Fuzzy Logic and High Performance Motion Control

by Chuck Raskin, P.E., CMCS

In the world of technology, new hardware and methodology have a tendency to change faster than the ability to keep up! With all of the hype generally attached to a new technology, it becomes increasingly more difficult to determine where "spec-manship" leaves off and reality begins.

The use of acronyms and personal credentials such as degrees and licenses generally leads people to believe whatever the so-called experts say *without qualification*. Thus, when I first heard of the term ***"Fuzzy Logic,"*** I simply shrugged it off as another passing fad. After all, why would I want to be fuzzy about what I do for a living?

But as time went on, ***"FL"*** hung on tight, and in fact, became increasingly popular. Since it had an attachment to the world of artificial intelligence (AI), it even gained in credibility. In an attempt to find out what all the hoopla was about, I acquired several FL books, a mass of FL reports and papers, and I then proceeded to read all about Fuzzy Logic.

The more I learned about FL, the more I knew I had been using similar control methods for years. I just didn't have an official name for it. The real problem I found in all FL discussions was that the presenters never seemed to get to the point! Other than the use of FL in slow, non-responsive, long-time, constant web, fan or temperature circuits, I didn't find many examples for the use of FL in high-performance motion control applications.

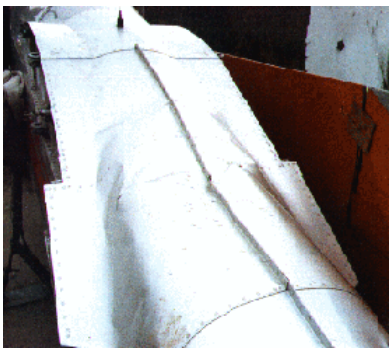
I asked myself if it were possible that FL cannot perform in high performance systems. This is the question I will address here, using simple, down-to-earth language.
To do this, I will have to resolve the following:

- **What are Crisp and Fuzzy logic?**
- **How do Crisp and Fuzzy work?**
- **Can Crisp and Fuzzy work together?**

2 examples of over stressing the design (Airplane wings)

Both powered and unpowered motion devices use the basic laws of physics to maintain their integrity and attitude

When these laws are exceeded the system will fail!



C&R Engineering Inc Osceola Wisconsin Craskin@CRengineering.com
Ph: 715 294-3753 Fx: 715 294-2558 http://www.CRengineering.com

Defining High-Performance Motion Control

Before we can say what will or won't work in the field of high-performance motion control, it is important to understand what we are promoting. The definition of motion control is very elusive and has different meanings to different people in the same, and different industries.

As an example, if you're a servo motor engineer, you might indicate that "closing a loop" is the key to controlling motion. On the other hand, stepper motors can operate "open loop" and do just as well as servo motors within the bounds of the steppers' available power and speed. Pneumatic and hydraulic applications also perform quite well in both open- and closed-loop environments. Conveyance of products using simple AC motor technology has never stopped anyone from positioning overhead cranes and other types of apparatuses by simple switch activation. These must also be included into the definition of motion control.

Unpowered motion devices such as satellites and gliders, which use basic physical laws to in order to operate, must also be included in the motion control definition. And, if we carry this reasoning a bit further, we will also need to include the controlling hardware, such as: PLC's, PC's, CNC's, board-level controller systems, computer chips, DSP's and ASIC's, and more. We also must include firmware, software, algorithms, and feedback technology. As I see it, this is really a team effort, and since my background qualifies me to express at least one opinion, I include all areas of engineering (and even some not in engineering) into the world of motion control, using the following simple definition:

Motion Control Means Coordinating Moving and Non-Moving Objects

Sound foolish? If you look back at your own design efforts, every system you worked on probably had some degree of crash potential inherent in its design. A robot navigating a room filled with tables and chairs is being coordinated in a space of non-moving objects. XYZ tables have hard (crash) travel limits. Even a simple rotary device probably has an associated arm doing work on the load it is rotating, so a crash potential exists.

Since everything we design has an element of motion and non-motion associated with it, the real effort is to develop a control system that will allow the motion elements to work safely within the bounds of the non-motion elements. As with the glider, the design must also be within the bounds imposed by the capability of the elements of which it is comprised. The key words here is ***"system."*** The design of the **system** must be within the bounds imposed by all of the motion elements.

The word ***system*** describes *component teamwork*. All system components are designed and interlocked to work as one—with no single component more critical than the other. Motion control is by definition a world without boundaries. It's a world in which the only limitation is the imagination of the designer.

The type of motion performance—low, medium or high—can only be defined when the factor of ***time*** is included. *"When the performance capability of the system and control with respect to the amount of work to be done and cost are combined, the degree of performance can be determined."* Performance by itself is simply how something is doing. In order to qualify as "high" performance, the real-time effort necessary to maintain control of the process must be borderline to the ability of the

process. In other words, you are approaching or exceeding the limits of the available technology to control the system.

But if high performance is based on technology alone, it would be reasonable to assume that given better technology, the performance could be easily achieved—and the term "high" performance downgraded to "medium" performance. And, there is also a budget to consider. Not every problem to be solved has unlimited funds, so cost becomes a major factor in the ability to achieve the desired performance. In the overall scheme of things, it is the specification including the budget placed on a design that is the *governing factor for determining the degree of performance*. If the specification outperforms the available technology and/or budget, then it will probably fall into the category of "high" performance design.

All mechanical idiosyncrasies must be understood for a high-performance control to work. A variety of considerations about the final design must not be taken for granted. Some of these include: how hot a motor can get; machine lubrication; preload forces; changing load weights; bending or twisting moments of machine parts; system friction, motor and amplifier limitations; motor operating mode; real-time mechanical capability; type of sensors and feedback in place or required; and so forth.

In addition, the ability of the customer to maintain and operate the system might determine the direction of the software efforts with regard to help screens, self-diagnostics, preventive maintenance, flags, etc. And finally, just how complex does the high-performance control really need to be? Are hardware solutions—relays, circuits, and so forth—viable solutions, or does software control have to be the prime effort? When should software control methods enter into the design? Can a change in the mechanics help the electronics or the software? How do you define nonlinear problems, and is nonlinearity a major stumbling block in the ability to control? If answering these questions were easy, then everyone would be a rich and successful motion control specialists.

