DEMO2: Modeling control

A TEACHER'S TUTORIAL AND GUIDE, PART 2

In Demo2, a block diagram of a control system is built up step by step. At each step, the user can manipulate the mouse to explore the properties of the model, and change parameters from the keyboard. After the complete model has been constructed, the program finishes by showing how the behavior of the model can be matched to that of a real person.

There is one program accompanying this manual. To run the program you must have a computer compatible with an IBM AT (80286 processor), 512K of memory, a Hercules-compatible monochrome graphics screen or a CGA, EGA, or VGA-compatible graphics screen, and either a mouse, a game port with a good joystick or one of the two supported analogue-to-digital boards with a control handle. In Appendix A, the required equipment is described in detail, with operating instructions.

Copy the master program onto working floppy programs or into an appropriately-named subdirectory on a hard program. Copy ALL the files. You can run from a floppy disc or the hard disc. The name of the program is DEMO2.EXE. Delete ADCONFIG, restart program to reconfigure. Type README for more installation information.

If used for teaching or commercial purposes, the price of this material is $60. It is understood that this amount will be remitted for each class-term or each significant other use made of it, and that the user will not re-sell any of the material. Payment is strictly on the honor system; the program may be reproduced and given away freely.

Some comments on modeling

The term "model" can mean many things, from an ideal to a statistical analysis to a miniature replica. We will use it here in the engineering sense: a simulation of a behaving system's components. The term simulation means a working model—one that behaves through time. But it does not mean recreating all the details of the real behaving system. To build a model, we imagine taking a system (or more often just part of a whole behaving system) apart into what we think are its major components. We then define each component in terms of variables that act on it, and another variable that represents its action. The action of any one component depends on the variables that affect it. The basic requirement is to state how the action of a component depends on the variables that affect it; how its output depends on its inputs. It isn't enough to claim that the output depends on the inputs; to make a proper model we must try to guess at the details, even though the first guess is likely to be wrong.

Success in modeling depends on breaking the system down to a level that is neither too coarse nor too fine. If we can find a level of analysis that is just right, we will be able to define each component in a way simple enough to understand, yet detailed enough to reveal relationships that are not trivial. When we have a model that works in the essential respects like the real system, we can then think of breaking each component down further, looking for a deeper level of understanding. Or we can try to add to the model so that more of the total behavior of the whole system is represented. We could go in either of these directions with profit. The first step, of course, is to find one model that works. That is what Part 2 is about: modeling the behavior called compensatory tracking.

What does it mean to say that a model "works?" It means that we should be able to compare at least some of the variables in the model with the real variables they represent, and show that when the model oper-

ates according to its own rules, its variables behave the same way as the real ones. We may propose variables in the model that have no observable counterparts, but there must be at least several variables that do have real counterparts. Otherwise a comparison with the real system would be impossible.

Models—simulations—behave through calculation. Given starting conditions for the variables, and given the rules that make some variables depend on each other, we apply our definitions of the components to convert hypothetical values of input variables into deduced values of output variables. The output variables become input variables for other components, allowing us to continue the calculations one component after another. In a control-system model we eventually find ourselves back where we started, having computed our way around a closed loop. We start the next round of calculations with the values of the variables that resulted from the first round. Each round gives us a sample of all the variables at one moment in time. Doing the calculations over and over develops a picture of how the variables change with time.

There are traps in using a digital computer to simulate a system with components that actually behave simultaneously and continuously through time. The computer can compute only one relationship at a time, so we must do the calculations for the various components in sequence. We have to do the computations so that the result is as if they were all done simultaneously. On a digital computer we can only approximate this simultaneity. As long as we are computing in a straight line, where one step leads to the next in a straightforward way, there is no problem if we make the rounds of calculation represent small enough intervals of time. But when the loop closes on itself, when we compute the value of the same variable we started with, the same variable is called on to have two different values at once.

In the real system, every variable is always being affected by variables just prior to it in the chain of calculations. If we could do parallel or analog computations this fact would take care of itself. But when we complete a round of digital calculation, for a moment we have the old value of a variable and its new value being different. How we handle that makes a lot of difference in the results. If we overlook this problem, we can reach wrong conclusions about how much feedback there can be in a closed-loop system, as well as making other major mistakes. These mistakes come about from not treating physical time correctly—from confusing computing cycles with the passage of physical time.

In the first seven steps of Part 2, we will build up a model of a person doing compensatory tracking, the same task we saw over and over in Part 1. The model will represent only a few aspects of the whole person. The same model applies to many other kinds of control tasks, but we will stick to the simplest one. As we construct the model, the problem just described will come up at the step where the loop is closed. The program will automatically, and behind the scenes, take care of physical time so that the correct outcome happens. Then the next two frames will show how this adjustment is made, so you can use this method in other applications. Part 2 ends with a comparison of the behavior of this model with the behavior of a real person—you, presumably—doing the tracking task. We will see how simulation allows us to characterize a rather complex ongoing behavior in terms of just a few parameters. When you finish this part, you will understand how modeling can be used to explore real systems.

Modeling the environment

We can construct a model of the environment (outside the behaving system) very easily, because everything that matters there in compensatory tracking is completely known (at least to the programmer of these demonstrations). We know that the control handle affects the cursor position by one path, and that an independent disturbance affects it by another path. The position of the cursor is completely specified at every instant by the angle of the control handle and the magnitude of the disturbance.

