$$B \triangleq \frac{e^{T_2/\tau} - e^{T_1/\tau}}{1 - e^{T_2/\tau}}$$

$$\bar{I}_{dc}R = kj + k\frac{T_1}{T_2} - K_2\phi\omega = \frac{T_s}{K_1\phi}R.$$

The effective current is given by

$$(I_{eff}R)^2 = \frac{1}{T_2}\int_0^{T_2}(I(t)R)^2\,dt = (kj + k - K_2\phi\omega)^2\frac{T_1}{T_2}$$

$$+ (kj - K_2\phi\omega)^2\left(\frac{T_2 - T_1}{T_2}\right) + k^2f(T_1,T_2)$$

$$= \left(k + \frac{T_sR}{K_1\phi} - k\frac{T_1}{T_2}\right)^2 + \left(\frac{T_s}{K_1\phi}R - k\frac{T_1}{T_2}\right)^2\left(\frac{T_2 - T_1}{T_2}\right)$$

$$+ k^2f(T_1,T_2)$$

where

$$f(T_1,T_2) = -\frac{\tau}{2T_2}B^2(e^{-2T_1/\tau} - 1) - \frac{\tau}{2T_2}(B + e^{T_1/\tau})^2$$

$$\cdot(e^{-2T_2/\tau} - e^{-2T_1/\tau}) + \frac{2\tau}{T_2}(B + e^{T_1/\tau})(e^{-T_2/\tau} - e^{-T_1/\tau})$$

and, finally,

$$(\mathrm{FF})^2 \triangleq \sigma \triangleq \frac{I_{eff}^2}{\bar{I}_{dc}^2}$$

$$= \frac{\left(k + \frac{T_sR}{K_1\phi} - k\frac{T_1}{T_2}\right)^2\frac{T_1}{T_2} + \left(\frac{T_sR}{K_1\phi} - k\frac{T_1}{T_2}\right)^2\left(\frac{T_2 - T_1}{T_2}\right) + k^2f(T_1,T_2)}{\left(\frac{T_sR}{K_1\phi}\right)^2}$$

$$= \frac{\left(\frac{T_sR}{K_1\Phi}\right)^2 + k^2\frac{T_1}{T_2}\left(1 - \frac{T_1}{T_2}\right) + k^2f(T_1, T_2)}{\left(\frac{T_sR}{K_1\phi}\right)^2}.$$

Appendix III

Numerical Example

Motor Data

Power:	10 kW.
Rated voltage:	$E_N = 220$ V.
Rated current:	52.2 A.
Rated speed:	$n_N = 2250$ r/min.
Rotor resistance:	$R = 0.274\ \Omega$.
Rotor inductance at rated speed:	$L = 0.01$ H.
Inertia:	$J = 1{,}2$ N·m·s².

Results

T_s	$= 43$ N·m.
$K_1\phi$	$= 0.084$.
$K_2\phi$	$= 0.87$.
ω_N	$= 235$ rad/s.

References

[1] C. E. Robinson, "Redesign of DC motors for applications with thyristor power supplies," *IEEE Trans. Ind. Gen. Appl.*, vol. IGA-4, pp. 508–514, Sept–Oct. 1968.
[2] C. J. Newell, "Matching DC driver and motors," *Electro-Technol.* (New York), pp. 38–41, June 1967.
[3] N. Kaufman, "An application guide for the use of D-C motors on rectified power," *IEEE Trans. Power App. Syst.*, vol. 83, pp. 1006–1009, Oct. 1964.

On-Line Learning Optimal Control Using Successive Approximation Techniques

M. D. LEVINE AND T. VILIS

Abstract—The application of learning theory to on-line optimization of unknown or poorly defined plants is discussed. An on-line optimization procedure is achieved by means of a learning algorithm which alters a trainable controller on the basis of an instantaneous performance criterion or subgoal. The subgoal is related to the overall goal, the integral cost, by means of successive approximations to the Hamilton–Jacobi equation. The resulting piecewise linear controller is implemented by means of an encoder consisting of threshold logic units and a classifier consisting of a set of logic switching functions. The classifier is determined by means of an algorithm developed by Arkadev and Braverman. Features of the learning algorithm are illustrated by minimum-time and minimum-time-fuel problems.

