ALGORITHM DESIGN FOR REAL-TIME DISTRIBUTED SYSTEMS

By

RAM ASHOK VISWANATH

Bachelor of Technology

Indian Institute of Technology

Madras, Tamilnadu

India

1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1992

ALGORITHM DESIGN FOR REAL-TIME

DISTRIBUTED SYSTEMS

Thesis Approved:

[illegible]
Thesis Advisor

[illegible]

[illegible]

Thomas C. Collins
Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my gratitude to my advisor, Dr. Gary Young for his guidance and advice throughout the course of this study. Many thanks also go to Dr. Eduardo Misawa for his helpful guidelines about literature research. To him and Dr. Hoberock, I extend my sincere appreciation for serving on my graduate committee. Dr. Martin Hagan of the Department of Electrical Engineering deserves special mention for the very helpful discussions we had during the latter part of this study.

I am indebted to the Department of Mechanical Engineering for having supported me over the major part of my graduate program. All my teachers whose stimulating lectures have helped me gain the necessary foundation for this work are also acknowledged.

My friends and relatives deserve heartfelt thanks for their constant encouragement which helped me in the study by no little measure.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Algorithms designed for implementation in an off-line environment do not always maintain their performance level when implemented in a real-time distributed environment.

Definitions

The Oxford Dictionary of Computing gives the following definition of a real-time system:

Any system in which the time at which the output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.

Young, S. J. [16] defines a real-time system to be:

any information processing activity or system which has to respond to externally-generated stimuli within a finite and specified period.

These systems are further divided into two classes (Burns and Wellings, [13]). Hard real-time systems are those where it is absolutely imperative that responses occur within the specified deadline. Soft real-time systems are those where response times are important but the system will still function correctly if deadlines are occasionally missed.

Also, in [13], a distributed computer system is defined to be :

a system of multiple autonomous processing elements, cooperating in a common purpose or to achieve a common goal.

Motivation

The significance of real-time systems in everyday life is increasingly evident. The vast range of application areas includes, in order of increasing complexity, controllers in washing machines, air traffic control and air defense systems, control and safety systems of nuclear power plants, and future military systems like the Strategic Defense Initiative (SDI). Also, many new and challenging applications are being developed in the areas of command and control systems, process control systems, automated manufacturing, flight control systems, avionics, space shuttle avionics, the space station, shipboard systems, submarine systems, visions and robotics, and teams of robots working in hazardous environments. Further, as [15] points out, real-time systems are an integral part of computer-integrated manufacturing systems (CIM), whose efficient utilization has a big influence on the competitiveness and prosperity of entire nations. All this calls for considerable effort in research and development of highly reliable real-time systems.

Too, recent advances in hardware and communications technology have made distributed systems a viable alternative to uniprocessor and centralized systems in many real-time application areas. As motivation for the use of distributed systems, [13] lists the following potential advantages of distribution:

1) improved performance through the exploitation of parallelism;

2) increased availability and reliability through the exploitation of redundancy;

3) dispersion of computing power to the locations where it is used; and

4) the facility for incremental growth through the addition or enhancement of processors and communications links.

Scope of Present Study

Consider that a solution method (algorithm) was sought for a particular class of problems. Further assume that a study of the nature of the problem (either by analytical or heuristic means) revealed an algorithm that provides results which are optimal in some specified sense. Then, implementation of the algorithm in an off-line environment may be reasonably expected to maintain optimality of the solution. In other words, a certain performance level of the algorithm can be guaranteed when implemented off-line.

However, certain characteristics of real-time distributed systems (described in the next chapter) impede the smooth execution of algorithms implemented on-line in a networked environment. This results in a reduced performance level. Not all related problems have been exhaustively addressed in the existing literature. This study attempts an alternative approach toward problems caused by a particular nature of real-time multiprocessing systems --- hard-time constraints, i.e., those constraints that absolutely have to be met for successful system operation.

The plausibility of modifying the available algorithms to account for the real-time nature of the problem is researched. The effect of such changes is analyzed statistically by Monte Carlo simulations of sample systems in order to justify the modifications. Also, mathematical analysis is done for a simplified case of such adaptation.

CHAPTER II

BACKGROUND

Off-line Implementation of Algorithms

Consider the basic assumption underlying the structure of a traditional computer program used to implement an algorithm off-line. The flowchart mechanism behind the conversion of an algorithm to a program reveals that execution of the code is assumed to proceed logically from statement to statement, beginning with the first and without discontinuities or interruptions.

It is the invalidity of this premise, engendered by the interrupt-driven nature of input/output (I/O) control mechanisms widely used in real-time distributed systems, that leads to performance degradation on-line. Discussion of this, as well as some of the other characteristics of real-time distributed systems, associated problems and some available solutions in the literature is contained in the following section.

Characteristics of Real-Time Systems

Deterministic Response

One of the requirements placed on a real-time system is that it be capable of providing deterministic response to inputs. In a broader sense, it should be able to perform particular tasks at certain prespecified instants of time. Because of the dynamic nature of the external processes, the value of information degrades with time. Consequently, the correctness of a real-time system depends not only on the logical result of the computation, but also on the time at which the results are produced. Absence of such predictability may result in a disaster. It is worthwhile to note that speed is not always the issue.

At this stage, it might occur that by tuning the computer program suitably to provide deterministic response, say through the use of simple, static priority schemes, one would satisfy the real-time constraint. However, it can be easily shown that, in general, fixed priority assignments are incapable of meeting activity deadlines, even when the computing capacity is far greater than the needs of the activities [14].

Asynchronous Input

This refers to the occurrence of input signals at non-sampling instants. Interrupts from other processors and / or unscheduled higher priority tasks due to certain critical events in other parts of the network fall under this category. An example of an interrupt can be found in air flight control systems. If an aeroplane has experienced depressurization there is an immediate need to give over all computing resources to handling the emergency. It is essential that a processor be able to handle such interrupts in a responsible manner while also executing its original task(s) within the allotted time. Failure to do so might lead to instability of other dependent tasks.

Notice that frequent occurrence of interrupts as also lengthy interrupt service routines take up a considerable amount of processor time. This leaves the processor with little time for completion of the original task, say, an optimal control algorithm. In the context of the Kalman filter (see Chapter VI for a detailed description), this problem can be explained using Figures 1 - 3. Fig. 1 shows the split-up of the time allocation for the various steps in the algorithm. Notice that the time interval, denoted (4), that is left unutilised by the algorithm, is the amount of time that may be spent on interrupt-handling without violating the hard-time constraint. Fig. 2 shows the reduction in unutilised time due to the occurence of interrupts A and B. Finally, Fig. 3 shows the violation of the hard-time constraint, as depicted by the extra time required, but unavailable, in order to complete the computations of the Kalman filter algorithm.

Interprocessor Communication

The potential increase in reliability brought about by distribution was pointed out in Chapter I. Then, it is paradoxical that, at the same time, distribution possibly introduces more failures in the system. The concept of a partial system failure is a case in point: it is possible for a single processor to fail while others continue to operate. In addition, the propagation delay through the underlying communication networks is unpredictable and messages may take various routes. This, in conjunction with an unreliable transmission medium, may result in messages being lost, corrupted, or delivered in an order different from the order in which they were sent. Alternative strategies to be implemented in case of lost or corrupted information, in order to sustain performance of the algorithm are extensively available in existing literature [2] - [7]. These are briefly explained in the next chapter.

However, there seems to be a lack of attention to the issue of interrupts in a hard-real time system. In this study, we address the need to design algorithms that are capable of accommodating the inevitable interrupts while also serving to maintain reasonable performance levels.

CHAPTER III

LITERATURE SURVEY

As indicated in the previous chapter, three salient features of a real-time distributed environment are deterministic response, asynchronous inputs and interprocessor communication. Such real-time problems that are caused by non-deterministic response and/or improper interprocessor communication have been the subject of research for a considerable time. In the context of a particular control problem, namely state estimation, this nature of problems has been variously addressed as "uncertain observation", "interrupted observation", "missing data", "loss of signals" etc. In applications related to target tracking, such mishaps lead to a non-zero false-alarm probability associated with the detection decision. In any case, a processor affected by such discrepancies will be unable to perform its task optimally.

In 1968, Middleton and Esposito [1], among the earliest researchers in this field, considered the problem of estimation (under uncertainty) of constant unknown signal parameters in a single observation interval, in the broader context of simultaneous detection and estimation.

Nahi [2] derived minimum mean-square estimators for two different forms of the uncertain observation problem --- one where a probability of false-alarm is associated with each individual observation, and the other where a probability of false-alarm is associated with an entire set of signals. Both the estimators he derived are linear and recursive. Also, necessary and sufficient conditions for the optimality of the estimators are derived, assuming that the probability of false-alarm is known *a priori*. Effectively, these estimators discount an estimated percentage of observations that lead to false-alarm. It may be noticed that the linearity constraint on the estimators prevent optimality in the Bayes sense. This is because a crucial factor in optimal estimation --- the conditional probability of false-alarm given the previous output history --- is non-linear with respect to observation. This information cannot be utilised if the estimators are constrained to be linear. Hence, Nahi's estimators are not Bayes optimal.

Srinath and Rajasekaran (1971) note in [3] that the observations may not contain the signal because of either of the following reasons:

1) the signal was transmitted, but due to some cause such as a cut in the communication link, the observation contained only noise, i.e., irrelevant data.

2) the signal was not transmitted, i.e., the signal was identically equal to zero.

The estimators in [1] relate more meaningfully to the latter case.

Whereas [3] applies the general likelihood ratio test for the minimum mean-square estimation of the signal when the presence of the signal at the receiver is uncertain during the entire observation interval, recursive Bayes optimal estimators are derived in [4] by Jaffer and Gupta (1971) for a discrete stochastic process when there is uncertainty regarding the presence of the signal at each stage of the observation sequence. It is noted that the optimal recursive Bayes estimators are non-linear in the observations and require an ever-growing amount of memory for their computation. The latter fact necessitates a recourse to sub-optimum procedures. However, the same authors derive in [5], a recursive optimal Bayes solution that does not require a growing amount of memory and computation for its implementation but that, however, requires recursion on continuous functions to be performed. Thus, actual implementation of the digital techniques would necessitate the quantization of the spaces of the continuously distributed random vectors denoting the process. The paper admits, however, that quantization is normally not feasible for multidimensional processes since the number of quantized points, and consequently the storage requirements, increase exponentially with the dimension of the process.

Comparison between the simulation results from its optimal estimators and from Nahi's linear estimators is shown to indicate the superior performance of the non-linear estimators.

Sawaragi, Katayama and Fujishige (1972), [6], consider the problem of estimation with interrupted observation and also derive Bayes optimal estimates, with a somewhat different formulation of results.

Apart from [3], the likelihood ratio test has been used as a tool also by Van Trees [7] for solving the detection problem in the face of uncertain observation.

All the above work use complicated estimation theory in deriving (sub)optimal estimates. In contrast, for the case of missing measurement data, Chen (1990), [9] suggests using the optimal predictor (which is readily available by simple open-loop simulation of the process) to replace the unknown optimal filtering estimator which can be calculated only if the measurement were available.The optimality of such an estimator, under the mean-square criterion, is proved by Chen (1987) in [8]. The crucial issue of analyzing the convergence of the proposed suboptimal filtering algorithm is also addressed in [8], but the main drawback of the analysis is the assumption that only a single bit of data (i.e., data at a single sampling period) is missing.

As is evident, the nature of problems addressed in the literature discussed above relates to non-deterministic responses and / or improper interprocessor communication. Attention to the important issue of hard-time constraint that cripples the computations of an algorithm, seems to be limited. However, on recognizing the reduction in the availability of computational resources to be a direct consequence of interrupts in a hard-real time system, the papers which deal with such modification of algorithms that leads to computational savings may be considered relevant to the present study. In the context of the Kalman filter several such papers are available in current literature. Singer and Sea's (1971) short paper, [17] presents an iterative processing technique that, when the additive noise vector of the system observation process can be partitioned into uncorrelated subvectors, can save upto 50 percent in computer time from the standard computational procedure. The technique essentially replaces the covariance update equation (equation 8 in Chapter VI) with several equations of the same form, say 8b, each involving corresponding matrices lower in dimension than the original ones. The computational savings result because the time required to compute a large matrix inverse (or multiplication) is greater than that to compute several small inverses (or multiplications). The iterative method of the paper has the additional advantage that the algorithm can be interrupted by a higher priority function between the iterations of 8b with no loss of information. If the conventional method were used, an interrupt would possibly cause a loss in data, thereby preventing the updating of the covariance equation at that point.