In the model, cursor position, handle position, and disturbance magnitude are represented by three numbers, which we symbolize c, h, and d. The physical connections between handle and cursor, and between disturbance and cursor, are represented as two constants: Kf for feedback constant, and Kd for disturbance constant. Kd and Kf begin at 1, so the initial relationship is just c = h + d.

After the text presentation, a graphic is displayed. You can use the control handle, joystick or mouse to position the little handle in the display. The pictorial handle is a frill; all that really matters is the number, h, representing the real handle position. The constants are shown in boxes. Coming out of each box is a line labeled h * Kf or d * Kd. The number shown at the output of each box is the amount of effect of each variable on the cursor. The effect will be different from the variable if the corresponding K-value is different from 1. The effects add. As you change the

handle position, you can see the value of h and of h * Kf changing. The disturbance d and the effect d * Kd change all by themselves. You can use the space bar to freeze h and d, to study the numbers. Another space resumes the action.

You can also use the up and down arrow keys to make a fat arrow on the display point to either of the constants. Make sure that the Num Lock is not active if you use the cursor/numerical keypad. Whichever constant is indicated, you can use the left and right arrow keys (or + and –) to increase or decrease it. The fat arrow isn't very conspicuous, so make sure you find it.

As you change the constants (which can be done whether h and d are frozen or active), you can check to see that the cursor position is always given by c = h * Kf + d * Kd. Play with this until you begin to get a feel for the relationships. With Kf large, for example. the cursor is very sensitive to handle movements. The variables c, d and h in the model depend in real time on the actual cursor position, disturbance, and handle position, and the Ks are the real sensitivity factors, so you can see how the model's behavior corresponds to the real behavior.

Modeling perception

The next graphics frame (after the text introduction) now begins to model the behaving system (above the dashed line). Eventually we will have a model that shows how handle position depends on cursor position via you, the behaving system. We will be guessing that the action depends on perceiving cursor position, comparing what is perceived with a reference position, and converting the difference into output. Right now we are modeling the perceiving component of the system.

The cursor is now called the input quantity qi, and the handle is called the output quantity, qo. These more general labels are meant to suggest that there can be other quantities under control, and other means of acting. The same kind of diagram would apply. In part 1 you saw different kinds of qi that were controllable—aspects of geometric figures, the pitch of a sound, the value of a printed number.

In this model, a perception is a signal inside the behaving system. This signal depends on the state of the cursor or input quantity. As the input quantity changes, the signal changes. The signal is the perceptual signal p. It can change only in magnitude. If effect, we're suppose that there are sensors and neural computers dedicated to sensing the position of the cursor, and that the result is generation of a real-time signal that always corresponds in magnitude to the position of the cursor—the state of the input quantity. As far as the behaving system is concerned, this signal is the cursor position. No matter what is being sensed, the dimension of change that is under control is represented by a single signal that has one value at a time.

There is a box with a constant, Ki, at the input. It represents the sensory apparatus. The value of Ki is 1, and although you can change it to see the effect (when the fat arrow points to the input box), you may as well leave it at 1. When we pass from the physical cursor to the signal, we also change the physical units—from a measure of distance to whatever Nervous System Unit (NSU) we adopt for neural signals. In the real nervous system we might find that a movement of one centimeter of the cursor caused a change in the neural signal of 37.2 impulses per second. We would then define one NSU as 37.2 impulses per second. A cursor position of 1.5 centimeters would result in a neural signal of 55.8 impulses per second; that would be 1.5 NSU. If we choose the right units for one NSU, then the input factor can always be 1. Of course we then have to measure all signals inside the system in the same units (assuming that we can do so in the real system). In the model we just say that one unit of perception equals one unit of cursor position.

With Ki = 1, you can see that the cursor position is directly represented by the magnitude of the perceptual signal p.

The process of comparison

We now have a perceptual signal whose magnitude represents at every instant the position of the cursor. What we need now is a way to represent the desired position of the cursor. As the cursor position is known to the system only as a perceptual signal, it makes sense to use another signal to represent the desired position. This second signal is called the reference signal, r. It is set now to 0, but you can change that value either up or down in magnitude when the fat arrow points to it.

The perceptual signal p and the reference signal r enter a new box called the "comparator." This box represents a function that subtracts the magnitude of the perceptual signal from the magnitude of the reference signal, and passes the resulting difference to its output in the form of an error signal, e. The error signal is thus given by e = r – p.

There are two ways to vary the error signal. If you change the input quantity by any means, the error signal will change in the opposite direction (because the comparator is subtracting p). If you freeze the handle and disturbance using the space bar, (with the input quantity somewhere between plus and minus 100), you can also vary the error signal by changing the reference signal. Because the reference signal enters the comparator with a positive sign, the error signal will change in the same direction as the reference signal.

The process of comparison

We now have a perceptual signal whose magnitude represents at every instant the position of the cursor. What we need now is a way to represent the desired position of the cursor. As the cursor position is known to the system only as a perceptual signal, it makes sense to use another signal to represent the desired position. This second signal is called the reference signal, r. It is set now to 0, but you can change that value either up or down in magnitude when the fat arrow points to it.