Since almost nothing is absolute or certain, how can engineers be sure their designs will work? The answer is, "They don't have to be!" Our *initial* job is only to tip the scale of certainty in our favor. This, then, indicates that designers ***must*** do their homework.

Engineers must utilize timing charts, equipment research, software analysis, and whatever else it takes to satisfy the major design requirements. Many of us who do high-performance motion control as a career already know how important it is to learn from the successes and failures of both our own and other designers' projects. The importance of maintaining accurate information databases, and of building "people networks" to allow designing with confidence, can't be overemphasized. Put these elements together, and we have garnered the ability to resolve problems never before encountered. The better we define the problem, the more certain we will be of the solution!

What is Fuzzy Logic?
How does it work?
Why is it called "fuzzy"?

When I first started reading about FL, I thought there was a conspiracy to ensure that ***crisp*** logic engineers could not dispute its usability. Rather than explaining FL's "functionality," FL practitioners seemed to stack the deck against competing control methods. FL was to be the perfect solution for nonlinear applications, and a definite replacement for the hard-to-tune PID gain structure.

The more I read, the more I realized that the primary function of FL is **not** to confuse engineers, but to relax the design and tuning efforts generally associated with long time-constant, high-phase-lag systems.
Since the terms *Crisp* and *Fuzzy* are so commonly used, I will define their meanings as I use them here. ***Crisp*** refers to unscaled, as-read or written signal information. This information can be digital (On/Off); or analog (±E). ***Fuzzy*** refers to crisp signals *after* they are scaled, weighted, or modified in some manner. ***Fuzzification*** is the official term used to describe the process of turning crisp signals into fuzzy ones. Once data has been fuzzified, it can be worked on by the FL rule base. As a side note, in my 35-plus years of design experience it would have been an unusual if input or output data had **not** been weighted to some degree in order to meet system requirements.

If you study the FL approach, you'll find five basic steps in its development and use:

1. **Input all system data, sensor and feedback information.**
2. **Convert, scale, or weight the input information for use with the FL rule base, as necessary.**
3. **Solve the FL rule base.**
4. **Convert, scale and weight the results for use as output data or control signals, as necessary.**
5. **Output the results to the system.**

Types of FL imprecision

The five-step sequence noted above should look familiar, since it is similar to the solution format used in standard PLC control. As a matter of fact, many FL concepts were employed in analog, PLC, and computer control, even before FL became an independent science. Much of the implementation of FL today is through the use of specialized FL chips. Attach some inputs, set up some scaling or weights, apply some rules, and out comes one or more control signals to operate your device.

The problem I had in accepting FL is the manner in which writers described it. For example, the following short list of statements was taken directly from various Fuzzy Logic writings:

- **Many sentences describe FL as an imprecise science . . .**
- **Fuzzy systems are universal approximations.**
- **Fuzzy has the ability to make recommendations in imprecise terms.**
- **Fuzziness is a measure of how well an instance conforms to a semantical concept.**
- **In FL, the truth of any statement is a matter of degree.**

C&R Engineering Inc Osceola Wisconsin Craskin@CRengineering.com
Ph: 715 294-3753 Fx: 715 294-2558 http://www.CRengineering.com

- **If the antecedent is true to some degree, *then* the consequent is true to some degree.**

I found it very hard to accept a design technology that promoted its problem solving techniques without the requirement of understanding the problem! That all I need to do is work with the principle of maybe in order to solve the unsolvable. This might work for the weather channel, but in the engineering world, a direct pipeline from the problem to the solution must exist. There must be reasons for the direction one takes and a predictability of the desired results. If FL is to be used as the vehicle taking us from the problem to the solution, then its use must be based on practical design reasoning and not inferences of design ineptitude.

All control systems are designed to "look" at crisp input information (from sensors, switches, signals, feedback, data, etc.) This is done under the real-time requirement of the system to insure the system is not only functioning, but that it is doing what it was told to. These inputs may or may not need to be conditioned, a term used in crisp logic to indicate the input value has been altered. (Another term associated with conditioning is preprocessing.) Input data may need to be conditioned, or preprocessed, prior to being worked on. In the world of FL, however, the term used to describe conditioning is fuzzifying.

The typical FL-style rule used to determine what the control must do, based on input signal conditions, is generally given in the form of an if/then statement such as:

IF **<Arg1>...<ArgN>**
THEN **<Result 1>...<ResultN>**

There can be just a few or many if/then statements in the FL rule base. In addition, the method of mathematically manipulating the rule base results may not yield the same answers, depending upon the FL designer. There are also situations in which two or more input conditions yield discontinuity in the output control signal, which is not allowed in a qualified motion control system.

A partial FL example listing used in a vacuum cleaner application follows. In this application, pressure and dirt level readings were used to vary the speed of a motor. The sensor values had to be conditioned to insure they worked together.

Many different vacuum cleaner designs can be reasonably based on fuzzy logic design concepts. In the following example, as pressure and dirt levels vary, their conditioned values (from 0 to 1) are presented to the rule base. The rule base, in turn, applies conditioned variables to the conditioned sensor values (depending upon the if/then arguments in use.) The FL conditioning values, in this example,

were selected by trial and error.

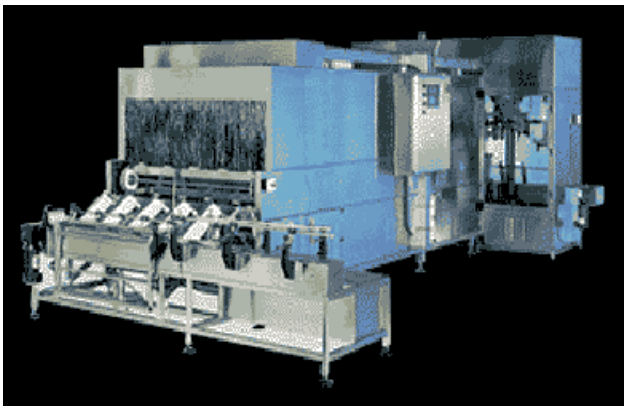
First, the rule base works on the input data and determines control requirements, such as: speed up, slow down, increase or decrease, etc.

The results are then applied to specific control devices such as motors, heater, valves, etc. The results may or may not have to be conditioned before being placed on actual signal control lines (referred to as outputs, control signals, data, etc.).

