A COMPUTER-SIMULATED ON-LINE EXPERIMENT IN LEARNING CONTROL SYSTEMS

J. D. Hill, G. J. McMurtry, and K. S. Fu
Control and Information Systems Laboratory
School of Electrical Engineering
Purdue University
Lafayette, Indiana

INTRODUCTION

The structure of a learning control system combines digital memory and logic circuitry with an adaptive system. Data obtained through adaptation is stored (remembered) and later utilized to improve the system performance. The learning control system will operate satisfactorily under changing environmental conditions in which an adaptive system fails to improve the performance. The general learning system operation is outlined in this paper and a simple experimental system which illustrates the improved performance is discussed. The example system is a second order plant in which the damping ratio and undamped natural frequency are considered to be affected by the environment in a piecewise constant manner. The system was simulated on a GEDA analog computer and the memory and logic functions were supplied by an IBM 1620 through IBM 1711 and 1712 analog to digital and digital to analog equipment.

LITURATURE SURVEY

The words “learning system” and “self-organizing” appear frequently in the literature in the area of automata studies and artificial intelligence. Hawkins\(^1\) has given an excellent review on self-organizing systems as well as an extensive bibliography, while Minsky\(^2\) has reviewed various aspects of artificial intelligence including hill-climbing, methods of classification for pattern recognition, reinforcement learning, problem solving machines and various other aspects of artificial intelligence. Nearly all of these systems utilize some form of memory and do improve their performance with time. However, they require an off-line training period involving a human operator to judge whether or not their response to a given stimulus is satisfactory, i.e., they lack a built-in index of performance.

A typical example of this is given by Gabor, Wilby and Woodcock\(^3\) who have constructed a universal nonlinear filter which will organize its internal parameters in such a way as to act as an optimum filter, predictor, or simulator of an unknown mechanism, depending upon how it is trained. The important point is, however, that it must be trained by a human operator before it can function in the desired manner. This is typical of the general class of so-called learning or self-organizing pattern recognition machines. “Pandemonium,” \(^4\) “Perceptron,” \(^5\) and “Adaline” \(^6\) are other pattern recognition devices with self-organizing capabilities. All, however, require off-line training by human operators.

Thus, though some of the techniques which have been developed for pattern recognition and automata theory may be useful when applied to the learning control problem, there has as yet
been very little direct application to control systems.

In the hierarchy of automatic control systems, adaptive systems are more sophisticated than standard control systems. Truxal\(^7\) gives a good general review of adaptive control systems while Mishkin and Braun\(^8\) discuss several specific examples. The next advancement beyond adaptive systems appears to be in the area of learning control systems.

Krug and Letskii\(^9\) have suggested the use of a learning process for optimum control. In their paper they suggest that the optimum control of slow but complex processes, such as chemical processes, might be found by a systematic evaluation of input and output data and an index of performance. Probabilistic methods are suggested for organizing the memory for optimum search procedure. The system as outlined is very general and a human operator is left to make the final decision as to whether the control found by the so-called automaton is satisfactory. Only cursory attention is given to the possibility of replacing the human operator.

A considerable effort has been directed toward the investigation of learning in automatic control systems at the Control and Information Systems Laboratory, School of Electrical Engineering, Purdue University. An informal introduction to learning control systems has been given by Fu.\(^10\) One of the specific systems investigated at Purdue University\(^11\) is discussed in this paper.

**Philosophy of Learning Control Systems**

The structure of a learning control system, as visualized at present, is distinguished from an adaptive system mainly in that in addition to a conventional adaptive system the learning system has memory and logic. Data obtained through adaptation is stored and later utilized to improve the system performance. Thus, whereas an adaptive system may take advantage only of its immediate past experience, a learning system is able to recall and use plant adjustment data obtained through adaptation while operating under similar environmental conditions in past time. An adaptive system will optimize a slowly time varying plant to a given index of performance (IP), often through a hill-climbing technique, by modifying the controller or plant parameters. Thus, if a sufficient range of parameters is available, the adaptive system will become optimum for a given index of performance. A basic constraint is that the plant should vary slowly enough that the adaptive loop is able to track the minimum of the index of performance and maintain relatively constant performance.