The perceptual signal p and the reference signal r enter a new box called the "comparator." This box represents a function that subtracts the magnitude of the perceptual signal from the magnitude of the reference signal, and passes the resulting difference to its output in the form of an error signal, e. The error signal is thus given by e = r – p.

There are two ways to vary the error signal. If you change the input quantity by any means, the error signal will change in the opposite direction (because the comparator is subtracting p). If you freeze the handle and disturbance using the space bar, (with the input quantity somewhere between plus and minus 100), you can also vary the error signal by changing the reference signal. Because the reference signal enters the comparator with a positive sign, the error signal will change in the same direction as the reference signal.

Use the up-down arrow keys to set the fat arrow at Kd, then set Kd to 0 with the left arrow key. Then use the up arrow key to point back to the reference signal. Now you can alter the error signal by moving the handle without the disturbance confusing matters. By adjusting either the reference signal or the input quantity, you can make the error signal approach zero (it's easier with the input quantity, as the reference signal changes in steps of 10 NSU).

Wherever you set the reference signal, there is a value of the input quantity that will make the error zero. All that is required is for the perceptual signal to match the reference signal. Then if the perceptual signal goes above the reference signal, the error signal will go negative, and if p goes below r, e will go positive. The error signal therefore represents by its magnitude the system's judgment as to whether the perceptual signal is too small (positive error), too large (negative error) or just right (zero error).

There are no adjustments in the comparator; any scaling factors here can be absorbed into the other system constants.

The output function

The error signal carries information telling whether the perceptual signal is too small, too large, or just right; it therefore also tells whether the cursor is below, above, or at the intended position. The last part of the system converts this information into some amount (and direction) of output action.

At the output there is a box representing the system's output function. We treat it here as a simple constant of proportionality, Ko, initially set to 3.00. You can change this constant when the fat arrow points to it. The output constant converts the error signal into a handle position, or in the more general notation, to a magnitude of the output quantity qo.

In this step the output quantity is shown twice, once as an input to the environment box (Kf), and again as an output of the system. In both places it is called qo. Obviously when the loop is closed, these two numbers will be just one number, the output quantity. For now we leave the loop broken, in order to present a small puzzle.

You will notice that the feedback factor Kf has been set to zero so the handle has no effect. All that is affecting the input quantity and the perceptual signal is the disturbance. As the disturbance varies, you can watch the input quantity, the perceptual signal, the error signal, and the output quantity changing.

Here is the puzzle. Try adjusting the output factor Ko until you think the output is varying in the way that would be needed to counteract the effects of the disturbance if the loop were closed and Kf were set back to 1. Ignore the number next to the Kf box—just compare the system's output with the disturbance magnitude. You can make this easy by freezing the disturbance variations with the space bar—you can still change Ko, and the computations will update the output quantity qo.

The point isn't to embarrass anyone, but to show that common sense doesn't work very well when it comes to control systems. It stands to reason that if there is, for example, a disturbance of +25 units, the output quantity should be –25 units if the perceptual signal is to match a reference signal of zero (the reference signal should be at zero). To achieve this equal-and-opposite relationship with the loop open, you will find that Ko has to be set to 1.00. This is the reasonable value, but as we will see next, it results in very poor control.

Closing the feedback loop

Now the function key F1 is active. At the system's output you will find the fat arrow and the legend OPEN, meaning that the feedback loop still hasn't been closed. Press F1. The legend will change to LOOP, showing that the handle position is now being determined by the system's output. You no longer can have any effect on the model's handle position (until you press F1 again). The two "qo" numbers will now be identical.

The output constant Ko is initialized at 1.00. If you found the "most reasonable" answer to the puzzle in the previous frame, Ko is now where you set it. You will see the handle varying position by itself, as qo changes. The system is now operating as a closed-loop control system, counteracting the effects of the disturbance on the input quantity.

But the model isn't controlling very well. Look at the error signal. It has the same size as the perceptual signal. By freezing the action with the space bar, you can see that the error is half the value of the disturbance—only half of the disturbance is being counteracted. And look at the real cursor. It is not staying very close to the center position.

Even though the fat arrow is not pointing at the output box, you can still change the output factor Ko using the left-right arrow keys. Now use the right arrow key to increase the value of Ko. One nice way to see the effect is to freeze the changes in disturbance with the space bar (this also freezes your control effects, but they're not active now anyway). Freeze it when there's a fairly large amount of error, like 15 units. Now the loop is still being computed over and over but the disturbance is constant.

As you increase Ko, you will see that the error gets smaller. By the time you have run Ko up to about 10, the output quantity has almost stopped increasing; it will be nearly equal and opposite to the disturbance magnitude. Go on increasing Ko to its limit of 50.00. The error signal just gets smaller and smaller, until it begins wavering near 1 or 2 units. This means that the handle position has changed until it is canceling all but about 1 or 2 units of the effect of the disturbance. You will see that over a range of Ko from 10 to 50, there is very little change in the error signal or the output quantity.

Press the space bar to start the action again. You will see that the error signal hardly changes as the disturbance changes—it stays within plus or minus 2 units of zero except when the disturbance is changing most rapidly. Now we have a good control system. It is keeping its own perceptual signal nearly in a match with the reference signal (which should still be zero). The real cursor is now being run by the model—you can see how well it is being controlled.

