A Unified Method for Evaluating Real-Time Computer Controllers and Its Application

KANG G. SHIN, SENIOR MEMBER, IEEE, C. M. KRISHNA, MEMBER, IEEE, AND YANN-HANG LEE, MEMBER, IEEE

Abstract—**A computer-controlled system is a synergistic coupling of the *controlled process* and the *controller computer*. We have defined new performance measures for real-time controller computers based on this coupling.**

We present a systematic study of a typical critical controlled process in the context of new performance measures that express the performance of both controlled processes and controller computers (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 monotonically nondecreasing 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.

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. Second, 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 potential use for designing and/or evaluating real-time controllers and controlled process. In common with all other control techniques, the computational complexity involved in obtaining these measures is susceptible to the curse of dimensionality.

I. INTRODUCTION

THE use of computers in the control of life-critical processes, such as aircraft or nuclear reactors, is becoming popular owing to the increased capability and reliability, and the reduced cost, of modern computers and the increasing sophistication of controlled processes.

It is becoming increasingly apparent that the highly fuel-efficient aircraft that are expected to be in use in the next century demand faster control reaction than is possible for any human being to provide. In attempting to improve fuel-efficiency, the aircraft designer pushes his design to the edge of instability. Control situations will therefore present themselves in which fast, accurate, and consistent action is called for. It is expected that the major performance bottleneck in the control chain will be the pilot who can seldom react as quickly and as correctly as emergencies may demand. For this reason, while the pilot should properly be in control of flight policy, it is unreasonable to expect him to act as a real-time, low-level controller of the actuators aboard the aircraft of the future. To take over as a low-level controller (essentially an interface between the pilot and the aircraft/environmental system), one requires a controller computer. This device would perform such critical tasks as maintaining aircraft stability. Such a computer cannot be manually overridden for the simple reason that there would be no point in so doing; future aircraft are expected to be so intrinsically unstable that the pilot could not possibly fly the aircraft without the computer.

The use of a computer in controlling a process is qualitatively different from using the more traditional hard-wired digital controllers. One can model the latter type of controller relatively simply as a digital filter. Fault-tolerant computers used in control (e.g., FTMP [14], SIFT [15]) are far more complex, owing to their fault-tolerance and capacity to degrade gracefully. Issues of control-task scheduling, allocation, and optimal queue control arise, which are, of course, quite unknown in hard-wired control. One also has the problem of optimal (more realistically, quasi-optimal) management of redundancy, and the distribution of the computer's executive.

For these reasons, one cannot simply integrate a computer into the feedback loop of a controlled process as one would a digital controller. The fact that the control computer and the controlled process are designed in a more or less disjoint manner only serves to compound the difficulty of achieving optimal and ultrareliable control based upon a computer.

When computer systems are introduced whose behavior is critical to the safety of an aircraft, there is the need for formal evaluation and validation procedures. This presupposes that there are performance measures to characterize the behavior of controller computers, describing precisely the goodness of the controller computer *in the context of the application*.

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.

To solve this problem several contortions of the conventional measures have been proposed. Generally, these involve representing real-time computer performance as a vector $\boldsymbol{p} \in \boldsymbol{R}^p$, made up of such traditional measures as (conventional) reliablity, throughput, survivability, availability, etc. However, to compare two vectors, we need to map them into a scalar quantity. One solution is to assign weights to the various components of the performance vector and form an inner product to produce a scalar. That is, if the weight vector is $\boldsymbol{w}^T = (w_1, \cdots, w_p)$, then the mapping is $f:\boldsymbol{R}^p \rightarrow \boldsymbol{R}$ with $f(\boldsymbol{p}) = \boldsymbol{w}^T\boldsymbol{p}$.

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. 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

Manuscript received September 26, 1983; revised March 4, 1984 and March 28, 1984. Paper recommended by Past Associate Editor, D. M. Wiberg. This work was supported in part by NASA under Grant NAG 1-296.

K. S. Shin is with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

C. M. Krishna was with the Department of Electrical Engineering, The University of Michigan, Ann Arbor, MI 48109. He is now with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01002.

Y.-H. Lee was with the Deaprtment of Electrical Engineering, The University of Michigan, Ann Arbor, MI 48109. He is now with IBM Hawthorne Research Laboratory, Yorktown Heights, NY 10598.

0018-9286/85/0400-0357$01.00

objectively in terms of the *response time* of the controller computer. *Controller response time* is the interval between the initiation of a controller *job*—which is a predefined program, or a mutually interacting set of programs—and the actuator or display output that results. 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 nonzero controller response time has a detrimental effect on process behavior, our measures take the form of a quantification of this. The performance measures are considered in Section II in some detail prior to the presentation of an idealized example of their derivation.

Our performance measures are based on the following elementary observations.

1) Nonzero controller response time has a negative impact on the behavior of the controlled process.

2) There is a limit to how great the response time can be before the process behaves unacceptably.

3) Even if this response time is kept within the above bounds, an incremental increase tends to lead to a deterioration in controlled process behavior.

The example is that of a controller 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 Fig. 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 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. Indeed, the performance measures that we illustrate in this paper could equally well be used in any of the above applications.

Fig. 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 controller computer at regular intervals. Data rates are usually low: sensors generally put out fewer than 20 words a second.

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. 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. That is, 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 override the automatic systems, generating and canceling 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 [8], a controller computer generally (with the exception of systems dependent on real-time image processing) 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.

