# Reconfigurable Autopilot Design for a High Performance Aircraft Using Model Predictive Control

by

Jose Pedro Ruiz

B.S. Aerospace Engineering with Information Technology
Massachusetts Institute of Technology, 2002

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS

AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SEPTEMBER 2004



Signature of Author: _____
Department of Aeronautics and Astronautics
July 2, 2004

Certified by: _____
Piero Miotto
Charles Stark Draper Laboratory, Inc.
Thesis Supervisor

Certified by: _____
John J. Deyst
Professor of Aeronautics and Astronautic, MIT
Thesis Advisor

Accepted by: _____
Jaime Peraire
Professor of Aeronautics and Astronautics
Chairman, Committee on Graduate Students

**[This Page Intentionally Left Blank]**

# Reconfigurable Autopilot Design for a High Performance Aircraft Using Model Predictive Control

by

Jose Pedro Ruiz

Submitted to the Department of Aeronautics and Astronautics on July 2, 2004 in partial fulfillment of the requirements for the degree of Master of Science in Aeronautics and Astronautics

ABSTRACT

The losses of military and civilian aircraft due to control surface failures have prompted research into controllers with a degree of reconfiguration. This thesis will describe a design approach incorporating Model Predictive Control (MPC) with a self updating model to achieve a level of reconfiguration in a generic high performance aircraft. MPC has the advantage of explicitly taking a model of the failed system and incorporating it into a receding horizon optimization problem. MPC also has the added benefits of allowing constraints on the inputs, outputs, and states of the system as well as tuning flexibility. This thesis describes the development of four types of MPC autopilots. A description of the controller implementation and failure implementation is also included. Each autopilot is subject to a surface failure during certain times in a sample maneuver and the resulting controller adaptation is analyzed. All MPC controllers are found to maintain good performance in the event of certain failures with an updated internal model. It is when the internal model is not updated that full performance is not recovered and in some cases, loss of the aircraft results.

Thesis Supervisor: Piero Miotto

Title: Senior Member of the Technical Staff, Charles Stark Draper Laboratory

Thesis Supervisor: John J. Deyst

Title: Professor of Aeronautics and Astronautics, MIT

**[This Page Intentionally Left Blank]**

# ACKNOWLEDGEMENTS

I would like to thank everyone here in Boston who helped me along the way over the past two years. I was very lucky to have a great supervisor and friends while here.
I would like to say thanks to my supervisor Piero Miotto. He was an excellent supervisor who always made time to help me through any issues this thesis might have caused. He taught me to think more critically and showed me that with a little cleverness any problem can be solved. I can only hope my future bosses are as fit as him both in the technical and managerial sense. Secondly I'd like to thank my thesis advisor John Deyst. He was instrumental in helping me to finally finish this thesis. Also thanks to Fred Boelitz and Leena Singh as part of the MPC group who helped motivate MPC research. Speaking with them was always enlightening and helpful.
I would also like to thank my mom for encouraging me all those years and my brother for always being there. They encouraged me even when I thought graduate school at MIT seemed more like a dream than a reality.
Thanks also to my housemates Seth, Chris, and Semi. They made my stay in Somerville and the experience of graduate school bearable. Those late night talks around the kitchen table helped me keep my sanity throughout these past two years.
I also must thank my buddies at Draper, Jillian, Abran and Tiffany. From everything to talks about the mysteries of MPC tuning to where our careers were going, they made Draper a warmer place to be.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc, under Internal Company Sponsored Research with contract number IRD04-0-5043, Model Predictive Control.



Jose P. Ruiz July 2, 2004

**[This Page Intentionally Left Blank]**

# Table of Contents

**[This Page Intentionally Left Blank]**

# List of Figures

# List of Tables

**[This Page Intentionally Left Blank]**

# Introduction

The design of autopilots for modern day aircraft often does not take into account failures in any of the controlling surfaces. The controller is merely tested in a wide variety of flight regimes and the resulting robustness is assumed to take into account a certain type of failure. These existing controllers however often cannot compensate for failures involving the flight surfaces. For instance on April 12, 1977, Flight 1080 an L-1011 flying out of San Diego had its left horizontal stabilizer jam in a full trailing edge up position just after take off. Fortunately the crew was able to learn to fly the aircraft using throttles as a supplement and landed safely. It is very possible however that a less able crew could have met with disastrous results. Flight accidents such as these have motivated researchers for decades to try and develop a reconfigurable flight control system.

In order to design a truly self reconfiguring controller many methods have been developed. These range in form from simply designing highly tuned controllers offline for specific failures to direct adaptive approaches involving neural networks. The first batch of design methods are known as automated failure dependent gain schedule techniques. These take both modern and classical controller design approaches such as PID and LQR, and extend them for a set of specific predetermined failures. If that particular failure occurs these controllers are brought online according to some failure detection unit [2]. The second class, known as control reallocation, actually tries to solve the torque allocation problem prompted by damaged surfaces by solving linear programs in real time [7]. The final class however is the closest attempt to realizing a truly reconfigurable control system. This is known as general constrained optimization and includes methods such as implicit model following, indirect and direct adaptive control and model predictive control [9].

This thesis will address the problem of developing a reconfigurable autopilot by applying Model Predictive Control (MPC) with an updatable internal model. The MPC controller will issue high level commands to an inner stabilizing loop containing a Control Augmentation System or CAS. The MPC controller will also assume to have a functioning failure detection and isolation (FDI) system onboard. The only duty of this FDI system is to give the MPC controller information on the control surface position. Using this information, the internal model will be updated and used to predict the aircraft's behavior with the damaged surfaces. The inner loop CAS aboard the aircraft however will not be updated to adapt to the surface failures. All reconfiguration will take place through the newly tailored control strategy constructed by the outer loop MPC controller. It is by using the predictions of the failure updated internal model that MPC will be able to completely restructure its own control strategy in real time.

## 1.1 Problem Motivation

As was previously stated, most non-experimental aircraft are not outfitted with a reconfigurable control system. In modern aircraft the burden of developing a new control strategy falls directly on the pilot if such a failure occurs. Thus the MPC control system would provide the added capability of automatically compensating for unforeseen failures. Its uses are also obvious for the newer autonomous aerospace vehicles in which there are no pilots to both recognize the failures and learn to fly the aircraft with the remaining control effectors.

With the advent of modern computers and the rapid gains in processor speeds, the large computational loads typically encountered by MPC controllers are becoming less of a problem. It is only now that MPC is being considered for aerospace applications which require higher controller bandwidths as opposed to the process industry in which it has been used for the past twenty years. One key aspect of MPC is its use of an internal model for control. It is with this model that it performs an optimization procedure to solve for the best inputs according to a cost function and the aircraft's perceived ability. Thus it is only MPC that has the ability to incorporate knowledge of the failure directly into its own control strategy using the updated internal model.

## 1.2 Overview

The goal of this research is to show the advantages and disadvantages of several reconfigurable autopilots designed with MPC. The results will show the benefit of having MPC's internal model updated with the failure in question as opposed to without it. It will also show MPC's ability to tailor a brand new control strategy with the failure in question.

The aircraft models used in these comparisons are identical except for the heading/altitude autopilot which incorporates a thrust vectoring system. Furthermore, each model is a six Degree-of-Freedom (6DOF) representation of a generic high performance aircraft. Each controller for the four individual autopilots developed guides the aircraft along a predetermined trajectory and is subject to a failure in the principal control surfaces.

## 1.3 Results Portability

This research uses a generic high performance aircraft and thus is considered to be representative of a large array of military aircraft flying today. The augmented dynamics due to the inner loop render most of these aircraft similar in flight characteristics due to the standards published by the military [1]. Although some aircraft may be unstable in open loop, the addition of the CAS establishes the aircraft to have similar flying characteristics defined in terms of frequency or time response. For instance, on the pitch axis the short period and phugoid damping are typically similar when the inner loop CAS is considered [5]. This fact coupled with the standard practice of using traditional control surfaces such as ailerons, rudders, and horizontal tails makes this aircraft model very similar to many flying today. Thus the MPC algorithms resulting in this research should be portable to most aircraft platforms.

## 1.4 Thesis Preview

This thesis is organized as described below following the introductory chapter:

Chapter 2 presents the mathematical theory behind the MPC controllers used in the autopilots. A general discussion is followed by the mathematical background for the

particular type of MPC used in the research, perturbational MPC. It also provides an example of the pertubational MPC method applied to a roll autopilot

Chapter 3 presents the details of the vehicle model used in this research. The model platform, coordinate frames and modifications are also discussed. A brief introduction to the autopilot mode definitions is also offered.

Chapter 4 presents the MPC controllers used throughout this research. It gives an example of each completing a sample maneuver. It also describes the controller implementation and MPC parameter selection.

Chapter 5 presents the failure scenario designs and methods. It describes both the type of failures introduced and their actual implementation.

Chapter 6 presents the results from the comparison of the MPC with and without an updated internal model reflecting failure for all the four autopilots introduced.

Chapter 7 summarizes the findings of this research and offers recommendations for further research.

# Chapter 2

# MPC Perturbation Theory

## 2.1 MPC Overview

Model Predictive Control (MPC) is a general control scheme in which an explicit model is incorporated into a dynamic control law. This method is computationally intensive and therefore has until recently only been applied to low rate controller and low bandwidth systems. In fact, since the 1980's MPC has been used in a wide variety of chemical and process control applications [5] [8]. Fortunately, significant increases in computer processor speeds have made MPC applicable to higher bandwidth systems such as flight vehicles.

MPC is a general methodology referring to an optimal receding horizon control strategy. There are various techniques, but in principle MPC is a multiple input multiple output (MIMO) constraint handling controller. The MPC architecture is robust and can handle non-minimum phase and unstable plants, as well as those with large time delays. This robustness is a result of the seemingly brute force method of control on which it operates. MPC essentially tailors a new control strategy at each sample time that is based on predicted outputs created with the internal model. MPC is especially capable of controller reconfiguration because of this inherent ability to re-compute its entire control strategy at each sample time. This remarkable adaptive quality of MPC will be used in this thesis to provide a degree of reconfiguration in a high performance aircraft.

A high level block diagram can be seen below to show the structure of MPC.

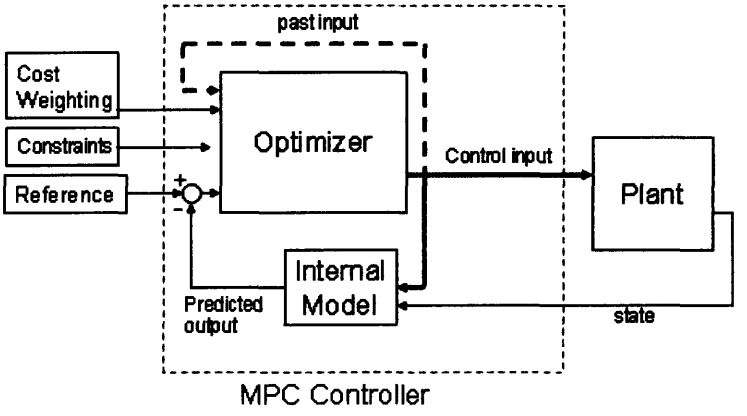


**Figure 1: MPC Graphical Layout**

As can be seen, the reference trajectory r(t) in Figure 2 is compared with the predicted output created by the internal reference model. The optimizer then solves for the control sequence while satisfying constraints and weighting information. The first N inputs are then applied and the entire process repeated with the propagation model initialized with the current states. The entire strategy consequently becomes closed loop by initializing a series of essentially open loop optimal control problems. It is by reiterating this process at each sample time that the finite optimization is moved forward along in time (see Figure 2). This creates the receding horizon principle that is essential in all MPC schemes.

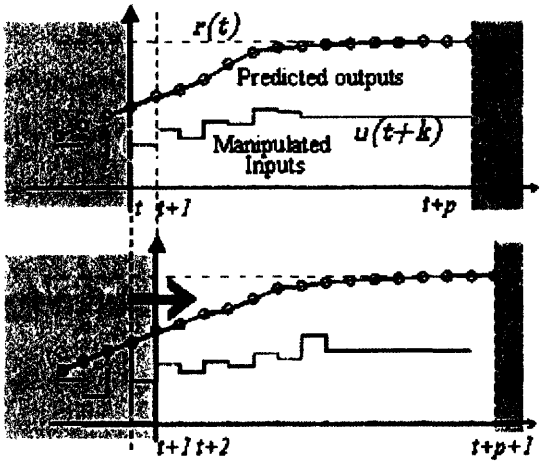


**Figure 2: MPC Receding Horizon Graphic**

The internal model is used to predict the plant's output over a certain prediction horizon, *Hp*. The control often has its own horizon in which free controls can be computed known

as the control horizon, *Hu*. If *Hp* > *Hu* then the last control input is typically kept constant until the end of the prediction. This is typically done to reduce the amount of control variables to be solved for. The predicted output is then compared with a reference output and incorporated into a cost function. The function is subsequently minimized solving for a complete set of optimal inputs. Thus at each time step the MPC system is solving for a complete control sequence over the prediction horizon.

The MPC methodology can be broken down into 4 basic parts.

- Propagation Model
- Performance Index
- Optimization
- Constraints

The following sections will introduce these elements in further detail.

### 2.1.1 Propagation Model

MPC predicts outputs based on its internal propagation model. The propagation model used in an MPC scheme can vary somewhat with the principal constraints being computation time and fidelity. It is common to use a linear model of the plant's dynamics. In most cases a state space model representation, as shown in Equation 1, is used for the propagator.

$$\begin{aligned} y(k) &= C(k)x(k) + D(k)u(k) \\ x(k+1) &= A(k)x(k) + B(k)u(k) \end{aligned} \tag{1}$$

The state space matrices A(k), B(k), C(k), and D(k) can change over the prediction horizon if the horizon is long enough. This makes the propagation algorithm cumbersome as the state space matrices must be continually updated along the prediction horizon. In addition to this, the propagation sequence used in MPC must then be able to update itself with the different state space models as it looks forward in time. These models are of course only valid at the operating point they were linearized and thus separate models must be used as the vehicle travels out of this point. Despite its drawbacks, this method of propagation is prevalent due to its compact description. A

proper mathematical treatment of this method can be found in reference [5]. Other plant descriptions can also be used such as step/impulse response models and polynomial models though they have similar constraints due to linearization. However, yet another MPC scheme employs the entire nonlinear model for propagation.

In this form the plant can be any function of the state x and the inputs u.

$$y(k) = g(x(k), u(k)) \qquad (2)$$
$$x(k+1) = f(x(k), u(k))$$

The propagator only has to initialize and propagate the nonlinear function to make the predicted outputs. This reduces the model into nothing more than an input output relationship and is the method this thesis incorporates. The model must only be initialized properly and then proceed with a discrete input to output relationship. This can be seen in the figure below.

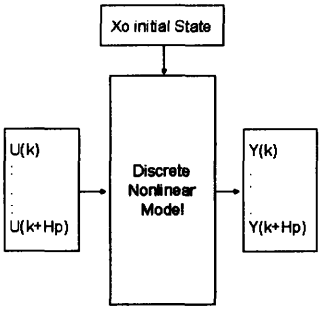


**Figure 3: Nonlinear Propagator**

Through this method, the model is valid at all operating points and would not have to be linearized at each point. The propagation algorithm is also simpler because all that is needed is proper initialization and input information. The algorithm itself does not have to take into account the changing of state space matrices along the prediction horizon. It must be stressed that in the limit all propagation schemes would approach the identical optimal output for that particular input. The only difference is the difficulty of the propagation system implementation.

## 2.1.2 Performance Index

In order to fold MPC into an optimal control technique, a performance index must be used. Typically the performance index, or cost function, is written in terms of norms as in Equation 3.

(3) $$\|x\|_Q^2 = x^T Q x$$

For a prediction horizon of *Hp* and control horizon of *Hu* the cost function J is as follows.

(4) $$J(k) = \sum_{i=0}^{Hp-1} \|z(k+i) - r(k+i)\|_{Q(i)}^2 + \sum_{i=0}^{Hu-1} \|u(k+i)\|_{R(i)}^2$$

Equation 4 shows the difference between the reference r and predicted output z to be weighted with the matrix Q. The input u is further weighted through the matrix R in the second term of the equation. The Q and R matrices are diagonal and are used to weight inputs and outputs both against each other and against themselves over the prediction horizon. It is also common for some methods to weight the control increment Δu or u(k)-u(k-1). This will drive the steady state error to zero by reducing the incremental control size. Since the cost function is quadratic and convex, an unconstrained minimum can always be found. Otherwise, standard QP algorithms are employed.

## 2.1.3 Optimization and Constraints

At the beginning of each MPC cycle the cost function is fully populated and ready to be optimized. If there are no constraints a closed form solution can easily be found and implemented. In general this form of quadratic optimization is represented as the following.

(5) $$J = \frac{1}{2} \underline{x}^T H \underline{x} + f^T \underline{x}$$

The closed form solution is found by setting the cost function's derivative equal to 0 and solving for x.

$$\underline{x_{opt}} = -H^{-1}\underline{f} \tag{6}$$

This closed form solution cannot be used however when linear constraints are applied to the vector x. In this event a quadratic programming algorithm must be used to compute the constrained optimal solution. Constraints are sent to the quadratic programming algorithm in the following form.

$$\underline{b_{\min}} \le A\underline{x} \le \underline{b_{\max}} \tag{7}$$

It will be shown that constraints can be placed on inputs, outputs and states within the plant. In the presence of linear constraints, a quadratic programming solver will find the global optimum. In some cases the solution will be on the constraint's boundary.

## 2.2 Perturbation Control

In some applications the MPC algorithm directly solves for all inputs in the control horizon. The number of variables to solve for $N_v$ then becomes related to the prediction horizon *Hp,* the prediction rate $\Delta t$, and the number of inputs $N_i$.

$$N_v = N_i * \frac{H_p}{\Delta t} \tag{8}$$

One way to massively reduce the order of this problem however is through perturbational control and basis functions. Basis functions are used to reduce the amount of variables solved for at each time step by the MPC controller. It does this by using basis functions to serve as the free variables in the optimization problem.

The number of variables to be solved for then becomes simply the number of inputs $N_i$ multiplied by the number of basis functions $N_b$.

(9) $$N_v = N_i * N_b$$

## 2.2.1 Basis Functions

Basis functions are a collection of elementary functions used to build up the input vector. The QP algorithm solves for the optimum linear combination of these primitive functions. The input signal, v(t), can be represented by a linear combination of basis functions, $S_i(t)$, in the following way.

(10) $$v(t) = \sum_{i=0}^{B} S_i(t)\alpha_i$$

Many types of basis functions can be used to create the input. In Section 4.4 I give a brief description of these functions. A special set of basis functions, often used in MPC controllers, are orthogonal functions that satisfy the following condition.

(11) $$\sum_{k=-\infty}^{\infty} S_i(k)S_j(k) = 0 \text{ for } i \neq j$$

$$= 1 \text{ for } i = j$$

The orthogonality condition often allows the number of independent variables, required to satisfactorily span the input space, to be reduced.

Other non orthogonal functions that are commonly used by MPC controllers are temporally spaced steps, ramps or impulses. Even though they do not satisfy the orthogonality condition, they have been proven highly effective in various applications (process control) because of their simplicity and flexibility. They can easily be spaced over the horizon to increase the degrees of freedom of the solution at the beginning of the time horizon. For example, the following figure shows a family of linearly spaced ramps that can be mathematically represented as:

$$v(t) = \sum_{i=1}^{N} S_i(t)\alpha_i$$

$$S_i(t) = A\frac{t - t_{i,start}}{t_{i,end} - t_{i.start}} \quad t_{i,start} \leq t \leq t_{i,end}$$

$$S_i(t) = A \quad t > t_{i,end}$$

$$S_i(t) = 0 \quad t < t_{i,start}$$

Where A is the level of the step, 1 in the following figure, and $t_{i,start}$ and $t_{i,end}$ are the initial and final times of the ramp.

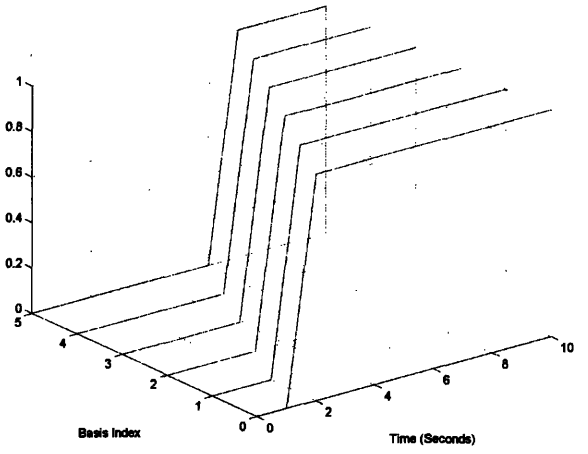


**Figure 4: Linearly Spaced Ramp Basis Functions**

## 2.2.2 Nominal Trajectories

In perturbation control basis functions are used to perturb the nominal input in order to record the response of the plant to small variations from the nominal trajectory. The nominal input trajectory plays an important role in perturbation control because it is the foundation from which all the perturbations are taken. There are two essential nominal trajectories in a system, these are the nominal input ($\underline{U}_N$) and the nominal output ($\underline{Y}_N$).

There is also the predicted output ($\underline{Y}$), control history ($\underline{U}$), output perturbation ($\underline{\Theta_Y}$) and input perturbation ($\underline{\Theta_U}$). For a SISO system, these can all be represented as follows:

$$(12)\quad \underline{U} = \begin{bmatrix} u(1) \\ u(2) \\ \vdots \\ u(Hp) \end{bmatrix} \in \Re^{Hp\times 1} \quad \underline{U_N} = \begin{bmatrix} u_N(1) \\ u_N(2) \\ \vdots \\ u_N(Hp) \end{bmatrix} \in \Re^{Hp\times 1} \quad \underline{\Theta_U} = \begin{bmatrix} \Theta u(1) \\ \Theta u(2) \\ \vdots \\ \Theta u(Hp) \end{bmatrix} \in \Re^{Hp\times 1}$$

$$(13)\quad \underline{Y} = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(Hp) \end{bmatrix} \in \Re^{Hp\times 1} \quad \underline{Y_N} = \begin{bmatrix} y_N(1) \\ y_N(2) \\ \vdots \\ y_N(Hp) \end{bmatrix} \in \Re^{Hp\times 1} \quad \underline{\Theta_Y} = \begin{bmatrix} \Theta y(1) \\ \Theta y(2) \\ \vdots \\ \Theta y(Hp) \end{bmatrix} \in \Re^{Hp\times 1}$$

The predicted output ($\underline{Y}$) can be represented as the sum of the nominal output ($\underline{Y_N}$) and the output perturbation ($\underline{\Theta_Y}$).

$$(14)\quad \underline{Y} = \underline{Y_N} + \underline{\Theta_Y}$$

The control history ($\underline{U}$) can similarly be represented as the sum of the nominal input ($\underline{U_N}$) and the input perturbation ($\underline{\Theta_U}$)

$$(15)\quad \underline{U} = \underline{U_N} + \underline{\Theta_U}$$

There are many ways to define the nominal inputs. They can be selected from a predefined set of valid maneuvers (variable bank rates, variable pitch rates, etc.). In this research we are not going to focus on the creation and selection of nominal trajectories, the nominal input is simply obtained by holding the last measurement of the input variable over the entire prediction horizon *Hp*.

