THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

A UNIFIED METHOD FOR EVALUATING REAL-TIME COMPUTER CONTROLLERS: A CASE STUDY

K.G. Shin, C.M. Krishna
and
Y.H. Lee

CRL-TR-23-83

JUNE 1983

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]This research has been supported in part by NASA Grant No. NAG 1-296. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agency.

A UNIFIED METHOD FOR EVALUATING REAL-TIME COMPUTER CONTROLLERS: A CASE STUDY[1]

K.G. Shin, Senior Member, IEEE, C. M. Krishna, Student Member, IEEE, and Y.H. Lee, Student Member, IEEE

Computing Research Laboratory
Department of Electrical and Computer Engineering
The University of Michigan
Ann Arbor, Michigan 48109

ABSTRACT

A real-time control system consists of a synergistic pair, that is, a *controlled process* and a *controller computer*. We have defined new performance measures for real-time controller computers on the basis of the nature of this synergistic pair.

In this report we present a case study of a typical critical controlled process in the context of new performance measures that express the performance of both controlled processes and real-time controllers (taken as a unit) on the basis of a single variable: *controller response time*. Controller response time is a function of current system state, system failure rate, electrical and/or magnetic interference, etc., and is therefore a random variable. Control overhead is expressed as a monotonically non-decreasing function of the response time and the system suffers catastrophic failure, or *dynamic failure*, if the response time for a control task exceeds the corresponding system *hard deadline*, if any. A rigorous probabilistic approach is used to estimate the performance measures.

The controlled process chosen for study is an aircraft in the final stages of descent, just prior to landing. Control constraints are particularly severe during this period, and great care must be taken in the design of controllers that handle this process. First, the performance measures for the controller are presented. Secondly, control algorithms for solving the landing problem are discussed and finally the impact of our performance measures on the problem is analyzed, showing that the performance measures and the associated estimation method have great potential use for designing and/or evaluating real-time controllers and controlled processes. Also, one application for the design of controller computers, presented in detail, is checkpointing for enhanced reliability.

Index Term- Controlled process(es), controller computers, hard deadlines, response time, performance measures, allowed state space, aircraft landing, checkpointing.

[1]*This work has been supported in part by NASA Grant No. NAG 1-296. Any opinions, findings, and conclusions or recommendations expressed in this report are those of the authors and do not necessarily reflect the views of the funding agency.*

1. INTRODUCTION

Any real-time system can be regarded as a composite of *controlled* subsystems (henceforth called *controlled processes)* and *controller* subsystem(s). Traditionally, the performance of real-time control computers has been analyzed separately from that of the corresponding controlled processes. For example, the response delay caused by the controller is neither studied rigorously nor reflected carefully into the design of control algorithms for the controlled processes. The design of the controller is frequently based on ad hoc requirements imposed by control designers. While this yields acceptable results in the control of non-critical processes, such an approach needs to be improved in the design of controllers for critical processes e.g. aircraft. What is called for is a procedure for specifying and evaluating controller performance, enabling systematic application and providing objective results that lend themselves to formal validation. The use of computers as real-time controllers is becoming increasingly attractive due to continuing advances in the development of inexpensive, powerful microprocessors and memories. However, performance measures presently used to characterize real-time computer systems are adapted versions of those employed for more conventional computers. There is a considerable mismatch between the requirements of real-time applications and what is provided by these measures.

To solve this problem, several contortions of the conventional measures have been proposed. Generally, these involve representing real-time computer performance as a vector $\mathbf{p} \in \mathbf{R}^p$, made up of such traditional indices as (conventional) reliability, throughput, survivability, availability, etc. However, it is impossible to compare two performance vectors (and therefore the corresponding computer systems) without an associated metric. One straightforward approach is to use a linear metric(i.e. inner product) to map the vector into a scalar that is then claimed to represent the performance of the system. For example, the mapping can be carried out by assigning weights to the various components of the performance vector and adding them to produce the scalar. That is, if the weight vector is $\mathbf{w}^T = (w_1, \ldots, w_m)$, then the mapping is $f : \mathbf{R}^m \rightarrow \mathbf{R}$ with $f(\mathbf{p}) = \mathbf{w}^T \mathbf{p}$.

This process of ascribing weights is largely subjective and is therefore inaccurate to begin with. Even if the weight ascription were completely objective, serious practical difficulties would remain. For example, since the components of the performance vector are mutually dependent (sometimes in a very complex manner), the weights (that are supposed to define the sensitivity of the scalar to the respective vector components) must be modified by (often very complex) correction factors to account for this coupling.[2] Furthermore, relating the resulting scalar to "real-world" performance parameters (such as operating cost, etc.) is difficult.

The performance measures we introduced in [1] are designed to get around these difficulties by expressing the performance objectively in terms of the response time of the computer-controller. From the point of view of the controlled process, the computer controlling it is a black box whose behavior is exemplified by its response time and reliability. It is well known that controller delay has a detrimental effect on process behavior, our measures take the form of a quantification of this. The performance measures are considered in Section 2 in some detail prior to the presentation of an idealized case-study of their application.

The case-study is that of a real-time computer in charge of an aircraft in its final phase of flight, just prior to touchdown. There are stringent control constraints that must be met. These consist of limits on the speed of touchdown (both horizontal and vertical), the angle of attack, α, and the pitch angle, ϑ. For a definition of these angles, see Figure 1. These constraints are variously intended to safeguard against running out of runway, undercarriage collapse, stalling, and landing either on the aircraft nose or tail. Insofar as this is a control problem with severe constraints, the

[2] *This is because the weights are supposed to represent the total derivatives of the mapped scalar to their respective components. However, since the components of $\mathbf{p}$ are not orthogonal, this is not true for the unmodified weights: they represent the partial derivatives which are not equal in this case to the respective total derivatives.*

problem is typical of many other critical applications, such as the control of nuclear reactors, the generation and distribution of electrical power, life-support systems, etc. Since our objective here is to illustrate the use of our performance measures and not to solve a control problem, the aircraft system is somewhat idealized in this report.

Figure 2 shows the block diagram of a typical control system. The inputs to the controller are from sensors that provide data about the controlled process, and from the environment. This is typically fed to the computer-controller at regular intervals. Data rates are usually low: generally fewer than 20 words a second for each sensor.

Central to the operation of the system is the trigger generator. In most systems, this is physically part of the controller itself, but we separate them here for purposes of clarity. It is the function of the trigger generator to initiate execution of a controller job (defined later). Triggers can be classed into three categories.

(1) *Time-generated trigger:* These are generated at regular intervals, and lead to the corresponding controller job being initiated at regular intervals. In control theoretic terms, these are open-loop triggers.

(2) *State-generated trigger:* These are closed-loop triggers, generated whenever the system is in a particular set of states. For practicality, it might be necessary to space these triggers by more than a specified minimum duration. If time is to be regarded as an implicit state variable, the time-generated trigger is a special case of the state-generated trigger. One can also have combinations of the two.

(3) *Operator-generated trigger:* The operator can generally over-ride the automatic systems, generating and cancelling triggers at will.

The output of the controller is fed to the actuators and/or the display panel(s). Since the actuators are mechanical devices and the displays are meant as a human interface, the data rates here are usually very low. Indeed, as we have pointed out elsewhere [13], a computer control system exhibits a fundamental dichotomy, with the I/O being carried out at rather low rates and the computations having to be carried out at very high rates owing to real-time constraints on control.

The controller in our case-study is a real-time computer. It executes predefined control jobs. There is a certain number of control jobs in any control system that are executed repeatedly.

A control system executes "missions." These are periods of operation between successive periods of maintenance. In the case of aircraft, a mission is usually a single flight. The operating interval can sometimes be divided down into consecutive sections that can be distinguished from each other. These sections are called *phases*. For example, Meyer *et. al* [6] define the following four distinct phases in the mission lifetime of a civilian aircraft:

(a) Takeoff/cruise until VHF Omnirange (VOR)/Distance Measuring Equipment (DME) out of range.

(b) Cruise until VOR/DME in range again.

(c) Cruise until landing is to be initiated.

(d) Landing.

The phase to be considered here is landing, it takes about 20 seconds. The controller job that we shall treat is the control of the aircraft elevator deflection during landing.[3]

The specific system employed is assumed to be organized as shown in Figure 3. Sensors report on the four key parameters: altitude, descent rate, pitch angle, and

[3] *The output of the controller is assumed to be fed into a peripheral processor that is dedicated to controlling the actuator -- in this case the elevator.*

pitch angle rate every 60 milli-seconds.[4] We have a time-generated trigger, with a time period of 60 milli-seconds. Every 60 milli-seconds, the controller computes the optimal setting for the elevator, which is the only actuator used in the landing phase.[5] The execution time for the computation is nominally 20 milli-seconds, although this can vary in practice due to failures. Since the aircraft is a dynamic system, the effects of controller delay are considerable -- as we shall see in this report.

Since the process being controlled is critical (i.e. in which some failures can lead to catastrophic consequences), variations of controller delay and other abnormal behavior by the controller must be explicitly considered. For simplicity, we do not allow job pipelining in the controller; in other words a controller job must be completed or abandoned before its successor can be initiated. The following controller abnormalities can occur:

(i) The controller orders an incorrect output to the actuator.

(ii) The controller takes substantially more than 20 milli-seconds (the nominal execution time) but less than the inter-trigger interval of 60 milli-seconds to complete executing.

(iii) The controller takes more than 60 milli-seconds to complete executing. In such a case, the abnormal job is abandoned and the new one initiated. We say that a control trigger is "missed" when this happens.

An analysis of controller performance during the landing phase must take each of the above abnormalities into account.

[4] The sensors and actuators are assumed to have their own dedicated processors for I/O purposes. When we speak of "controller delay," we also include the delay in these processors. Also, the period of 60 milli-seconds is arbitrary, and the choice of this period does not alter the method developed here.

[5] There are other actuators used aboard the aircraft for purposes of stability, horizontal speed control, etc. We do not however consider them here, concentrating exclusively on the control of the elevator.

This report is organized as follows. In Section 2 we present the performance measures that will be used, and Section 3 contains a description of the controlled system. In Section 4, we derive the measures associated with the controlled process (the aircraft), and in Section 5 we consider one example of their application for the design of real-time controllers. The report concludes with Section 6.

2. PERFORMANCE MEASURES

2.1. Review of the Performance Measures

For completeness, we review briefly in this section the performance measures to be used, which were introduced by us in [1].

The measures are all based on a single attribute: computer controller response time distribution. A real-time computer controller in general exhibits stochastic behavior.[6] Real-time computer controllers repeatedly execute predefined control jobs which are initiated either by environmental stimuli or internally.

Central to our performance measures are the concepts of *dynamic failure* and *allowed* or *admissible* state-space. Every critical process must operate within a state-space circumscribed by given constraints. This is the allowed state-space. Leaving this state-space constitutes *dynamic failure.*[7] In the example we treat here, the states are the altitude, the vertical speed, the pitch angle, and the pitch angle rate. Each of these has a constraint. For example, the aircraft must not touch down with too great a downward velocity or the undercarriage will collapse.

The performance of the controlled process naturally depends on the speed of the controller. If the controller takes longer than a certain duration to formulate the control, dynamic failure becomes possible. This duration is the *hard deadline.*

[6] *This is partially because failures are assumed to occur randomly over the operating interval. The failure law for the components of the computer is assumed to be known. Furthermore, execution of control tasks is stochastic due to the blocking at shared resources, conditional branches in task code, etc.*

[7] *Dynamic failure is so termed since it is a failure that can occur as a result of the controller not responding fast enough to the environment. It expresses the fact that slowness of the controller can be a cause of catastrophic failure.*

We define a cost function $C_\alpha(\xi)$ associated with controller response time ξ for controller job α. The cost function takes the following form:

$$C_\alpha(\xi) = \begin{cases} g_\alpha(\xi) & \text{if } 0<\xi\leq\tau_{d\alpha} \\ \infty & \text{if } \xi>\tau_{d\alpha} \\ 0 & otherwise \end{cases} \qquad (1)$$

where $g_\alpha(\bullet)$ is a suitable continuous non-decreasing function of ξ and $\tau_{d\alpha}$ is the hard deadline associated with the job α. Clearly, since the environment influences the quality of system performance the cost function is implicitly a function of the system state. Also, if $\tau_{d\alpha}$ is a finite quantity in some region of the state-space, the job is *critical* in that region. The determination of the hard deadline is treated in detail in Sections 2.2 and 2.3.

For controller response times less than the hard deadline, the cost function in (1) above is continuous, monotonically non-decreasing, and therefore always bounded for finite response time. For consistency, it is assumed that the costs accrue as the execution proceeds.