In lieu of modifying the covariance equation, changes can be made in the gain-update equation. Friedland (1967), [10] simplifies the mechanization of a Kalman filter by approximating the optimal gain. To evaluate the effect of the erroneous gain, he derives an expression for the increase in the covariance matrix of the estimate due to the difference between the optimal gain and the nonoptimum gain actually employed. It must be noted that he deals with a continuous-time system, rather than disrete. When the gain matrix is approximated the innovations process gets colored. Scharf (1973), [11] examines, for discrete-time systems, the state model of the pseudoinnovations process resulting from an arbitrary gain modification, and shows that the level (or variance) of the process depends linearly on the difference the gain matrix being employed and the optimum Kalman gain matrix.

A specific change in the Kalman filter algorithm would be to replace the optimal gain by its corresponding steady-state limit. The resultant is the Wiener filter. Singer and Frost (1969), [18], aiming at sub-optimal filter design, derive upper and lower bounds on the error covariance matrices of the Kalman and the Wiener filters for linear time-invariant systems. The analytical results obtained for the general scalar problem indicate that there are relatively few instances when the Kalman filter provides a significant improvement over the performance of the corresponding Wiener filter.

These papers indicate that the computational burden in a hard-real time system is a legitimate reason for studying the effects of approximations in an algorithm. However, papers could not be found that, at once, study the relative performance of a number of specific sub-optimal schemes, and are also directed at satisfying the hard-time constraints of a real-time system. To fill this void is the need for the present work.

CHAPTER IV

PROBLEM STATEMENT

With the background provided by the previous sections, the problem addressed in this study is now stated.

How can the performance level of an algorithm be maintained at an acceptable level in the face of interrupts that violate the hard-time constraint, using only the available computing resources?

The performance level of an algorithm relates to the optimality of the results that it produces. When an algorithm is designed for a class of control problems, minimization of a certain cost-function is usually the criterion. If a theoretical lower bound can be identified it serves as an ideal measure of the performance level of the implemented algorithm. Lower the cost-function during implementation (i.e., closer to the lower bound), better the performance level is said to be.

By "available computing resources" is implied that it is not sought to solve the hard-time constraint by recourse to enhanced data-processing ability, such as due to a faster CPU. Such augmentation of resources is assumed to be unwarranted because interrupts violating the hard-time constraint may, in some cases, be expected to occur only over short periods of time.

The approach taken in the study is to compromise on the optimality of an algorithm (in the sense described above) while maintaining its stability. In other words, a non-minimal value of the cost-function provided by a sub-optimal algorithm is tolerated if the results of interest do not deviate largely from those of the optimal algorithm. Since the choice of the cost-function is problem-specific, typical examples of control problems are presented in the following chapters.

CHAPTER V

PARAMETER IDENTIFICATION

Review of Optimal Least-Squares Algorithm

Consider the discrete-time system

$$\begin{tikzcd} v(k) \arrow[r] & \boxed{\begin{array}{c}\text{Linear}\\ \text{System}\end{array}} \arrow[r] & y(k) \end{tikzcd}$$

It is described by the equation

$y(k) = a_1y(k-1) + a_2y(k-2) + ... + a_py(k-p) + v(k)$

i.e., $y(k) = \Phi(k)\Theta + v(k)$

where, $\Phi(k) = [y(k-1) \quad y(k-2) \quad ... \quad y(k-p)]$, the regressor vector;

$\Theta = [a_1 \quad a_2 \quad ... \quad a_p]^T$, the vector of unknown parameters;

$v(k)$ is the system input;

$y(k)$ is the output;

and k is the time-index of the discrete-time system.

The autoregressive system is said to be of order p. $y(k)$ is of size $n \times 1$, Φ is $n \times p$, and Θ is $p \times 1$.

We want to estimate Θ, using measurements of the output, $y(k)$. Let $\hat{\Theta}$ be the estimate of Θ.

Defining the error, $\varepsilon(k) = y(k) - \hat{y}(k)$, where $\hat{y}(k) = \Phi(k)\,\hat{\Theta}(k)$, we then make the choice of the cost-function that is to to be minimized. As is typical, we consider the magnitude of the error.

Minimize $J(\hat{\Theta}) = \varepsilon^T \varepsilon$.

Setting

$$\frac{\partial J(\hat{\Theta})}{\partial \hat{\Theta}} = 0$$

and proceeding with the derivation, we arrive at a set of equations (not shown here) that make up the least-squares identification algorithm. Converting them into a recursive form to enable on-line implementation, we have

$$\hat{\Theta}(k+1) = \hat{\Theta}(k) + K(k)\,\varepsilon(k), \qquad \text{----- 1}$$

where $$K(k) = P(k)\Phi^T(k)[I + \Phi(k)P(k)\Phi^T(k)]^{-1} \qquad \text{----- 2}$$

and $$P(k+1) = [I - K(k)\Phi(k)]P(k) \qquad \text{----- 3}$$

Equations 1, 2 & 3 constitute the optimal recursive least-squares algorithm (refer equations 4-24 to 4-26, [12]).

We see that equation 1 is in the form of an update, with the second term on the right side providing the correction. The optimality of this correction is obtained from equation 2, which calculates the gain matrix, K(k), each time-step. The covariance of the estimation error is given by P(k), which is computed every time-step by equation 3.

Sub-Optimal Identification

In the above algorithm, we recognize that equation 2 is the most crucial one, for it is that gain matrix which provides optimality of the estimates. However, also note that it has the maximum cost in computation time, as shown below by simple arithmetic:

equation 1 requires 2np number of operations;

equation 2 requires $n^3 + np(2n+p)$ operations; and

equation 3 requires $p^2[n + (p+1)/2]$ operations.

Now, from a stability point of view, it is essential that equation 1 always be computed since typically, some other task is dependent on a value for the parameter estimate. Then, it is clear that in the event of external interrupts during a time-step, most time can be saved for the interrupt service routines by substituting equation 2 with a sub-optimal gain equation.

In seeking to simplify the gain-equation for time-steps with interrupts, we have the following options:

(i) we could set the value of the gain matrix to zero. This amounts to not updating the parameter estimate in equation 1, i.e.,

$$K(k) = 0;$$

and $\hat{\Theta}(k+1) = \hat{\Theta}(k)$.

In other words, the parameter estimate found in the last step is used for the current step also. Let us call this the "zero gain when interrupt" scheme.

(ii) we may utilize the previous value of the gain matrix for the current step also. That is, we now update the estimate of the parameter using the last value of the gain matrix, i.e., $K(k) = K(k-1)$. For this scheme, we shall use the term "last gain when interrupt".

Notice that both these schemes would help keep the algorithm going, even with little cost in computation time. However, they involve an insurance cost in that equation 1 will have to be computed with a sub-optimal gain at the beginning of every time period. This is because of the unpredictable nature of the interrupts, which does not guarantee safe completion of the computations at any time-step. Once a sub-optimal estimate is available, we may proceed to calculate, using equations 2 and 3, the correction needed to make it optimal. That way, if computation of the correction is interrupted a sub-optimal estimate would still be available. In other words, we first calculate the hard-constraint portion given by $\hat{\Theta}(k+1) = \hat{\Theta}(k) + K_{\text{sub-opt}}(k)\,\varepsilon(k)$

and then update this estimate using

$$\hat{\Theta}(k+1) = \hat{\Theta}(k+1) + [K(k) - K_{\text{sub-opt}}(k)]\varepsilon(k),$$

which is only a soft-constraint.

The cost of this insurance results from adding and subtracting the sub-optimal correction term, $K_{\text{sub-opt}}(k)\,\varepsilon(k)$, which requires np number of operations for its computation.

It remains to be verified whether the results provided by such adapted algorithms are acceptable. If so, we seek to make a comparison between them and to choose a particular scheme as worthy of recommendation. We attempt to do this first by mathematical analysis. Then, bearing in mind the random nature of interrupts, we shall employ the technique of Monte Carlo simulation to verify our analysis.

Mathematical Validation of Sub-Optimal Identification

In this section, the aim is to establish the convergence of the sub-optimal algorithms with respect to optimal one, for the case of a single hard-time interrupt occurring at an arbitrary time-step.

Consider the least-squares algorithm for parameter identification, equations 1 - 3. At an arbitrary time-step, k_0, optimally

$$\hat{\Theta}(k_0) = \hat{\Theta}(k_0-1) + K(k_0-1)[y(k_0-1) - \Phi(k_0-1)\hat{\Theta}(k_0-1)].$$

With an interrupt, the zero-gain scheme gives

$$\hat{\Theta}_{zg}(k_0) = \hat{\Theta}(k_0-1).$$

Then the estimation error with reference to the optimal algorithm is

$$\varepsilon_{zg}(k_0) = K(k_0-1)[y(k_0-1) - \Phi(k_0-1)\hat{\Theta}(k_0-1).$$

At the next step,

$$\hat{\Theta}(k_0+1) = \hat{\Theta}(k_0) + K(k_0)[y(k_0) - \Phi(k_0)\hat{\Theta}(k_0)].$$

Assuming no interrupt at this time-step, we get from the zero-gain scheme

$$\hat{\Theta}_{zg}(k_0+1) = \hat{\Theta}_{zg}(k_0) + K(k_0)[y(k_0) - \Phi(k_0)\hat{\Theta}_{zg}(k_0)].$$

Now, the error is

$$\varepsilon_{zg}(k_0+1) = [I - K(k_0)\Phi(k_0)]\varepsilon_{zg}(k_0).$$

In general then, at time-step (k_0+l),

$$\varepsilon_{zg}(k_0+l) = [I - K(k_0+l-1)\Phi(k_0+l-1)]...[I - K(k_0)\Phi(k_0)]\varepsilon_{zg}(k_0)$$

or, $$\varepsilon_{zg}(k_0+l) = [I - K(k_0+l-1)\Phi(k_0+l-1)]\varepsilon_{zg}(k_0+l-1). \qquad \text{----- 4}$$

The aim is to prove that the above error tends to zero as $l \rightarrow \infty$. We notice that the above equation for error propagation is of the same structure as for the error covariance, equation 3. Reference [20] indicates that, for an observable system with bounded measurement noise level, the error covariance of the least-squares algorithm converges. From this knowledge of the convergence of error covariance, we conclude that the error in equation 4 also vanishes.

Next, considering the last-gain scheme, the error is

$\varepsilon_{lg}(k_0) = [K(k_0-1) - K(k_0-2)]\ [y(k_0-1) - \Phi(k_0-1)\hat{\Theta}(k_0-1).$

Then the error propagation equation is the same as for the other scheme, i.e., equation 4. Only the initial value of the error is different. Convergence to the optimal algorithm is again evident.

The relative speed of convergence of the two sub-optimal schemes is clearly a function only of the initial error, i.e., during the interrupt time-step, due to gain adaptation. Therefore, closer the sub-optimal gain to the optimal value, smaller the initial error and hence quicker the convergence.

Simulation and Results

Simulation of Interrupts

In order to simulate the sub-optimal algorithms and study their performance in a hard-time constrained environment, it is necessary to simulate the occurrence of random interrupts. This was done in the following way:

A random-number generator was initially set to generate numbers uniformly distributed in the interval [0, 1]. At each time-step of simulation, a random number was generated. If its value lay in a specific quarter, say [0, 0.25], of the possible range then an interrupt was assumed to have occurred.

Notice that this specific choice results in an overall interrupt rate of 25%, i.e., about a fourth of the time-steps have hard-time constraints. Now, since the interrupts are made random in order to duplicate the real-world scenario as closely as possible, the time-steps at which they occur are not known a priori. Different simulation runs would yield different sequences of interrupts. Clearly an interrupt occurring during the initial (transient) steps of the identification process is more critical than the later ones as it contributes to a larger error in the sub-optimal schemes. In order to study the effect due to an "average" interrupt sequence, the technique of Monte Carlo simulation is used.

Sample Systems for Simulation

A number of systems were used to study the performance of the sub-optimal schemes relative to the optimal scheme. Two samples are shown here and the corresponding results presented in the next section. An autoregressive model of second order was chosen for the study, i.e., two parameters were to be identified. An interrupt rate of 25% was used for all simulations. This rate was considered high enough to correspond to a real-world system, since typical real-time systems software would not be expected to be designed with a hard-time constraint that might be violated much more frequently.