In some instances, the plant parameters vary so quickly that the adaptive loop cannot maintain optimum performance although some adaptive action does take place. This is one type of situation in which the learning system proposed by the authors may be applicable. Learning systems of the type discussed also yield superior performance in systems with slowly varying parameters which change suddenly but are relatively constant between changes. The learning system would be designed so that it would store in memory a quantized measure of the plant parameters which are varying or of the environment causing the variation, the best index of performance obtained by the adaptive system, and the corresponding corrections necessary to obtain this performance.

It is important to realize that the hill-climbing technique used in the adaptive portion of the system may not converge to the minimum of the IP surface if the measurements of the IP are noisy. Thus all environmental parameters affecting the IP must be measured. Only if a separate hill-climb is performed for each combination of environmental parameters and all environmental parameters are considered will the IP surface be free of noise and the adaptive portion of the system operate properly.

Thus when a previously occurring set of plant parameters is again encountered, the best learned corrections would be immediately set from memory and adaptation carried out from that point. The system would, of course, be designed so that the contents of the memory are continuously updated by storing the most recent best settings of the corrections.

Assuming then, that the varying parameters may be quickly identified, and that the memory interrogation is faster than the adaptive action, the system performance, after sufficient operat-
In the general learning situation the plant and plant parameters are shown in Fig. 1. In Fig. 1, $e = (e_1, e_2, \ldots, e_n)$ is the environment vector which is considered to be measurable but uncontrolled. It includes all plant inputs and environmental factors which cannot be changed in order to improve the IP. $d = (d_1, d_2, \ldots, d_p)$ is the plant parameter adjustment vector and includes plant inputs which may be changed in order to improve the IP. $c = \text{plant output vector}$, and $I$ is the plant index of performance.

Now each component $e_i$ of the plant environment vector $e$, corresponds to a separate environmental or input variable such as temperature, pressure, etc.

For measurement purposes, each of these variables, $e_i$, is assumed to be quantized into $q_i$ levels where the $q_i$'s for different values of $i$ ($i = 1, \ldots, n$) are not necessarily equal. Then the number $N$ of possible different environment vectors $e$ is

$$N = \prod_{i=1}^{n} q_i$$

$e$ will thus be relabeled $e = e^r$; $r = 1, 2, \ldots$, $N$ when referring to the $r$th possible environment.

The index of performance, $I$, is considered to be a scalar function of $e^r, d$ and $c$, i.e.,

$$I = f(e^r, d, c)$$

where $c = F(e^r, d)$. Thus, for a given $e^r$, I has at most a single minimum as a function of $d$. I is then a p-dimensional hyper-surface in the adjustment parameters $d_j$ ($j = 1, 2, \ldots, p$).

There are several methods of hill-climbing the $d$ vector to the minimum of the IP surface, but the one chosen here is to hill-climb each component of $d$ separately and sequentially, the best values of each of the $p$ parameters being stored in memory corresponding to each $e^r$, along with the corresponding direction of increase or decrease in each parameter of $d$ and the lowest value of $I$. The stored adjustment vector corresponding to $e^r$ is labeled $d^M$. (Superscript $M$ indicates a stored or memorized parameter.)

The operation of the system is then as follows. Upon recognition of the occurrence of $e^r$, $d$ is set to $d^M$ from a search of memory, and hill-climbing proceeds. The computer program should be arranged to facilitate a fast search of memory for previously occurring $e^r$. This may be implemented by programming the computer to search the most frequently occurring $e^r$'s first. The program must also be capable of setting up new storage locations for $e^r$'s which have not previously occurred.

Now let us consider that a particular $e^r$ occurs, given that it has previously occurred $k$ times. Immediately $d^M(k)$ is set from memory and parameter $d_j$ is to be adjusted, where:

$$j = k - a \quad \text{p}$$

$$p \geq j > 0$$

$$\frac{p}{a} \leq k \leq 2p$$

$$a = \begin{cases} 1 & \text{p} + 1 \leq k \leq 2p \\ \text{etc.} & \end{cases}$$

