JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 27, No. 5, September–October 2004

Anytime Control Algorithm: Model Reduction Approach

Raktim Bhattacharya*
California Institute of Technology, Pasadena, California 91125
and
Gary J. Balas†
University of Minnesota, Minneapolis, Minnesota 55455

Recently, there has been considerable interest in anytime algorithms for real-time systems. Anytime algorithms are computational models that compromise quality of result for computational time. The tolerance to fluctuating CPU time makes anytime algorithms operationally optimal for real-time task scheduling. A methodology is presented that transforms linear control algorithms into anytime control algorithms. Implementation of a linear control algorithm involves matrix–vector multiplications that require a fixed computational time. Such algorithms fail to compute the controller output if the alloted CPU time is less than required and cannot make use of any excess CPU time that might be available. When implemented as a real-time system, the static nature of the required computational time makes it operationally suboptimal for task scheduling. Linear control algorithms are transformed to anytime control algorithms by switching between controllers of different order. Balanced truncation and residualization are considered as model reduction tools to generate a set of reduced-order controllers, and a switching algorithm is presented that smoothly switches between them to accommodate variation in available computational time.

I. Introduction

IN recent times, advancement in digital technology has led to the design of complex computational systems. These systems usually interact with an environment that demands more out of some algorithms and less out of others, at different times in their operation life. Therefore, it is not feasible to perform accurate computation at all times by all of the algorithms in the system. Anytime algorithms provide a technique for allocating computational resources to the most useful algorithm, thereby enabling optimal usage of hardware resources.

Anytime algorithms differ from conventional computational procedures in several ways.[1] Anytime algorithms are algorithms that compromise performance for computational time. They are capable of providing results at any point in their execution. The quality, accuracy, or performance of the algorithm improves with increased processing time. The improvement in the solution is large in the early stages of computation but diminishes over time.

Anytime algorithms first emerged in the area of artificial intelligence. Early applications of such algorithms can be found in medical diagnosis and mobile robot navigation. The term anytime algorithm was coined by Dean and Boddy[2,3] in the late 1980s in the context of their work on time-dependent planning. They used this idea to solve a path-planning problem involving a robot assigned to deliver packages to a set of locations. Horvitz introduced a similar idea, called flexible computation, to solve time-critical decision problems.[4] In 1991, Liu et al.[5] introduced the concept of imprecise computation and applied it to real-time systems. They showed that imprecise computation techniques provide scheduling flexibility by trading off the quality of result to meet computational deadlines. Ever since, the concept of imprecise computation has been applied to solve several diverse problems.[6–9] The idea of anytime algorithms is also similar to the notion of rationality in automated reasoning and search, investigated by Russel and Wefald,[10,11] Doyle,[12] and D'Ambrosio.[13]

Real-time computational tasks that are anytime algorithms prove to be useful in the design of real-time systems. The property of anytime algorithms to trade computational time for decision quality results in optimal hardware utilization in real-time systems.[14–16] Zilberstien[17] defines a real-time task to be operationally rational if it optimizes the allocation of resources to its performance components in order to maximize its overall expected utility. Such tasks are able to vary their execution time according to time pressure. This capability can be achieved if traditional algorithms, whose expected run times are normally fixed, are replaced by more flexible computational modules, namely, anytime algorithms.

Currently, control algorithms are implemented as digital control systems. Such systems are real-time systems with a dedicated application and are resource limited because of constraints on size, weight, space, or power. From a functional point of view, it is critical to provide a jitter-free periodic execution while meeting execution deadlines. From the point of view of economics, it is critical to maximize hardware utilization. Clearly, if control algorithms demonstrated characteristics of anytime algorithms, they would be well suited for implementation under dynamic resource management (Chapter 3 in Ref. 18). This serves as the motivation for the research work presented in this paper. We define control algorithms that behave as anytime algorithms as anytime control algorithms.

In the controls literature, control algorithms that are based on online optimization, such as receding horizon control,[19–22] can be tuned to vary the required computational time. The optimization problem for these algorithms is essentially minimization of a cost function with state and control constraints. The computational time for such algorithms can be varied dynamically by varying the number of decision variables, the complexity of system model, the accuracy of numerical algorithms, or the optimality of the solution.[23] Also, because the process of minimization is iterative in nature, it can be preempted to obtain the current best solution. Therefore, any control algorithm that is based on online optimization can potentially qualify as an anytime control algorithm.

For linear controllers, as defined by Eqs. (1) and (2), computation of the controller output involves matrix–vector multiplications and vector–vector additions.

Continuous time:

$$\dot{x}_c = Ax_c + Bu_i, \qquad u_0 = Cx_c + Du_i \tag{1}$$

Received 6 December 2002; revision received 6 October 2003; accepted for publication 8 March 2004. Copyright © 2004 by Raktim Bhattacharya and Gary J. Balas. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. Copies of this paper may be made for personal or internal use, on condition that the copier pay the $10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 0731-5090/04 $10.00 in correspondence with the CCC.

*Postdoctoral Scholar, Control and Dynamical Systems; raktim@cds.caltech.edu. Student Member AIAA.

†Professor, Aerospace Engineering and Mechanics Department; balas@aem.umn.edu. Associate Fellow AIAA.

Discrete time:

$$x_c^{k+1} = \hat{A}x_c^k + \hat{B}u_i^k, \qquad u_0^k = \hat{C}x_c^k + \hat{D}u_i^k \tag{2}$$

The number of floating-point operations (FLOPs) required to perform these computations is constant and, hence, will require a fixed amount of CPU time. However, the computational time can be reduced by reducing the order of the controller. It is obvious that the reduced-order controllers must ensure closed-loop stability. It is also expected that they will achieve poorer performance. Thus, it is possible to vary the computational time of linear controllers by executing a reduced-order controller. From the implementation point of view, the available time will dictate the order of the controller that can be implemented. Clearly, the computational time of the lowest-order controller that guarantees robust stability will be the minimum time that must be alloted to the algorithm for the control system to function properly. Therefore, controllers are switched from higher to lower order to accommodate shortage in CPU time. Once the available CPU time increases, the higher controller is switched back. Note that the switching has to be done smoothly to enable bumpless transfer and also with minimal CPU overhead.

The remainder of the paper is organized as follows: First we present preliminaries on complexity analysis of computation. We then present the fact that the computational time of linear controllers can be varied by reducing the order of the controller. This is demonstrated using two model reduction techniques, namely, balanced truncation and residualization. We then propose a switching algorithm, for both balanced and residualized systems, that ensures bumpless transfer with minimal CPU overhead. Construction of the switching algorithm is the main challenge of this research because it should have minimal CPU overhead for feasibility. Proof of stability for the switched linear systems is then presented followed by a numerical example. The paper concludes with a comparison of the proposed anytime control algorithm with traditional anytime algorithms.

II. Complexity Analysis via FLOP Count

The computational overhead of a numerical algorithm is often measured in terms of the number of FLOPs. In this paper we define a FLOP as one addition, subtraction, multiplication, or division of two floating-point numbers. FLOP counts in the early days of computers gave a good estimate of the computation time of an algorithm. This, however, is not true anymore. Features such as cache boundaries and locality of reference of code and data can dramatically affect the speed of computation of numerical algorithms. However, for the purpose of analysis of the switching algorithm presented in this paper, we will assume that the FLOP count of the algorithm gives us a good estimate of its computational overhead.

The numerical algorithms in this paper are mostly matrix–vector multiplications and vector–vector additions. Therefore, for a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $x \in \mathbb{R}^n$, the FLOP count for the computation Ax is $2mn$. Similarly, the computation $x + y$, where $y \in \mathbb{R}^n$ will have a FLOP count of n. We will use these definitions of FLOP counts to analyze the CPU overhead of the switching algorithms presented.

III. Variation of CPU Time for Linear Controllers

Digital implementation of linear state-space control equations of the form

$$x_c^{k+1} = Ax_c^k + Bu_c^k, \qquad y_c^k = Cx_c^k \tag{3}$$

involves a fixed number of scalar multiplications and additions. These computations require a fixed amount of time T_c to complete. It is clear that, if the available CPU time T_{cpu} is less than T_c, the controller will fail to compute an output. It also cannot make use of extra CPU time, if available. Thus, for dynamic scheduling we would require control algorithms to be able to produce output within a range of time, that is, $T_c \in [T_{\min}, T_{\max}]$. In this paper we use linear model reduction techniques, namely, balanced truncation and residualization to generate a set of linear time invariant (LTI) systems with varying run times.

