Anytime Control Algorithm: Model Reduction Approach

Raktim Bhattacharya * Gary J. Balas †

Department of Aerospace Engineering & Mechanics,

University of Minnesota,

Minneapolis, MN-55455

October 31, 2002

Abstract

Recently, there has been considerable interest in *anytime algorithms* for real-time systems. Anytime algorithms are computational models that tradeoff quality of result for computational time. The tolerance to fluctuating CPU time makes anytime algorithms operationally optimal for real-time task scheduling. In this paper we present a methodology that transforms linear control algorithms into *anytime control algorithms.* Linear control algorithms require fixed computational times to produce controller output. Hence they fail to compute the output if the alloted CPU time is less than required, and can't make use of any excess CPU time to improve performance. When implemented as a real-time system, the static nature of the required CPU time makes it operationally sub-optimal for task scheduling. We transform linear controllers to anytime control algorithms by switching between controllers of different order. In this paper, we consider balanced truncation and residualisation as model reduction tool to generate a set of lower order controllers that require lesser computational time and present a switching algorithm that smoothly switches between these controllers to accomodate variation in available computational time.

*Graduate Assistant, raktim@aem.umn.edu

†Professor, balas@aem.umn.edu

1 Introduction

Anytime algorithms are algorithms that trade performance for time. They are capable of providing results at any point in their execution. The quality, accuracy or performance of the algorithm improves with increase in processing time. Therefore, for anytime algorithms, the binary notion of the correctness associated with traditional computational procedures is replaced by the set of quality measured answers. In this paper we propose a methodology that will transform linear time invariant control algorithms into anytime algorithms with the help of model reduction and bumpless switching of linear systems.

In recent times, the advance 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 complete computation at all times by all the algorithms in the system. Anytime algorithms provide a technique for allocating computational resources to the most useful algorithm and enables optimal usage of hardware resources.

Specifically, anytime algorithms differ from conventional computational procedures in several ways [1]. Anytime algorithms return a result with a measure of its quality. This may be a best guess or a group of possible answers. They also contain some information about the output quality given a certain amount of time and information about the input it receives. Anytime algorithms can be interrupted and the solution available at that point of execution can be returned. They can also continue executing past the deadline they are given. This allows systems that use such algorithms to change the computational time, allocated to an anytime algorithm, during run-time. Anytime algorithms always improve the output quality as they are given more time. The improvement in the solution however is larger in the early stages of computation and diminishes over time.

Anytime algorithms first emerged in the area of artificial intelligence. Early application of such algorithms can be found in medical diagnosis and mobile robot navigation. The term *anytime algorithm* was coined by Dean and Boddy in the late 1980s in the context of their work on time-dependent planning [2, 3]. 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 in [4]. In 1991, Jane Liu [5] introduced a similar idea, termed as *imprecise computation*, and applied it to real-time systems. They showed that imprecise computation techniques provide scheduling flexibility by trading off result quality to meet computational deadlines. Ever since then, the concept of imprecise computation has been applied to solve several diverse problems [6, 7, 8, 9]. The idea of anytime algorithms is also similar to the notion of rationality in automated reasoning and search investigated by Russel *et al.* in [10, 11] Doyle in [12] and D'Ambrosio in [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 performance of real-time systems [14, 15, 16]. Zilberstien in [17] defines a real-time task to be *operationally rational* if it optimises the allocation of resources to its performance components so as to maximise its overall expected utility. Such tasks are able to vary processing time according to "time-pressure". This capability can be achieved if traditional algorithms, whose expected run-time is normally fixed, is replaced by more flexible computational modules, namely anytime algorithms.

Currently, control algorithms are implemented as digital control systems, which in essence are real-time systems with a dedicated application. The control algorithms are typically, traditional computational procedures with fixed run-times. In order to make these tasks operationally rational, it is necessary to transform them into anytime algorithms. This serves as the motivation for the research work presented in this paper. We will term control algorithms that behave as anytime algorithms as *anytime control algorithms.*

In the controls literature, control algorithms that are based on online optimisation, like receding horizon control [18, 19, 20, 21] can be tuned to vary the computational time it requires. The optimisation problem for these algorithms is essentially minimisation of a cost functional with state and control constraints. The computational time for such algorithms can be varied dynamically, by varying number of decision variables, complexity of system model, accuracy of numerical algorithms and optimality of control action [22]. Also, since the process of minimisation is iterative in nature, it can be pre-empted to obtain the current best solution. However, determination of the minimum time that should be alloted to the optimisation algorithm, to generate a stabilising solution, may not be trivial. Therefore, any control algorithm that is based on online optimisation can potentially qualify as anytime control algorithm.

However, control algorithms that are linear and synthesised off-line, as shown in equation(1, 2),

$$\begin{aligned}
&\textit{Continuous Time :} \\
\dot{x_c} &= Ax_c + Bu_i \qquad (1) \\
u_o &= Cx_c + Du_i \\
&\textit{Discrete Time :} \\
x_c^{k+1} &= \hat{A}x_c^k + \hat{B}u_i^k \qquad (2) \\
u_o^k &= \hat{C}x_c^k + \hat{D}u_i^k
\end{aligned}$$

the computational time required to compute the output of a linear system is fixed. Computation of controller output involves matrix-vector multiplications and vector-vector additions. The number of flops required to perform these computations is constant and hence will require a fixed amount of CPU time. The computational time can however be reduced by reducing the order of the controller. It is obvious that the reduced order controllers must ensure closed-loop stability and the price for reducing computational time should be at the cost of degraded performance. Thus we see that it is possible to vary the computational time of linear controllers, at the cost of quality of result, by executing a reduced order controller. From the implementation point of view, the available time will dictate the order of controller that can be implemented. Computation of the lowest order controller that guarantees robust stability will be the minimum time that must be alloted to the algorithm for the functioning of the control system. Thus the controllers are switched from higher to lower order to accomodate shortage in CPU time. Note that the switching has to be done smoothly to enable bumpless transfer and also with minimal CPU overhead.

The paper is organised as follows. First we present the fact that the required CPU time of linear controllers can be varied by reducing the order of controller. Model reduction technique used to generate a set of controllers with decreasing order, each rendering closed-loop stability and requiring decreasing computational time. Obviously, the price for reduction in computational time is paid by reduction in achievable closed-loop performance. We propose a switching algorithm that smoothly switches between controllers of different order to accomodate changes in available CPU time. Construction of the switching algorithm is the main challenge of this research, as it should have minimal CPU overhead for feasibility. Finally we implement this algorithm on a B737-100 TSRV (Transport System Research Vehicle) linear longitudinal motion model and present the simulation results.

