BYTE 2: July 1979

The Nature of Robots

PART 2: SIMULATED CONTROL SYSTEM

In This BYTE

In part 2 of **The Nature of Robots,** William T Powers presents a BASIC simulation of a control system. By experimenting with this simulator, the reader is able to work with the concepts of a closed loop control system.
. *Page 134*

This article appeared in BYTE magazine, volume 4, number 7, JULY 1979.

The Nature of Robots

PART 2: SIMULATED CONTROL SYSTEM

In part 1, we went through a chain of reasoning that ended with the conclusion that the behavior of an organism is not what it seems. Behavior appears to be at the end of a cause and effect chain that starts with the inputs to a nervous system, but that chain is subject to disturbances that can occur *after* the output of the nervous system. Nevertheless, the behavior at the end of this chain is stable and repeatable, while events closer to the organism become less predictable as we get nearer to the neural signals at the output of the nervous system. By analyzing an example in which a car is maintained in the center of its lane, we saw that this measure of behavior belongs at both the cause and effect ends of the chain, and that if this variable is shown only once in the diagram, a closed loop results.

We are going to look in more detail at the behaving system in this closed loop, to see how it might be organized to produce the results seen. We will start using a simulator written in BASIC which allows the user to vary many parameters of the control system to see the effects on its actions. Human behavior will not be mentioned much in this installment; there are many fundamentals to cover before we can get back to the main purpose of this series. The object here is to retrain the intuition so that the closed loop way of seeing behavior becomes as natural as the old straight through cause and effect way.

Note on North Star BASIC

The method of accessing strings in North Star BASIC is different from that of Microsoft and other BASICs. Translate as follows:

A$(1,n)	becomes	LEFT$(A$,n)
A$(n)	becomes	RIGHT$(A$,n)
A$(m,n)	becomes	MID$(A$,m,n)

Figure, table, and listing numbering continued from part 1.

Organization of a Control System

The simulator (listing 2) is set up to demonstrate the properties of a standard sort of control system organization. We will first look at that organization, then at the simulator itself, and finally at some details of the operation of the control system. You can do much more experimenting than we will discuss here.

***Listing 2:** A control system simulator written in North Star BASIC.*

```
1   PRINT "PROGRAM TWO: SIMULATION OF CONTROL SYSTEM
        BEHAVIOR"
2   PRINT
3   PRINT "AFTER PROMPT (COLON), YOU MAY TYPE"
4   PRINT "'PLOT XXXXXX', WHERE XXXXXX MEANS"
5   PRINT "ANY ONE OR MORE CHARACTERS FROM THE"
6   PRINT "SET P,E,R,I,O,D, IN ANY SEQUENCE."
7   PRINT
8   PRINT "YOU MAY ALSO SET PARAMETERS BY TYPING IN"
9   PRINT "THE PARAMETER SYMBOL IMMEDIATELY FOLLOWED"
10  PRINT "BY AN EQUAL SIGN AND THE VALUE (NO SPACES)."
11  PRINT
12  PRINT "PARAMETERS ARE L, K1, K2, S1, S2, O, P, R, AND D"
13  PRINT "DEFAULT VALUES 16, 1, 2, 1, 1, 0, 0, 0, AND 15"
14  PRINT
15  PRINT "TO RUN, TYPE '.' (INITIALIZE), OR '/' (DON'T INIT)."
16  PRINT
17  K1 = 1
18  K2 = 2
19  S1 = 1
20  S2 = 1
21  P0 = 0
22  O0 = 0
23  R0 = 0
24  D0 = 15
25  V(4) = 1
26  V(5) = 1
27  V(6) = 1
28  INPUT "DISPLAY WIDTH: ",W
29  W = W – 2
30  C = W/2 \ REM                   CENTER OF DISPLAY
31  DIM Z$(W),M$(W),A$120),B$(6),K(6),U(6),E$(72)
32  B$ = "PERIOD"
33  L1 = 15
34  FOR J = 1 TO W
35  Z$(J,J) = " "
36  NEXT J \ REM                    CREATE BLANK FILE
37  DEF FNI(X)\ REM                 INPUT FUNCTION
38  P = P + S1*(K1*X–P)
39  RETURN P
40  FNEND
41  DEF FNO(X) \ REM                OUTPUT FUNCTION
42  O = O + S2*(K2*E–O)
43  RETURN 0
44  FNEND
45  DEF FNF(X) = 0.5*X \ REM        FEEDBACK FUNCTION
46  DEF FND(X) = 0.8*X \ REM        DISTURBANCE FUNCTION
47  REM * *
48  REM * * COMMANDS FOR SETTING PARAMETERS
49  GOTO 51
50  A$ = " " \ IF EI > LEN(E$) THEN 51 ELSE 53
51  INPUT ":",E$ \A$ = " "\E1 = 1
52  IF LEN(E$)< > 0 THEN 53 \ PRINT \ GOTO 51
53  E1 $ = E$(E1,E1) \ E1 = E1 + 1
54  IF EI $ = "," THEN 57 ELSE IF EI >LEN(E$) THEN 56
55  A$ = A$ + E1 $ \ GOTO 53
56  A$ = A$ + E1 $
57  IF A$ = "." THEN 95
58  IF A$ = "/" THEN 99
59  IF A$ < >"?" THEN 62
60  PRINT \ PRINT%7F3,"K1 = ",K1," K2 = ",K2," S1 = ",S1,"  S2 = ", S2
61  GOTO 51
62  IF LEN(A$)< 5 THEN 72
63  IF A$(1,5)< >"PLOT" THEN 72
64  A$ = A$(6)
65  FOR J = 1 TO 6 \ REM            TAG VARIABLES TO
66  V(J) = 0 \ REM                  BE PLOTTED.
67  FOR K = 1 TO LEN(A$)
68  IF A$(K,K) = B$(J,J) THEN V(J) = 1
69  NEXT K
70  NEXT J
71  GOTO 50
72  IF LEN(A$)< 3 THEN 91
73  IF A$(1,3)< >"K1 = " THEN 75
74  K1 = VAL(A$(4)) \ GOTO 50
75  IF A$(1,3)< >"K2 = " THEN 77
76  K2 = VAL(A$(4)) \ GOTO 50
77  IF A$(1,3)< >"S1 = " THEN 79
78  S1 = VAL(A$(4)) \ GOTO 50
79  IF A$(1,3)< >"S2 = " THEN 81
80  S2 = VAL(A$(4)) \ GOTO 50
81  IF A$(1,2)< >"0 = " THEN 83
82  O0 = VAL(A$(3)) \ GOTO 50
83  IF A$(1,2)< >"P = " THEN 85
84  P0 = VAL(A$(3))\ GOTO 50
85  IF A$(1,2)< >"R = " THEN 87
86  R0 = VAL(A$(3))\ GOTO 50
87  IF A$(1,2)< >"D = " THEN 89
88  D0 = VAL(A$(3))\ GOTO 50
89  IF A$(1,2)< >"L = " THEN 91
90  L1 = VAL(A$(3))\ GOTO 50
91  PRINT "???", \ GOTO 50
92  REM **
93  REM **                          SIMULATION AND PLOTTING LOOP
94  REM **
95  P = P0 \ REM                    ENTRY WITH INITIALIZATION
96  O = 00\D = D0\R = R0
97  I = FNF(O) + FND(D)
98  E = R – P \ GOSUB 109 \ REM     PLOT INIT. CONDITIONS
99  D = D0 \ REM                    ENTRY, NO INITIALIZATION
100 R = R0
101 FOR L = 1 TO L1 \ REM           CONTROL LOOP SIMULATION
102 I = FNF(O) + FND(D)
103 P = FNI(I)
104 E = R–P
105 O = FNO(E)
106 GOSUB 109 \ REM                 CALL PLOTTING SUBROUTINE
107 NEXT L
108 GOTO 50
109 REM * "
110 REM **                          PLOTTING SUBROUTINE
111 REM
112 U(1) = P + C
113 U(2) = E + C
114 U(3) = R + C
115 U(4) = I + C
116 U(5) = 0 + C
117 U(6) = D + C
118 PRINT
119 M$ = Z$ \ REM                   CLEAR OUTPUT BUFFER
120 M $(C + I ,C + 1) = "." \ REM   MARK SCREEN CENTER
121 FOR J = 1 TO 6 \ REM            LOAD BUFFER
122 U = INT(U(J) + .5) + 1
123 IF U< 1 THEN U = 1
124 IF U> W THEN U = W
125 IF V(J) = 1 THEN M$(U,U) = B$(J,J)
126 NEXT J
127 PRINT M$, \ REM                 PRINT BUFFER
128 RETURN
999 END
```