Sample System 1. Denoted system (1) hereafter, its parameters were chosen as 1 and -0.1, i.e.,

$$\Theta = \begin{bmatrix} 1 \\ -0.1 \end{bmatrix}$$

The noise level for v(k) was set at 0.01.

The initial conditions were set as y(1) = 1 and y(2) = 1.

Sample System 2. System (2) was chosen as having

$$\Theta = \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix}$$

The other conditions were the same as for system (1).

Simulation Results

The results of the Monte Carlo simulation with 1000 runs are shown in the Figures section of the Appendix. Figure 4 displays the running values of the estimates of parameter (1) in system (1), due to the optimal as well as sub-optimal schemes. We see that, as expected, the optimal scheme gives the fastest and closest convergence to the true parameter. Of the two sub-optimal schemes the zero-gain-when-interrupt scheme comes closer to the optimal one. However, since the statistical mean (expectation) is not a sufficient measure of a random variable (process), the mean square estimation error is considered. Figure 5 shows that the zero-gain scheme again has a smaller error than the last-gain scheme.

In order to gain a better understanding of the reason for these trends, we look at the fundamental difference, namely the gain matrix. Figure 6 shows the running statistical mean values of the first element in the gain matrix. It can be seen that the sequence of gains is distorted less by the zero-gain scheme than by the last-gain scheme.

The corresponding results for parameter (1) in system (2) are shown in Figures 7, 8 and 9. The same trends as discussed above are again evident. Though not shown here parameter (2), for both systems, revealed similar trends.

These trends can be interpreted as follows: when an interrupt occurs at an arbitrary time-step during the identification process the optimal gain is, on an average, closer to zero than to the previous value. This leads to better estimates from the zero-gain scheme.

CHAPTER VI

STATE ESTIMATION

Review of the Kalman Filter

Consider the discrete-time system

$$\begin{tikzcd} u(k) \arrow[r] & \boxed{\begin{array}{c}\text{Linear}\\ \text{System}\end{array}} \arrow[r] & y(k) \end{tikzcd}$$

It is described by the set of equations

$$X(k+1) = \Phi X(k) + \Psi u(k) + \Gamma v(k)$$

$$y(k) = C\, X(k) + Du(k) + e(k),$$

where v(k) and e(k) are independent white noise sequences with intensities R_1 and R_2 respectively, k is the time-index, X is of size n x 1, u is l x 1, and y is m x 1.

We want to estimate the state vector X(k), using the measurements of u(k) and y(k).

$\hat{X}(k)$ is the estimate of X(k).

Estimation error, $\tilde{X}(k) = X(k) - \hat{X}(k)$.

We look for a solution of the form

$X(k+1) = X(k+1|k) + K(k+1)[y(k+1) - CX(k+1|k)]$, where

$X(k+1|k) = \Phi X(k) + \Psi u(k)$

Choose K(k) to minimize the variance of the estimation error, P(k).

Then, the following set of equations is obtained (cf. [12], equations 17-11 to 17-14):

$$X(k+1) = X(k+1|k) + K(k+1)[y(k+1) - CX(k+1|k)] \qquad \text{----- 5}$$

where $$K(k+1) = P(k+1|k)C^T[CP(k+1|k)C^T + R_2]^{-1} \qquad \text{----- 6}$$

$$P(k+1|k) = \Phi P(k|k)\Phi^T + \Gamma Q \Gamma^T \qquad \text{----- 7}$$

$$P(k+1|k+1) = [I-K(k+1)C]P(k+1|k) \qquad ----- 8$$

The above equations 5 - 8 constitute the optimal recursive stochastic estimator, commonly known as the Kalman filter.

Equation 5 yields the update of the estimate. It has two parts to it, the first term on the right side corresponding to the open-loop simulation (prediction), and the latter to the correction due to the prediction error. The optimal gain for this correction is provided by equation 6. The covariance of the estimation error is computed in equation 8.

Computational Requirements:- Each of the steps in the Kalman filter algorithm places a computational burden on the processor, as indicated in Table 1 below:

TABLE 1

BREAK-UP OF COMPUTATIONAL BURDEN

Equation #	Number of operations
5	$n^2 + nl + 2mn$
6	$mn[m+2n+(m+1)/2] + m^3$
7	$2n^2(n+1)$
8	$mn(n+1)/2$

Note that the use of canonical forms can simplify the Φ matrix such that the computations for equation 7 are negligible relative to those for equations 6 and 8. Clearly then, equation 6 (gain update) requires the most number of operations for its computation.

Sub-Optimal Estimation

As in the case of parameter identification, we find that also in the algorithm for the Kalman filter, the equation for gain update is the most time-consuming one. The rationale for gain-modification in the event of interrupts therefore holds for this case equally. The first two of the following options for the sub-optimal schemes are then familiar.

(i) zero gain when interrupt,

(ii) last gain when interrupt,

(iii) steady-state gain when interrupt.

This third scheme is motivated by the fact that in the Kalman Filter algorithm

$$P(k+1) = P(k) = P_s$$

and $K(k+1) = K(k) = K_s$, for large k (see [9] for proof):.

This value of the gain is undoubtedly the best choice for the sub-optimal scheme, at least during the steady state. It might be worth studying its performance during interrupts at random points over the entire estimation process.

Again, we shall begin with a mathematical analysis and then use Monte Carlo simulations for verification.

Mathematical Validation of Sub-Optimal Estimation

In this section, we establish that the results from the sub-optimal algorithms converge to those from the optimal Kalman filter algorithm for the case where an arbitrary time-step has an interrupt violating the hard-time constraint.

Refer to the Kalman filter algorithm in equations 5 - 8. At an arbitrary time-step, k_0, the optimal estimate is given as

$$X(k_0) = \Phi X(k_0-1) + \Psi u(k_0-1) + K(k_0)[y(k_0) - CX(k_0|k_0-1)] \qquad \text{----- } 9$$

With an interrupt, the zero-gain scheme gives

$$X(k_0) = \Phi X(k_0-1) + \Psi u(k_0-1) \qquad \text{----- } 10$$

The estimator in equation 10 imitates the one in equation 3 in [9], where it is used as an optimal estimator when measurement data $y(k_0)$ is missing. On recognizing this fact, we see that the proof of convergence of our zero-gain scheme (for a single interrupt) derives from that of equation 3 in [9]. Though that proof will not be reproduced here, we shall state the error propagation equation in order to extend the proof to the last-gain scheme. First, the relative error between the optimal and zero-gain estimates at k_0 is

$$\varepsilon_{zg}(k_0) = K(k_0)[y(k_0) - CX(k_0|k_0-1)]. \qquad \text{----- } 11$$

Assuming no interrupts thereafter, at the time-step k_0+l, we have

$$\varepsilon_{zg}(k_0+l) = [[I - K(k_0+l)C]\Phi ... [I - K(k_0+1)C]\Phi]\varepsilon_{zg}(k_0). \qquad \text{----- } 12$$

The following lemma is used to prove that the covariance of the error tends to zero exponentially.

Lemma. Consider a linear time-invariant system which is both completely controllable and observable. Let $\{K(k)\}$ be the sequence of the Kalman gain matrices, which converges to the limiting Kalman gain matrix K_s. Then, for any sufficiently small $\delta > 0$, there exists an integer $N > 0$ such that all eigenvalues of all matrices $P[I-K(l)C]$ with $l \geq N$ are of magnitude less than or equal to $1-\delta$.

This lemma is used in Theorem 1 of [9] to establish the exponential convergence of the covariance of the error (in equation 12) to zero as l tends to infinity.

We then consider the last-gain scheme. The error between the optimal and the last-gain schemes at k_0 is

$$\varepsilon_{lg}(k_0) = [K(k_0) - K(k_0-1)]\ [y(k_0) - CX(k_0|k_0-1)]. \qquad \text{----- } 13$$

However, the propagation equation for the error is the same as that for the zero-gain scheme, equation 12. Hence, the same proof as referenced above applies here to guarantee convergence of the last-gain estimate to the optimal estimate.

Though the same argument can be extended to the steady-gain scheme, [8] already contains proof of its convergence for a 100% interrupt rate. There, chapter 6, "Limiting Kalman Filter" considers replacing the optimal gain at every time-step by the constant, steady-state gain matrix in order to save computation time. In that chapter, theorem 6.2 shows that the resultant estimate is an asymptotically optimal estimate of the *actual* system states, i.e.,

$$\underset{k\to\infty}{\mathrm{Lt}} \left\| X(k) - \hat{X}_{sg}(k) \right\|_n^2 = \underset{k\to\infty}{\mathrm{Lt}} \left\| X(k) - \hat{X}(k) \right\|_n^2 .$$

Next, theorem 6.3 shows that the error between the optimal and steady-gain schemes also tends to zero exponentially fast, i.e.,

$$\operatorname{tr}\left\| X(k) - \hat{X}_{sg}(k) \right\|_n^2 \leq Cr^k ,$$

where C is a positive constant independent of k, and r is a real number, $0 < r < 1$.

Having discussed the stability of the sub-optimal schemes, we now turn to their relative performance.

Choice of Sub-Optimal Scheme

One of the earlier research papers, [10], derives the following expression for the increase in the estimation covariance due to a difference, Δ, between the optimal gain, $\hat{K}$, and a sub-optimal gain, K, actually employed:

$$dS/dt = (A-KC)S + S(A-KC)^T + \Delta\, R.\Delta^T \qquad \text{----- } 14$$

where A is the transition matrix of the continuous-time system,

C is the observation matrix,

R is the measurement noise matrix, and

$(.)^T$ denotes transposition.

Also, [11] further indicates that, in a discrete-time system, this increase is linearly dependent upon the deviation, Δ. Clearly, a larger deviation in the value of the gain will produce a larger increase in the covariance matrix. We wish to use this fact to identify the sub-optimal scheme that will result in the least increase in the covariance. Evidently, we need to choose that scheme which distorts the gain by the least amount. This is done by the following procedure, developed in this study.

Consider a scalar, discrete-time system, with the same notation as indicated in the beginning of the chapter. Let the optimal gain at the first step of the estimation process be K(1). Also, let K_s denote the steady-state gain. Referring to Fig. 10, we see that the optimal gain at the second time-step would lie in the region AD. Stated mathematically, $K_s < K(2) < K(1)$.

Notice that the value of the last-gain at this time-step is K(1). If K(2) lay closer to K_S than to K(1), i.e., if K(2) lay in the region AC, where C is the mid-point of AD, then the steady-gain would be a closer approximation to the optimal gain than would the last-gain be, i.e., $| K(2) - K_S | < | K(2) - K(1) |$. The choice of the steady-gain would hence lead to less gain-distortion and in turn to a lower increase in estimation covariance.

On the other hand, if K(2) lay in CD, its proximity to K(1) would make the last-gain a better choice than the steady-gain. In that case, we would have $|K(2) - K(1)| < | K(2) - K_S |$. We see that the point C serves as a point of demarcation, separating regions of different choices for the sub-optimal scheme.

Using the same approach as above, this time to compare the last-gain and zero-gain schemes, it is evident that the point B, representing a one-half value of K(1), serves as the demarcation point. If K(2) lay in AB, zero-gain would preferable to last-gain, while the latter is to be chosen if BD contained K(2).

Also, by inspection we see that, since K(2) always lies in AD, the steady-gain is always closer than zero-gain to the optimal gain, i.e., $| K(2) - K_S | < K(2)$. Hence, lower estimation errors would always result from the steady-gain rather than the zero-gain scheme.

Combining the three results derived above, we arrive at the following preferences for the sub-optimal schemes (shown in Table 2 on next page).

Remarks:-

1. For systems with a negative value of gain, the same discussion and results as above apply, on considering the mirror-image of the graph (in Fig. 10) about the x-axis.

2. Even though the above discussion analyses the pattern of the gains occuring only at the second time-step, thereby seeming to ignore the effects due to possibly contradictory patterns over subsequent time-steps, its validity is strengthened by the following facts considered together:

a) owing to the nature of the equations describing the optimal gain as well as the locus of the demarcation points (denoted B and C in the above discussion), the pattern of gains does not change more than once, if at all, over the entire estimation process, and

b) the pattern of gains established at the second time-step is dominant in its effect, not only because of the larger magnitude of the gain during this initial transient period, but also because its effect accumulates over successive time-steps, as indicated by the integral term in equation (14) of [10].