2 Complexity Analysis via `FLOP` Count

The computational overhead of a numerical algorithm is often measured in terms of the number of floating point operations or `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 in modern times. Features like cache boudaries 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 give 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 $A \in \mathcal{R}^{m \times n}$ and a vector $x \in \mathcal{R}^n$, the `FLOP` count for the computation Ax is $2mn$. Similarly, the computation $x + y$, where $y \in \mathcal{R}^n$ will have a `FLOP` count of n. We will use these definitions of `FLOP` counts to analyse the CPU overhead of the switching algorithms presented in this paper.

3 Variation of CPU Time for Linear Controllers

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

$$\begin{aligned} x_c^{k+1} &= Ax_c^k + Bu_c^k \\ y_c^k &= Cx_c^k \end{aligned} \tag{3}$$

involve 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, i.e $T_c \in [T_{\min}, T_{\max}]$. In this paper we use linear model reduction technique, namely balanced truncation and residualisation to generate a set of linear time invariant (LTI) systems with varying run-times.

3.1 Balanced Truncation

If the controller K defined by eqn.(3) is a balanced realisation[23, 24, 25], then we can partition the state space as

$$x_c = \left\{ \begin{array}{c} w \\ z \end{array} \right\} \tag{4}$$

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

$$\begin{aligned} \left\{ \begin{array}{c} w^{k+1} \\ z^{k+1} \end{array} \right\} &= \begin{bmatrix} A_{ww} & A_{wz} \\ A_{zw} & A_{zz} \end{bmatrix} \left\{ \begin{array}{c} w^k \\ z^k \end{array} \right\} + \begin{bmatrix} B_w \\ B_z \end{bmatrix} u_c^k \\ y_c^k &= \begin{bmatrix} C_w & C_z \end{bmatrix} \left\{ \begin{array}{c} w^k \\ z^k \end{array} \right\} \end{aligned} \tag{5}$$

Since the states z are weakly observable and controllable, they do not contribute significantly, in terms of Hankel singular values, to the controller output y_c. A reduced order model of the controller K can be obtained by ignoring these states. The computational time depends on the number of states rejected, i.e. the size of z. The dynamics of the reduced order controller can be written as

$$\begin{aligned} w^{k+1} &= A_{ww}w^k + B_w u_c^k \\ y_c^k &= C_w w^k \end{aligned} \tag{6}$$

3.2 Residualisation

In this approach, a 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 (i.e. stable). Therefore, the transfer function of the reduced order LTI system K_{red} can be written as,

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

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

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

Then the dynamics of the reduced order system is

$$\begin{aligned} x_s^{k+1} &= A_s x_s^k + B_s u_c^k \\ y_c^k &= C_s x_s^k + D_{ss} u_c^k \end{aligned} \tag{7}$$

where $D_{ss} = C_f(I - A_f)^{-1}B_f$ is the steady-state contibution of K_f.

3.3 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 $\mathcal{K} = \{K_i\}, 1 \leq i \leq p$, 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, i.e. $T_{max} = T_c(K_1) > \ldots > T_c(K_p) = T_{min}$. 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 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 accomodate 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 $T_c(C_i) \leq T_{cpu}$. Thus we can accomodate 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.

4 Controller Switching Algorithm with Minimal CPU Overhead

Switching theory of controllers is a well established area[26]. Unfortunately, those ideas cannot be used directly as many switching algorithms require simultaneous operation of controllers prior to switching or require significant CPU time. This is opposite of the motivation of the posed problem where switching is done to accomodate changes in T_{cpu}. 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, to construct the proposed switching algorithm.

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 $\mathcal{K} = \{K_1, K_p\}$, i.e. we switch between the highest and the lowest order controller, then we extend it to the case where $\mathcal{K} = \{K_1, K_2, ..., K_p\}$.

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

4.1 Switching Algorithm for Balanced Systems

4.1.1 Switching from higher to lower order controller

If $K_1, K_p \in \mathcal{K}_{bal}$, then by simply switching from the full order controller (K_1) in eqn.(5) to the reduced order controller (K_p) in eqn.(6), there will be an undesirable impulse effect in y_c^k if z states of the controller dynamics are just truncated. 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 as,

$$\begin{aligned} w^{k+1} &= A_{ww}w^k + A_{wz}z^k + B_w u_c^k \\ z^{k+1} &= z^k \\ y_c^k &= C_w w^k + C_z z^k \end{aligned} \tag{8}$$

Thus, in the modified reduced order controller, w and y_c evolve with z held constant and this preserves continuity of controller output at the time of switching.

Comments

1. Ideally we would want to fade the z states to zero so that eqn.(8) transforms into eqn.(6). Unfortunately, the fading out process would require computation, which is

expensive.

2. The term $A_{wz}z^k$ in eqn.(8) is a vector that need not be computed at every time step. Since 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, the addition of terms $A_{wz}z^k$ and $C_z z^k$ to eqn.(8) results in smooth switching from K_1 to K_p and requires minimal CPU overhead.

3. The constant z vector in eqn.(8) acts as a disturbance at the input and output of the controller. The worst-case effect of this piece-wise 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(a), where r, e, y_c, y and u are the exogenous input, exogenous output, controller output, plant output and plant input respectively. The disturbace z acts on the system as shown in Fig. 1(b). The worst-case effect of z on e and z on u is then the ∞-norm of the corresponding transfer functions. Since $||z||_2$ 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.

4. The disappearance of the z dynamics will cause a change in the "derivative" of the controller output and may cause transients to appear in the closed-loop response. These transients could be removed with the help of a filter that smoothens the "derivative" of the controller output at switching instances. This however will require more computation.

4.1.2 Switching from reduced to full order controller

If $K_1, K_p \in \mathcal{K}_{bal}$, switching from a lower order controller to a higher order controller is more complicated. While K_p is active the w states have evolved with z held constant. In general, if the dynamics of z is suddenly added to the system, there could be large transients z trajectory and consequently in the closed-loop response. To minimise 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 since we have enough computational time to implement K_1. If k_0 is the time when K_1 switched to K_p, then the controller dynamics during this phase is given by eqn.(9).

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

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

Comments

1. The time taken to decay the z states to zero depends on λ. If the z states are decayed too fast, transients will appear in the plant input and consequently in the close-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.

2. Switching back the dynamics of z, with initial condition $z = 0$, will result in transients appearing in the trajectory of z. Since z is weakly controllable, we expect these transients to be small. Moreover, since z is weakly observable, we expect the effect of these transient on y_c^k to be small. The transients in z^k and its effect on y_c^k however will increase from K_1 to K_p as the observability and controllability of the states z

increase. These transients can by removed with a low pass filter. The implementation of the filter will however add computational overhead to the algorithm.

4.1.3 Generalisation of Algorithm

In this section the switching scheme is generalised 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, prior to 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 as constants.

4.2 Switching Algorithm for Residualised Systems

4.2.1 Switching from higher to lower order controller

If $K_1, K_p \in \mathcal{K}_{resid}$, then at the time of switching, K_1 in eqn.(3.2) and K_p in eqn.(7) will have different outputs, in general. This difference will cause an impulse like behaviour 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, at the time of switching, to the output of K_p.