A control system executes *missions.* These are periods of operation between successive periods of maintenance. In the case

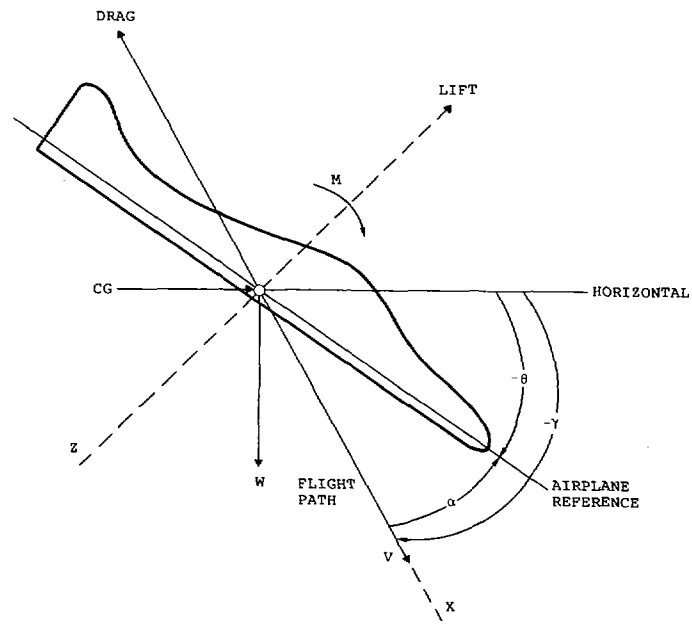


Fig. 1. Definition of aircraft angles [from (4)].

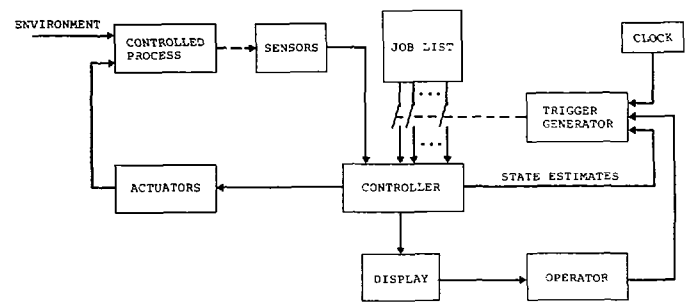


Fig. 2. A typical real-time control system.

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.

1) Takeoff/cruise until VHF omnirange (VOR)/distance measuring equipment (DME) is out of range.

2) Cruise until VOR/DME is in range again.

3) Cruise until landing is to be initiated.

4) Landing.

The phase to be considered here is landing; it takes about 20 s. The controller job that we shall treat is the control of the aircraft elevator deflection during landing.

The specific system employed is assumed to be organized as shown in Fig. 3. Sensors report on the four key parameters: altitude, descent rate, pitch angle, and pitch angle rate every 60 ms. We have a time-generated trigger, with a time period of 60 ms. Every 60 ms the controller computes the optimal setting of the elevator. 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. The controller response time is nominally 20 ms, although this can vary in practice due to failures or conditional branches in the executed code.

Since the process being controlled is critical (i.e., in which some failures can lead to catastrophic consequences), variations of controller response time 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

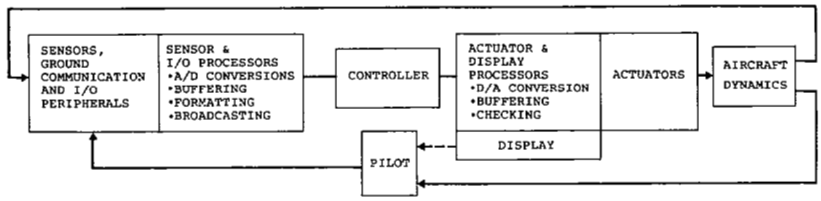


Fig. 3. Aircraft control system schematic.

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 response time is substantially more than 20 ms (the nominal execution time) but less than the inter-trigger interval of 60 ms.

iii) The response time is greater than 60 ms. 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.

This paper is organized as follows. In Section II we introduce the performance measures that will be used, and Section III contains a description of the controlled process. Since our goal is to illustrate the use of our performance measures and not to solve a control problem as such, the controlled process studied will be slightly idealized in this paper. In Section IV, we derive the measures associated with the controlled process (the aircraft). Section V identifies some of important potential uses of the performance measures, and the paper concludes with Section VI.

II. Performance Measures

A. Review of the Performance Measures

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

Our measures are all based on a single attribute: the probability distribution of the controller response time. Owing to random failures and conditional branches in the executed code, a controller computer in general exhibits stochastic behavior.

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. 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. In the aircraft-landing example we treat here, the states are the altitude, the vertical speed, the pitch angle, and the pitch angle rate. Each of these has an associated 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*.

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:

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

where $g_\alpha(\cdot)$ is a suitable continuous nondecreasing function of the response time 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 II-B and C and that of $g_\alpha(\cdot)$ is considered briefly below.

For controller response times less than the hard deadline, the cost function in (1) above is continuous, monotonically nondecreasing, 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*) $g_\alpha(\cdot)$ can be obtained using the performance indexes of the controlled process. These performance indexes are well-known to control theory and express the consumed energy, fuel, time, or some other physical parameters 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:

$$\Theta(t_0, t_f) = \int_{t_0}^{t_f} E[f_0(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \mid \boldsymbol{y}(\tau), t_0 \le \tau \le t]\ dt \tag{2}$$

where $E[\cdot \mid \cdot]$ represents conditional expectation, f_0 is the instantaneous index of performance at time t, and $\boldsymbol{x}(t) \in \boldsymbol{R}^n$, $\boldsymbol{u}(t) \in \boldsymbol{R}^l$, and $\boldsymbol{y}(t) \in \boldsymbol{R}^m$ represent the state, input, and measurement vectors, respectively. $\boldsymbol{u}(t)$ consists of two components, namely, the control input $\boldsymbol{u}_c(t) \in \boldsymbol{R}^{l_c}$ and the environmental input $\boldsymbol{u}_e(t) \in \boldsymbol{R}^{l_e}$. A good representation for $g_\alpha(\xi)$ is given by

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

where $\Psi_\alpha(\eta)$ = expected contribution of $\boldsymbol{u}_c$ to $\Theta(t_0, t_f)$ if response time of that particular execution of job α is η.

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

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

$$\tilde{S}(t) \equiv \sum_{i=1}^{r} \sum_{j=1}^{q_i(t)} f_i(RESP(V_{ij})). \tag{4}$$

Let $L(t)$ denote the probability distribution function of the mission lifetime which is defined as the operating interval duration between two successive service stages.

Our performance measures are then[1] as follows.