I. Introduction

This paper considers the on-line regulator problem for an unknown continuous plant whose outputs (states) are sampled at discrete-time intervals and which can be described by

$$x(m + 1) = f(x(m),\mu(m)) + z(m + 1) \tag{1}$$

where $u(m)$ is an r-dimensional control vector, $x(m)$ is an n-dimensional state vector, and $z(m)$ is an n-dimensional vector representing a stationary Gaussian process. A computer (Fig. 1) is used to synthesize a state feedback controller $u(m) = u(x(m))$ which will return the plant to some desired terminal manifold, denoted by the set of states M, from any disturbed state $x(m) \not\subset M$ while minimizing a performance criterion

$$V(x(m)) = \sum_{i=m}^{p-1} L(x(i),u(i))\cdot T \tag{2}$$

where $L(x,u)$ is the instantaneous cost and $(p - m)T$ is the time to reach M. The minimal $V(x(m))$ is denoted by $V^*(x(m))$ and the corresponding controller is denoted by $u^*(x(m))$.

A method of successive approximations which computes a converging sequence of controllers $u_0(x),u_1(x),\cdots,u_{k-1}(x),\cdots,u^*(x)$, each of which provides stable control over the entire X space, has been developed for known continuous plants [6], [7]. This method also converges for known discrete plants, as shown in Vilis [8].

To find a controller $u_k(x)$, given a stable controller $u_{k-1}(x)$, we employ the following steps.

Step 1: Evaluate the general performance criterion which corresponds to the controller $u_{k-1}(x(m))$,

$$V_k(x(m)) = \sum_{i=m}^{p-1} L(x(i),u_{k-1}(i))\cdot T. \tag{3}$$

Step 2: Synthesize a new controller $u_k(x(m))$ by minimizing

$$H(x(m + 1),x(m),u_k(m),V_k) = \min_{u\in U}\,[H(x(m + 1),x(m),u(m),V_k)] \tag{4}$$

where

$$H(x(m + 1),x(m),u(m),V_k) = L(x(m),u(m))\cdot T + V_k(x(m + 1)) - V_k(x(m)). \tag{5}$$

Manuscript received June 19, 1972; revised January 16, 1973. Paper recommended by G. Saridis, Chairman of the IEEE S-CS Adaptive and Learning Systems, Pattern Recognition Committee. This work was supported by the National Research Council of Canada under Grant A4156.
M. D. Levine is with the Department of Electrical Engineering, McGill University, Montreal, P.Q., Canada, on leave at the Image Processing Development and Application Group, Jet Propulsion Laboratory, Pasadena, Calif.
T. Vilis was with the Department of Electrical Engineering, McGill University, Montreal, P.Q., Canada. He is now with the Otolaryngology Research Laboratory, Department of Physiology, McGill University, Montreal, P.Q., Canada.

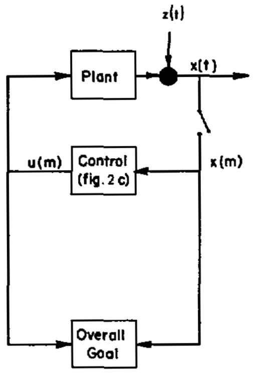

Fig. 1. Goal training phase.

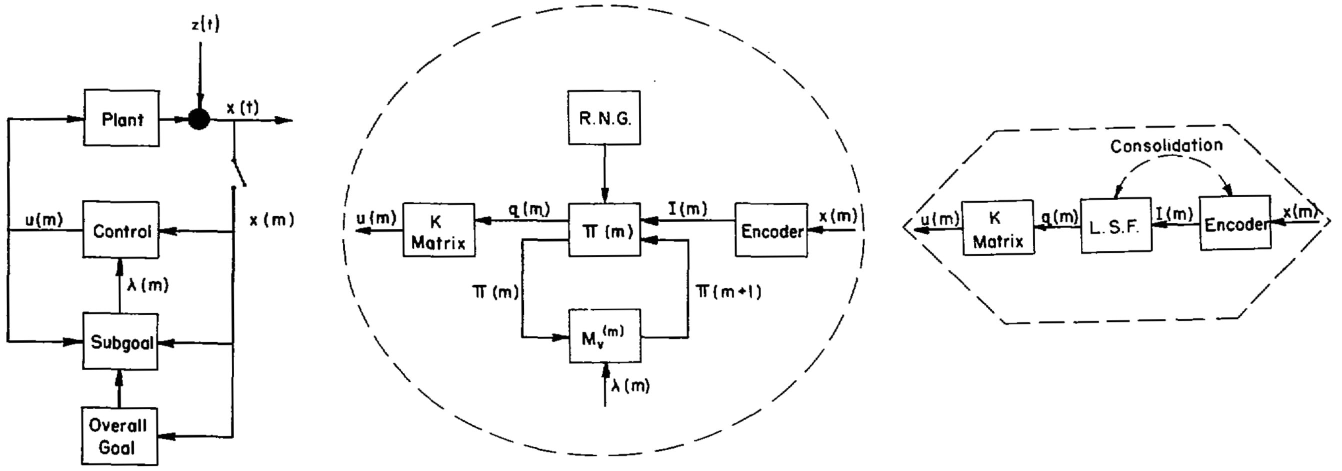

Fig. 2. Controller synthesis. (a) Controller training phase. (b) During the controller training phase, the control in (a) is represented by the stochastic automaton shown above. (RNG is random number generator.) (c) After the training phase, the controller is consolidated into the above configuration. (LSF is logic switching function.)