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 modifies to,

$$\begin{aligned} x_s^{k+1} &= A_s w^k + B_s u_c^k \\ x_f^{k+1} &= x_f^k \\ y_c^k &= C_s x_s^k + D_{ss} u_c^k + \Delta y_c^{k_0} \end{aligned} \tag{10}$$

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

$$\begin{aligned} \Delta y_c^{k_0} &= (C_f x_f^{k_0} + C_s x_s^{k_0}) - (C_s x_s^{k_0} + D_{ss} u_c^{k_0}) \\ &= (C_f x_f^{k_0} - D_{ss} u_c^{k_0}) \\ &= C_f x_f^{k_0} - C_f (I - A_f)^{-1} B_f u_c^{k_0} \\ &= C_f (x_f^{k_0} - \bar{x}_f^{k_0}) \end{aligned} \tag{11}$$

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

Comments

1. The vector $\Delta y_c^{k_0}$ in eqn.(10) can be treated as a piece-wise constant disturbance, added at the output of K_p as shown in Fig.1(c), and its worst case effect on e and u can be quantified by the ∞-norm of the corresponding transfer functions.

2. Since we are equating the controller output over two consecutive time-steps, it means that the controller output is constant over this interval. This happens everytime the controller switches from a higher order system to a lower order system.

3. When residualisation 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 doesn't cause any effect on the dynamics of x_s. This however will cause a sudden change in the "derivative" of y_c^k, which will cause transients to appear in the closed-loop system response. These transients can be removed with a low-pass filter at the cost of adding computation.

4.2.2 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, then the difference in the controller output at k_1 is

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

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

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

This difference can be reduced to zero by initialising 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 while $\Delta y_c^{k_0}$ is decaying to zero is given by

$$\begin{aligned}
x_s^{k+1} &= A_s x_s^k + B_s u_c^k \\
x_f^{k+1} &= x_f^k \\
x_1^{k+1} &= x_2^k \\
x_2^{k+1} &= -\lambda^2 x_1^k - 2\lambda x_2^k \\
y_c^k &= C_s x_s^k + D_{ss} u^k + x_1^k \Delta y_c^{k_0}
\end{aligned} \tag{12}$$

In eqn.(12), the states $x_1, x_2 \in \mathcal{R}$ are the states of the second order filter with initial condition $x_1 = 1, x_2 = 0$ 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 initialised to its steady-state value, based on the current input, and the controller switches to K_1.

Comments

1. The activation of K_1 is delayed by the time $\Delta y_c^{k_0}$ takes to reach zero. Therefore, it is desirable that the decay rate of $\Delta y_c^{k_0}$, determined by λ in eqn.(12), 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.

2. 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 i^{th} 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

$$A_f^{-1}B_f = \begin{bmatrix} A_{p+1}^{-1}B_{p+1} \\ A_{p+2}^{-1}B_{p+2} \\ \vdots \\ A_n^{-1}B_n \end{bmatrix} \tag{13}$$

where A_i, B_i are the A, B matrices of the modal system Γ_i. Therefore, if $A_i^{-1}B_i$ is computed off-line, then the computational overhead in initialising x_f is only that required to multiply the matrix in eqn.(13) with vector u_c. This computation is feasible since it is equivalent to the computation $B_f u_c$.

3. The storage of $A_i^{-1}B_i$ for $i \in \{1, 2, \cdots, n\}$, increases the memory overhead of the implementation.

4. 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.

5. Since $D_{ss}u_c$ 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 everytime the controller switches from a lower order system to a higher order system.

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

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

The `FLOP` count for the decay process is

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

Therefore, if the computation of the decay process is feasible then $F1 - F2 \geq 0$

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

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

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

4.2.3 Generalisation of the Switching Algorithm

In this section we generalise the switching scheme for any two systems $K_i, K_j \in \mathcal{K}_{resid}$. 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 $\mathcal{K}_{resid}$ are arranged in the decreasing order, of the order of the controller. That is, if $i < j$ it implies, order of controller K_i is higher than that of controller K_j. In general, K_i is defined as,

$$\begin{aligned} x_{s_i}^{k+1} &= A_{s_i} x_{s_i}^k + B_{s_i} u_c^k \\ K_i:\ x_{f_i}^{k+1} &= x_{f_i}^k \\ y_{c_i}^k &= C_{s_i} x_{s_i}^k + D_{ss_i} u^k + \Delta y_{c_i} \end{aligned}$$

Switching from K_i to K_j

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

$$x_{s_i} \equiv \left\{ \begin{array}{c} x_{s_j} \\ \hat{x}_{s_j} \end{array} \right\}, x_{f_j} \equiv \left\{ \begin{array}{c} \hat{x}_{s_j} \\ x_{f_i} \end{array} \right\}$$

and the difference in the output of the two controllers is

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

where k_0 is the time K_i switches to K_j.

Switching from K_j to K_i

When switching from a lower order controller to a higher order controller, i.e. from K_j to

K_i, $i < j$, some of the states of x_{f_j} will start evolving and affect the output of K_i. Let those states be $\hat{x}_{f_i}$. Therefore,

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

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

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

If the controller K is in modal form, i.e,

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

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

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

Therefore

$$\begin{aligned} \Delta y_{c_i} &= (D_{ss_j} - D_{ss_i}) u_c^k - \hat{C}_{f_i} \hat{x}_{f_i}^k + \Delta y_{c_i} \\ &= \hat{D}_{ss_i} u_c^k - \hat{C}_{f_i} \hat{x}_{f_i}^k + \Delta y_{c_j} \end{aligned} \tag{14}$$

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 initialised to their steady-state values based on the current input, i.e. $\hat{x}_{f_i} = -\hat{A}_{f_i}^{-1} \hat{B}_{f_i} u$ and the controller K_j switches to K_i.

5 Proof of Stability

The effect of the switching algorithm on the stability of the closed loop system is analysed 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. Since we don't expect T_{cpu} to fluctuate rapidly, it is reasonable to assume that there are finite switches in finite time.