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. For our purposes, a hard deadline is missed whenever the controller produces no output within the deadline or an erroneous output. 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

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

total probability of failure. This is most decidedly not the case in real-time control systems.

Mean Modified Cost, $\tilde{M} \equiv \int_0^\infty E\{\tilde{S}(t) | \text{ System Never Suffers Dynamic Failure}\}\, dL(t)$: It can be shown that, for physical systems, this integral always exists since the mission lifetime is always finite. The qualifier "modified" is used because, implicit in the analysis, is the assumption that all initiated control jobs finish executing by the time the mission ends. This assumption need not always hold, but does not introduce any significant inaccuracy in our calculations, since the mission lifetime is usually much longer (e.g., several hours for aircraft and even several days or months for spacecraft) than the execution time of a job (this is at most a few seconds). Were this assumption not introduced, computing the mean cost would be a very tedious process.

The performance measures can be used to rank rival controller computers and to help design improvements to existing controllers *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 controller computers of the next decade handling a 10 h civilian flight. The mean cost is then employed to rank controllers that have passed the dynamic failure criterion. For further 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 capabilities. This is, as far as we know, a novel approach which ensures that the performance measures are specifically indicative of the controller performance in a given application. For this reason, these measures are intrinsically more reliable and effective than others in use.

B. 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 controller designers 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.

Hard deadlines can be obtained most easily 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 20 s or so that it takes to complete the landing, and 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 undershoot, 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. Let the state of the controlled process at time t be denoted by $\boldsymbol{x}(t)$. State transitions are characterized by a mapping $\phi: T \times T \times X \times U \to X$ where $T \subset \boldsymbol{R}$ represents the time region, $X \subset \boldsymbol{R}^n$ the state-space, and $U \subset \boldsymbol{R}^l$ the input space; that is,

$$\boldsymbol{x}(t_1) = \phi(t_1, t_0, \boldsymbol{x}(t_0), \boldsymbol{u}) \tag{5}$$

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

$$\tau_{d\alpha}(\boldsymbol{x}(t_0)) \equiv \inf_{u \in \Omega} \sup \{\tau \,|\, \phi(t_0+\tau, t_0, \boldsymbol{x}(t_0), \boldsymbol{u}) \in X_A\}. \tag{6}$$

Thus, for every point in the state-space, we have for each critical job a corresponding hard deadline. When closed-form solutions for the hard deadlines as a function of the state and the input are not available, the unconditioned hard deadlines are generally almost impossible to obtain. A case in point is the aircraft-landing example that we consider below.

To remedy this, we allow for the calculation of the infimum over only a subset of the admissible input or state-space, and define *conditional hard deadlines*. Assume that the sets $\omega \subset \Omega$ and $\sigma \subset X_A$ are specified, and also that $\boldsymbol{x}(t_0) \in \sigma$. The conditional hard deadline of job α, denoted by $\tau_{d\alpha|\omega,\sigma}$, is defined as

$$\tau_{d\alpha|\omega,\sigma}(\boldsymbol{x}(t_0)) \equiv \inf_{u \in \omega} \sup \{\tau \,|\, \phi(t_0+\tau, t_0, \boldsymbol{x}(t_0), \boldsymbol{u}) \in \sigma\}. \tag{6a}$$

In this paper, we calculate for the aircraft-landing case study, the hard deadlines conditioned on the assumption that the elevator deflection applied in every instance is dictated by a predefined control law [defined later in (13)].

If the environment is stochastic in character, 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. Notice that the concept of a random hard deadline is a significant departure from the conventional approach where a hard deadline is assumed to be given as a deterministic constant.

C. Allowed State-Space and Its Decomposition

When closed-form solutions are not available for the hard deadlines, a numerical solution must be derived. To do this, we divide the allowed state-space down into disjoint *state-subsets* which are aggregates of states in which the system exhibits roughly the same behavior. Even if there do not exist clear boundaries for these state-subsets, one can always force the allowed state-space to be divided into state-subsets so that a sufficient safety margin can be provided. In each such state-subset, each critical controller job has a unique hard deadline and finite cost function. For convenience, a controller "task" is defined as a controller job operating within a designated state-subset. By definition, each task has a state-independent cost function.

Remarks: In some state-subsets, 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 state-subset. What *does* usually happen in these circumstances is that the system moves into a new state-subset—or at the least toward the state-subset boundary—in which the dangers of catastrophic failure are greater. In this state-subset, 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 state-subset, it only has to map into a critical *task* in at least one state-subset. Also, these state-subsets are job-related, i.e., the same allowed

state-space can divide into a different set of state-subsets for each control job.

Let S_i for $i = 0, 1, \cdots, s$ be disjoint state-subsets of X_A with $X_A = \cup_{i=1}^{s} S_i$ and let J denote a controller job. Then, we need the projection: $(J, X_A) \rightarrow ((T_0, S_0), (T_1, S_1), \cdots, (T_s, S_s))$ 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 mutual coupling problem mentioned above.

1) Allowed State-Space: The allowed 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) 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 that apply, it becomes possible to satisfy the terminal constraints.

Note that leaving X_A^1 means that no matter how good our subsequent control, failure has occurred.[2] On the other hand, altering the allowed input space, i.e., changing the control available can affect the set X_A^2. The allowed state-space is then defined as $X_A \equiv X_A^1 \cap X_A^2$.

Obtaining state-space 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 usually much easier to obtain a reasonably large subset $X_A^a \subset X_A$. 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, and by so doing, err on the side of safety. Also, the information we need about X_A may be determined to as great a 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, X_A^a, simply as the "allowed state-space."

2) On Obtaining the State-Subsets: 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 state-subsets corresponding to these intervals. To take a concrete example, consider a state-space $X \subset R^n$ 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 state-subset 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 $\Delta_1, \Delta_2, \cdots$ such that the state-subsets were defined by intervals with the $\Delta's$ as their endpoints. This would correspond to quantizing with variable step sizes. With this complete, the individual state-subsets thus derived can be further divided using mean cost or some other function of the finite cost function as a quantization criterion.