On repeating the sequence we find that $V_k(x(m)) \geq V_{k+1}(x(m))$ and that the series converges to $V^*(x(m))$ as $k \to \infty$.[1]

We intend in this paper to show how the above steps can be implemented on-line for an unknown plant subject to noise. The learning control algorithm proposed consists of two training phases which correspond to the two steps above: 1) an overall goal training phase in which the general performance $V_k(x)$ is evaluated (Fig. 1), and[2] 2) a controller training phase in which the subgoal $H(x(m+1), x(m), u(m), V_k)$ is used to synthesize $u_k(x)$ (Fig. 2).

As pointed out by Jones [3] and Lambert and Levine [4], it is important that the overall goal and subgoal be related so that optimization of the subgoal leads to eventual optimization of the overall goal. Such a relationship between H and V is proven in [6]–[8].

The controller training phase is shown schematically in Fig. 2. Initially the controller can be considered a stochastic automaton being modified according to the subgoal H [shown in Fig. 2(a), (b)]. Fu and McMurtry [2] have demonstrated the effectiveness of this method in a noisy environment. Following this, the controller is consolidated into a minimized switching function shown in Fig. 2(c). This controller is then used in the goal training phase (Fig. 1).

II. Methods of Function Storage

The learning algorithm must optimize both the subgoal H and the goal V on-line. This involves storing and updating values of the goal, controller, and subgoal as functions of their respective variables.

The goal function is stored as $V(h(m))$ where $h_i(m)$ corresponds to the quantized value of the ith component of the vector $x(m)$ such that $h_i(m)$ can take on the values $\cdots -2\Delta, -\Delta, 0, \Delta, 2\Delta \cdots$.

Since the controller $u(x)$ may be a multidimensional function of a multidimensional vector, its storage requires a different approach. Suppose a number of planes are drawn where the intercepts are uniformally random in some region of X space. For a point x on one side of the plane $g_i(x) > 0$, while for x on the opposite side $g_i(x) < 0$. Let this relationship be encoded by a binary variable I_i where I_i can take on the set of values $B = [0, 1]$. Then, if $g_i(x) \geq 0$, let $I_i = 1$ and, if $g_i(x) < 0$, let $I_i = 0$.

In general, for k planes, the Cartesian product $B^k = B \times B \times \cdots B$ is formed. Since $B = [0, 1]$, each combination of k variables defines a vertex on a k-dimensional real unit cube which can be expressed functionally by a standard product $f(I)$. In Fig. 3, for example, region 7 in X space is represented by the standard product $f(I_1,I_2,I_3) = \bar{I}_1 \cap I_2 \cap \bar{I}_3$. Encoding the X space in this manner allows the direct synthesis of a piecewise linear feedback controller in such a way that it can be stored in a digital computer as a series of binary terms, or easily implemented by logic hardware.

The function $u(I(m))$ is restricted to a finite set of quantized values. These consist of the set $U = [u^1, u^2, \cdots, u^s]$ where s is the number of different control choices and each element u^i is an r-dimensional vector in the control space.

Finally, the subgoal function H is stored in terms of the quantized elements u^i, $i = 1, \cdots, s$, and I^j, $j = 1, \cdots, d$. Thus, one value of the subgoal will be stored for each element of the Cartesian product $u^i \times I^j$, $i = 1, \cdots, s$, $j = 1, \cdots, d$.

III. Controller Training Phase

In Fig. 3 each vertex of the hypercube must be assigned to some element u^i of the control set to define the function $u(I(m))$. This is

[1] Note that, in (4), $V_k(x(m))$ is independent of $u(m)$ and thus can bere written as $\min_{u \in U} [L(x(m),u(m)) \cdot T + V_k(x(m+1))]$, the equation used in Bellman's dynamic programming. The difference here is that k refers to iterations in the strategy space and is unrelated to m, which specifies temporal progression along any plant trajectory. Thus one cannot simply equate this minimum to $V_{k+1}(x(m))$, but instead must repeat Step 1.

[2] If no stable u_{k-1} is available to evaluate V_k, one can train such a controller by starting in phase 2) with some arbitrary stable subgoal such as $H = |x(m+1)|$.