5.1 Balanced Realisation Reduction

Let the plant dynamics be given by

$$\begin{aligned} x_p^{k+1} &= A_p x^k + B_p u_p \\ y_p^k &= C_p x_p \end{aligned} \tag{15}$$

where x_p, u_p and y_p denotes plant states, input and output. If the inter-connection of the plant and the controller in eqn.(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

$$\left\{ \begin{array}{c} x_p^{k+1} \\ w^{k+1} \\ \hline z^{k+1} \end{array} \right\} = \left[\begin{array}{cc|c} A_p & B_p C_w & B_p C_z \\ B_w C_p & A_{ww} & A_{wz} \\ \hline B_z C_p & A_{zw} & A_{zz} \end{array} \right] \left\{ \begin{array}{c} x_p^k \\ w^k \\ \hline z^k \end{array} \right\} \tag{16}$$

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

$$f_0 : \left\{ \begin{array}{c} v^{k+1} \\ z^{k+1} \end{array} \right\} = \left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array} \right] \left\{ \begin{array}{c} v^k \\ z^k \end{array} \right\} \tag{17}$$

where A_{ij} is defined from the partition of the A matrix in eqn.(16). The closed loop with controller from eqn.(8) and eqn.(9) are

$$f_1 : \begin{Bmatrix} v^{k+1} \\ z^{k+1} \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ 0 & I \end{bmatrix} \begin{Bmatrix} v^k \\ z^k \end{Bmatrix} \tag{18}$$

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,

$$f_2 : \begin{Bmatrix} v^{k+1} \\ z^{k+1} \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} \begin{Bmatrix} v^k \\ z^k \end{Bmatrix} \tag{19}$$

Thus we have three stable systems f_0, f_1, f_2 and a switching sequence that switches between them. We use multiple Lyapunov function approach[27] to prove stability of this switched linear system.

The switching sequence S for this switched linear system is

$$S = x_0, (f_0, T_0), (f_1, T_1), (f_2, T_2), (f_0, T_3), \ldots \tag{20}$$

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

$$\mathcal{I}(i) \in \{[T_i, T_{i+1}), [T_{i+3}, T_{i+4}), \ldots, [T_{i+3j}, T_{i+3j+1}), \ldots\} \tag{21}$$

where $j \in \mathcal{Z}^*$, $\mathcal{Z}^*$ is the set of non-negative integers. Define $\mathcal{E}(i)$ as the set of times when system f_i is switched to, i.e.

$$\mathcal{E}(i) = \{T_i, T_{i+3}, \ldots, T_{i+3j}, \ldots\},\; j \in \mathcal{Z}^* \tag{22}$$

Definition (2.2 in [27]) - *Given a strictly increasing sequence of times $\mathcal{T} = \{t_k\},\ k \in \mathcal{Z}^*$, 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:*

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 non-increasing on $\mathcal{E}(i)$

Denoting x_S^k as the state trajectory of the closed-loop system under switching sequence S, Branicky in [27] 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.

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

$$V_i(x) = x^T P_i x; \quad i = 0, 1, 2 \tag{23}$$

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

$$V_i(x^{k+1}) < V_i(x^k); \quad i = 0, 1, 2 \tag{24}$$

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

$$\begin{aligned} ||x_r||_2 &\geq ||x_{r+1}||_2; \quad r \in \mathcal{Z}^* \\ \Rightarrow \quad ||x_r||_2 &\geq ||x_{r+3j}||_2; \quad r \in \mathcal{Z}^*, \ j \in \mathcal{Z}^+ \end{aligned} \tag{25}$$

where $\mathcal{Z}^+$ denotes positive integers. Therefore V_i in eqn.(23) is non-increasing in $\mathcal{E}(i)$. This completes the proof on the stability of the switched linear system in context.

5.2 State Reduction via Residualisation

When $K_1, K_p \in \mathcal{K}_{resid}$, the closed-loop systems switches between three systems f_0, f_1 and f_2 . They are constructed with controllers defined in equations 3.2, 10 and 12. If the plant-actuator model in Fig. 1(a) 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 eqn.(12) 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 eigen-values 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

$$f_0 \;:\; \left\{ \begin{array}{c} x_p \\ x_s \\ x_f \\ \Delta y_c \end{array} \right\}^{k+1} = \begin{bmatrix} A_p & B_2C_s & B_2C_f & 0 \\ B_sC_2 & A_s & 0 & 0 \\ B_fC_2 & 0 & A_f & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \left\{ \begin{array}{c} x_p \\ x_s \\ x_f \\ \Delta y_c \end{array} \right\}^{k} \tag{26}$$

$$f_1 \;:\; \left\{ \begin{array}{c} x_p \\ x_s \\ x_f \\ \Delta y_c \end{array} \right\}^{k+1} = \begin{bmatrix} A_p & B_2C_s & B_2C_f & B_2 \\ B_sC_2 & A_s & 0 & 0 \\ B_fC_2 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \left\{ \begin{array}{c} x_p \\ x_s \\ x_f \\ \Delta y_c \end{array} \right\}^{k} \tag{27}$$

$$f_2 \;:\; \left\{ \begin{array}{c} x_p \\ x_s \\ x_f \\ \Delta y_c \end{array} \right\}^{k+1} = \begin{bmatrix} A_p & B_2C_s & B_2C_f & B_2 \\ B_sC_2 & A_s & 0 & 0 \\ B_fC_2 & 0 & I & 0 \\ 0 & 0 & 0 & A_d \end{bmatrix} \left\{ \begin{array}{c} x_p \\ x_s \\ x_f \\ \Delta y_c \end{array} \right\}^{k} \tag{28}$$

As in section 5.1, 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 $\{T_0, T_3, \cdots\}$ and $\{T_1, T_4, \cdots\}$ 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 section 5.1, we have to prove V_i is non-increasing in $\mathcal{E}(i)$. In the proof we represent time by k. Therefore, the set $\mathcal{E}(i)$ in eqn.(22), is defined as

$$\mathcal{E}(i) = \{k_i, k_{i+3}, \ldots, k_{i+3j}, \ldots\},\; j \in \mathcal{Z}^* \tag{29}$$

Since 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,

$$\left\| \left\{ \begin{array}{c} x_p \\ x_s \end{array} \right\} \right\|_2^{k+1} \leq \left\| \left\{ \begin{array}{c} x_p \\ x_s \end{array} \right\} \right\|_2^{k}, \forall k \tag{30}$$

If x_f is initialised at time k_{3j} as,

$$\begin{array}{rcl} x_f^{k_{3j}} & = & (I - A_f)^{-1} B_f C_2 x_p^{k_{3j}} \\ \Rightarrow \ \|x_f^{k_{3j}}\|_2 & \leq & \|(I - A_f)^{-1} B_f C_2\|_\infty \|x_p^{k_{3j}}\|_2 \\ & = & \gamma_1 \|x_p^{k_{3j}}\|_2 \end{array}$$

Since f_0 is stable,

$$\gamma_1 \|x_p^{k_{3j}}\|_2 \geq \|x_f^k\|_2 \ , k_{3j} \leq k < k_{3j+1}$$

Therefore, the Lyapunov function for f_0 can be defined as

$$V_0(x_p^k, x_s^k) = \|x_p^k\|_2^2 + \|x_s^k\|_2^2 + \gamma_1 \|x_p^{k_{3j}}\|_2 \ , k_{3j} \leq k < k_{3j+1}, j \in \mathcal{Z}^+ \tag{31}$$

Therefore, from eqn.(30), we can conclude that $V_0(x_p^k, x_s^k)$ is non-increasing in $\mathcal{E}(0)$.