TABLE 2

THEORETICAL PREDICTION OF PERFORMANCE

Range of optimal gain, K(2)	Order of preference of sub-optimal schemes
1. $K(2) < K(1) / 2$	a. steady-gain b. zero-gain c. last-gain
2. $K(1) / 2 < K(2) < (K(1) + K_S) /2$	a.steady-gain b. last-gain c. zero-gain
3. $K(2) > (K(1) + K_S) / 2$	a. last-gain b. steady-gain c. zero-gain

Simulation Results

To test the hypothesis derived above, regarding the choice of the sub-optimal schemes, Monte Carlo simulations were carried out. Results from three systems, each providing a different pattern of gains, are documented here for discussion. The Monte Carlo simulations involved 500 runs, each with an interrupt rate of 25%.

Sample System 3. Referred to as system (3) hereafter (the previous chapter contained the first two systems), it was chosen with the following parameters:

$\Phi = 0.8$;

$\Gamma = C = 1$;

$D = 0$;

The noise levels were set as $R_1 = R_2 = 1$;

$P(0) = 10^4$.

A unit step input was chosen, i.e., $u = 1$.

Sample System 4.

$\Phi = 0.8$;

$\Gamma = C = 1$;

$D = 0$;

$R_1 = 1$; $R_2 = 100$;

$P(0) = 10^8$.

Sample System 5. This system only differed from system (4) in that it had $P(0) = 50$.

Discussion of Results. Calculation of K(1), K(2) and K_s for each of the above three systems using the Kalman filter algorithm enables the following predictions regarding the relative performance of the schemes (shown in Table 3).

Referring to the rms errors in figures 11 - 13 respectively for systems 3 - 5, we see that these predictions are indeed realized. In system 3, the covariance has the least magnitude when the "steady gain when interrupt" scheme, rather than one of the other two, is employed. Also, the last-gain scheme performs better than the zero-gain scheme. Similarly, systems 4 and 5 reveal covariance trends that correlate with the predictions in the above table. This strengthens the belief that the theory developed in this study will apply to other scalar systems also. In other words, *a priori* calculation of the steady-state gain as also the optimal gain at the first two time-steps (all these from the knowledge of the system parameters alone) seems to enable a proper choice of the sub-optimal scheme for state estimation in hard-time systems with interrupts.

TABLE 3

PERFORMANCE OF SUB-OPTIMAL SCHEMES

System #	Order of performance
3	steady-gain last-gain zero-gain
4	steady-gain zero-gain last-gain
5	last-gain steady-gain zero-gain

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

The plausibility of using certain sub-optimal control algorithms for implementation in a real-time distributed environment is verified. From mathematical analysis as well as extensive Monte Carlo simulations of several sample systems, it is concluded that with a small cost in the optimality of results, the hard-time constraint of the real-time problem can be effectively satisfied by the judicious simplification of certain complex computational steps, for instance, through splitting a hard-constraint into hard and soft-constraint portions. Such adaptation is especially feasible when dealing with recursive algorithms (such as the example cases considered) or lattice filters. Based on the results of this study, the following suggestions for the choice of the sub-optimal schemes are made.

The "zero gain when interrupt" scheme may be used for the least-squares parameter identification algorithm.

For the Kalman filter algorithm, the various combinations of possible values for the system parameters influence the choice of the sub-optimal scheme. Therefore, Table 2 in the previous chapter should be used as a guide for making the right choice.

It is to be noted that due to the heuristic nature of the choice of the sub-optimal schemes selected for consideration in this study, it is not possible to recommend any particular scheme as the "best" one. In other words, the existence of other sub-optimal schemes with better performance is not ruled out. Hence, it is recommended that further work be done toward determining the "best" sub-optimal scheme for the various classes of control problems.

A SELECTED BIBLIOGRAPHY

[1] Middleton, D., and Esposito, R. Simultaneous Optimum Detection and Estimation of Signals in Noise, *IEEE Transactions on Information Theory*, vol. IT-14, May 1968, pp 434 - 444.

[2] Nahi, N. E. Optimal Recursive Estimation with Uncertain Observation, *IEEE Transactions on Information Theory*, 1969.

[3] Srinath, M. D., and Rajasekaran, P. K. Estimation of Randomly Occurring Stochastic Signals in Gaussian Noise, *IEEE Transactions on Information Theory* (Correspondence), vol. IT-17, Mar. 1971, p 206.

[4] Jaffer, A. G., and Gupta, S. C. Recursive Bayesian Estimation with Uncertain Observation, *IEEE Transactions on Information Theory* (Correspondence), vol. IT-17, Sept. 1971, pp 614 - 616.

[5] Jaffer, A. G., and Gupta, S. C. Optimal Sequential Estimation of Discrete Processes with Markov Interrupted Observations, *IEEE Transactions on Automatic Control*, vol. AC-16, No. 5, 1971, pp. 471 - 475.

[6] Sawaragi, Y., Katayama, T., and Fujishige, S. Sequential State Estimation with Interrupted Observation, *Information and Control* , 1972.

[7] Van Trees, H. L. Detection, Estimation and Modulation Theory, Part 1, John Wiley, NewYork, 1968.

[8] Chui, C. K., and Chen, G. Kalman Filtering With Real-Time Applications. New York: Springer Verlag, 1987.

[9] Chen, G. A Simple Treatment for Suboptimal Kalman Filtering in Case of Measurement Data Missing, *IEEE Transactions on Aerospace and Electronic Systems*, 1990.

[10] Friedland, B. On the Effect of Incorrect Gain in Kalman Filter, *IEEE Transactions on Automatic Control*, Oct. 1967, p 610.

[11] Scharf, L. L. State Model of a Pseudoinnovations Process, *IEEE Transactions on Automatic Control*, Oct. 1973, pp 547 - 548.

[12] Mendel, J. M. Lessons in Digital Estimation Theory. Prentice Hall, New Jersey, 1987.

[13] Burns, A., and Wellings A. Real-Time Systems and Their Programming Languages. Addison-Wesley, Wokingham, England, 1990.

[14] Northcutt, J. D. Mechanisms for Reliable Distributed Real-Time Operating Systems. Academic Press, Florida, 1987.

[15] Editorial, The Journal of Real-Time Systems 1, Kluwer Academic Publishers, Boston, 1989.

[16] Young, S. J. Real-Time Languages: Design and Development. Chichester: Ellis Horwood, 1982.

[17] Singer, R. A., and Sea, R. G. Increasing the Computational Efficiency of Discrete Filters, *IEEE Transactions on Automatic Control*, June 1971, pp 254 - 257.

[18] Singer, R. A., and Frost, P. A. On the Relative Performance of the Kalman and Wiener Filters, *IEEE Transactions on Automatic Control*, Aug. 1969, pp 390 - 394.

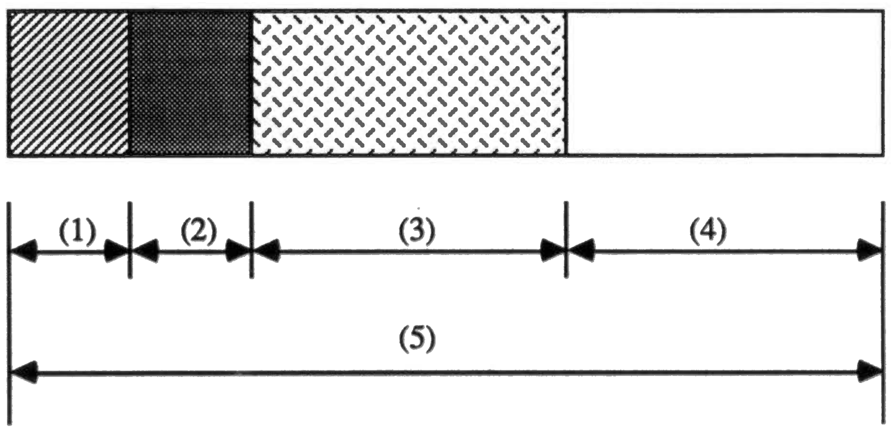
[19] Chen, G. Convergence Analysis for Inexact Mechanization of Kalman Filtering, *IEEE Transactions on Aerospace and Electronic Systems*, vol. 28, no. 3, July 1992, pp 612 - 621.

[20] Gelb (editor), Applied Optimal Estimation. The MIT Press, Massachusetts, 1974, p 343.

APPENDIXES

APPENDIX A

FIGURES



(1) Time for the calculation of covariance matrix, P

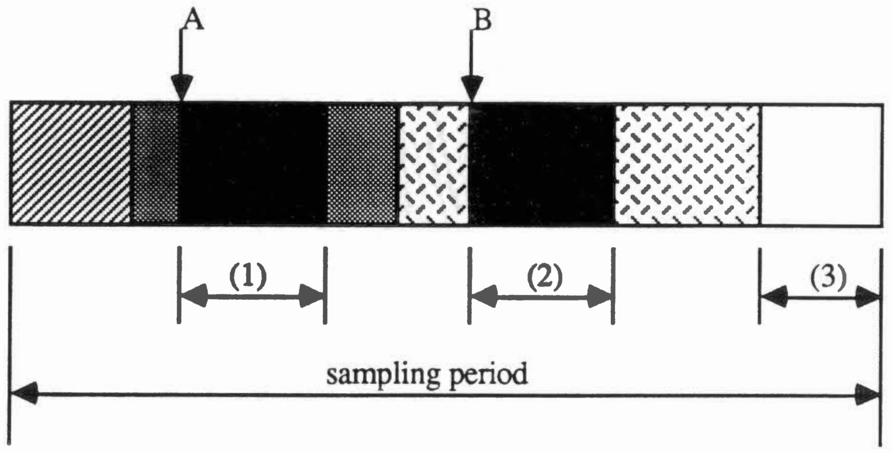
(2) Time for the calculation of state update, X

(3) Time for the calculation of optimal gain, K

(4) Time available for interrupt service routines

(5) Total time of a sampling period

Figure 1. Execution Time Allocation in a Sampling Period

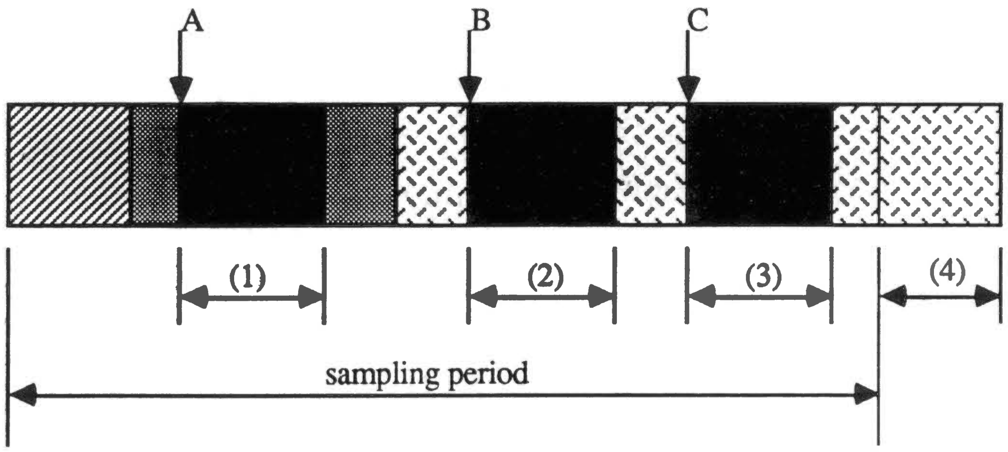


(1) Interrupt A service routine

(2) Interrupt B service routine

(3) Time left in a sampling period

Figure 2. Execution Time Allocation in a Sampling Period with Real-Time Interrupts



(1) Interrupt A service routine

(2) Interrupt B service routine

(3) Interrupt C service routine

(4) Extra time needed to get the optimal result

Figure 3. Execution Time Allocation in a Sampling Period with Real-Time Interrupts Violating the Hard-Time Constraint