A. Balanced Truncation

If the controller K defined by Eq. (3) is a balanced realization,[24–26] then we can partition the state space as

$$x_c = \begin{Bmatrix} w \\ z \end{Bmatrix} \tag{4}$$

where z represents the weakly controllable and observable states. The controller dynamics in Eq. (3) can be written as

$$\begin{Bmatrix} w^{k+1} \\ z^{k+1} \end{Bmatrix} = \begin{bmatrix} A_{ww} & A_{wz} \\ A_{zw} & A_{zz} \end{bmatrix} \begin{Bmatrix} w^k \\ z^k \end{Bmatrix} + \begin{bmatrix} B_w \\ B_z \end{bmatrix} u_c^k$$

$$y_c^k = [C_w \quad C_z] \begin{Bmatrix} w^k \\ z^k \end{Bmatrix} \tag{5}$$

Note that the states z are weakly observable and controllable; hence, they do not contribute significantly to the controller output y_c in terms of Hankel singular values. A reduced-order model of the controller K can, therefore, be obtained by ignoring these states. The computational time depends on the number of states rejected, that is, the size of z. The dynamics of the reduced-order controller is, therefore,

$$w^{k+1} = A_{ww}w^k + B_w u_c^k, \qquad y_c^k = C_w w^k \tag{6}$$

B. Residualization

In this approach, an LTI system K is decomposed into two systems K_s and K_f such that

$$K(s) = K_s(s) + K_f(s)$$

The system K_s contains the slow modes and K_f contains the fast modes of K. Model reduction is achieved by assuming that the states of K_f are much faster in reaching their equilibrium than those of K_s. Hence, these faster states can be approximated by their steady-state contributions. This implies that the poles of K_f are large compared to those of K_s and are in the left-half plane, that is, stable. Therefore, the transfer function of the reduced order LTI system K_{red} can be written as

$$K_{\text{red}}(s) = K_s(s) + K_f(0)$$

where $K_f(0)$ is the steady-state contribution of $K_f(s)$.

If the dynamics of the LTI system K, in discrete-time, is given by

$$\begin{Bmatrix} x_f^{k+1} \\ x_s^{k+1} \end{Bmatrix} = \begin{bmatrix} A_f & 0 \\ 0 & A_s \end{bmatrix} \begin{Bmatrix} x_f^k \\ x_s^k \end{Bmatrix} + \begin{bmatrix} B_f \\ B_s \end{bmatrix} u_c^k$$

$$y_c^k = [C_f \quad C_s] \begin{Bmatrix} x_f^k \\ x_s^k \end{Bmatrix}$$

then the dynamics of the reduced-order system is

$$x_s^{k+1} = A_s x_s^k + B_s u_c^k, \qquad y_c^k = C_s x_s^k + D_{ss} u_c^k \tag{7}$$

where $D_{ss} = C_f(I - A_f)^{-1}B_f$ is the steady-state contribution of K_f. The computational time of the controller depends on the size of x_s.

C. Variation of Run-Time

If the controller K has order n, then by model reduction, it is possible to generate a set of p controllers $\mathcal{K} = \{K_i\}$, $1 \le i \le p$, each with order $(n - i + 1)$. The computational time of controller $K_i \in \mathcal{K}$, denoted by $T_c(K_i)$, decreases with increasing i, that is, $T_{\max} = T_c(K_1) > \cdots > T_c(K_p) = T_{\min}$. As we reduce the order of the controller, the performance of the closed-loop system may also degrade. However, there is a limit beyond which the stability of the closed-loop system is compromised. Thus, the choice of p should be such that each $K_i \in \mathcal{K}$, guarantees acceptable performance and closed-loop stability.

Once the set $\mathcal{K}$ has been constructed and the execution time of each K_i is known, it is possible to accommodate changes in T_{cpu} by selecting the best controller that can be executed within the alloted time. The best controller is obviously the highest-order controller with $T_c(C_i) \le T_{\mathrm{cpu}}$. Thus, we can accommodate changes in T_{cpu} by switching between controllers of different orders. However, the switching has to be smooth to prevent impulse-like effect in the response of the overall system. A switching scheme with minimal CPU overhead is presented in the next section.

An important aspect of anytime algorithms is the ability to construct their performance profiles and use them according to the available computational times. In controls, the performance of a controller is often measured by the induced norm of the exogenous input on the output. This is measured by the ∞ norm of the closed-loop system. It is expected that the reduced-order controllers will achieve poorer attenuation and, hence, the ∞ norm of the closed-loop system will be larger for lower-order controllers. Therefore, the induced norm of the closed-loop system is a possible metric that can be used to determine the performance profile of the reduced-order controllers.

D. Controller Switching Algorithm with Minimal CPU Overhead

Switching theory of controllers is a well-established area.[27] Unfortunately, many of these ideas cannot be applied directly to this problem because many switching algorithms require simultaneous operation of controllers before switching or require significant CPU time. This is contrary to the motivation of the posed problem, where switching is done to accommodate changes in T_{cpu}. To construct the proposed switching algorithm, we make use of the fact that the states of the controllers $K_i \in \mathcal{K}$ are subsets of the states of the controller K.

There are two cases of switching that might occur. We either switch from a higher- to a lower-order controller, or from a lower- to a higher-order controller. The switching algorithm is different for the two cases. We first present the switching scheme assuming $\mathcal{K} = \{K_1, K_p\}$, that is, we switch between the highest- and the lowest-order controller, then we extend it to the case where $\mathcal{K} = \{K_1, K_2, \ldots, K_p\}$.

Because we consider two model reduction techniques to construct the set $\mathcal{K}$, namely, balanced truncation and residualization, we will represent $\mathcal{K}_{\mathrm{bal}}$ as the the set of controllers obtained using balanced truncation and $\mathcal{K}_{\mathrm{resid}}$ as the set of controllers obtained using residualization.

IV. Switching Algorithm for Balanced Systems

A. Switching from Higher- to Lower-Order Controller

For K_1 and $K_p \in \mathcal{K}_{\mathrm{bal}}$, switching from the full-order controller K_1 in Eq. (5) to the reduced-order controller K_p in Eq. (6) will cause an undesirable impulse effect in the controller output y_c^k. This will be due to the sudden truncation of the z states. For smoothness of y_c^k at the time of switching, it is required that the output and dynamics of both controllers are identical at that time. This can be achieved by modifying the dynamics of the reduced-order controller to

$$w^{k+1} = A_{ww} w^k + A_{wz} z^k + B_w u_c^k, \qquad z^{k+1} = z^k$$
$$y_c^k = C_w w^k + C_z z^k \tag{8}$$

Thus, in the modified reduced-order controller, w and y_c evolve with z held constant, and this preserves the continuity of the controller output at the time of switching. Ideally, we would want to fade the z states to zero so that Eq. (8) transforms into Eq. (6). Unfortunately, the fading out process would require expensive computation.

The term $A_{wz} z^k$ in Eq. (8) is a vector that need not be computed at every time step. Because z is held constant, $A_{wz} z^k$ is a constant vector that needs to be computed only once. Extra computation is required only to add this constant vector. This CPU overhead is a small percentage of the total, especially when the size of z is large. The same argument holds for adding $C_z z^k$ to y_c^k. Therefore,

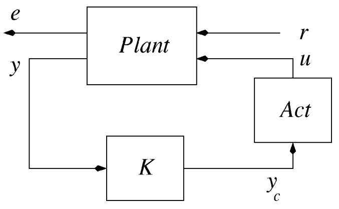

Fig. 1 Closed-loop system with disturbance z.

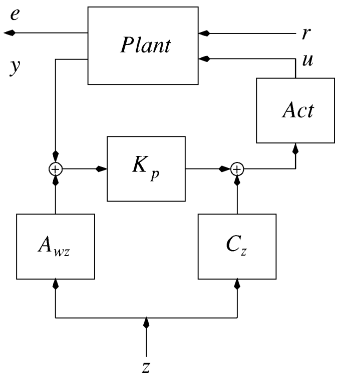

Fig. 2 Closed-loop system with disturbance z.

the addition of terms $A_{wz} z^k$ and $C_z z^k$ to Eq. (8) results in smooth switching from K_1 to K_p and requires minimal CPU overhead.

The constant z vector in Eq. (8) acts as a disturbance at the input and output of the controller. The worst-case effect of this piecewise constant disturbance on the exogenous output and plant input of the closed-loop system can be determined as follows: Let us represent the closed-loop system by Fig. 1, where r, e, y_c, y, and u are the exogenous input, exogenous output, controller output, plant output, and plant input respectively. The disturbance z acts on the system as shown in Fig. 2. The worst-case effect of z on e and z on u is then the ∞ norm of the corresponding transfer functions. Because $\|z\|_2$ is bounded by $\|r\|_2$, a more accurate bound can be obtained by scaling z by $\|G_{rz}\|_\infty$, where G_{rz} is the transfer function from r to z.

The sudden removal of the z dynamics will result in an impulse to the system. Because z states are weakly observable, the effect of its locking on the controller output will be small. It will get more pronounced as more states are locked. This could be removed with the help of a filter that smooths the derivative of the controller output at switching instances. This, however, will require more computation.

B. Switching from Reduced- to Full-Order Controller

Switching from a lower-order controller to a higher-order controller is more complicated. Although K_p is active, the w states have evolved with the z states held constant. In general, if the dynamics of z is suddenly added to the system, there could be large transients in the z trajectory and consequently in the closed-loop response. To minimize and possibly avoid this undesirable effect, we switch to the higher-order controller in two steps.