The nominal output is the response of the plant, over the prediction horizon, to the nominal input. In the following equation the output expression for the plant is represented as the function g.

$$
\begin{aligned}
& y_N(k) = g(x(k), u_N(k), k) \\
& y_N(k+1) = g(x(k+1), u_N(k+1), k+1) \\
& y_N(k+2) = g(x(k+2), u_N(k+2), k+2) \\
& \vdots
\end{aligned}
\tag{16}
$$

At each step also the states are updated according to Equation 2.

## 2.2.3 Perturbations

As previously stated, the individual basis functions serve as the perturbation set to be run through the model. The perturbed trajectories at time k ($\underline{Y_i}(k).i = 1..B$) are the result of the application of the input perturbation basis function ($\underline{\Theta_{U,i}}(k)$, $i = 1..B$) over the entire prediction horizon ($t(k) \le t \le t(k) + Hp$). B is defined as the total number of basis functions used in the set.

$$
\begin{aligned}
& \underline{Y_1}(k) = g(\underline{X}(k), \underline{U_N}(k) + \underline{\Theta_{U,1}}(k), k) \in \Re^{Hpx1} \\
& \underline{Y_2}(k) = g(\underline{X}(k), \underline{U_N}(k) + \underline{\Theta_{U,2}}(k), k) \in \Re^{Hpx1} \\
& \vdots \\
& \underline{Y_B}(k) = g(\underline{X}(k), \underline{U_N}(k) + \underline{\Theta_{U,B}}(k), .k) \in \Re^{Hpx1}
\end{aligned}
\tag{17}
$$

In the calculation of the output vector $\underline{Y_i(k)}$ the state vector is also updated in accordance with Equation 2.

In the following equations, we are going to drop the time index k for the sake of brevity. As the output from the perturbed input are gathered however, the original nominal output must again be subtracted out to calculate the output perturbation. This creates B number of output perturbations ($\underline{\Delta Y_i}.i = 1..B$).

$$
\begin{aligned}
& \underline{\Delta Y_1} = \underline{Y_1} - \underline{Y_N} \\
& \underline{\Delta Y_2} = \underline{Y_2} - \underline{Y_N} \\
& \vdots \\
& \underline{\Delta Y_B} = \underline{Y_B} - \underline{Y_N}
\end{aligned}
\tag{18}
$$

These output differences are then collected and used to create the output perturbation matrix S.

**(19)** $$S = \lfloor \underline{\Delta Y_1} \quad \underline{\Delta Y_2} \quad \cdots \quad \underline{\Delta Y_B} \rfloor \in \Re^{HpxB}$$

In a similar fashion all the input perturbations ($\underline{\Delta U_i} = \underline{U_i} - \underline{U_N}, i = 1..B$) are collected and used to create the input perturbation matrix D.

$$D = \lfloor \underline{\Delta U_1} \quad \underline{\Delta U_2} \quad \cdots \quad \underline{\Delta U_B} \rfloor \in \Re^{HpxB}$$

The output perturbation matrix S can then be used to form the generic perturbed output ($\underline{\Theta_Y}$) in Equation 13 as a linear combination of the $\underline{\Delta Y_i}$ vectors.

**(20)** $$\underline{\Theta_Y} = S\underline{\alpha} \in \Re^{Hpx1}$$

The predicted output trajectory in Equation 14 now takes the following form for a generic output.

**(21)** $$\underline{Y} = \underline{Y_N} + S\underline{\alpha} \in \Re^{Hpx1}$$

In this formulation the independent variables that we are going to optimize are the scale factors $\underline{\alpha}$.

The input perturbations ($\underline{\Theta_U}$) in Equation 12 can now be expressed in terms of the basis functions and scaling factors.

**(22)** $$\underline{\Theta_U} = D\underline{\alpha} \in \Re^{Hpx1}$$

This leads to the following form of Equation 15.

**(23)** $$\underline{U} = \underline{U_N} + D\underline{\alpha} \in \Re^{Hpx1}$$

Where $\underline{U_N}$ is the nominal input and D is the input perturbation matrix. At this point we are making the fundamental linearity assumption between the input and output mapping.

Equation 24 is only valid for small perturbations from the nominal input. In fact, this is where the linearization assumption comes into play. Under this assumption the same scale factors applied to the output space can be applied to the input perturbations as well.

(24) $$\underline{Y} = \underline{Y_N} + S\underline{\alpha} \xrightarrow[linear]{} \underline{U} = \underline{U_N} + D\underline{\alpha}$$

This linearization assumption works also for the optimal scale factors that are calculated in the optimization performed in the output space. This fact is vital for MPC to be able to solve for the optimal input by applying the optimal scale factors found for the output. The optimization is done in the output space and the optimal scale factors applied to the input space.

## 2.3 Unconstrained Solution

We will now present how perturbation control can be formulated with the performance index described in Equation 4. The predicted output z of Equation 4 is replaced by the output Y and the reference output r is replaced by $Y_R$.

Equation 4 in matrix notation takes the following form.

(25) $$J = (\underline{Y} - \underline{Y_R})^T Q(\underline{Y} - \underline{Y_R}) + \underline{U}^T R\underline{U}$$

From Equations 21 and 23 we have:

$$\underline{Y} = \underline{Y_N} + S\underline{\alpha}$$
$$\underline{U} = \underline{U_N} + D\underline{\alpha}$$

Replacing the above definitions into Equation 25 we have:

(26) $$J = (\underline{Y_N} - \underline{Y_R} + S\underline{\alpha})^T Q(\underline{Y_N} - \underline{Y_R} + S\underline{\alpha}) + (\underline{U_N} + D\underline{\alpha})^T R(\underline{U_N} + D\underline{\alpha})$$

This is then further expanded into terms containing the scaling vector $\underline{\alpha}$ and those without.

$$J = (\underline{Y_N} - \underline{Y_R})^T Q(\underline{Y_N} - \underline{Y_R}) + (S\underline{\alpha})^T Q(S\underline{\alpha}) + (\underline{U_N})^T R(\underline{U_N}) + (D\underline{\alpha})^T R(D\underline{\alpha}) + (\underline{Y_N} - \underline{Y_R})^T Q(S\underline{\alpha}) + (S\underline{\alpha})^T Q(\underline{Y_N} - \underline{Y_R}) + (\underline{U_N})^T R(D\underline{\alpha}) + (D\underline{\alpha})^T R(\underline{U_N}) \quad (27)$$

It can be seen that terms not containing the scaling vector do not change the minimum of J because they only add a constant amount to the total cost. It is because of this that these terms can be neglected and the cost function further simplified.

$$J = (S\underline{\alpha})^T Q(S\underline{\alpha}) + (D\underline{\alpha})^T R(D\underline{\alpha}) + 2(\underline{Y_N} - \underline{Y_R})^T Q(S\underline{\alpha}) + 2(\underline{U_N})^T R(D\underline{\alpha}) \quad (28)$$

All that remains is to collapse the cost function into the standard form of a quadratic programming problem.

$$J = \frac{1}{2}\underline{x}^T H\underline{x} + f^T\underline{x} \quad (29)$$

This is done in the following equations by rearranging Equation 28.

$$J = \underline{\alpha}^T S^T QS\underline{\alpha} + \underline{\alpha}^T D^T RD\underline{\alpha} + 2(\underline{Y_N} - \underline{Y_R})^T QS\underline{\alpha} + 2(\underline{U_N})^T RD\underline{\alpha}$$

$$J = \underline{\alpha}^T (S^T QS + D^T RD)\underline{\alpha} + 2((\underline{Y_N} - \underline{Y_R})^T QS + \underline{U_N}^T RD)\underline{\alpha} \quad (30)$$

The problem is now in the standard form for a QP solution by only adding a scaling factor of 2 to the H matrix

$$H = 2(S^T QS + D^T RD) \in \Re^{B\times B}$$

$$f = 2((\underline{Y_N} - \underline{Y_R})^T QS + \underline{U_N}^T RD)^T \in \Re^{B\times 1} \quad (31)$$

The optimal α can now be easily solved for by setting the derivative of the cost function to zero.

$$J = \frac{1}{2}\underline{\alpha}^T H\underline{\alpha} + f^T\underline{\alpha}$$

$$\frac{dJ}{d\alpha} = \frac{1}{2}\left[H\underline{\alpha} + H^T\underline{\alpha}\right] + f = H\underline{\alpha} + \underline{f} = 0 \quad (32)$$

$$\underline{\alpha_{opt}} = -H^{-1}\underline{f}$$

The optimal scaling vector $\alpha_{opt}$ can be applied to the output perturbation matrix and added to the nominal output for the projected output.

(33) $$\underline{Y_{opt}} = \underline{Y_N} + S\underline{\alpha_{opt}}$$

The key to this entire perturbation process is that at this point we assume a linear relationship between the input and output of the plant and therefore the optimal scaling factor applies also to the input. The optimal input takes the following form.

(34) $$\underline{U_{opt}} = \underline{U_N} + D\underline{\alpha_{opt}} = \underline{U_N} - DH^{-1}\underline{f}$$

Again this optimal input vector is computed for the entire control horizon. The designer can choose whether to implement the first N inputs before having to repeat the process. The designer then has the choice to set the MPC frequency and consequently the amount of time the control is used. At the beginning of the next cycle this entire process is then repeated.

## 2.4 Constrained Solution

The presence of constraints can make the unconstrained solution invalid. This is true because the unconstrained solution is only valid as long as the constraints are not violated. Once the constraints are violated it becomes a quadratic programming (QP) problem and must be handed over to a dedicated QP solver. QP solvers operate in the following framework.

(35) $$\min J = \frac{1}{2}\underline{x}^T H \underline{x} + f^T \underline{x}$$

$$\text{subject to } \underline{b_{\min}} \le A\underline{x} \le \underline{b_{\max}}$$

Since we are solving for the optimal α, but we want to set constraints on the input and/or the output, these constraints must be written in terms of α. The H and f matrices of

Equation 31 are identical to the ones described in the previous section. In order to construct the proper constraint matrices the following relationship must be upheld.

$$\begin{aligned} \underline{Y_{\min}} &\le \underline{Y_N} + S\underline{\alpha} \le \underline{Y_{\max}} \\ \underline{U_{\min}} &\le \underline{U_N} + D\underline{\alpha} \le \underline{U_{\max}} \end{aligned} \tag{36}$$

It is now apparent that in order to place input and output constraints on $\underline{\alpha}$, the nominal must be subtracted from the actual minimum and maximum values. This is further illustrated below.

$$\begin{bmatrix} \underline{Y_{\min}} - \underline{Y_N} \\ \underline{U_{\min}} - \underline{U_N} \end{bmatrix} \le \begin{bmatrix} S \\ D \end{bmatrix} \underline{\alpha} \le \begin{bmatrix} \underline{Y_{\max}} - \underline{Y_N} \\ \underline{U_{\max}} - \underline{U_N} \end{bmatrix} \tag{37}$$

Thus the constraint matrices for input and output to be sent to the QP solver are as below.

$$\underline{b_{\min}} = \begin{bmatrix} Y_{\min} - Y_N \\ U_{\min} - U_N \end{bmatrix} \quad \underline{b_{\max}} = \begin{bmatrix} Y_{\max} - Y_N \\ U_{\max} - U_N \end{bmatrix} \quad A = \begin{bmatrix} S \\ D \end{bmatrix} \tag{38}$$

After the optimal $\underline{\alpha}$ is calculated, the result is multiplied by the perturbation matrix and the nominal input added to create the optimal input.

$$\underline{U_{opt}} = \underline{U_N} + D\underline{\alpha_{opt}} \tag{39}$$

## 2.5 MPC Example

In order to further clarify the MPC nonlinear perturbational control method, an example is offered. In this example a complete six degree of freedom nonlinear discrete high performance fighter aircraft model will serve as the plant. This simulation begins trimmed at an altitude of 25,000 feet and a speed of Mach 0.8. The maneuver will be a controlled bank from 0 to 10 degrees with a 5 second ramp. The aircraft already has a stabilizing inner loop and receives roll rate commands from the MPC controller (P Cmd). The Control Augmentation System (CAS) sends the control surface commands to the aircraft. This can all be seen in Figure 5.

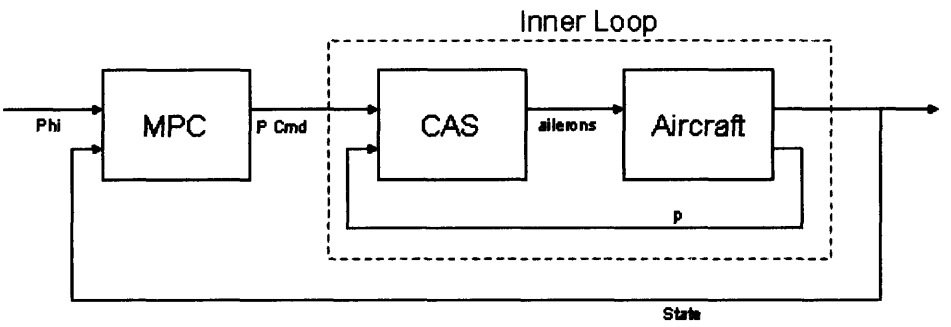


**Figure 5: Roll Control Loop**

The MPC controller will receive the bank profile Phi and the state at which to initialize its propagator. It is important to understand that the aircraft with the inner loop is the internal model of the MPC controller. The internal model is sampled at 50 Hertz. In order to create the input, linearly spaced step functions are used. All that remains to be chosen are: the number of step functions, the prediction horizon, and the state weightings in the cost function.

The number of step functions can be chosen in a straight forward method. It must first be realized that basis functions are mere approximations to a complete nominal control profile. This nominal profile is the result of having each possible control free while executing the optimization. For instance, with a prediction horizon of 4 seconds and internal model sampled at 50 Hz, the total number of free controls is 200. Moreover, if one iteration of the MPC algorithm is run, one complete control profile tailored for the entire prediction horizon is produced. If all controls are left free, a specific optimal cost can then be associated with this optimal control profile. This is the ideal optimal cost and as basis function numbers are increased, the cost associated with them will approach this ideal cost. This exercise was done with a linear spaced basis set. The cost was recorded running the first iteration cycle of the MPC controller. The results can be seen in Figure 6.

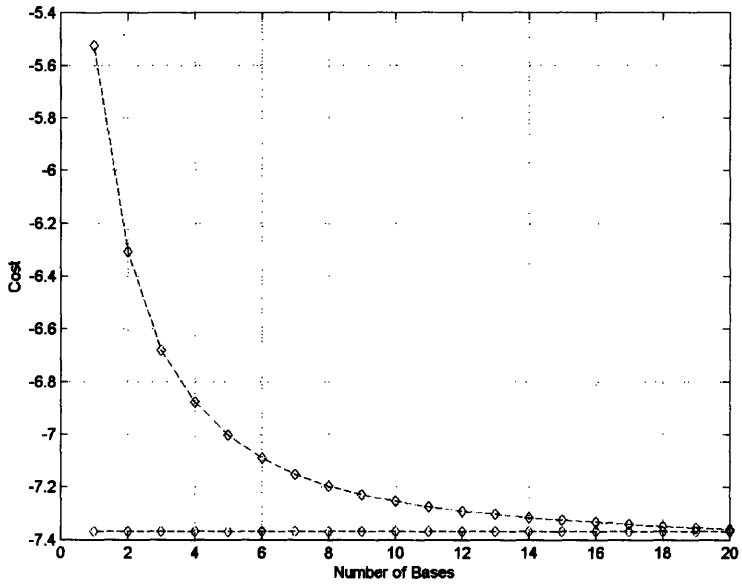


**Figure 6: Basis Cost Comparison**

The bottom line in Figure 6 is the minimum optimal cost achieved when we allow the input to change at every step. Figure 6 shows that as the number of basis functions is increased, the cost approaches its minimum. Using this information a basis number can be systematically chosen. To do this, a percentage error can be defined to show how close the basis set is to the ideal in the following manner.

**(40)**

$$\%CostError = \frac{(J_B - J_I)}{J_I} * 100$$

$J_B$ is the cost associated with each specific basis number and $J_I$ is the ideal cost when there are no basis functions used. The percentage cost error is plotted in Figure 7.

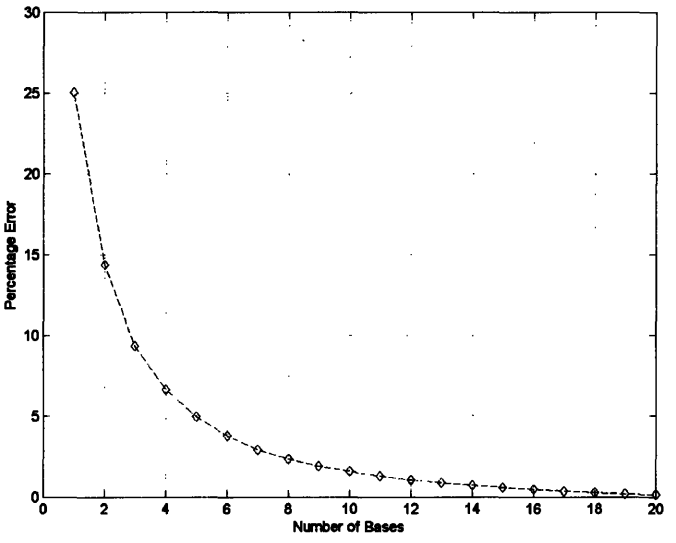


**Figure 7: Basis Percentage Error**

For this example a 2% error is chosen for the acceptable percent error which corresponds to 10 basis functions. A plot of these functions can be seen in Figure 8.

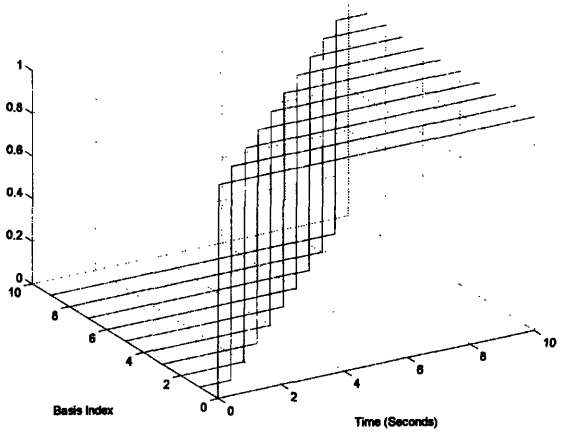


**Figure 8: 10 Linearly Spaced Step Basis Functions**

The prediction horizon can be chosen in a similar systematic way. It must first be realized that the minimum prediction horizon has to be large enough to include the full dynamic response of the system to be controlled. For instance, if a second order damped system was to be controlled, the prediction horizon should be long enough to include at least the 5% settling time. If the MPC controller propagates for an amount shorter than this, it might not have enough information about the dynamic behavior of the system.

The result will be a potentially unstable controller. Since MPC can be computationally intensive, the goal is to reduce the prediction horizon to its minimum. The only constraint therefore is this slowest system dynamic time. Making the prediction horizon longer than that, will not increase performance but only penalize computational time. For this example, the full MPC algorithm was run with 10 bases and prediction horizons from 2 to 6 seconds.

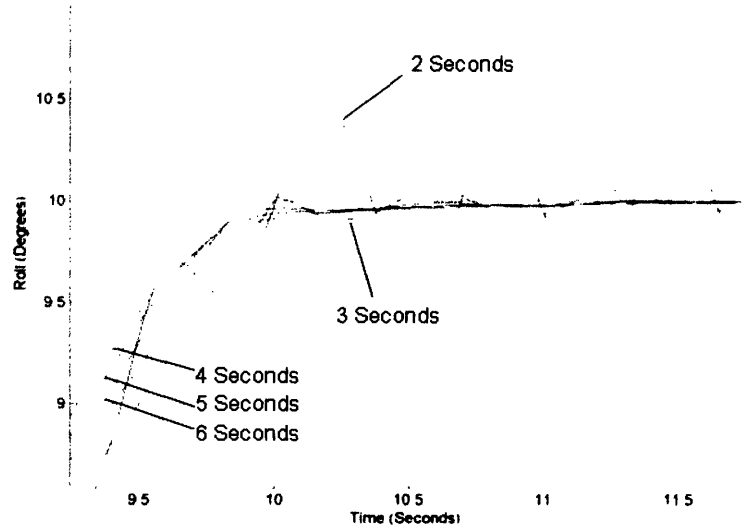


**Figure 9: Prediction Horizon Comparison**

It can be seen in Figure 9 that as the prediction horizon increases, the response stabilizes. Based on this plot, a prediction horizon of 4 seconds can be selected.

The final parameters to be tuned are that of the Q and R matrices used for state and input weighting. Because this is a single input and output system, there can be no use of the Q and R matrices to weight against other inputs and outputs. In the case of a multiple input and output system, Q and R can be used to penalize one output or input more than another. The next degree of freedom is to weight the Q and R matrix diagonal within the prediction horizon itself. This can be used to penalize output errors or inputs at earlier or later times in the response. Again these weightings are relative to within the prediction horizon so typical design involves holding one gain at unity and varying another. In this example, a weighting of 1 across the prediction horizon was nominal.

Now that all the free parameters have been chosen, the only remaining element is to incorporate the receding horizon principle into the simulation. The MPC cycle can be run with 10 linearly space basis functions and a 5 second prediction horizon iterating at 5 Hz. Figure 10 shows the response of the system to a ramp roll input.

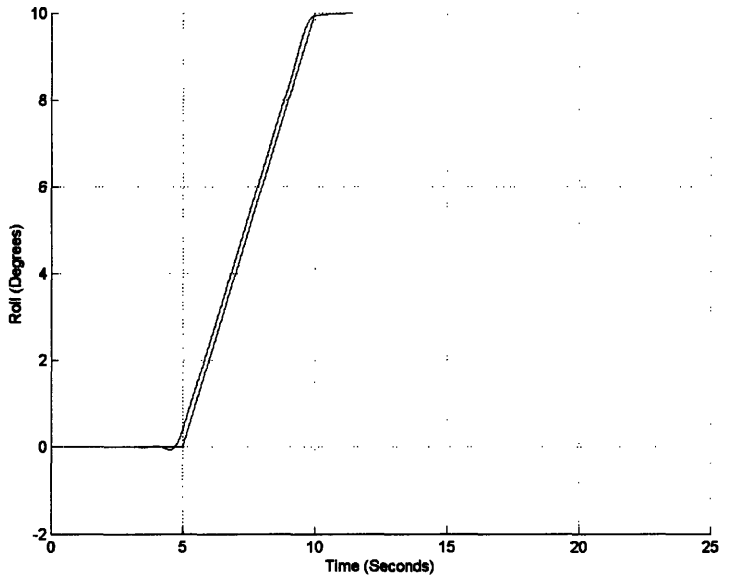


**Figure 10: Roll Control Final Implementation**

The MPC anticipatory behavior can be noticed as the system starts responding to the input ramp before encountering it. The controller starts reacting as soon as the commanded maneuver enters into the prediction horizon. The initial oscillation in the controller is due to the fact that we are minimizing the quadratic difference between the reference and the response. From a mathematical point of view the solution with this small oscillation is the one that uses the minimum cost. The input profile for the maneuver can also be seen below in Figure 11.