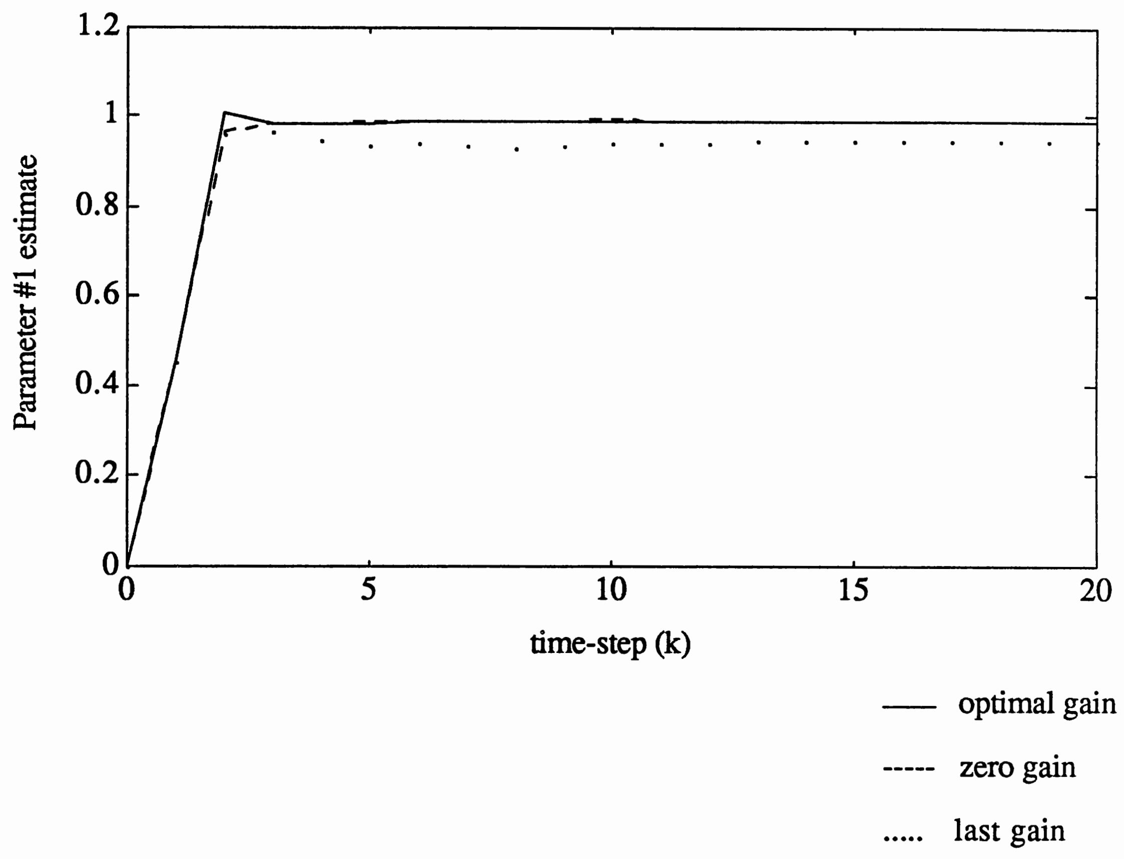


Figure 4. Average Estimates for System (1)

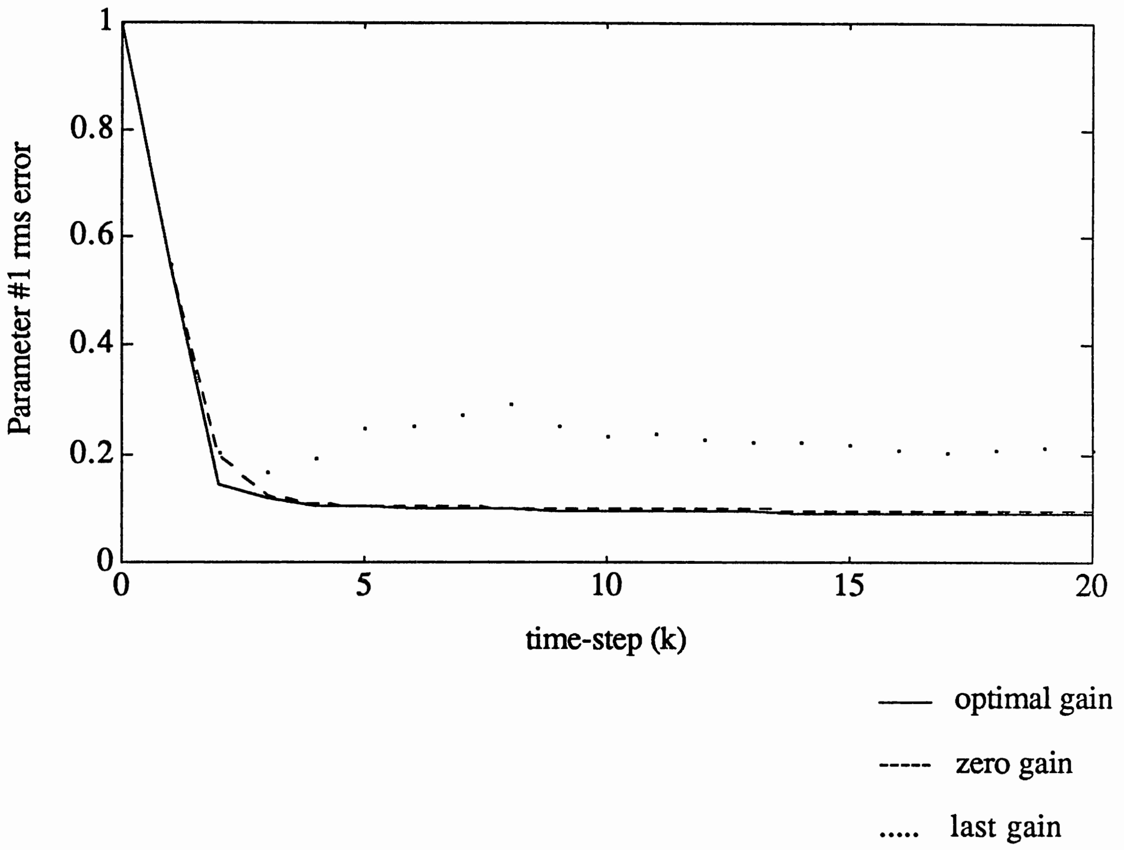


Figure 5. RMS Errors for System (1)

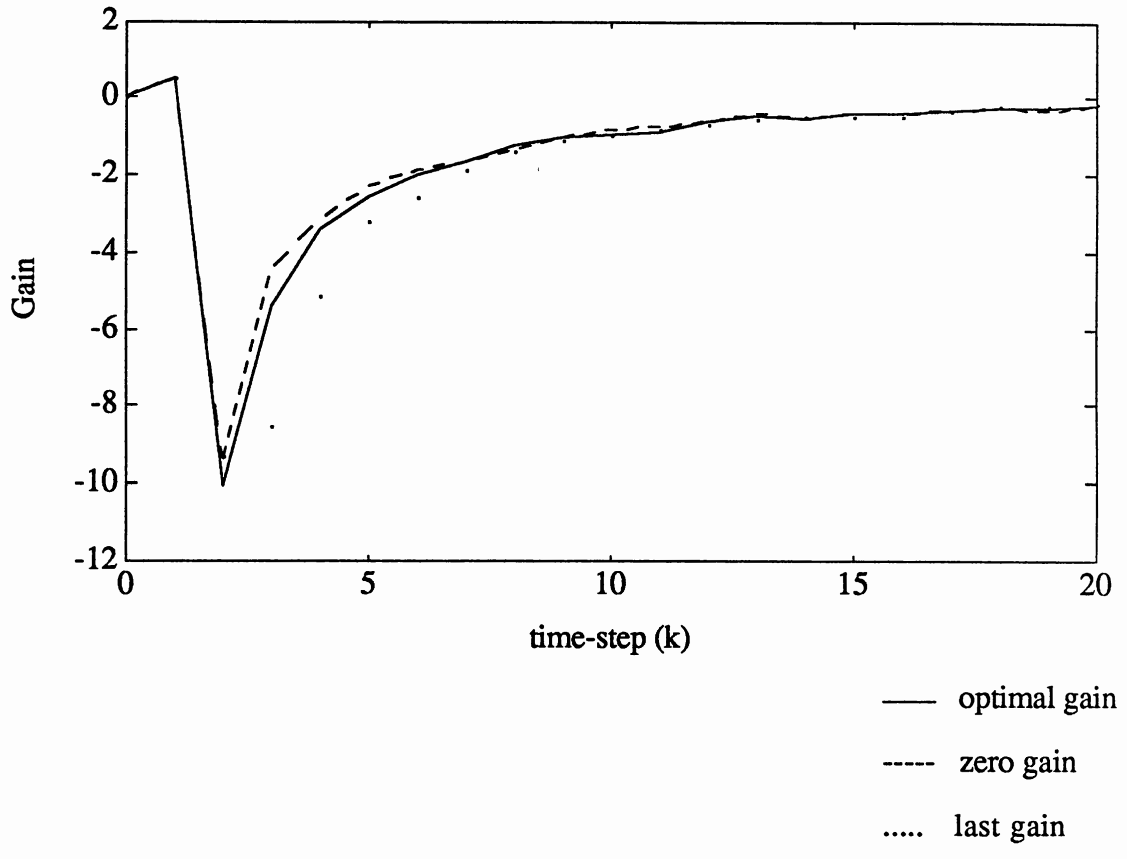


Figure 6. Average Gain for System (1)

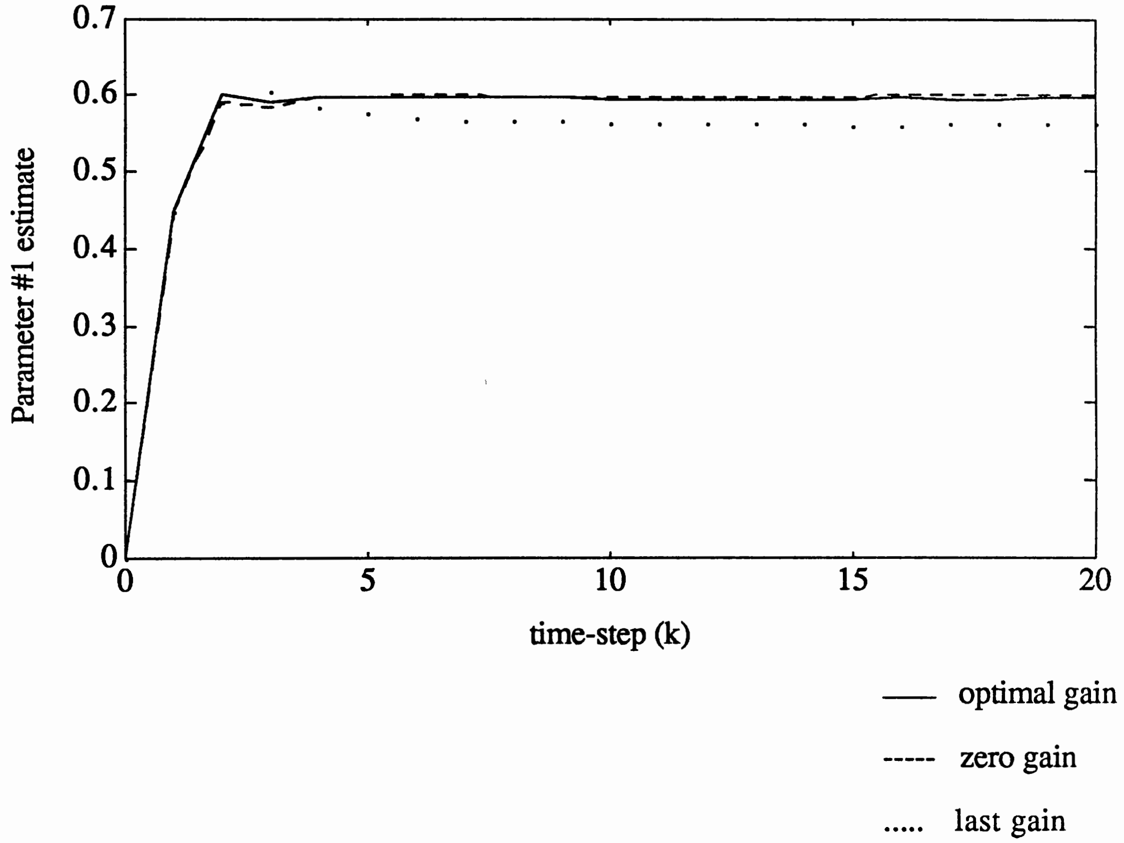
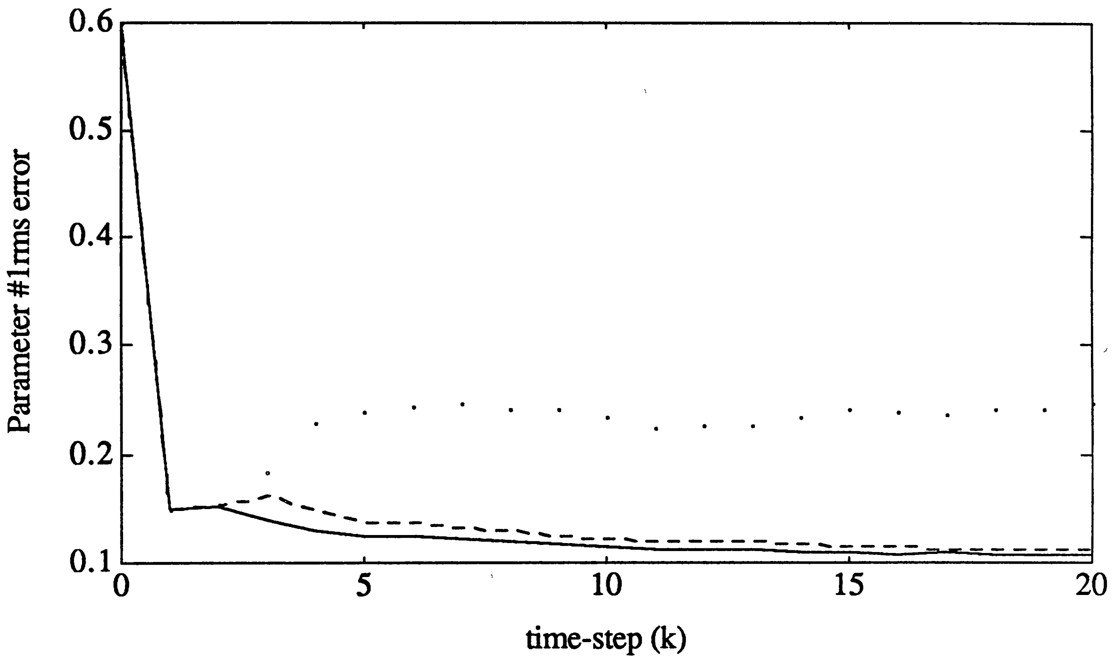