In crisp logic, output data may or may not need to be conditioned, or *postprocessed,* prior to being sent to the output devices. In the world of FL, the term used to describe the postprocessing of data is *defuzzify*. There is nothing new about how input and output signals are handled when dealing with either crisp or fuzzy logic. But between preprocessing *(fuzzifying)* input data, and post-processing *(defuzzifying)* output data, there is the rule-based logic unit. The rule-based logic unit is where the actual FL solution determining algorithms reside. This is where crisp logic and FL part company.

The objective of any control logic is to maintain the system specification. If the specification is velocity stability, or position accuracy, or a temperature setting, or whatever… then that is what the control objective will be. How the control logic accomplishes this is not of prime concern, as long as all of the specified control requirements are met, and the project is within budget!

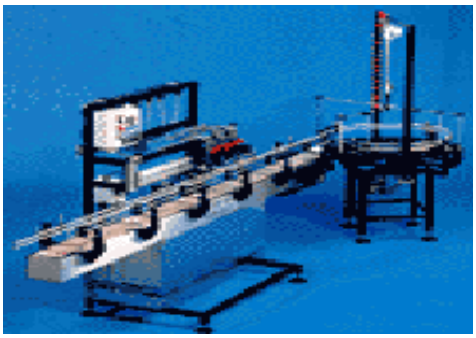
What FL can do is add a dimension to the *supervisory* control algorithm. It can allow the system to zero in on the requirement, rather than to project what will happen based on current and past events. This is not to say that FL cannot alter its reasoning as more is learned about system reaction to system data—it can. But the objective of FL remains the same: to *zero in* on the target as time progresses, and not to accommodate momentary or instantaneous system reactions to outside influences.



FL is good for both Large. . . .

Finally, there is a need to model the system to determine if the control is *valid*. Whether crisp or fuzzy logic is used, the control design must be tested prior to putting the application into service. If the project cost is less than several hundred dollars (like the vacuum cleaner example), it might be reasonable to test and adjust controls manually on the actual system. But if the controls are headed for a power company or are a part of helicopter design, it would be a definite advantage to check the validity of the controls prior to use. The real question to ask yourself is: *Would you want an untested control to be put on a system you were paying for?* Some FL advocators talk about the use of FL in systems not capable of being modeled. Yet, in every FL writing I've come

across, the example system was capable of being easily modeled.



Aachen University Professor Hans Juergen Zimmermann, founder of ELITE (European Laboratory for Intelligent Techniques in Engineering) has said, “Fuzzy technology reduces the complexity of problems; helps to move from symbol processing in expert systems to meaning, preserving reasoning; and improves the capabilities of existing modeling technologies.” The notion that modeling is not required because you are using FL is simply wrong.

and Small Machines.

In addition, FL today has been applied to systems that had been in operation successfully using lower cost crisp logic! What this indicates is that FL has not yet found its calling. It’s still looking for that “killer app.”

Based on what I’ve seen so far, the use of FL to overcome the problems of the PID gain structure in set-point and general registration applications is sound.

But the implementation of FL to take the place of controls already in place doing solid control work is simply salesmanship. designing bottling operations, determining separate motion control requirements for interlocked smaller systems can actually cut both time and expense.

Getting the best from both worlds

Is high-performance motion control capable of being enhanced by the abstract reasoning of fuzzy logic? What improvements or advantages could be realized if FL is incorporated unto the design process? After months of studying FL and using many FL learning aids, I still do not consider myself a qualified fuzzy logician. Nor do I believe this method of control is new to the industry. The more I read, the more I know I’ve used FL before, but under the alias of “simple control design.” The FL gurus seem to attack the infamous PID at all levels of conversation. But the PID was, and still is, used extensively in process control applications. The real problem with PID process control is not the difficulty in tuning a specific PID, but the lack of understanding and the difficulty in coordinating multiple PIDs when more than one is involved in a process. To get them all to work together as one transparent device is extremely difficult because of long system time lags and general system inconsistencies.

The difference between process control and high-performance motion control is simply that process control is generally considered an outer or supervisory control loop used to monitor, and correct for, multiple events based on many system requirements. Motion control, by itself, is an inner control loop that monitors and adjusts motion within its own small environment. We could possibly revise our definition to say that process control monitors and adjusts systems of events, while direct motion control monitors and adjusts single motion events. A process control

should *always* be thought of as *an outer loop in the control environment*.

But each type of system has to be understood within its ability to monitor and control. For example, if I were commissioned to develop a control system for a bottling company, I would first go over the system requirements, then the mechanical design requirements. I would try to scale down the design effort by dividing the entire process into subsystems (smaller processes) and develop interlocking control sequences for them. Finally, I would determine the motion-control requirement for each subtask. This might seem like a lot of work, but all systems are generally built up of smaller subsystems interlocked together to become the final process. Although dividing the system into subsystems is more work, each of the individual tasks is simpler to resolve, and thus, the work effort is actually reduced.

Once the complete system has been defined, we can determine the type of control logic each subsystem will require. Conceivably, crisp or fuzzy can fit into any or all of the system control areas. FL might be the means of maintaining a single operation, or an environmental parameter within a given range. It might even be used to coordinate several subsystems. Then again, it might not be used at all.

Let's review the five steps for good control.

1. **Input all system data, sensor and feedback information.**
2. **Convert, scale, weight the input information for use with the FL rule base, if necessary.**
3. **Solve the FL rule base.**
4. **Convert, scale, or weight the results for use as output data or control signals, if necessary.**
5. **Output the results to the system.**

Steps 1 and 5 are contained in every control system known to mankind (The inputting and outputting of signals and/or data). Steps 2 and 4 simply refer to preprocessing or postprocessing of the inputted and outputted signal data. These steps may or may not be applicable in a given control requirement. Step 3, however, is where designer must be knowledgeable enough to choose between fuzzy and crisp for the real-time solution.

Applying Fuzzy and Crisp

In high-performance motion control, decisions must be made faster than the system's ability to go outside the required tolerance windows. These types of systems usually require answers within microseconds of the event. FL chips in todays market process their data in hundreds to thousands of microseconds. It seems obvious then that FL would not be effective in high-performance motion control applications. *Not true!* If the principle of FL and its relationship to the control requirement is understood, then FL can be used to *supplement* the main control software and improve the overall operation of the process.