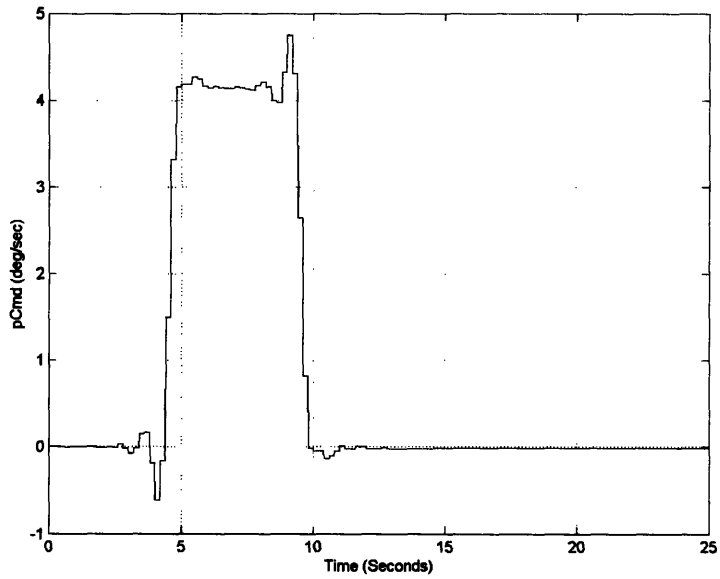


**Figure 11: Final Control Profile**

The MPC perturbational technique described in this chapter will now be used to develop various autopilots for a generic high performance aircraft. The following chapter will provide a detailed description of the vehicle and a few modifications introduced to augment the redundancy of the system.

[This Page Intentionally Left Blank]

# Chapter 3

# Vehicle Truth Model

## 3.1 Vehicle Description

This thesis will employ a high performance aircraft as the plant to be controlled. This craft consists of a generic aircraft twin rudder swept wing frame with twin turbofan engines capable of 16,000 pounds of static thrust. Table 1 shows the aircraft's physical specifications.

**Table 1: Physical Characteristics**

| Physical Characteristic of High Performance Aircraft | |
|---|---|
| Weight, lb | 30,802 |
| Reference wing area, $ft^2$ | 400 |
| Reference m.a.c., ft | 11.52 |
| Reference Span, ft | 37.42 |
| Wing Aspect Ratio | 3.5 |

The craft has a total of 10 aerodynamic surfaces for control, arranged as 5 pairs. There are three main pairs of control surfaces that induce the largest moments in the three principal axis. They are-

- Horizontal tails- Engaged in symmetric deflection for longitudinal maneuvers
- Rudder pair- Engaged symmetrically for both heading changes and roll maneuvers.
- Ailerons- Used differentially to induce roll moments

The remaining two pairs are the wing leading and trailing edge control surfaces. These are typically used for trim at various flight points and for other minor attitude corrections. Table 2 shows the complete specifications of all aerodynamic surfaces.

**Table 2: Flight Surface Characterizations**

| Surface | Saturations Limit (deg) | Rate Limit | Bandwidth (Hz) |
|---|---|---|---|
| Elevator (Left & Right) | -24 to 10.5 | -40 to 40 | 6.9 |
| Rudder (Left & Right) | -30 to 30 | -82 to 82 | 11.7 |
| Ailerons (Left & Right) | -25 to 45 | -100 to 100 | 13.8 |
| Leading Edge (Left & Right) | -3 to 33 | -3 to 33 | 3.9 |
| Trailing Edge (Left & Right) | -8 to 45 | -18 to 18 | 5.5 |

In addition to these aerodynamic control surfaces the aircraft is equipped with a thrust vectoring system. External paddles are used to deflect exhaust and produce additional aircraft moments. These paddles can be deflected either symmetrically or differentially to produce moments in any direction. This thrust vectoring system will prove to be extremely effective in controlling the aircraft in case of a failure.

The aircraft model is decomposed into its major subsystems: the aircraft's aerodynamics, the propulsion systems, the six degree of freedom full nonlinear differential equations, the atmospheric model, the inner loop CAS, and the air data subsystem. The model's major assumptions are as follows:

- Aircraft body is rigid
- Earth is an inertial reference frame
- Aircraft mass is constant
- The aircraft is symmetric across the x-z plane.

## 3.2 Model Framework

Typically stand alone executables written in C, C++, and FORTRAN demonstrate high computational speed and accuracy. Other simulations however employ more user friendly environments such as Simulink by MathWorks, Inc, but sacrifice speed. SIMULINK is a standard simulation environment which works in unison with MathWork's Matlab. This is a powerful simulation tool with an intuitive graphic interface, input/output support, and basic linear algebra functionality. Due to Simulink's graphic interface, signals can be injected anywhere in the system and loops broken at any spot. Unfortunately, this is all but absent in the faster stand alone executables.

Simulink allows functions written in C, C++, and other languages to be executed within its environment. These are known as S-Functions or system functions. This hybrid environment combines the rapid and intuitive capability of Simulink with the speed of such languages as C, C++, Ada, and FORTRAN. For example, the aerodynamic, engine, and rigid body equations of motion subsystems are all included as S Functions in the aircraft model shown in Figure 12.

## 3.3 Six Degree of Freedom Non-linear Vehicle Model

This thesis will employ a full six degree of freedom high performance aircraft simulation composed of many separate S-Functions written in the C language (C S-Functions). The simulation is depicted in Figure 12.

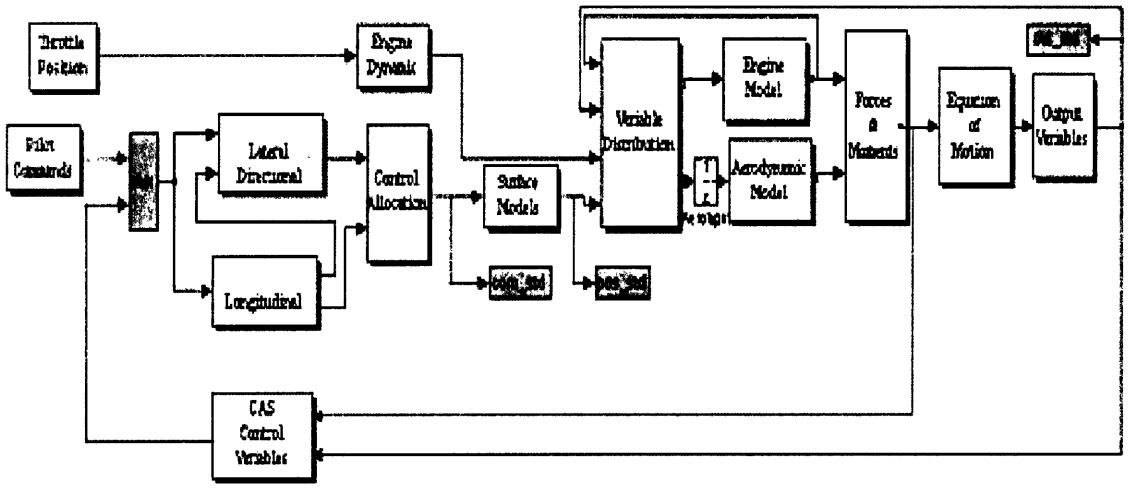


Figure 12: Truth Model

As with most high performance aircraft, there is a closed loop with Control Augmentation System (CAS). The CAS function is to augment the plant open loop dynamics and achieve the desired flight characteristics. It is a classical flight control system that heavily relies upon gain scheduling to obtain consistent performance throughout the flight envelope. Due to the high level of decoupling between the longitudinal and lateral dynamics, the CAS is separated into longitudinal and lateral systems. The CAS takes pilot stick inputs and uses them to compute the proper surface deflections.

At the beginning of each simulation an initialization function must be run in order to load all constants and initial states for the appropriate C S-Functions. Furthermore, a flight condition must be selected and pre-computed trim data loaded to start the simulation.

### 3.3.1 Subsystem Description

Certain boxes in the truth model shown in Figure 12 are C S-Functions while others are simulink blocks combining other functions. The following is a description of each C S-Function.

- Engine Dynamics: Implements a second order dynamic system when throttle input changes with time. The input is throttle and the output a time lagged throttle command.

- Engine Model: Calculates thrust and moment due to engines, in the body axes, as functions of Mach number, altitude, angle of attack, throttle, and thrust vector vane deflection.
- Aerodynamic Model: Calculates aerodynamic forces and moments as well as the six aerodynamic coefficients. Its inputs are Mach number, altitude, angle of attack, throttle, thrust vector vane deflection, all control surface positions, geometric data of the plane, and center of gravity position. The aerodynamic coefficients are then calculated by means of complex interpolation tables.
- Equation of Motion: Calculates all six degrees of freedom and derivatives by solving the complete nonlinear equations of motion. The inputs are all forces and moments and the aircraft mass properties.
- Output: In addition to complete position and rate, uses state variables to compute other information such as flight path angle, gamma, and sideslip angle. Also uses an atmospheric model to look up the speed of sound, air density, gravity, static pressure and air temperature as a function of altitude.

Other blocks in this simulation are constructed of various simulink functions. The following is a list of them and their specific functions.

- Lateral CAS: Calculates commands for the ailerons, leading and trailing edges, and vertical tail based on lateral stick commands. It does this by using angular rate and acceleration data from gyros and accelerometers.
- Longitudinal CAS: Calculates commands for the ailerons, leading and trailing edges, and horizontal tail based on longitudinal stick commands. It does this by using angular rate and acceleration data.
- Surface Models: Utilizes second and fourth order models to simulate actuator/control surface dynamics. It also incorporates rate and deflection limit saturations.
- Variable Distribution: Organizes inputs appropriately to be sent to the various aerodynamic and engine models.
- Forces and Moments: Combines force and moments due to aerodynamics and the engine to be sent to the rigid body dynamics.

These system blocks are combined to create the full non-linear closed loop simulation. The simulation is run at 50 Hertz with a Runge Kutta integration scheme.

### 3.3.2 Coordinate Frame and Output Description

The aircraft motion is expressed by defining the following frames.

- The inertial frame (i): The inertial reference frame is Earth centered with due north as the positive x axis and due east as the positive y axis and the z axis pointing into the ground.
- The local horizontal frame (h): The local horizon frame is identical to the earth inertial frame but is centered on the vehicle's center of mass.
- The body frame (b): The body frame is determined by attaching a frame to the aircraft with the x axis pointing out the nose, y axis out the right wing, and z axis positive down to the earth.

The major attitude tracking variables are then defined from the difference between the body frame and the local horizontal frame. This can be seen in the following set of figures.

The heading angle ($\psi$) is defined as the angle between the x body axis and the x local horizontal axis when the body frame is rotated relative to the local horizontal frame about the z axis, as shown in Figure 13.

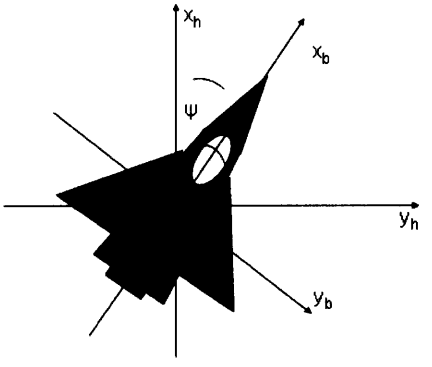


**Figure 13: Heading Description**

Figure 14 shows theta (θ) to be defined as the angle between the x body axis and the x local horizontal axis, when the body frame is rotated relative to the local horizontal frame about the y axis. This angle is also referred to the Euler pitch angle. P is then the roll rate about the x body axis and r the heading rate about the z body axis

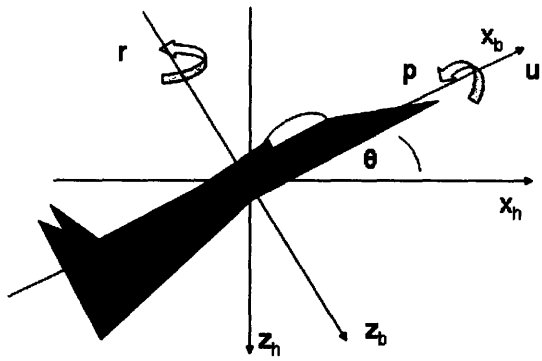


**Figure 14: Theta Description**

Phi (φ) is defined as the angle between the y body axis and the y local horizontal axis, when the body frame is rotated relative to the local horizontal frame about the x axis. It is referred to as the Euler roll angle. The variable q is then the rotation rate about the y body axis and v is the speed along this same axis. These are referred to as pitch rate and lateral speed.

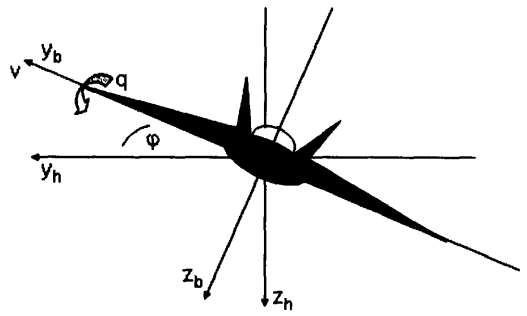


**Figure 15: Phi Description**

The Euler rotation sequence used in the simulation is a 321 or ψ, θ, and φ sequence. The aircraft motion is also described in terms of its flight path components. These

components are defined relative to the flight path of the vehicle and can be derived from the Euler angles and velocity components. The simulation outputs both of these variable types. A table featuring a complete list of the simulation outputs can be seen below.

Table 3: Output Variables

| | | |
|---|---|---|
| U | Velocity in x body axis | ft/s |
| V | Velocity in y body axis | ft/s |
| W | Velocity in z body axis | ft/s |
| phi | Euler roll angle | rad |
| theta | Euler pitch angle | rad |
| psi | Euler yaw angle | rad |
| p | Roll rate | rad/s |
| q | Pitch rate | rad/s |
| r | Yaw rate | rad/s |
| Pos_north | Position North | ft |
| Pos_east | Position East | ft |
| altitude | Altitude | ft |
| Vt | Total true airspeed | ft/s |
| Mach | Mach number | - |
| alp | Angle of attack | rad |
| beta | Sideslip angle | rad |
| qbar | Dynamic pressure | Slug/(ft*s$^2$) |
| ps | Static pressure | psf |
| mu | Flight path angle | rad |
| gamma | Angle between mu and horizontal plane | rad |

## 3.4 Modifications

In order to have a redundant system capable of recovering from surface failures a few modifications to the baseline aircraft have been made. The input to the CAS had to be modified and a thrust vectoring system was added to the plant.

In order to design an autopilot for this CAS the existing stick inputs had to be replaced with appropriate command signals for all three principal axes. The selected autopilot commands were the roll, pitch and yaw rates. Error signals were calculated using the feedback attitude rates as shown in Figure 16. Furthermore, the inner loop CAS was

augmented with a control law for the thrust vectoring system since one was not present in the baseline plant. A simple proportional controller was designed around the thrust vectoring system. The criteria for gain selection were based on the responses of the system to failures of the longitudinal and lateral-directional control surfaces.
This set up is shown in Figure 16.

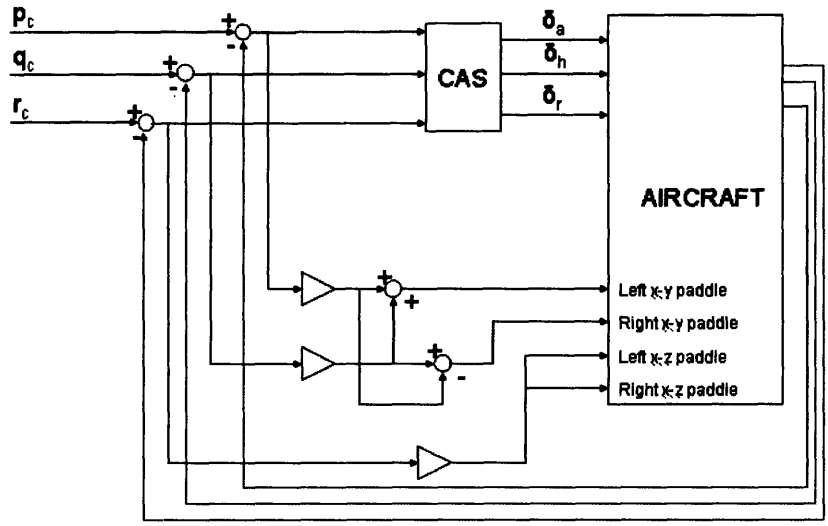


**Figure 16: Thrust Vectoring Control**

A gain multiplies the pitch rate error and is sent to both thrust vector vanes creating symmetric movements in the x-y paddles to produce moments about the y axis. Another gain then multiplies roll rate error and sends a differential signal to the two x-y paddles. Finally a third gain multiplies the yaw rate error and creates a symmetric input to the x-z paddles to produce moments about the z axis.

An additional modification was needed to achieve redundancy in the control surfaces. The three main surface pairs were artificially split in order to simulate multiple surfaces. Each of these surfaces was divided into two equal independent control effectors able to receive separate command inputs. This is further described in Chapter 5.

## 3.5 Autopilot Mode Definitions

The MPC autopilot uses the inner loop command signal to achieve different types of trajectory and attitude tracking. Depending on the mode of operation the high level

autopilot commands such as altitude to be gained, pitch angle to be held, are translated into basic commands (pitch, roll, and yaw rates) for the inner loop CAS. It is then the responsibility of the inner loop to respond and track the outer loop commands by issuing the proper surface deflections. The inner loop CAS is typically a higher bandwidth function then that of the autopilot and operates at a higher loop rate. In this thesis the following autopilot modes will be implemented using MPC:

- Pitch Hold: Acquires and maintains aircraft pitch attitude
- Bank Hold: Maintains wings level flight or tracks a bank angle
- Altitude Capture: Tracks an altitude input profile
- Heading Hold: Acquires and maintains aircraft heading

These separate autopilot modes can be combined to perform an entire mission profile for the aircraft to fly autonomously. For instance, an altitude profile with various heading or bank commands can by flown by the aircraft selecting the appropriate autopilot modes. For example, Figure 17 shows an altitude capture followed by heading hold and followed by another altitude capture.

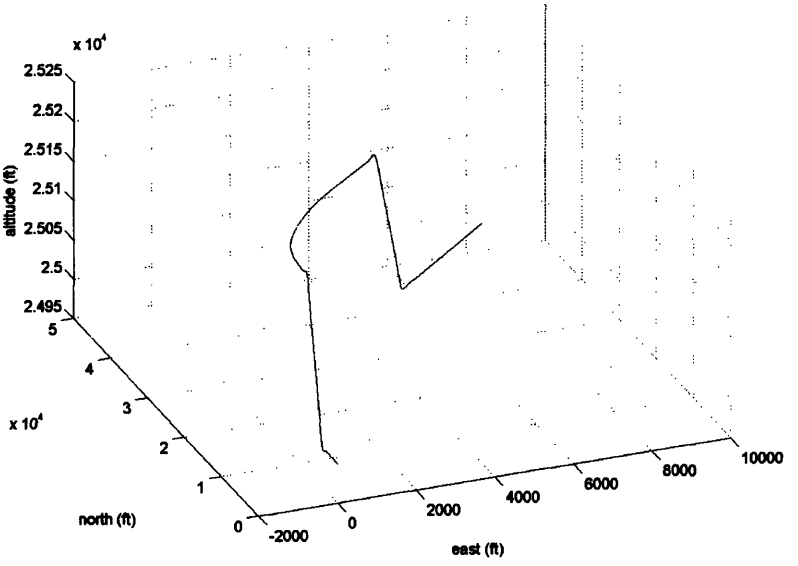


**Figure 17: Sample Flight Path**

# Chapter 4

## MPC Controllers

## 4.1 Controller Overview

Figure 18 shows the general structure of the flight control system used in this research. The fighter truth model includes the aircraft dynamics and the inner loop CAS controller. In this particular case the MPC autopilot combines the altitude capture and heading hold mode of operation. The outputs of the MPC controller are the pitch and yaw rates. The reference inputs are the heading and altitude profiles. Full state feedback is provided to the MPC controller in order to initialize the rigid body states and the CAS states. The nominal trajectory is generated by holding the current pitch and yaw rates constant.

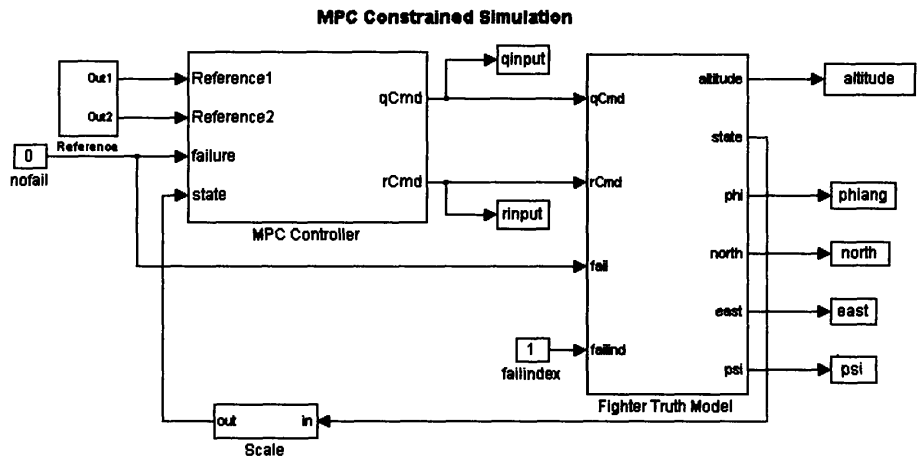


**Figure 18: MPC Outer Loop**

The MPC controller is implemented in two main C S functions. The first C S function is essentially preprocessing for initialization. In this function, references, failure signals and the initial state are received. The matrices H, f, A, bmin, and bmax (described in Section 2.5) are assembled according to the MPC perturbation methodology described in Chapter 2. This function can be seen in the actual controller implementation shown in

Figure 19 as the block MPC PREP. A description of how this S function was implemented can be found in Appendix B.