The functions (called the *finite cost functions)* g_α can be obtained using the performance indices of the controlled process. These performance indices are well-known to control theory and express the consumed energy, fuel, time, or some other physical parameter associated with the trajectory of the system as it travels from its initial to its final state. See, for example, [2,3], for details. The cost of running the controlled process over, say, an interval of time $[t_0,t_f]$, is usually expressed by:

$$\Theta = \int_{t_0}^{t_f} E[f_0(\mathbf{x}(t),\mathbf{u}(t),t)\,|\,\mathbf{y}(\tau),t_0\leq\tau\leq t]dt \qquad (2)$$

where $E[\bullet|\bullet]$ represents conditional expectation, f_0 is the instantaneous index of performance at time t, and $\mathbf{x}(t)\in\mathbf{R}^n$, $\mathbf{u}(t)\in\mathbf{R}^l$ and $\mathbf{y}(t)\in\mathbf{R}^m$ represent the state, input, and measurement vectors respectively. A good representation for $g_\alpha(\xi)$ is given by:

$$g_\alpha(\xi) = \begin{cases} \Psi_\alpha(\xi) - \Psi_\alpha(0) & \text{for } 0 \le \xi \le \tau_{d\alpha} \\ 0 & otherwise \end{cases} \tag{3}$$

where $\Psi_\alpha(\eta)$ = expected contribution of $\mathbf{u}_{c\alpha}$ to Θ if response time of that particular execution of job $\alpha = \eta$, and $\mathbf{u}_{c\alpha}$ = control input subvector associated with job α. Note that the input vector $\mathbf{u}$ for job α consists of the control input subvector, $\mathbf{u}_{c\alpha}$, as well as the environment (random) input subvector, $\mathbf{u}_{e\alpha}$.

A *version* is an instance of the execution of a task. Versions are numbered in sequence of initiation: successive versions of task i being denoted by $V_{i1}, \ldots, V_{in}$. The response time associated with a version V_{ij} is denoted by $RESP(i_j)$.

Let $q_i(t)$ represent the number of versions executed for task i over the interval $[0,t)$, and r the number of distinct tasks. Then define

$$S(t) \equiv \sum_{i=1}^{r} \Gamma_i(t)$$

where $\Gamma_i = \sum_{j=1}^{q_i(t)} h_i(RESP(i_j))$ and $h_i(t) = \begin{cases} g_i(t) & \text{if } 0 < t \le t_{di} \\ g_i(t_{di}) & \text{if } t > t_{di} \end{cases}$

For an illustration, see Figure 4. Clearly, h_i is the cost function C_i "hard-limited" at $g_i(t_{di})$. Γ_i is essentially the finite operating cost associated with task i.

Remark: It might legitimately be argued that to associate a contribution to finite cost after the hard deadline has been missed is inconsistent with the notion of hard deadlines being "absolute" in the sense that missing a hard deadline, by definition, has catastrophic consequences (e.g., an airplane crash). By this argument, $h_i(t) = 0$ for all $t > t_{di}$. However, such an assignment would, while pacifying the purists, lead to unpleasant anomalies, not the least of which is that a very poor system which almost always misses deadlines would exhibit a smaller finite operating cost than a counterpart that almost always fulfills them.

Also, assume that the computer system is modelled as a Markov process. This is clearly possible. The number of states[8] depends on the extent to which the system is capable of graceful degradation. Let B be the set of states where the probability of failure is unity. These states represent the states when the extent of

[8]The word "state" used here has a different connotation from the "state" discussed in the preceding sections. The latter has a control-theoretic meaning. On the other hand, the state of controller computers usually means the number of functioning processors, buses, memories, jobs, etc. However, both forms of usage conform to the essential concept of "state," as a codification of relevant system condition. For this reason, the same word has been used for the two different purposes, following the usual practice. Its interpretation should be made from the context.

hardware/software collapse is so great that there is a zero probability of successful execution of any task in finite time. Let L(t) denote the probability distribution of the operating interval duration between two successive service stages.

Our performance measures are then:[9]

Probability of Dynamic Failure, p_{dyn}: This is the probability that over the operating interval, at least one hard deadline is missed for whatever reason. This probability incorporates within it the probability of *static failure*, which is the probability that so massive a hardware failure has occurred that the system utilization is greater than unity. Static failure probability has erroneously been treated in most of the literature as expressing the total probability of failure. This is most decidedly not the case in real-time systems.

Mean Modified Cost, $M \equiv \int_0^\infty E\{S(t) \mid system\ never\ enters\ state\ set\ B\} dL(t)$

It can be shown that, for physical systems, this integral always exist since the lifetime is always finite with probability 1.

The performance measures can be used to rank rival computer systems and to help design improvements to existing systems *in the context of the control application*. Typically, the probability of dynamic failure is used as a pass/fail test for candidate controllers. This test can be very severe: for example, 10^{-9} is the specification for failure probability adopted by NASA for computer controllers of the next decade handling a 10-hour civilian flight. The mean cost is then employed to rank controllers that have passed the dynamic failure criterion. For fuller details, see [1].

Note that all parameters associated with these performance measures can either be definitively estimated or objectively measured. Also, the measures specifically incorporate the controlled process into a determination of the controller's

[9] *There are other performance measures developed in [1], but not considered here. For our purposes, the measures listed here are sufficient.*

capabilities. This is, as far as we know, a novel approach which ensures that the performance measures are not generally, but instead specifically, indicative of the controller performance in a given application. For this reason, these measures are intrinsically more reliable than others in use.

2.2. Hard Deadlines

Roughly speaking, hard deadlines are deadlines that must be met if catastrophic failure is to be avoided by a critical process. In other words, it is the deadline that, if met by the controller in (correctly) formulating its response, ensures that the system remains in its allowed state-space. Traditionally, it has been assumed by computer engineers that the hard deadlines for each critical controller job are somehow "given". Unfortunately, this presupposes a precise definition of the hard deadline and a means for obtaining it. Neither seems to exist in the literature.

At first glance, it might seem that the hard deadlines can be obtained relatively easily from an analysis based on the state equations of the controlled process. This is the case when individual controller actions are decoupled from each other and the process is simple. For an example in which this is the case, see [1]. However, when there is a considerable coupling between individual controller jobs, i.e., when two or more controller jobs mutually affect each other, or when no closed-form solutions are available for the process state equations, obtaining a hard deadline for each can be difficult. For example, in the aircraft landing problem, the controller has over the twenty seconds or so that it takes to complete the landing, to compute the elevator deflection a number of times (in our example about 330 times). The constraints are on the *final* values (except for the angle of attack), i.e., as long as the aircraft touches down on the runway without over- or under-shoot, with an acceptable velocity and at a proper pitch angle, dynamic failure has not occurred. The problem here is that it is not just a *single* controller action that determines whether catastrophic failure will occur or not; it is the cumulative effect of, in this case, 330 or so distinct controller actions. How then is one to allocate deadlines to the individual actions?

It is clear that we need a more carefully defined framework to handle these problems in a feasible manner. For this it is convenient to represent the controlled process by a state-space model. Let the state of the system at time t be denoted by $\mathbf{x}(t)$. State transitions are characterized by a mapping $\varphi : T \times T \times X \times U \to X$ where $T \subset \mathbf{R}$ represents the time region, $X \subset \mathbf{R}^n$ the state space, and $U \subset \mathbf{R}^m$ the input space; that is,

$$\mathbf{x}(t_1) = \varphi(t_1, t_0, \mathbf{x}(t_0), \mathbf{u}) \tag{4}$$

where $\mathbf{u} \in U$ represents the controls (inputs) applied to the process in the interval $[t_0, t_1)$. Let $\Omega \subset U$ be the admissible input space (i.e. the range of inputs that it is possible to apply), and $X_A \subset X$ the allowed state-space. Then, the hard deadline associated with controller job α triggered at t_0 when the system is in state $\mathbf{x}(t_0)$ is given by

$$\tau_{d\alpha}(\mathbf{x}(t_0)) \equiv \inf_{\mathbf{u} \in \Omega} \sup \{\tau \mid \varphi(t_0+\tau, t_0, \mathbf{x}(t_0), \mathbf{u}) \in X_A\} \tag{5}$$

Thus, for every point in the state space, we have for each critical controller job a corresponding hard deadline.[10]

It should be noted that the calculation in (5) is performed over the entire admissible state and input space and is thus difficult to achieve. One might wish to perform the calculation over only a subset of the admissible input or state space. To allow for this, the notion of *conditional hard deadlines* can be employed. Let us assume that the sets $\omega \subset \Omega$ and $\sigma \subset X_A$ are specified, and also that $\mathbf{x}(t_0) \in \sigma$.

[10] *Notice in this context that it is not possible for the hard deadline as defined above to be negative unless this is the first instance during the current mission that the controller job is being executed. This is because if this were the case, failure would already have occurred on the previous execution of the controller job by definition. Also, the deadline on the first instance of execution of the control function cannot be negative, since that would mean that the controller-process system had been improperly designed.*

The *conditional hard deadline*[11] of job α, denoted by $\tau_{d\alpha|\omega,\sigma}$, is defined as

$$\tau_{d\alpha|\omega,\sigma}(\mathbf{x}(t_0)) \equiv \inf_{\mathbf{u}\in\omega} \sup \{\tau \,|\, \varphi(t_0+\tau,\, t_0,\, \mathbf{x}(t_0),\, \mathbf{u}) \in \sigma\} \tag{5a}$$

For the purposes of the case-study in this report, however, we shall restrict ourselves to the unconditioned hard deadlines.

As it stands, the expression for the hard deadline (and for the conditional deadlines) is not easy to obtain for the entire state-space. In fact, it is almost impossible to obtain in closed form in all but the simpler control systems. We shall later see how to obtain a good approximate expression of $\tau_{d\alpha}$.

Also, if the environment is stochastic in character and not deterministic, the hard deadline is a random variable. Assuming that the environment is stochastically stationary leads to the existence of the distribution function of the hard deadline.

2.3. Allowed State-Space and Its Decomposition

As we said above, it is difficult to determine the hard deadline and the finite cost function as a function of the state over the entire state space. To take our present example of an aircraft, the solution of the state equations is not obtainable in closed form when controller delay is considered. To obtain the functional dependence of the hard deadlines or the finite cost function of each controller job on the current state vector is therefore impossible to do analytically, and prohibitively expensive to do numerically for a large number of sample states.

To get around this problem, we divide the allowed state-space down into *subspaces*. Subspaces are aggregates of states in which the system exhibits roughly the same behavior.[12] In each subspace, each critical controller job has a unique hard

[11] *Unlike the unconditioned hard deadline, it is possible for the conditional hard deadline to be negative since no specific relationship is required between the subsets ω and σ.*

[12] *Even if there do not exist clear boundaries for these subspaces, one can always force the admissible state space to be divided into subspaces so that a sufficient safety margin can be provided. This is a designer's choice for approximation.*

deadline.

Remarks: In some subspaces, a job described in general as "critical" might not be critical in the sense that even if the execution delay associated with it is infinity, catastrophic failure does not occur. That is, the associated hard deadline may be infinity for a particular subspace. What *does* usually happen in these circumstances is that the system moves into a new subspace — or at the least toward the subspace boundary — in which the dangers of catastrophic failure are greater. In this subspace, the requirements on controller delay are more stringent, and there might well be a hard deadline, representing a critical task. Thus a "critical" job need not be truly critical in every subspace, it only has to map into a critical *task* — defined in the sequel — in at least one subspace. Also, subspaces are job-related, i.e. the same allowed state space can divide into a different set of subspaces for each control job.

For convenience, a controller "task" is defined as follows.

Definition: A controller task, often abbreviated to "task", is defined as a controller job operating within a designated subspace of the allowed state space.

Let S_i for $i=0,1,\ldots,s$ be disjoint subspaces of X_A with $X_A = \bigcup_{i=1}^{s} S_i$ and let J denote a controller job. Then, we need the projection: $(J, X_A) \to ((T_0, S_0), (T_1, S_1), \ldots, (T_s, S_s))$ where T_i is the controller task generated by executing J in S_i. With each controller task, we may now define a hard deadline without the coupling problem mentioned above. We denote it by t_{di}^{J} for critical task T_i (for convenience, however, the superscript J will be omitted in the sequel). We will see that a critical job can possibly map into a non-critical task for one or more allowed subspace; it only needs to map into a critical task in *at least one* such subspace to be considered critical.

A. Allowed State-Space

The admissible state-space is the set of states that the system must not leave if catastrophic failure is not to occur. Consider the two sets of states X_A^1 and X_A^2 defined as follows.

(i) X_A^1 is the set of states that the system must reside in if catastrophic failure is not to occur *immediately*. For example, we may define in the aircraft landing problem, a situation in which the aircraft flies upside down as unacceptable to the passengers and as constituting failure. Notice that terminal constraints are not taken into consideration here unless the task in question is executed just prior to mission termination.

(ii) $\mathbf{X}_A^2$ is the set of acceptable states given the terminal constraints, i.e., it is the set of states from which, given the constraints on the control, it becomes possible to satisfy the terminal constraints.

Note that leaving $\mathbf{X}_A^1$ means that no matter how good our subsequent control, failure has occurred.[13] On the other hand, changing the control available can affect the set $\mathbf{X}_A^2$. The admissible state space is then defined as $\mathbf{X}_A \equiv \mathbf{X}_A^1 \cap \mathbf{X}_A^2$.

Obtaining state-space $\mathbf{X}_A^2$ can be difficult in practice. The curse of dimensionality ensures that even systems with four or five state variables make unacceptable demands on computation resources for the accurate determination of the allowed state-space. However, while it can be very difficult to obtain the entire allowed state-space, it is somewhat easier to obtain a reasonably large subset, $\mathbf{X}_A^a \subset \mathbf{X}_A$. By defining this subset as the actual allowed state-space, (i.e., by artificially restricting the range of allowed states), we make a conservative estimate for the allowed state-space. Note that by making a conservative approximation, we err on the side of safety. Also, the information we need about $\mathbf{X}_A$ may be determined to as much precision as we are willing to invest in computing resources.