With Ko set to 50.00 (make it 50 if it isn't), press F1 to open the loop again and watch the system's output quantity. Freeze the disturbance with the space bar. You will see the output quantity changing until it gradually approaches a final value. That value will be 50 * e. Press the space bar twice to let a different error appear. The output will change until, finally, it again becomes 50 * e. With an error of, say, 30 units, the output quantity will become 1500 units.

You will notice that this output is opposite in direction to the disturbance, but is far larger than it would need to be to cancel the disturbance—almost 50 times as large. This excess is called the "loop gain" of the control system. For good control, loop gain must be high, meaning 10 or 20 or much more, depending on how good the control has to be. Some real control systems have a loop gain of 10 million. Loop gain is actually the product of Kf, Ki, and Ko, but here it is determined by Ko because Kf and Ki are 1.

If you hand-calculated the variables in this loop naively, you would find that for any loop gain greater than 1, the calculations would blow up. They don't blow up in this model because of the slowing that you can now clearly see. When the error signal changes, the output quantity doesn't suddenly change to the corresponding new value. It starts changing toward the value Ko * e, but gets there slowly. Eventually it does get to the right value (if the disturbance is steady). But the slowing is all that allows the system to be stable when the loop is once again closed.

The slowing doesn't actually slow the system's operation. Remember that when the error signal changes, the output starts to head toward a value that is as much as 50 times the amount necessary to correct the error (or much more). That means it reaches the

required amount very quickly. In fact, as we will see in the next frames, the output, given the correct slowing factor, comes to the required value in one computing cycle. The optimum tradeoff between loop gain and slowing leaves the system reacting just as fast as it would have with no feedback present. If you have heard that negative feedback systems are slower than open-loop systems "because they have to wait for the feedback," you can now go back where you heard that and tell whoever it was that they were wrong.

A higher level control system

In the next frame, the system comes up with the loop closed, the output factor set to 50.00, and the fat arrow pointing to the reference signal. By using the left/right arrow keys you can change the reference signal in the range from –100 to +100. This demonstration is suppose to show how higher level control systems can be related to lower levels systems in a hierarchical model of behavior. We won't get into such models here, but seeing how this works may be interesting.

When you're controlling the cursor, you can easily decide to keep it somewhere above or below the target marks. You just "do" it. But how does that decision end up causing the real cursor to move to a new position, and how does it then see to it that the handle position changes to keep disturbances from moving the cursor away from the new position? A two-level model explains this.

Suppose that inside you there is one control system like the one on the screen, that makes a perceived cursor move to a position specified by the reference signal. But suppose there is also a higher-level system, concerned not with absolute cursor position but with the relationship of cursor position to the target marks. This new control system would perceive not only the cursor position, but the target position. In effect there would be a second input box at the lower level sensing the target and creating a target position perceptual signal. A higher system receiving both that target perception signal and the perceptual signal already present that represents cursor position (a copy of it) could now generate a higher-level perceptual signal using a higher level input computation. This higher-level signal would represent the difference between cursor and target positions. The higher level system could then receive a reference signal saying whether this difference in positions should be positive or negative, meaning that the cursor was above or below the target. If the amount of separation was wrong, the output function of the higher system would respond to the error by altering the reference signal for the lower system that already controls cursor position.

You can now act like the higher-level system. Use the left and right arrow keys to change the reference signal for the lower-level system. You will see the cursor moving exactly as you specify.

But watch the model's handle as you do this. The handle continues to move as the disturbance varies; it is automatically canceling the effect of the disturbance. You're not telling the lower-level system what action to perform. You're telling it how much perceptual signal to maintain. The lower-level system then produces whatever handle position is needed to do that. The reference signal does not command an action; it specifies the amount of a perception that is to be maintained.

If the disturbance were zero (you can try this by setting Kd = 0), it would seem that the reference signal causes the handle to move, thus causing the cursor to move. The appearance would then be that the reference signal commands an action, and that the action in turn moves the cursor. That is the interpretation of neuromuscular behavior that is normally accepted. In many instances of motor behavior, the pathways and connections in the nervous system corresponding to our block diagram are known to exist; even the feedback effects are known to exist. But when one is measuring the relationship between a "command" signal and the ensuing action, it is not customary to apply an independent disturbance to the same action—that would spoil the data! As a result, the role of feedback has been missed (except by a handful of workers) and the actual organization of the behaving system has been misinterpreted. Only by applying disturbances to the effects of the action can we get a true picture of how systems like this work. That is why disturbances are always included in models of control systems.

The same arrangement illustrated here can be extended both upward and downward to include many more levels. Each muscle in your arm has a little control system associated with it that maintains a specified perception of tension (sensed in the tendons). The reference signal for tension comes from control systems that perceive and control joint angle, and those systems receive reference signals from higher systems that perceive and control the visual objects you act upon. Those systems get their reference signals from higher systems concerned with the control of motion, events, relationships, and on upward to cognitive levels that perceive in terms of

varying symbols. The principles of control that you see here can form the basis for a complete model of behavioral organization, one that is very different from the model that has always been assumed to be correct.