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

[2] Strictly speaking, of course, there can be no subsequent control since by leaving X_A^1 the system has failed catastrophically before the next control could be implemented.

D. Computational Complexity

When closed-form solutions are available for the state equations, they can be solved backwards from the terminal constraints to obtain analytical results for X_A^2. When closed-form solutions are not available, numerical techniques must be used, and solving differential equations backwards can give rise to numerical instability. In the most general case, the amount of computations required increases rapidly with the number of state variables and the complexity of the state equations; therefore, the exact amount of required computations depends on the solution approach employed and the actual system under consideration.

One begins by deriving the optimal trajectory, which requires the problem- and solution-specific amount of computations. Then, obtaining X_A^2 consists of searching the n-dimensional state-space surrounding this trajectory. In general, this becomes an exhaustive search for the boundary of X_A^2. If the search is confined to an $(n+1)$ dimensional parallelopiped (the $n+1$th dimension represents time) with edge lengths $k_1, k_2, \cdots, k_{n+1}$, and the digital resolution is Δ, then defining $m_i = \lceil k_i/\Delta \rceil$, $i = 1, \cdots, n+1$, the state equations must be solved $O(\prod_{i=1}^{n+1} m_i)$ times. The values for k_i must be set by the user.

In the event that X_A^2 is known to be convex, a binary search can be instituted along one of the dimensions, reducing the search complexity to $O(m_1 \cdots m_{j-1} m_{j+1} \cdots m_{n+1} \log_2 m_j)$ where $m_j = \max\{m_1, \cdots, m_n\}$.

As for the cost functions and the hard deadlines, these must also be derived, in general, by an exhaustive analysis. The allowed state-space is quantized, and deadlines and cost functions must be found for each of the resulting "cells."

It is therefore clear that for a process with large n, these methods become prohibitively expensive. Two comments are now in order. The first is that this approach is not uniquely susceptible to the curse of dimensionality: all optimal control theory techniques suffer this problem when complex processes are considered. The second comment is that, in certain cases, additional knowledge about the process dynamics can be used to improve the efficiency of the search for the X_A^2 boundary, and for the hard deadlines and cost functions.

III. 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:

$$\dot{x}_1(t) = b_{11}x_1(t) + b_{12}x_2(t) + b_{13}x_3(t) + c_{11}m_1(t, \xi) \tag{7a}$$

$$\dot{x}_2(t) = x_1(t) \tag{7b}$$

$$\dot{x}_3(t) = b_{32}x_2(t) + b_{33}x_3(t) \tag{7c}$$

$$\dot{x}_4(t) = x_3(t) \tag{7d}$$

where x_1 is the pitch angle rate, x_2 the pitch angle, 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 I. Recall that ξ denotes controller response time.

The phase of landing takes about 20 s. Initially, the aircraft is at an altitude of 100 feet, traveling at a horizontal speed of 256 ft/s. This latter velocity is assumed to be held constant over the entire landing interval. The rate of descent at the beginning of this phase is 20 ft/s. 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

TABLE I
FEEDBACK VALUES

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

0° and 10° to avoid landing on the nosewheel or on the tail, and the angle of attack (see Fig. 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 ft/s so that the undercarriage can withstand the force of landing.

The desired altitude trajectory (feet) is given by

$$h_d(t) = \begin{cases} 100e^{-t/5} & 0 \le t \le 15 \\ 20 - t & 15 \le t \le 20 \end{cases} \tag{8}$$

while the desired rate of ascent (ft/s) is

$$\dot{h}_d(t) = \begin{cases} -20e^{-t/5} & 0 \le t \le 15 \\ -1 & 15 \le t \le 20. \end{cases} \tag{9}$$

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

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

$$\Theta(\xi) = \int_{t_0}^{t_f} e_m(t, \xi)\, dt \tag{10}$$

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

$$\begin{aligned} e_m(t, \xi) = {} & \phi_h(t)[\dot{h}_d(t) - x_4(t)]^2 + \phi_{\dot{h}}(t)[\dot{h}_d(t) - x_3(t)]^2 \\ & + \phi_\theta(t)[x_{2d}(t) - x_2(t)]^2 \\ & + \phi_{\dot{\theta}}(t)[x_{1d}(t) - x_1(t)]^2 + [m_1(t, \xi)]^2 \end{aligned} \tag{11}$$

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

$$\phi_h(t) = \phi_4(t) + \phi_{4,t_f}\delta(20 - t) \tag{12a}$$

$$\phi_{\dot{h}}(t) = \phi_3(t) + \phi_{3,t_f}\delta(20 - t) \tag{12b}$$

$$\phi_\theta(t) = \phi_{2,t_f}(t)\delta(20 - t) \tag{12c}$$

$$\phi_{\dot{\theta}}(t) = \phi_1(t) \tag{12d}$$

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 II.

The control law for the elevator deflection is given by

$$\begin{aligned} m_1(t, \xi) = {} & \omega_s^2 K_s T_s[k_1(t - \xi) - k_{11}(t - \xi)x_1(t - \xi) - k_{12}(t - \xi)x_2(t - \xi) \\ & - k_{13}(t - \xi)x_3(t - \xi) - k_{14}(t - \xi)x_4(t - \xi)] \end{aligned} \tag{13}$$

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

TABLE II
WEIGHTING FACTORS

Weighting Factor	Value
$\phi_1(t)$	99.0
$\phi_{2,t_f}(t)$	20.0
$\phi_3(t)$ $(0 \le t < 15)$ $\phi_3(t)$ $(15 \le t \le 20)$ ϕ_{3,t_f}	0.0 0.0001 1.000
ϕ_4 ϕ_{4,t_f}	0.00005 0.001

IV. 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 response time increases, how much extra overhead is added to the performance index? Also, it is intuitively obvious that too great a response time 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 response times that permit a safe landing.