In what follows, to avoid needless pedantry, we shall refer to the artificially restricted allowed state-space, $\mathbf{X}_A^a$, simply as the "allowed state-space."

B. On Obtaining the Subspaces

While the methods used to isolate the subspaces for each particular control application will probably be different, the basic approach is much the same in all cases. Let $\mathbf{N}(\mathbf{x})$ represent a neighborhood of $\mathbf{x} \in \mathbf{X}_A$, $C_i^{\mathbf{x}}(\xi)$ and $t_{di}(\mathbf{x})$ denote respectively the cost function and the hard deadline associated with T_i where ξ represents

[13] Strictly speaking, of course, there can be no subsequent control since by leaving $\mathbf{X}_A^1$ the system has failed catastrophically before the next control could be implemented.

the controller response time, and $d: \mathbf{X} \times \mathbf{X} \rightarrow \mathbf{R}$ is a metric function. Then the subspaces $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_s$ of $\mathbf{X}_A$ can be obtained by the following steps.

S1. Choose a set of l points, $\mathbf{p}_i \in \mathbf{X}_A,\ i=1,2,\ldots,l$.

S2. Construct a neighborhood around each of these points, $\mathbf{N}(\mathbf{p}_i)$ for $\mathbf{p}_i$, such that

(i) All $\mathbf{N}(\mathbf{p}_i)$'s are disjoint and $\mathbf{X}_A \subset \bigcup_i \mathbf{N}(\mathbf{p}_i)$.

(ii) $|t_{di}(\mathbf{x}) - t_{di}(\mathbf{p}_i)| \leq K_1$ for all $\mathbf{x} \in \mathbf{N}(\mathbf{p}_i)$ and for some $K_1 > 0$.

(iii) $d(C_i^{\mathbf{x}}(\xi), C_i^{\mathbf{p}_i}(\xi) \leq K_2 f_i(\xi)$ for all $\mathbf{x} \in \mathbf{N}(\mathbf{p}_i)$ and for some monotonically non-decreasing positive function $f_i(\xi)$ and $K_2 > 0$.

S3. if $d(C_i^{\mathbf{p}_i}, C_{i+1}^{\mathbf{p}_i+1}) \leq K_3 f'(\xi)$ for $i=1,2,\ldots,(l-1)$, some $K_3 > 0$, and monotonically non-decreasing positive function $f'(\xi)$ then merge $\mathbf{N}(\mathbf{p}_i)$ and $\mathbf{N}(\mathbf{p}_{i+1})$ forming subspaces $\mathbf{S} = (\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_s)$.

S4. if $\mathbf{S}$ satisfies all the requirements of the application jobs then successful decomposition else choose a different set of points and go to S2.

The job of dividing $\mathbf{X}_A$ into $\mathbf{S} = (\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_s)$ is sometimes made easy by the existence of natural cleavages in the state-space, when the latter is viewed as an influence on system behavior. In most cases, however, such conveniences do not exist, and artificial means must be found. The problem then becomes one of finding discrete subdivisions of a continuum.

The method we employ is to quantize the state continuum in much the same way as analog signals are quantized into digital ones. Intervals of hard deadlines and expected operating cost (i.e. the mean of the cost function conditioned on the controller delay time, and using the distribution of the latter) are defined. Then, points are allocated to subspaces corresponding to these intervals. To take a concrete example, consider a state-space $\mathbf{X} \subset \mathbf{R}^n$ that is to be subdivided on the basis of the hard deadlines. The first step is to define a quantization for the hard deadlines. Let this be Δ. Then, define subspace S_i as containing all states in which the hard deadline lies in the interval $[(i-1)\Delta, i\Delta)$. Alternatively, one might define a sequence of numbers Δ_1, Δ_2, ... such that the subspaces were defined by intervals with the Δ's as their end-points. This would correspond to quantizing with variable step sizes. The subspace in which the job under consideration maps into a non-critical task is a special case and is denoted by S_0.

Subspaces can also be defined based on a quantization of the expected operating cost or on both the operating cost and the hard deadlines. We provide an example of subdivision by hard deadlines in Section 4.

The size of each subspace will depend on the process state equations, the environment, and how much computing effort it is judged to be worth spending on obtaining the subspaces. Naturally, all other things being equal, the smaller a subspace the greater the accuracy of the inherent approximation.[14]

In the rest of the report, to illustrate the derivation of the performance measures, we carry out their evaluation when the controlled process is an aircraft in the phase of landing. Also, an optimal checkpointing is considered for the design of a reliable controller.

3. The CONTROLLED PROCESS

The controlled process is an aircraft, in the phase of landing. The model and the optimal control solution used are due to Ellert and Merriam [4].

The aircraft dynamics are characterized by the equations:

$$\dot{x}_1(t) = b_{11}x_1(t)+b_{12}x_2(t)+b_{13}x_3(t)+c_{11}m_1(t,\xi) \quad (6a)$$

$$\dot{x}_2(t) = x_1(t) \quad (6b)$$

[14] *The error that ensues as a result of quantization of the state space can be estimated in the same way that quantization error is estimated in signal processing theory.*

$$\dot{x}_3(t) = b_{32}x_2(t)+b_{33}x_3(t) \tag{6c}$$

$$\dot{x}_4(t) = x_3(t) \tag{6d}$$

where x_2 is the pitch angle, x_1 the pitch angle rate, x_3 the altitude rate, and x_4 the altitude. m_1 denotes the elevator deflection, which is the sole control employed. The constants b_{ij} and c_{11} are given in Table 1. Recall that ξ denotes controller response time.

The phase of landing takes about 20 seconds. Initially, the aircraft is at an altitude of 100 feet, travelling at a horizontal speed of 256 feet/sec. This latter velocity is assumed to be held constant over the entire landing interval. The rate of ascent at the beginning of this phase is -20 feet/sec. The pitch angle is ideally to be held constant at 2°. Also, the motion of the elevator is restricted by mechanical stops. It is constrained to be between -35° and 15°. For linear operation, the elevator may not operate against the elevator stops for nonzero periods of time during this phase. Saturation effects are not considered. Also not considered are wind gusts and other random environmental effects.

The constraints are as follows: The pitch angle must lie between 0° and 10° to avoid landing on the nose-wheel or on the tail, and the angle of attack (see Figure 1) must be held to less than 18° to avoid stalling. The vertical speed with which the aircraft touches down must be less than around 2 feet/sec so that the undercarriage can withstand the force of landing.

The desired altitude trajectory is given by

$$h_d(t) = \begin{cases} 100e^{-t/5} & 0 \le t \le 15 \\ 20-t & 15 \le t \le 20 \end{cases} \tag{7}$$

while the desired rate of ascent is

$$\dot{h}_d(t) = \begin{cases} -20e^{-t/5} & 0 \le t \le 15 \\ -1 & 15 \le t \le 20 \end{cases} \tag{8}$$

The desired pitch angle is 2° and the desired pitch angle rate is 0° per sec.

The performance index (for the aircraft) chosen by Ellert and Merriam and suitably adapted here to take account of the nonzero controller response time ξ is given by

$$\Theta(\xi) = \int_{t_0}^{t_f} e_m(t,\xi)dt \tag{9}$$

where t represents time, and $[t_0, t_f]$ is the interval under consideration, and where

$$e_m(t,\xi) = \varphi_h(t)[h_d(t)-x_4(t)]^2+\varphi_{\dot{h}}(t)[\dot{h}_d(t)-x_3(t)]^2+\varphi_\vartheta(t)[x_{2d}(t)-x_2(t)]^2 + \varphi_{\dot{\vartheta}}(t)\,[x_{1d}(t)-x_1(t)]^2+[m_1(t,\xi)]^2.$$

where the d-subscripts denote the desired (i.e. ideal) trajectory. To ensure that the touch-down conditions are met, the weights φ must be impulse weighted. Thus we define:

$$\varphi_h(t) = \varphi_4(t) + \varphi_{4,t_f}\delta(20-t) \tag{10a}$$

$$\varphi_{\dot{h}}(t) = \varphi_3(t) + \varphi_{3,t_f}\delta(20-t) \tag{10b}$$

$$\varphi_\vartheta(t) = \varphi_{2,t_f}(t)\delta(20-t) \tag{10c}$$

$$\varphi_{\dot{\vartheta}}(t) = \varphi_1(t) \tag{10d}$$

where the functions φ must be given suitable values, and δ denotes the Dirac-delta function. The values of the φ are given based on a study of the trajectory that results. The chosen values are listed in Table 2.

The control law for the elevator deflection is given by:

$$m_1(t,\xi) = \omega_s{}^2K_s T_s[k_{11}(t-\xi)-k_{11}(t-\xi)x_1(t)-k_{12}(t-\xi)x_2(t-\xi) -k_{13}(t-\xi)x_3(t-\xi)-k_{14}(t-\xi)x_4(t-\xi)]$$

where the aircraft parameters are given by: $K_s = -0.95 \text{ sec}^{-1}$, $T_s = 2.5$ sec, $\omega_s = 1\ \textit{radian}\ \text{sec}^{-1}$ and the constants k are the feedback parameters derived (as shown in [4]) by solving the Riccatian differential equations that result upon minimizing the process performance index. For these differential equations we refer the reader to [4].

4. DERIVATION OF PERFORMANCE MEASURES

We consider here only one controller task: that of computing the elevator deflection so as to follow the desired landing trajectory. The inputs for the controller here are the sensed values of the four states.

We seek the following information. As the controller delay increases, how much extra overhead is added to the performance index? Also, it is intuitively obvious that too great a delay will lead to a violation of the terminal (landing) conditions, thus resulting in a plane crash. This corresponds to dynamic failure, and we are naturally interested in determining the range of controller delays that permit a safe landing.

Consider first a formal treatment of the problem. The control problem is of the linear feedback form. The state equations can be expressed as:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

where the symbols have their traditional meanings. Define the feedback matrix by $\Xi(t)$. Then, clearly,

$$\mathbf{u}(t) = \Xi(t-\xi)\mathbf{x}(t-\xi)$$

For a small controller delay (i.e., a small ξ), the above can be expanded in a Taylor series and the terms of second order and higher discarded for a linear approximation. By carrying out the obvious mathematical steps, we arrive at the equation:

$$\dot{\mathbf{x}}(t) = \mathbf{E}(t,\xi)\mathbf{x}(t) + \beta(\xi)$$

as representing the behavior of the system (assuming the given initial conditions). For further details, see Figure 5.

Given a closed-form expression for the $k_{ij}(t)$ that appear in $\mathbf{E}(t,\xi)$, we could then proceed to study the characteristics of the system as a function of the matrix $\mathbf{E}$. However, in the absence of such closed formulations for the k_{ij}, we must take recourse to the less elegant medium of numerical solution.

The procedures we follow for obtaining the numerical solution are as follows. First, the feedback values are computed by solving the feedback differential equations that define the k_{ij}. These are not affected by the magnitude of the controller delay. Then, the state equations are solved as simultaneous differential equations. These are used to check that the terminal constraints have been satisfied, and in the event that they are the performance functional is evaluated. This procedure must be repeated for each new subspace. Since the environment is deterministic in this case (no wind gusts or other random disturbances are permitted in the model(6)), the hard deadline associated with each process subspace is a constant and not a random variable.

The trajectory followed by the aircraft when the delay is less than about 60 milli-seconds follows the optimal trajectory closely although the elevator deflections required would be intuitively assumed to increase as the delay increases. Also, the susceptibility of the process to failure in the presence of incorrect or no input is expected to rise with the introduction of random environmental effects.

The control that is required for various values of controller delay is shown in Figure 6. Due to the absence of any random effects, elevator deflections for all the delays considered tend to the same value as the end of the landing phase (20 seconds) is approached, although much larger controls are needed initially. In the presence of random effects, the divergence between controls needed in the low and the high delay values of controller delay is even more marked. We present an example of this in Figure 7. The random effect considered here is the elevator being stuck at -35° for 60 milli-seconds 8 seconds into the landing phase due to a faulty controller order. The controlled process is assumed in Figure 8 to be in the subspace in which the landing job maps into a non-critical process (defined in the sequel as S_0). The diagrams speak for themselves. We shall show later that this demand on control is fully represented by the nature of the derived cost function. Also, above a certain threshold value for controller delay, we would expect the system to become unstable. This is indeed the case in the present problem, although this point occurs beyond a delay of 60 milli-seconds for all points in the allowed state space (obtained in the next section), which cannot by definition occur here.

4.1. Allowed State Space

In this subsection, we derive the allowed state space of the aircraft system. To do so, note that in Ellert and Merriam's model, $\mathbf{X}_A^1$ does not exist. The reason is that the state equations do not take into account the angle of attack. In the idealized model we are considering, it is implicitly assumed that the constraint on the angle of attack is always honored, so that the only constraints to be considered are the terminal constraints.

The terminal constraints have been given earlier but are repeated here for convenience. The touchdown speed must be less than 2 feet/sec in the vertical direction, and the pitch angle at touchdown must lie between $0°$ and $10°$. To avoid overshooting the runway, touchdown must occur at between 4864 and 5120 feet in the horizontal direction from the moment the landing phase begins. The horizontal velocity is assumed to be kept constant throughout the landing phase at 256 feet/sec.[15] Thus, touchdown should occur between 19 and 20 seconds after the descent phase begins.[16] The only control is the elevator deflection which must be kept between $-35°$ and $15°$.