Why not use off-the-shelf FL chips currently available with the registration philosophy embedded within? An analogy would be to consider a velocity mode

motor/amplifier package. The motion control device cannot operate its output control signal faster—at a higher bandwidth—than the motor/amplifiers ability to respond. If it does, the motion will become unstable. In high-performance motion control, the CPU must have all control information within a real-time window based upon *required performance*. If another processing device (FL CPU) is used to feed information to the main processing CPU, it must be able to deliver all of its information when the main CPU requires it. If not, the system will have to slow down. In other words, the control CPU used in high-perform-ance motion applications must have all pertinent sensory feedback and control information faster than the real-time *window of opportunity.* Thus, as an inner control loop, FL logic would not be usable due to its slow response rate. But as an outer supervisory control, the FL chip would perform more than satisfactorily.

An Application Example

Years ago, before "fuzzy" was introduced to the control world, a lot of time and energy was spent on registration control. Registration is a means *of controlling multiple axes of motion in order to coordinate the relative positions of the product they handle.* To manufacture a multilayer product such as a bandage (multilayer web control), registration is used.

Degrees of motion adjustment can be made based on the conditions initially imposed by the operation. As system capabilities change, the adjustment base must change with it. This adaptive capability was developed to accommodate a wide variety of system variances in many applications, such as web handling, labeling, flying cutoffs, etc.

The quilted and layered bandage in the two figures to the right, were designed for a horse's cannon bone. To manufacture them, motions had to be velocity controlled and position coordinated.

But does all this mean that FL cannot be part of the inner control loop? Not at all. The FL chip or the communication rate may be too slow, but by embedding the FL software into the main CPU software logic, the approach could work without requiring inter-chip communication speeding up the control process. What is necessary is to determine what the problem requires. Which control method, crisp or fuzzy, will perform more acceptably and within the bounds of the mechanical, electrical and budgetary requirements of the system? If they are both at the same level, then it simply becomes a matter of which control method you feel most comfortable with. As an old saying goes: "If you put 50 engineers into a room and give them a

problem, they'll come up with 500 possible solutions." The best solution, however, is the one which meets the specification and fulfills as many of the following requirements as possible:

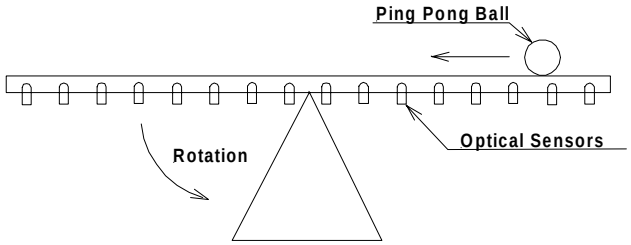
- **Utilizes the fewest parts (least complex and shortest design time).**
- **Is the lowest cost (high volume production).**
- **Is the simplest to operate and requires low technical expertise.**
- **Is the most reliable for long-term on-line requirements.**
- **Enables the highest throughput.**
- **Has the least software involvement, which means less debugging and tuning.**
- **Meets the system delivery schedule.**

The Ball-on-a-Beam Balancer

FL practitioners indicated that the Ball-on-a-Beam Balancer is a prime example of a non-linear system. Hua Li et al. (1994) tried to demonstrate the ability of FL to control a ball on a balance beam better than a crisp analog PID control could.

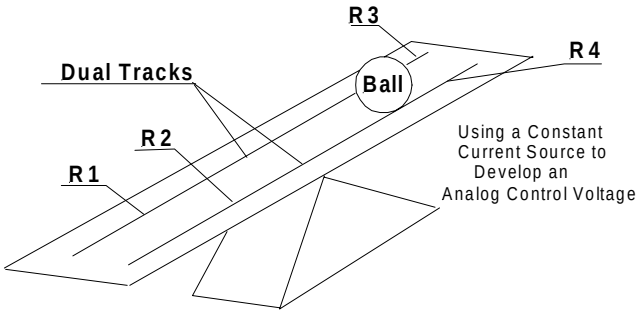
The basic problem for the balance beam was to move a ball from a position on one side of the beam pivot point to a position on the other side of the beam pivot point in the shortest possible time, with minimal to zero overshoot.

The unit derived by the FL group used a series of 16 photo-eye sensors spaced over a 36-inch long beam.
The sensors looked up through holes in the beam to sense a ping-pong ball passing over it. The controls update time was tailored to the application.
The FL rule base would then determine the direction and magnitude of beam tilt required in order to move the ball from its current location to a new location and stop without overshoot.



Ball-On-A-Beam Balancer

Bob Pease (1995) tried this using a pure analog approach. Using a metal ball rolling on a two-rail track, Pease applied a constant current power source to the rail system. The ball simply acted as a potentiometer wiper. Voltage information was then sent to the control for position feedback.



'H' Bridge Analog Design

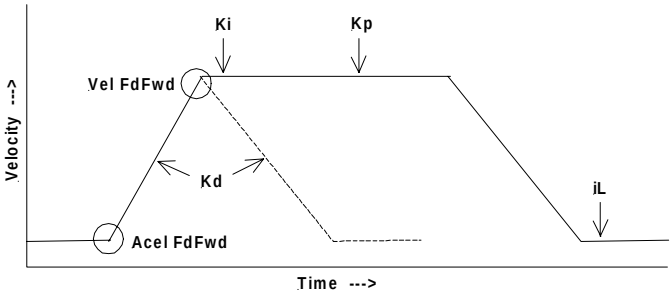
In both the Hua Li and Peases experiments, a manual joystick allowed participants to try their hand at moving the ball between various positions on the beam. Since stopping the ball with minimal overshoot in the least possible time was the heart of the operation, the heavier the ball, the more difficult the test.

Both systems worked!

However, the FL approach was embedded in a PC, while the analog approach used approximately $20 worth of components on a simple breadboard. If no specifications had been given for the balancing application, then any solution derived by any engineer would have proven a point. But any solution, as ingenious as it might be, would then be based on proving other solutions wrong. A good solution should be able to stand on its own merits, provided it satisfies the customer specification, and meets the suggested criteria. We need to stay focused by indicating that the FL solution did not fit the requirement list noted in the previous section. I will present this Ball-on-a Beam challenge in more detail at the end of this report.