First z is decayed to zero like a second-order system, with K_p still active. This computation is feasible because we have enough computational time to implement K_1. If k_0 is the time when K_1 switches to K_p, the controller dynamics during this phase is given by

$$w^{k+1} = A_{ww} w^k + A_{wz} z^k + B_w u_c^k, \qquad x_1^{k+1} = x_2^k$$
$$x_2^{k+1} = -\lambda^2 x_1^k - 2\lambda x_2^k, \qquad z^k = z^{k_0} x_1^k$$
$$y_c^k = C_w w^k + C_z z^k \tag{9}$$

where x_1 and $x_2 \in \mathbb{R}$ are the states of the second-order filter and are initialized as $x_1 = 1$ and $x_2 = 0$. Once the z states have decayed to zero, we switch to the higher-order controller K_1.

It is necessary to decay the z states to zero because a reset in this variable will cause transients in the w trajectory and the controller output. This in turn will cause transients to appear in the plant output, which is undesirable. Decaying the z states to zero and then adding the dynamics of z ensures smoothness in w dynamics and the controller output.

The time taken to decay the z states to zero depends on λ. If the z states are decayed too fast, large transients are expected to appear in the plant input and consequently in the closed-loop response. On the other hand, if z is decayed too slowly there will be a large delay in the activation of K_1. Therefore, there is a tradeoff between the transients due to the switching and the delay in the activation of the higher-order controller.

Switching back the dynamics of z, with initial condition $z = 0$, will result in transients appearing in the trajectory of z. Because z is weakly controllable, we expect these transients to be small. Moreover, because z is weakly observable, we expect the effect of these transients on y_c^k to be small as well. However, the transients in z^k and its effect on y_c^k will increase from K_1 to K_p as the observability and controllability of the states z increases. These transients can be removed with a low-pass filter. The implementation of the filter will, however, add computational overhead to the algorithm.

C. Generalization of Algorithm

In this section the switching scheme is generalized for switching between any two controllers of different order. For the purpose of discussion, let us denote w_i and z_i as the w and z states of controller K_i.

In general, for a smooth switching from K_i to K_j, where $1 \geq i > j \geq p$, the vector z_j would include the states from w_i, whose dynamics are ignored in K_j, along with z_i. Thus, the ignored states from w_i are stacked on top of z_i to form z_j.

If the switching occurs from K_j to K_i, where $1 \geq i > j \geq p$, we need to decay the states in z_j, whose dynamics are present in K_i, before switching. Once these states decay to zero, they are stacked below w_j to form w_i. The remaining states in z_j become z_i and remain constant.

D. Proof of Stability

The effect of the switching algorithm on the stability of the closed-loop system is analyzed in this section. In the following analysis, we assume that the closed-loop system as well as the controller are stable systems.

For the purpose of proving stability of the switching logic, we assume that the closed-loop system starts with the highest-order controller, switches to the lowest-order controller, and switches back to the highest-order controller. This sequence may occur repeatedly. Because we do not expect T_{cpu} to fluctuate rapidly, it is reasonable to assume that there are finite switches in finite time.

Let the plant dynamics be given by

$$x_p^{k+1} = A_p x^k + B_p u_p, \qquad y_p^k = C_p x_p \tag{10}$$

where x_p, u_p, and y_p are plant states, input, and output. If the interconnection of the plant and the controller in Eq. (5) is such that $y_p = u_c$ and $y_c = u_p$, then the dynamics of the closed loop with the full-order controller K is

$$\begin{Bmatrix} x_p^{k+1} \\ w^{k+1} \\ \hline z^{k+1} \end{Bmatrix} = \left[\begin{array}{cc|c} A_p & B_p C_w & B_p C_z \\ B_w C_p & A_{ww} & A_{wz} \\ \hline B_z C_p & A_{zw} & A_{zz} \end{array}\right] \begin{Bmatrix} x_p^k \\ w^k \\ \hline z^k \end{Bmatrix} \tag{11}$$

If we denote $v^T = \{x_p \; w\}$, the closed-loop dynamics with the full-order controller can be written as

$$f_0 : \begin{Bmatrix} v^{k+1} \\ z^{k+1} \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{Bmatrix} v^k \\ z^k \end{Bmatrix} \tag{12}$$

where A_{ij} is defined from the partition of the A matrix in Eq. (11). The closed loop with controller from Eqs. (8) and (9) are

$$f_1 : \begin{Bmatrix} v^{k+1} \\ z^{k+1} \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ 0 & I \end{bmatrix} \begin{Bmatrix} v^k \\ z^k \end{Bmatrix} \tag{13}$$

For the purpose of proving stability, let us assume that z decays as $z^{k+1} = A_{zz} z^k$. Therefore, the closed-loop system, while z is decaying, is given by

$$f_2 : \begin{Bmatrix} v^{k+1} \\ z^{k+1} \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} \begin{Bmatrix} v^k \\ z^k \end{Bmatrix} \tag{14}$$

Thus, we have three stable systems, f_0, f_1, and f_2, and a switching sequence that switches between them. We use multiple Lyapunov function approach (see Ref. 28) to prove stability of this switched linear system.

The switching sequence S for this switched linear system is

$$S = x_0, (f_0, T_0), (f_1, T_1), (f_2, T_2), (f_0, T_3), \ldots \tag{15}$$

which means that this hybrid system starts at time T_0, with initial condition $x_0 = \{v_0 \; z_0\}^T$ and dynamics given by f_0. At time T_1, the system switches to f_1, and so on. Thus, the system f_i is active in the time intervals $\mathcal{I}(i)$ given by

$$\mathcal{I}(i) \in \{[T_i, T_{i+1}), [T_{i+3}, T_{i+4}), \ldots, [T_{i+3j}, T_{i+3j+1}), \ldots\} \tag{16}$$

where $j \in \mathbb{Z}^*$ and $\mathbb{Z}^*$ is the set of nonnegative integers. Define $\mathcal{E}(i)$ as the set of times when system f_i is switched to, that is,

$$\mathcal{E}(i) = \{T_i, T_{i+3}, \ldots, T_{i+3j}, \ldots\}, \qquad j \in \mathbb{Z}^* \tag{17}$$

Definition (definition 2.2 in Ref. 28): Given a strictly increasing sequence of times $\mathcal{T} = \{t_k\}, k \in \mathbb{Z}^*$, we say that $V_i(x^k)$ is a Lyapunov-like function for system f_i and trajectory $x^k = \{v^k \; z^k\}^T$ over $\mathcal{T}$ if the following conditions hold.

1) V_i is a positive definite, continuous function about the origin (zero).

2) $V_i(x^{k+1}) \leq V_i(x^k), \forall t \in \mathcal{I}(i)$.

3) V_i is monotonically nonincreasing on $\mathcal{E}(i)$.

Denoting x_S^k as the state trajectory of the closed-loop system under switching sequence S, Branicky[28] shows that, if for S we have that for all i, V_i is Lyapunov like for f_i and x_S^k over $\mathcal{I}(i)$, then the system is stable in the sense of Lyapunov. We now show that our switched linear system satisfies this.

Because f_0, f_1, and f_2 are stable systems, there exists positive definite matrices P_0, P_1, and P_2 such that

$$V_i(x) = x^T P_i x, \qquad i = 0, 1, 2 \tag{18}$$

that are candidate Lyapunov functions for f_0, f_1, and f_2 and

$$V_i(x^{k+1}) < V_i(x^k), \qquad i = 0, 1, 2 \tag{19}$$

To prove that V_i is monotonic nonincreasing in $\mathcal{E}(i)$, let us denote x_r as the system states at switching times T_r. Because f_0, f_1, and f_2 are exponentially stable autonomous systems and the switching does not cause any impulsive jumps in the state trajectories, we can claim that

$$\begin{aligned} &\|x_r\|_2 \geq \|x_{r+1}\|_2, \qquad r \in \mathbb{Z}^* \\ &\Rightarrow \|x_r\|_2 \geq \|x_{r+3j}\|_2, \qquad r \in \mathbb{Z}^*, \qquad j \in \mathbb{Z}^+ \end{aligned} \tag{20}$$

where $\mathbb{Z}^+$ denotes positive integers. Therefore, V_i in Eq. (18) is nonincreasing in $\mathcal{E}(i)$. This completes the proof on the stability of the switched linear system in context.