The control for various values of fixed controller response time is computed using (13) and is shown in Fig. 4. Due to the absence of any random effects, elevator deflections for all the response times considered tend to the same value as the end of the landing phase (20 s) 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 values of controller response time is even more marked. We present an example of this in Fig. 5. The random effect considered here is the elevator being stuck at $-35°$ for 60 ms 8 s into the landing phase due to a faulty controller order. The controlled process is assumed in Fig. 5 to be in the state-subset in which the landing job maps into a noncritical task (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 response time, we would expect the system to become unstable. This is indeed the case in the present problem, although this point occurs beyond a response time of 60 ms for all points in the allowed state-space (obtained in the next section), which cannot by definition occur here.

A. 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, 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 ft/s 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 ft in the horizontal direction from the moment the landing phase

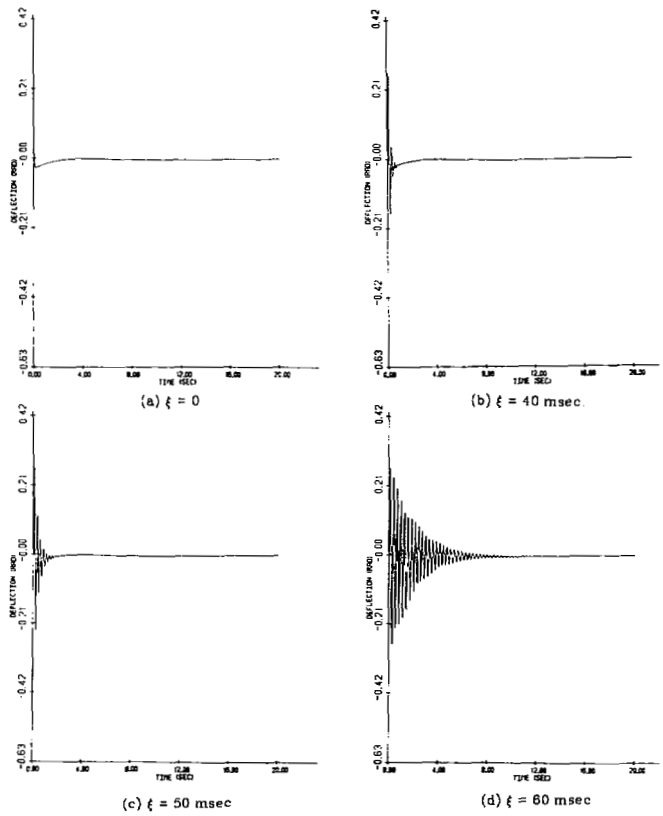


Fig. 4. Elevator deflection.

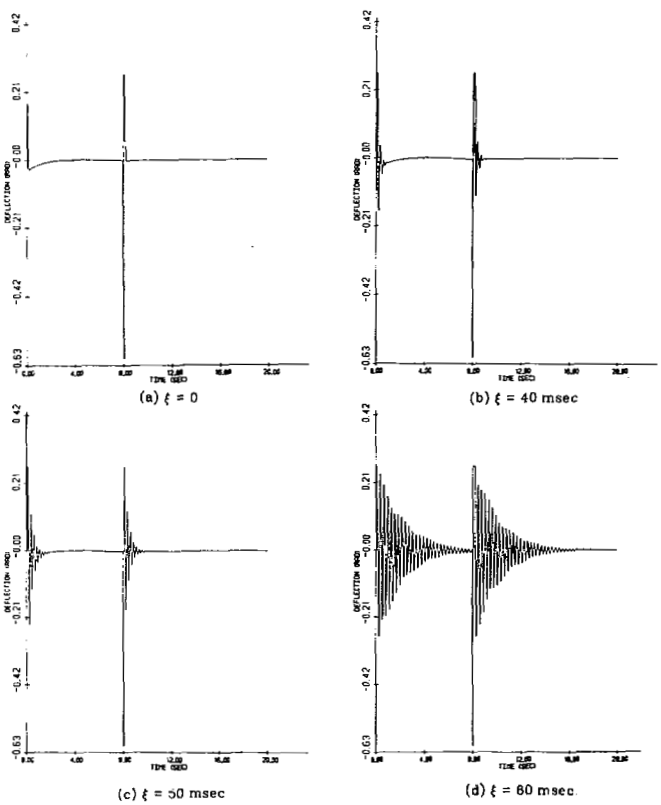


Fig. 5. Elevator deflection with abnormality.

begins. The horizontal velocity is assumed to be kept constant throughout the landing phase at 256 ft/s (this would constitute a separate controller job; we do not consider here how that is to be done). Thus, touchdown should occur between 19 and 20 s after the descent phase begins. The only control is the elevator deflection which must be kept between $-35°$ and $15°$.

The restriction that we place on the allowed state-space, X_A^a, is for ease of representation. We define allowed "component state-spaces" for the individual components of the state-vector such that any point in the four-dimensional state-space whose individual components each lie in the corresponding component state-spaces is in the overall allowed state-space. The component state-spaces are determined by perturbing the state-values around the desired trajectory and determining the maximum perturbation possible under the requirement that no terminal constraint be violated. They were obtained by a search process and are plotted in Fig. 6. It should again be stressed that what we plot in Fig. 6 is a *subset* of X_A.

B. Designation of State-Subsets

We subdivide the allowed state-space found above using the method described in Section II. 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 allowed state-space. The value of Δ chosen is 60 ms. In other words, we wish to consider only the case where a trigger is "missed."

The allowed state-space in Fig. 6 is subdivided into two state-subsets S_0 and S_1. These correspond to the deadline intervals [120, ∞) and [60, 120], respectively. S_0 is the noncritical region corresponding to the [120, ∞) 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 controller 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 an air crash.