The set of allowed states is generally found by solving the differential equations for the system backwards from the point of landing. However, this can be computationally expensive, so we follow a cheaper alternative. The initial conditions of the process as it enters the landing stage are known. Also known is that the controller is triggered every 60 milli-seconds. It is assumed that the computations take a minimum of 20 milli-seconds to complete. Using these data, it becomes possible to determine that portion of the allowed state-space that the controlled process is ever

[15] *We do not consider here how that is to be done; in practice this will constitute a second controller job. We do not treat this here.*

[16] *This makes time an "implicit" state variable.*

likely to enter to a good approximation. In Figure 8, we plot the range of allowed state values that we obtain. As indeed it should be, the allowed state-space is a function of time.

4.2. Designation of Subspaces

We subdivide the allowed state-space found above using the method described in Section 2. The criterion used is the hard deadline, since the finite cost function (derived in the next subsection) is found not to vary greatly within the whole of the admissible state-space. The value of Δ chosen is 60 milli-seconds. In other words, we wish to consider only the case where a trigger is "missed."

The allowed state-space in Figure 8 is subdivided into two subspaces, S_0 and S_1. These correspond to the deadline intervals $[60, 120)$ and $[120, \infty)$. S_0 is the non-critical region corresponding to the $[120, \infty)$ interval. Here, even if the controller exhibits any of the abnormalities considered in the Introduction, the airplane will not crash. In other words, if the controllers orders an incorrect output, exhibits an abnormal execution delay or simply provides no output at all before the following trigger, the process will still survive at the end of the current inter-trigger interval if, at the beginning of that interval, it was in S_0.

On the other hand, if the process is in S_1 at the beginning of a inter-trigger interval, it may safely endure a delay in controller response. However, if the controller behaves abnormally in either providing no output at all for the current trigger cycle or in ordering an incorrect output, there is a positive probability of a air crash.

Notice that we explicitly consider only missing a single trigger, not the case when two or more triggers might be missed in sequence. This is because dynamic failure is treated here as a function of the state at the moment of triggering. If two successive triggers are missed, for example, we have to consider two distinct states, namely the states the process is in at the moment of those respective triggers. To speak of deadline intervals beyond 120 milli-seconds is therefore meaningless in this case since the triggers occur once every 60 milli-seconds. This is why the second deadline interval considered is $[120, \infty)$, not $[120,180)$.

The hard deadline may conservatively be assumed to be 60 milli-seconds in S_1. By definition it is infinity in S_0.

4.3. Finite Cost Functions

As indicated in the preceding section, the finite cost does not vary greatly within the entire allowed state-space. It is therefore sufficient to find a single cost function for S_0 or S_1.

The determination of the cost function is carried out as a direct application of its definition. That is, the process differential equations are solved with varying values of ξ. The value of ξ cannot be greater than the inter-trigger interval of 60 milli-seconds since, by assumption, no job pipelining is allowed and the controller terminates any execution in progress upon receiving a trigger. The finite cost function is defined as[17]

$$g(\xi) = \Psi(\xi) - \Psi(0) \tag{12}$$

This function is found by computation to be approximately the same over the entire allowed state-space as defined in Figure 8.

In Figure 9, the finite cost function is plotted. The costs are in arbitrary units. Bear in mind that these measures are the result of an idealized model. We have, for example, ignored the effects of wind gusts and other random effects of the environment. When these are taken into account, the demands on controller speed get even greater, i.e. the costs increase.

[17] Recall that $\Psi(\xi)$ represents the contribution to the performance functional by a version that takes ξ units of time to compute. Since there is only one controller job under consideration, the subscript on Ψ has been suppressed.

The reader should compare the nature of the cost function with the plots showing elevator deflection in Figure 6, and notice the correlation between the marginal increase in cost with increased execution delay and the marginal increase in control needed, also as a function of the execution delay.

5. APPLICATION EXAMPLE FOR CONTROLLER DESIGN: CHECKPOINTING

With stringent requirements on the reliability of any computer controlling a highly critical system, it becomes necessary to obtain mechanisms to identify and correct controller errors. Two characteristics must be exhibited by any such mechanism: a high probability that errors once existing are caught in time, and a fast means for recovering from the error. In this section, we deal with the latter.

One common recovery method is the use of the *recovery block* or *recovery region* which establishes checkpoints and saves the current job states during normal execution. When an error is detected, the system rolls back to the state saved at the previous checkpoint and the affected task-version is resumed. Clearly, checkpoints can enhance the reliability of execution and reduce the recovery overhead. They can also, however, lead to increased controller overhead since the insertion of checkpoints increases controller delay. It is therefore important to carefully check during design if any overall benefits accrue from the installation of checkpoints, and not to include them anyway through an ad-hoc design procedure. Checkpoints should be regarded as useful supplementary devices to enhance reliability in certain cases, not as a panacea for reliability problems. It is the purpose of this section to demonstrate the use of the performance measures described above in a study of the effectiveness of checkpointing. Specifically, we shall in this section consider (a) whether checkpointing is indicated in our aircraft landing problem, and (b) if so, what the optimal number of checkpoints is.

Several methods for analyzing the rollback recovery system have been proposed [8 - 12]. They in general compute the optimum inter-checkpoint interval for minimum total execution time. In [12], a complete expression for the characteristic function of total execution time is given which takes into account imperfections in the checkpoints, the occurrence of error during recovery, and multi-step rollback. However, when mean time between failure (MTBF) is many orders of magnitude larger than both the nominal task execution time (ξ) and the duration of the phase (t_p), the model for solving the probability distribution of the task execution time and the probability of dynamic failure with checkpoints can be simplified.

Let $MTBF=10^4$ hours. This is a reasonable assumption given contemporary processors ([7], page 161). Then in the landing phase, the ratio of ξ to $MTBF$ is of the order of 10^{-10}. It is therefore an acceptable approximation to assume in computing the execution time that no further errors occur during error-recovery.

Let the occurrence of error be a Poisson process with rate $\lambda=1/MTBF$. Let t_b, t_s, t_{ov} denote the time needed to set up rollback, restart, and checkpoint. Also, we assume that the saved state may be contaminated with probability p_s which means the system can be recovered using rollback with probability $p_b=1-p_s$, and has to restart with probability p_s. Thus, we have the total execution time of one version, ξ_t,

$$\xi_t = \xi + nt_{ov} + t_{rec} \tag{13}$$

where n is the number of checkpoints inserted and t_{rec} is the time overhead used for recovery. t_{rec} is a random variable which depends on the probability of failure, p_b, and p_s.

$$t_{rec} = \begin{cases} 0 & \text{if no error occurs} \\ t_b + t_{roll} & \text{if error occurs and the version is recovered by rollback} \\ t_s + t_{start} & \text{if error occurs and the version restarts} \end{cases} \tag{14}$$

where t_{roll} and t_{start} are the computation undone because of rollback and restart, respectively. Let t_{inv} be the interval between checkpoints and equal to $\xi/(n+1)$. The density function of t_{roll} and t_{start} are given by $f_{roll}(t) = \lambda e^{-\lambda t}/(1 - e^{-\lambda t_{inv}})$ for $t \in [0, t_{inv}]$ and $f_{start}(t) = \lambda e^{-\lambda t}/(1 - e^{-\lambda \xi})$ for $t \in [0, \xi]$, respectively.

Let the density of total execution time solved from above equation be $f_{\xi}(t)$. Then the mean execution cost and the probability of dynamic failure are given by

$$COST = \sum_{j=1}^{q_i} \int_0^{\infty} f_{\xi}(t) h_{ij}(t) dt \tag{15}$$

$$p_{dyn} = 1.0 - \prod_{j=1}^{q_i} (1.0 - \int_{t_{di}^j}^{\infty} f_{\xi}(t) dt - p_{st}^j) \tag{16}$$

where q_i is the number of versions executed for T_i during a phase, h_{ij} the cost function for executing the j-th version of T_i, t_{di}^j the deadline associated with the j-th version of T_i, and p_{st}^j is the probability of static failure for the j-th version of T_i during ξ which can be regarded as the probability of resource exhaustion, unsuccessful recovery, successive failures during recovery, etc.

Using the cost function and hard deadlines given in the above section, and assuming that p_{st} is the probability of having an error during recovery, the improvement in the probability of dynamic failure, p_{dyn}, upon insertion of checkpoints are shown in Table 3.[18] The probability of dynamic failure does indeed decrease as more checkpoints are inserted. Unfortunately, the mean execution cost increases in as this is done. Through the cost functions it is possible to express the precise extent of this cost increase, and decisions about tradeoffs can be made.

When the nominal execution time is 20 milli-seconds as assumed in the rest of this report, all that checkpoints do is to increase the overhead, i.e. the mean finite cost. No discernible drop exists in the probability of dynamic failure when checkpoints are added. The marginal gain in reliability for any payment in finite operating cost is therefore about zero. This was only to be expected since the nominal execution time is one-third that of the hard deadline and the probability of dynamic failure

[18] *Since the landing job is noncritical in S_0, the issue of checkpointing does not arise there. All re-*

without checkpoints was vanishingly small.

However, as the nominal execution time increases (the hard deadline being assumed to be its S_1 value of 60 milli-seconds), the benefits of checkpointing emerge. This is made clear through the fourth column in Table 3 where we present the marginal tradeoff ratio between the benefits gained in the form of improved reliability and the loss in the form of increased controller overhead. As is evident from this, for 20 milli-seconds nominal execution time, the optimal number of checkpoints is zero. For 30 milli-seconds, there is something to be gained in reliability by putting in one checkpoint, for 40 milli-seconds, there is a gain to be had on adding up to two checkpoints although the marginal gain falls off sharply after the first. For a nominal execution time of 50 milli-seconds, the benefits continue rather steadily until four checkpoints have been added. The fifth checkpoint provides some noticeable improvement in reliability, although the marginal gain is distinctly smaller than for the first four. The recommendations for design are now rather clear: use no checkpoints if the nominal execution time is 20 milli-seconds, and use Table 3 to decide on the optimal number of checkpoints for the other cases.

6. DISCUSSION

In this report, we have presented a case-study of the determination of performance measures introduced in [1], and considered an important application in controller design.

Central to our report is the idea that it is possible to objectively quantify the performance of a controller. Owing to this objectivity, there are many possible extensions to this work. One extension, presently under study, is the issue of distributed control, and the cost of transmitting global status information to all the local controllers. For guaranteed reliability, the local controllers require a complete knowledge of

marks in this section are therefore concerned with the characteristics of the process in S_1.

the global state of the system. This, however, has a cost in terms of the extra response times exhibited by the overall controller. If the local controllers have less than complete information, their actions cannot be optimal and might even be incorrect. However, the response time of the controller could be significantly reduced, with errors occurring rarely enough to make that an improvement. Readers will recognize this formulation as an example of the application of Markov decision theory with costly information with the cost functions for the controller jobs now providing the cost of status information.

Other applications include the quasi-optimal allocation and reallocation of control jobs to different processors in a multiprocessor controller, the dynamic control of queues in controllers, and the objective ranking of rival computer systems as controllers of any specific process.

ACKNOWLEDGMENT

The authors are indebted to Rick Butler and Milton Holt at NASA Langely Research Center for their technical and financial assistance.

REFERENCES

[1] C. M. Krishna and K. G. Shin, "Performance Measures for Multiprocessor Controllers," *Performance '83: Ninth Int'l Symp. Comp. Perf., Meas., and Eval.*, pp. 229-250.

[2] D. E. Kirk, *Optimal Control Theory*, Prentice Hall, Englewood Cliffs, NJ, 1970.

[3] A. P. Sage, *Optimum Systems Control*, Prentice Hall, Englewood Cliffs, NJ, 1970.

[4] F. J. Ellert and C. W. Merriam, "Synthesis of Feedback Controls Using Optimization Theory -- An Example," *IEEE Trans. Auto. Control*, Vol. AC-8, No. 4 April 1963, pp. 89-103.

[5] L. T. Wu, "Models for Evaluating The Performability of Degradable Computing Systems," *Computing Research Laboratory Report CRL-TR-7-82*, The University of Michigan, Ann Arbor, June 1982.

[6] J. F. Meyer, *et. al*, "Performability Evaluation of the SIFT Computer," *IEEE Trans. Comput.*, Vol. C-29, No. 6, pp. 501 - 509, June 1980.

[7] J. H. Wensley, *et. al*, "Design Study of Software-Implemented Fault-Tolerance Computer," *NASA Contractor Report 3011*, 1982.

[8] K. M. Chandy, J. C. Browne, C. W. Dissly and W. R. Uhrig, "Analytic Models for Rollback and Recovery Strategies in Data Base Systems," *IEEE Trans. Softw. Engg.*, Vol. SE-1, No. 1, March 1975, pp. 100-110.

[9] K. M. Chandy and C. V. Ramamoorthy, "Rollback and Recovery Strategies for Computer Programs," *IEEE Trans. Comput.*, Vol. C-21, No. 6, June 1972, pp. 546-556.

[10] E. Gelembe and D. Derochette, "Performance of Rollback Recovery Systems under Intermittent Failures," *Comm. of the ACM*, Vol. 21, No. 6, June 1978, pp. 493-499.