The switching scheme is next applied to a B737-100 transport system research vehicle linear longitudinal motion model. The aircraft model has four states, longitudinal velocity V (feet per second) angle-of-attack α (radians), pitch rate q (radians per second), and pitch angle θ (radians); two control inputs, thrust T (pounds) and elevator deflection δ_e (degrees). The elevator actuator and the engine are modeled as $16/(s+16)$ and $20/(s^2+12s+20)$, respectively. The control objective is to achieve decoupled tracking response of V and γ reference signals. The controller was designed using $\mathcal{H}_\infty$ theory and has 18 states.[29]

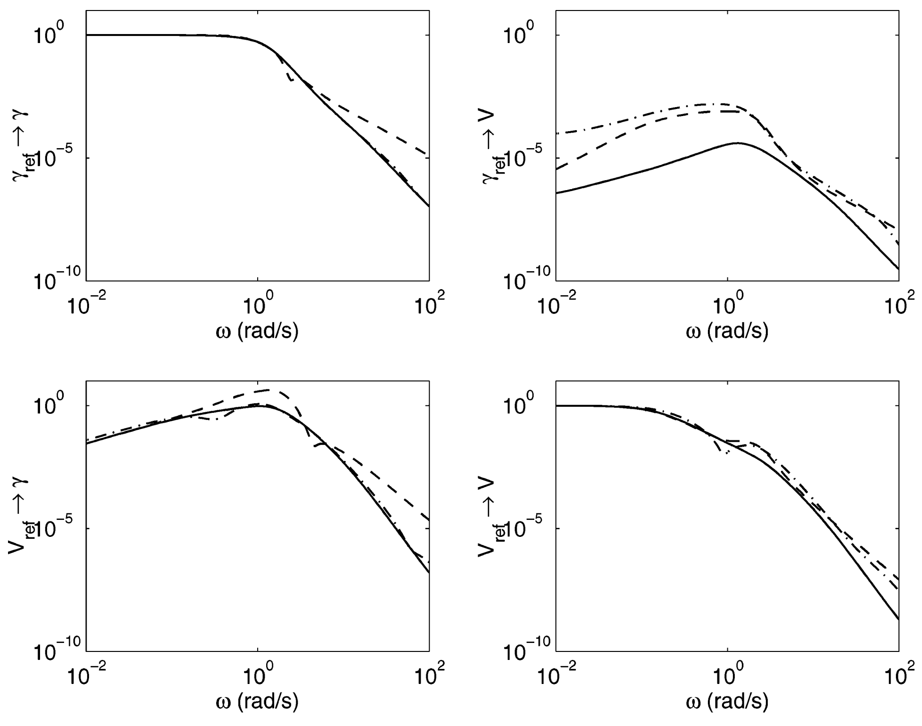

Fig. 3 Frequency response: ——, $(\gamma_{ref}, V_{ref}) \to (\gamma, V)$ for full-order controller; —·—·, $K_p \in \mathcal{K}_{bal}$; and – – –, $K_p \in \mathcal{K}_{resid}$.

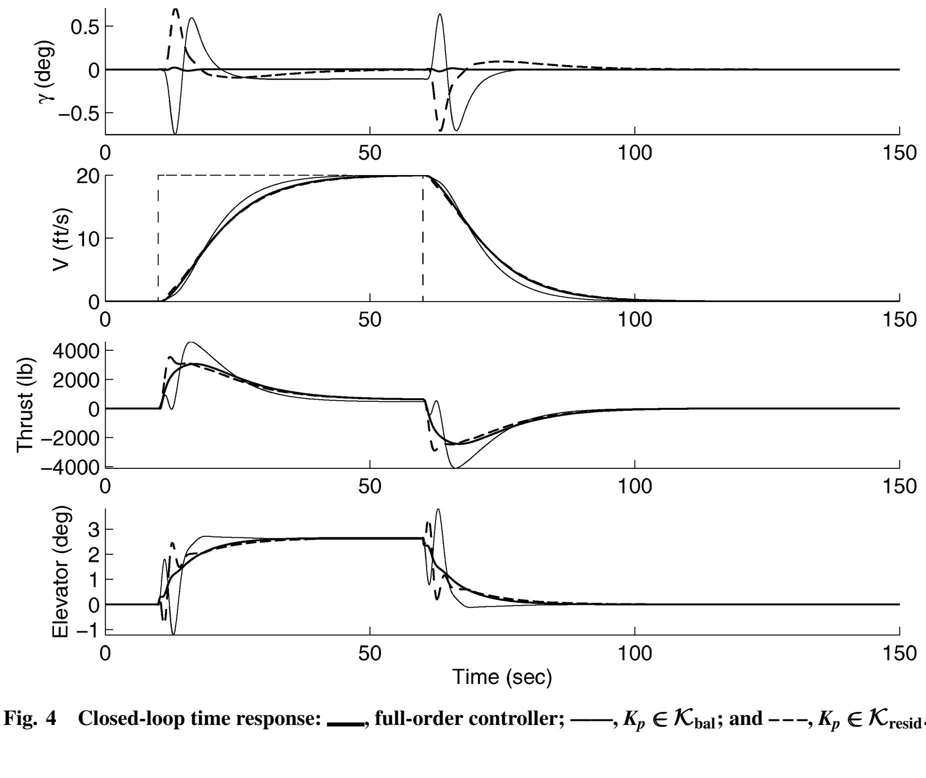

Fig. 4 Closed-loop time response: ▬, full-order controller; ——, $K_p \in \mathcal{K}_{bal}$; and – – –, $K_p \in \mathcal{K}_{resid}$.

E. Example

Here we study the effect of the switching algorithm on the B737 model with controllers K_1 and K_p obtained using balanced trucation. The lowest-order controller that guarantees stability of the closed loop with acceptable performance has 13 states. Therefore, for this example we will be switching between a 18 state controller K_1 and a 13 state controller K_p.

In Fig. 3, we present the frequency response of the four transfer functions,

$$\begin{Bmatrix} \gamma \\ V \end{Bmatrix} = \begin{bmatrix} G_{\gamma\gamma_{ref}} & G_{\gamma V_{ref}} \\ G_{V\gamma_{ref}} & G_{VV_{ref}} \end{bmatrix} \begin{Bmatrix} \gamma_{ref} \\ V_{ref} \end{Bmatrix} \tag{21}$$

Let us represent the closed-loop system in Eq. (21), with controller K_i, as P_i. From the frequency-magnitude plots in Fig. 3 we observe that there is a small difference in the performance of P_1 and P_p. The two controllers achieve desired decoupling between γ and V, in response to reference input. The induced norm of P_1 and P_p are 1.12 and 1.27, respectively. Therefore, in terms of the induced norms, there is not much degradation in the performance of the reduced-order controller.

Time response of the closed-loop system to a step command of 20 ft/s in V, with the full-order controller, is also shown in Fig. 4. Note that there is no γ response and the control actions are smooth. The full-order controller achieves decoupled response to V and γ reference signals as desired.

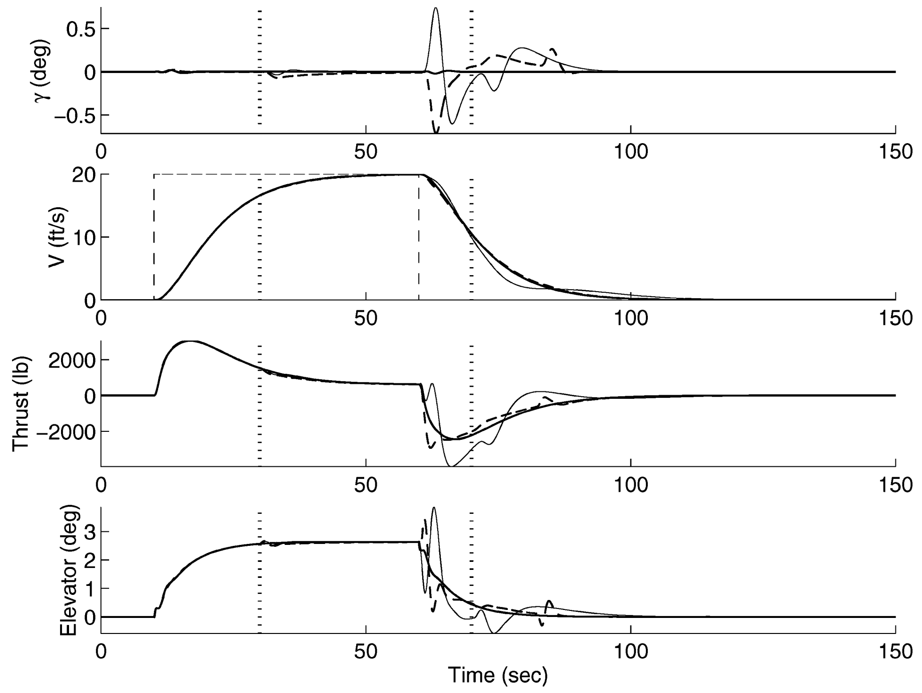

Fig. 5 Closed-loop time response with switching of controllers: ▬, full-order controller; ——, $K_p \in \mathcal{K}_{\text{bal}}$; and – – –, $K_p \in \mathcal{K}_{\text{resid}}$.

The closed-loop response to the same V command, with the lower-order controller (without constant z states), is shown in Fig. 4. We observe degradation in performance as the velocity command induces a γ response, with peak amplitude of approximately 0.7 deg.

In terms of the computational time, the 18 state controller takes 1033 MATLAB® FLOPs and the 13 state controller takes 648 FLOPs. The overhead of adding constant vectors in K_p is 13 FLOPs (2%). Therefore, in this example, we can accomodate approximately 37% drop in T_{cpu} by implementing the lower-order controller. The CPU overhead for ensuring smooth switching is also minimal.