Then $d_j(k + 1)$ is adjusted to

$$d_j(k + 1) = d_j^M(k) + s_j^M(k) \Delta d_j$$

where

$$\Delta d_j = \text{increment of } d_j$$

$$s_j^M(k) = \pm 1 \text{ (stored direction in which to increment } d_j^M(k))$$

Let us now consider the logic required to program the hill-climbing operation and the modification of computer memory. Define

$$m_j(k + 1) = \begin{cases} 1 \text{ for } (I^*(k + 1) - I^M(k)) \leq 0 \\ 0 \text{ for } (I^*(k + 1) - I^M(k)) > 0 \end{cases}$$

where

$$I^*(k + 1) = I(e^r, d^*(k + 1))$$

$$I^M(k) = I(e^r, d^M(k))$$

From the collection of the Computer History Museum (www.computerhistory.org)
The memorized compensation parameter then becomes
\[ d_i^{RM}(k+1) = d_i^{RM}(k) + m_i^r(k+1) s_i^{RM}(k) \Delta d_i \]
The memorized compensation vector is given by
\[ d^{RM}(k+1) = d^{RM}(k) - d_i^{RM}(k) + d_i^{RM}(k+1) \]
where
\[ d_i^{RM}(k) = (0, 0, \ldots, 0_{j-1}, d_i^{RM}(k), 0_{j+1}, \ldots, 0_p) \]
\[ d_i^{RM}(k+1) = (0, 0, \ldots, 0_{j-1}, d_i^{RM}(k+1), 0_{j+1}, \ldots, 0_p) \]
The memorized value of the index of performance becomes
\[ I^{RM}(k+1) = I^{RM}(k) + m_i^r(k+1) \]
\[ (I^r(k+1) - I^{RM}(k)) \]
and the stored direction in which to increment \( d_i^{RM}(k+1) \) is given by
\[ s_i^{RM}(k+1) = - \text{sgn} \left[ (I^r(k+1) - I^{RM}(k)) (d_i^r(k+1) - d_i^{RM}(k)) \right] \]
In the above development, all environmental parameters affecting the system are assumed to be measurable and measurement noise is neglected. All environmental parameters \( e^r \) are assumed to be constant during one hill-climbing step. In the experimental example the same assumptions were made except that \( e^r \) was assumed constant for four hill-climbing steps.

**Experimental System**

A learning system of the type discussed above with second order plant has been simulated on the IBM 1710-GEDA hybrid computer installation. The system was assumed to be subjected to a pseudo-random sequence of environmental conditions which change the damping coefficient and undamped natural frequency of the system at discrete intervals. The environment was changed sufficiently often that purely adaptive action (i.e., hill-climbing on the IP surface) could not optimize the system during the period of constant environment.

![Figure 2. Experimental Learning System.](From the collection of the Computer History Museum (www.computerhistory.org))
The system was subjected to a fixed amplitude square wave input in order to facilitate the computation of an index of performance of the quadratic type. In the plant, as shown in Fig. 2, $e_1$ and $e_2$ are environmental parameters which are constrained to remain constant over two periods of the square wave input. Thus, at each occurrence of an $(e_1, e_2)$ pair four computations of the IP are carried out, one at each transition of the square wave. After each computation of the IP, the compensation parameters $d_1$ and $d_2$ are adjusted by a two-dimensional hill-climbing technique described above in order to minimize the performance index. Memory is used to store the current best values of $d_1$ and $d_2$ for a given set of $(e_1, e_2)$ values, as well as the directions in which $d_1$ and $d_2$ were being adjusted, and the best value of the IP found previously. Thus when a previously occurring $(e_1, e_2)$ pair recurs, the best values of $d_1$ and $d_2$ are set from memory and the best direction to increment $d_1$ and $d_2$ is known. Adaptation then proceeds and the latest values of the best $d_1$ and $d_2$, direction and IP replace the old values in memory.

For the particular system under investigation, $e_1$ and $e_2$ were each allowed to take on five different values, i.e., $q_1 = q_2 = 5$. Thus twenty-five different $(e_1, e_2)$ combinations were possible. In order to simulate the limitations on computer memory capacity, only the sixteen most probable $(e_1, e_2)$ combinations were allowed to have corresponding $d_1$, $d_2$, etc. data stored with them. In this system no compensation or hill-climbing took place on the nine least probable $(e_1, e_2)$ combinations. Probabilities were computed by counting occurrences of $(e_1, e_2)$ pairs and a continuous check was made for the 16 most probable. In the hill-climbing technique, the increments used for changing $d_1$ and $d_2$ were of fixed size and were not reduced even after learning was essentially complete.