Figure 5 is a diagram of a typical control system. Almost every control system can be expressed in this form, although in the real system, functions that are shown here as separate are often combined into one physical entity. The symbols for functions and variables are those which appear in the BASIC simulator.

The behaving system is entirely above the boundary line. All that is not the behaving system (or systems inside the organism at a higher level, not considered here) is called the *environment* of the system. Variables inside the system will always be called *signals,* and variables in the environment will always be called *quantities.*

In the environment we have three quantities mentioned in part 1. The *input quantity* is a physical variable that the system can sense. The state of this quantity is the result of all influences acting on it (which in our limited universe means the influence from the system's own output) and one representative *disturbing quantity* that can vary independently from what the system does. The system's output is represented by the *output quantity.* The input quantity is called I, the output quantity O, and the disturbing quantity D.

The output and disturbing quantities are separated in space from the input quantity, and they influence the input quantity through properties of the intervening environment. The connection that translates the state of the output quantity into an influence on the input quantity is called the *feedback function,* symbolized in BASIC as FNF. The function that translates the state of the disturbing quantity into another influence on the input quantity is the *disturbing function,* symbolized FND. If the input quantity is associated with some physical object, then FNF and FND may both contain properties of that object (eg: its mass). There are less redundant ways to handle this in special cases.

The meaning of the previous paragraph is summed up in line 102:1 = FNF(O) + FND(D). The state of the input quantity is the sum of the influences from the output quantity and the disturbing quantity. In the real world, both the output quantity and the disturbing quantity may have many effects other than those on I, but those effects are irrelevant to the operation of this system (perhaps not to the designer or user of the system, if it is artificial). We have therefore considered everything about the environment that is of interest here.

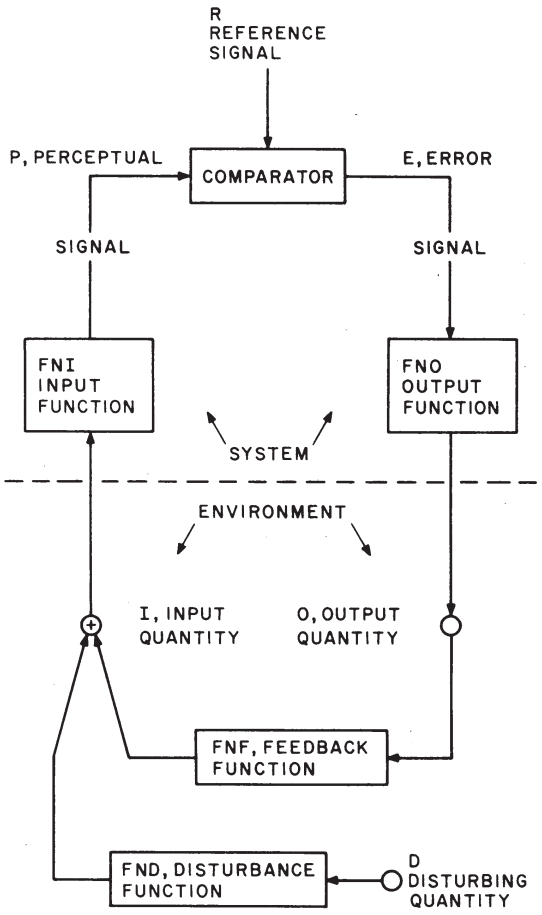

Figure 5: *The system's output quantity, 0, influences the input quantity, I, via the feedback function, FNF. The disturbing quantity, D, influences the input quantity via the disturbance function FND. Both FNF and FND represent physical links in the environment. The state of the input quantity is determined by the sum of these two influences.*

The system's input function, FNI, converts the state of the input quantity into a magnitude of the perceptual signal P. P is compared with the reference signal R in the comparator function, which emits an error signal E = R – P. The error signal is converted into a magnitude of the output quantity via the output function, FNO.