PID Review

If you have a means of watching the reaction of a system with respect to P, I, and D tuning changes, the next figure should apply. It shows the location on a Trapezoidal profile to look at when making specific PID adjustments. Always adjust the Kp first. Kd should be entered next, with care, to stabilize any instability caused by Kp adjustments. Once Kp is done, then use a Triangle profile to adjust Kd. It is checked when halfway up and down the accel/decel slopes for a balanced following error equal to the value achieved with the Kp adjustment. Next, set the IL to five times the Kp value, and adjust the Ki until an acceptable round out is noted at the knee between the acceleration slope and the top velocity. Finally, lower or raise the IL to end the move with zero position error. This is a very simple approach to motor tuning, and it has worked very well on over 95 percent of the systems I have tuned.



Where the Tuning is Done

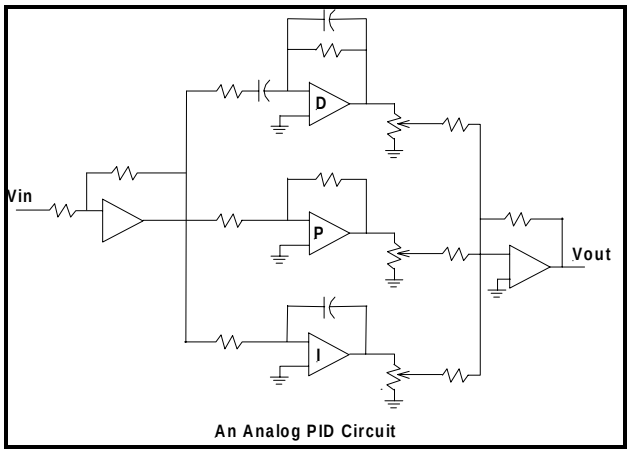
In the typical PID profile, the velocity feedforward (Vff) and the acceleration feedforward (Aff) assist the gain structure. They are implemented after the PID is tuned.

Analog PID Control

The included figure shows a simplified version of the PID control circuit devel-oped by Bob Pease of National Semiconductor to resolve the Ball-on-a-Beam balancing problem. Although not a *high-performance* design, the circuit shows very simply the integration of the three PID elements. A variation of the circuit was built for less than $20—and was able to make a ball move 22-inchs in just under 8 seconds from start to finish with very little end-of-move oscillation.

The thing to note about this circuit, and in fact all analog circuits, is the infinite nature of the control. This feature, however, can be a double-edged sword. On the one hand, continuously monitoring motion will not allow the mechanism to get out of

control once it has been tuned for the operation. But on the other hand, this circuit is a velocity loop control requiring high gain at low velocity (at or near standstill).



An Analog PID Circuit

If precise positioning is the main system requirement, an analog circuit by itself is not usable (FL or Crisp). However, with a little ingenuity, this circuit was able to move the ball and stop reasonably smoothly. The secret was to make the ball part of the feedback network. Had the ball not been part of the feedback, the task would have been a bit more expensive, but still doable with the same result.

Microprocessor PID Control

The microprocessor approach to PID control is simply a math calculation. The motion trajectory is calculated, then the actual position of the device is checked against it, and the error between the two is used by the PID to close up the gap.

Simple math required to do S-curve generation, including the PID gain calculations with velocity and acceleration feedforward follows. Note that the time to calculate the trajectory profile and then perform the gain calculations is dependent on the processors speed or the ability to preplan the profile.

1. Get the current axis position from the encoder counter (AP).
2. A = A + J — Calculate the new desired acceleration increment.
3. If (A > Alim) then (A = Alim) — Set to limit if acl inc is larger than allowed.
4. V = V + A — Calculate the new desired velocity increment.
5. If (V > Vlim) then (V = Vlim) — Set to limit if vel inc is larger than allowed.
6. P = P + V — Calculate the desired position increment.

At this point the desired trajectory path profile values have been determined.

7. FolErr = P - AP — Calculate the Pos Error.
8. Total_FolErr = +/- (Total_Error + FolErr)
9. Vel = AP - Last_AP — (Vel using fixed time intervals = P/T)
10. VelErr = +/- (V - Vel)
11. Acl = Vel - Last_Vel
12. Acl_Err = A - Acl

At this point, all of the required trajectory errors have been determined.

The DAC Output is the sum of the gain terms multiplied by one of the calculated errors.

13. DAC = (Kp*FolErr_Err) + (Ki*Total_Err) +(Kd*VelErr) + (Kvff*V) + (Kaff*AclErr)

C&R Engineering Inc Osceola Wisconsin Craskin@CRengineering.com
Ph: 715 294-3753 Fx: 715 294-2558 http://www.CRengineering.com

This simplified trajectory generator operates very comfortably in update intervals under 25 microseconds using a TI or Analog Devices DSP. This is definitely real-time when it comes to controlling a motion profile.

FL PID Approach Example

The following FL PID approximation was taken from an application report (#104) produced by Adaptive Logic Inc. located in San Jose, CA. It uses an AL220 Fuzzy Logic Controller, developed by them, for doing the PID effort. The algorithm utilizes two inputs and four outputs. The two inputs are Error and Speed The four outputs are Control (the command signal), Integral, Last_Error, and Timer (the process sampling rate control), all used as internal feedback variables.

The internal timer is incremented during each process cycle. The Timer varies, based on the desired speed of response of the system. The timer is recycled between *1* and a *Timer_is_Max* membership function. During the period when *Timer_is_Max* is nonzero, a group of rules increments or decrements the integral output register by an amount roughly proportional to the error signal. Next, the output register control is set equal to the current value stored in the integral. Control is then incremented or decremented by an amount roughly proportional to the error signal by another group of rules.

Finally, a third group of rules increments the control register by an amount proportional to the derivative of the error. Thus, the final output of control is the sum of the integral, proportional, and derivative components. By changing the sampling rate of the timer, the response and smoothness of the controllers output can be adjusted.

But you must take note that the sample time is being adjusted! This means that as the control signal is being manipulated to arrive at its set-point destination, any disturbance in the motion, due perhaps to a tool touching the work being moved, can happen at a point in which the PID algorithm to be unprepared to react. The sample interval may now be to slow to respond properly allowing motion instability to ensue.