The equations of control

From the block diagram we have now completed, we can read off the individual equations that describe each component of the system. You can't really claim to have a clear understanding of control systems until you can reproduce these equations, solve them, and understand what the solutions mean. But they are very simple equations: all that's needed is a little elementary algebra.

After the text frame you will see these equations, and in the next frame the same equations being evaluated over and over in real time, about 10 times per second (you will notice some little flickers). The solutions are being computed by running through the equations that you see, one at a time around and around. The slowing factor is always set to the optimum value, so the fourth equation is essentially correct, although you can't make the system of equations work by hand-calculation unless you substitute the expression with the slowing in it. Two frames further on we will show the same system of equations with the slowing explicit, and then you will see how it really works. Then, if you have the patience, you can do the calculations by hand and see exactly what is happening.

The point for now is to see that this system of equations is satisfied all of the time. You can change the two independent variables, r and d, and you can change all the constants. Of the constants, only the output constant Ko is allowed to go negative, so you can see the effects. A negative output constant will result in positive feedback and runaway for any value of Ko more negative than –0.9999... . There is already a negative factor built into the comparator, which subtracts the perceptual signal; making the output constant also negative leads to a net positive overall loop gain—positive feedback. You will see that positive feedback is not nice. To keep the program from stopping on overflow errors, the output quantity is limited to values of plus and minus 10,000 (which would put the cursor about 50 feet off the screen).

As you first see the equations, the four variables on the left are zero. To see something else, set either the disturbance d or the reference signal r to some nonzero amount. Each time you enter a new value (terminating with the Enter, Up-arrow, or Down-arrow keys), new values of the variables instantly appear next to the equations.

There are two basic relationships to notice. First, when the output constant is large the perceptual signal is essentially the same as the reference signal. The larger you make Ko, the more nearly this will be true. With Ko large, the effect of changing the disturbance on the perceptual signal is small, and can be made smaller by increasing Ko. Use variations in the disturbance to see how good the control is, and how much effect there is on each of the four variables down the left side.

The second relationship is best seen with the reference signal set to zero. With a large value of Ko, you will see that the output quantity, qo, remains essentially equal and opposite to the disturbance, d. The larger Ko gets, the more nearly this is true. A control system changes its output to cancel the effect of the disturbance. If you set the reference signal to some non-zero value, you will see that this is still true, but that the effect is centered upon the amount of output that is needed to keep the perceptual signal matching the reference signal when the disturbance is zero. Try disturbances of 100, 0, and –100 with the reference signal set to 50, 0, and –50 to see what is happening. If you want the control to be really tight, set Ko to 5000. To make control begin to look lax, you have to reduce Ko to 10 or less.

There is an important objective measure of control behavior that you can illustrate with these equations. We can't see the real system's reference signal from outside it, but we can deduce its setting. Adjust the reference signal to some number like 75 or 100. Then adjust the disturbance until the output quantity qo, at the bottom left, goes exactly to zero. You will see that the input quantity is then exactly at the level that makes the perception p match the reference signal r. When the output of a control system goes exactly to zero, the input quantity is at its "reference level." That is the level of input quantity at which the perception just matches the inner reference signal, leading to zero error and zero output. We thus have an indirect way of measuring the reference signal, assuming that the real system is organized as the model is. The observable reference level is independent of the output constant Ko (if Ko is greater than 0), as you can verify.

The equations we are seeing are actually expressions of the steady-state conditions that will satisfy the differential equations that are the exact representation of a linear control system. You can solve the equations

as a simultaneous set (by successive substitutions) to get the solutions shown on the screen. Doing this proves that the input and output quantities are both dependent variables. The independent variables are the disturbance (which originates in the environment) and the reference signal (which originates in higher-level systems inside the behaving system). You can verify that environmental disturbances are the primary cause of outputs or actions, while the inner reference signal is the primary determinant of what the system perceives. A control system controls its own input, while varying its output as necessary to counteract external disturbances.

Dynamics: the slowing factor

After the introductory text frame, we look at just the output part of the control system where the slowing takes place. The equation shows how a new value of the output quantity is computed on the basis of the error signal and the old value of the output quantity.

The computation begins as if we are computing the output as Ko * e (inside the brackets on the right side). That number represents the value of output that would occur with no slowing. Then we subtract the old value of qo, shown as qo(old). That tells us how big the jump to the new value would be. Next we divide the size of that jump, Ko*e – qo(old), by the slowing factor S. The result is the size of the jump that will actually be used. Finally, the computed jump is added to the old value of qo, to get the new value.

You can adjust Ko, e, and S. The input variable is e, so to see the effects on the final value of qo, vary e. The screen is set up initially with a slowing factor of 2, meaning that half the calculated distance to the next qo is actually allowed to occur. By hitting the Enter key over and over you can cause the calculation to occur over and over. You will see the output quantity gradually approach a final value, and that value will be Ko * e, no matter what e is.

This is "temporal filtering." We aren't affecting the steady-state computation, but only the way we approach the computed value through time. In the previous frame we just used the equation qo = Ko * e for the output quantity. In terms of steady-state conditions that is the correct relationship, and if you solve the equations as a simultaneous algebraic set, the solutions will correctly describe the state of the system after all transient effects have settled down. But algebra knows nothing of time.