Above the line we have the behaving system. We cross the boundary at the *input function,* FNI. This is the function which turns the state of an external quantity, I, into the magnitude of a *perceptual signal,* P. Both sensors and computing processes may be involved in a complex input function. The outcome, however, is always the magnitude of a single signal, whatever it represents. This signal can only increase or

Anatomy of the Simulator

Let's run through the simulator quickly before we start using it, to see how this control organization operates.

Lines 1 thru 16 are user instructions. Lines 17 thru 27 initialize the system in a way that will be used to illustrate a point. Lines 28 thru 33 do more initializing, and ask for the width of your display. Lines 34 thru 36 create a blank string in case your BASIC doesn't set dimensioned strings initially to spaces.

Lines 37 thru 46 define the various functions of the control system. If your BASIC can't do multiline functions, you can substitute subroutines here. The idea is to make it easy to try out different kinds of functions in the control system.

Lines 49 thru 91 comprise the interpreter, which accepts character strings and sets initial conditions and parameters before each run. Variables are initialized and constants are set by typing a string of the form A=m or An=m (no spaces; terminated by a carriage return). To set up the plotter, the statement is PLOT XXXXXX, where XXXXXX is one or more characters from the set P,E,R,I,O, and D, in any sequence. The plotter comes up set to plot P, E, and R. If you forget the last values of the parameters K1, K2, S1, and S2, type ? and they will be printed out. We will eventually define them,

The control system itself is simulated from line 95 to line 108. Entering the simulator at line 95 initializes the perceptual and output variables to values given to the interpreter. Entering at line 99 runs the simulation from the conditions left at the end of the last run. This is taken care of by the two run commands in the interpreter: a dot (.) means run with initialization, and a slash (/) means run without initialization. All commands require a carriage return termination.

The plotting subroutine goes from line 112 to line 128. Its operation deserves a note, since it was arrived at after some more normal schemes were rejected for being too slow. When the interpreter is given a string of symbols to set up the plotting, a table is set up (V(j)) in which a 1 means plot and a 0 means don't plot. When the plotter is entered, it transfers all six variables to another table, U(j). The output buffer is then cleared, and a short loop scans the V table, picking up variables from the U table when V(j)=1, and putting the symbol into the output buffer in a position corresponding to the value of the variable. Then the output buffer is printed out. This eliminates sorting the variables by size or printing the line as many times as there are variables. This method nicely cures the fundamental "rheumatism" of BASIC, as it is able to plot about two lines per second on my Polymorphics VTI display.

When two variables fall on the same spot, the variable that actually appears is the latest one in the series PERIOD. Thus far it has always been easy to figure out where a missing variable is hidden.

Once we have a set of variables connecting functions together, and an overall arrangement, we can treat the system by assembling it piece by piece. Let's look at the pieces we have, represented by the four statements in listing 2 from line 102 to 105:

```
102  I = FNF(O) + FND(D)
103  P = FNI(I)
104  E =  R – P
105  O = FNO(E)
```

Looking at figure 5, we can see that these four statements lead us clockwise around the closed loop. I is the result of combining the outputs of the feedback and disturbance functions. it becomes the input to the input function, producing a value of the perceptual signal P. P is one of the inputs to the comparator, which produces the error signal E.

E is the input to the output function that produces O, the output quantity. The output quantity is the input to the feedback function, which leads us back to the start.

It might seem that all we have to do now is to supply some specific forms for the functions, and turn the system on to see what it will do. In a sense, this is right. If this were an analogue computation, we might even get a correct idea of how the system works. However, it is unlikely that anyone who hasn't done this before would plug in the right functions to make a digital computer give us anything more than a fairy tale. It is so important to understand this point that l have written the simulator to come up initialized in order to illustrate it.

Using the Simulator

The simulator is run from the keyboard, using commands that tell it which variables to plot and what values of variables and parameters to start with. The instructions can be given one at a time, terminated by carriage returns, or they can be given in a continuous string with commands separated by commas. The latter is useful for altering parameters in the middle of a plot in order to see their effects.

The only time a space is permitted in a command or string of commands is when it is separating the word PLOT from the string of variable symbols to be plotted.

In order to tell the simulator what variables to plot, type:

PLOT XXXXXX

where XXXXXX means a string of 1 to 6 symbols from the set PERIOD. The order of the symbols makes no difference. When two or more symbols land on the same plot, the one that you see is the latest in the series PERIOD, regardless of the order in which they were given.

To start a plotting run, type a period followed by a carriage return or comma if initialization is to occur first, and type a slash (/) if the run is to start from the conditions at the end of the previous run. Initializing creates one extra line of plot showing the initial conditions.

The parameters and variables that can be set are as follows:

L	Number of lines to be plotted in any plotting run.
K1	Steady state proportionality factor of the input function.
S1	Slowing factor for the input function; positive and between 0 and 1.
K2	Steady state proportionality factor of the output function.
S2	Slowing factor for the output function; positive and between 0 and 1.
O	Initial value of output quantity.
P	Initial value of perceptual signal.
R	Setting of reference signal.
D	Magnitude of disturbing quantity.

Examples: (colon is prompt from computer. Always terminate a string with a carriage return).

Set L to 16	:L=16
Set D to 0, run without initializing	:D=O,/ or :D=O :/
Set D to 0, plot 2 points after initializing, set D to 10, plot 13 points from previous conditions. Plot P, E, and R	:PLOT PER,D=0,L=2,.,D=10, L=13,/

The program is written so that after a plot is completely done (a complete string has been interpreted), the prompt character appears to the right without a carriage return. That allows a 16 point plot to be shown on a 16 line video display screen without the final carriage return bumping the first line off the screen. If you want your next string to start at the left, just hit a carriage return.

To find out the values of K1, K2, S1, and S2 when you forget them, type "?" followed by carriage return and they will be printed.

decrease; we will always work with one-dimensional control systems, treating multi-dimensional control phenomena by using multiple control systems. The perceptual signal is the system's internal representation of the external world—its only such representation.

Line 103 expresses the definition of the input function and the way it relates the input quantity and perceptual signal: P = FNI(I).

Inside the system is another signal, the *reference signal,* R. In living systems, this signal is generated elsewhere in the organism; it is not accessible from outside. The reference signal, along with the perceptual signal, enters a function called the *comparator,* which subtracts one signal from the other and emits an *error signal,* E, representing the signed difference of magnitudes. It does not matter which signal is subtracted from which, but for uniformity we will always treat the reference signal as the positive input and the perceptual signal as the one subtracted from it. Thus, a positive error signal always means that the reference signal is larger than the perceptual signal. This function does not have to be generalized, as nonlinearities and amplification can always be absorbed into one of the other functions.