[11] J. W. Young, "A First Order Approximation to the Optimum Checkpoint Interval," *Comm. of the ACM*, Vol. 17, No. 9, Sept. 1974, pp. 530-531.

[12] Y.-H. Lee and K. G. Shin, "Design and Evaluation of a Fault-Tolerant Multiprocessor Using Hardware Recovery Blocks," *Computing Research Laboratory Report CRL-TR-6-82*, The University of Michigan, Ann Arbor, August 1982.

[13] K. G. Shin and C. M. Krishna, "A Distributed Microprocessor System for Controlling and Managing Military Aircraft," *Proc. Distributed Data Acquisition, Computing and Control Symp.*, pp. 156-166, Miami, FL, December 1980.

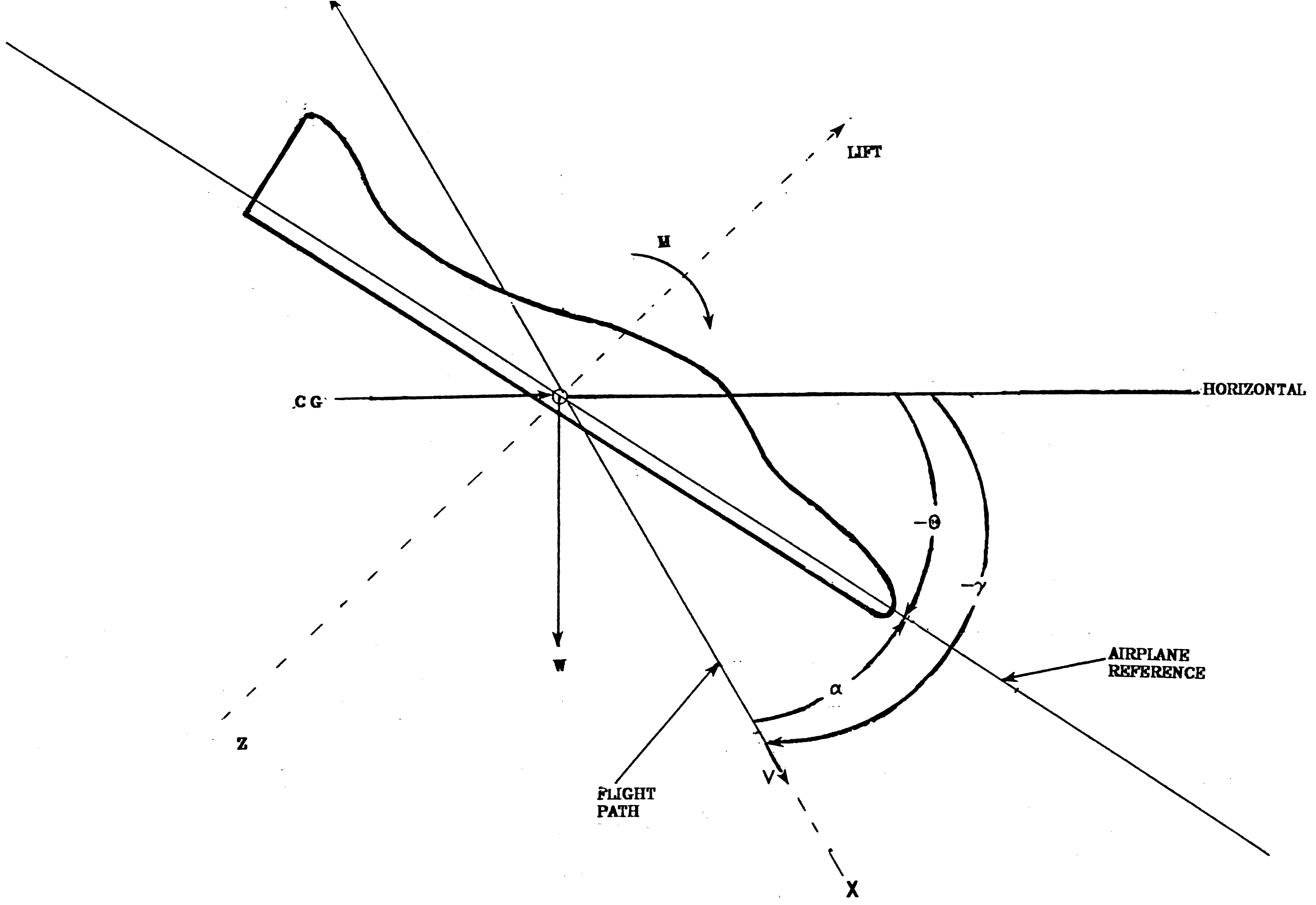


Figure 1. Definition of aircraft angles (from [4])

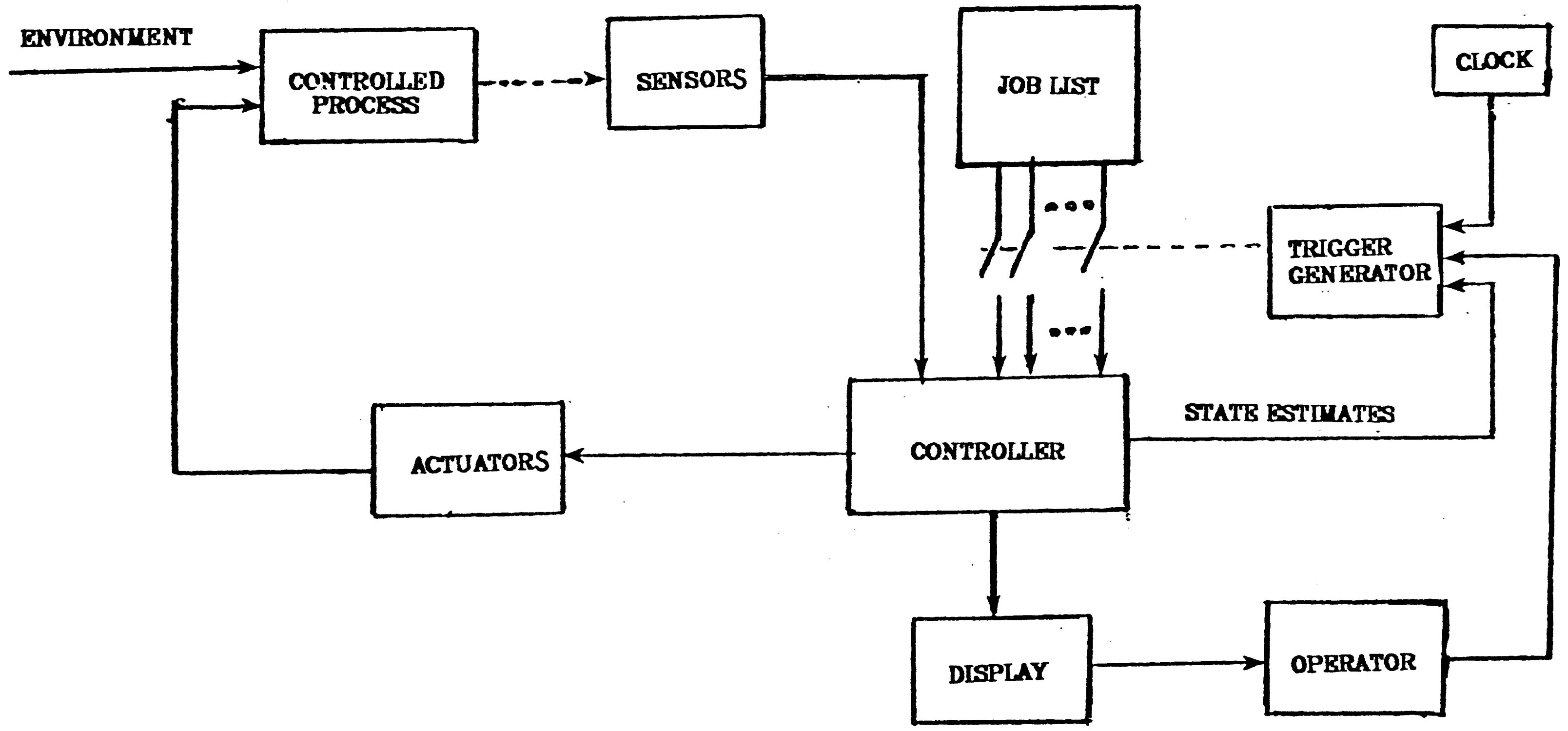


Figure 2. A typical real-time control system

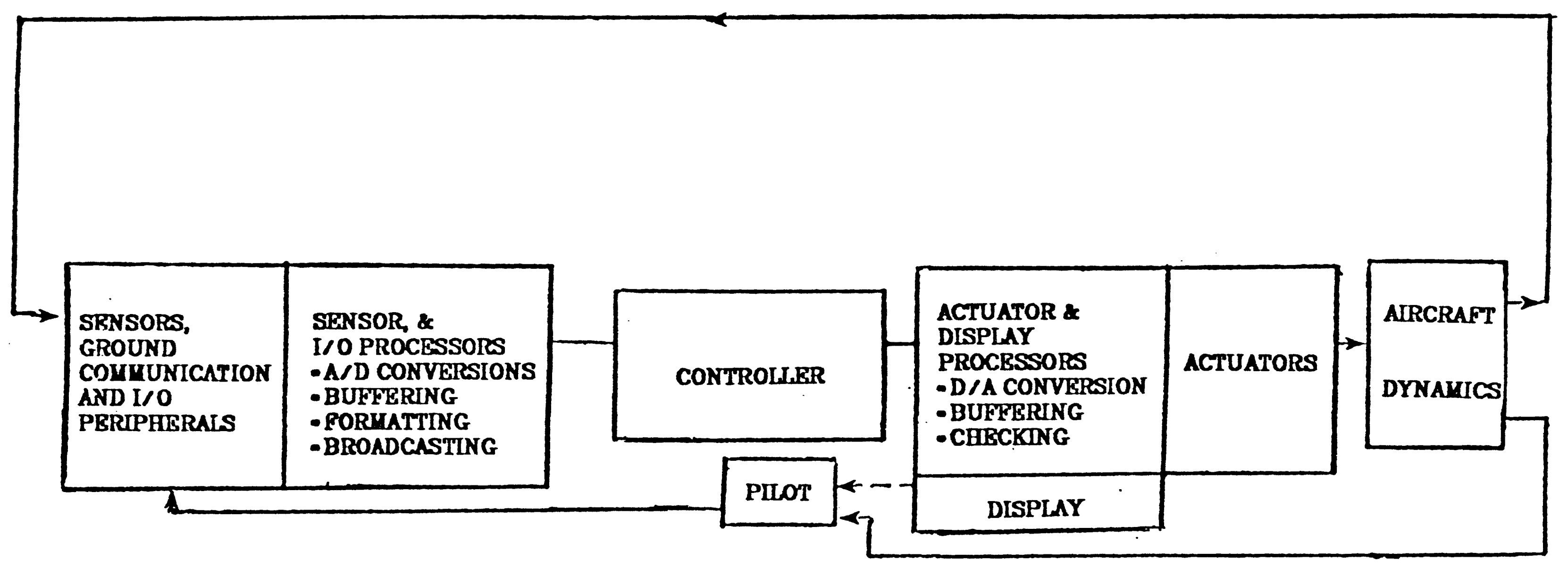
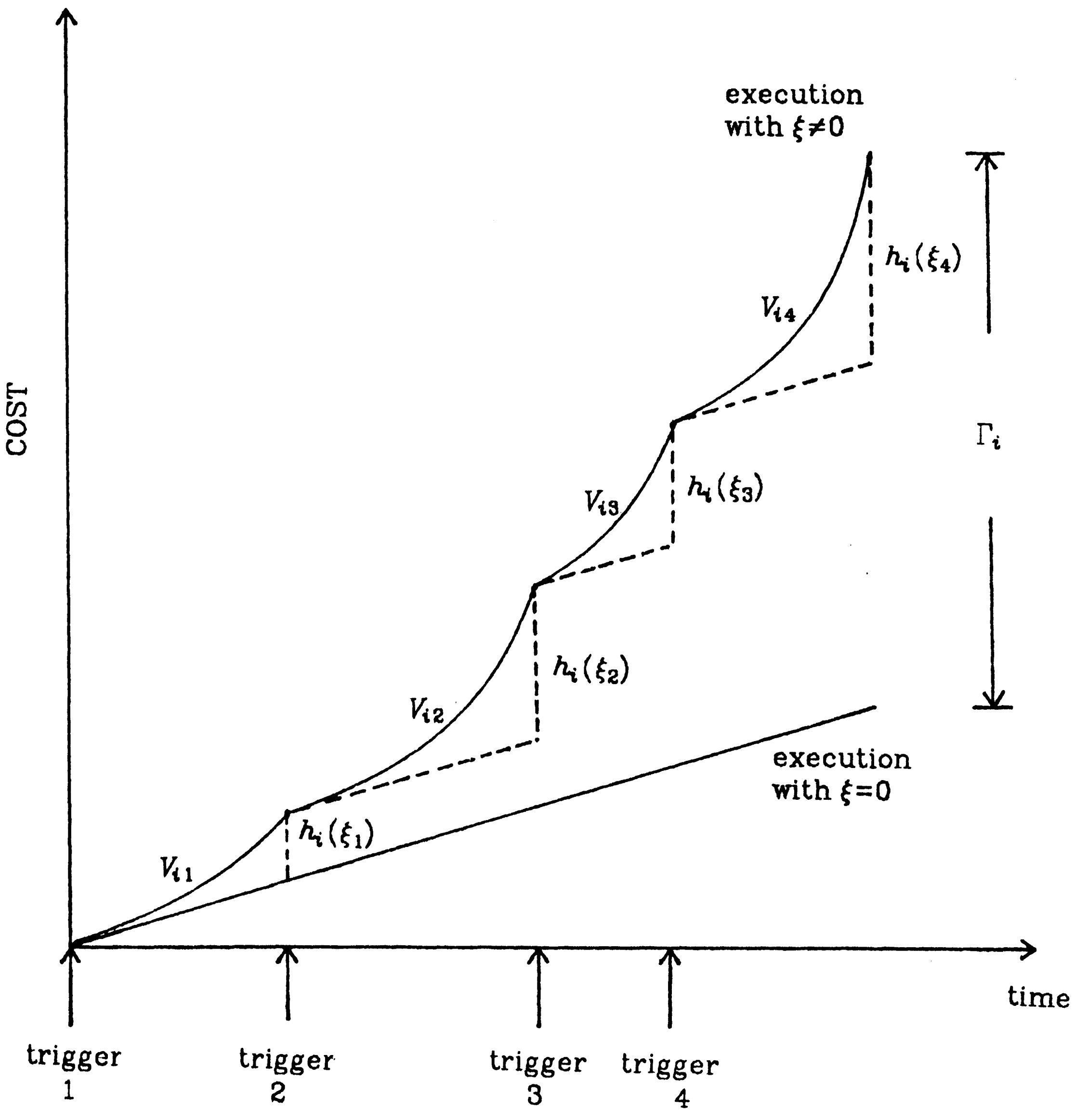