Figure 5 shows the closed-loop response with the switching algorithm incorporated. In Fig. 5, the simulation starts with the full-order controller. At 30 s, T_{cpu} falls to 67% and the controller switches K_p. The lower-order controller tracks the reference signal till 70 s. Note the coupled response in γ and V tracking during this time. At 70 s, T_{cpu} comes back to 100%, and the z states begin to decay. Once the z states have decayed to zero, the full-order controller is switched back. This happens at 112.23 s. Thus, the dynamics of the controller is changed at times $t = 30, 70$, and 112.23 s. We observe from the trajectories that there are no impulselike transients at these times due to the switching. The oscillations in $\gamma(t)$, in Fig. 5, are because of the lower-order controller operating at that time. The dotted vertical lines in the state trajectories denote the times when T_{cpu} changes.

In the simulation for this example, we chose $\lambda = 0.2$. With this value of λ, it takes 42.23 s for the z states to decay to zero from its value at $t = 30$ s. Depending on the application, this may be too long to recover the higher-order controller, and so λ has to be chosen accordingly. Because we are dealing with a transport aircraft model, this blending time may be acceptable. We see that there are small transients in $[\gamma \;\; V \;\; \delta_e \;\; T]$, occuring between 70 and 112.23 s, due to the decaying of z. If a faster decay of z is desired, then it will cause transients of greater magnitude to occur in the closed-loop response. Also, note that the sudden elimination of z dynamics at time 30 s results in small transients in the elevator angle at 30 s. Furthermore, the addition of the z dynamics at time 112.23 s does not induce any transients in the closed-loop response.

Thus, the simulation in Fig. 5, shows that the proposed transformation of the designed controller K to a switched linear system enables the control algorithm to execute in an environment of dynamically scheduled CPU time.

V. Switching Algorithm for Residualized Systems

A. Switching from Higher- to Lower-Order Controller

If K_1 and $K_p \in \mathcal{K}_{\text{resid}}$, at the time of switching K_1 in Eq. (7) and K_p in Eq. (7) will have different outputs in general. This difference will cause an impulselike behavior in the close-loop response, which is undesirable. Therefore, it is necessary to modify the output of K_p so that the outputs of both the systems are identical at the time of switching. This can be achieved by adding the difference between the output of K_1 and K_p, to the output of K_p at the time of switching.

Therefore, if k_0 is the time of switching, the error between the output of the two controllers is

$$\Delta y_c^{k_0} = y_c^{k_0}(K_1) - y_c^{k_0}(K_p)$$

where $y_c^{k_0}(K_i)$ is the output of the controller K_i at time k_0. Note that the only additional computation involved is the subtraction of two vectors. The output of K_1 is already known from the previous time step, and the output of K_p is computed in the present time step. This subtraction is also done only once at the time of switching.

The definition of K_p then modifies to

$$x_s^{k+1} = A_s w^k + B_s u_c^k, \qquad x_f^{k+1} = x_f^k$$
$$y_c^k = C_s x_s^k + D_{ss} u_c^k + \Delta y_c^{k_0} \tag{22}$$

The difference in the controller output at the time of switching is a result of the difference between x_f and its steady-state value, denoted by $\bar{x}_f$. This can be shown as follows:

$$\begin{aligned}\Delta y_c^{k_0} &= \left(C_f x_f^{k_0} + C_s x_s^{k_0}\right) - \left(C_s x_s^{k_0} + D_{ss} u_c^{k_0}\right)\\ &= \left(C_f x_f^{k_0} - D_{ss} u_c^{k_0}\right)\\ &= C_f x_f^{k_0} - C_f (I - A_f)^{-1} B_f u_c^{k_0}\\ &= C_f\left(x_f^{k_0} - \bar{x}_f^{k_0}\right)\end{aligned} \tag{23}$$

Therefore, if $x_f = \bar{x}_f$ at the time of switching, there will no difference between the outputs of K_1 and K_p.

The vector $\Delta y_c^{k_0}$ in Eq. (22) can be treated as a piecewise constant disturbance, added at the output of K_p as shown in Fig. 6. Its worst

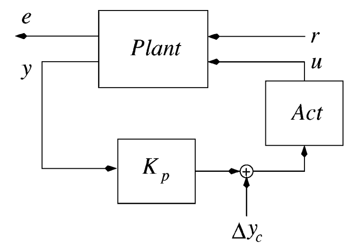

Fig. 6 Closed-loop system with decaying disturbance Δy_c.

case effect on e and u can be quantified by the ∞ norm of the corresponding transfer functions.

Because we equate the controller output over two consecutive time steps, it causes the controller output to be constant over this interval. This happens every time the controller switches from a higher-order system to a lower-order system.

When residualization is used as a model reduction tool to generate K_1 and K_p, the dynamics of x_s and x_f are decoupled. Therefore, the sudden elimination of x_f dynamics does not affect on the dynamics of x_s. This, however, causes a sudden change in the derivative of y_c^k, which in turn causes transients to appear in the closed-loop system response. These transients can be removed with a low-pass filter at the cost of added computation.

B. Switching from Reduced- to Full-Order Controller

When we switch from K_p to K_1, the output of both the systems at the time of switching must be identical. If we denote k_1 as the time when K_p switches to K_1, the difference in the controller output at k_1 is

$$\begin{aligned}\Delta y_c^{k_1} &= \left(C_s x_s^{k_1} + D_{ss} u_c^{k_1} + \Delta y_c^{k_0}\right) - \left(C_s x_s^{k_1} + C_f x_f^{k_1}\right)\\ &= \left(D_{ss} u_c^{k_1} - C_f x_f^{k_1}\right) + \Delta y_c^{k_0}\end{aligned}$$

The difference in the controller output can be made zero by decaying $\Delta y_c^{k_0}$ to zero before switching. The difference in the controller output then becomes

$$\Delta y_c^{k_1} = \left(D_{ss} u_c^{k_1} - C_f x_f^{k_1}\right)$$

This difference can be reduced to zero by initializing x_f as $x_f^{k_1} = -A_f^{-1} B_f u_c^{k_1}$, where k_1 is the time when $\Delta y_c^{k_0}$ reaches zero. The dynamics of the controller when $\Delta y_c^{k_0}$ is decaying to zero is

$$x_s^{k+1} = A_s x_s^k + B_s u_c^k, \qquad x_f^{k+1} = x_f^k, \qquad x_1^{k+1} = x_2^k$$

$$x_2^{k+1} = -\lambda^2 x_1^k - 2\lambda x_2^k, \qquad y_c^k = C_s x_s^k + D_{ss} u^k + x_1^k \Delta y_c^{k_0} \quad (24)$$

where the states x_1 and $x_2 \in \mathbb{R}$ are the states of the second-order filter with initial condition $x_1 = 1$, and $x_2 = 0$ at time $k = k_0$. The filter is used to smoothly decay the contribution of $\Delta y_c^{k_0}$ in the output of the controller.

Once $\Delta y_c^{k_0}$ reaches zero, x_f is initialized with its steady-state value based on the current input, and the controller switches to K_1. The activation of K_1 is delayed by the time it takes $\Delta y_c^{k_0}$ to reach zero. Therefore, it is desirable that the decay rate of $\Delta y_c^{k_0}$, determined by λ in Eq. (24), is fast. At the same time, λ has to be such that the transients in e and u, because of changing $\Delta y_c^{k_0}$, should be small. The transients in e and u will be small for slowly decaying $\Delta y_c^{k_0}$; however, it will delay the switching time of K_1. Therefore, there is a tradeoff between the transients in e and u and the delay in the activation of K_1.

If K is in modal form, then it can be written as

$$K = \Gamma_1 + \Gamma_2 + \cdots + \Gamma_n$$

where Γ_i is the ith modal system. If the modes of Γ_i are faster than the modes of Γ_j, for $i > j$, then K_p in discrete time is defined as

$$K_p(z) = \sum_{i=1}^{i=p} \Gamma_i(z) + \sum_{i=p+1}^{i=n} \Gamma_i(1)$$

where $\Gamma_i(1)$ is the steady state contribution of $\Gamma_i(z)$. Therefore, x_f is the state vector of

$$\sum_{i=p+1}^{i=n} \Gamma_i(z)$$

The corresponding matrix $(-A_f^{-1} B_f)$ can, therefore, be written as

$$A_f^{-1} B_f = \begin{bmatrix} A_{p+1}^{-1} B_{p+1} \\ A_{p+2}^{-1} B_{p+2} \\ \vdots \\ A_n^{-1} B_n \end{bmatrix} \quad (25)$$

where A_i and B_i are the A and B matrices of the modal system Γ_i. Therefore, if $A_i^{-1} B_i$ is computed off-line, then the computational overhead in initializing x_f is only that required to multiply the matrix in Eq. (25) with vector u_c. This computation is feasible because it is equivalent to the computation $B_f u_c$. The storage of $A_i^{-1} B_i$ for $i \in \{1, 2, \ldots, n\}$ increases the memory overhead of the implementation.

Switching back the dynamics of x_f from its steady state will cause transients to appear in the output of the controller. These transients can be removed with the help of a low-pass filter. However, implementation of the filter will add computational overhead.