Therefore line 104 represents the comparator without using a function; it is the comparator function itself: E = R – P.

The error signal drives the output of the system via the *output function,* FNO. The output of the system, therefore, depends not on the input quantity or the perceptual signal alone, but on the *difference* between the perceptual signal and the reference signal. The output function translates a signal inside the system into a quantity outside it, according to whatever rule is described by FNO. If the error signal changes sign, the output quantity also changes; in other words, we assume that output functions have no constant term. Any such constant term would have the same effect as a reference signal, creating an offset in the overall system response. Not every system can handle error signals and output quantities that go through zero and thus change sign, but the principles remain the same in the region where the system works.

Line 105 expresses the operation of the output function: O = FNO(E). This closes the loop of cause and effect since the output quantity appears in line 102 where the input to the system is calculated.

If the system functions are properly designed for the properties of the system's environment, this entire closed loop will seek an equilibrium state. Our simulator will let us look at time-varying effects, but for the most part we will be concerned with steady state relationships.

Once we have seen how time variations come into the picture, we will concentrate on variations that occur slowly enough that the system and its environment never get far from a steady state relationship. This is the whole trick in grasping how control systems work. If you allow yourself to become embroiled in the interesting details of stabilization, or interested in the limits of performance in the presence of large and rapidly changing disturbances, you may learn a lot about one control system, but you will miss the organizational features that are obvious only when the system is not being subjected to unusual stresses. We will be concerned mainly with the *normal range of operation,* the range within which this system can behave very nearly like an ideal control system. Once that mode of operation is understood, there is plenty of time to explore the limits of operation. (See "Anatomy of the Simulator" text box).

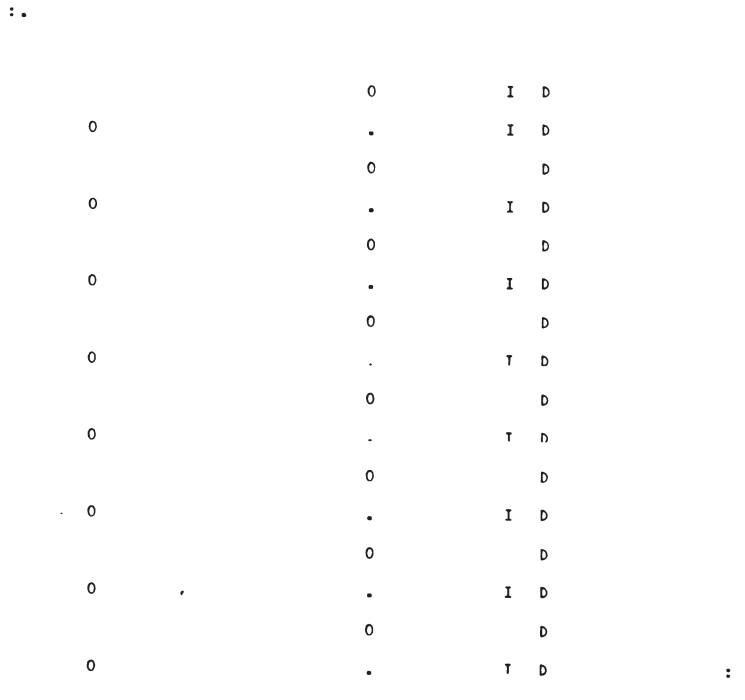

Figure 6: *The initial plot generated by the BASIC simulator. Disturbance is set to 15 units and the reference signal is initialized to O. The system is in a state of oscillation.*

A Wrong Approach

Let us start off by assuming that we have a simple linear system. The input function is a multiplier of 1, the comparator is already simple and linear, the output function is a multiplier of 2, the feedback function is a multiplier of 0.5, and the disturbance function is a multiplier of 0.8. These choices are dictated partly by the need to keep variables from falling on each other when we plot them. The simulator initializes D to 15.

Our four system equations, with these values substituted, now look like this:

```
I = 0.5xO + .8xD=0.5xO + 12      (1)
P = I                            (2)
E = R–P                          (3)
O  = 2xE                         (4)
```

This system of equations is iterated during a simulation of behavior.

The above is a pretty simple system of equations. So why can't we just solve it algebraically and skip the rest? I suggest, in fact, that you do solve it (by successive substitutions). Solve for the value of the perceptual signal in terms of R and D. You'll get P=I=(R – 0.8 x D)/2.

Ready for a shock? Your computer can't come up with that solution! Let's fire up the BASIC simulator, which is initialized according to equations 1 thru 4 above, and plot I, D, and O. Type RUN, and answer the question with a reply that tells the width of your display. After the colon prompt appears, type in the following:

```
: .
```

I trust nobody had trouble with that.

The dot says "do a plotting run after initializing the variables." A slash (/) would say "do the run from where the last run left off." The result can be found in figure 6.

The disturbance is set to a steady + 15 units, and the reference signal is initialized to 0. According to the algebraic solution above, the input signal should be a steady 0.8 x 15/2, or 6 units, to the right of center (dots indicate center when nothing is there). It is clear that something else happened. The whole system is in a state of endless *oscillation*. (When variables fall on top of each other in a plot, the visible one is the latest in the sequence PERIOD.)

Nature has a way of slapping your wrist when you forget something important. Our wrist has just been slapped. Naturally we do not get the same result that algebra gives: the algebraic solution comes from treating all of those relationships *simultaneously*. Our computer program is treating them *one at a time*. The algebra says that if one variable changes, they all change. The computer, being a purely sequential machine, thinks it can change one variable without changing the others. If the physical system being modeled is of that nature—if it, too, is a sequential state machine—then the computer will produce a correct picture of behavior. But, if the system being modeled works in terms of continuous variables, *even in part,* the computer will turn it into a sequential-state machine and analyze *that* kind of system instead of the one we actually have. That is what has happened here. We forgot to tell the computer that these variables can't change as fast as the computer can compute.