Fuzzy Variables	Center	Width	Shape
Timer is MAX	250	0	Right Inclusive
Timer is NOTMAX	249	0	Left Inclusive
Timer is ANYTHING	0	0	Right Inclusive
Timer is ZERO	0	0	Symmetric Inclusive
Error is ZERO	128	2	Symmetric Inclusive
Error is SMNEG	124	4	Symmetric Inclusive
Error is SMPOS	132	4	Symmetric Inclusive
Error is MEDNEG	116	6	Symmetric Inclusive
Error is MEDPOS	140	6	Symmetric Inclusive
Error is LRGNEG	100	12	Symmetric Inclusive
Error is LRGPOS	156	12	Symmetric Inclusive
Error is MAXNEG	128	24	Right Inclusive
Error is MAXPOS	LAST_ERR	24	Left Inclusive
Error is NOCHG	LAST_ERR	2	Symmetric Inclusive
Error is SMDROP	LAST_ERR	2	Right Inclusive
Error is SMRISE	LAST_ERR	2	Left Inclusive
Error is MEDDROP	LAST_ERR	6	Right Inclusive
Error is MEDRISE	LAST_ERR	6	Left Inclusive
Error is LRGDROP	LAST_ERR	20	Right Inclusive
Error is LRGDROP	LAST_ERR	20	Left Inclusive
Error is MAXDROP	LAST_ERR	40	Right Inclusive
Error is MAXRISE	LAST_ERR	40	Left Inclusive
Speed is SLOW	0	63	Symmetric Inclusive
Speed is MED	128	63	Symmetric Inclusive
Speed is FAST	255	63	Symmetric Inclusive

Rule Base

1. If Timer is Zero then Integral = 128 (Initialize Integral on start-up.)
2. If Timer is NotMax then Integral = 0 (This rule will win, overriding 3 - 11 if Timer is still incrementing.)
3. If Error is Zero then Integral + 0
4. If Error is Zero then Integral + -7
5. If Error is Zero then Integral + 7

6. If Error is Zero then Integral + -4
7. If Error is Zero then Integral + 4
8. If Error is Zero then Integral + -2
9. If Error is Zero then Integral + 2
10. If Error is Zero then Integral + -1
11. If Error is Zero then Integral + 1
    (This is the last rule in the group whose consequence affects Integral).

The winning rule is selected at this point.

12. If Timer is Max then Control = Integral
13. If Timer is AnyVal then Integral + 0
14. If Timer is NotMax then Control + 0

Note: Rule 14 separates rules above from those below. It will allow a winner to be selected from each group.

16. If Error is MaxPos then Control + -20
17. If Error is MaxNeg then Control + 20
18. If Error is LrgPos then Control + -9
19. If Error is LrgNeg then Control + 9
10. If Error is MedPos then Control + -4
21. If Error is MedNeg then Control + 4
22. If Error is SmPos then Control + -1
23. If Error is SmNeg then Control + 1
    Note: Rule 23 is the last rule with a consequence affecting Control.

The winning rule is selected at this point.

24. If Timer is AnyVal then Integral + 0
    This rule separates the Kp group above from the Kd group below.
25. If Timer is NotMax then Control + 0
    (This rule insures no change in output if Timer is still counting.
    It selects the winner over rules 26-34 if Timer is NotMax.)
26. If Error is NoChg then Control + 0
27. If Error is MaxDrop then Control + -40
28. If Error is MaxRise then Control + 40
29. If Error is LrgDrop then Control + -20
30. If Error is LrgRise then Control + 20
31. If Error is MedDrop then Control + -6
32. If Error is MedRise then Control + 6
33. If Error is SmDrop then Control + -2
34. If Error is SmRise then Control + 2

Note: This is the last rule in the group with a consequence affecting Control.

A winning rule is selected at this point.

35. If Timer is Max then Last_Err = Error
    Store the current value of error for the next cycle.
36. If Timer is Max then Timer = 1 Reset the timer.
37. If Speed is Fast then timer + 20
38. If Speed is Med then timer + 4
39. If Speed is Slow then timer + 1

At this point, the PID loop is finished. The question I put here is: What exactly is

the FL PID controlling?

If we look back at key phrases, the words speed, velocity, small, large, etc. indicate *pure velocity* control. This particular operation is trying to do the do the same job the analog PID circuit discussed previously is doing, but at a significantly slower rate (many milliseconds).

Also, the number of pseudo code instructions listed in the FL routine (39) are significantly greater than the number of instructions in the previously described microprocessor model (12).

There is no position precision included in the FL model, but there is in the microprocessor example. The FL example is simply a velocity or set-point controller model.

Because the ability to control position is at the basis of high-performance motion control the FL model would not be adequate. If, however, the FL model were refined for position control as well as velocity, the sample time would not allow high-performance positioning to occur (less than 25 microsecond updates.)

A better application for FL, however, would be to supervise a registration application, for coordinating the tolerance adjustment of the applied materials. In the case of "flying" applications, it could be well suited to maintaining the operation tolerances while the high-performance motion was handled by a more suitable 'crisp' methodology. It becomes a matter of balance. Making the best operation by designing with a full compliment of devices. Not allowing only one technology to stifle an operation requiring two or more different ones. Allowing the application specification to control the solution.

Two Applications and a PID Comparison

In this section I will discuss the use of FL and crisp logic in two high-performance motion control examples, cam follower and registration. In each of these applications, the performance specifications will be given as they were given to me. The design thoughts implemented in these applications will also be discussed. We will see where FL could fit into the design. Although the FL approach was not implemented in these applications, its use would have only been restricted to the registration application. A third specification will be given for the now-infamous Ball-On-A-Beam balancer, which is claimed to be the example of a true nonlinear system by FL practitioners. Readers are invited to find their own solution(s) (PID or otherwise) to the problem.

I will also include a PID application note, which will compare strategies used in PID control structures. The comparison is made using analog, fuzzy, and microprocessor-based approaches. As we all know, any logical solution is restricted only by the designers' ability to understand the problem. So for this problem, understanding PID is the key to its usefulness.