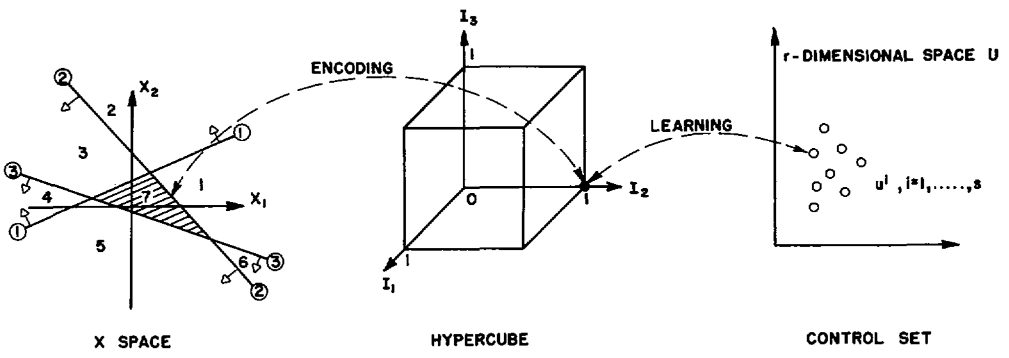

Fig. 3. Quantization of the control function $u(x)$.

accomplished by the on-line reinforcement training of a stochastic automaton, a method used successfully for controller training [5], [7], [9]. State encoding as described here provides a simpler method of transition from stochastic control to deterministic control.

First, let us summarize the method of training the stochastic automaton [7]. The latter [Fig. 2 (b)] is restricted to outputs q^i such that $q_j^i = 1$ or 0 and $\sum_{j=1}^{s} q_j^i = 1$. With this restriction we can decode q^i into the proper member of U by simply multiplying the former by a gain matrix K. The gain matrix consists of s columns with each column composed of the vector $u^i = [u_1^i,\cdots,u_r^i]^T$, $i = 1,\cdots,s$. Let the probability of q^i for input I^v be Π_i^v, a component of the s-dimensional probability vector Π^v. The relationship between probability vectors of two successive time instances is given by the transition equation

$$\Pi^v(m+1) = \alpha\Pi^v(m) + (1-\alpha)\lambda^v(m), \qquad 0 < \alpha < 1 \qquad (6)$$

where α controls the rate of convergence and $\lambda^v(m)$ is the reinforcement vector with components $\lambda_i = 1$ or 0 and $\sum_{i=1}^{s} \lambda_i = 1$. These are chosen on the basis of the subgoal $H^v(u^i) = H[x(m+1),x(m), u^i(I^v(m)),V(h(m))]$, which in turn is evaluated from repeated trajectories in the same way as the subgoals of [5] and [9]. If, for a particular encoded state I^v, the stored $H^v(u^i)$ is such that

$$H^v(u^i) = \min_{j=1,s} [H^v(u^j)], \qquad (7)$$

then the reinforcement vector λ^v has components $\lambda_i = 1$ and $\lambda_j = 0$, $j \neq i$ and $j = 1,\cdots,s$.

Training is terminated when the stochastic automaton has reached a certain prescribed level of confidence. A deterministic relationship between q^i and I^v is then defined by a set of logic functions F_j, $j = 1,\cdots s$, where

$$q_j^i = F_j(I) = \bigcup_{k=1}^{dj} f_{jk}(I) \qquad (8)$$

such that, if, for the vertex I^v, represented by the standard product $f(I)$, $\Pi_j^v = \max_{i=1,\cdots,s} [\Pi_i^v]$, then $f(I)$ is included in the union of standard products defined as $F_j(I)$. This particular $f(I)$ is not included in any other union, thus assuring that, for vertex I^v, $F_m(I^v) = 0$ for $m \neq j$ and $m = 1,\cdots,s$, and that the gain matrix K is still compatible. The controller is represented by the three elements shown in Fig. 2(c). The next step is consolidation of the encoder.

IV. Controller Consolidation

Consolidation is necessary for two reasons: 1) the method of Section III does not assure that q^i will be defined for every I^v, and 2) (8) is not a minimal form, which implies unnecessary quantization of the X space. These inadequacies can be corrected by the dissecting planes algorithm first proposed by Arkadev and Braveman [1], whose steps are formalized here in terms of minimizing switching functions. The objective is to find a minimal representation of (8), while maintaining the mutual exclusive property of the elements of q^i. This is achieved by first assigning DON'T CARE conditions to all vertices for which no control is defined and then performing the following steps.