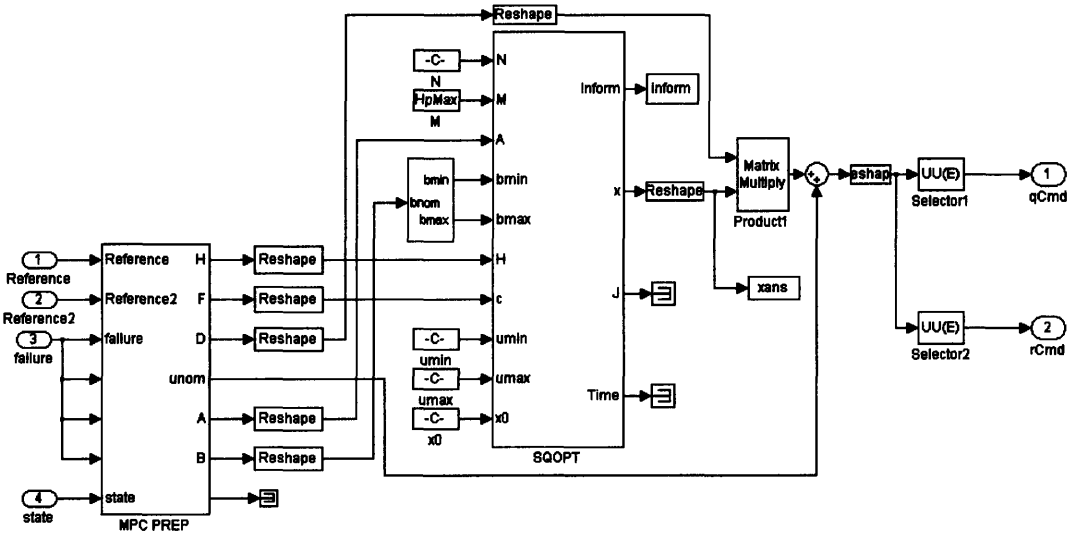


**Figure 19: MPC Controller Structure**

The H and f matrices together with the constraints A, bmin, and bmax are then sent to the QP solver for optimization. The QP solver uses SQOPT (see next section), a commercial quadratic programming software package. The optimum scaling vector (x output of the SQOPT block) is multiplied by the D matrix in the matrix multiply block. The result is added to the nominal input and then sent to the appropriate control. A selector is also used to select the first input from the control sequence. The sampling rate of the MPC controller is set in the MPC PREP S function and the SQOPT S function.

The setup described in Figure 19 is a general structure developed at Draper Laboratory and used in a variety of applications. This Simulink implementation has been proven particularly useful because of its flexibility and simplicity.

## 4.2 Quadratic Problem Solver

The QP solver used is called SQOPT and it is a product of the Systems Optimization Laboratory of Stanford University. SQOPT is distributed by Stanford Business Software Inc. It is a general large scale linear and quadratic program solver. More information can be found at website http://www.sbsi-sol-optimize.com. The optimization software is written in FORTRAN and was inserted into a S function for use with the MPC algorithm.

## 4.3 Internal Model

The internal model MPC is implemented in C. It was created by simplifying the original truth model and using Real Time Workshop to automatically code a self contained C function. Real Time Workshop is a product of MathWorks, that generates C code from simulink files. The resulting code has an initialization function and a step function that can be used to propagate the input over the time horizon.

The inputs into the model include the p, q, and r commands as well as the 16 failure signal and control surface positions for each surface. The control surface positions are needed to override the model control surface positions in case of surface failures (for a detailed description of the mechanization of the MPC internal model update in case of a failure see Section 5.3). The outputs are the twelve rigid body states and the angle of attack. The MPC propagator uses the full nonlinear model of the plant to predict the plant's behavior. A more detailed explanation of how this model was created can be found in Appendix A.

### 4.3.1 Model Modifications

In order to autocode the simulink diagram, certain modifications had to be made. This involved removing the integrators from all the continuous S functions. The integration is then performed outside the S function using the Simulink discrete integrator block in the resetable mode. In this way all the states of the model are collected by Real Time Workshop in one C data structure that can be easily initialized at a given point before starting the propagator.

Furthermore, the internal model had to have the ability to be updated to reflect the actual failure occurrence in the plant. Only aerodynamic control surface failures were modeled. No thrust vector control failures were considered. The internal model thus has separate inputs for actual measured control surface positions. Surface deflections due to failures are inserted into the model when a control surface failure is simulated. Otherwise the model uses the plant's own control surface position predictions. A proper description of the failure types and method of implementation can be seen in Chapter 5.

## 4.4 Basis Function Selection

Basis functions can be broadly divided into two categories for the purpose of this research. The first consist of identical functions that are spaced in time relative to their basis index. Examples of these are the linearly spaced step functions, linearly spaced tent functions and linearly spaced ramp functions. The amount by which these functions are displaced temporally can be altered as well. This is typically done to increase the fidelity of control in a specific region in the control horizon. In any case, the control horizon is spanned by tiling in time a simple function throughout the set.

The other family consists of orthogonal polynomial functions which span the entire control horizon. For these basis sets, the individual functions are "smooth" and change throughout the horizon. Typically there are no regions with constant values for extended time periods. Their differences are not created by simple translation in time but through the nature of their recurrence algorithm. Some examples of these are Laguerre, Legendre, Chebyshev and Hermite polynomials. Thanks to the orthogonality of these functions, it was found that fewer basis functions were needed to achieve performance similar to that achieved using temporally spaced basis functions. In some cases only fourth order Laguerre polynomials were needed instead of $10^{th}$ order time spaced basis functions. As described eariler, reducing the number of basis functions reduces the search space of the QP algorithm.

The basis functions used in this research are exponentially weighted Laguerre polynomials. Laguerre polynomials are solutions to the Laguerre differential equation.

$$L_n(x) = \frac{e^x}{n!}\frac{d^n}{dx^n}(x^n e^{-x}) \text{ where n=0,1,2,3...}$$

A plot of the first 6 Laguerre functions can be seen below.

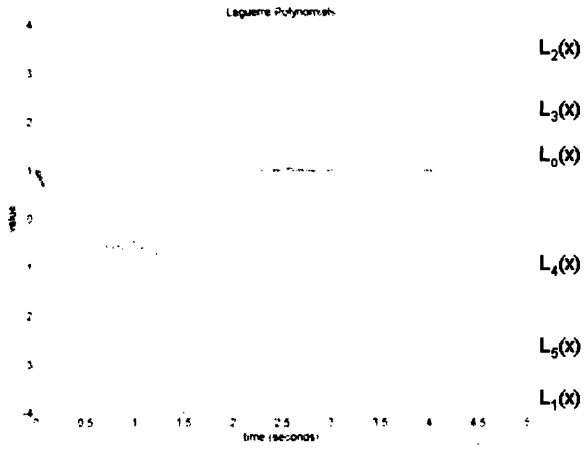


**Figure 20: Laguerre Polynomials**

In order to make these functions orthogonal an exponential weighting term of $e^{-x}$ must be added. Laguerre polynomials $L_n(x)$, n=0,1,2,3..., only form a complete orthogonal set over the internal $0<x<\infty$ with respect to the weighting function. This is represented mathematically in the following.

$$\int_0^{\infty} e^{-x} L_m(x) L_n(x) dx = \begin{cases} 0 & m \neq n \\ 1 & m = n \end{cases}$$

Thus the orthogonal basis function itself can be represented as the following.

$$b_i(x) = e^{-\frac{x}{2}} * L(x)$$

The algorithm incorporated in the autopilot MPC controllers satisfies the following recurrence relationship.

$$(n+1)L_{n+1}(x) = (2n+1-x)L_n(x) - nL_{n-1}(x)$$

After the Laguerre polynomial is calculated it is then multiplied by $e^{-x/2}$ and used as the basis set. Again, these polynomials are used throughout this research. Although the set is not truly orthogonal over the 4 second prediction horizon they are used, the integrals of non identical ones converge to near zero within this time frame. Typically only 4 weighted polynomial basis functions are needed to provide effective control signals.

A more quantitative rationale for choosing exponentially weighted Laguerre basis functions over linearly spaced functions is now presented. Both exponentially weighted Laguerre and linearly spaced basis functions were used to track an exponential roll

command. This is using the same roll control loop with inner lops CAS as described in Section 2.5. The form of the tracking variable phi is:

$$\phi = 0 \quad \text{for } t < 5$$

$$\phi = (1 - e^{-x}) * 20 \quad \text{for } t \geq 5$$

A plot of this maneuver using both Laguerre and linearly spaced steps with 6 basis functions can be seen in Figure 21 for further clarity.

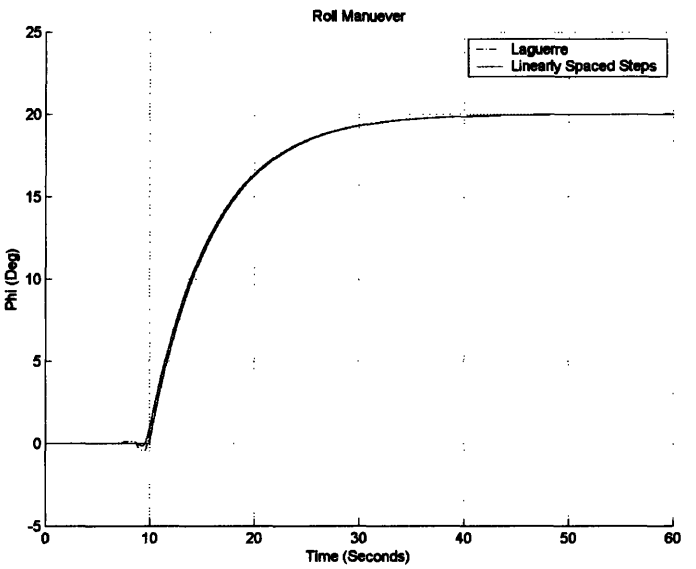


**Figure 21: Exponential Maneuver Example**

The 2 norm error, between the reference and the output, over the entire interval was recorded using an increasing number of basis functions for both Laguerre and linearly spaced steps. The results are shown in Figure 22.

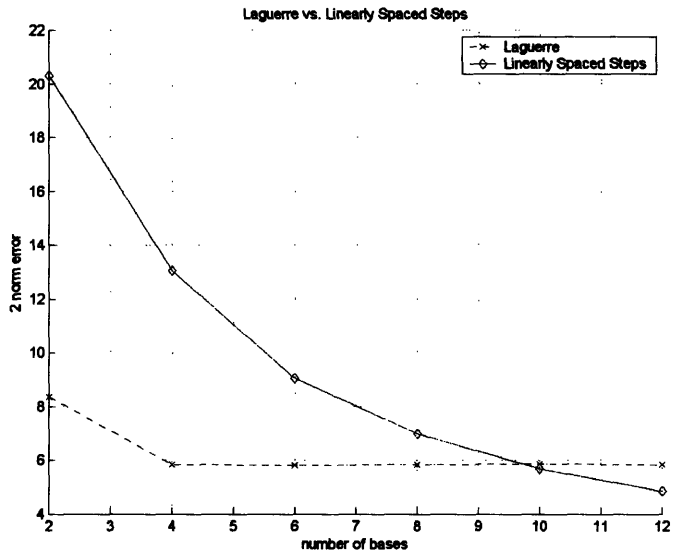


**Figure 22: Laguerre/Linearly Spaced Step Error Comparison**

It can be seen that the weighted Laguerre polynomials provide a substantially lower 2 norm error initially. Furthermore, after only four bases the 2 norm error settles to about 5.8. Although linearly spaced steps eventually drop below this constant error, it does not happen until after 10 bases are used. Actual implementation of 10 bases proved to be impractical due to the large computational time using this many bases demands. For this reason, four exponentially weighted Laguerre basis functions were chosen to serve as the basis function set. A detailed discussion of further basis function comparisons can be found in reference [4].

## 4.5 Simulation Rates

There are four simulation rates that must be assigned: overall simulation rate, inner loop rate, outer loop rate, and prediction rate. The simulation rate is the core rate of the simulation. This must be equal to or a multiple of the fastest rate in the simulation. A simulation rate of 50 Hz was selected. The prediction frequency must also be 50 Hz because it is modeling the CAS inner loop system which runs at this rate. The internal model must be discretized at the rate at which the inner loop CAS is run. The outer loop however can afford to be the slowest of all because its commands are being tracked with the help of the stabilizing inner loop. In this research an outer loop rate of 5 Hz is used.

This is to say that the MPC controller only applies a different control input at every 0.2 seconds. The inner loop then keeps this constant control and simulates at its rate of 50 Hz. In most atmospheric vehicles, with a stabilizing inner loop, an autopilot rate of 5-20 Hz is sufficient.

## 4.6 Autopilot Design

In the following sections we present four autopilot modes used in this research: Pitch Hold, Bank Hold, Altitude Capture, and Altitude and Heading Hold.

### 4.6.1 Pitch Hold

The first autopilot mode is the simple pitch hold. In this mode, the pitch angle can be commanded to any position while maintaining steady flight. The control input to the aircraft CAS is q or pitch rate and the variable tracked is pitch angle theta ($\theta$). The specifications for this mode are summarized in Table 4.

**Table 4: Pitch Controller Specifications**

| Tracking Variable | MPC Output | CAS Output | Controller Rate |
|---|---|---|---|
| $\theta$ | q | Horizontal tail surface position | 5 Hz |

The weightings for this controller are similar to those found in the rest of the research. All weightings will have the first portion of their time horizon more heavily weighted then later portions in the diagonal Q matrix. This is due to the tendency of the controller to anticipate any movement in the future directly as it comes into the prediction horizon view. Laguerre polynomials are relatively complex and their intrinsic oscillations force the controller to find scaling factors inheriting these oscillations. Fortunately, these premature oscillations can be muted by simply weighting the first half of the prediction horizon. In this pitch controller, as well as with all controllers, the first half of the prediction horizon was weighted at a ratio of 200:1.

The following is a simple demonstration of the pitch autopilot. Figure 23 shows a pitch doublet from 0° to +30° to -30° to 0° .

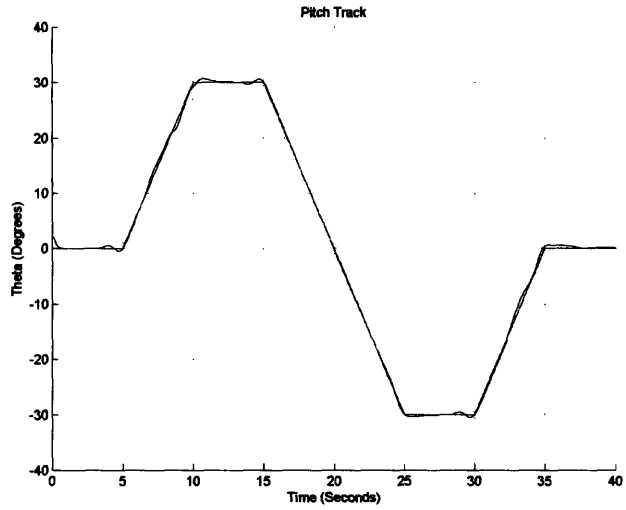


**Figure 23: Pitch Doublet**

The input profile q can also be seen in Figure 24.

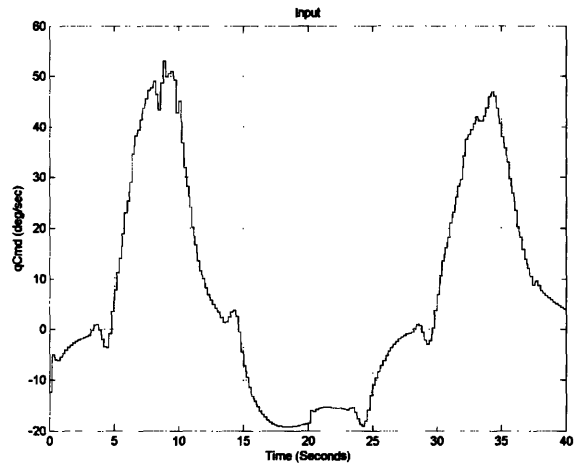


**Figure 24: Pitch Doublet Input**

As mentioned, the first 4 Laguerre polynomials are used as basis functions. The time horizon is four seconds and the controller rate is 5 Hz.

## 4.6.2 Bank Hold

The next autopilot mode is bank hold. This will allow the aircraft to be commanded to any bank angle during flight. The input to the CAS now becomes p or roll rate and the angle phi is tracked. Table 5 shows the specifications for this autopilot mode.

**Table 5: Bank Hold Controller Specifications**

| Tracking Variable | MPC Output | CAS Output | Controller Rate |
|---|---|---|---|
| φ | p | Aileron position | 5 Hz |

As with the pitch hold controller, the first half of the time horizon must be weighted to prevent premature oscillations in the controller. The following is an example of this autopilot mode. The roll maneuver is an aggressive doublet in phi between 50 and -30 degrees. This can be seen in Figure 25

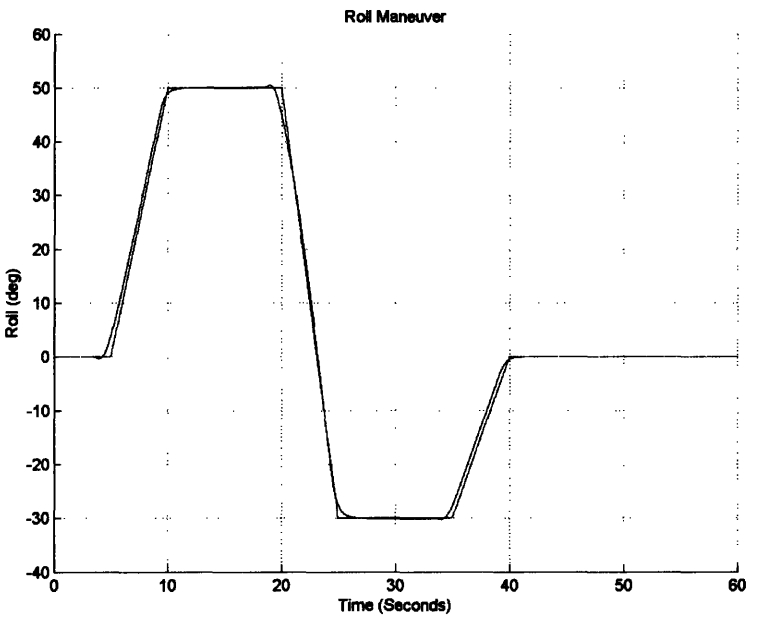


**Figure 25: Bank Hold**

Due to the aggressive nature of this maneuver, the MPC controller hits its constraints on the maximum allowable roll rate. The controller was limited to sustain a maximum rate of +/- 100 deg/sec. In Figure 26, we can see how the lower constraint becomes active around 23 seconds as the controller tries to transition from the 80 deg/sec ramp to a -30 deg bank angle.

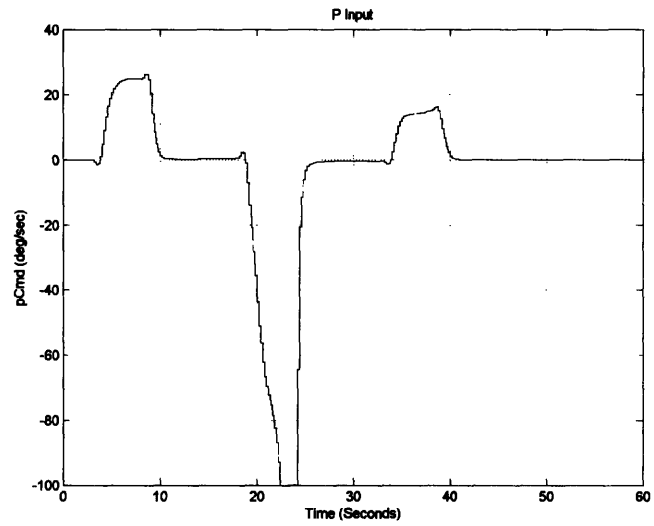


**Figure 26: P Input**

The same maneuver with input constraints tightened to +/-25 degrees can be seen in Figures 27 and 28. The constraint proves to be too tight to accurately track the abrupt change from 50 to -30 degrees. This example shows how the optimal solution of the MPC controller is bounded by the pre-selected constraints.

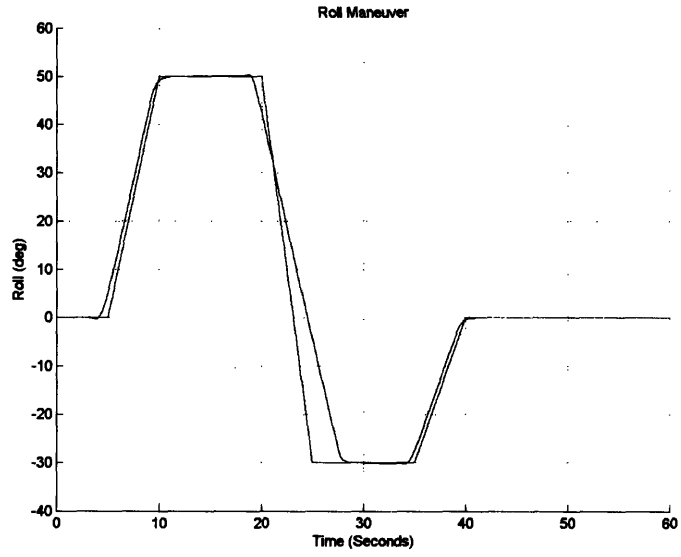


**Figure 27: Bank Hold Response**

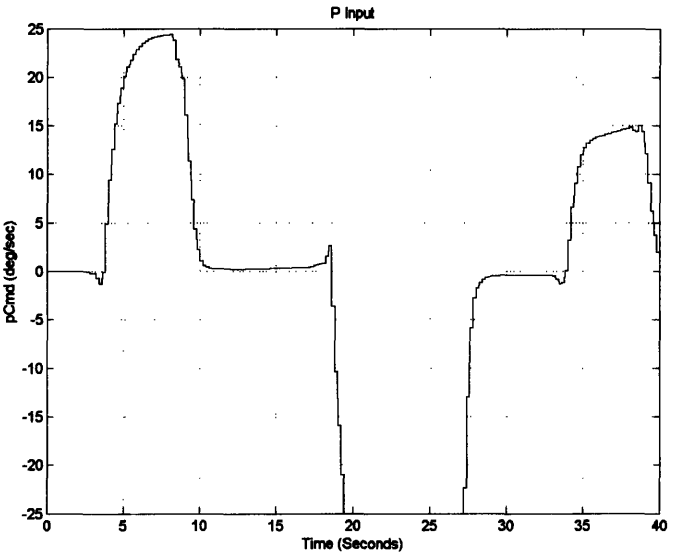


**Figure 28: Bank Hold Input**

This aggressive banking causes the aircraft to veer in two directions. This can be seen in the ground track of the craft in Figure 29. As expected, the craft first flies abruptly to the right and then tracks left.