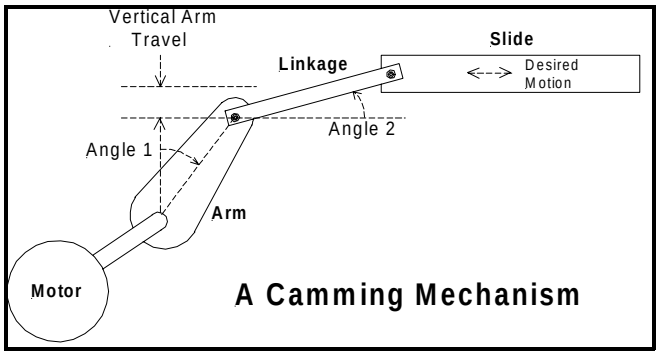
Since both PID and FL structures require feedback, the use of one or more control loops is inherent. Understanding how loops are generally organized and what their

intended purposes are simply makes the design effort easier.

Single Axis Cam Operation

The specification for this application were to operate a slide mechanism in a simple back-and-forth motion at a constant velocity (a pure trapezoidal profile). But the calculations required to convert the rotary motion of the motor shaft to the required linear slide motion through the linkages as shown is not a trivial task! The calculations to do this are given in the equations shown on the following page.

This design requires that the profile generator's velocity and position must be changed continuously. This requirement enables the complex motor motion necessary to ensure linear slide motion. It involves intense calculations and complex profiling to maintain the smoothest possible linear slide motion in the fastest possible time. The longer the delay between calculation updates, the more erratic the slide motion.



The motor shaft arm forward vector velocity (Afv), as it travels through the arc, is:

Afv = [Arv_Vel / sin(90-Ang1) in/min

where ANG1 is the difference between the vertical, and the motor arm position.

The forward vector velocity is modified again, as the linkage arm raises and lowers due to the motor shaft arm progressing through ANG1. This forms ANG2 between the linkage arm and the horizontal. The calculation of the slides forward velocity is then:

Sfv = [Afv cos(Ang2) in/min

It became immediately evident that the type of calculations involved here would require more CPU power than a simple Rack-&-Pinion design using a simple trapezoidal profile. Not so evident, however, is the increase in interrupt processing, software writing, and general cost of the control this method will have to bear.

The real question at this point is, would FL work in the application as presented in the preceding figure? Looking closely at the operation, it is a trajectory profiling problem. In order to maintain the smoothest possible linear slide motion, the motor velocity would have to be adjusted at an infinite rate (pure analog simulation). This, of course, is impossible in a digital environment. But just where does the break exist between stable and unstable motion in this application? We asked the customer. Velocity stability was 0.5 percent maximum.

With this information we could determine the system configuration and the update rate for the calculations. Note that as the motor shaft arm rotates, the horizontal velocity anywhere along the arm length is altered by a sine function. In turn, as the motor shaft arm rotates, it also moves the linkage arm up and down, changing its vertical angle.

The slide's forward velocity is modified by a cosine of the linkage arm angle. To help maintain a slide velocity within 0.5 percent, the trajectory controller will use a 12-bit analog output (DAC). Each DAC step is worth 100 percent / 2048 steps = 0.04 percent of maximum velocity capability. The motor/amplifier package will be operated in the voltage mode with a sensitivity of no greater than 4mv. This will insure that when a velocity step change is commanded, the motor will respond.

Finally, to determine the calculation update rate, we need to work backward from the required maximum load profile to the motor when the arm is at the 90-degree vertical position. At the 90-degree arm position:

Given:

Slide Velocity = 100fpm x (12in/ft) / (60sec/min) = 20in/sec

Where:

Slide motion = 6 inches
Motor arm length = 6 inches
Arm/slide linkage = 8 inches
Linkage angle = 10 degrees
Max arm rotation = ±30 degrees from vertical

Then:

$$Sfv = \left[\frac{MotorVel}{\sin(90 - Ang1)}\right]\left[\cos(ANG2)\right]$$

And:

$$20 = \left[\frac{MotorVel}{\sin(90 - 0)}\right]\left[\cos(10)\right]$$

Therefore:

$$MotorVel = \left(20\frac{in}{sec}\right)\left(\frac{1}{.984807}\right)$$

And:

$$MotorVel = 20.3\frac{in}{sec}$$

A 6-inch arm length yields a 37.7-inch circumference. To move the motor shaft at a rate of 20.3 in/sec, the motor Rpm would be 31.83 Rpm. If the controller output is maintained at that value while operating the motor at 31.83 Rpm when the arm is 90 degrees vertical, the slide will begin to slow down as the arm advances. (This is also known as *Chordal Action*). At what arm angle will the next command have to be given to the motor amp in order to compensate for this effect to be nullified at the slide? For simplicity, we will not include the electrical time constant of the motor or the mechanical time constant of the system.

To maintain a 0.5 percent worst case velocity shift when the arm is moved off center from its vertical position, the motor velocity will have to be increased. The calculation sequence to determine the time in which the motor's velocity must be adjusted is:

NewSVatTimeT = 20in/sec x 0.995 = 19.9 in/sec

If we assume for the moment that the linkage angle has changed less than 1 degree, the error induced by the linkage angle would be less than 0.3 percent.

Thus the arm angle would have shifted:

Where:

$(ANG1=\sin^{-1} .995)$

and:

Shift = 90 - ANG1 = 90 - 84.268 = 5.732 deg.

And finally, the time for the motor to travel 5.73 degrees at 31.83 Rpm is 0.0005 seconds (500 microseconds).

As the arm continues to rotate, the time to alter the motor velocity increases since the forward vector of the arm (slide motion) is changing at a slower rate.

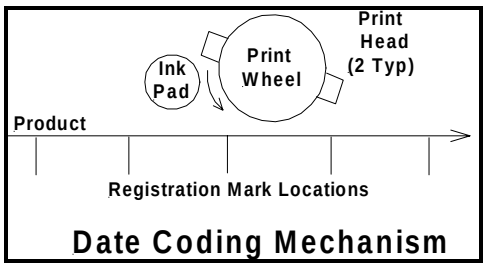
Although other considerations need to be taken into account to solve the whole problem, we have successfully estimated a time requirement for the update period. To guarantee the gearing operation is doable, system time constants must be part of the calculations; the linkage angle must be included; and the motor velocity update period should be at least five times faster than the calculated profile alteration time. This will insure that the desired continuous rate can be achieved. Because FL is not a position-oriented algorithm, and because of the speed and pure mathematical nature of the cam operation, and the degree of exactness required of the motions position throughout the move, the 'crisp' microprocessor-based PID is the most viable solution.