A More Accurate Approach

In order to make this simulated system behave the way the algebra says it should, we have to slow down changes in one or more variables to take account of the fact that we are dealing with real, physical variables and not abstract numbers. The simulator does this in the input and output functions, lines 37 thru 40 (input) and 41 thru 44 (output). We will be basically dealing with a linear system in which both the input and output functions are constants of proportionality. As you can see from listing 2, however, there's a little more to it than that.

Consider line 42: O = O + S2* (K2*E – O). The O on the left side is the new value of that quantity after this program step has been executed. On the right side, O indicates the last value of the output quantity. We recognize K2*E as a calculation of the output quantity as if it were simply proportional to the error signal, E. The expression in parentheses, therefore, is the *difference* between this calculated new value and the old value of O. This is how much the output quantity would change if it could change instantly.

This calculated amount of change is multiplied by S2, a *slowing factor,* and the result is added to the old value of O. We calculate the amount of change that an instantly reacting system would produce, but allow only a fraction S2 of it to occur on any one iteration. S2 is a positive number between zero and one. We've put a low-pass filter into the output function, without affecting the *steady state* proportionality constant.

The same thing is done for the input function. A slowing factor S1, between zero and one, acts to slow P down. We need only one slowing factor to make this simulator behave realistically, but there is provision for two, so that you can explore the effect of having two if you wish. In all the plots to follow, we'll use a modest slowing factor of S1=0.5 in the input function, and essentially all of the required slowing in the output function. Once you get the hang of this you can put slowing factors into *any* of the functions.

The simulator is initialized with S1 and S2 set to 1, which reduces O + S2x (K2xE – O) to O + K2xE – O or just K2xE (no slowing at all). The same is done for the input function. Let's set them to other values and see what happens. The values of S1 and S2 can be set by typing S1=n or S2=n and a carriage return:

```
:S1=0.5
:S2=0.2
:.    (run with initialization)
```

Suddenly we see nice, smooth relationships (figure 7). If you measure, you'll see that the input signal, I, ends up just six units to the right; the same solution given by the algebraic approach.

Does this mean we can just use algebra to analyze a control system? Not at all. We won't delve into this, but the algebraic solutions are valid *only* if the differential equations which really describe the system have steady state solutions. Then the algebraic solutions *are* the steady state solutions. In our simulator, we see all the time variations that lead toward the steady state, and the algebra says nothing about these. By putting the slowing factors into our calculations we have caused this system to seek a steady state. Therefore, it is the stability of the system that tells us we can use algebra, not the other way around. Predicting stability can become a messy process. We fiddle around with slowing factors until we *get* stability, which is more or less how Nature does it anyway.

```
:S1=0.5
:S2=0.2        O           I  D
:.           O .           I  D
          O    .          I   D
       O       .        I     D
     O         .       I      D
    O          .      I       D
   O           .      I       D
   O           .     I        D
   O           .     I        D
   O           .     I        D
   O           .     I        D
   O           .     I        D
   O           .     I        D
   O           .     I        D
   O           .     I        D
   O           .     I        D                :
```

Figure 7: *The slowing factors have been changed. S1 equals 0.5 and S2 is 0.2. We now have a much smoother curve.*

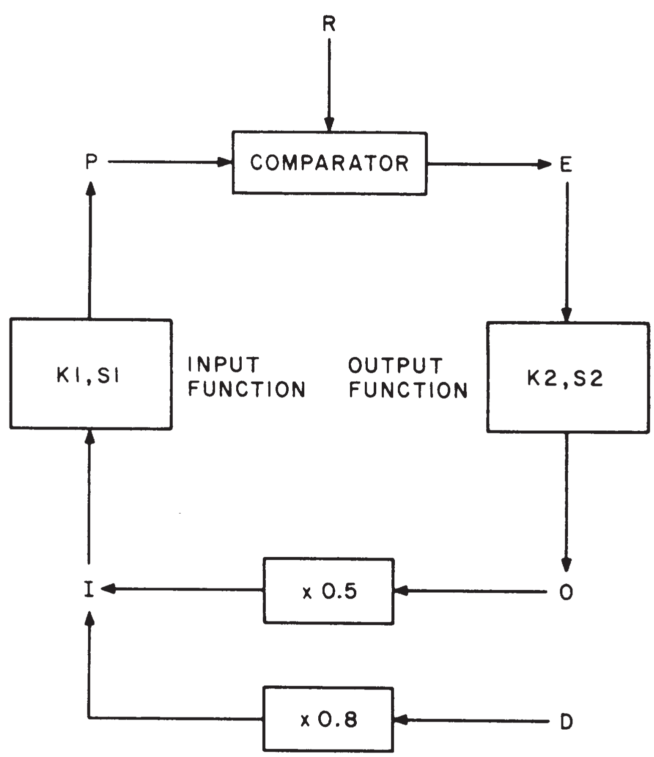

Figure 8: *Adjustable parameters are K1 (input sensitivity), S1 (input slowing factor), K2 (output sensitivity), and S2 (output slowing factor). P and O can be initialized to any starting value (normally zero). R and D can be set, and remain the same during a run. The value of the feedback function is set at 0.5, the value of the disturbance function at 0.8.*

We have now established the fact that using natural logic and following causes and effects around the closed loop as a sequence of events will lead to a wrong prediction of control system behavior. This immediately eliminates three-quarters of what biologists, psychologists, neurologists, and even cyberneticians have published about control theory and behavior. We are just beginning to see that one must view all the variables in a control system as changing together, not one at a time. This is what I mean by retraining the intuition. Cartesian concepts of cause and effect, and Newtonian physics, have trained us to think along directed lines. What we need to do to understand control systems is to learn how to think in circles.

Properties of a Control System

Figure 8 shows the control system and its environment as we will be dealing with it from now on. Let's start with some definitions:

Loop Gain means the product of all the steady state factors encountered in one trip around the closed loop, counting the comparator as a factor of –1. In the initial setup, K1 was 1, K2 was 2, and the feedback function FNF was a multiplier of + 0.5, so the loop gain was –1. The sign of the loop gain is the sign of the feedback; we have (and will continue to have) negative feedback.

Error Sensitivity is the factor K2, the steady state proportionality factor in the output function FNO. This number expresses how much output will be generated by a given amount of error signal.

Input Sensitivity is the factor K1, the steady state proportionality factor in the input function FNI. This number expresses how much perceptual signal will be generated by a given amount of input quantity.