To show V_1 is non-increasing in $\mathcal{E}(1)$ consider the following. Since 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 when switching from f_0 to f_1, the following is true

$$\gamma_1 \|x_p^{k_{3j}}\|_2 \geq \|x_f^k\|_2 \ , k_{3j+1} \leq k < k_{3j+2}$$

The discontinuous jump in Δy_c^k is defined as

$$\begin{array}{rcl} \Delta y_c^k & = & C_f x_f^k - C_f (I - A_f)^{-1} B_f C_2 x_p^k \\ \Rightarrow \ \|\Delta y_c^k\|_2 & \leq & \|C_f\|_\infty \|x_k^k\|_2 + \|C_f (I - A_f)^{-1} B_f C_2\|_\infty \|x_p^k\|_2 \\ & \leq & \gamma_1 \|C_f\|_\infty \|x_p^{k_{3j}}\|_2 + \|C_f (I - A_f)^{-1} B_f C_2\|_\infty \|x_p^k\|_2 \\ & = & \gamma_2 \|x_p^{k_{3j}}\|_2 + \gamma_3 \|x_p^k\|_2 \end{array}$$

Therefore, the Lyapunov function for f_1 can be defined as

$$V_1(x_p^k, x_s^k) = (1 + \gamma_3) \|x_p^k\|_2^2 + \|x_s^k\|_2^2 + \gamma_2 \|x_p^{k_{3j}}\|_2 \ , k_{3j+1} \leq k < k_{3j+2}, j \in \mathcal{Z}^+$$

and from eqn.(30), we can conclude that $V_1(x_p^k, x_s^k)$ is non-increasing in $\mathcal{E}(1)$.

When f_2 is active, x_f^k is constant and Δy_c^k decays to zero. The bound 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 non-increasing in $\mathcal{E}(2)$.

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

6 Example

The switching scheme is next applied to a B737-100 TSRV (Transport System Research Vehicle) linear longitudinal motion model. The aircraft model has four states: longitudinal velocity V, angle-of-attack α, pitch rate q and pitch angle θ; two control inputs: thrust T(lb) and elevator deflection δ_e. The elevator actuator and the engine are modelled 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. Details of the controller design can be obtained from [28].

6.1 Balanced Realisation Reduction

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. 2, we present the frequency response of the four transfer functions,

$$\left\{ \begin{array}{c} \gamma \\ V \end{array} \right\} = \left[\begin{array}{cc} G_{\gamma\gamma_{ref}} & G_{\gamma V_{ref}} \\ G_{V\gamma_{ref}} & G_{VV_{ref}} \end{array} \right] \left\{ \begin{array}{c} \gamma_{ref} \\ V_{ref} \end{array} \right\} \tag{32}$$

Let us represent the closed-loop system in eqn.(32), with controller K_i, as P_i. From the frequency-magnitude plots in Fig. 2, there is small difference in the performance of P_1 and

P_p. The two controllers achieve desired decoupling between γ and V response to reference input. The induced norm of P_1 and P_p is 1.12 and 1.27 respectively. Therefore, we see that there is not significant degradation in the performance of the controller by using K_p as the controller.

In time domain, the closed-loop response to a step command of 20 ft/s in V, with the full order controller is shown in Fig. 3. 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. The closed-loop response to the same V command with the lower order controller (without constant z states), is shown in Fig. 3. We observe degradation in the performance as the velocity command induces a γ response with peak amplitude of approximately 0.7 degrees.

In terms of computational times, 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, by implementing the lower order controller we can accomodate approximately 37% drop in T_{cpu} in this example, and the CPU overhead for smooth switching is minimal.

The closed-loop response with the switching algorithm incorporated is shown in Fig. 4. In the plots, the simulation starts with the full order controller. At 30 sec, T_{cpu} falls to 67% and the controller switches K_p. The lower order controller tracks the reference signal till 70 sec. Note the coupled response in γ and V tracking during this time. At 70 sec 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 sec. Thus the dynamics of the controller is changed at times $t = 30, 70$ and 112.23 seconds. We observe from the trajectories that there are no impulse transients at these times due to switching. The oscillations in $\gamma(t)$ in Fig. 4 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

seconds for the z states to decay from its value at $t = 30s$ to zero. Depending on the application this may be too long to recover the higher order controller and λ has to be chosen accordingly. Since 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 seconds, because of 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 $30s$ results in small transients in the elevator angle at $30s$. The addition of the z dynamics at time $112.23s$ also doesn't induce any transients in the closed-loop response.

Thus, the simulation plots in Fig. 4 show 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.

6.2 State Reduction via Residualisation

The effect of the switching algorithm on the B737 model, with controllers K_1 and K_p obtained using residualisation, is presented in this section. The lowest order controller obtained using residualisation, 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 $K_p \in \mathcal{K}_{resid}$ has fewer number of states than $K_p \in \mathcal{K}_{resid}$.

Let us represent the closed-loop system in eqn.(32), with controller K_i, as P_i. The frequency response of the transfer functions, defined by eqn.(32), with $K_1, K_p \in \mathcal{K}_{resid}$ is shown in Fig. 2. From these plots we see that $K_p \in \mathcal{K}_{resid}$ causes differences from P_1 at high frequency. 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, with the $K_p \in \mathcal{K}_{resid}$ is shown in Fig. 3. We observe that the degradation in the performance as the velocity step command induces a significantly larger γ response and causes oscillatory control actions in both thrust and elevator, as compared with that of K_1.

With the switching algorithm implemented, the time response to the same velocity step command is shown in Fig.4. As in the case for $K_1, K_p \in \mathcal{K}_{resid}$, the controller starts with K_1, switches to K_p at time $30s$. The computational resources come back to 100% and Δy_c starts to decay. It reaches zero at time $82.4s$ and the controller switches back to K_1 with appropriately initialised x_f. The savings in computational overhead, for this example is approximately 59%.

We see that there are noticeable transients, especially in the elevator angle, at times when the controller dynamics changes. The transient at $30s$ is because of the sudden disappearance of x_f dynamics, the transient at $70s$ is induced by the decaying of Δy_c and the transient at $82.4s$ 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 $70s$ and K_1 will take effect earlier. Note that minimising this transient doesn't require any additional computation. However, reduction of the transients at $30s$ and $82.4s$ will require implementation of appropriately defined filters, which will add computational overhead.

Comparing the two switching algorithms, based on the two techniques of model reduction, we observer the following,

1. $K_p \in \mathcal{K}_{resid}$ has 10 states and $K_p \in \mathcal{K}_{bal}$ has 13 states. Therefore, for this example, the switching algorithm based on residualisation can accomodate a larger reduction in CPU time.