High Speed Date Code Printer

The specifications for this application were:

- **Maximum line speed = 4000 inches per minute**
- **Minimum print-to-print spacing = 2 in.**
- **Print tolerance = ±0.015 inch (±1/64)**
- **Print min/max length = 1/8 to 1 inch**
- **One or two prints can be made per print wheel rotation**
- **Print head diameter = 2.25 inches**
- **Must be able to print in either direction**

The print rate was calculated to be 33.3 prints per second: 4000(in/min)/2 (in) / 60 (sec). This yields one print every 30 milliseconds. To properly size the motor, the required print wheel profile had to reflect the relationship between two inches of material line movement and a 180-degree print wheel rotation from a "dead-stop" in the same time period.



Date Coding Mechanism

The feedback requirement and device was chosen, then the system torques were

calculated, then the motor was picked. The feedback line count was chosen to insure compliance with the tolerance specification. Next, a software flowchart was developed to allow a visual understanding of system handling requirements. Finally, the motion control computer was selected.

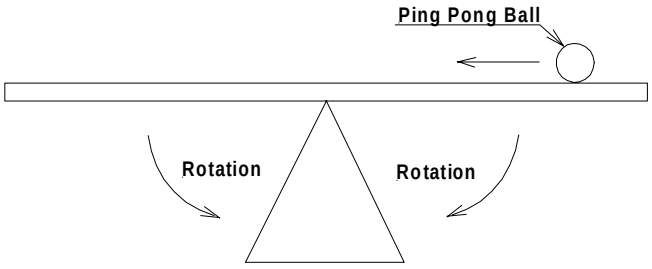
The layout of the printing mechanism is shown. The coordination of the print position begins when a preprinted registration mark is located by the controller. If the controller fails to spot the mark, it would have to initialize the printing operation anyway—at the position it computed where the mark should have been. Failure to see the mark could have been due to noise, interference, material vibration and/or borderline optical adjustment, etc.

In this operation, FL could have helped to insure that the print position tolerance was met, but only as a supervisory outer loop. Prior to applying, a date-code, an inspection station could monitor the print placement. This information could then be used to adjust the motion controller's starting point to accommodate the variation.

Could FL have fit into this control scheme? You bet, but because I was *not* fuzzy about the operational requirements, and because FL chips and methods did not exist then as they do now, FL was not used. Would FL have made the job simpler? No—due to the fact that the monitoring and adjusting of tolerance positions was only a small part of the overall control requirement (<15%). But there was definitely a place for FL in this application.

Back to the Ball-on-a-Beam Balancer...
How would you solve this one?

This specification was taken directly from a report titled "Fuzzy Logic and Intelligent Systems" written by Nowell Godfrey, Hua Li, and William Marcy from Texas Tech University in Lubbock, Tx. and by Yuandong Ji from Case Western Reserve University.



Original Ball-On-A-Beam Problem

The specifications:

1. Beam length 36 inches
2. 16 Photo sensors on 2 inch spacing (1 inch on center two units - Ref. Marker)
3. Stepper motor - 4 Phase - .22 amp. - 1.8 degrees / step
4. Analog joystick for "human" interface
5. Quad. Incremental encoder on beam for tilt information - 0.143 deg/cnt max.
6. Estimated data transfer rate required = 3.3 Mbyte/second
7. The ball is a standard ping-pong ball
8. Motor-to-beam mechanical gearing approximately 12:1

Note:

Nonlinearity may or may not be a factor in the ability to do this job

You may not deviate from specification numbers 1, 4, and 7. All other items may

be modified or changed as desired.

If you're really curious, the FL approach to the problem may be obtained from:

Library of Congress
ISBN 0-7923-9575-1
QA76.87.F89 1995

Conclusion

Motion control is more than just moving something. It's coordinating complex profiles involving multiple moving and non-moving devices and objects. It's the constant vigil to overcome the many factors trying to cause motion instability—friction, viscous torques, inertia, load changes, and other erratic disturbances. It's insuring that the planned trajectory is followed— and perhaps being maintained—with respect to other motions (circular, spline, or parabolic interpolation, master/slave operation, cam operations, robotics).

The PID by itself is a gain structure that can handle most problems encountered in motion applications. With simple feature enhancement, such as velocity and acceleration feed-forward, notch filtering, adaptive control, etc., the PID can well serve at least 99 percent of today's applications.

Gain algorithms, however, do not work alone. The best mathematical formula in the world is useless if it cannot be resolved in the real-time window of the system it is controlling. This is heart of the FL problem. Intense conditioning of data and elaborate rules to be resolved require more time than most high-performance motion control CPU's can afford to give. Thus, FL will remain in a supervisory capacity until available CPU's can process FL data in the high-performance loop times more effectively.

FL, however, is not going to go away. In fact, its use will grow as DSP motion control chips continue to get faster. The purpose of FL, however, should not be to *take over*, but to *enhance* motion control applications. In an article written by Stan Baker of Programmable Performance Electronics Corp. entitled "*Kill the Killer App*", the author stated, "over hype is closely related to the search for killer apps." Rich Terrill with Altera said, "At rare and exciting points in time, we have the tools and techniques before we have the applications." Keeping this in the proper prospective, I feel that FL has the tools and techniques to fit into some of today's applications, but its use in high-performance motion control is still awaiting that high-speed DSP to make it a viable solution."

Curtis Wilson, VP of engineering at Delta Tau in CA, describes FL as

"Approximate solutions for poorly defined or misunderstood problems"

Many people feel that the correctness of a statement is simply dependent upon who you're talking to. In that case, from a motion control high-performance/real-time standpoint, and as a systems engineer and analysis for over 35 years, I agree with him 100 percent!

If you look at the problem as a "***real-time***" issue, requiring microsecond solutions, and "high-performance" as something other than just turning up the thermostat, FL

can be seen for what it is, an ***outer loop approximation controller*** whose use can enhance crisp logic devices.

Projected for the year 2002 are DSPs operating at speeds up to 400 Mips. TI already has a 5nsec device (the TI-67 series). As chips like this becomes more prevalent in the industry (lowering the cost) the use of FL will grow. Hopefully, the designers who employ FL will do so based on application need. But FL, like any other technology will still not be the single solution that will fit all problems.

C&R Engineering Inc Osceola Wisconsin Craskin@CRengineering.com
Ph: 715 294-3753 Fx: 715 294-2558 http://www.CRengineering.com