Figure 7. Average Estimates for System (2)

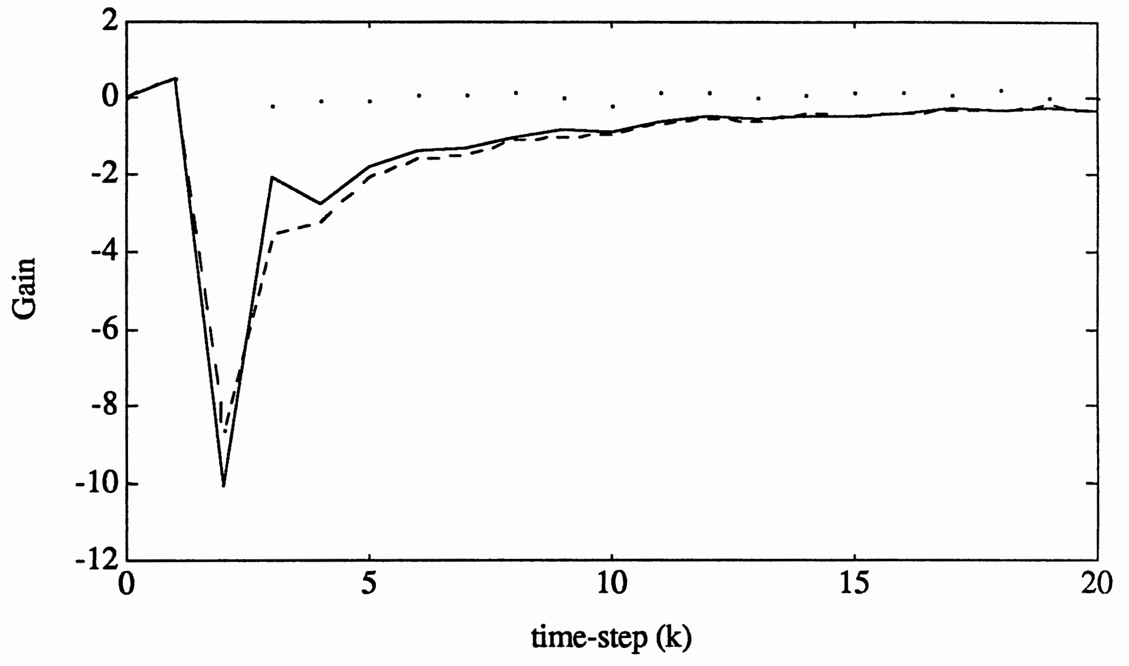


—— optimal gain

----- zero gain

..... last gain

Figure 8. RMS Errors for System (2)



—— optimal gain

----- zero gain

..... last gain

Figure 9. Average Gain for System (2)

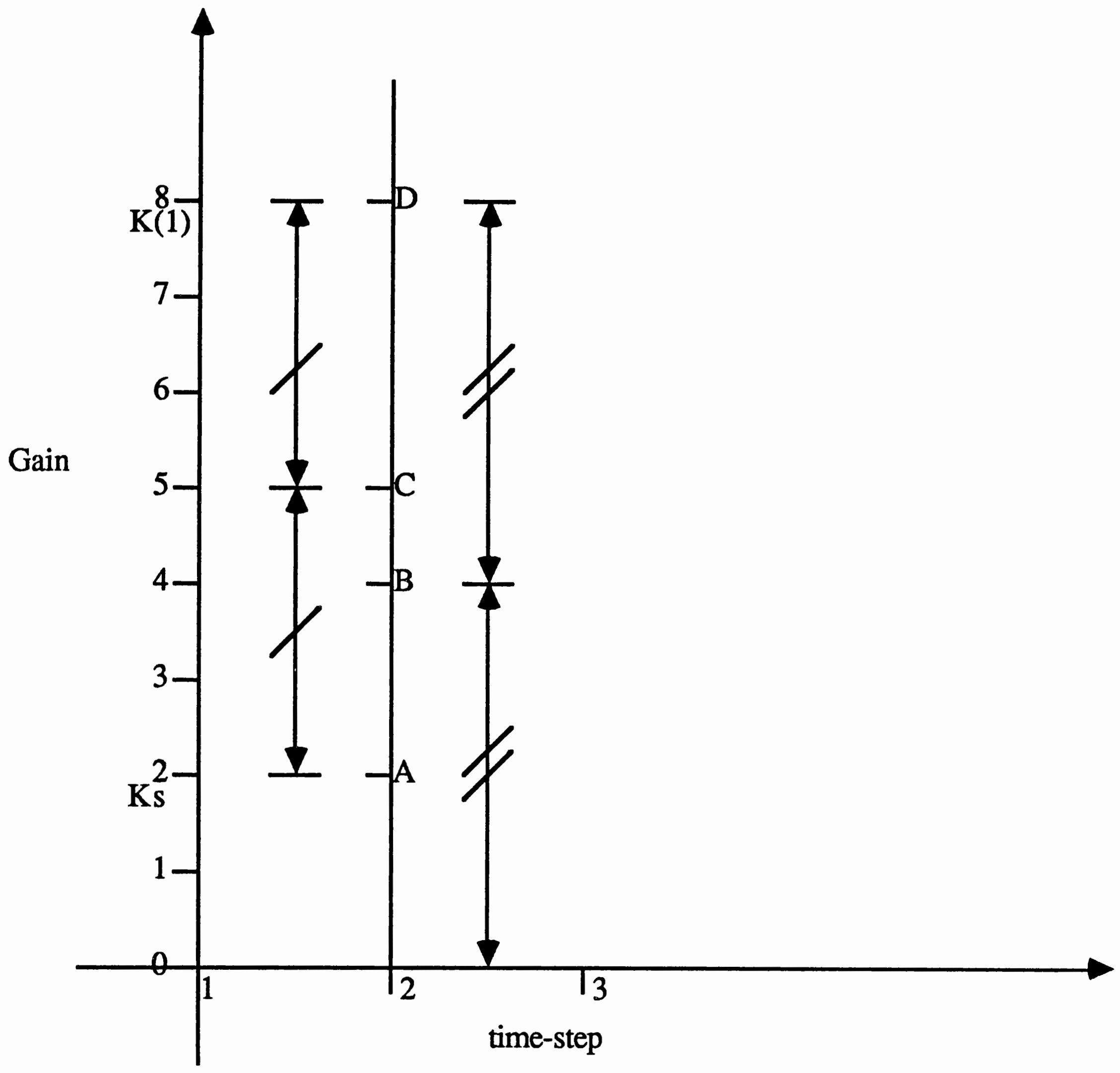
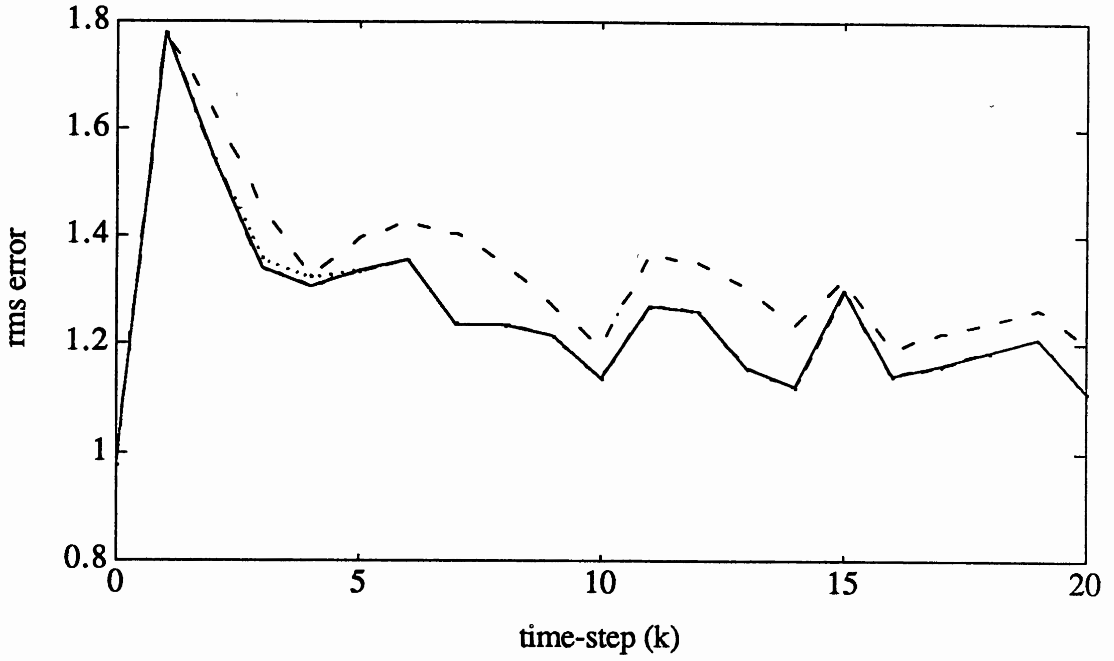


Figure 10. Pattern of Gains



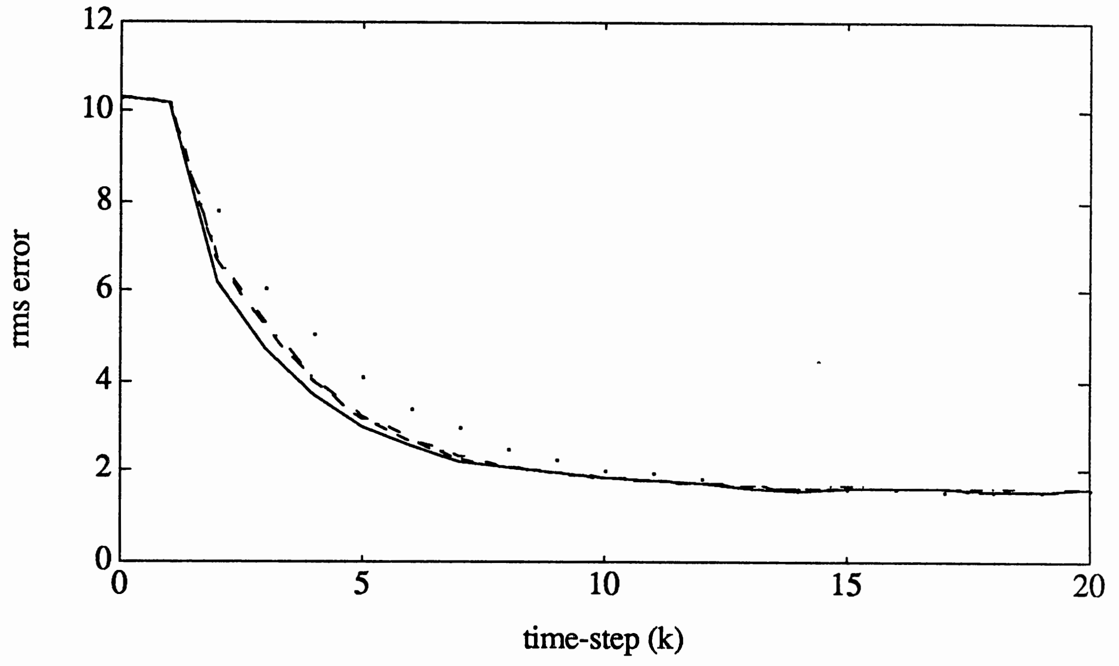
—— optimal gain

----- steady gain

..... last gain

_._. zero gain

Figure 11. RMS Estimation Errors for System (3)