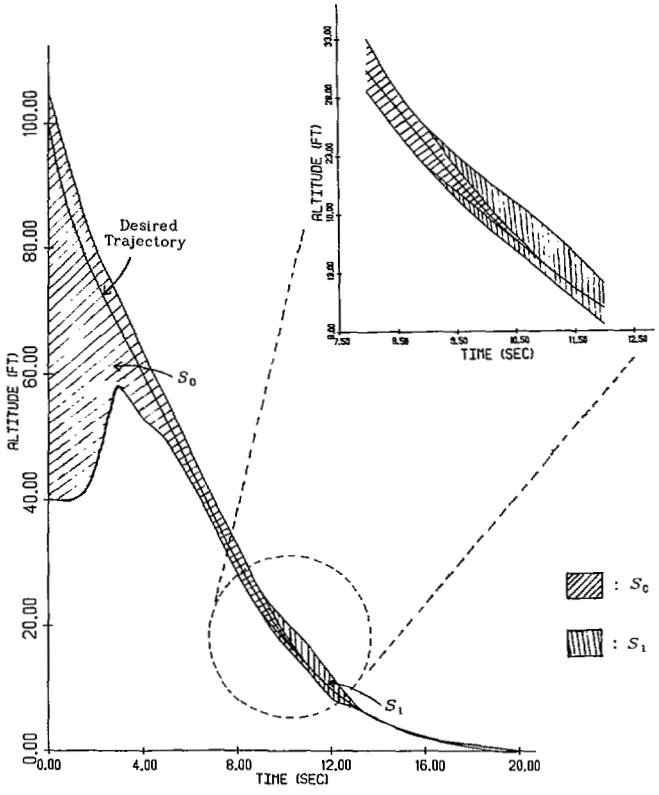
Notice that we explicitly consider only missing a single trigger, not the case when two or more triggers might be missed in sequence. This is because dynamic failure is treated here as a function of the state at the moment of triggering. If two successive triggers are missed, for example, we have to 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 ms is therefore meaningless in this case since the triggers occur once every 60 ms. This is why the second deadline interval considered is [120, ∞), not [120, 180).

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

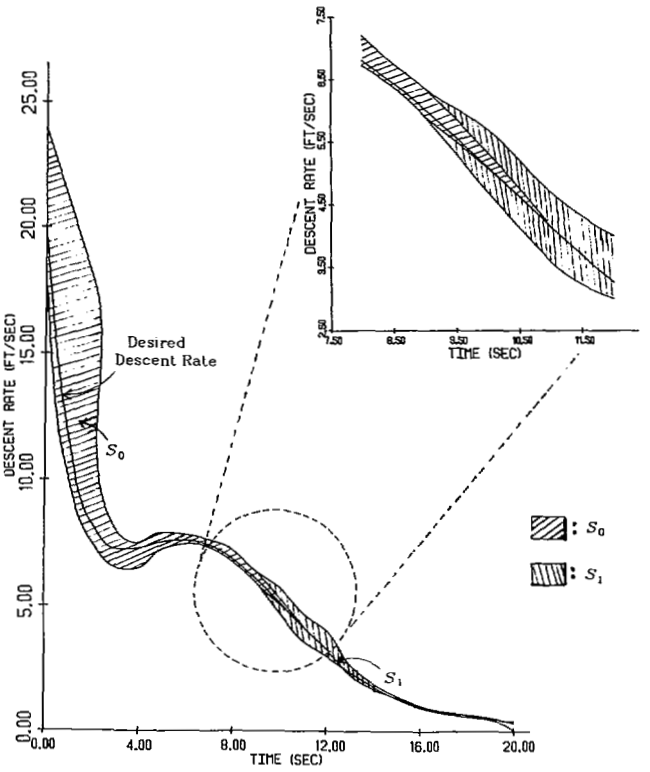
C. Finite Cost Functions

As indicated in the preceding sub-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.

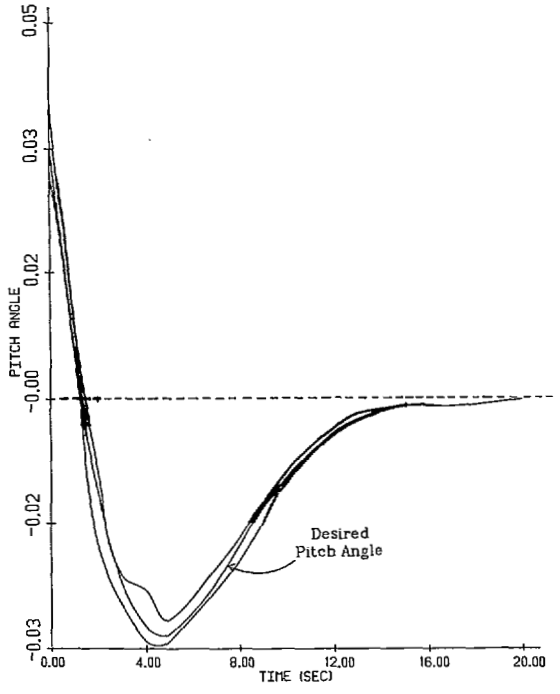
The determination of the cost function is carried out as a direct application of its definition. That is, the process differential equations are solved with varying values of ξ. The value of ξ cannot be greater than the inter-trigger interval of 60 ms since, by assumption, no job pipelining is allowed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



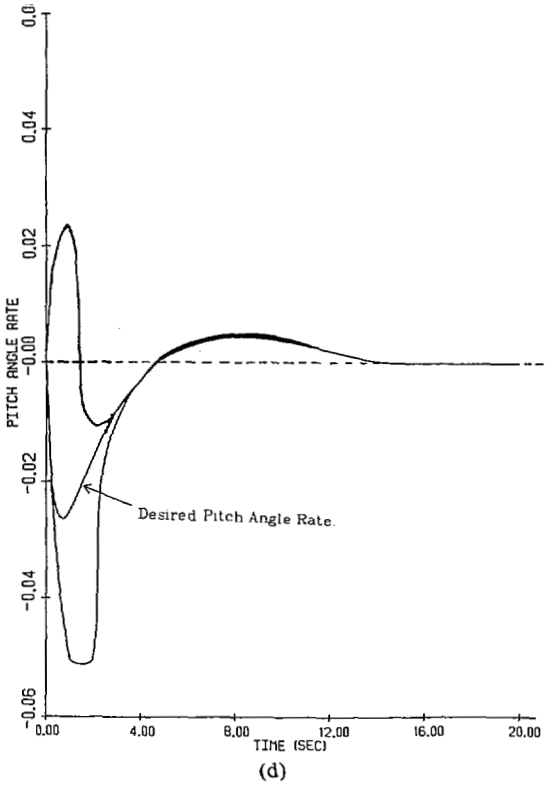
(a)



(b)



(c)



(d)

Fig. 6. (a)Allowed state-space: altitude. (b) Allowed state-space: descent rate. (c) Allowed state-space: pitch angle. (d) Allowed state-space: pitch angle rate.

trigger. The finite cost function is defined as[3]

$$g(\xi) = \Psi(\xi) - \Psi(0). \tag{14}$$

This function is found by computation to be approximately the same over the entire allowed state-space as defined in Fig. 6.