We introduce system properties related to physical time by using the slowing factor. If we say that one computing cycle represents, say, 0.1 second of physical time, we will find that a certain slowing factor is needed to match the real behavior. If we then say that the same computing cycle represents only 0.01 second of time, we will find that we need a slowing factor 10 times as large to get the same result in real time—we allow changes to be only 1/10 as large as before. Without the slowing factor, there's no way to say what one computing cycle (once around the equations) means in terms of real elapsed time.

If you set the slowing factor to 1, you will see that this is equivalent to no slowing. The first computation following a change of e will give qo = Ko * e. For positive slowing factors less than 1 and greater than 0.5, the final value of qo will be reached by values that alternate above and below the final value.

These oscillations are an artifact of digital calculations, and do not represent oscillations in the system being modeled. Likewise, for values of S less the 0.5 the values of qo show larger and larger oscillations that simply run away. These, too, are artifacts of the computation and mean nothing about the real system. For real systems you must choose slowing factors greater than 1, and usually far greater.

Dynamics: slowing factor in the control equations

We now return to the full set of control-system equations, but now the fourth equation is the one we have just been exploring, with the slowing factor included. You can change all the system constants and the independent variables r and d as before, but now you can also change the slowing factor S.

Instead of computing the equations repetitively as before, the program now updates the four equations only once, and does this only when the Enter key is pressed. You can enter numbers and terminate them with an up or down arrow key without causing a recomputation; thus you can change several numbers before recomputing.

On each recomputation, the program starts with equation 1 and evaluates qi. It then uses that value of qi in the next equation and evaluates p; it uses that value of p to compute e, and that value of e to compute qo(next). In effect it is solving the four equations by successive substitution. On the first step of the next calculation, it uses the value of qo(new) found at the end of the previous step. Before the very first calculation, qo is initialized to 0, because it's never been calculated before.

These are the equations that were actually being computed previously when equation 4 said just "qo = Ko * e." When the present screen comes up, the slowing factor is set to the optimum value, and you will see (as you vary r or d) that this results in the whole system coming to its final state on the first calculation after the new numbers are entered.

We will now see that there is a second artifact of computation to watch out for, related to loop gain and the slowing factor. The optimum slowing factor is 1 plus the loop gain: 1 + Ko*Ki*Kf. This amount of slowing makes the computation come to its final value in one computing cycle. For any value of slowing factor less than the optimum amount and greater than half the optimum amount, the system will appear to oscillate above and below the final value, converging more and more slowly as the slowing factor approaches half the optimum value. These oscillations are artifacts of computation; they do not represent oscillations in the real system. Furthermore, when the slowing factor becomes half or less of the optimum amount, the oscillations build up toward infinity. That effect, too, is an artifact and should not be taken as meaningful.

If too large a physical time interval is chosen to be represented by a single computing cycle, the slowing factor needed to match the model's behavior to real behavior may prove to be less than the optimum amount, given the output constant also needed. In that case the model can't be used. The time-scale must be made finer, the intervals shorter, until the slowing factor that is called for becomes greater than the computed optimum. It should be considerably greater, so that its precise value can be found. The term "optimum" refers to the calculations, not to the real system.

You can choose any value of the output constant, then adjust the slowing factor to see its effects on the calculations. You won't see any effect unless you then change the disturbance or the reference signal (if the system is already at equilibrium it will just stay there).

You will see that when you make the slowing factor much larger than the loop gain, the system takes many cycles to approach a new final state. In fact, the larger the slowing factor becomes, the more nearly the output function behaves like a pure "time integrator." A perfect integrator simply sums its inputs to produce an output. The output equation with slowing is qo(new) = qo(old) + [Ko*e – qo(old)]/S. Expanding this equation we have qo(new) = qo(old) + (Ko/S)e – qo(old)/S. The new value comes from the old value plus a constant (Ko/S) times the error, minus a "leakage" term, qo/S. The bigger S is, the less this leakage is. When S is large enough, the leakage becomes so small that it makes no practical difference. Then the output function acts like an integrator with an "integration factor" of Ko/S.

The model parameters that fit real behavior the best have a very large value of S, so the integration factor dominates and the leakage term is essentially zero. Above some size of Ko, the best S is so large that only Ko/S matters—at least for compensatory tracking with a free-moving control handle. Thus a simpler model could be used, in which we just say that qo(new) = qo(old) + K*e, where K is the integration factor. In the next two frames you will see that Ko/S determines how well the model fits, with very little sensitivity to the absolute magnitudes of Ko and S. The model works just the same when Ko/S is 200/1500 as it does when it is 1000/7500, and fits real behavior just as well.

By varying Ko, and then trying different magnitudes of disturbance, you can make the system come to its final state with varying amounts of remaining error. The larger Ko is, the smaller the final error will be. So a large Ko corresponds to fussiness, and a small Ko to "who cares?" Ko is a measurement of the "importance" of an error to the controlling system.

There are two constants in this control system that can be adjusted to match its behavior with the behavior of a real person. The slowing factor S matches the speed of correction, and the output constant Ko matches the degree of action that is generated for a given amount of error. Of course the reference signal, too, must be adjusted so that the model brings the cursor to the same final position that the person does.