The changes in $e_1$ and $e_2$ were generated by storing in memory a pseudo-random sequence of $(e_1, e_2)$ pairs two hundred long. The $(e_1, e_2)$ pairs were set from memory on the digital pots just prior to the square wave transition and the sequence of $(e_1, e_2)$ pairs repeated after each two hundred values (fifty different $(e_1, e_2)$ pairs since each pair was repeated at least four times in a row).

The performance index (IP) used was of the form

$$ IP = T \int_0^T (\gamma_1 e^2 + \gamma_2 x_2^2 + \gamma_3 u^2) \, dt $$

where $e$, $x_2$, $u$ are as defined in Fig. 2, and $\gamma_1$, $\gamma_2$, $\gamma_3$ may be changed as desired. In this example, the parameter $e_1$ was allowed to take on the values 4, 5, 6, 7, or 8 and $e_2$ to have the values 20, 30, 40, 50 or 60.

An outline of the digital computer program and flow diagram for the system under consideration are given in Appendix A.

**Discussion of Results**

An investigation of the IP surface, for the particular system and performance indices under investigation, was carried out. The system may be redrawn as shown in Fig. 3 if we let $a = e_1 + d_1$ and $b = e_2 + d_2$. Let the input $R(s)$ be a step input of amplitude $A$. Then

$$ R(s) = \frac{A}{s} $$
$$ e(s) = \frac{A(s + a)}{s^2 + as + b} $$
$$ x_2(s) = \frac{A}{s^2 + as + b} $$
$$ u(s) = \frac{As}{s^2 + as + b} $$

For an index of performance given by

$$ I = \int_0^\infty (\gamma_1 e^2 + \gamma_2 x_2^2 + \gamma_3 u^2) \, dt $$

Figure 3. Simplified System Block Diagram.
Parseval's theorem yields
\[ I = \frac{1}{2\pi} \int_{-\infty}^{\infty} \{\gamma_1 [x(s) x(-s)] + \gamma_2 [x(s) x(-s)] + \gamma_3 [u(s) u(-s)]\} ds \]

After some manipulation, the integrand becomes
\[ A^2(\gamma_1 + \gamma_3) \left[ \frac{s + \sqrt{\gamma_2 + a^2 \gamma_1}}{\gamma_1 + \gamma_3} \right] \frac{s + \sqrt{\gamma_2 + a^2 \gamma_1}}{\gamma_1 + \gamma_3} \]

The IP may be evaluated from standard tables to yield
\[ I = \frac{A^2[b(\gamma_1 + \gamma_3) + \gamma_2 + a^2 \gamma_1]}{2ab} \]

Then for given \( \gamma_1, \gamma_2, \) and \( \gamma_3, \) contours of IP may be plotted in the \( (a, b) \) plane.

For the performance index used in this example
\[ \gamma_1 = 0, \quad \gamma_2 = 1, \quad \gamma_3 = 1 \]

Thus
\[ I = \frac{A^2[b + 1]}{2ab} = \frac{k(b + 1)}{ab} \]

The constant IP contours for this performance index are shown in Fig. 4 for \( k = 1.0. \) We note that for this case, the index of performance is very insensitive to \( e_2 + d_1 = b \) but very sensitive to \( e_1 + d_1 = a. \) This theoretical conclusion was verified experimentally as may be seen from the trajectories plotted in Fig. 4 for several \( (e_1, e_2) \) values. The system trajectories required approximately two and one-half hours operation to reach the points indicated. The hill-climbing was not completed at this stage but the time was sufficient to illustrate that the system was learning. In this

![Figure 4. Index of Performance for \( I = k(b + 1) \frac{1}{ab} \)](from the collection of the Computer History Museum (www.computerhistory.org))
length of time the IP decreased by a factor of approximately six. The improvement in IP for each different \((e_1, e_2)\) pair may also be seen from Fig. 5, which was plotted from computer type-out data as discussed in Appendix A. Note the IP scale in Fig. 5 was not normalized as are the contours in Fig. 4. It is noted that this particular performance index becomes zero as \(a \to \infty\), \(b \to \infty\), and does not have a particular minimum. Thus after sufficient operating time, the \(d_1\) and \(d_2\) adjustments would simply reach the maximum values allowed for in the system design (100 and 1000 respectively). It appears that the IP is zero for \(b = -1\) but this value...
of b invalidates the derivation for the IP since Parseval's theorem is valid only for asymptotically stable systems.