——	optimal gain
-----	steady gain
.....	last gain
_._.	zero gain

Figure 12. RMS Estimation Errors for System (4)

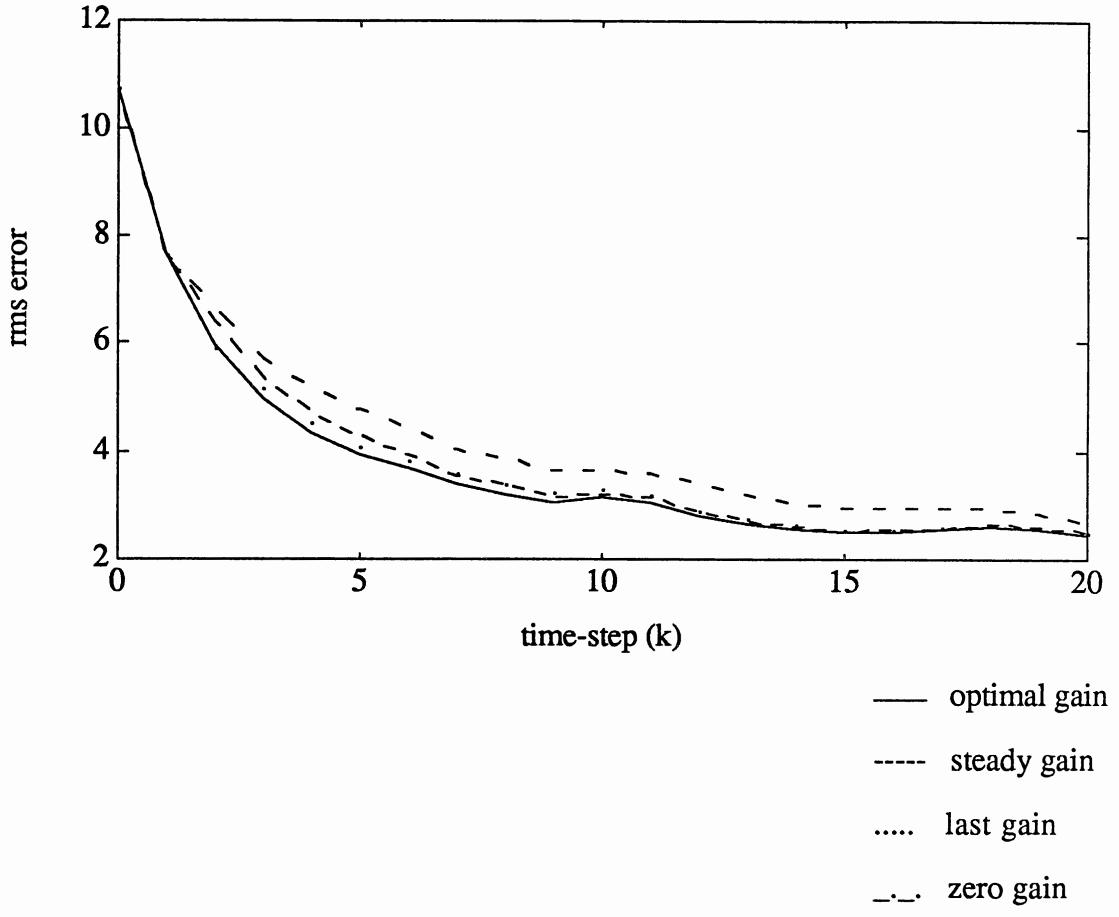


Figure 13. RMS Estimation Errors for System (5)

APPENDIX B

MATLAB CODE FOR PARAMETER IDENTIFICATION

```
% A MATLAB function to perform Monte Carlo simulation of real-time
% parameter identification.
%
% Format:
%       [a, b, c, d, e, f, g, h, i] = mcs(runs)
%
% Input: number of simulation runs          (runs)
% Outputs:- (Monte Carlo averages)
%   1st element: optimal gain history       (a)
%   2nd element: last gain history          (b)
%   3rd element: zero gain history          (c)
%   4th element: optimal parameter estimates (d)
%   5th element: last gain estimates        (e)
%   6th element: zero gain estimates        (f)
%   7th element: optimal errors             (g)
%   8th element: last gain errors           (h)
%   9th element: zero gain errors           (i)
%
% Function called: pariden(no. of time steps in each run)
%
% Note: Checks validity of simulation for inclusion


function [mean_k, mean_k_l, mean_k_z, mean_t, mean_t_l, mean_t_z, ...
        rms_err_t, rms_err_t_l, rms_err_t_z] =  mcs(runs)


 %   n = no. of time steps in each run
```

```
disp('');
n = input('Enter number of time steps in each run:');   % read n


mat_theta = ones(n+1, 2) * [0.6  0; 0  0.3];
sum_t = zeros(n+1, 2);
sum_t_l = zeros(n+1, 2);
sum_t_z = zeros(n+1, 2);


sum_sq_err_t = zeros(n+1, 2);
sum_sq_err_t_l = zeros(n+1, 2);
sum_sq_err_t_z = zeros(n+1, 2);


sum_k = zeros(n+1, 2);
sum_k_l = zeros(n+1, 2);
sum_k_z = zeros(n+1, 2);


sum_sq_err_k_l = zeros(n+1, 2);
sum_sq_err_k_z = zeros(n+1, 2);



count = 0;
t = [0 : n]';


for k = 1: runs,
        disp('run =');
        disp(k);
        [vec_k, vec_k_l, vec_k_z, t_opt, t_l, t_z] = pariden(n);
```

```
if [vec_k, vec_k_l, vec_k_z] < 50,
   sum_t = sum_t + t_opt;
   sum_t_l = sum_t_l + t_l;
   sum_t_z = sum_t_z + t_z;

   err_t = mat_theta - t_opt;
   sq_err_t = err_t .^2;
   sum_sq_err_t = sum_sq_err_t + sq_err_t;

   sum_k = sum_k + vec_k;

   err_t_l = mat_theta - t_l;
   sq_err_t_l = err_t_l .^2;
   sum_sq_err_t_l = sum_sq_err_t_l + sq_err_t_l;

   sum_k_l = sum_k_l + vec_k_l;

   err_k_l = vec_k - vec_k_l;
   sq_err_k_l = err_k_l .^2;
   sum_sq_err_k_l = sum_sq_err_k_l + sq_err_k_l;

   err_t_z = mat_theta - t_z;
   sq_err_t_z = err_t_z .^2;
   sum_sq_err_t_z = sum_sq_err_t_z + sq_err_t_z;

   sum_k_z = sum_k_z + vec_k_z;
```

```
            err_k_z = vec_k - vec_k_z;
            sq_err_k_z = err_k_z .^2;
            sum_sq_err_k_z = sum_sq_err_k_z + sq_err_k_z;

            count = count + 1;
        end;

end;

disp('count = ');
disp(count);
disp('');
disp('Press any key to continue!');
pause;

mean_t = sum_t / count;
mean_t_l = sum_t_l / count;
mean_t_z = sum_t_z / count;

mean_sq_err_t = sum_sq_err_t / count;
mean_sq_err_t_l = sum_sq_err_t_l / count;
mean_sq_err_t_z = sum_sq_err_t_z / count;

rms_err_t = mean_sq_err_t .^0.5;
rms_err_t_l = mean_sq_err_t_l .^0.5;
rms_err_t_z = mean_sq_err_t_z .^0.5;
```

```
mean_k = sum_k / count;

mean_k_l = sum_k_l / count;

mean_k_z = sum_k_z / count;


mean_sq_err_k_l = sum_sq_err_k_l / count;

mean_sq_err_k_z = sum_sq_err_k_z / count;


% plot gains


clg;

subplot(121);

plot(t, mean_k(:,1), 'w+', t, mean_k_l(:,1), 'r-', t, mean_k_z(:,1), 'w-.');

title('gains: +++ opt; --- last-gain; -.-. zero-gain');

subplot(122);

plot(t, mean_k(:,2), 'w+', t, mean_k_l(:,2), 'r-', t, mean_k_z(:,2), 'w-.');

pause;


% plot mean square errors in gains


clg;


subplot(121);

plot(t, mean_sq_err_k_l(:,1), 'r-', t, mean_sq_err_k_z(:,1), 'w-.')

title('mean sq err of gains');

subplot(122);

plot(t, mean_sq_err_k_l(:,2), 'r-', t, mean_sq_err_k_z(:,2), 'w-.')

pause
```

```
% plot estimates


clg;
subplot(121);
plot(t, mean_t(:,1), 'w+', t, mean_t_l(:,1), 'r-', t, mean_t_z(:,1), 'w-.')
title('+++ optimal; --- last gain; -.-. zero gain')
xlabel('time-step (25% interrupt rate)')
ylabel('Parameter #1 Estimate')
 subplot(122);
 plot(t, mean_t(:,2), 'w+', t, mean_t_l(:,2), 'r-', t, mean_t_z(:,2), 'w-.')
 xlabel('time-step (25% interrupt rate)')
 ylabel('Parameter #2 Estimate'), pause


% estimation errors


clg;
subplot(121);
plot(t, rms_err_t(:,1), 'w+', t, rms_err_t_l(:,1), 'r-', t, rms_err_t_z(:,1), 'w-.')
xlabel('time-step (25% interrupt rate)')
ylabel('Parameter #1 rms error'),
title('+++ optimal; --- last gain; -.-. zero gain')


 subplot(122);
 plot(t, rms_err_t(:,2), 'w+', t, rms_err_t_l(:,2), 'r-', t, rms_err_t_z(:,2), 'w-.')
 ylabel('Parameter #2 rms error'), pause
```

```
% This is a program for Real-Time parameter identification using recursive-
% least-squares algorithm.
%
% Format:
%         [a, b, c, d, e, f] = pariden(n)
%
% Input: no. of time steps                     (n)
% Outputs:-
%      1st element: optimal gain history       (a)
%      2nd element: last gain history          (b)
%      3rd element: zero gain history          (c)
%      4th element: optimal estimates          (d)
%      5th element: last gain estimates        (e)
%      6th element: zero gain estimates        (f)
%
% Functions called : NONE

% Three methods are considered:
%   (i) optimal gain
%  (ii) last gain when interrupt
% (iii) zero gain when interrupt
%
%            The system considered is of the form:
%               x(k+1) = 0.6 * x(k) + 0.3 * x(k-1) + vk,
%         i.e., y    =  phi * theta + vk;
%                             vk is system noise;
%                             'theta' is to be identified.
```

```
function [vec_k, vec_k_last, vec_k_zero, vec_tetahat, vec_tetahat_last, vec_tetahat_zero] =
pariden(n)

% Note:-  unsuffixed variables refer to optimal scheme

theta = [0.6; 0.3];

p = 10e6 * eye(2);
 p_last_k_int = 10e6 * eye(2);
  p_zero_k_int = 10e6 * eye(2);

tetahat = [0; 0];           %  initial estimate of parameters
 tetahat_last_k_int = [0; 0];
  tetahat_zero_k_int = [0; 0];
vec_tetahat = tetahat';
 vec_tetahat_last = tetahat_last_k_int';
  vec_tetahat_zero = tetahat_zero_k_int';

k = [0; 0];
 k_last_k_int = [0; 0];    %  initialise gain to create vector
  k_zero_k_int = [0; 0];
vec_k = k';
 vec_k_last = k_last_k_int';
  vec_k_zero = k_zero_k_int';

x(1) = 1;  x(2) = 1;      %  initial state of the system
rand('uniform');
```

```
intr = rand(n, 1);          %  interrupt sequence generation
intr(1, 1) = 0;
rand('normal');
vk =  0.1*rand(n,1);   %   system noise ; noise level = 0.1^2

for i = 1 : n;

 phi = [x(i+1) x(i)];
 x(i+2) = phi * theta + vk(i, 1);   %  System Simulation
 y(i+1) = x(i+2);                   %  Data Point Generation

 k = p * phi'/(1 + phi * p * phi');

 if intr(i,1) < 0.750             % checking for interrupt
   k_last_k_int = p_last_k_int * phi'/(1+ phi * p_last_k_int * phi');
   k_zero_k_int = p_zero_k_int * phi'/(1+ phi * p_zero_k_int * phi');
 else
   k_last_k_int = k_last_k_int;
   k_zero_k_int = [0; 0];
 end

 p = (eye(2) - k * phi) * p;
  p_last_k_int = (eye(2) - k_last_k_int * phi) * p_last_k_int;
   p_zero_k_int = (eye(2) - k_zero_k_int * phi) * p_zero_k_int;

 tetahat = tetahat + k * (y(i+1) - phi * tetahat);
  tetahat_last_k_int = tetahat_last_k_int + k_last_k_int * (y(i+1) - phi * tetahat_last_k_int);
```

```
    tetahat_zero_k_int = tetahat_zero_k_int + k_zero_k_int * (y(i+1) - phi *
tetahat_zero_k_int);

  vec_k = [vec_k; k'];
   vec_k_last = [vec_k_last; k_last_k_int'];
    vec_k_zero = [vec_k_zero; k_zero_k_int'];

  vec_tetahat = [vec_tetahat; tetahat'];
   vec_tetahat_last = [vec_tetahat_last; tetahat_last_k_int'];
    vec_tetahat_zero = [vec_tetahat_zero; tetahat_zero_k_int'];
end
```