Matching the model to behavior

We now come to the ultimate test of this model: using it to match and then to predict a person's real tracking behavior. You will have a choice of three kinds of controlled variable.

The first step is to choose a disturbance, a difficulty factor, and a controlled variable (bar, tone, or number), then do a one-minute run. You would probably do best to choose difficulty 0 (the easiest) unless you're had a lot of practice. Difficulty 3 makes the tracking extremely difficult even for an expert. As you go through the run, the values of handle position are being stored in a long table (1800 entries). Later the controlled variable values are recreated by

reading values from this table and adding them to corresponding values in the table from which disturbances are taken.

After the run you will see a graphics screen on which the experimental run is plotted: the heavy trace is the handle position against time, the mirror-image of it is the disturbance variations, and the trace that wobbles along in the middle is the controlled variable. A straight line is drawn to show where screen center was. The data for the person is plotted in the upper part of the screen; after a moment the same kind of plot is shown for a run of the model, in the lower part of the screen. The model's first run uses default parameters. Every sixth data point is displayed. You've already seen the model running by itself in earlier frames. During the model's run we don't bother to display the controlled variable itself or the handle position; the whole model run takes just a few seconds. Its simulated handle positions are stored just as the real handle positions were.

In the left center of the screen is a trace that shows the result of subtracting the model's handle position from the real handle position for each data point. For each point on the display this plot shows how much difference there is between what the model did during its run and what the real person did.

Just to the right center, the "RMS" value of this difference is shown. It is found by adding together the squares of all the differences, dividing the result by the number of items, then taking the square root. This gives the square root of the average squared difference, known for short as the root-mean-square value, or RMS. It gives a statistically-useful measure of the fluctuations. The fit of the model to the real behavior is best when this RMS value is smallest. To see the RMS value when the model doesn't behave at all, set Ko to a small value (like 1) and do a model run. This shows how much of the variation in handle position is accounted for by the model.

At the top right of the screen is a fat arrow pointing to one of three places where you can adjust the model's parameters:

Ko (output constant), S (slowing factor), and r (reference signal). Use the Up/Down arrow keys to point to one of these parameters, then the Left/Right arrow keys to increase or decrease the values. For Ko and S, the increments and decrements are 2 per cent of the current value, so the size of the jumps gets bigger as the numbers get bigger. This enables you to cover a wide range of values in a reasonable amount of time (the arrow keys repeat automatically if you hold them down).

Below the parameters, near the middle of the screen, the ratio Ko/S is calculated each time you change Ko or S. You will find that wherever you set Ko, if you select a value of S to make the "integration constant" Ko/S come back to the same value, the model will behave very nearly as it did before. Only for values of Ko less than about 10 is there any visible difference. You may want to experiment with very small values of Ko, but to match the real run best you will find that Ko should be around 500 to 1000. In fact you can set it to 1000, and just adjust S.

The adjustments are made while you watch the RMS value. You're trying to make that value as small as possible, meaning that the model's handle behavior differs from the real handle behavior as little as possible. After you've adjusted a parameter up or down, press the Enter key to cause a new run of the model. The lower part of the screen will blank for a few seconds, then the new plots will appear along with a new RMS value. Make fairly large changes in the parameters at first, so you can get an idea of whether the RMS value is increasing or decreasing. Then use smaller changes to find the spot where you get a minimum RMS value.

Adjust the reference signal in the same way, looking for a minimum in the RMS value. You may have to change Ko and S after changing the reference signal, and vice versa.

As you jockey Ko and S values up and down, they won't quite repeat, because the values are being changed by a fixed percentage instead of by a fixed amount. Going up two percent from a value isn't the same change as coming back down two percent from the new value. Just find the best minimum you can without spending a lot of time at it.

You can calculate or recalculate the correlation between the model's handle positions and the person's at any time. Type "c" or "C". In a couple of seconds the new correlation will appear in the right center of the screen (with a beep to tell you it's done—the number may not change).

When you have the lowest RMS value you can get, you will have found the best Ko, S, and r for the model. Because the model is so insensitive to the absolute amount of Ko and S, the most meaningful number is the integration constant Ko/S. As this number changes even a small amount, the RMS value will change significantly.

After you've got the best values of the parameters, exit with the End key. You will be asked if you want to do another run. Answer Y for yes the first time through.

Select a disturbance different from the one you just used, but select the same degree of difficulty, 0 to 3.

After you have done the run, the person and model data will be redisplayed. You will see that the parameters are still set the way you left them. That means that the model is now predicting the behavior, because its parameters haven't been adjusted yet to fit the new run. Type "c" to calculate the correlation of model and person handle. You will see that it is quite high. The disturbance pattern is completely different, yet the model reproduces the handle movements essentially as well as it did before.

If you now write down Ko/S and r, then adjust the parameters for the best fit, you can see how much the best-fit parameters change from one disturbance to another. The change will be very small. These measures are very stable over repeated runs.

You might also want to try using different degrees of difficulty. You will see that Ko/S is sensitive to degree of difficulty (meaning that a different best value will be found). In terms of the model, these differences suggest that the comparator is nonlinear—its response to large errors falls below the response of a perfectly linear device. This means that the loop gain declines when the error is large, as it is (more often) when the difficulty increases. We can't measure the linearity of the comparator independently, so the result we see looks like a change in the output factor (that is, can be imitated by changing the output factor).