Figure 3. Aircraft control system schematic



$\xi_j = RESP(i_j)$ *for* $j = 1,2,3,4.$

Figure 4. Illustration of Cost Functions.

$$\mathbf{E}(t,\xi) = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & b_{32} & b_{33} \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$a_{11} = [1-c_{11}^{2}k_{11}(t)\xi]^{-1}\,[b_{11}-k_{11}(t)c_{11}^{2}-c_{11}^{2}\xi\{\varphi_{\dot{\vartheta}}(t)+2b_{11}k_{11}(t)+k_{12}(t)-c_{11}^{2}k_{11}^{2}(t)\}]$$

$$a_{12} = [1-c_{11}^{2}k_{11}(t)\xi]^{-1}\,[b_{12}-c_{11}^{2}k_{12}(t)-c_{11}^{2}\xi\{b_{11}k_{12}(t)+b_{12}k_{11}(t)+k_{22}(t)-c_{11}^{2}k_{11}^{2}(t)\}]$$

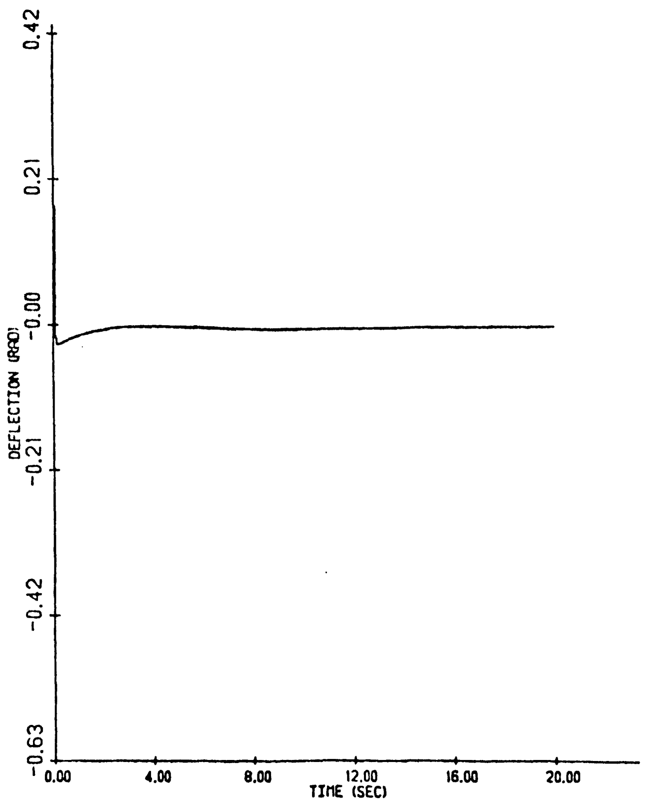
$$a_{13} = [1-c_{11}^{2}k_{11}(t)\xi]^{-1}\,[b_{13}-c_{11}^{2}k_{13}(t)+b_{13}k_{11}(t)+k_{23}(t)-c_{11}^{2}k_{11}(t)k_{13}(t)\}]$$

$$a_{14} = [1-c_{11}^{2}k_{11}(t)\xi]^{-1}\,[-k_{14}(t)-b_{11}k_{14}(t)\xi-k_{24}(t)\xi+c_{11}^{2}k_{11}(t)k_{14}(t)\xi]$$

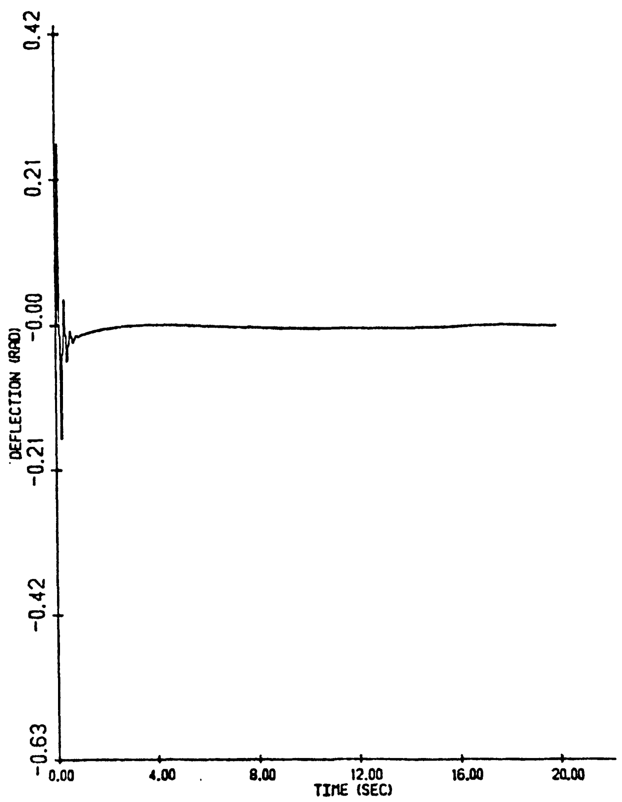
When the execution delay is ξ, the approximate state equations are

$$\dot{\mathbf{x}}(t) = \mathbf{E}(t,\xi)\mathbf{x}(t) + \begin{bmatrix} c_{11}^{2}[k_{1}(t)+\xi\{\varphi_{\dot{\vartheta}}(t)\vartheta_{\dot{d}}(t)+b_{11}k_{1}(t)+k_{2}(t)-c_{11}^{2}k_{1}(t)k_{11}(t)\}] \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

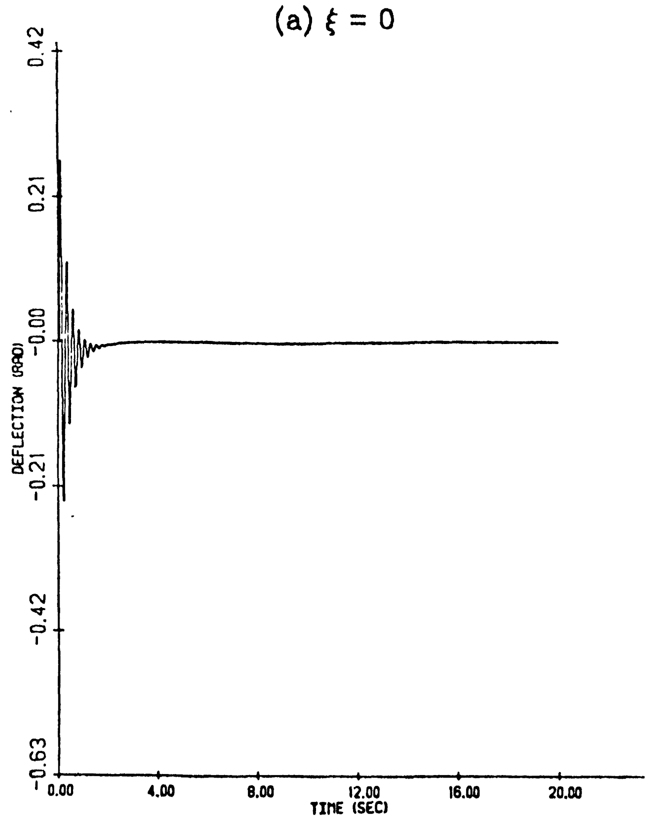
Figure 5. The Approximate State Equations



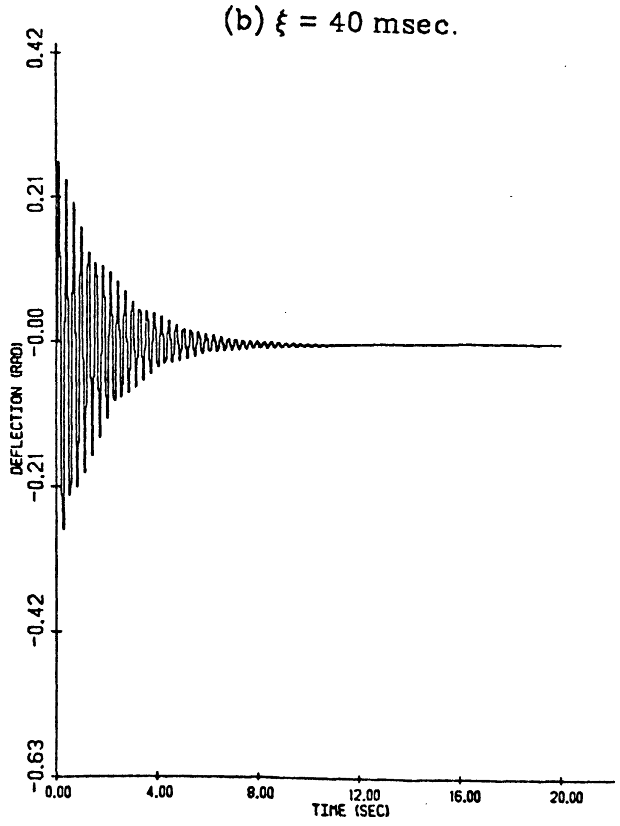
(a) $\xi = 0$



(b) $\xi = 40$ msec.

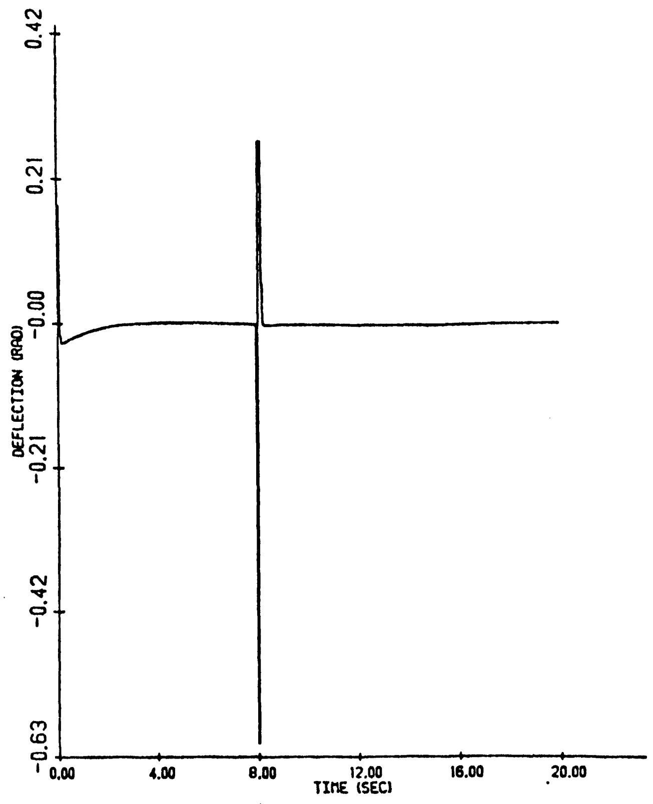


(c) $\xi = 50$ msec.