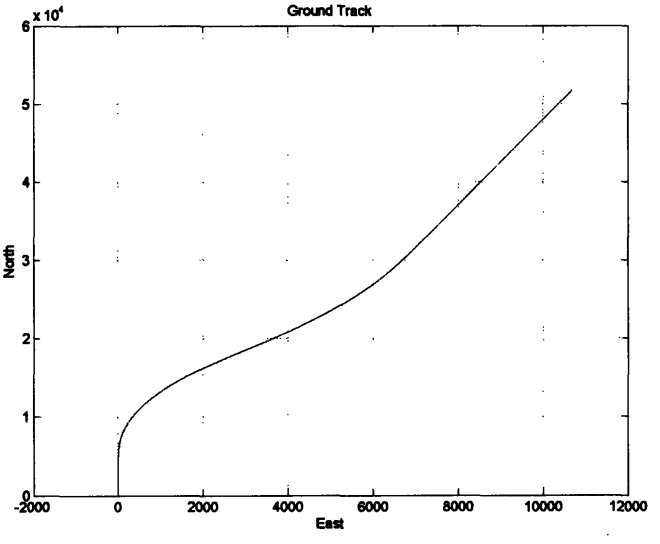


**Figure 29: Ground Track**

## 4.6.3 Altitude Capture

The altitude capture mode autopilot is perhaps the most practical of all. As the title suggests, this autopilot will fly the aircraft to any given altitude and hold it at that altitude. The MPC controller sends the pitch rate q as the command signal to the inner loop CAS. Table 6 summarizes the specifications for this autopilot mode.

**Table 6: Altitude Capture Specifications**

| Tracking Variable | MPC Output | CAS Output | Controller Rate |
|---|---|---|---|
| h (altitude) | q | Horizontal tail surface position | 5 Hz |

The same weighting applies for this controller, with the usual 4 second horizon and 5 Hz controller rate. The following is a demonstration of the autopilot's ability. Figure 30 shows an aggressive altitude maneuver doublet beginning at 25,000 feet. There is a steep 1000 foot climb at a rate of 100 ft/sec as the maximal height is approached. The accompanying control profile in q can also be seen in Figure 31.

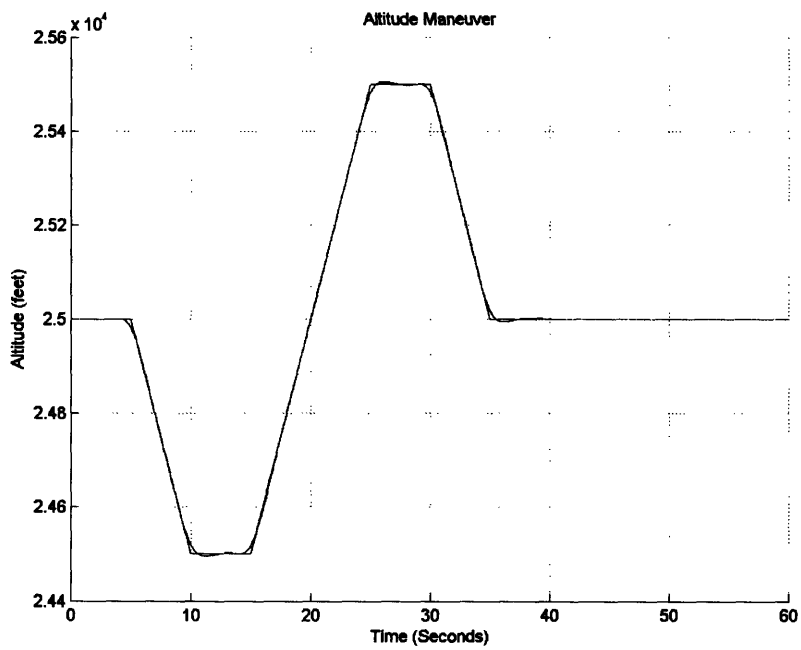


**Figure 30: Altitude Capture**

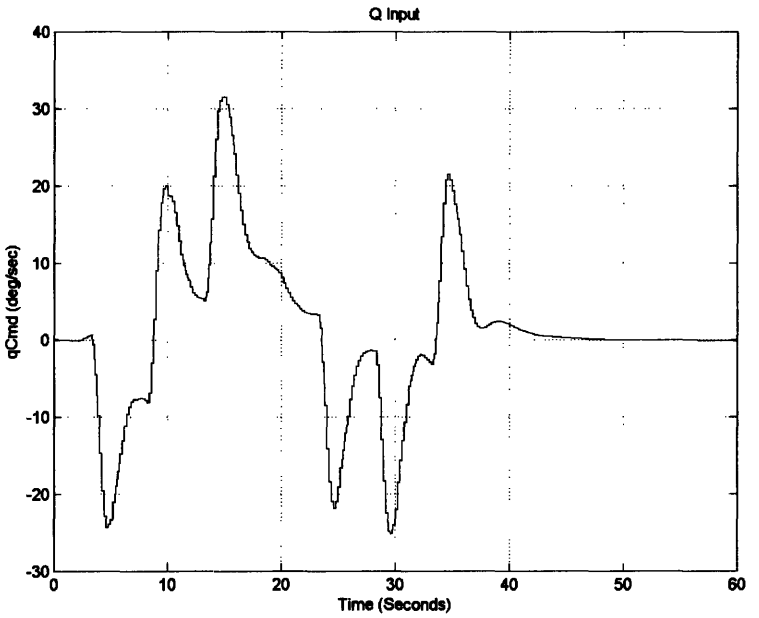


**Figure 31: Q Input**

In addition to input and output constraints, states within the model can be constrained as well. For example, in the altitude capture controller the altitude rate was constrained to 80 ft/sec. The response of the system to a step input from 25,000 to 25,500 feet is shown in Figure 32. Figure 33 shows the input to the CAS, pitch rate, while Figure 34 shows the altitude rate state, limited to 80 ft/sec.

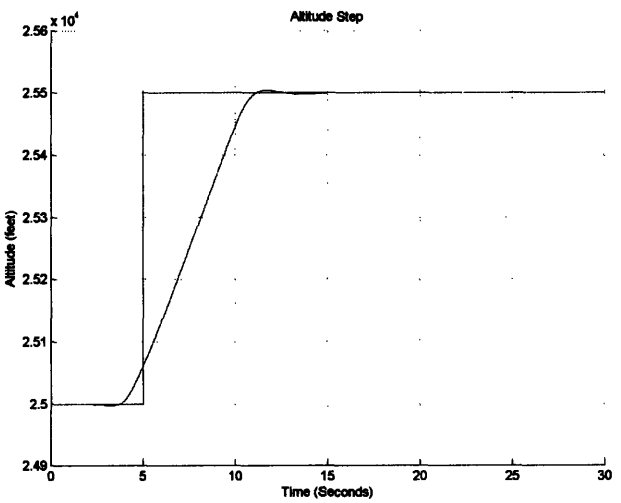


**Figure 32: Altitude Step with Rate Constraint**

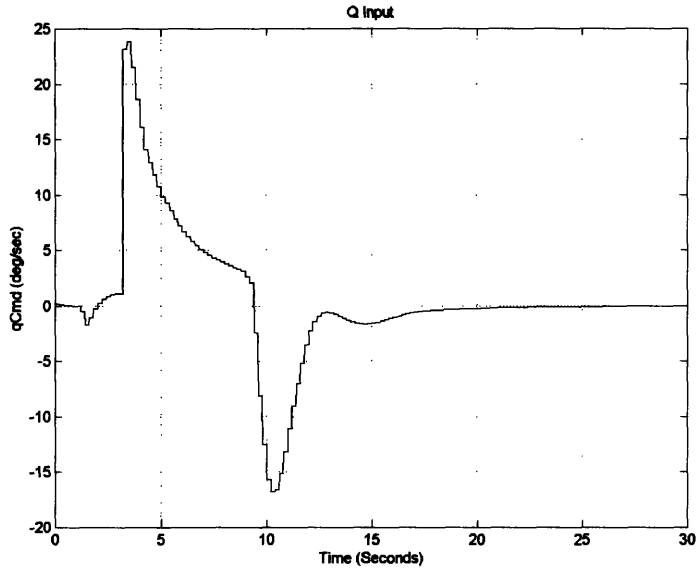


**Figure 33: Pitch Rate Command for Altitude Step with Rate Constraint**

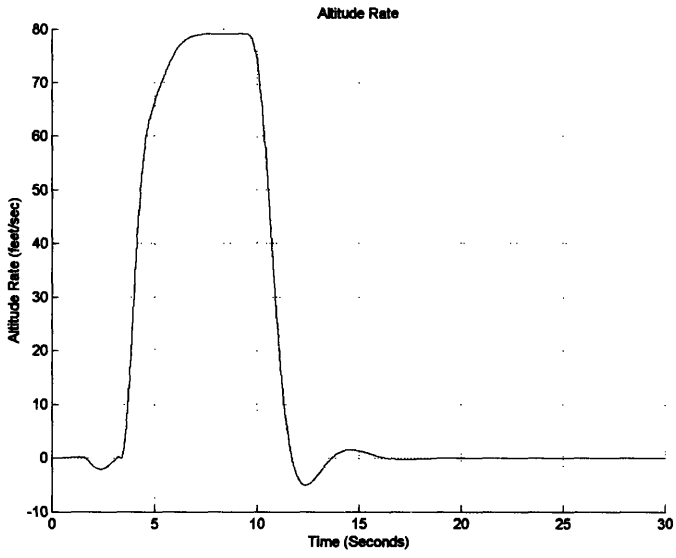


**Figure 34: Altitude Rate for Altitude Step with Rate Contraint**

## 4.6.4 Altitude and Heading Hold

The final autopilot mode is an altitude capture and heading hold autopilot. The autopilot has two variables to be tracked, altitude and heading, and the inner loop CAS is now actively controlling the thrust vectoring system. The inputs are pitch rate q, yaw rate r,

thrust vane angle in the x-z plane, and thrust vane angle in the x-y plane. Table 7 summarizes the specifications for this autopilot mode.

**Table 7: Altitude and Heading Hold Specifications**

| Tracking Variable | MPC Output | CAS Output | Controller Rate |
|---|---|---|---|
| h (altitude) | q | Horizontal tail surface position | 5 Hz |
| $\psi$ | r | Rudder Position | 5 Hz |
| | | TVC x-z vane angle | |
| | | TVC x-y vane angle | |

In addition, the MPC outputs are bounded as well as the roll angle ($\varphi$). It will be shown that the aircraft must implement a coordinated turn to maintain psi without sideslip and maintain altitude. This means that the aircraft will bank as it makes its heading changes while maintaining altitude. An example of this autopilot response can be seen below. It shows the aircraft make two climbs of 400 feet each with a heading change of 15 degrees between them. Figure 35 shows the altitude profile with reference trajectory.

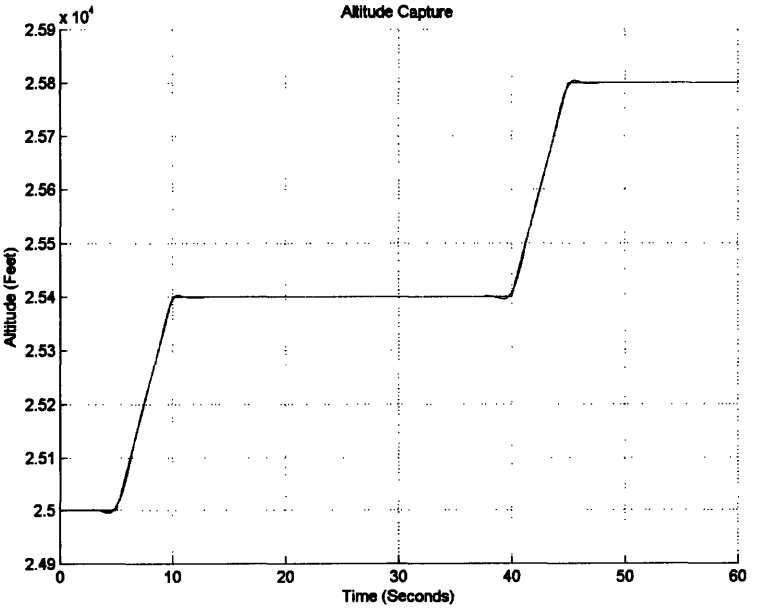


**Figure 35: Altitude Capture**

Figures 36 and 37 show how the aircraft makes a coordinated turn beginning at 10 seconds into the flight. These turns are often made quite slowly to keep the aircraft stable, hence the low yaw rate.

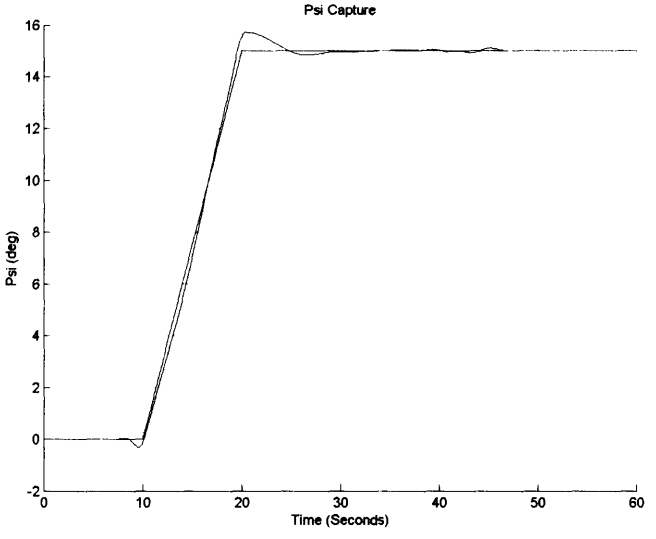


**Figure 36: Heading Capture**

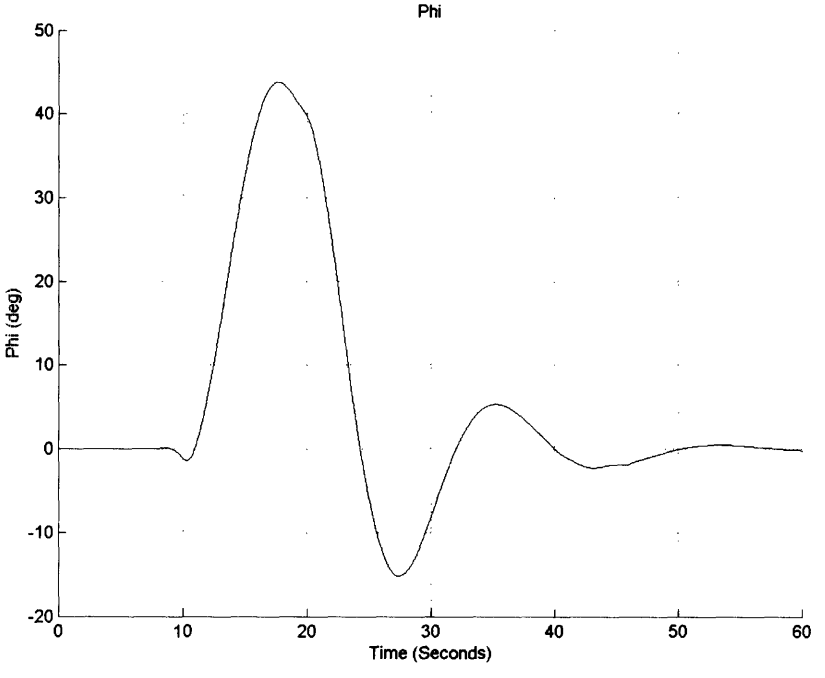


**Figure 37: Phi**

The following are the inputs into the CAS and thrust vectoring systems. The two CAS inputs are shown below in Figures 38 and 39. The q input rides both active constraints of +/- 20 degrees.

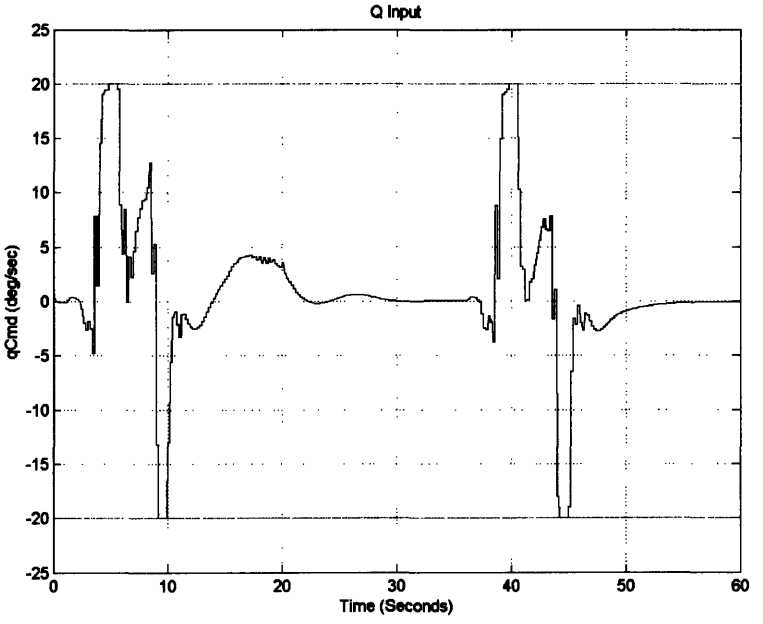


**Figure 38: Q Input**

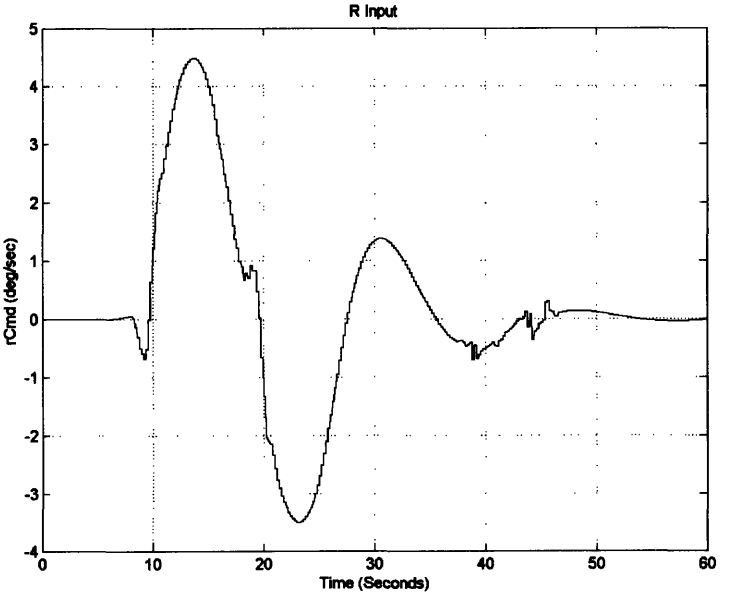


**Figure 39: R Input**

The following are the inputs commanded by the CAS to the thrust vectoring system during the maneuver.

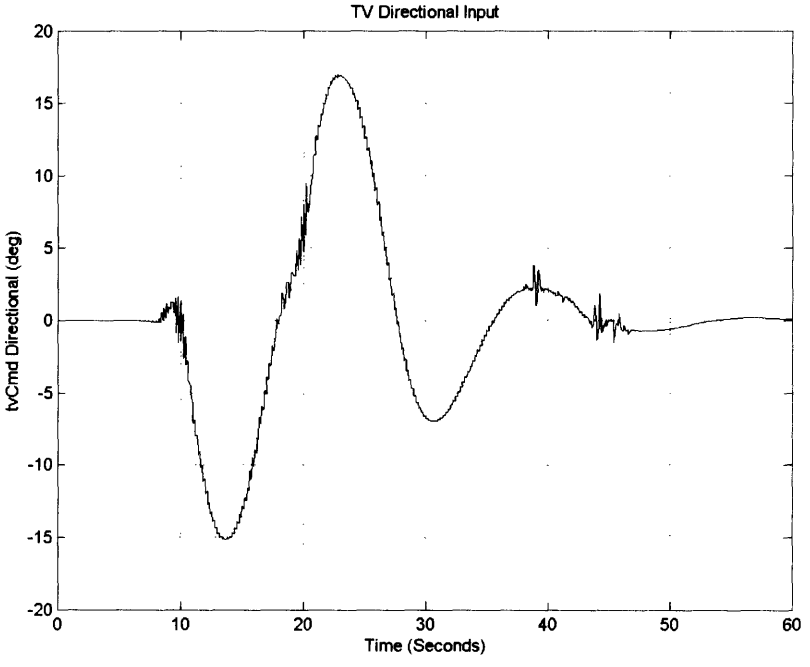


**Figure 40: TV Directional Command**

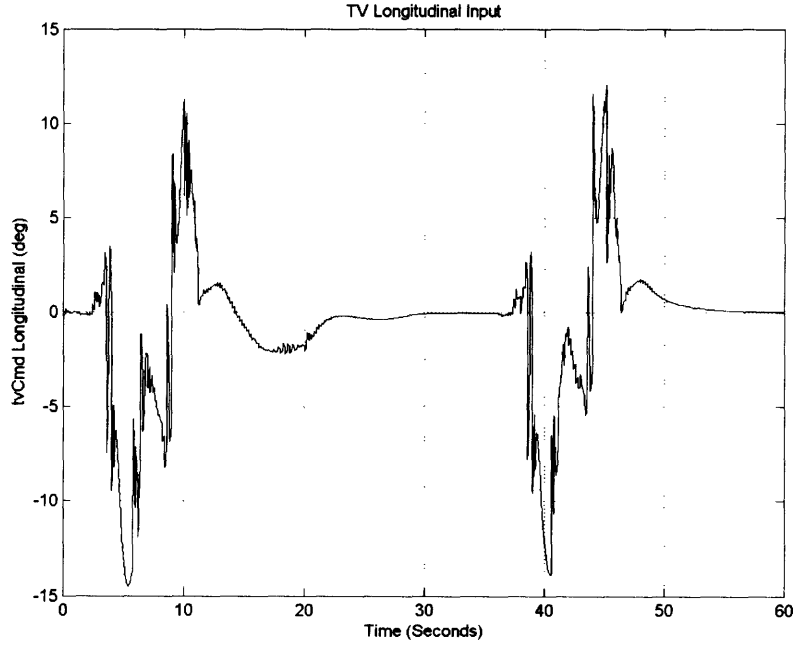


**Figure 41: TV Longitudinal Command**

Finally the entire trajectory the aircraft traces out can be seen in Figure 42.