You should be getting the hang of the method of modeling by now. The basic procedure is to propose a working model, then run it repeatedly to find the values of its parameters that make it fit real behavior as well as possible. Those parameters then amount to measures of the parameters of the real system, assuming, of course, that the real system is organized the way the model is.

Improving the model

The last demonstration is just like the previous one, but a small change has been made in the model. Now we are assuming that the input function contains a slowing factor, too.

The old input function was just a multiplier of 1: the perceptual signal, p, was just 1 times the controlled variable; when that variable is cursor position, c, p = c. Now we assume that same steady-state proportionality, but introduce a slowing factor that puts a lag into the perceptual process as well as the output process. We know that visual perception does involve some time-lags, so this proposition is not unreasonable. The new adjustable factor is called the "sensory slowing factor" on the graphics screen of this step. The effect of this change will be to slow the feedback a little, which will tend to make the system overshoot somewhat. The result is a small but quite reliable improvement in the fit of the model (for a person who has had enough practice to control skillfully).

This step begins as the previous one did, with an experimental run. If you have already done an experimental run, however, you don't have to do a new one—the data are still stored. You will be asked if you want to do a new run. You can answer the question with 'N' for no, and proceed right to the graphics screen.

Now there is one added parameter that can be changed, a sensory slowing factor. It is initially set to 1, which means no sensory slowing. You should see the same results you found in the previous demonstration, if you did one. If you have done a new run, adjust Ko, S, and r as before for the smallest value of RMS, leaving the sensory slowing factor at 1.

The slowing factor works like the output factor; it's a divisor. As it gets bigger there is more slowing. The actual formula for the perceptual signal p is now

p(new) = p(old) + [Ki * c – p(old)]/Ss, where Ss means sensory slowing factor.

If the controlled variable were to jump from zero to 1 unit, and if the slowing factor Ss were 2, the perceptual signal would become 0.5, 0.75, 0.875, 0.9375 ... and so on. These jumps would occur once per computing cycle. The computing cycles are, in this step and the previous one, synchronized with half the vertical scan rate of the display screen: 1/25 second for a Hercules monochrome graphics screen, 1/30 second for CGA, and 1/35 second for VGA graphics . This gives a physical meaning to the slowing factor.

The slowing factor is adjustable in steps of 0.5. You will find that a value between 3 and 8 will give you a new minimum in the RMS value. But as you adjust the sensory slowing factor, you will find that you have to find a new best value for the integration factor Ko/S, which you can reach just by adjusting S. The best strategy is first to find a new minimum using the sensory slowing factor, then improve that minimum by adjusting the output slowing factor S, then readjust the sensory slowing factor and finally make a last adjustment of S. The best setting of the reference signal may also change. You will probably find that the correlation between model and person handle is now slightly higher, and the RMS value is

perhaps 5 to 10 percent lower than it was without sensory slowing. The correlation is a poor indicator when it is so high—RMS is more sensitive.

For a well-practiced participant, there is always at least some improvement when the sensory slowing factor is adjusted away from 1, so this appears to be a valid addition to the model.

Before you leave this series, try doing a run with a non-zero value of your personal reference signal—that is, keep the cursor or other controlled variable some fixed distance away from the target value. You can measure your own reference signal by adjusting r in the model to get a minimum of the RMS value. Then you can adjust the other parameters in the same way. They should be nearly the same as when you are keeping the controlled variable at the suggested value. You may be surprised at how constant the reference signal remains during the course of a run—a drift will show up in the central "difference" plot as long-term changes in the difference. The reason is that the model definitely maintains a constant reference signal; if you don't, there will be a change in addition to the short-term changes that we normally see. You will probably do better than you would guess just from the way it feels.

Conclusions

I hope you've found this Introduction to Control Theory instructive. I hope it suggests a whole new way of proposing and testing ideas about how behavior works.

I hope, too, that you have been surprised at how well this model reproduces the simple behavior we've been exploring. Common wisdom says that the behavior of organisms is highly variable, so much so that we must do many experiments with many subjects just to get a hint of some regularity. These experiments should have made you skeptical of any such opinion. It's true that the handle movements in these experiments are extremely irregular; if all you could see or hear were the controlled variable and the handle movements, you might conclude that they are both random. But we know there is a hidden disturbance and we know exactly how it affects the variable that the person is controlling. Given that knowledge, and a model that actually fits, we find that the handle movements are almost perfectly predictable for every 1/30 second during a one-minute run. The "variability" is only about 5 percent of the range of movement.

Furthermore, when the model is improved we find that the variability gets a little less. Surely we could find still more improvements, and remove still more of the unpredictable component. An ultimate limit will be set by the natural noise level of the behaving system—but we now know that this natural variability is far smaller than is normally assumed, an order of magnitude smaller. Behavior looks highly variable only when you see it through the eyes of the wrong model.

I suggest that you learn how to construct models and run them on a computer as we have been doing here. Then you won't have to wait on the sidelines while someone else finds the answers to the many questions about the uses of control theory in the behavioral sciences. Why don't you pitch in to help answer them?

Good Luck.
William T. Powers