2. Both the techniques induce transients whenever dynamics of states are either discarded or added to the controller. The transients because of the z states are small compared to those induced by x_f states.

3. The transients because of the decaying of z states or Δy_c is governed by the choice of λ. A value of λ reduces the transients because of the decaying process, but it delays the activation of the higher order system. Therefore, in both the switching schemes there is a tradeoff between magnitude of transients and delay in activation of the higher order control algorithm.

7 Comparison with Anytime Algorithms

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

1. *Anytime algorithms return a result with a measure of quality* - The measure of quality for a 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 pre-empted anytime* - Anytime control algorithms, as defined in this paper, cannot be preempted once it starts. The available CPU time must be known prior to 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 alloted CPU time. To obtain a valid result, the algorithm must execute till completion.

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. Since we achieve anytime behaviour of control algorithms by switching controllers of different order, this feature is not as pronounced as it is in iterative algorithms.

Therefore, we see that the *anytime control algorithm*, proposed in this paper, could exhibit the most salient feature of *anytime algorithms*, namely the trade off between quality of output and computation time. It however has limitations in some of the other anytime properties.

8 Summary

In this paper we have presented a method to transform linear controllers to behave similarly as anytime control algorithms, i.e. they are able to compromise on the quality of solution for 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 considered balanced truncation and residualisation as model reduction trechniques and developed appropriate switching algorithms for each. The switching algorithms proposed, require minimal overhead to ensure smoothness of 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 accomodated while still keeping degradation in controller performance within acceptable limits.

9 Acknowledgements

This work was funded by DARPA under the Software Enabled Control program with Dr. John Bay as the program manager. The contract number is USAF/AFMC F33615-99-C-1497 and Dale Van Cleave is the technical contract monitor.

References

[1] S. Zilberstein. Using Anytime Algorithms in Intelligent Systems. *Artificial Intelligence Magazine*, 17(3):73–83, 1996.

[2] M. Boddy and T. L. Dean. Deliberation Scheduling for Problem Solving in Time-Constrained Environments. *Artificial Intellegence*, 67(2):245–285, 1994.

[3] M. Boddy and T. L. Dean. Solving Time-Dependent Planning Problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 979–984, Menlo Park, California, 1989. International Joint Conferences on Aritificial Intelligence.

[4] E. J. Horvitz. *Computation and Action under Bounded Resources.* PhD thesis, Department of Computer Science and Medicine, Stanford University, 1990.

[5] J.W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. Algorithms for Scheduling Imprecise Computations. *IEEE Computer*, 24(5):58–68, 1991.

[6] H. Yoshimoto, D. Arita, and R. Taniguchi. Real-time Communication for Distributed Vision Processing based on Imprecise Computation Model. *Proceedings of International Parallel and Distributed Processing Symposium*, pages 128–133, 2002.

[7] V. Millan-Lopez, W. Feng, and J. W. S. Liu. Using the Imprecise-Computation Technique for Congestion Control on a Real-Time Traffic Switching Element. *International Conference on Parallel and Distributed Systems*, pages 202–208, 1994.

[8] Xiaofen Huang and A. M. K. Cheng. Applying Imprecise Algorithms to Real-Time Image and Video Transmission. In *Proceedings International Conference on Parallel and Distributed Systems*, pages 96–101, 1995.

[9] J. Lee, E. Kim, and D. Lee. Imprecise Data Computation for High Performance Asynchronous Processors. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 261–266, 2001.

[10] S. J. Russel and E. H. Wefald. Principles of Metareasoning. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Pronciples of Knowledge Representation and Reasoning*, pages 400–411, San Mateo, California, 1989. Morgan Kaufmann.

[11] S. J. Russel and E. H. Wefald. *Do the Right Thing: Studies in Limited Rationality.* MIT Press, Cambridge, Mass., 1991.

[12] J. Doyle. Rationality and Its Roles in Reasoning. In *Proceedings of the Eight National Conference on Artificial Intelligence*, pages 1093–1100, Menlo Park, California, 1990. American Association for Artificial Intelligence.

[13] B. D'Ambrosio. Resource Bounded Agents in an Uncertain World. In *Working Notes of the IJCAI-89 Workshop on Real-Time Artificial Intelligence Problems*, Detroit,Michigan, 1989.

[14] T. L. Dean and M. Boddy. An Analysis of Time-Dependent Planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, Minneapolis, Minnesota, 1988.

[15] E. J. Horvitz. Reasoning about Beliefs and Actions under Computational Resource Constraints. In *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, pages 429–444, Seattle, Washington, USA, 1987.

[16] K. J. Lin, S. Natarajan, J. W. S. Liu, and T. Krauskopf. Concord: A System of Imprecise Computations. In *Proceedings of COMPSAC*, pages 75–81, Tokyo, Japan, 1987.

[17] S. Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms.* PhD thesis, Department of Computer Science, University of California at Berkeley, 1993.

[18] R.R. Bitmead, M. Gevers, and V. Wertz. *Adaptive Optimal Control: The Thinking Man's GPC.* International Series in Systems and Control Engineering. Prentice Hall, 1990.

[19] R. Soeterboek. *Predictive Control: A Unified Approach.* International Series in Systems and Control Engineering. Prentice Hall, 1992.

[20] J.A. Primbs. *Nonlinear Optimal Control: A Receding Horizon Approach.* PhD thesis, California Institute of Technology, 1999.

[21] A. Jadbabaie, J. Yu, and J. Hauser. Stabilizing receding horizon control of nonlinear systems: A control lyapunov function approach. *American Control Conference*, 3:1535–9, 1999.

[22] R. Bhattacharya, G. J. Balas, M. A. Kaya, and A. Packard. Nonlinear Receding Horizon Control of an F-16 Aircraft. *Journal of Guidance, Control, and Dynamics*, 25(5):924–931, 2002.

[23] B. C. Moore. Principal Component Analysis in Linear Systems: Controllablity, Observablity, and Model Reduction. *IEEE Transactions in Automatic Control*, AC-35:203–208, 1981.

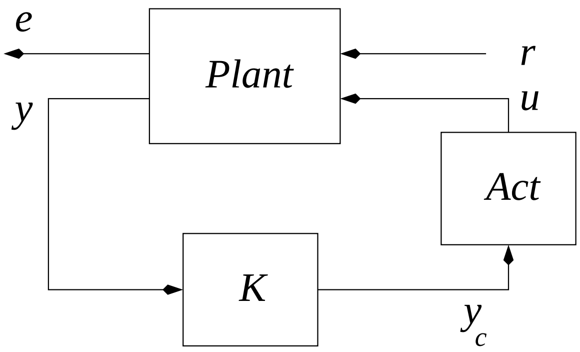
[24] Dale Enns. *Model Reduction for Control Design.* PhD thesis, Department of Aeronautics and Astronautics, Stanford University, 1984.