In Fig. 7, 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.

The reader should compare the nature of the cost function to the plots showing elevator deflection in Fig. 4, 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.

V. Applications of the Measures

By objectively associating a cost with incremental controller response delay, it becomes possible to use these performance measures in the design and evaluation of real-time control systems.

It is not difficult to see why the measures should be so useful. The distribution of the hard deadline and the shape of the finite cost function, together with job execution time distributions, contain almost all the information about the controlled process that is needed by the controller designer.

We briefly indicate some of the applications below.

1) *Task Scheduling:* Efficient task scheduling on the controller computer requires that the tasks be assigned priorities objectively and appropriately. The cost functions make this possible, by specifying deadlines and the cost of incremental controller response delay. Dynamic control of queues in front of shared resources is also facilitated.

2) *Task Allocation:* Since the cost functions are objective, they can be used as appropriate criterion functions in the allocation of control tasks to processors in a multiprocessor controller. The criterion function that comes immediately to mind is the expected total cost of operation, as exemplified by the cost functions and the multiprocessor response time distribution. The objective would be to minimize this mean cost, subject to constraints on the probability of dynamic failure. This would be a considerable advance over current allocation criteria [9] which are based on intuitive and ad-hoc measures of performance.

3) *Specification and Evaluation of Controllers:* The mean cost and the probability of dynamic failure can be used to specify and rank controller computers for control applications. Generally, one would envisage a maximum tolerable probability of dynamic failure being specified. As long as this overriding constraint was satisfied, one would then be interested in minimizing the control overhead as represented by the mean cost. A multiplicity of controllers that all passed the dynamic failure criterion could then be ranked in the order of the total expected cost associated with them.

4) *Optimal Number of Checkpoints:* Checkpoints [10] are authenticated copies of status information about currently executing computer jobs. They store intermediate results so that if a failure is detected, it might be possible to avoid reexecution of the entire job affected, and to restart computation from one of these intermediate points. The problem of how many checkpoints should be used is relevant since the establishment of a checkpoint takes time, which is a nonreplenishable resource. We have shown [11] that our performance measures can be used to determine the optimal number of checkpoints.

[3] Recall that $\Psi(\xi)$ represents the contribution to the performance index 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.

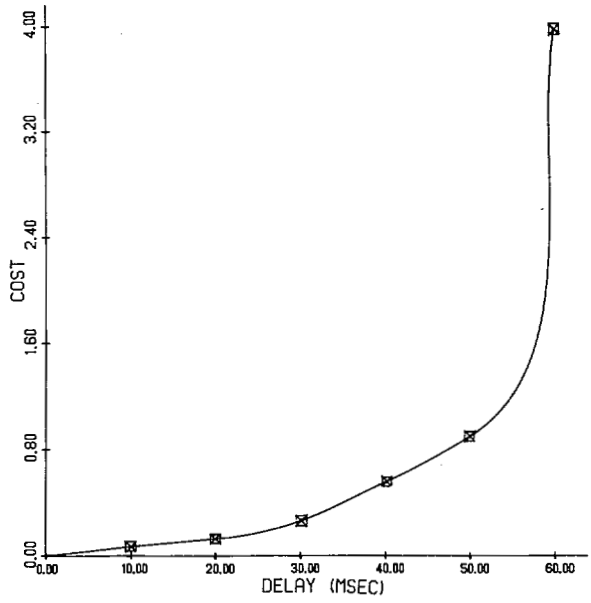


Fig. 7. Finite cost function.

5) *Optimal Management of Redundancy:* The probability of failure of controllers used in critical applications such as the one considered above is usually far lower than the probability of failure of any one of its components. One favorite means of enhancing overall system reliability over that of individual components is to use NMR clusters, i.e., clusters of N processors each executing the same job and voting on the results [12]. It is traditionally assumed that an increase in the size of the cluster leads to an increase in overall reliability. This is true as long as the applications are not time-critical. Unfortunately, for time-critical control applications, this is not the case: the time overheads imposed by management of redundancy is considerable. For example, in the SIFT aircraft-control computer currently under test at the NASA Langley Research Center, the above overheads for a 5-MR cluster account for about 78 percent of the execution time [13]. This overhead grows rapidly with increase in cluster size. There is therefore a tradeoff here: as the cluster size increases, the probability of a crippling hardware failure declines, but the nonstatic component of the failure probability increases. The optimal size of an NMR cluster depends on the resolution of this tradeoff. Our cost functions and hard deadlines have shown themselves effective in this matter.

VI. Discussion

In this paper, we have presented a unified method for evaluating controller computers, and considered an important application in controller design.

Central to our paper is the idea that it is possible to objectively quantify the performance of a controller by linking it to the controlled process. Owing to this objectivity, there are many possible extensions to this work. One extension, currently 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 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.

It would be useful at this point to review what we believe is the main contribution of this work.

That the controller delay, i.e., feedback delay, is detrimental to the performance of a controlled process is well known. For some reason, however, there has been (to our knowledge) no attempt to systematically link the effect of feedback delay to the design of the controller computer. The purpose of this work has therefore been to show the following.

1) It is possible, from a series of relatively simple observations, to derive a set of performance measures that are objective, and that are more appropriate to control applications than other performance measures, of which *performability* [5] is a notable example. Without performance measures of this kind, the quasi-optimal design of controller computers (including the algorithms used to control the computers themselves) and their validation would be difficult.