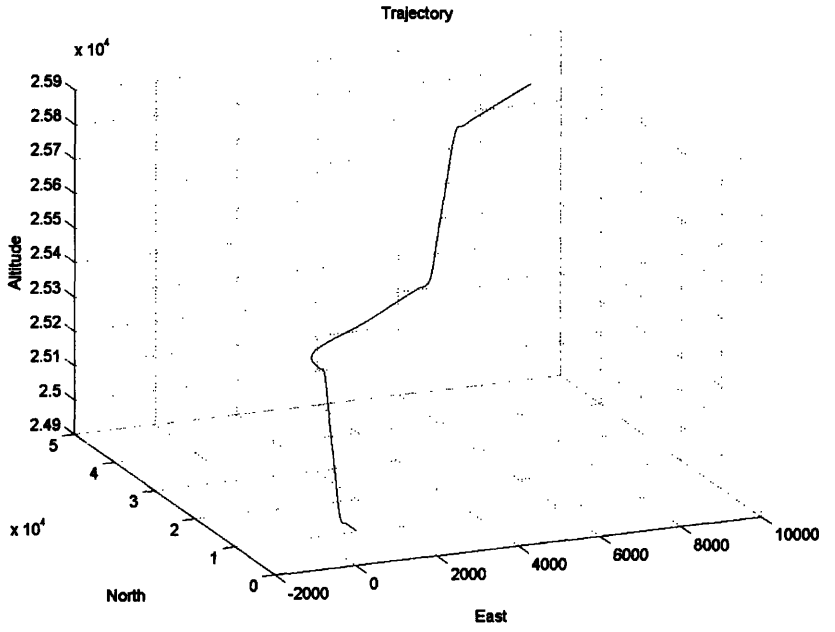


**Figure 42: Trajectory**

Again, for this autopilot mode 4 Laguerre basis functions were used with a 4 second prediction horizon and 5 Hz controller rate.

# Chapter 5

# Failure Scenario Design Methods

## 5.1 Redundant Control Authority

In order for the autopilot to reconfigure its control scheme in the event of a failure, there must be some degree of redundant control authority. For instance, if while engaged in a steady climb the entire horizontal tail fails in one direction, there is no other control effector to be used to recover from this type of failure. For this reason each of the rudder, aileron and horizontal tail surface pairs were mathematically divided into two. This was accomplished in simulink by splitting the actuator signal and then recombining it again before sending it to the aerodynamic and engine models.

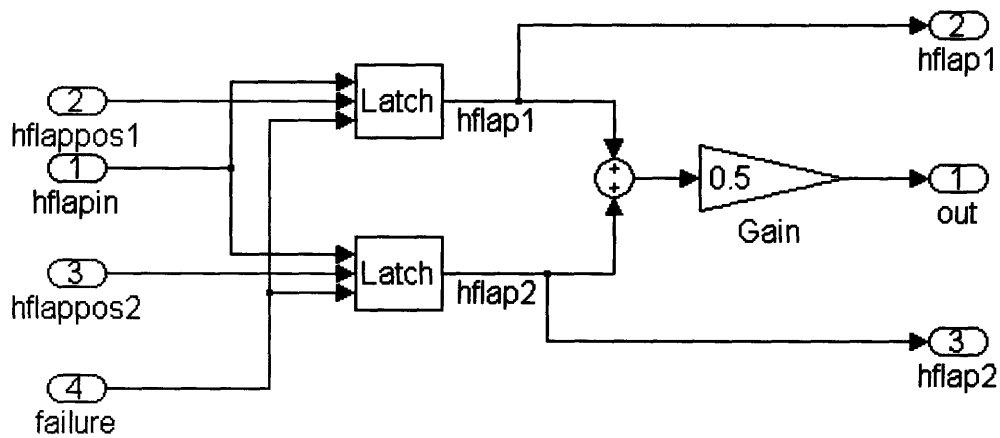


**Figure 43: Control Surface Split**

In Figure 43, the signal hflapin is split into two different signals, hflap1 and hflap2. For all intentional purposes, these two signals become two separate control surfaces that can be failed independently. The figure shows also that they are recombined and multiplied

by a gain of 0.5 before being sent to the aerodynamic model. In addition to the added control surfaces, the thrust vectoring system is also engaged. The deflection of the thrust creates a large moment. It will be shown that this large moment is very effective in reducing the transient seen once a failure has occurred.

## 5.2 Failure Types

This research explores a special type of failure in the aircraft. This is when the control surfaces fail to track positions commanded by the CAS. We are going to assume that these failures occur within a symmetric bound. For instance, if the vehicle is in a pitch hold maneuver then ,for example, both horizontal tail surfaces must fail in the same way in order to create an almost symmetric failure in the longitudinal plane. Alternatively both outboard horizontal tail surfaces might fail in the same way. Completely asymmetric failures can cause a lateral motion that is uncontrollable by the lateral-directional channel.

There are three types of failures explored in this research. The first is the hard over failure in which one pair of surfaces proceeds to its maximal positive limit. The moment this is triggered is determined by the operator and has a time constant of 0.25 seconds. An example of this can be seen below in Figure 44.

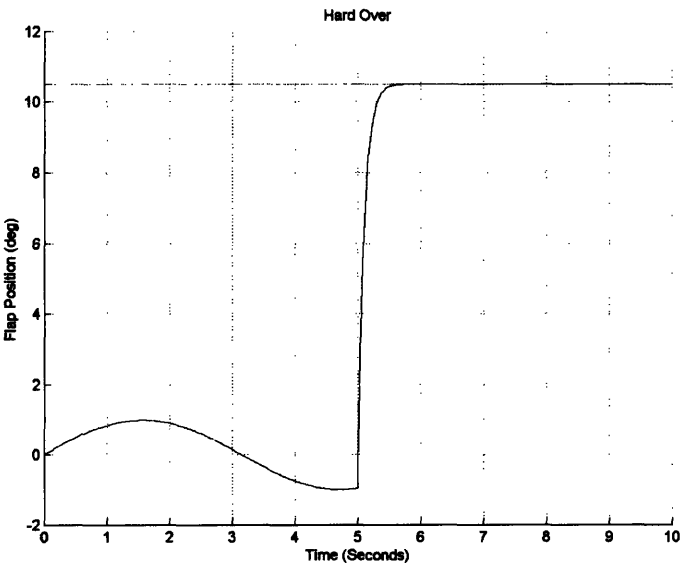


**Figure 44: Hard Over Demonstration**

In this figure a sine wave with amplitude 1 and frequency of 1 rad/sec is the commanded control surface position. At 5 seconds a hard over failure occurs driving the position to its maximal limit of 10.5 degrees as in the case of the horizontal tail surfaces. This limit is approached with a time constant of 0.1 seconds.

The second type of failure is the frozen or stuck surface, seen below in Figure 45. As would be expected the surface simply stays locked in its position at the point of failure. Figure 46 is a demonstration of the final failure type, the hard under failure. In this scenario, the surface proceeds to its maximal negative deflection of -24 degrees with a time constant of 0.1 seconds.

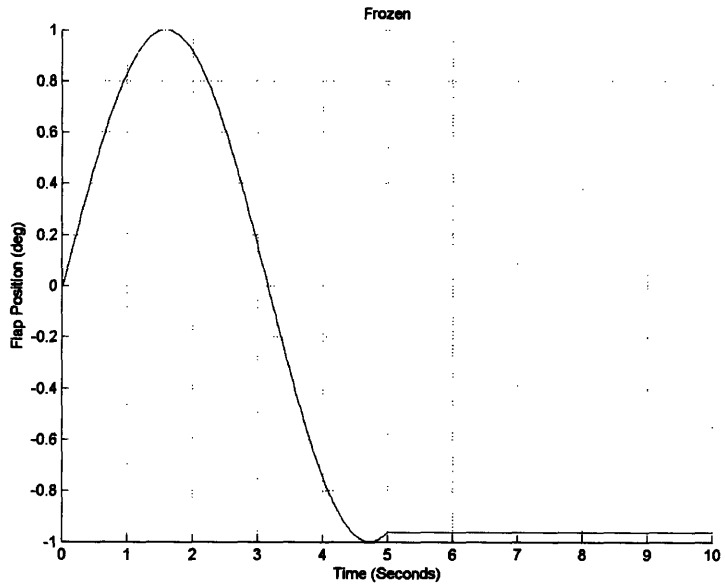


**Figure 45: Frozen Failure Demonstration**

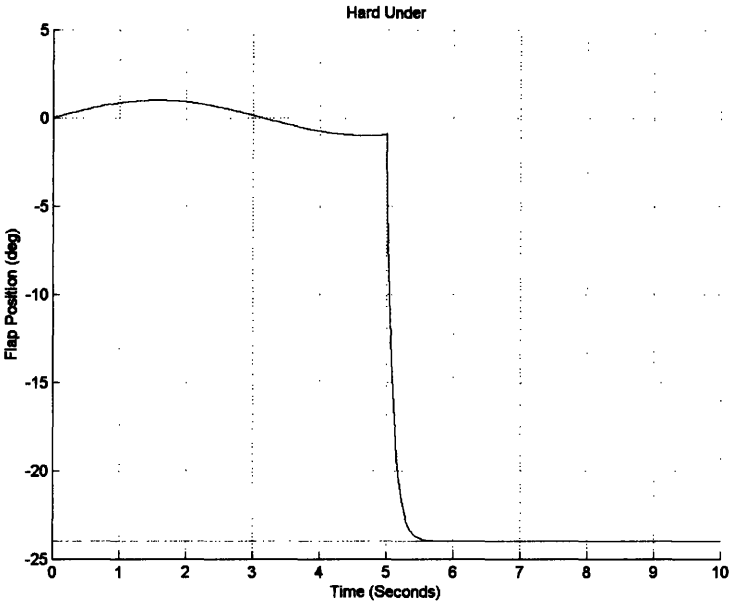


**Figure 46: Hard Under Demonstration**

## 5.3 Failure Simulation and Update

The key enabling technology that allows these autopilots to reconfigure their control strategy is their ability to update the internal model in real time. In the case of control surface failures, this means that the model must have an input for the actual surface position when a failure occurs. The scope of this research does not involve the actual failure detection and isolation (FDI) method. This type of FDI system is already assumed to be installed, and knowledge of the failure is assumed to be given to the controller.

In order to trigger a failure in the truth model a failure signal engages one of the three failure types in a symmetric set of surfaces. This can be seen below for one surface in Figure 47.

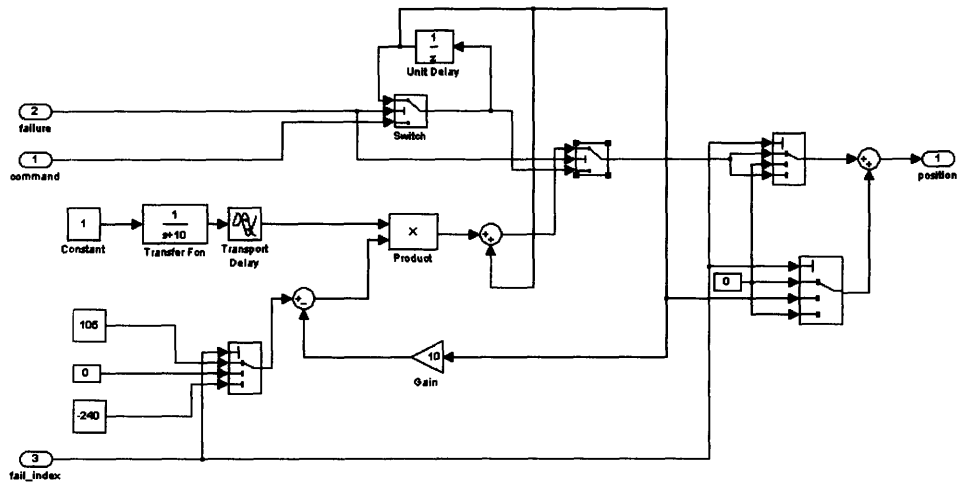


**Figure 47: Failure Implementation**

In this implementation the signal 2 failure triggers the system to stop using the original commands sent by the CAS signal 1. The type of failure initiated by the system is dependent on the fail_index signal. In this implementation a fail_index of 1 is hard over, 2 is frozen, and 3 is hard under.

The method for internal plant update consists of a switch in the actuator models of the plant. Knowledge of a failure in the MPC controller serves only to ignore the actuator models in favor of the actual actuator feedback positions from the truth model. Thus, during a failure the model begins making predictions with valid information about the disabled control surface positions.

**[This Page Intentionally Left Blank]**

# Chapter 6

# Simulation Results

In this chapter, we are going to show how the four MPC autopilot modes respond to simulated control surface failures occurring at various times over various prescribed maneuvers. In order to stress the response of the system, failures were chosen to occur during a transient, just before or after a change of reference. The aircraft was then allowed to fly the rest of the trajectory with the failed surface. For each controller three failure cases were considered: hard over, frozen and hard under. These were then plotted against the nominal non-failed response. In addition, the MPC controller was tested with its internal model not updated during a failure.

## 6.1 Pitch Hold Autopilot Mode Reconfiguration

In order to test the pitch hold autopilot mode, a maneuver from the nominal pitch angle of 2.11 degrees up to 30 degrees and back was selected. A failure in the two inboard horizontal tail surfaces was then injected after 14 seconds. The controller update is at 5 Hz, with a 4 second prediction horizon and 4 Laguerre bases.

Pitch Test Case 1

In this failure scenario, the two inboard horizontal tail surfaces proceed hard over to 10.5 degrees at 14 seconds. In Figure 48 a small transient can be seen when the failure occurs and the MPC controller, along with its adapted model, proceed to track the rest of the reference profile.

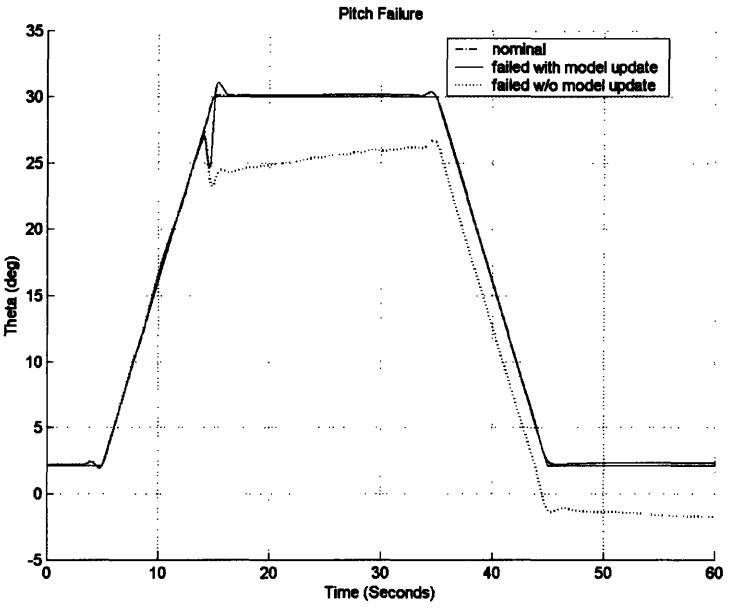


**Figure 48: Pitch Failure**

However, it can be seen that when the MPC model is not updated there is a larger transient and substantial steady state error. This is due to the fact that the model is now incorrect and the MPC internal propagator does not follow the real behavior of the failed aircraft. A close up view of this discrepancy can be seen in Figure 49.

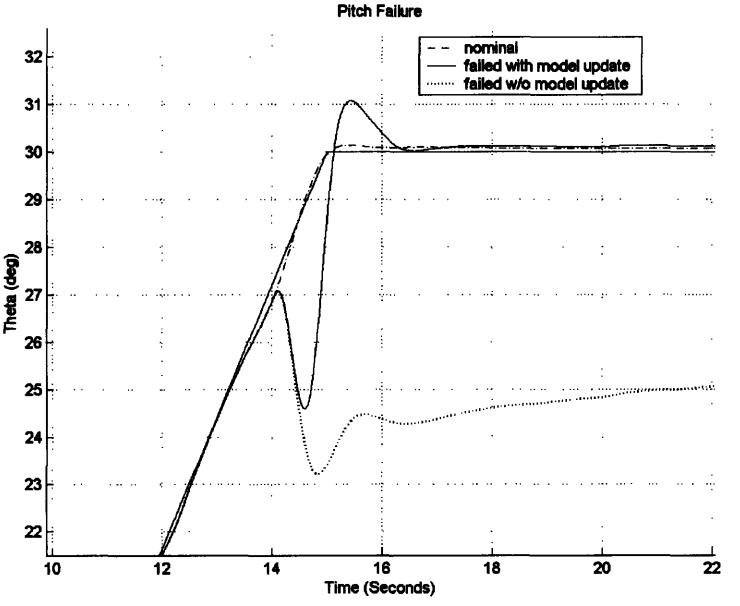


**Figure 49: Pitch Failure Zoom**

This shows the ability of the controller to reconfigure its own control law as a result of an updated internal model. The MPC controller responds to the failure by commanding aggressive inputs to correct the large error induced by the failed control surfaces. This change in control strategy can be seen against the nominal one with no failures in Figure 50. Also shown is the control strategy when the internal model is not updated. Figure 51 shows a close up view of the input at the time of the failure.

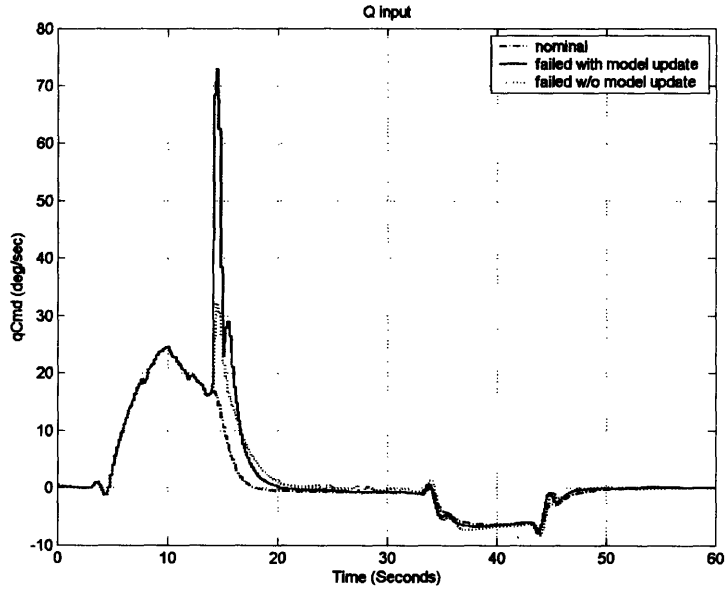


**Figure 50: Pitch Controller Reconfiguration**

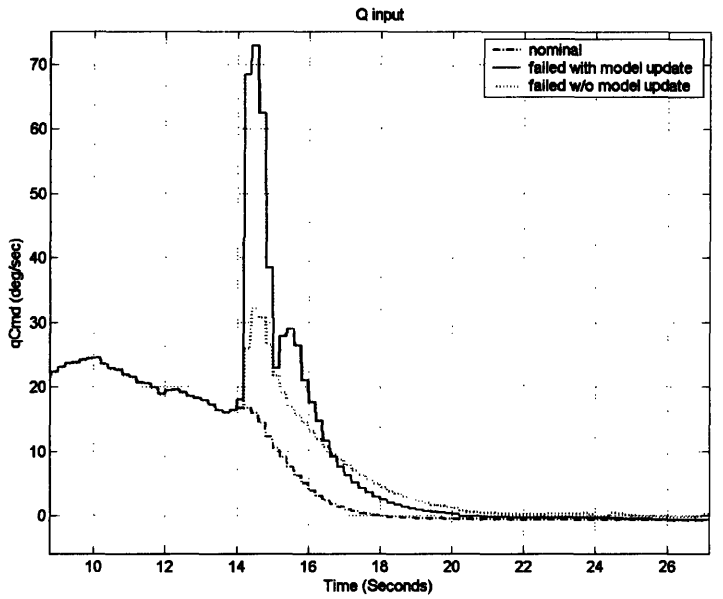


**Figure 51: Pitch Controller Reconfiguration Zoom**

It is important to point out that the failure reconfiguration is confined to the MPC autopilot controller only. The inner loop CAS remains unchanged in the case of a failure; it simply continues to track the commanded attitude rates. This is in fact where the MPC reconfigurable control strategy differs from others, where in a moment allocation scheme is used. In moment allocation strategies, a constrained linear programming problem is set up to solve for new control surface positions which will induce the desired moments. Thus the solution is the appropriate surface positions that maintain the proper moment. In the MPC reconfigurable control case, the moment allocation problem is always solved by the internal CAS system. In fact, MPC is aware of the CAS capabilities because of its own internal model. The MPC controller inputs have been optimized to take into consideration the new CAS tracking abilities.

Figure 52 show the response of the remaining control srufaces to the failure. After responding to the new commands issued by the MPC, the CAS reaches a new steady state value to trim the aircraft.

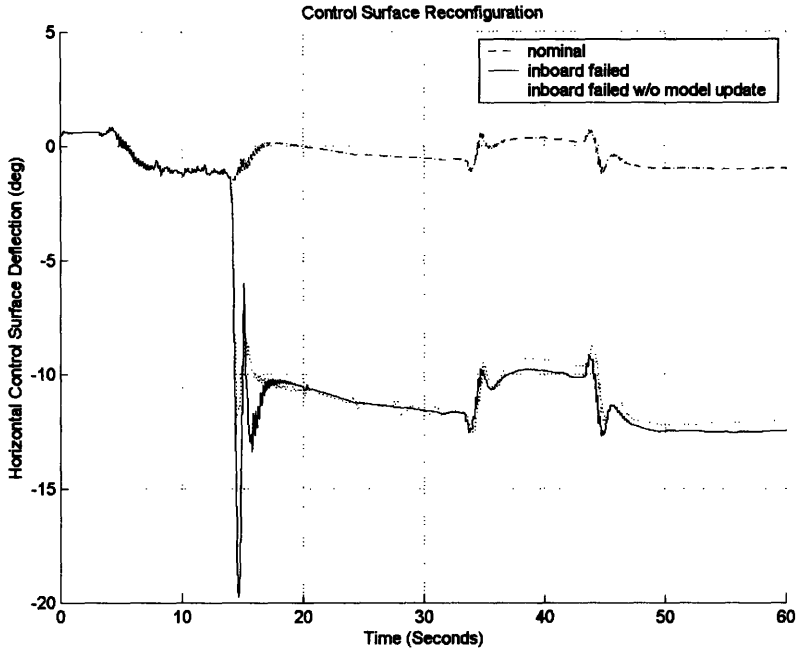


**Figure 52: Control Surface Reconfiguration**

Pitch Test Case 2

The following scenario has the same pitch profile as before with a failure at 14 seconds, but now a hard under excursion of the inboard horizontal tail surfaces to -10 degrees is injected. The resulting trajectory can be seen in Figures 53 and 54.