APPENDIX C

MATLAB CODE FOR STATE ESTIMATION

```
% This is a Matlab function to perform Monte Carlo simulation of
% state estimation for a first order system.
%
% Format:
%       [a, b, c, d, e, f, g] = mck(runs)
%
% Input: number of simulation runs                    (runs)
% Outputs:-
%      1st element: optimal-gain rms error            (a)
%      2nd element: last-gain rms error               (b)
%      3rd element: zero-gain rms error               (c)
%      4th element: steady-gain rms error             (d)
%      5th element: optimal gain                      (e)
%      6th element: ave. of last and steady gains     (f)
%      7th element: ave. of last and zero gains       (g)
%
% Functions called: kal(number of time steps in each run, phi, k_steady)


function [h_rms_err, h_rms_err_stdy, h_rms_err_last, ...
        h_rms_err_zero, h_k, h_k_ls, h_k_lz] = mck(runs)


% n = no. of time steps in each run


disp('')
n = input('Enter number of time steps in each run:');   % read n


phi = 0.8; gama = 1;
```

```
c = 1;
p = 1e6*eye(1);
r1 = 1*eye(1);
r2 = 1;
k = 0;
h_k = k';
for i = 1:n,            % find k_steady
   pp = phi*p*phi' + gama*r1*gama';
   k = pp*c'*inv(c*pp*c' + r2);
   p = (eye(1) - k*c) * pp;
   h_k = [h_k; k'];
end;
k_steady = k,
disp('Hit any key!');
pause
    h_k = h_k(2:n+1,:);       % actual gain
    long_k_steady = ones(n, 1)* k_steady';
    h_k_ls = 0.5*(h_k + long_k_steady);
    h_k_ls = [h_k(1,:); h_k_ls(1:n-1,:)];


    h_k_lz = 0.5 * h_k;
    h_k_lz = [h_k(1,:); h_k_lz(1:n-1,:)];


    t = [1: n];
     plot(t, h_k, t, h_k_ls, '--', t, h_k_lz, ':');
    pause;
sum_x = zeros(n+1, 1);
```

```
sum_x_hat = zeros(n+1, 1);
sum_x_hat_last = zeros(n+1, 1);
sum_x_hat_zero = zeros(n+1, 1);
sum_x_hat_stdy = zeros(n+1, 1);

h_sum_sq_err = zeros(n+1, 1);
h_sum_sq_err_last = zeros(n+1, 1);
h_sum_sq_err_zero = zeros(n+1, 1);
h_sum_sq_err_stdy = zeros(n+1, 1);

t = [0 : n]';

for r = 1: runs,
     disp('run =');
    disp(r);
    [h_x, h_x_hat, h_x_hat_last, h_x_hat_zero, h_x_hat_stdy] = kal(n, phi, k_steady);
    sum_x = sum_x + h_x;
    sum_x_hat = sum_x_hat + h_x_hat;
    sum_x_hat_last = sum_x_hat_last + h_x_hat_last;
    sum_x_hat_zero = sum_x_hat_zero + h_x_hat_zero;
    sum_x_hat_stdy = sum_x_hat_stdy + h_x_hat_stdy;

    h_err = h_x - h_x_hat;
    h_sq_err = h_err .^2;
    h_sum_sq_err = h_sum_sq_err + h_sq_err;

    h_err_last = h_x - h_x_hat_last;
    h_sq_err_last = h_err_last .^2;

    h_sum_sq_err_last = h_sum_sq_err_last + h_sq_err_last;


    h_err_zero = h_x - h_x_hat_zero;

    h_sq_err_zero = h_err_zero .^2;

    h_sum_sq_err_zero = h_sum_sq_err_zero + h_sq_err_zero;


    h_err_stdy = h_x - h_x_hat_stdy;

    h_sq_err_stdy = h_err_stdy .^2;

    h_sum_sq_err_stdy = h_sum_sq_err_stdy + h_sq_err_stdy;


end;


h_x = sum_x / runs;

h_x_hat = sum_x_hat / runs;

h_x_hat_last = sum_x_hat_last / runs;

h_x_hat_zero = sum_x_hat_zero / runs;

h_x_hat_stdy = sum_x_hat_stdy / runs;


h_mean_sq_err = h_sum_sq_err / runs;

h_mean_sq_err_last = h_sum_sq_err_last / runs;

h_mean_sq_err_zero = h_sum_sq_err_zero / runs;

h_mean_sq_err_stdy = h_sum_sq_err_stdy / runs;


h_rms_err = h_mean_sq_err .^0.5;

h_rms_err_last = h_mean_sq_err_last .^0.5;

h_rms_err_zero = h_mean_sq_err_zero .^0.5;
```

```
h_rms_err_stdy = h_mean_sq_err_stdy .^0.5;


% plot
clg;


% state
plot(t, h_x(:,1), 'r-', t, h_x_hat(:,1), 'w+', t, h_x_hat_last(:,1), 'r:', ...
        t, h_x_hat_zero(:,1), 'wo', t, h_x_hat_stdy(:,1), 'bx');
xlabel('time-step (25% interrupt rate)')
ylabel('Estimate')
title('--- actual; +++ opt k; ... last k; ooo zero k; xxx stdy k');
pause


% error
plot(t, h_rms_err(:,1), 'w+', t, h_rms_err_last(:,1), 'r-', ...
        t, h_rms_err_zero(:,1), 'wo', t, h_rms_err_stdy(:,1), 'bx');
xlabel('time-step')
ylabel('Estimation rms error')
title('+++ opt k; --- last k; ooo zero k; xxx stdy k');
pause
clg
```

```
% This is a Matlab function to perform real-time state estimation
% for a first order system.
%
% Format :
%       [a, b, c, d, e] = kal(number of time steps, phi, k_steady)
%
% Input: number of time steps, phi, steady gain
% Outputs:-
%       1st element: actual states              (a)
%       2nd element: optimal estimates          (b)
%       3rd element: last-gain estimates        (c)
%       4th element: zero-gain estimates        (d)
%       5th element: steady-gain estimates      (e)
%
% Functions called: NONE

function [h_x, h_x_hat, h_x_hat_last, h_x_hat_zero, h_x_hat_stdy] = kal(n,phi,k_steady)

% n = no. of time steps

 gama = 1;
 c = 1;
 d = 0;
 u = 1;

rand('uniform');
intr = rand(n,1);      % generate interrupt sequence
```

```
intr(1,1) = 0;        % no interrupt first step
rand('normal');
r1 = 1;               % noise levels
r2 = 1;
v = rand(n,1);
e = rand(n,1);
p0 = 1e6*eye(1);      % initial covariance
p = p0;
 p_last = p0;
  p_zero = p0;
   p_stdy = p0;


% initialise system

x = (r2^0.5) * rand;
h_x = x';

x_hat = 0;
h_x_hat = x_hat';
 x_hat_last = 0;
 h_x_hat_last = x_hat_last';
  x_hat_zero = 0;
  h_x_hat_zero = x_hat_zero';
   x_hat_stdy = 0;
   h_x_hat_stdy = x_hat_stdy';
```

```
for i = 1: n;

  y = c*x + 1*e(i,1);              % system simulation
  x = phi*x + u + 1*v(i,:)';
  h_x = [h_x; x'];

  pp = phi*p*phi' + gama*r1*gama';             % optimal scheme
  k = pp*c'*inv(c*pp*c' + r2);
  p = (eye(1) - k*c) * pp;

  if intr(i,1) > 0.750                 % on interrupt
    pp_last = phi*p_last*phi' + gama*r1*gama';
    k_last = k_last;
    p_last = (eye(1) - k_last*c) * pp_last;
     pp_zero = phi*p_zero*phi' + gama*r1*gama';
     k_zero = 0;
     p_zero = (eye(1) - k_zero*c) * pp_zero;
      pp_stdy = phi*p_stdy*phi' + gama*r1*gama';
      k_stdy = k_steady;
      p_stdy = (eye(1) - k_stdy*c) * pp_stdy;
  else
    pp_last = phi*p_last*phi' + gama*r1*gama';
    k_last = pp_last*c'*inv(r2 + c*pp_last*c');
    p_last = (eye(1) - k_last*c) * pp_last;
     pp_zero = phi*p_zero*phi' + gama*r1*gama';
     k_zero = pp_zero*c'*inv(r2 + c*pp_zero*c');
     p_zero = (eye(1) - k_zero*c) * pp_zero;
```

```
      pp_stdy = phi*p_stdy*phi' + gama*r1*gama';
      k_stdy = pp_stdy*c'*inv(r2 + c*pp_stdy*c');
      p_stdy = (eye(1) - k_stdy*c) * pp_stdy;

   end;

   xx_hat = phi*x_hat + u;
   x_hat = xx_hat + k*(y - c*xx_hat);
   h_x_hat = [h_x_hat; x_hat'];
    xx_hat_last = phi*x_hat_last + u;
    x_hat_last = xx_hat_last + k_last*(y - c*xx_hat_last);
    h_x_hat_last = [h_x_hat_last; x_hat_last'];
     xx_hat_zero = phi*x_hat_zero + u;
     x_hat_zero = xx_hat_zero + k_zero*(y - c*xx_hat_zero);
     h_x_hat_zero = [h_x_hat_zero; x_hat_zero'];
      xx_hat_stdy = phi*x_hat_stdy + u;
      x_hat_stdy = xx_hat_stdy + k_stdy*(y - c*xx_hat_stdy);
      h_x_hat_stdy = [h_x_hat_stdy; x_hat_stdy'];

end;     % end for loop

     end;
```

2

VITA

Ram Ashok Viswanath

Candidate for the Degree of

Master of Science

Thesis: ALGORITHM DESIGN FOR REAL-TIME DISTRIBUTED SYSTEMS

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Tirunelveli, Tamilnadu, India, June 05, 1968, the son of Ram Balaji and Lakshmi Viswanath.

Education: Graduated from DAV Higher Secondary School, Madras, India, in June 1985; received Bachelor of Technology degree in Mechanical Engineering from Indian Institute of Technology at Madras in May 1990; completed requirements for the Master of Science degree at Oklahoma State University in December 1992.

Professional Experience: Research Assistant, Dr. Gary Young, Department of Mechanical Engineering, Oklahoma State University, May 1991, to December 1991; Teaching Assistant, Department of Mechanical Engineering, Oklahoma State University, January 1991, to May 1992.

Professional Organizations: American Society of Mechanical Engineers.