Fig. 5 is a plot of IP versus \((e_1, e_2)\) pairs for the index of performance investigated. Four points are plotted for each \((e_1, e_2)\) pair corresponding to four different times in the learning process. Level A represents the IP measured before compensation. The particular sequence of \((e_1, e_2)\) pairs given in this test was such that the first twenty-five pairs seen by the system were all different (from left to right as plotted in Fig. 5). After all twenty-five pairs occurred once, the order of occurrence was no longer the same. Level B represents the IP for each pair after the first four adaptive steps for each \((e_1, e_2)\). The first sixteen \((e_1, e_2)\) pairs seen by the system were compensated initially and information was stored. The last nine, in order of first appearance, had no information initially stored, and thus showed no improvement.

Level C represents the IP after approximately one hour of running. Here it is seen that the system had determined the sixteen \((e_1, e_2)\) pairs occurring most frequently (most probable) and had continuously learned on them. The nine least probable pairs had not improved from their initial values.

Level D represents the IP at the conclusion of the test for each pair. Here again, of the sixteen most probable, the ones with highest probability learned faster. Of course adaptation would normally be carried out on all \((e_1, e_2)\) pairs and not just on the 16 most probable.

The learning system then operates as an adaptive system when a given \((e_1, e_2)\) pair or environment first occurs, but has the capability of utilizing past information about the best compensation parameter setting when a previously occurring environment reoccurs. In this case, given sufficient operating time, the learning system is capable of reducing the index of performance to the minimum possible value for the 16 most probable \((e_1, e_2)\) pairs (environmental conditions) even though the environment is changing so rapidly that an ordinary adaptive system would fail to improve the index of performance significantly.

Concluding Remarks

Several particular problems currently under investigation are the application of stochastic approximation techniques\(^{13,14,15}\) to hill-climbing when the IP surface is disturbed by noise due to unmeasurable environmental parameters and the application of pattern recognition techniques to measurements of the IP surface in order to increase the rate of learning. Present studies are also concerned with methods of quantizing the environment parameters to yield efficient learning operation subject to the constraint of finite computer memory capacity.

The principle of learning outlined in this paper, that is, partial hill-climbing of the IP surface at each occurrence of a particular environment and memorization of the best compensation found for that environment, seems to be a relatively simple idea; and yet, as illustrated by the example, it yields great improvement in the system performance over that for a simple adaptive system. It is felt that this concept of learning control is particularly applicable to process control.

Acknowledgement

The authors wish to thank Dr. J. E. Gibson for his stimulating discussions. This work is in part supported by the Research Contract AF AFOSR 62-351 and National Science Foundation Grant G-14609.

APPENDIX

Description of the Computer Program

The flow diagrams shown in Figs. 6, 7, and 8 illustrate the system logic and timing. A more detailed examination follows:

1. The \((e_1, e_2)\) values were punched on cards and stored in memory at the beginning of the program. The program did not take advantage of this fact in determining the probability of each \((e_1, e_2)\) pair.

2. The type-out of results was controlled manually by the computer operator by means of a Program Switch. If the switch was on and a cycle of 200 \((e_1, e_2)\) values had been completed, the type-out was performed, except at the beginning of the program where type-out was performed after the first 100 \((e_1, e_2)\) values.
Figure 6. Flow Diagram (Part I).

Figure 7. Flow Diagram (Part II).
3. When $10^5$ $(e_1, e_2)$ values had occurred, the total number of recorded occurrences of each $(e_1, e_2)$ pair was divided by ten and a counter was reset to count the next $10^5$ values. Then each new occurrence of a particular $(e_1, e_2)$ pair caused the number of recorded occurrences for that $(e_1, e_2)$ to be increased by one. Thus the more recent occurrences were weighted more heavily.

