigned State Logic for an Easier to Use Control Language”, World Batch Forum, Toronto, May 13-15, '96; also ISA Transactions, Vol. 35 (1996), pp. 245-257.