Step 1: To check whether a redundant plane g_i can be eliminated, let $I_i = 1$ and test whether

$$F_v(I_i = 1)\cdot\left[\sum_{\substack{j=1,s\\ j\neq v}} F_j(I_i = 1)\right] = 0, \qquad \forall\, I^v, v = 1,d. \qquad (9)$$

If so, then the I_ith variable is removed (new $k \leftarrow k - 1$); if not, I_i is reinstated. All planes ($i = 1,k$) are tested consecutively in this way.

Step 2: Step 1 can result in redundant representation of vertices. To eliminate these from F_v, test each of the terms consecutively as follows. If

$$f_{vi}\cdot f_{vj} = f_{vj}, \qquad \text{for any } i = 1,d_v \text{ and } j = i+1,d_v, \qquad (10)$$

then f_{vj} is redundant and can be eliminated (new $d_v \leftarrow d_v - 1$). Each F_v function is tested in this way.

Step 3: To remove the redundant segments of plane g_i in the output function F_v, let $I_i = 1$ *in the vertex* f_{vn}. If

$$f_{vn}(I_i = 1)\cdot\left[\sum_{\substack{j=1,s\\ j\neq v}} \sum_{r=1,d_j} f_{jr}(I_1,\cdots,I_k)\right] = 0, \qquad \forall\, I^v, v = 1,d, \qquad (11)$$

then the term f_{vn} becomes the reduced term $f_{vn}(I_i = 1)$; otherwise, the former is resinstated. All the vertices are tested for the dimension I_i in this way. After this, the process is repeated for the next dimension, and so on, until all k dimensions have been tested.

Step 4: As in Step 1, the merging of small regions into larger ones has produced redundant vertices, and thus Step 2 must be repeated.

The result of removing redundant planes and redundant segments of planes is that all unassigned regions I^v are assigned controls. Also, the algorithm unites into one region any two adjacent regions assigned the same control. This gives a set of switching functions in a minimal sum-of-product form which, together with the encoder and decoder, constitutes the consolidated controller.

V. Goal Training

Using the controller obtained above, the algorithm returns to retrain the goal function V. The stable controller generates trajectories from random disturbed states to M. During each trajectory a series of instantaneous costs $L[x(m), u(I(m))]\cdot T$ are stored in the computer together with their associated states $x(m)$. Once M is reached, these costs are summed backwards to evaluate the summed costs $V(x(m))$ for each point along the trajectory. The stored values of state are then quantized, and $V(x(m))$ is stored in its respective quantized region as $V(h(m))$. Several trajectories are calculated in this way. If parts of these trajectories overlap, values of $V(h)$ are averaged with those having a corresponding h. If at the end of a given number of trajectories some regions are undefined, the computer generates values of $V(h)$ for these regions based on adjacent regions.

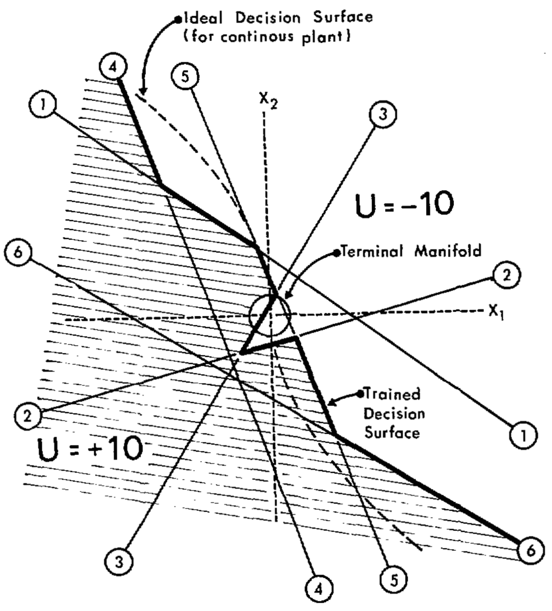

Fig. 4. Solution to minimum-time problem.

VI. Examples

The behavior of the learning algorithm was studied by simulation. For this purpose the particular unknown plant was chosen to be the following:

$$\begin{bmatrix} x_1(m+1) \\ x_2(m+1) \end{bmatrix} = \begin{bmatrix} 0 & T \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1(m) \\ x_2(m) \end{bmatrix} + \begin{bmatrix} 0 \\ T \end{bmatrix} u(m) + \begin{bmatrix} z_1(m+1) \\ z_2(m+1) \end{bmatrix} \quad (12)$$

where $T = 0.1$ s and z represents a Gaussian process of zero mean and a standard deviation of 0.05. The terminal manifold was defined as $|x_i| < 1.0$ and the disturbance was given a uniform probability density of $|x_i| < 10$. Two overall goals were tested.