We are going to perform a series of experiments with this control system in order to arrive at some useful rules of thumb for thinking about how control systems work. These rules are approximations, but by doing the experiments and seeing how good the approximations are, you will learn to think precisely about control phenomena, even when using approximate language.

We will set the system parameters to give a loop gain of –10. As a way of summarizing where we are (refer to figure 8), the commands are given one at a time with annotations:

```
:K1=1      Input sensitivity = 1.
:K2=20     Error sensitivity = 20.
:S1=0.5    Input slowing factor = 0.5.
:S2=0.07   Output slowing factor = 0.07.
:R=0       Reference signal = 0.
:O=0       Output initialization = 0.
:P=0       Perception initialization = 0.
:D=0       Disturbance = 0.
```

Type those commands, and the system is now set up in a "home base" condition. Remembering that the comparator is equivalent to the factor of –1 and the feedback function is permanently set to be a factor of + 0.5, this combination of parameters gives a loop gain of 1 x (–1) x 20 x 0.5 = –10.

There are two fundamental rules of thumb: a control system keeps its perceptual signal matching its reference signal, and the output of a control system cancels the effects of disturbances on the input quantity. We will take these up in order.

Rule 1: P = R

We're looking at the system with no disturbance acting (D=0). If you want to be sure that everything stays at zero, type PLOT PERIOD . followed by a carriage return. You will see a row of Ds, D being the last symbol in the sequence PERIOD and hence the only one visible when all variables are at zero.

Now we will plot just the reference signal and the perceptual signal. The first two points will be done with the initial conditions set up above. The reference signal will then be set to + 25 units, and the plot will be continued for 13 more points. Since this plot will commence *with* initialization (the dot command), an extra line showing the initial conditions will be plotted first. This makes a total of 16 lines, which will fit on most video displays. Of course, if you're doing this on paper you don't have to worry about the number of points plotted. Here is the command string:

```
:PLOT PR,L=2,.,R=25,L=13,/
```

Before discussing this, let's do another run of 13 points (figure 9), setting the reference signal to –25 units and continuing without initialization (the slash command,/):

```
:R= –25,/
```

It is clear that the perceptual signal comes to a steady state quite close to the magnitude of the reference signal, whatever the reference signal may be. The question is, how critically does this tracking effect depend on the input sensitivity and error sensitivity?

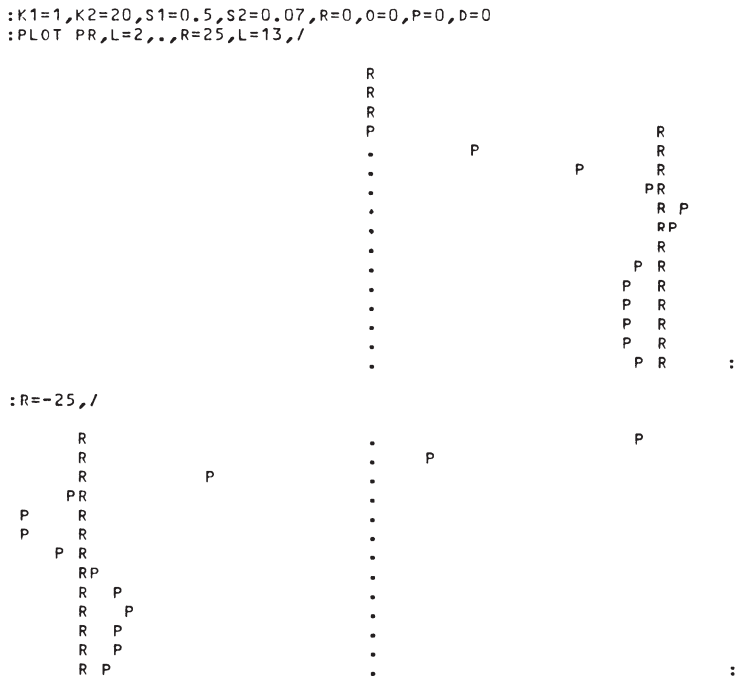

Figure 9: *The values of variables are listed in this plot. The disturbance value is changed from +25 to –25.*

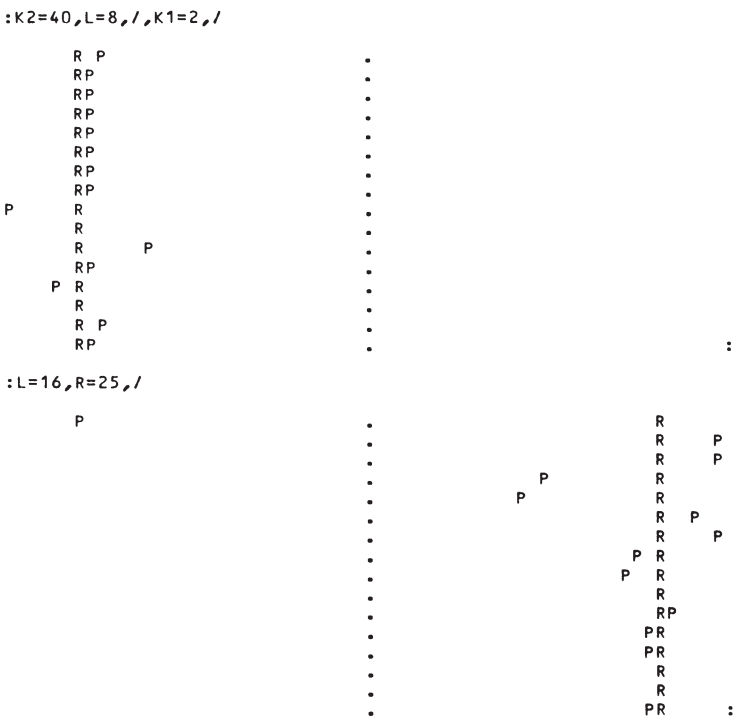

Figure 10: *Change of gain during plot. After 8th line, gain goes from 20 to 40. Reference signal is changed to check operation.*

Let's leave the reference signal at –25 and do a run in which the error sensitivity is doubled at the start, and the input sensitivity is doubled halfway through the run. We will start from the previous conditions. The loop gain will now be –40 instead of –10.

:K2=40,L=8,/,K1=2,/

To insure that everything is working correctly, let's flip the reference signal to + 25 units (figure 10):

: L=16, R=25,/

While there is an effect on the *way* the tracking takes place, the only effect of these rather drastic changes in input and error sensitivity is to make the tracking a little better. What about a *decrease* in these parameters?