Because $D_{ss} u_c$ is equal to $C_f x_f$ at the time the controller switches to K_1, the output of the controller is identical over two consecutive time steps. This happens every time the controller switches from a lower-order system to a higher-order system.

Depending on the dimension of x_f, u_c, and y_c, the computational overhead incurred to smoothly decay $\Delta y_c^{k_0}$ may exceed the computational overhead of K_1, which may be undesirable. If n_{s_f}, n_{s_s}, n_u, and n_y are the dimensions of x_f, x_s, u_c, and y_c, respectively, then the FLOP count of K_1 is given by

$$F_1 = 2n_{s_s}\left(n_{s_s} + n_u\right) + 2n_{s_f}\left(n_{s_f} + n_u\right) + 2n_y\left(n_{s_s} + n_{s_f}\right)$$

The FLOP count for the decay process is

$$F_2 = 2n_{s_s}\left(n_{s_s} + n_u\right) + 2n_y\left(n_{s_s} + n_u + 1\right) + 4$$

Therefore, if the computation of the decay process is feasible, then $F_1 - F_2 \geq 0$

$$\Rightarrow n_{s_f}^2 + n_{s_f}(n_u + n_y) - (n_y n_u + n_y + 2) \geq 0$$

Because $n_{s_f} \in \mathbb{Z}^*$, the minimum value of n_{s_f} can be given by

$$\min n_{s_f} = \left\lceil \tfrac{1}{2}\left\{\sqrt{(n_y + n_u)^2 + 4(n_y n_u + n_y + 2)} - (n_u + n_y)\right\}\right\rceil$$

C. Generalization of the Switching Algorithm

In this section, we generalize the switching scheme for any two systems K_i and $K_j \in \mathcal{K}_{\text{resid}}$. For the purpose of discussion, let us denote x_{f_i}, x_{s_i} as the x_f and x_s states of controller K_i. Let us also denote Δy_{c_i} as the Δy_c of K_i. The superscript of Δy_c has been dropped for notational convenience. We will also assume that the controllers in $\mathcal{K}_{\text{resid}}$ are arranged in the decreasing order, of the order of the controller. That is, if $i < j$, it implies the order of controller K_i is higher than that of controller K_j. In general, K_i is defined as

$$x_{s_i}^{k+1} = A_{s_i} x_{s_i}^k + B_{s_i} u_c^k, \qquad K_i : x_{f_i}^{k+1} = x_{f_i}^k$$

$$y_{c_i}^k = C_{s_i} x_{s_i}^k + D_{ss_i} u^k + \Delta y_{c_i}$$

When switching from a higher-order controller to a lower-order controller, that is, K_i to K_j, where $1 \leq i < j \leq p$, some states from x_{s_i} will be residualized. Let those states be $\hat{x}_{s_j}$. Therefore,

$$x_{s_i} \equiv \begin{Bmatrix} x_{s_j} \\ \hat{x}_{s_j} \end{Bmatrix}, \qquad x_{f_j} \equiv \begin{Bmatrix} \hat{x}_{s_j} \\ x_{f_i} \end{Bmatrix}$$

and the difference in the output of the two controllers is

$$\Delta y_{c_j} = \left(y_{c_i}^k - C_{s_j} x_{s_j}^k - D_{ss_j} u_c^k\right)\Big|_{k=k_0}$$

where k_0 is the time K_i switches to K_j.

When switching from a lower-order controller to a higher-order controller, that is, from K_j to K_i, $i < j$, a subset of x_{f_j} will start evolving and affect the output of K_i. Let those states be $\hat{x}_{f_i}$. Therefore,

$$x_{f_j} \equiv \begin{Bmatrix} \hat{x}_{f_i} \\ x_{f_i} \end{Bmatrix}, \qquad x_{s_i} \equiv \begin{Bmatrix} x_{s_j} \\ \hat{x}_{f_i} \end{Bmatrix}$$

The difference in the controller output Δy_{c_i} can be written as

$$\begin{aligned} \Delta y_{c_i} &= \left(C_{s_j} x_{s_j}^k + D_{ss_j} u_c^k + \Delta y_{c_j}\right) - \left(C_{s_i} x_{s_i}^k + D_{ss_i} u_c^k\right) \\ &= C_{s_j} x_{s_j}^k + D_{ss_j} u_c^k + \Delta y_{c_j} - C_{s_j} x_{s_j}^k - \hat{C}_{f_i} \hat{x}_{f_i}^k - D_{ss_i} u_c^k \\ &= \left(D_{ss_j} - D_{ss_i}\right) u_c^k - \hat{C}_{f_i} \hat{x}_{f_i}^k + \Delta y_{c_j} \end{aligned}$$

If the controller K is in modal form, that is,

$$K := \left[\begin{array}{cccc|c} A_n & \cdots & 0 & 0 & B_n \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \cdots & A_2 & 0 & B_2 \\ 0 & \cdots & 0 & A_1 & B_1 \\ \hline C_n & \cdots & C_1 & C_2 & 0 \end{array}\right]$$

then the A matrix of K is block diagonal and D_{ss_j} can be simplified as

$$\begin{aligned} D_{ss_j} &= C_{f_j}\left(I - A_{f_j}\right)^{-1} B_{f_j} \\ &= \begin{bmatrix} \hat{C}_{f_i} & C_{f_i} \end{bmatrix} \begin{bmatrix} \left(I - \hat{A}_{f_i}\right)^{-1} & 0 \\ 0 & \left(I - A_{f_i}\right)^{-1} \end{bmatrix} \begin{bmatrix} \hat{B}_{f_i} \\ B_{f_i} \end{bmatrix} \\ &= \hat{C}_{f_i}\left(I - \hat{A}_{f_i}\right)^{-1} \hat{B}_{f_i} + C_{f_i}\left(I - A_{f_i}\right)^{-1} B_{f_i} \\ &= \hat{D}_{ss_i} + D_{ss_i} \end{aligned}$$

Therefore,

$$\begin{aligned} \Delta y_{c_i} &= \left(D_{ss_j} - D_{ss_i}\right) u_c^k - \hat{C}_{f_i} \hat{x}_{f_i}^k + \Delta y_{c_i} \\ &= \hat{D}_{ss_i} u_c^k - \hat{C}_{f_i} \hat{x}_{f_i}^k + \Delta y_{c_j} \end{aligned} \tag{26}$$

Hence, when switching from K_j to K_i, first the vector Δy_{c_j} needs to be decayed to zero. Then, the states $\hat{x}_{f_i}$ are initialized to their steady-state values based on the current input, that is, $\hat{x}_{f_i} = -\hat{A}_{f_i}^{-1} \hat{B}_{f_i} u$ and the controller K_j switches to K_i.

D. Proof of Stability

When K_1 and $K_p \in \mathcal{K}_{\text{resid}}$, the closed-loop system switches between three systems f_0, f_1, and f_2. They are constructed with controllers defined in Eqs. (7), (22), and (24), respectively. If the plant-actuator model in Fig. 1 is represented as

$$\begin{bmatrix} x_p^{k+1} \\ \hline e^k \\ y^k \end{bmatrix} = \left[\begin{array}{c|cc} A_p & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] \begin{bmatrix} x_p^k \\ \hline r^k \\ y_c^k \end{bmatrix}$$

the output of the plant in the absence of external input r^k and assuming $D_{22} = 0$ can be written as $y^k = C_2 x_p^k$. For the purpose of proving stability, let us assume that Δy_c^k in Eq. (24) decays as $\Delta y_c^{k+1} = A_d \Delta y_c^k$ and treat it as another state of the closed-loop system, where A_d is a matrix with eigenvalues within the unit circle.

Therefore, the closed-loop systems f_0, f_1, and f_2, with Δy_c as an adjoint state can be written as

$$f_0: \begin{Bmatrix} x_p \\ x_s \\ x_f \\ \Delta y_c \end{Bmatrix}^{k+1} = \begin{bmatrix} A_p & B_2 C_s & B_2 C_f & 0 \\ B_s C_2 & A_s & 0 & 0 \\ B_f C_2 & 0 & A_f & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} x_p \\ x_s \\ x_f \\ \Delta y_c \end{Bmatrix}^k \tag{27}$$

$$f_1: \begin{Bmatrix} x_p \\ x_s \\ x_f \\ \Delta y_c \end{Bmatrix}^{k+1} = \begin{bmatrix} A_p & B_2 C_s & B_2 C_f & B_2 \\ B_s C_2 & A_s & 0 & 0 \\ B_f C_2 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \begin{Bmatrix} x_p \\ x_s \\ x_f \\ \Delta y_c \end{Bmatrix}^k \tag{28}$$

$$f_2: \begin{Bmatrix} x_p \\ x_s \\ x_f \\ \Delta y_c \end{Bmatrix}^{k+1} = \begin{bmatrix} A_p & B_2 C_s & B_2 C_f & B_2 \\ B_s C_2 & A_s & 0 & 0 \\ B_f C_2 & 0 & I & 0 \\ 0 & 0 & 0 & A_d \end{bmatrix} \begin{Bmatrix} x_p \\ x_s \\ x_f \\ \Delta y_c \end{Bmatrix}^k \tag{29}$$

As in Sec. IV.D, let us assume that the switching sequence S for this switched linear system is $S = \{x_0, (f_0, T_0), (f_1, T_1), (f_2, T_2), (f_0, T_3), \ldots\}$. The systems f_0, f_1, and f_2 are stable systems, with discontinuous jumps in x_f and Δy_c at times $\{T_0, T_3, \ldots\}$ and $\{T_1, T_4, \ldots\}$, respectively.