Goal 1: The overall goal for the minimum-time problem was to minimize

$$V(x(m)) = \sum_{i=m}^{p-1} T = (p-m)T \quad (13)$$

where the control is limited to $|u| < 10$. The minimum principle then enables one to choose the initial control set as $U = [+10, -10]$. The function V was stored as a 41×41 grid in the interval $-20 \leq x_i \leq 20$ with $\Delta = 1.0$. For $u(x)$, the X space was quantized by ten random planes. A stable controller $u_0(x)$ was trained using an initial subgoal $H = |x(m+1)| - |x(m)|$. Each controller training phase was completed in less than 200 trajectories. The resulting switching surfaces after the sixth iteration are shown in Fig. 4.

Goal 2: The overall goal for the minimum-time-fuel problem was

$$V(x(m)) = \sum_{i=m}^{p-1} (1 + |u|)T. \quad (14)$$

Except for changing $U = [+10, 0, -10]$, function storage was as above. The algorithm was initialized by using $u_6(x)$ from the above example as the initial controller. The resulting decision surfaces after the third iteration are shown in Fig. 5, and the controller which would produce them is shown in Fig. 6. An interesting property of this controller should be noted. The encoder, by mapping adjacent regions of X space into adjacent vertices, maintains the spatial relationship of elements in X space in the quantized space. Single-bit errors in the I code would only change the representation to some adjacent region. Also, since the control code q^i is linearly independent, single-bit errors can easily be detected. Thus the codes I and q may be transmitted along a noisy channel enabling remote on-line control.

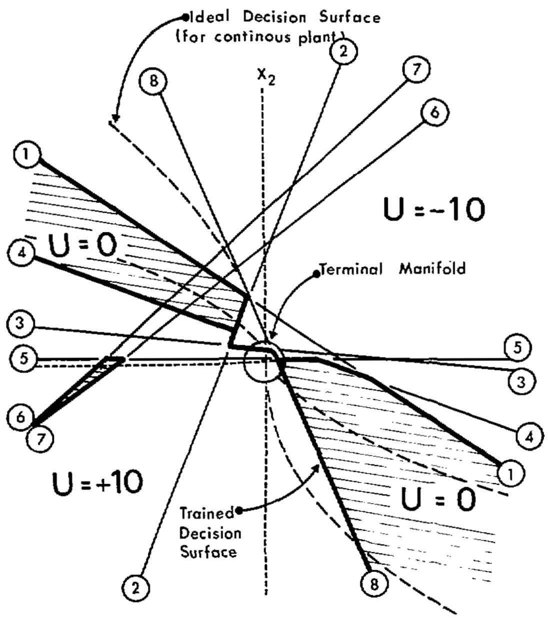

Fig. 5. Solution to minimum-time-fuel problem.

VII. Conclusions

This paper has studied the use of the method of successive approximations in generating a subgoal suitable for on-line learning control. Although simulations were limited to linear plants, the subgoal is theoretically applicable to nonlinear unknown plants. The usefulness of dissecting planes as a state encoder was also demonstrated.