[25] K. Glover. All Optimal Hankel-norm Approximations of Linear Multivariable Systems and their $\mathcal{L}_\infty$-error Bounds. *Int. J. Control*, 39:1115–1193, 1984.

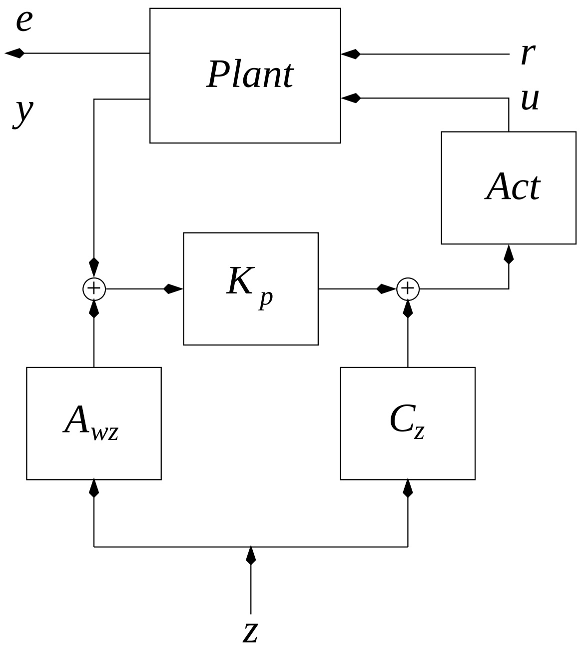
[26] M. V. Kothare, P. J. Campo, M. Morari, and C. N. Nett. A Unified Framework for the Study of Anti-Windup Designs. *Automatica*, 30(12):1869–1883, 1994.

[27] M. S. Branicky. Multiple Lyapunov Functions and Other Analysis Tools for Switched and Hybrid Systems. *IEEE Transactions in Automatic Control*, 43(4):475–482, 1998.

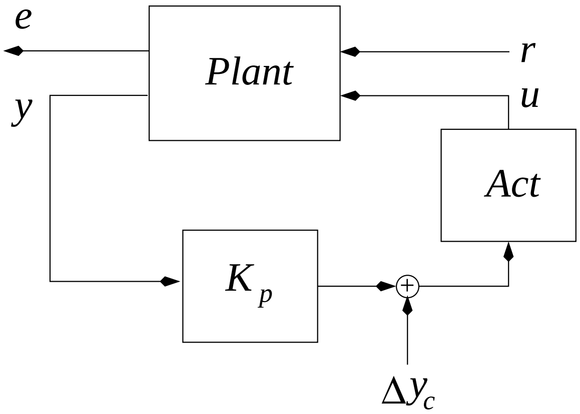
[28] S. Ganguli and G. Balas. A TECS Alternative Using Robust Multivariable Control. *AIAA Guidance, Navigation and Control Conference*, Aug 2001.



(a) Closed-loop System



(b) Closed-loop System with Disturbance z



(c) Closed-loop System with Decaying Disturbance Δy_c

Figure 1: System Interconnections

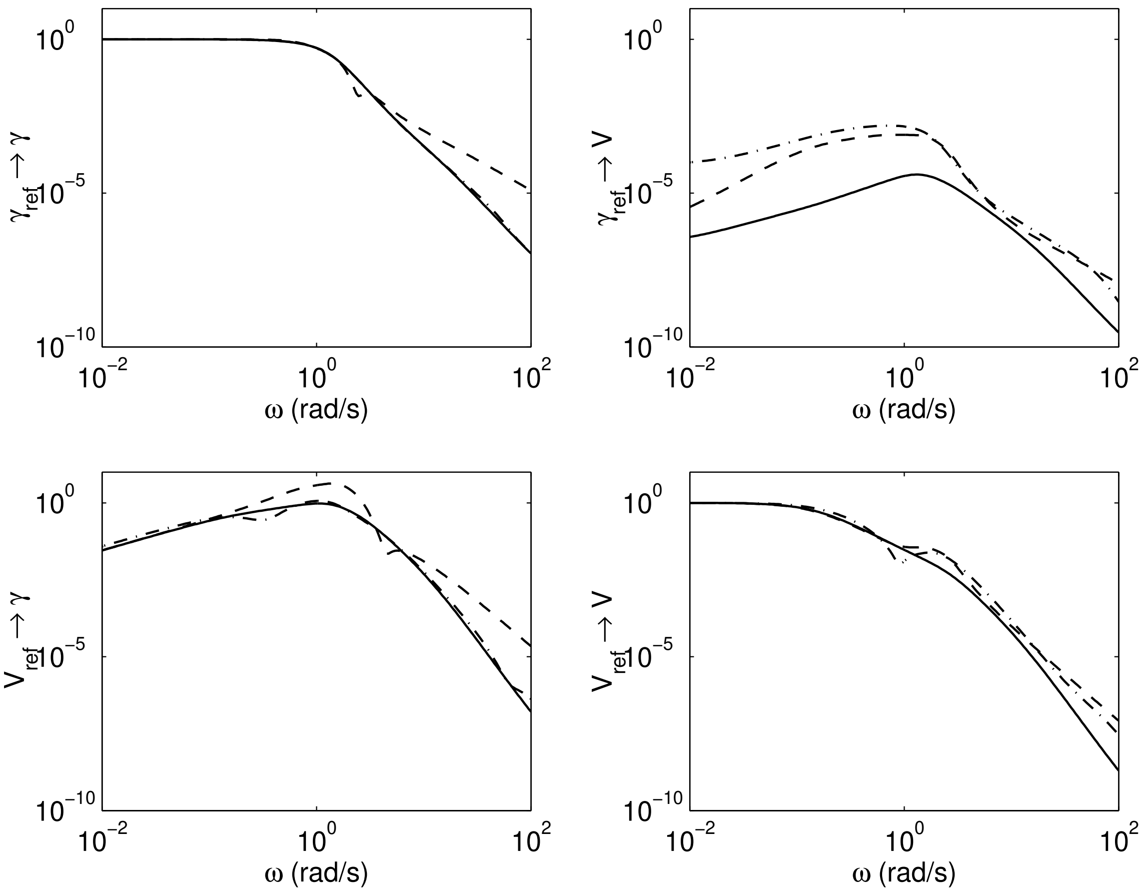


Figure 2: Frequency Response of $(\gamma_{ref}, V_{ref}) \rightarrow (\gamma, V)$ for full order controller (solid), $K_p \in \mathcal{K}_{bal}$ (dash-dot) and $K_p \in \mathcal{K}_{resid}$ (dash-dash).