4. When a new pair, $(e_1, e_2)$, was selected, it was compared to the previous pair, $(e_1, e_2)_{i-1}$. If they were the same, the search procedure for $(e_1, e_2)$ and its information was bypassed, since the location of information for $(e_1, e_2)$ was the same as for $(e_1, e_2)_{i-1}$. As shown in 8 below, an indicator was placed in memory for each $(e_1, e_2)$ pair showing which $d_j$ was operated on last. Then the program would operate on the other $d_j$ when that $(e_1, e_2)$ pair occurred again. In order for the program to operate on the same $d_j$ when an $(e_1, e_2)$ pair occurred twice or more in a row, the indicator was changed when $(e_1, e_2)$ was determined to be the same as $(e_1, e_2)_{i-1}$.

5. If $(e_1, e_2) \neq (e_1, e_2)_{i-1}$, then memory was searched to see if $e_1$ had ever occurred previously. Next $e_2$ was checked for previous occurrence. Then $(e_1, e_2)_{i}$ was assigned an address where information was stored concerning its probability and location of index of performance and compensation information. The probability was incremented next (actually the number of recorded occurrences for $(e_1, e_2)_{i}$ was increased by one).

6. At this point memory was organized so that only the sixteen most frequent $(e_1, e_2)$ pairs had compensation information stored. If $(e_1, e_2)_{i}$ had no information stored, then $d_1$ and $d_2$ were set to zero.

7. The following information was stored for the sixteen most probable $(e_1, e_2)$ pairs:
   a) whether $d_1$ or $d_2$ was operated on most recently
   b) the value of $d_j$ ($j = 1, 2$) that resulted in the best (lowest) IP value
   c) the slope (direction of change) for $d_j$ ($j = 1, 2$)
   d) the sign of $d_j$ ($j = 1, 2$)
   e) the best previous value of IP.

8. If information was stored for $(e_1, e_2)_{i}$, the program next determined from an indicator which $d_j$ was operated on last. Assume this was $d_1$. The stored value of $d_1$ was then set, and operations commenced on $d_2$. The indicator was then set to show that the program acted on $d_2$ last (see 4 above). Next, the slope was determined.
   a) If the slope was positive, the program added the standard increment, $\Delta d$, to $d_2$ and the sum was compared with 9999, since this was the maximum setting on the digital potentiometer. If the sum was greater than 9999, then $d_2$ was set to 9999. If not $d_2$ was set equal to the sum in question.
   b) If the slope was negative, and $d_2$ was greater than $\Delta d$, then $\Delta d$ was subtracted from $d_2$. If $d_2$ was less than $\Delta d$, the sign of $d_2$ was changed and the magnitude of $d_2$ was not changed. The sign of $d_2$ was made negative or positive, respectively, by switching an amplifier in or out of the feedback loop (see Fig. 2). If the sign was negative the amplifier was in, thus yielding a subtractive correction.
c) Note that the previous values of $d_1$ and $d_2$ were left in memory. The new $d_j$ were stored later only if the new IP to be measured was lower than the previously stored value of the IP (see 13 below).

9. The proper values of $e_1$, $e_2$, $d_1$, and $d_2$ were then set on the analog computer and the IP integrator was set for integration.

10. The step input was generated next, using a comparator on the GEDA. A delay followed, during which the IP was computed. At the end of the delay, the IP was measured and the IP integrator was disconnected and reset to zero.

11. If no information was stored for $(e_1, e_2)$, the system had completed the cycle and the next $(e_1, e_2)$ pair was obtained.

12. When information was stored for $(e_1, e_2)$, and if the measured IP was greater than the stored best previous IP, then the slope indicator was changed on $d_j$ so that $d_j$ would be incremented in the opposite direction the next time the same $(e_1, e_2)$ pair occurred. If the slope of $d_j$ was changed two consecutive times, then the present measured value of IP was stored and $d_1$ and $d_2$ were not changed in memory. This procedure was implemented to allow for the possibility that an erroneously low value of IP was stored due to any random noise pulses introduced into the system at some previous time. If the slope of $d_j$ had not changed twice consecutively, only the slope indicator was changed in memory and the system had completed the cycle.

13. If the measured value of IP was lower than the stored value, then the measured value, along with the present $d_1$ and $d_2$ values, were placed in memory and the cycle was complete.

References