:L=16,K1=0.5,K2=10,/, R=25,/
(Loop gain now 2.5)

Figure 11 shows that the approximation P=R isn't very accurate any more. For loop gains smaller in magnitude than about 10 (negative), the approximation begins to lose accuracy.

You will notice that doubling the error sensitivity, which doubles the amount of output generated by a given error, does *not* double the amount of output that actually occurs. Far from it. When, for any reason, the loop gain goes up, the steady state error simply gets smaller, assuming that the system remains stable. This fact does violence to the popular idea that the brain commands muscles to produce behavior. If that were the case, doubling the sensitivity of a muscle to the nerve signals reaching it ought to produce twice as much muscle tension. Nothing of the sort happens, unless you've lopped off the rest of the nervous system, particularly the feedback paths.

As long as the loop gain is sufficiently large and negative (–10 or more negative will do for a number), a stable control system will match its perceptual signal nearly to its reference signal, regardless of the reference setting. We are ignoring, of course, transient effects.

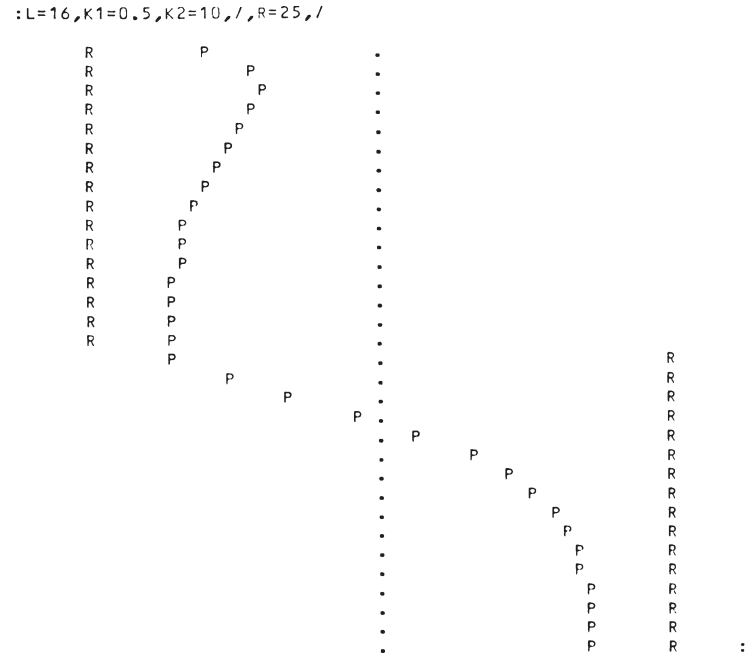

Figure 11: *The simulation parameters have been changed to produce a gain of 2.5. Notice that the approximation P=R is now inaccurate.*

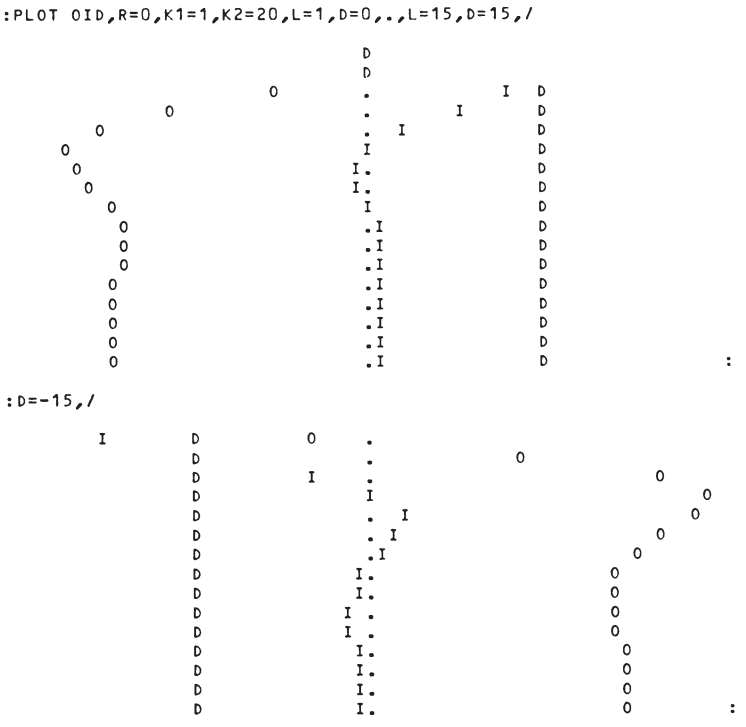

Figure 12: *The reference signal has been set to zero. This plot shows us the input quantity, the output quantity and the disturbance signal for D= +15 and then D= –15.*

All of this was done with the disturbance set to zero. Now let us set the reference signal to zero, and check the second fundamental rule of thumb.

Rule 2: (delta O) = –(delta D)

This rule requires some interpretation. It says, for the sake of brevity, that (with the reference signal constant) a change in the output quantity is equal and opposite to (the minus sign) a change in the disturbing quantity. Generally, the input and disturbing quantities will affect the input quantity through different physical paths. In our model, the output quantity acts through a multiplier of 0.5, and the disturbance through a multiplier of 0.8. The rule has to be interpreted to mean that the *effects* of the changes *on the input quantity* are equal and opposite. We will see this demonstrated.

We will now plot the output quantity, O, the disturbing quantity, D, and the input quantity, I (to make the above clear). The reference signal could be left where it is, but to avoid confusion let's set it to zero for this set of plots. The loop gain is set to –10.

:PLOT OID, R=0,K1=1,K2=20,L=1,D=0,
.,L=15, D=15,/
Let this plot run out, then:
:D= –15,/

There is some lurching back and forth in figure 12, but in the steady state the behavior of the input quantity shows that the effect of the disturbance is essentially cancelled by the final effect of the output quantity.

If you did some measuring on the plot, you would find that the final value of the output quantity is very close to 8/5 of the value of the disturbing quantity. This follows from three facts: the input quantity ends up nearly at zero; one unit of output has 0.5 unit of effect on the input quantity; one unit of disturbance has 0.8 unit of effect on the input quantity. This is the kind of reasoning that helps in understanding how a control system works.