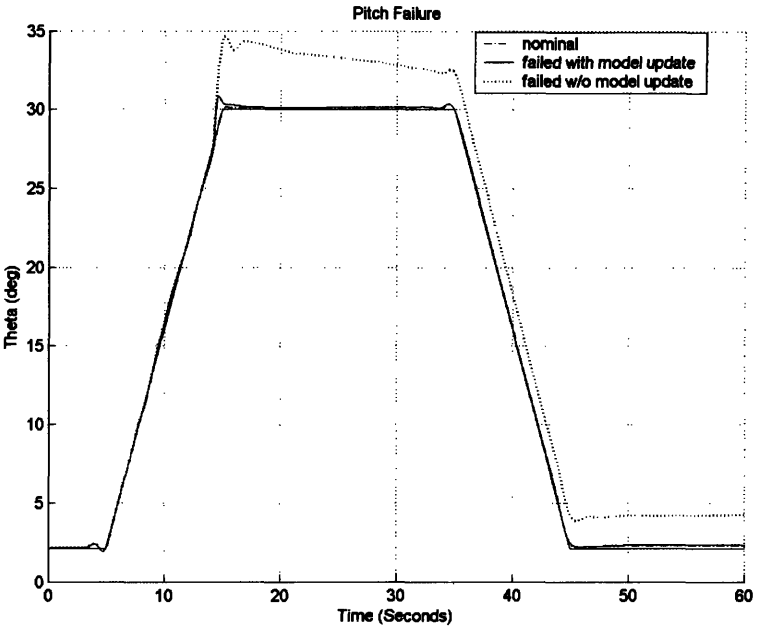


**Figure 53: Pitch Hard Under Reconfiguration**

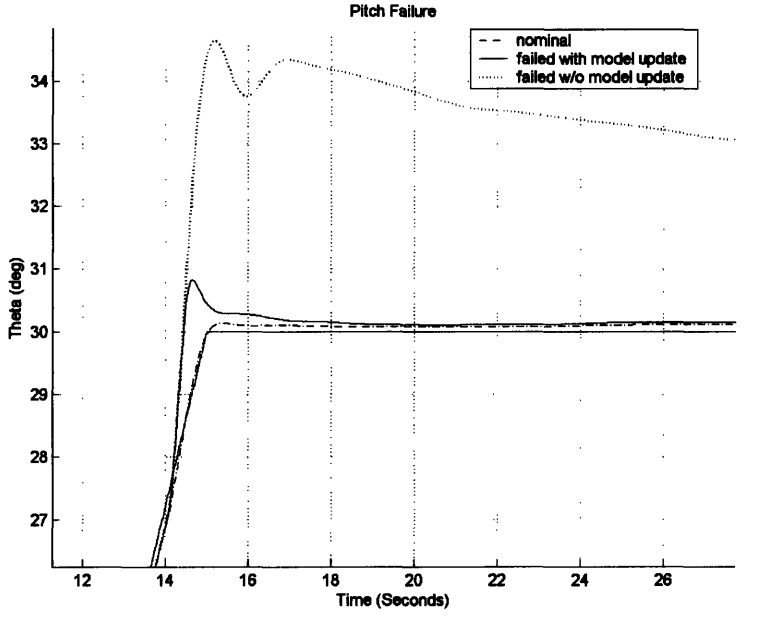


**Figure 54: Pitch Hard Under Reconfiguration Zoom**

Again, we can see the typical steady state error encountered by the non updated model. The resulting control profile can also be seen in Figure 55 and the reconfigured horizontal tail surfaces positions in Figure 56.

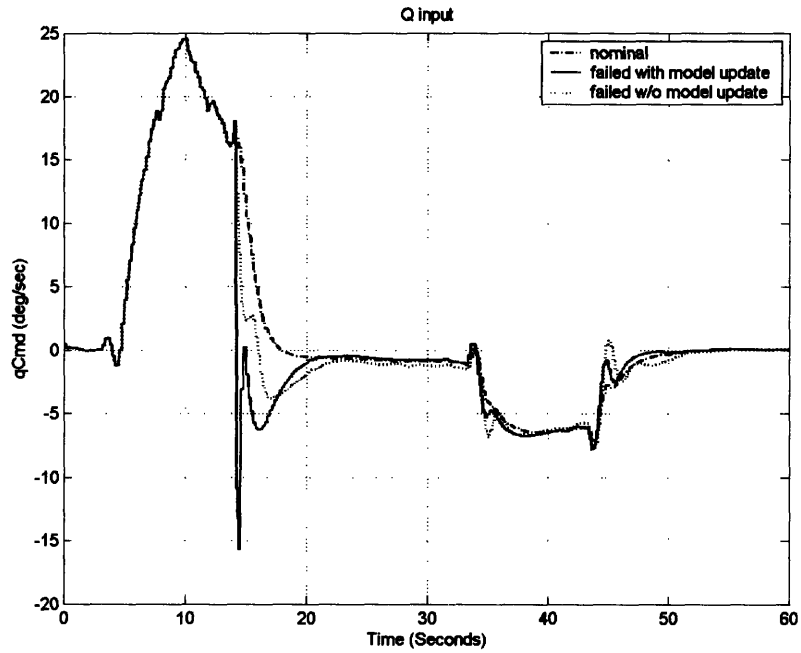


**Figure 55: Pitch Hard Under Controller Reconfiguration**

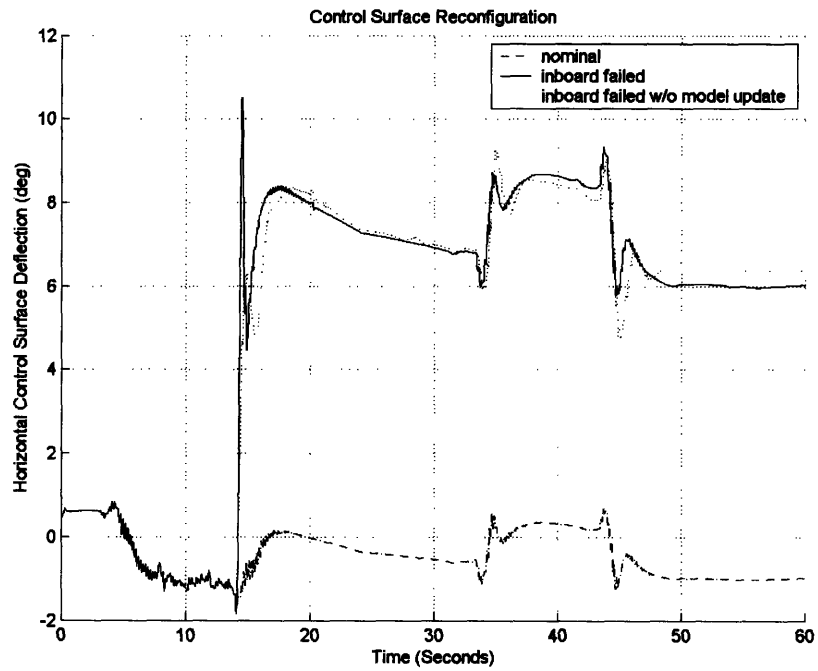


**Figure 56: Pitch Hard Under Control Surface Reconfiguration**

Pitch Test Case 3

In the final pitch failure test, the two inboard horizontal tail surfaces were frozen at 14 seconds into the flight. Figures 57 and 58 show how the MPC controller closely tracks the pitch reference signal.

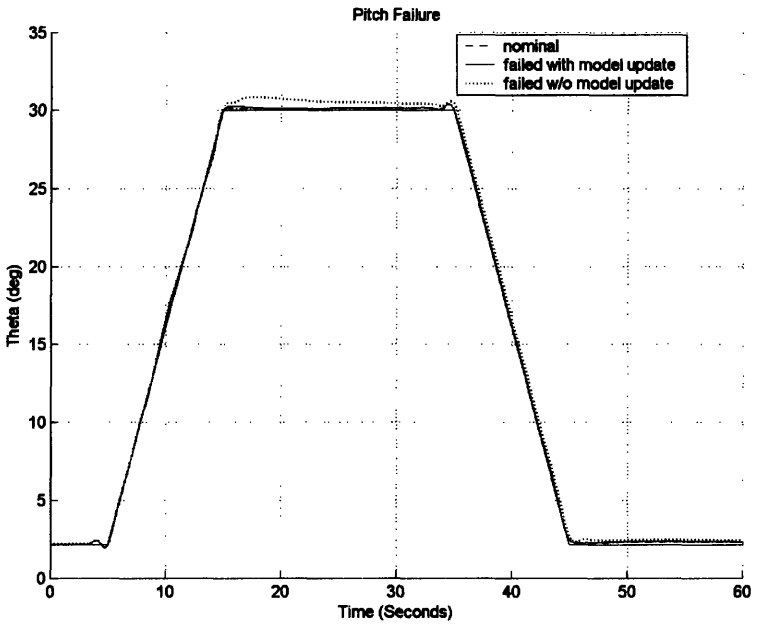


**Figure 57: Pitch Frozen Reconfiguration**

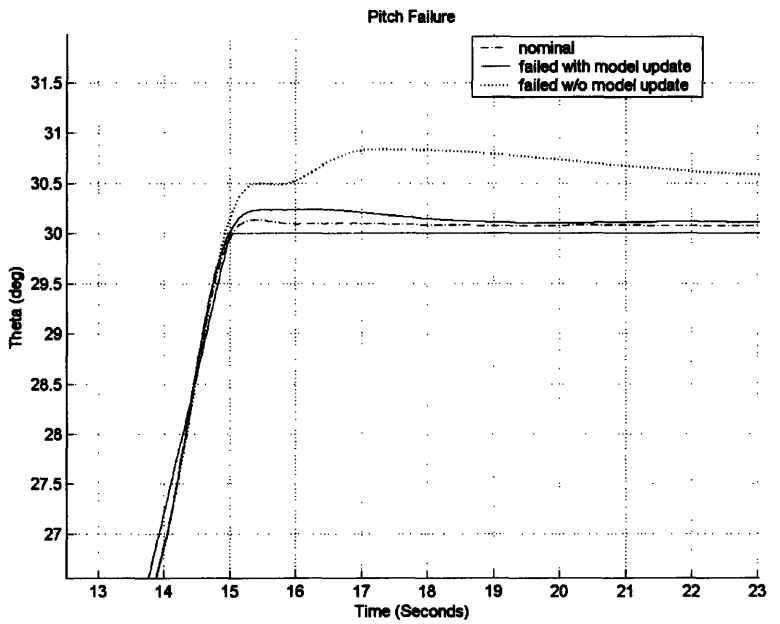


**Figure 58: Pitch Frozen Reconfiguration Zoom**

Both updated and unupdated MPC controllers adapt well to the failure with slight transients of no more than 1 degree. The ease to which the MPC controllers adapted can also be seen in the control profiles in Figure 59 and 60. In this example there is only a very slight discrepancy between the nominal and reconfigured control profiles.

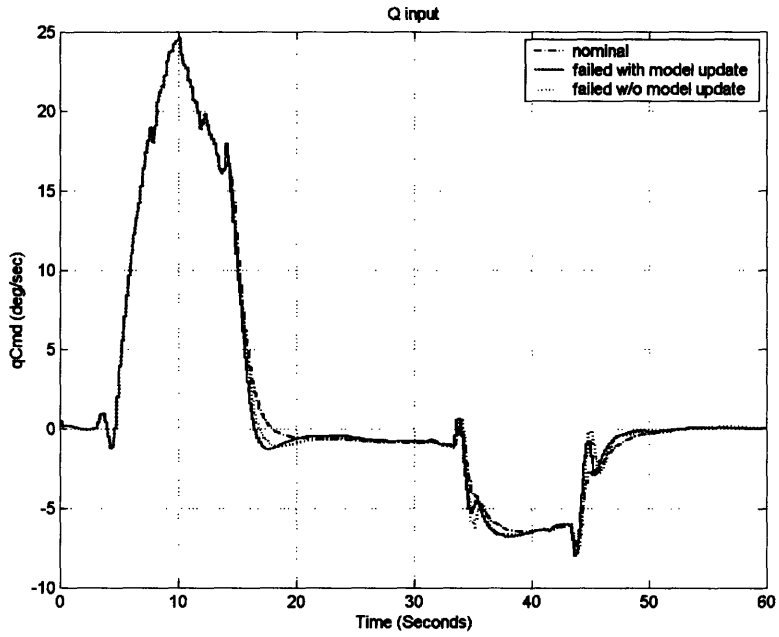


**Figure 59: Pitch Frozen Controller Reconfiguration**

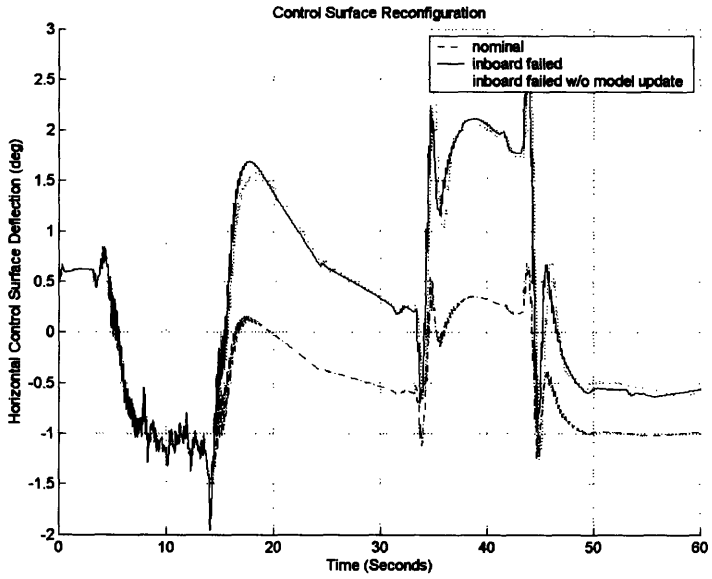


**Figure 60: Pitch Frozen Horizontal Tail Reconfiguration**

## 6.2 Bank Hold Autopilot Mode Reconfiguration

The bank hold autopilot mode was tested by failing half of an aileron at 10.4 seconds into the maneuver. The trajectory is a bank singlet from 0 degrees to 20 degrees and back. The MPC controller update rate had to be increased in order to maintain control of the vehicle throughout the failure. The moment of inertia about the roll axis of the aircraft is the smallest of all and hard deflections of even half of the aileron cause such a fast response that a controller running at 5 Hz just cannot compensate for it. For this case the controller update was increased to 25 Hz with the same 4 second prediction horizon and 4 Laguerre bases.

Roll Test Case 1

The first failure example has the outboard left aileron failing to 5 degrees at 10.4 seconds.

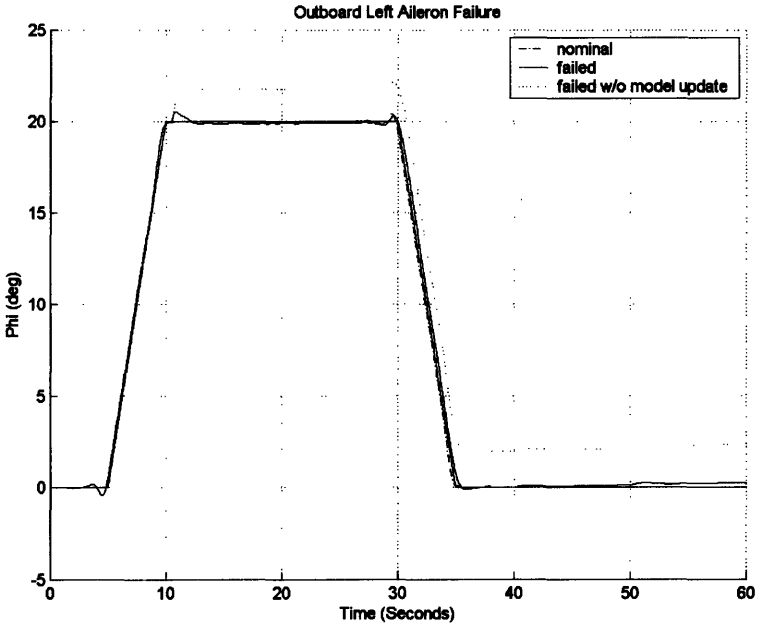


**Figure 61: Outboard Left Aileron Hardover Failure Reconfiguration**

The failure causes only a very slight transient for the MPC controller with model update and a 2 degree bias for the non updated controller as shown in Figure 61. This can be seen in a close up view in Figure 62.

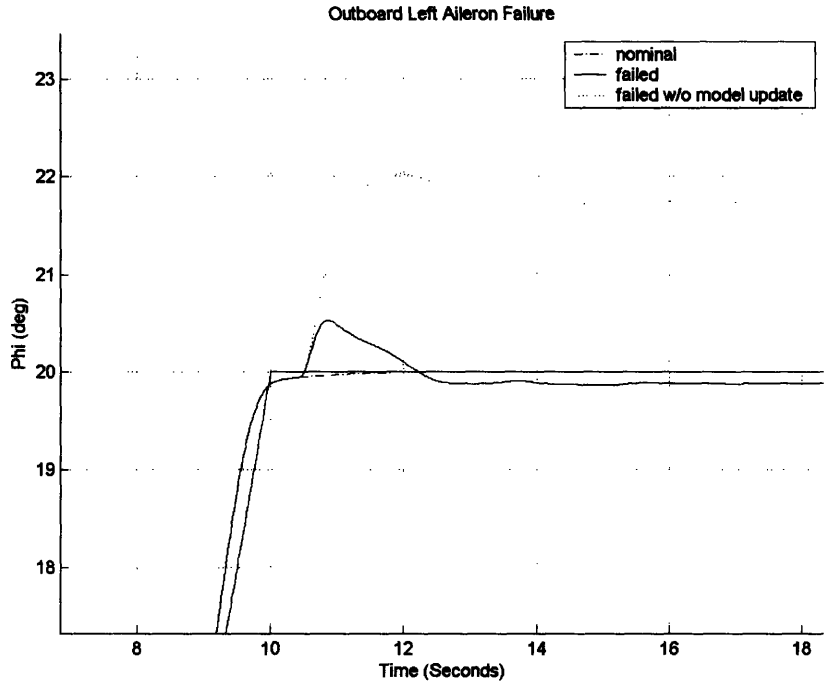


**Figure 62: Hardover Outboard Left Aileron Failure Reconfiguration Zoom**

These slight transients can be deceiving as to the amount of controller reconfiguration put forth to maintain this profile. The MPC control profile of commanded roll rate p, shows inputs almost up to the 80 deg/sec constraint. Moreover, the controller does not find a steady value to maintain wings level as it comes out of the maneuver. Thus the controller must command a constant roll rate input to counteract the moment generated by the failed surface. It must also be noted that the controller signal is only the amount of roll rate it must command for the existing CAS to maintain its trajectory with the failure. The actual roll rate of the vehicle goes to 0 to maintain constant bank angle. This large departure from the original control scheme can be seen clearly in Figure 63. The healthy inboard left aileron position and both right aileron positions can also be seen in Figures 64 and 65.

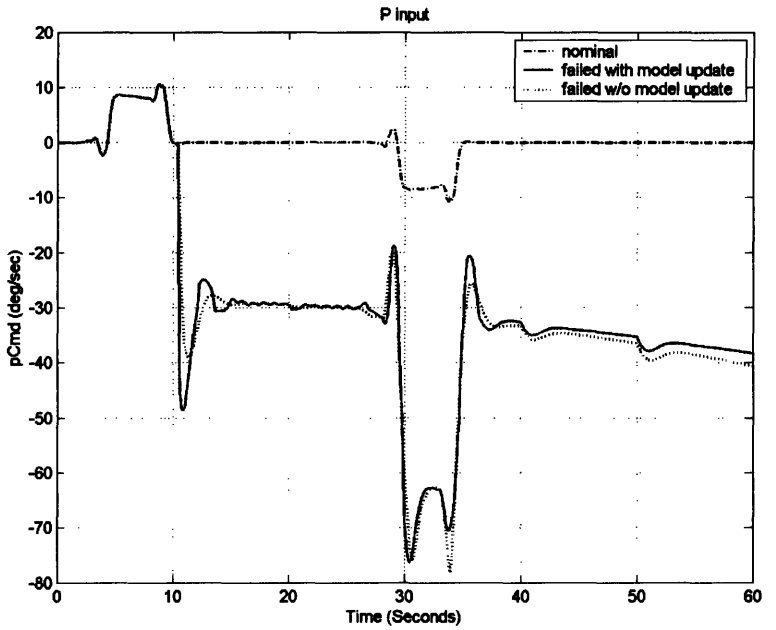


**Figure 63: Roll Rate Command for Hardover Outboard Left Aileron Controller Reconfiguration**

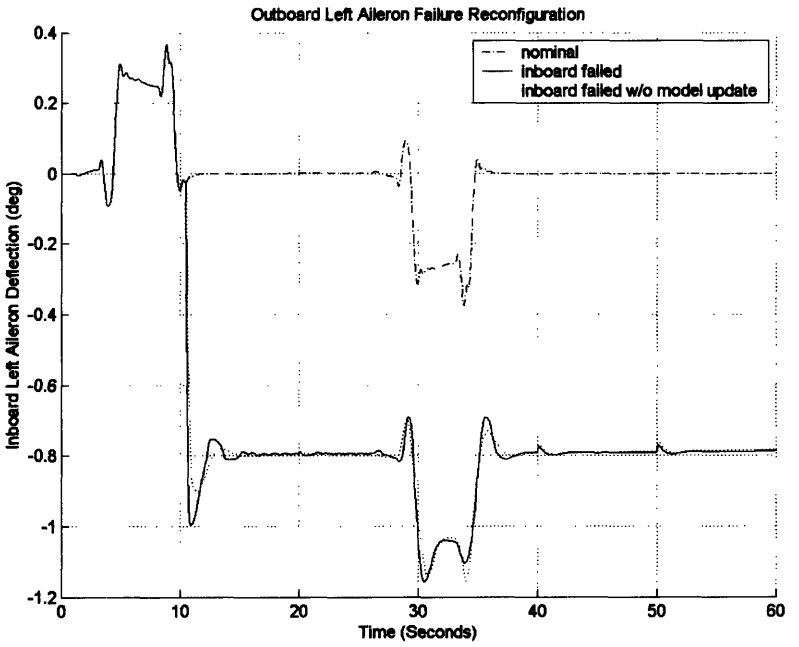


**Figure 64: Hardover Outboard Left Aileron Failure Reconfiguration**

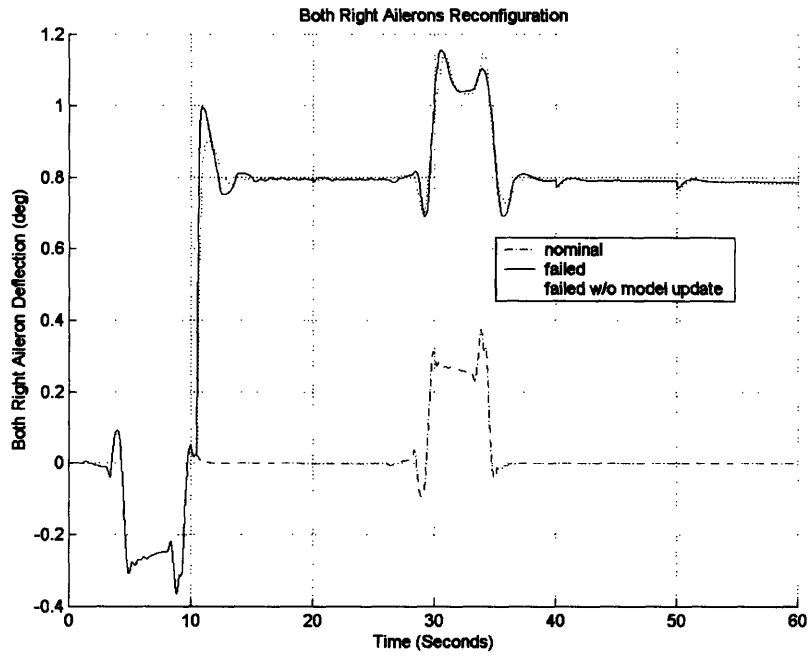


**Figure 65: Inboard and Outboard Right Aileron Failure Reconfiguration**

Due to the nature of the inner loop CAS, the left and right aileron sets can only deflect differentially. This means that the moment about the roll axis cannot be brought to zero using only the aileron system because all remaining ailerons can only proceed to either + or – a certain value. This explains why all remaining ailerons proceeded to either 0.8 or -0.8 degrees in this failure case. In order to compensate for this the MPC engine uses differential deflections of the horizontal tail surfaces as well. This creates the moment necessary to counter that caused by the 5 degree aileron deflection on the left side. The deflections of the horizontal tail surface can be seen in Figure 66. Only the deflections for a failure with internal model update are given for clarity.