We will prove the stability of the switched linear system, with discontinuous jumps in x_f and Δy_c, using multiple Lyapunov functions. Let us define V_i to be the Lyapunov function for closed-loop system f_i. Therefore, as in Sec. IV.D, we have to prove V_i is nonincreasing in $\mathcal{E}(i)$. In the proof we represent time by k. Therefore, the set $\mathcal{E}(i)$ in Eq. (17) is defined as

$$\mathcal{E}(i) = \{k_i, k_{i+3}, \ldots, k_{i+3j}, \ldots\}, \qquad j \in \mathbb{Z}^* \tag{30}$$

Because the three systems, f_0, f_1, and f_2, are stable and there are no discontinuous jumps in x_p and x_s, the following is true:

$$\left\| \begin{Bmatrix} x_p \\ x_s \end{Bmatrix} \right\|_2^{k+1} \le \left\| \begin{Bmatrix} x_p \\ x_s \end{Bmatrix} \right\|_2^k, \qquad \forall k \tag{31}$$

If x_f is initialized at time k_{3j} as

$$x_f^{k_{3j}} = (I - A_f)^{-1} B_f C_2 x_p^{k_{3j}}$$

$$\Rightarrow \left\| x_f^{k_{3j}} \right\|_2 \le \left\| (I - A_f)^{-1} B_f C_2 \right\|_\infty \left\| x_p^{k_{3j}} \right\|_2 = \gamma_1 \left\| x_p^{k_{3j}} \right\|_2$$

Because f_0 is stable,

$$\gamma_1 \left\| x_p^{k_{3j}} \right\|_2 \ge \left\| x_f^k \right\|_2, \qquad k_{3j} \le k < k_{3j+1}$$

Therefore, the Lyapunov function for f_0 can be defined as

$$V_0\left(x_p^k, x_s^k\right) = \left\| x_p^k \right\|_2^2 + \left\| x_s^k \right\|_2^2 + \gamma_1 \left\| x_p^{k_{3j}} \right\|_2$$

$$k_{3j} \le k < k_{3j+1}, \qquad j \in \mathbb{Z}^+ \tag{32}$$

Therefore, from Eq. (31), we can conclude that $V_0(x_p^k, x_s^k)$ is nonincreasing in $\mathcal{E}(0)$.

To show V_1 is nonincreasing in $\mathcal{E}(1)$, consider the following: Because x_f^k is constant in $k \in [k_{3j+1}, k_{3j+2})$, f_0 is stable and there are no discontinuous jumps in x_f while switching from f_0 to f_1, the following is true:

$$\gamma_1 \left\| x_p^{k_{3j}} \right\|_2 \ge \left\| x_f^k \right\|_2, \qquad k_{3j+1} \le k < k_{3j+2}$$

The discontinuous jump in Δy_c^k is defined as

$$\Delta y_c^k = C_f x_f^k - C_f (I - A_f)^{-1} B_f C_2 x_p^k$$

$$\Rightarrow \left\| \Delta y_c^k \right\|_2 \le \|C_f\|_\infty \left\| x_k^k \right\|_2 + \left\| C_f (I - A_f)^{-1} B_f C_2 \right\|_\infty \left\| x_p^k \right\|_2$$

$$\le \gamma_1 \|C_f\|_\infty \left\| x_p^{k_{3j}} \right\|_2 + \left\| C_f (I - A_f)^{-1} B_f C_2 \right\|_\infty \left\| x_p^k \right\|_2$$

$$= \gamma_2 \left\| x_p^{k_{3j}} \right\|_2 + \gamma_3 \left\| x_p^k \right\|_2$$

Therefore, the Lyapunov function for f_1 can be defined as

$$V_1\left(x_p^k, x_s^k\right) = (1 + \gamma_3) \left\| x_p^k \right\|_2^2 + \left\| x_s^k \right\|_2^2 + \gamma_2 \left\| x_p^{k_{3j}} \right\|_2$$

$$k_{3j+1} \le k < k_{3j+2}, \qquad j \in \mathbb{Z}^+$$

and from Eq. (31), we can conclude that $V_1(x_p^k, x_s^k)$ is nonincreasing in $\mathcal{E}(1)$.

When f_2 is active, x_f^k is constant and Δy_c^k decays to zero. The bounds on x_f^k and Δy_c^k for f_1 are also valid for f_2. Therefore, the Lyapunov function defined for f_1 can also be the Lyapunov function for f_2, and it can be shown similarly that it is nonincreasing in $\mathcal{E}(2)$.

This completes the stability proof of the switched linear system in context.

E. Example

The effect of the switching algorithm on the B737 model, with controllers K_1 and K_p obtained using residualization, is presented in this section. The lowest-order controller obtained using residualization that guarantees stability of the closed-loop system has 10 states. Therefore, in this case we will be switching between a 18-state controller K_1 and a 10-state controller K_p. Note that $K_p \in \mathcal{K}_{\text{resid}}$ has fewer number of states than $K_p \in \mathcal{K}_{\text{bal}}$.

Let us represent the closed-loop system in Eq. (21) with controller K_i as P_i. The frequency response of the transfer functions, defined by Eq. (21), with K_1, and $K_p \in \mathcal{K}_{\text{resid}}$ is shown in Fig. 3. From Fig. 3, we see that P_p is different from P_1 at high frequencies. The induced norm of the closed-loop system P_p is 4.29, which is quite large compared to 1.12, the induced norm of P_1.

The closed-loop response to a step command of 20 ft/s in V is shown in Fig. 4. In comparison with that of K_1, we observe a degradation in the performance of the reduced-order controller. A velocity step command induces a significantly larger γ response and causes oscillatory control actions in both thrust and elevator.

With the switching algorithm implemented, the time response to the same velocity step command is shown in Fig. 5. As in the case for K_1, and $K_p \in \mathcal{K}_{\text{resid}}$, the controller starts with K_1 and switches to K_p at time 30 s. The computational resources come back to 100%, and Δy_c starts to decay. It reaches zero at time 82.4 s and the controller switches back to K_1 with appropriately initialized x_f. For this example, the saving in computational overhead is approximately 59%.

We observe noticeable transients, especially in the elevator angle, at times when the controller dynamics changes. The transient at 30 s is because of the sudden disappearance of x_f dynamics, the transient at 70 s is induced by the decaying of Δy_c, and the transient at 82.4 s is due to the transients in x_f when it begins to evolve from steady state under the action of u_c.

The value of $\lambda = 0.4$ was chosen to decay Δy_c for this implementation. A smaller value of λ will reduce the transients at 70 s and K_1 will take effect earlier. Note that minimizing this transient does not require any additional computation. However, reduction of the transients at 30 and 82.4 s will require implementation of appropriately defined filters, which will add computational overhead.

VI. Comparison with Anytime Algorithms

In this paper we have presented a method to transform linear controllers to algorithms that can accommodate changes in the CPU time allotted to compute the controller output. The behavior of the transformed controller is similar to anytime algorithms where the accuracy is traded off for computational time. The differences and similarities between the transformed controller algorithm and anytime algorithms can be determined as follows. In general, an anytime algorithm has the following properties:

1) Anytime algorithms return a result with a measure of quality. The measure of quality for an anytime control algorithm could be defined as the infinity norm of the transfer function from disturbance to error. If the performance of the controller degrades from higher to lower order, larger value of this metric would indicate poorer quality of result.

2) Anytime algorithms can be preempted anytime. Anytime control algorithms, as defined in this paper, cannot be preempted once started. The available CPU time must be known before execution of the algorithm. Once the available time is known, the complexity of the algorithm is chosen so that its run time is less than allotted CPU time. To obtain a valid result, the algorithm must execute till completion. Such anytime algorithms have been studied in the past under the term contract algorithms.[30,31]

3) Increasing CPU time at run-time increases quality of result. The anytime control algorithm, as defined, cannot make use of extra CPU time that becomes available during run time. This feature would require the algorithm to switch to the higher-order controller while it is implementing a lower-order controller. Such a transition may be possible but has not been considered in this research work.

4) Anytime algorithms always improve output quality as they are given more time. There is a upper limit on the performance of the anytime control algorithm, and it is equal to that of the highest-order controller. This property of anytime algorithms is due to the iterative nature of the computation. Because we achieve anytime behavior of control algorithms by switching controllers of different order, this feature is not as pronounced as it is in iterative algorithms.

Therefore, the anytime control algorithms proposed in this paper exhibit the most salient feature of anytime algorithms, namely the tradeoff between quality of output and computation time. It, however, has limitations in some of the other anytime properties.