The primary observation about a control system is always the existence of an input quantity which is stabilized against disturbances by variations in the output quantity. If the input quantity is held essentially constant (in the steady state), then one *can deduce the* relationship *between* disturbances and the system's output quantity simply

Figure 13: *A look at different reference signal effects. As explained in the text, the disturbance signal has made the perceptual signal match the reference signal.*

from observing the properties of the system's environment. On inspection, an external observer can see both the feedback function and the disturbance function, here multipliers of 0.5 and 0.8 respectively. For any given disturbance, the effect on the input quantity for a constant output quantity can be calculated on purely physical grounds. Since the input quantity remains undisturbed in the steady state, one can then look at the connection between the output quantity and the input quantity, and deduce how the output quantity must change to account for the fact that the input quantity *doesn't* change.

Thus, in order to predict how this system will react to any external disturbance, it is necessary only to know that the system is a control system and to look closely at its environment. The kind and amount of reaction follow from the nature of the feedback and disturbance functions which are properties of the visible environment.

Most important, as far as the life sciences are concerned, the form and amount of *reaction* do *not* depend *on any* property *of* the control system; not enough to make any difference. Therefore, when you apply a stimulus and see a response, you are using the organism as a complicated analogue computer in order to study the physics of the local environment. This is not what the life sciences have thought they were doing.

All that remains to wrap up this section is to see the effects of disturbances when the reference signal is set to different values. This will lead to the definition of a useful technical term: the *reference /eve/ of the input quantity* (see figure 13):

```
:PLOT RIOD,D=0,R=0,L=1,.,R=12,
L=15,/,D=15,/
```

If you have a 16 line video display this will scroll past you, losing the early parts, but no matter. The first event is that the reference signal is set to 12, and the input quantity moves essentially to + 12. The output quantity goes to + 24 in order to accomplish this. Then the disturbing quantity goes to + 15, which has the exact effect on the input quantity that + 24 units of output have. As a result, the output quantity drops to zero—*exactly* zero, if you look at the numbers.

In effect, the disturbance, by itself, has enough effect to make the perceptual signal match the reference signal. Looking at figure 8, you can see that this would mean a zero error signal and no drive to the output function. So, whenever the output drops to zero, we know that the perceptual signal is matching the reference signal, even if we can't see it.

In our model right now, the input sensitivity is 1, so the perceptual signal is numerically equal to the input quantity. That's a coincidence, since the units are different: physical units outside, impulses per second inside. Even if K1 wasn't 1, the output would still drop to zero when P = R. Thus, we can give a special name to the particular value of input quantity (however created) that brings the error signal, and hence the output quantity, to zero: *the reference level* of the input quantity. The reference signal clearly determines what this reference level will be, but so does the form of the input function.

Main Points Reviewed

All of this is supposed to have established two principal ideas. The first is that control systems control what they sense, not what they do. The second is that control systems act on the outside world only in order to protect a controlled perception against disturbance.

As we have demonstrated these principles, we have established some other odd facts. We have found that the main effect of negative feedback in a control loop is to diminish the effects which disturbances would otherwise have on the system's input quantity. While we have had only one disturbance at our disposal, it should be clear that the number or the causes of disturbances make no difference. If ten different disturbances were acting at once, they could only end up increasing or decreasing the value of the controlled input quantity. Since the system maintains control by acting directly on the input quantity, and not by acting to oppose the cause of the disturbance, the system does not have to take account of the number of causes acting, or the phenomena that are involved. It acts to oppose the *net effect* of any disturbances on the input quantity.

From the point of view of the behaving system itself, reality consists of the magnitude of one perceptual signal, because that is the only internal representation of the outside world. If the system can be said to have a purpose or intention, it must be to maintain the perceptual signal matching the reference signal. The reference signal specifies to the system what it is to sense, but not what it is to do. The output that matches perceptual and reference signals is determined by the nature of the feedback function and by the strength and direction of any disturbances that may be acting. Whatever sets the reference signal, thus effectively controlling the perceptions of this system, does not have to know anything about *how* the control system comes up with a matching perception.

What is perhaps most amazing to a person who has not previously worked with negative feedback systems is the capability that this system has to maintain quite precise control over its own perceptual signal, even if its own properties change. If its output apparatus becomes stronger or weaker, or its perceptual apparatus becomes more or less sensitive, there is scarcely any effect on the perceptual signal. As long as some *minimum* loop gain is maintained and the system does not become unstable and begin oscillating, it does not really matter how much loop gain there is, or whether most of it is in the output or the input function.

A servomechanism engineer might find this approach somewhat odd. Why all this fuss about the system's internal perceptual signal? When you build a control system for a practical use, you worry more about the external variables than internal variables, because the customer is interested in the external variables.

This is exactly the point. Living control systems are *not* interested in the external variables. They can't be. They don't know about them, except indirectly. All they know is what happens to themselves. The point of behavior is not to accomplish something for a user in the external world, but to affect the system itself. Everything that a living system knows about the outside world has to first exist in the form of perceptual signals, or some other internal effect of external events (not all organisms have nervous systems).

In part 3 we will start looking at living systems more directly, and this will become much clearer. We now know that control systems control, above all, their own internal perceptual signals. Next time we will see *why* they do that.

In the meantime you might enjoy using this simulator to do further explorations. We have looked into only a few of the questions that might be raised about control systems. The simulator can reveal far more than we have seen. For example, it is instructive to look at the effects of the disturbance strictly from the external point of view (plotting I, O, and D), and then to look at exactly the same effects from inside (plotting P, E, and R). We haven't even raised the question of what a control system looks like when it becomes unstable, how the slowing factors interact with loop gain to determine stability, or what happens when the input function, the output function, or both are nonlinear. Speaking of nonlinearity, you might try rewriting the definition of the feedback function as follows:

```
45 DEF FNF(X)=X*X*X/2048 + X/2
```

and then performing some of the experiments again. Try to make the input function *logarithmic* (adding a constant to make sure you don't make the perceptual signal negatively infinite), and see how the input quantity and perceptual signal behave as the reference signal or disturbance is changed.

The main objective before the next article in this series appears is to understand how a control system controls its perceptual signal, and why an external observer, who doesn't know about the controlled input quantity, might think the disturbance acts on the system to make it respond, like a doorbell. The simulator is there to help you grasp this closed loop phenomenon. I hope it *does* help. ■