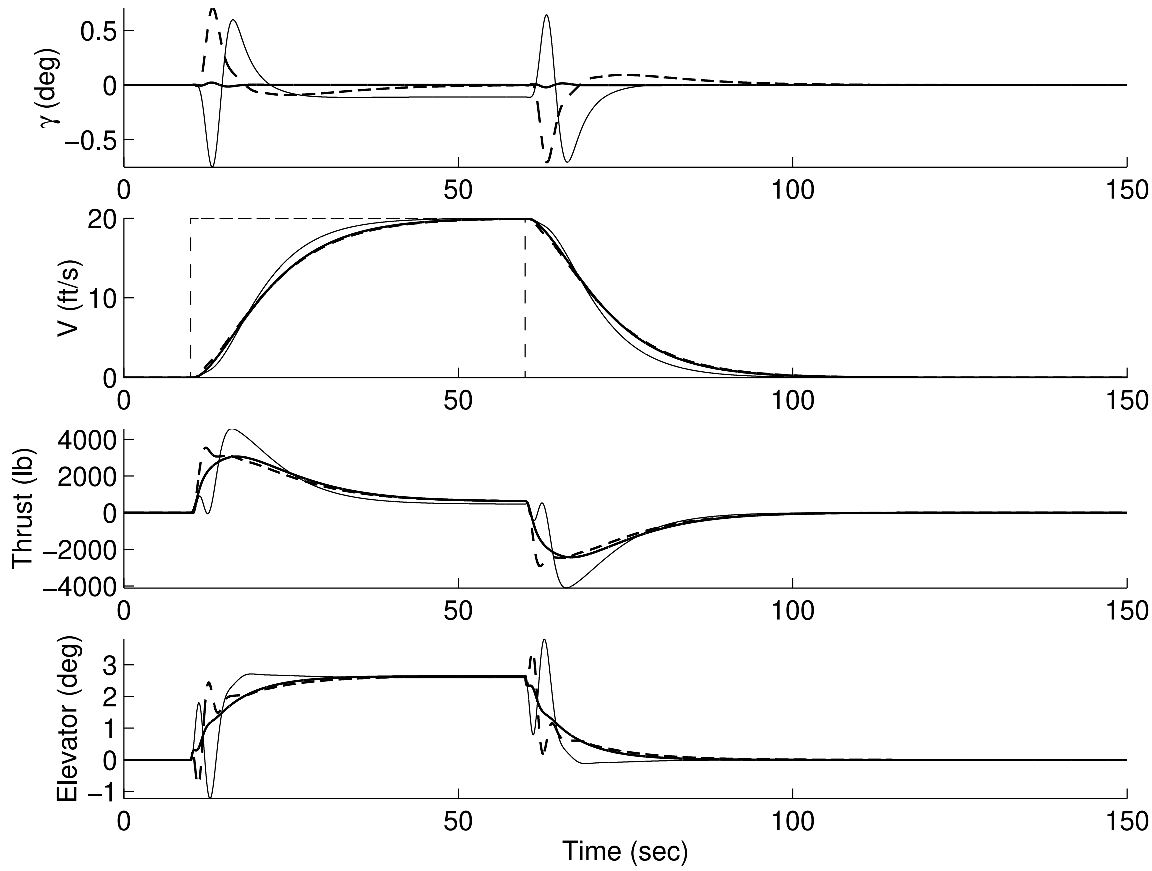


Figure 3: Closed-loop time response with full order controller (thick solid), $K_p \in \mathcal{K}_{bal}$ (thin solid) and $K_p \in \mathcal{K}_{resid}$ (dashed).

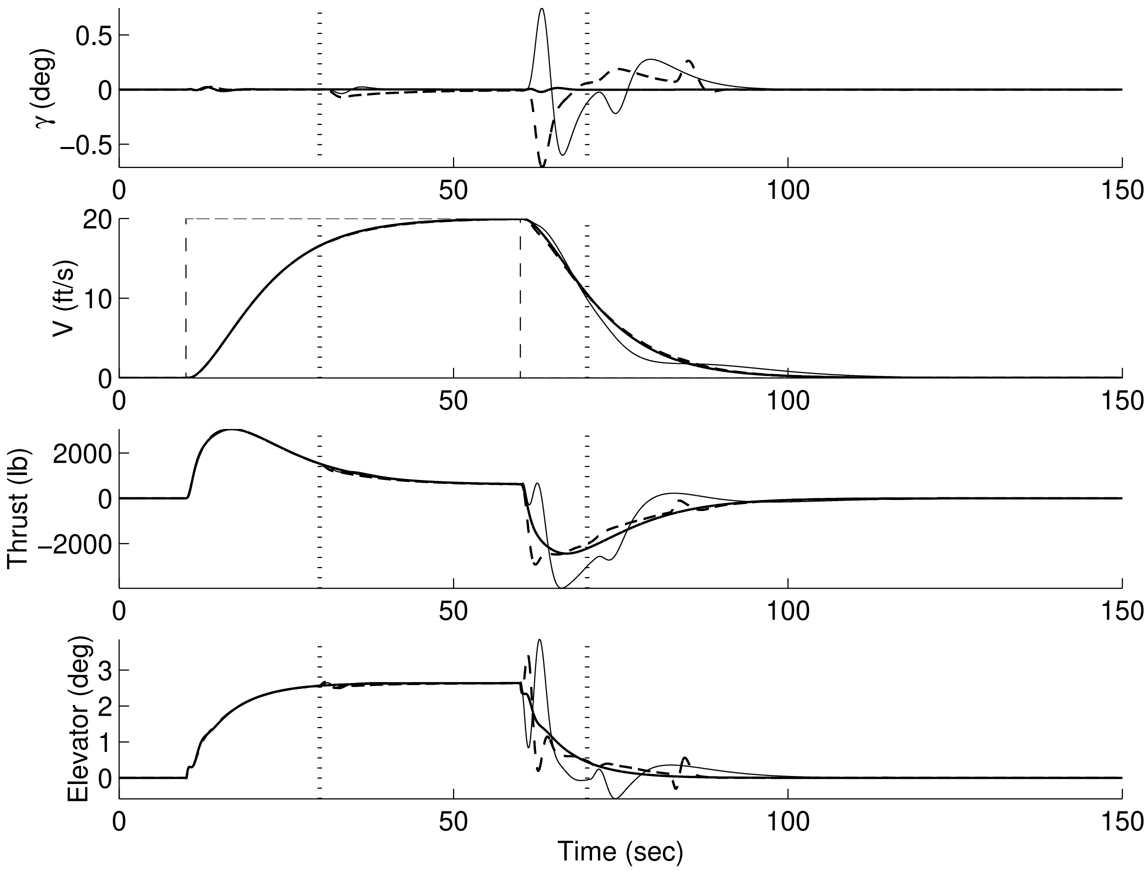


Figure 4: Closed-loop time response with switching of controllers, full order controller (thick solid), $K_p \in \mathcal{K}_{bal}$ (thin solid) and $K_p \in \mathcal{K}_{resid}$ (dashed).