VII. Summary

In this paper, we present a method to transform linear controllers to behave similarly as anytime control algorithms. That is, they are able to trade the quality of solution for reduced computational time. The transformation is achieved by generating a set of reduced-order controllers via model reduction and implementing a switching algorithm that smoothly switches between these controllers based on the available computational time.

We consider balanced truncation and residualization as model reduction techniques and develop appropriate switching algorithms for each. The switching algorithms proposed require minimal overhead to ensure smoothness of the controller output. This makes it feasible for implementation in an environment where computation is expensive. The idea of anytime control algorithm is then applied to a realistic flight control problem. From the example presented, we observe that substantial reduction in computational time can be accommodated while still keeping the degradation in controller performance within acceptable limits.

If the LTI controller cannot be decoupled into fast and slow modes without leading to degradation of the controller performance, the algorithms proposed in this paper will not be very useful. If the observable states alone require far too much computation time, then again, the algorithms described in this paper will not be useful. The algorithms described are best suited for system implementations that have a split of fast and slow modes and the fast modes can be updated within the given time.

Acknowledgments

This work was funded by Defense Advanced Research Projects Agency under the Software-Enabled Control program with John Bay as Program Manager. The contract number is USAF/AFMC F33615-99-C-1497 and Dale Van Cleave is the Technical Contract Monitor.

References

[1]Zilberstein, S., "Using Anytime Algorithms in Intelligent Systems," *Artificial Intelligence Magazine*, Vol. 17, No. 3, 1996, pp. 73–83.

[2]Boddy, M., and Dean, T. L., "Deliberation Scheduling for Problem Solving in Time-Constrained Environments," *Artificial Intelligence*, Vol. 67, No. 2, 1994, pp. 245–285.

[3]Boddy, M., and Dean, T. L., "Solving Time-Dependent Planning Problems," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, pp. 979–984; also available at URL: http://ijcai.org [cited Jan. 2002].

[4]Horvitz, E. J., "Computation and Action under Bounded Resources," Ph.D. Dissertation, Dept. of Computer Science and Medicine, Stanford Univ., Stanford, CA, Dec. 1990.

[5]Liu, J. S., Lin, K. J., Shih, W. K., Yu, A. C., Chung, J. Y., and Zhao, W., "Algorithms for Scheduling Imprecise Computations," *IEEE Computer*, Vol. 24, No. 5, 1991, pp. 58–68.

[6]Yoshimoto, H., Arita, D., and Taniguchi, R., "Real-Time Communication for Distributed Vision Processing based on Imprecise Computation Model," *Proceedings of International Parallel and Distributed Processing Symposium*, IEEE Computer Society, 2002, pp. 128–133; also available at URL: http://www.ipdps.org.

[7]Millan-Lopez, V., Feng, W., and Liu, J. W. S., "Using the Imprecise-Computation Technique for Congestion Control on a Real-Time Traffic Switching Element," *International Conference on Parallel and Distributed Systems*, IEEE Publications, Piscataway, NJ, 1994, pp. 202–208; also available at URL: http://www.computer.org [cited Jan. 2002].

[8]Huang, X., and Cheng, A. M. K., "Applying Imprecise Algorithms to Real-Time Image and Video Transmission," *Proceedings International Conference on Parallel and Distributed Systems*, IEEE Publications, Piscataway, NJ, 1995, pp. 96–101; also available at URL: http://www.computer.org [cited Jan. 2002].

[9]Lee, J., Kim, E., and Lee, D., "Imprecise Data Computation for High Performance Asynchronous Processors," *Proceedings of the Asia and South Pacic Design Automation Conference*, IEEE Publications, Piscataway, NJ, 2001, pp. 261–266; also available at URL: http://www.computer.org [cited Feb. 2001].

[10]Russel, S. J., and Wefald, E. H., "Principles of Metareasoning," *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, edited by R. J. Brachman, H. J. Levesque, and R. Reiter, Morgan Kaufmann, San Mateo, CA, 1989, pp. 400–411; also available at URL: http://www.mkp.com.

[11]Russel, S. J., and Wefald, E. H., *Do the Right Thing: Studies in Limited Rationality*, MIT Press, Cambridge, MA, 1991; also available at URL: http://mitpress.mit.edu.

[12]Doyle, J., "Rationality and Its Roles in Reasoning," *Proceedings of the Eighth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Menlo Park, CA, 1990, pp. 1093–1100; also available at URL: http://www.aaai.org.

[13]D'Ambrosio, B., "Resource Bounded Agents in an Uncertain World," *Working Notes of the Workshop on Real-Time Artificial Intelligence Problems*, URL: http://ijcai.org, 1989 [cited Jan. 2002].

[14]Dean, T. L., and Boddy, M., "An Analysis of Time-Dependent Planning," *Proceedings of the Seventh National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Menlo Park, CA, 1988, pp. 49–54; also available at URL: http://www.aaai.org [cited Jan. 2002].

[15]Horvitz, E. J., "Reasoning about Beliefs and Actions under Computational Resource Constraints," *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, Association for Uncertainty in Artificial Intelligence, 1987, pp. 429–444; also available at URL: http://www.auai.org [cited Jan. 2002].

[16]Lin, K. J., Natarajan, S., Liu, J. W. S., and Krauskopf, T., "CONCORD: A System of Imprecise Computations," *Proceedings of COMPSAC, Computer Software and Application Conference*, IEEE Publications, Piscataway, NJ, 1987, pp. 75–81; also available at URL: http://www.computer.org [cited Jan. 2002].

[17]Zilberstein, S., "Operational Rationality through Compilation of Anytime Algorithms," Ph.D. Dissertation, Dept. of Computer Science, Univ. of California at Berkeley, Berkeley, CA, 1993.

[18]Samad, T., and Balas, G. J., *Software-Enabled Control, Information Technology for Dynamical Systems*, Wiley Interscience URL: http://www.interscience.wiley.com [2003], Chap 3.

[19]Bitmead, R., Gevers, M., and Wertz, V., *Adaptive Optimal Control: The Thinking Man's GPC*, International Series in Systems and Control Engineering, Prentice–Hall, Englewood Cliffs, NJ, 1990; also available at URL: http://www.prenhall.com [cited Jan. 2002].

[20]Soeterboek, R., *Predictive Control: A Unified Approach*, International Series in Systems and Control Engineering, Prentice–Hall, Englewood Cliffs, NJ, 1992; also available at URL: http://www.prenhall.com [cited Jan. 2002].

[21]Primbs, J., "Nonlinear Optimal Control: A Receding Horizon Approach," Ph.D. Dissertation, Dept. of Control and Dynamical Systems, California Inst. of Technology, Pasadena, CA, Jan. 1999.

[22]Jadbabaie, A., Yu, J., and Hauser, J., "Stabilizing Receding Horizon Control of Nonlinear Systems: A Control Lyapunov Function Approach," *American Control Conference*, Vol. 3, American Automatic Control Council, Dayton, OH, 1999, pp. 1535–1539; also available at URL: http://www.a2c2.org [cited Jan. 2002].

[23]Bhattacharya, R., Balas, G. J., Kaya, M. A., and Packard, A., "Nonlinear Receding Horizon Control of an F-16 Aircraft," *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 5, 2002, pp. 924–931.

[24]Moore, B. C., "Principal Component Analysis in Linear Systems: Controllablity, Observablity, and Model Reduction," *IEEE Transactions in Automatic Control*, AC-35, Vol. 26, No. 1, 1981, pp. 203–208.

[25]Enns, D., "Model Reduction for Control Design," Ph.D. Dissertation, Dept. of Aeronautics and Astronautics, Stanford Univ., Stanford, CA, 1984.

[26]Glover, K., "All Optimal Hankel-norm Approximations of Linear Multivariable Systems and their L_∞-error Bounds," *International Journal of Control*, Vol. 39, No. 6, 1984, pp. 1115–1193.

[27]Kothare, M. V., Campo, P. J., Morari, M., and Nett, C. N., "A Unified Framework for the Study of Anti-Windup Designs," *Automatica*, Vol. 30, No. 12, 1994, pp. 1869–1883.

[28]Branicky, M. S., "Multiple Lyapunov Functions and Other Analysis Tools for Switched and Hybrid Systems," *IEEE Transactions in Automatic Control*, Vol. 43, No. 4, 1998, pp. 475–482.

[29]Ganguli, S., and Balas, G., "A TECS Alternative Using Robust Multivariable Control," *AIAA Guidance, Navigation, and Control Conference*, AIAA, Reston, VA, URL: http://www.aiaa.org [cited Aug. 2001].

[30]Zilberstein, S., Charpillet, F., and Chassaing, P., "Real-Time Problem-Solving with Contract Algorithms," *Proceedings of International Joint Conference on Artificial Intelligence*, URL: http://ijcai.org, 1999 [cited Jan. 2002].

[31]Bernstein, D., Perkins, T. J., Zilberstein, Z., and Finkelstein, L., "Scheduling Contract Algorithms on Multiple Processors," *Proceedings of AAAI*, American Association for Artificial Intelligence, URL: http://www.aaai.org [cited Jan. 2002].