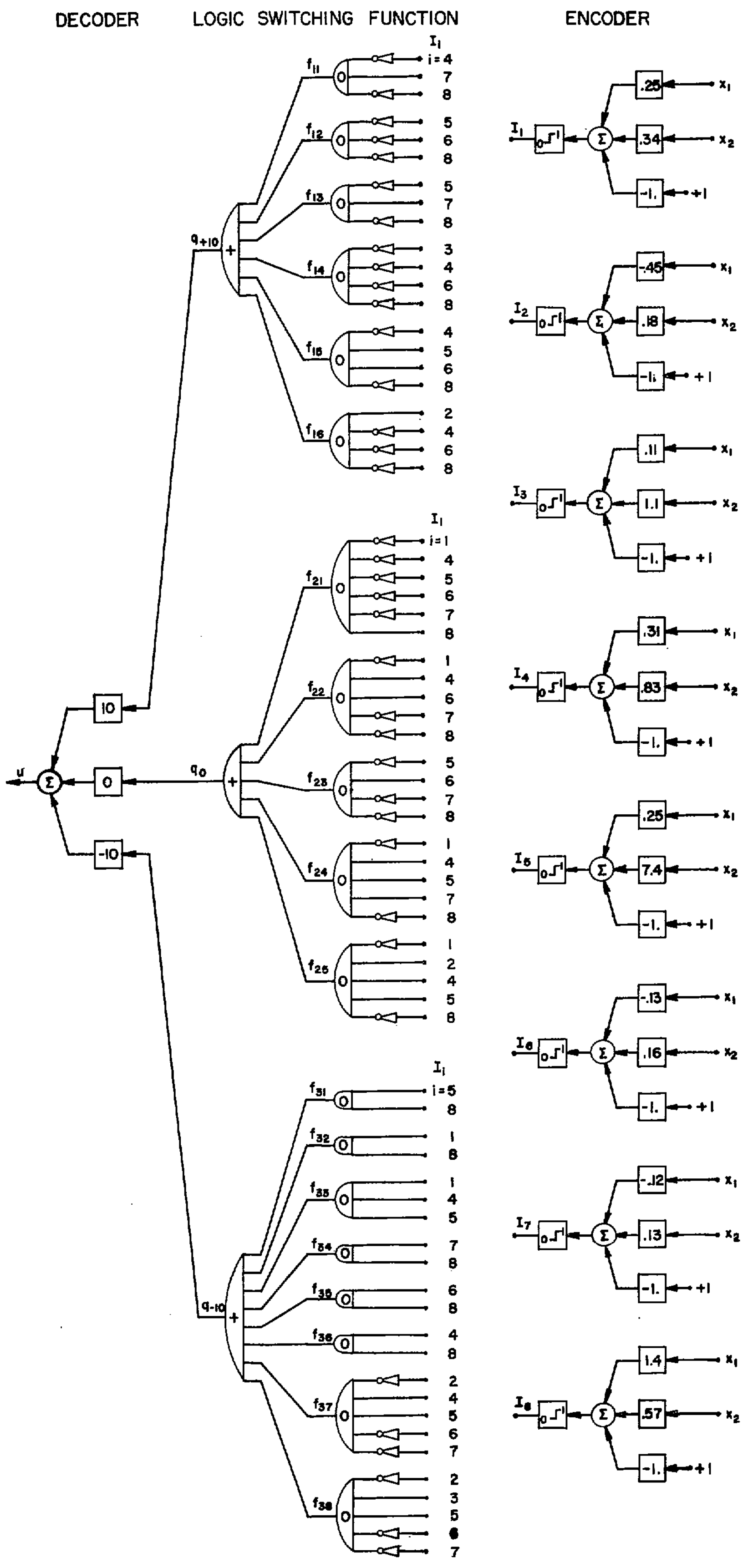

Fig. 6. Feedback controller for minimum-time-fuel problem.

References

[1] A. G. Arkadev and E. M. Baverman, *Computers and Pattern Recognition* (transl. from Russian). Washington, D.C.: Thompson, 1967.
[2] K. S. Fu and G. J. McMurtry, "A study of stochastic automata as models of adaptive and learning controllers," Purdue Univ., Lafayette, Ind., Rep. TR-EE65-8, June 1965.
[3] L. E. Jones. III, "On the choice of subgoals for learning control systems," *IEEE Trans. Automat. Contr.*, vol. AC-13, pp. 613–621, Dec. 1968.
[4] J. D. Lambert and M. D. Levine, "Learning control heuristics," *IEEE Trans. Automat. Contr.* (Corresp.), vol. AC-13, pp. 741–742, Dec. 1968.
[5] ——, "A two-stage learning control system," *IEEE Trans. Automat. Contr.* (Short Papers), vol. AC-15, pp. 351–354, June 1970.
[6] G. N. Mil'shtein, "Successive approximations for one optimal problem," *Automat. Remote Contr.* (U.S.S.R.), vol. 25, no. 3, p. 298, 1964.
[7] E. M. Vaisbord, "An approximate method for the synthesis of optimal control," *Automat. Remote Contr.* (U.S.S.R.), vol. 24, p. 1482, 1963.
[8] T. Vilis, "On line learning using successive approximations techniques," M.S. thesis, McGill Univ., Montreal, P.Q., Canada, 1971.
[9] M. D. Waltz and K. S. Fu, "A heuristic approach to reinforcement learning control systems," *IEEE Trans. Automat. Contr.*, vol. AC-10, pp. 390–398, Oct. 1965.

Stability Criterion for N-Dimensional Digital Filters

J. H. JUSTICE AND J. L. SHANKS

Abstract—**The stability requirement for one-dimensional recursive filters is well known. A stability theorem for n-dimensional recursive filters is proved wherein the denominator of the filter is an n-dimensional power series. A Tauberian theorem due to Wiener yields the desired result in the general case.**

I. Introduction

Linear digital filtering is a useful tool for processing discrete sequences of data [1],[2]. It is used in a variety of applications, including processing of seismic data, radar signals, cardiographic recordings, and many other "signals" which have been sampled and stored in digital form.