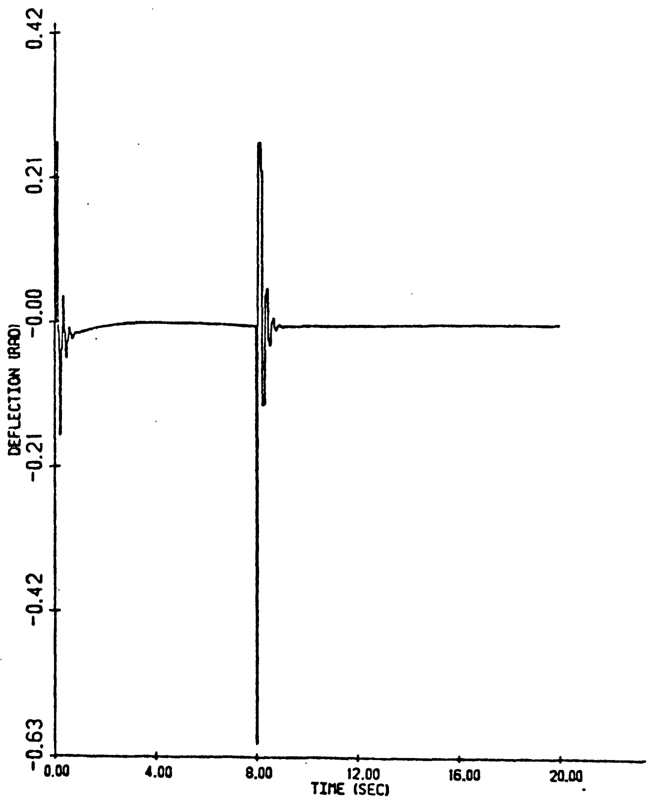


(d) $\xi = 60$ msec.

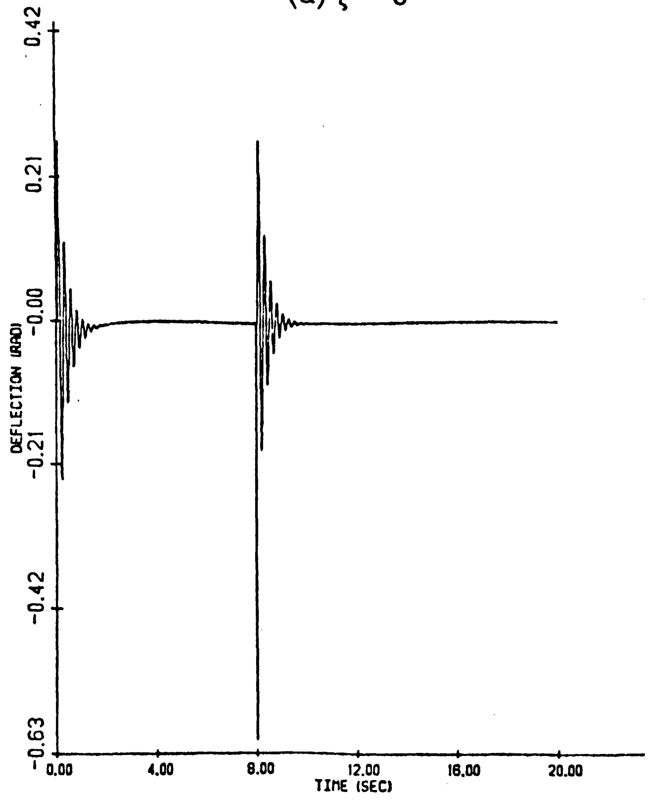
Figure 6. Elevator Deflection.



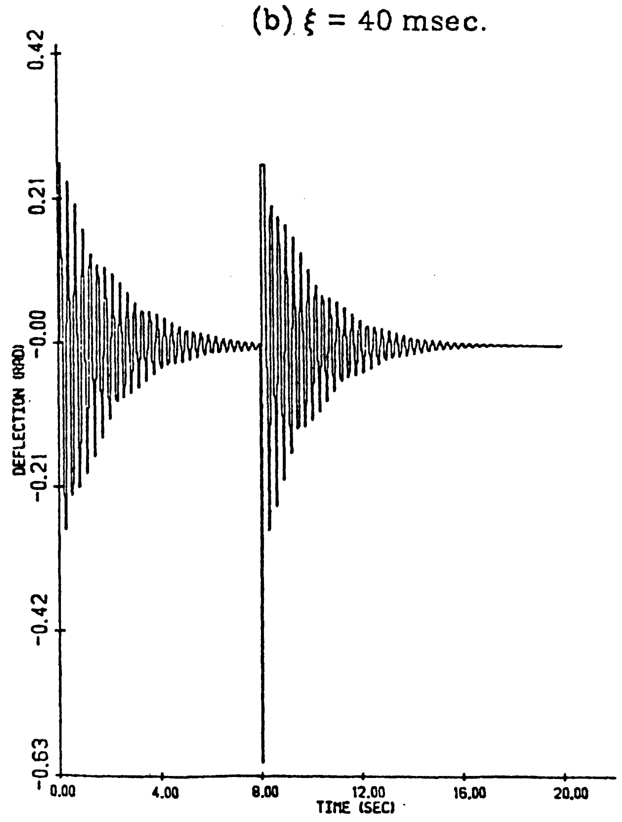
(a) $\xi = 0$



(b) ξ = 40 msec.



(c) ξ = 50 msec.



(d) ξ = 60 msec.

Figure 7. Elevator Deflection with Abnormality.

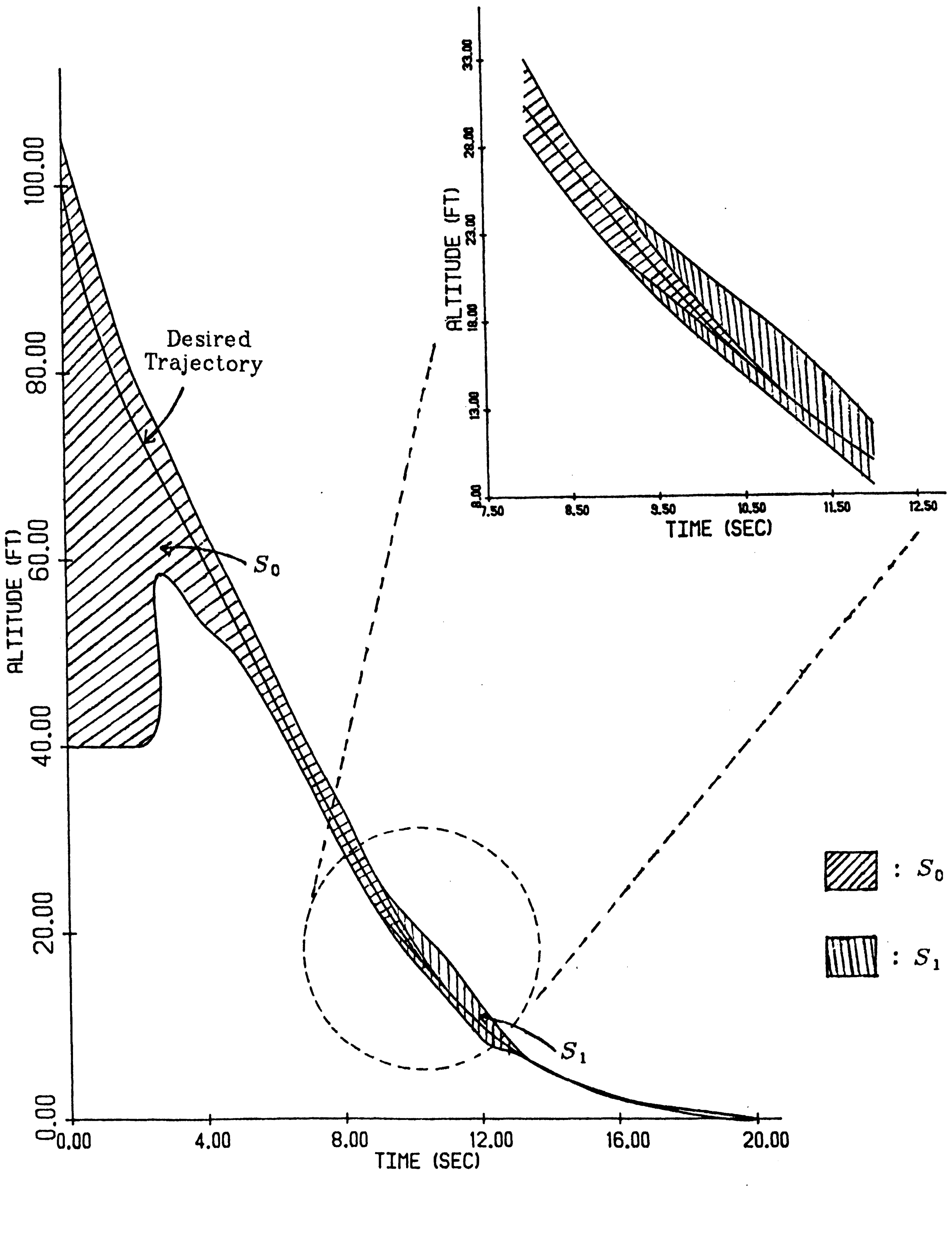


Figure 8(a). Allowed State Space: Altitude.

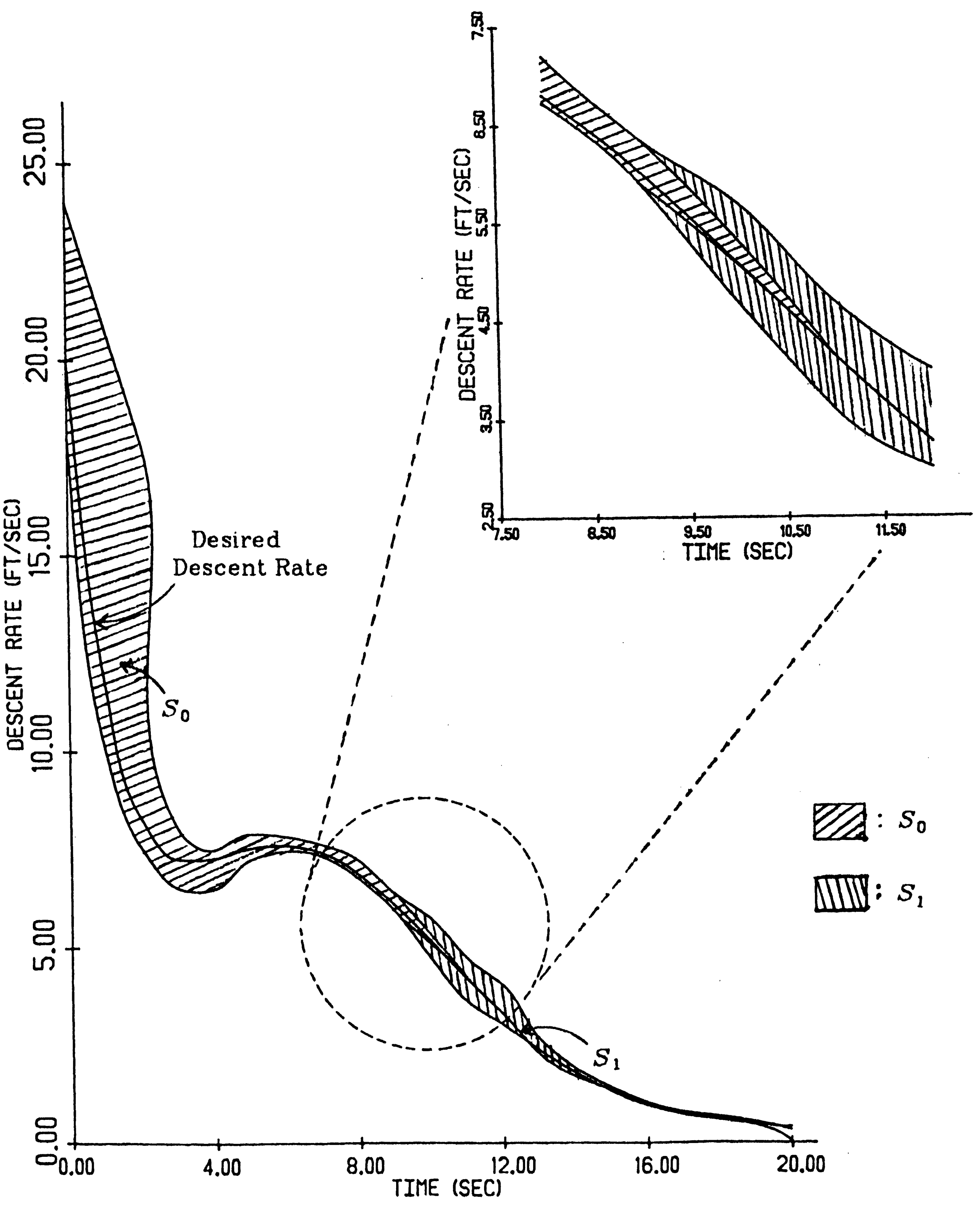


Figure 8(b). Allowed State Space: Descent Rate.