2) Using performance measures such as the ones presented, one can abstract into a compact form the needs of the controlled process. Given the hard deadlines, the cost functions, and the control algorithms, one can then proceed to design the controller computer without any extra detailed knowledge of the controlled process. This also has the useful side effect of making the design interface between the controller computer and the controlled process "clean," making for greater reliability of design, and lowered developmental costs.

The two disadvantages of this approach are 1) that when the controlled process is very complex, the computational resources needed to calculate the performance measures may be very large, and 2) a performance index that is "natural" to the controlled process must exist for this approach to be feasible. We have already considered the first disadvantage in Section II. We can see no way of relaxing the second limitation, since the absence of a natural performance index for the controlled process means that it is impossible to accurately quantify the performance of that process (or indeed to state precisely what is meant by "performance"). Since our whole approach focuses on quantifying the impact that increments in controller response time have on the performance of the controlled process, if the latter cannot be defined, neither can the impact of response time increments.

ACKNOWLEDGMENT

The authors thank the anonymous referees for their careful reading of the manuscript, and their useful suggestions. The assistance of R. Butler and M. Holt of the NASA Langley Research Center is also acknowledged.

REFERENCES

[1] C. M. Krishna and K. G. Shin, "Performance measures for multiprocessor controllers," in *Performance '83*, A. K. Agrawala and S. K. Tripathi, Eds. Amsterdam, The Netherlands: North-Holland, May 1983, pp. 229–250.

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

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

[4] F. J. Ellert and C. W. Merriam, "Synthesis of feedback controls using optimization theory—An example," *IEEE Trans. Automat. Contr.*, vol. AC-8, pp. 89–103, Apr. 1963.

[5] L. T. Wu, "Models for evaluating the performability of degradable computing systems," Univ. Michigan, Ann Arbor, Comput. Res. Lab. Rep. CRL-TR-7-82, June 1982.

[6] J. F. Meyer *et al.*, "Performability evaluation of the SIFT computer," *IEEE Trans. Comput.*, vol. C-29, pp. 501–509, June 1980.

[7] J. H. Wensley *et al.*, "Design study of software-implemented fault-tolerance computer," NASA Contract. Rep. 3011, 1982.

[8] K. G. Shin and C. M. Krishna, "A distributed microprocessor system for controlling and managing military aircraft," in *Proc. Distributed Data Acquisition, Comput. Contr. Symp.*, Miami, FL, Dec. 1980, pp. 156–166.

[9] W. W. Chu *et al.*, "Task allocation in distributed data processing," *Comput.*, vol. 13, pp. 57–70, Nov. 1980.

[10] K. M. Chandy, "A survey of analytic models of rollback and recovery strategies," *Comput.*, vol. 8, pp. 40–47, May 1975.

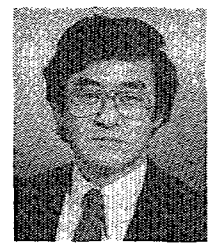
[11] C. M. Krishna, K. G. Shin, and Y. H. Lee, "Optimization criteria for checkpoint placement," *Comm. ACM*, vol. 27, pp. 1008–1012, Oct. 1984.

[12] D. P. Siewiorek, "Reliability modeling of compensating module failures in majority voted redundancy," *IEEE Trans. Comput.*, vol. C-24, pp. 525–533, May 1975.

[13] D. Palumbo and R. Butler, "SIFT—A prelimiary evaluation," in *Proc. Fifth Digital Avionics Symp.*, Seattle, WA, Oct. 1983.

[14] A. L. Hopkins *et al.*, "FTMP—A highly reliable fault-tolerant multiprocessor for aircraft," *Proc. IEEE*, vol. 66, pp. 1221–1239, Oct. 1978.

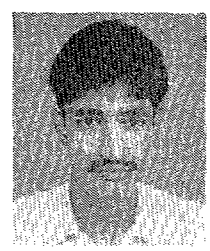
[15] J. H. Wensley *et al.*, "SIFT—Design and analysis of a fault-tolerant computer for aircraft control," *Proc. IEEE*, vol. 66, pp. 1240–1255, Oct. 1978.



Kang G. Shin (S'75–M'78–SM'83) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

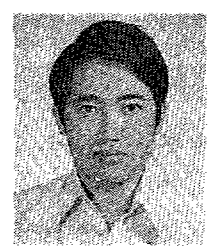
From 1978 to 1982 he was an Assistant Professor at Rensselaer Polytechnic Institute, Troy, NY. He was also a Visiting Scientist at the U.S. Air Force Flight Dynamics Laboratory in 1979 and at Bell Laboratories, Holmdel, NJ, in 1980, where his work was concerned with distributed airborne computing and cache memory architecture, respectively. He taught short courses for the IBM Computer Science Series in the area of computer architecture. Since September 1982, he has been with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, where he is currently an Associate Professor. His current teaching and research interests are in the areas of distributed real-time and fault-tolerant computing, computer architecture, and robot control, planning,,and programming.

Dr. Shin is a member of the Association for Computing Machinery, Sigma Xi, and Phi Kappa Phi.



C. M. Krishna (S'84–M'84) received the B.Tech. degree from the Indian Institute of Technology, Delhi, India, in 1979, the M.S. degree from Rensselaer Polytechnic Institute, Troy, NY, in 1980, and the Ph.D. degree from the University of Michigan, Ann Arbor, in 1984, all in electrical engineering.

Since September 1984, he has been on the Faculty of the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst. His research interests include reliability modeling, queueing and scheduling theory, and distributed architectures and operating systems.



Yann-Hang Lee (S'81–M'84) received the B.S. degree in engineering science and the M.S. degree in electrical engineering from National Cheng Kung University, Taiwan, Republic of China, in 1973 and 1978, respectively, and the Ph.D. degree in computer information and control engineering from the University of Michigan, Ann Arbor, in 1984.

Currently, he is with IBM Hawthorne Research Laboratories, Yorktown Heights, NY. His research interests include distributed computer systems, multiprocessor, performance evaluation, and fault-tolerant computing.