One of the more efficient types of digital filters is the "recursive filter" [3],[4]. For one-dimensional sequences, the recursive filter can be described by its z-transform

$$F(z) = \sum_{i=0}^{M} a_i z^i \Big/ \sum_{j=0}^{N} b_j z^j \tag{1}$$

where the a and b coefficients define the filter. In applying this filter to a data sequence, we use the recursive algorithm

$$y_n = \frac{1}{b_0}\left\{\sum_{i=0}^{M} a_i x_{n-i} - \sum_{j=1}^{N} b_j y_{n-j}\right\} \tag{2}$$

where the x_k, $k = 0,1,2,\cdots$, represent the input data sequence and the y_k, $k = 0,1,2,\cdots$, represent the output sequence. In using this algorithm, we assume that the x_k and y_k are zero for all $k < 0$.

This type of filter is used extensively in processing one-dimensional sampled data. It is also possible to extend this technique to n-dimensional data [5], [6], [10]. Such filters are useful in processing two-dimensional data, such as seismic data sections, digitized photographic data, and gravity and magnetic maps. In the case of a two-dimensional recursive filter, the filter can be described using two-dimensional polynomials or power series in (z_1,z_2), such as

$$F(z_1,z_2) = A(z_1,z_2)/B(z_1,z_2) \tag{3}$$

where

$$A(z_1,z_2) = \sum_{i=0}^{M_1}\sum_{j=0}^{M_2} a_{ij} z_1^i z_2^j \tag{4}$$

Manuscript received October 27, 1972; revised January 29, 1973. Paper recommended by J. C. Willems, Past Chairman of the IEEE S-CS Stability, Nonlinear, and Distributed Systems Committee.

J. H. Justice is with the Department of Mathematics, University of Tulsa, Tulsa, Okla. 74104.
J. L. Shanks is with the Research Center, Amoco Production Company, Tulsa, Okla. 74102.

and

$$B(z_1,z_2) = \sum_{k=0}^{N_1}\sum_{l=0}^{N_2} b_{kl} z_1^k z_2^l. \tag{5}$$

A filtering algorithm similar to (2) can be written for the two-dimensional and higher dimensional filters.

One of the problems in using recursive filters is stability. We require that the output of the filter not become unbounded if the input is bounded. The stability depends on the coefficients of the denominator of the recursive filters. In the case of the one-dimensional recursive filter, it has been shown in many places that the filter will be stable if the roots of the denominator polynomial $B(z)$ are all outside the z-plane unit circle [7]. However, these proofs all depend on our ability to factor the polynomial $B(z)$ into its distinct roots. In the case of n-dimensional polynomials or power series, no such factorization exists. Huang [11] has shown a proof for the two-dimensional case in which the $B(z_1,z_2)$ is a finite polynomial. Therefore, it is the purpose of this paper to state and prove the conditions on the denominator polynomial or power series of an n-dimensional recursive filter which will allow that filter to be stable.

II. Development

Let us begin by developing the rationale for the precise definition of stability which we shall use. It is well known that multiplication of two power series may be performed by convolving their sequences of coefficients; this is the process inherent in recursive digital filtering. We shall not distinguish between a power series and its sequence of coefficients but shall refer to a power series as a sequence, or vice versa, when convenient. The term stability of a filter is generally used to indicate that the result of convolving the filter with some bounded input sequence should have, in some sense, a bounded output. Since all of this is rather vague, let us be more precise. One of the simplest classes of power series which we might choose to work with is the class of power series in z and $1/z$ which has absolutely summable coefficients. That is, those series of the form

$$\sum_{n=-\infty}^{\infty} a_n z^n$$

where

$$\sum_{n=-\infty}^{\infty} |a_n| < \infty.$$

This class offers the advantage that a product (convolution) of two members of the class is again of this class. As a result, if a filter and an input sequence are chosen from this class, the output sequence must also be of this class, and so the filter is necessarily of the type we choose to think of as stable. Since the recursive filter is in general a quotient of two power series, we shall require that the two series be of this class and seek the conditions which will guarantee that the resulting quotient will again be represented by a power series in this class. Our procedure will be to use a Tauberian theorem proved by Wiener [9, p. 37] to derive the necessary criterion. Because this result does not rely on dimension, but only on the algebraic and topological structure of the class of absolutely summable sequences, we are able to derive the stability criterion for the large class of N-dimensional recursive filters. Our ultimate aim is to give the necessary and sufficient condition that, given an N-dimensional absolutely summable power series in the denominator of the filter, the filter will be stable no matter what N-dimensional absolutely summable numerator may be chosen for the recursive filter.

To simplify our work in N-dimensions, let us use the following notation.

Notation: We shall represent the integers by Z, the set of nonnegative integers by P, and the set of nonpositive integers by N.

The sequences (coefficients of power series) which we use must be indexed. We shall consider index sets which belong to the set $Z^\alpha \times P^\beta \times N^\gamma$ where α, β, γ are nonnegative integers. A zero exponent on