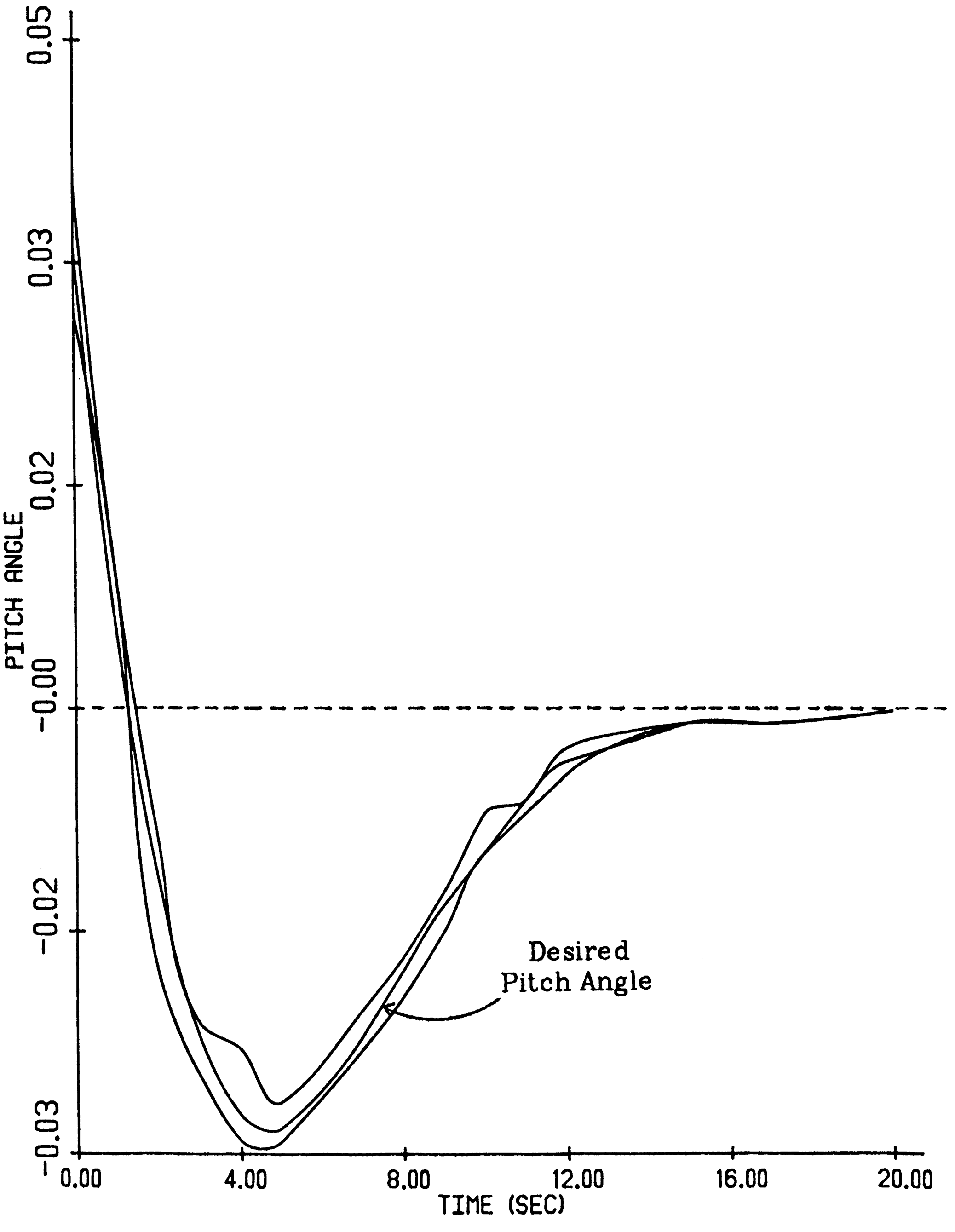


Figure 8(c). Allowed State Space: Pitch Angle.

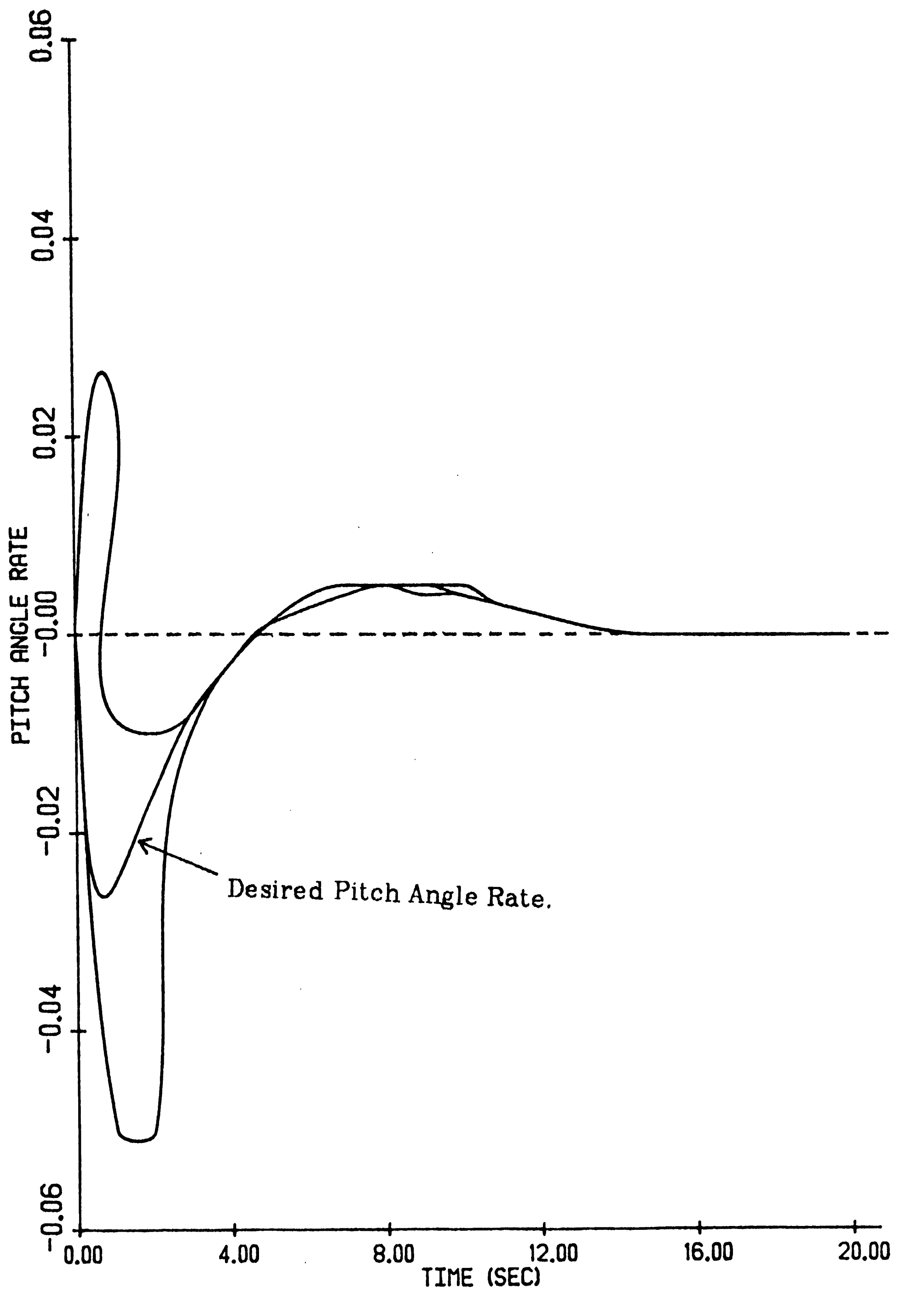


Figure 8(d). Allowed State Space: Pitch Angle Rate.

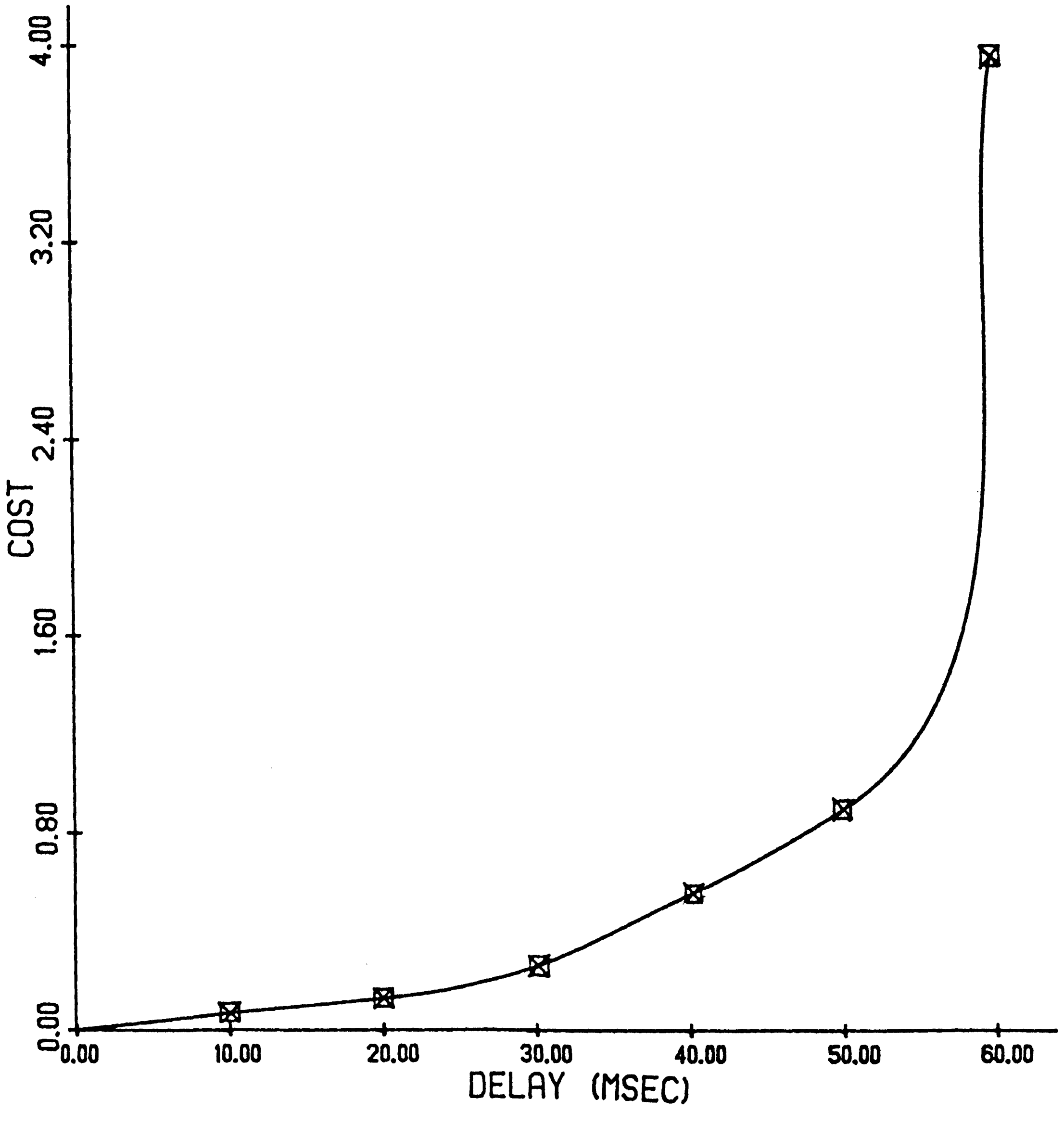


Figure 9. Landing Job Cost Function.

Feedback Term	Value
b_{11}	-0.600
b_{12}	-0.760
b_{13}	0.003
b_{32}	102.4
b_{33}	-0.4
c_{11}	-2.374

Table 1. Feedback Values

Weighting Factor	Value
$\varphi_1(t)$	99.0
$\varphi_{2,t_f}(t)$	20.0
$\varphi_3(t)$ $(0 \le t < 15)$ $\varphi_3(t)$ $(15 \le t \le 20)$ φ_{3,t_f}	0.0 0.0001 1.000
φ_4 φ_{4,t_f}	0.00005 0.001

Table 2. Weighting Factors

n	MEAN COST	p_{dyn}	$Tradeoff \times 10^7$
0	0.12848	0.3086E-15	-
1	0.12909	0.3086E-15	0.0
2	0.12971	0.3086E-15	0.0
3	0.13033	0.3086E-15	0.0
4	0.13095	0.3086E-15	0.0
5	0.13157	0.3086E-15	0.0

(a). Nominal execution time 20 msec.

n	MEAN COST	p_{dyn}	$Tradeoff \times 10^7$
0	0.26156	0.37037E-07	-
1	0.26431	0.37037E-08	121.5
2	0.26709	0.37037E-08	0.0
3	0.26991	0.37037E-08	0.0
4	0.27272	0.37037E-08	0.0
5	0.27567	0.37037E-08	0.0

(b). Nominal execution time 30 msec.

n	MEAN COST	p_{dyn}	$Tradeoff \times 10^7$
0	0.55352	0.30555E-06	-
1	0.55472	0.43055E-07	2177.0
2	0.55586	0.30555E-07	109.8
3	0.55694	0.30555E-07	0.0
4	0.55795	0.30555E-07	0.0
5	0.55891	0.30555E-07	0.0

(c). Nominal execution time 40 msec.

Table 3. Checkpoints

n	MEAN COST	p_{dyn}	$Tradeoff \times 10^7$
0	0.89694	0.46666E-06	-
1	0.90848	0.35666E-06	95.6
2	0.92025	0.26166E-06	80.6
3	0.93231	0.16666E-06	78.7
4	0.94466	0.71666E-07	76.9
5	0.95730	0.46666E-07	19.8

(d). Nominal execution time 50 msec.

$p_b = 0.9$, $MTBF = 10^4$ *hours*, $t_{ov} = 0.1$ *msec.* $t_b = 2.0$ *msec.* $t_s = 2.0$ *msec.*

Tradeoff ratio for n checkpoints $(n \geq 1)$

$$= \frac{\textit{COST with n chechpoints - COST with n-1 checkpoints}}{p_{dyn} \textit{ with n-1 checkpoints - } p_{dyn} \textit{ with n checkpoints}}$$

Table 3. (Cont.) Checkpoints