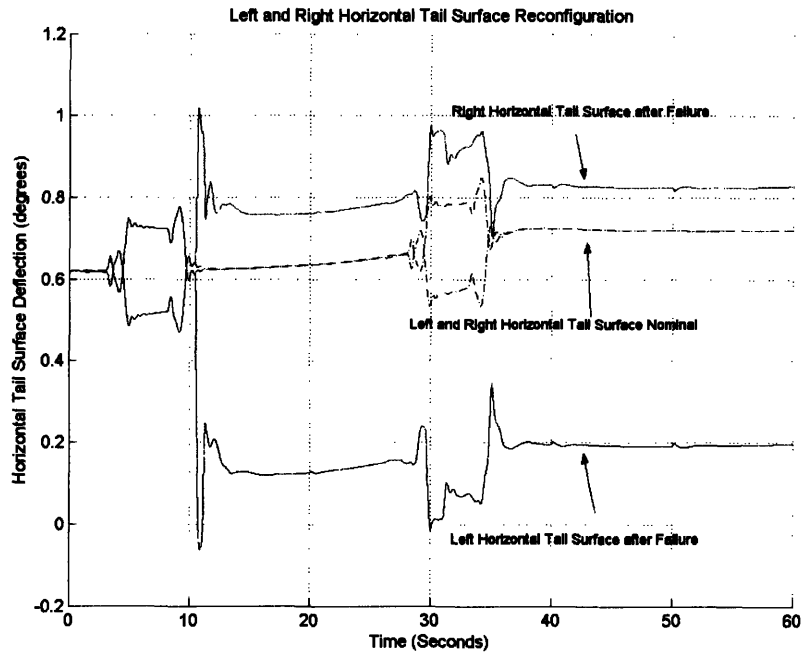


**Figure 66: Horizontal Tail Reconfiguration**

Roll Test Case 2

The next failure experiment was to fail the left outboard aileron to -5 degrees at the same time as in case 1 (10.4 seconds). The results are similar to the hard over experiment except for a slight oscillation of the MPC controller with the updated model, as shown in Figure 67.

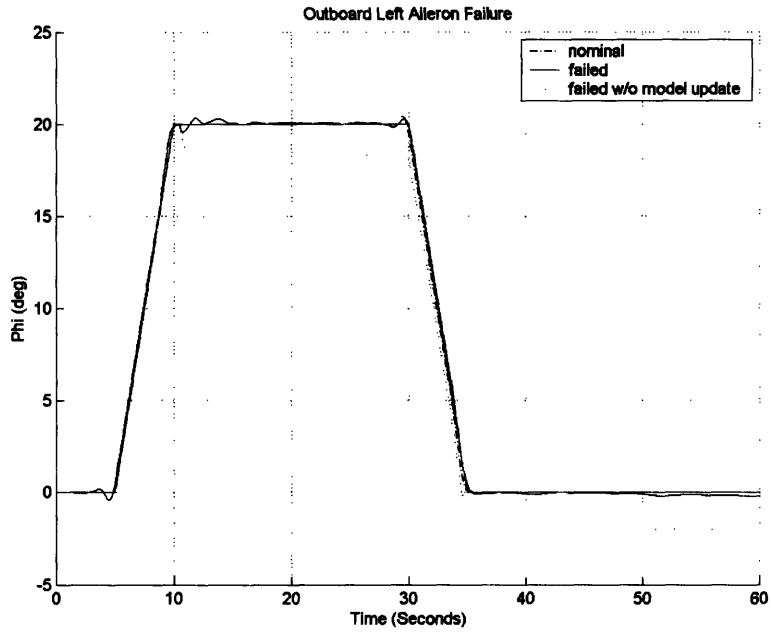


**Figure 67: Outboard Left Aileron Hardunder Failure Reconfiguration**

There also is a p input control profile similar to the hard over case (see Figure 68). The oscillations can be seen in the controller roll rate input. Also note the drifting control as the aircraft tries to maintain wings level.

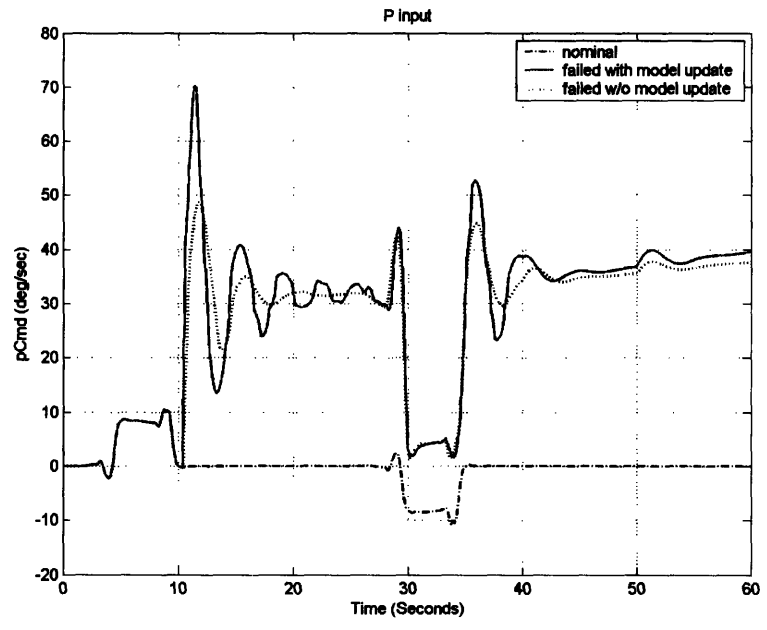


**Figure 68: Roll Rate Command for Hardunder Outboard Left Aileron Controller Reconfiguration**

The healthy inboard left and both right aileron deflections can be seen in Figures 69 and 70.

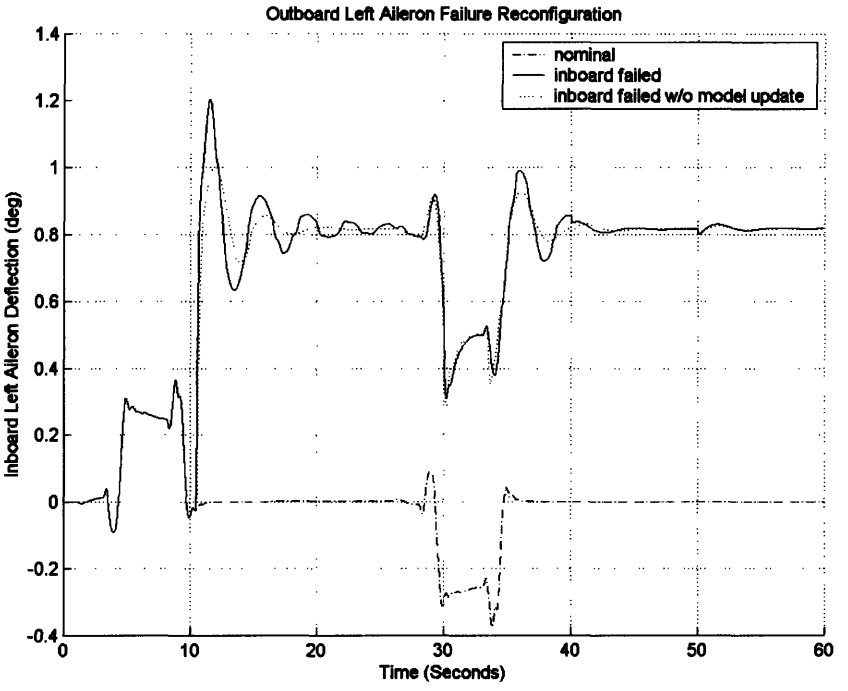


**Figure 69: Hardunder Outboard Left Aileron Failure Reconfiguration**

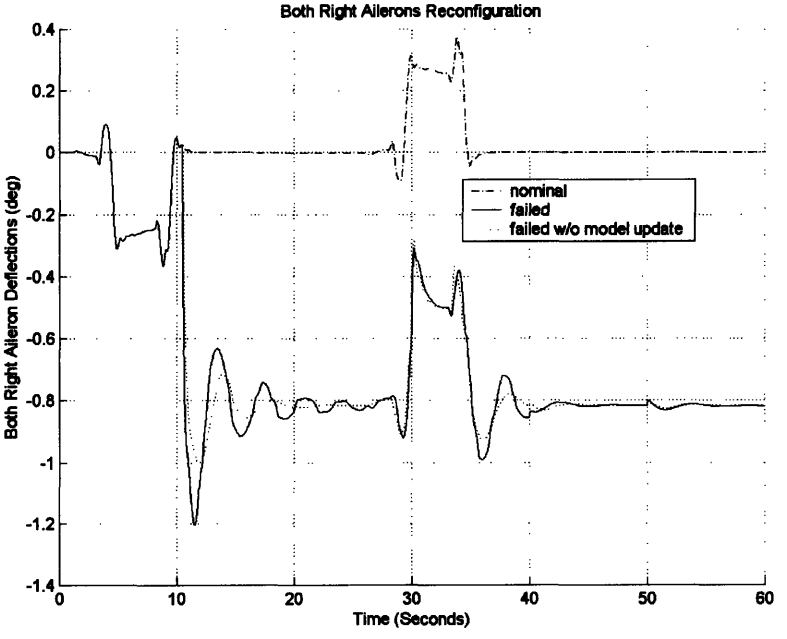


**Figure 70: Inboard and Outboard Right Aileron Failure Reconfiguration**

Again, in order to compensate for the added moment of the hardunder aileron failure the horizontal tail surfaces are used as well. The deflections of the horizontal tail surface can be seen in Figure 71. Only the deflections for a failure with internal model update are given for clarity.

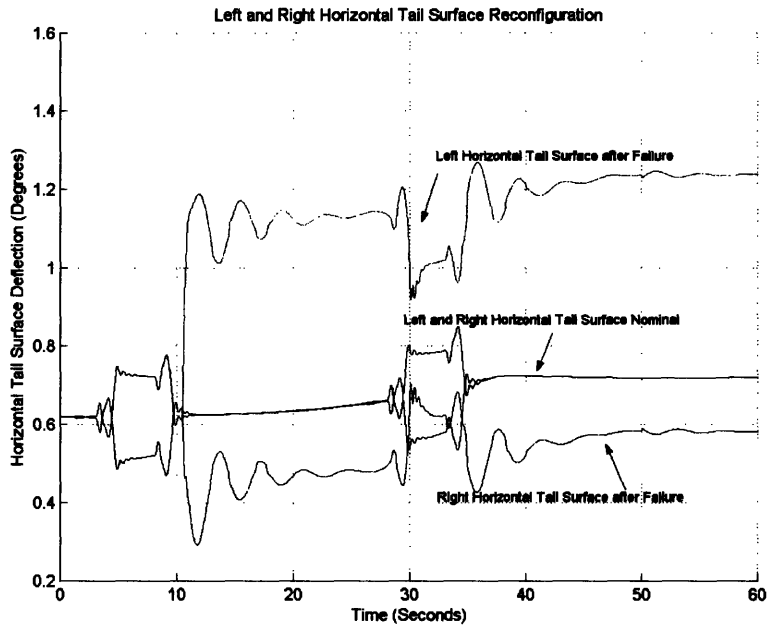


**Figure 71: Horizontal Tail Reconfiguration**

Roll Test Case 3

The frozen case scenario proved to be less enlightening. For brevity those results are not presented. The results are similar to the ones of the pitch frozen case, showing only slight differences between the nominal and failed cases.

## 6.3 Altitude Capture Mode Failures

The following are a series of failures in the two inboard split horizontal tail surfaces while the altitude capture autopilot mode is engaged. The maneuver is a simple 200 foot climb and then decent to the original altitude of 25,000 feet. The failure occurs at 10 seconds or just as the 200 foot plateau is reached. The change in altitude is mainly governed by the aircraft pitch angle and its dynamics are slow enough to allow for a 5 Hz

MPC controller rate update. Again, the same 4 second prediction horizon and 4 Laguerre bases are used

Altitude Test Case 1.

The first test is for the two inboard horizontal surfaces to go hard over to +10.5 degrees. In this test the capability of the MPC controller to constrain its output is shown. Figure 72 shows a plot of the altitude profile with a pitch rate constraint of +/- 15 degrees/sec.

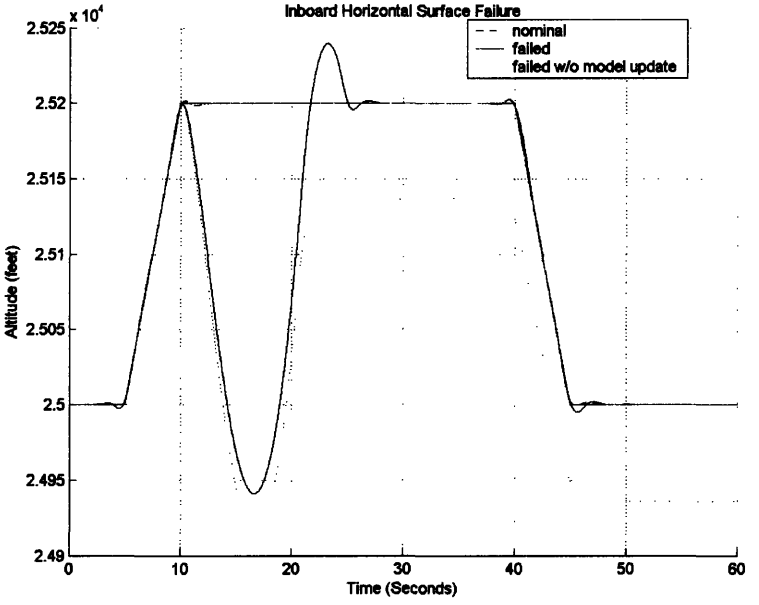


**Figure 72: Hardover Inboard Horizontal Tail Surface Failure Reconfiguration**

Since the input constraint is so small, a large transient is induced as the aircraft hits its pitch rate limit of 15 deg/sec and tries to maintain its trajectory. Again, the MPC controller, without model correction, has a large bias as it computes its trajectory with faulty information. The control profile can be seen in Figure 73. As mentioned, the MPC controller hits the constraint of 15 deg/sec as it tries to regain altitude. The resulting horizontal tail surface deflections can be seen in Figure 74.

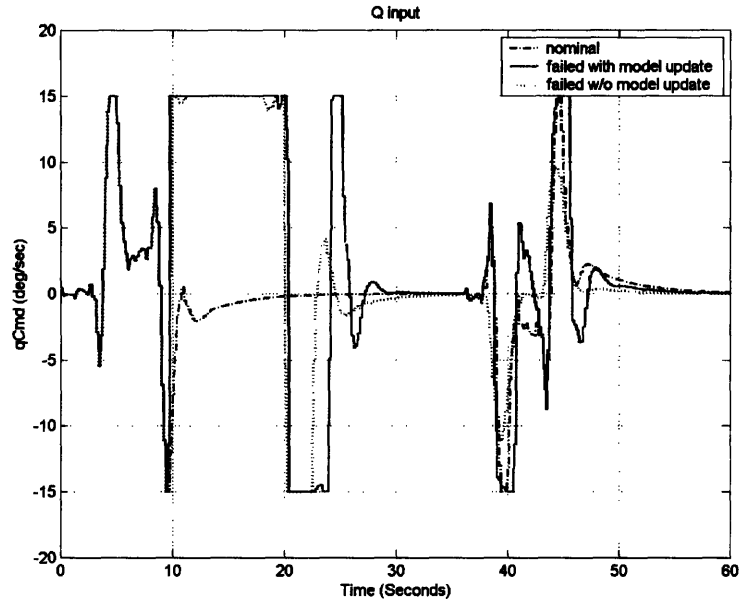


**Figure 73: Pitch Rate Command for Hardover Inboard Horizontal Tail Surface Controller Reconfiguration**

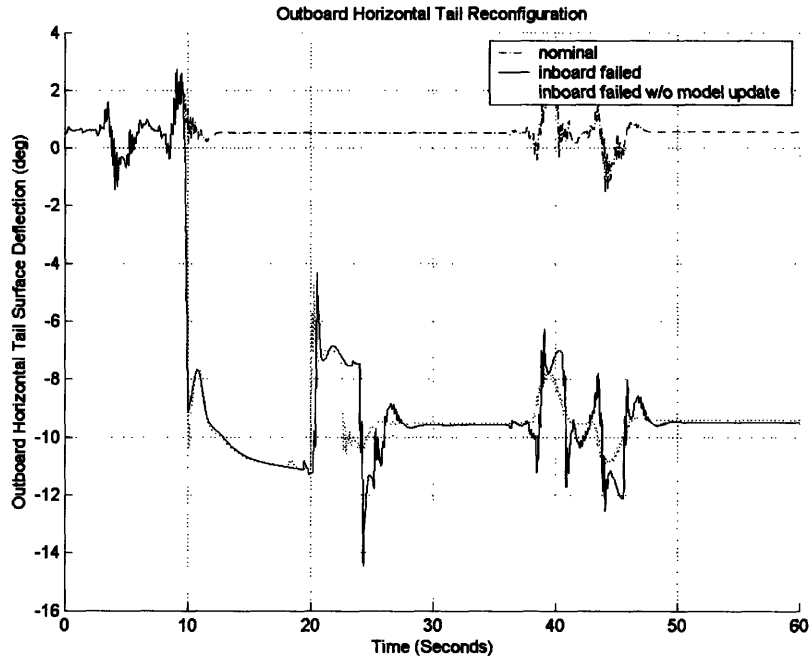


**Figure 74: Outboard Horizontal Tail Surface Reconfiguration**

The same hard over experiment was also run with the pitch rate constraint expanded to +/- 40 deg/sec. The results can be seen in Figures 75 and 76.

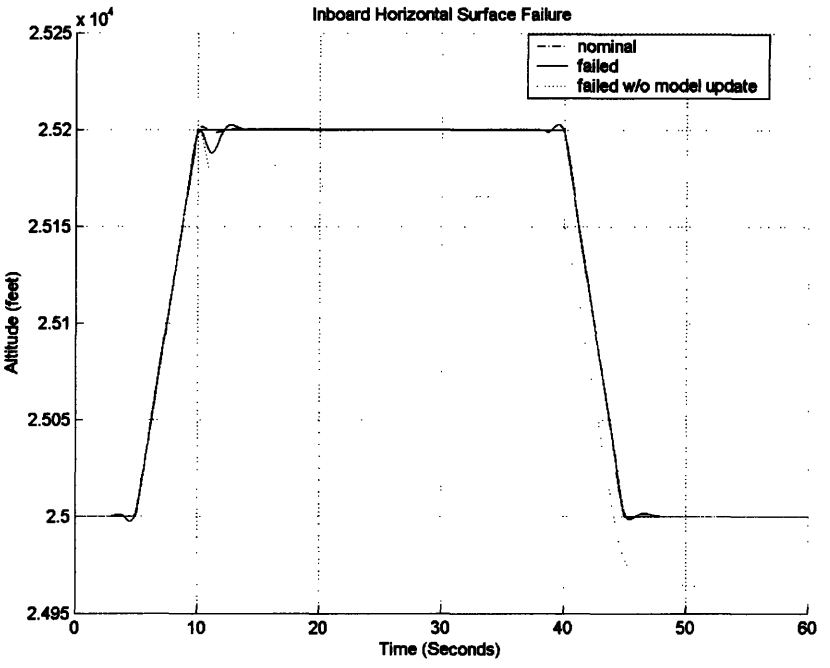


**Figure 75: Hardover Inboard Horizontal Tail Surface Failure Reconfiguration with Relaxed Contraints**

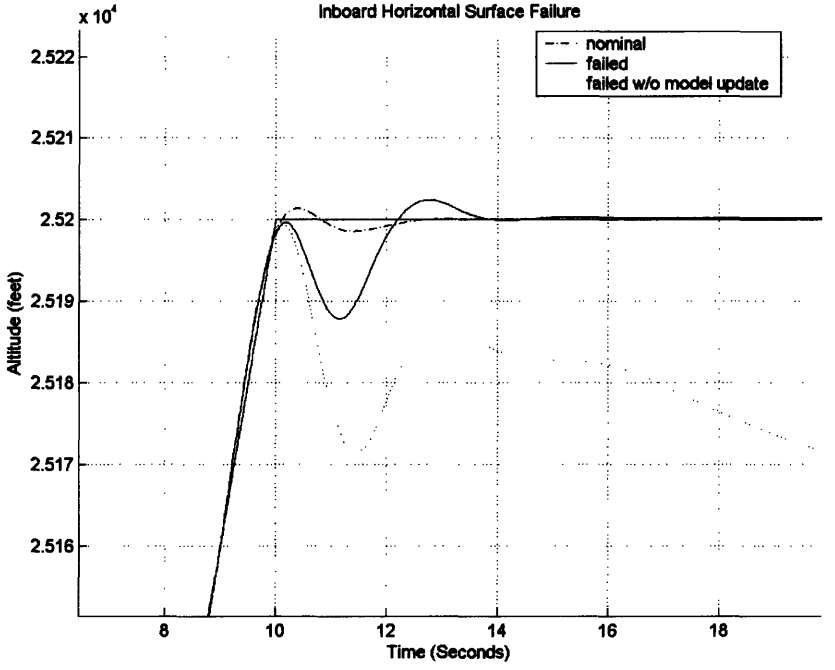


**Figure 76: Hardover Inboard Horizontal Tail Surface Failure Reconfiguration with Relaxed Contraints Zoom**

The transient is reduced dramatically as the constraints are relaxed. There is a transient of only 10 feet for the MPC controller with the updated model. The control profile in Figure 77 shows how the added spike of commanded pitch rate, at 10 seconds, serves to bring the aircraft quickly back on its target altitude.

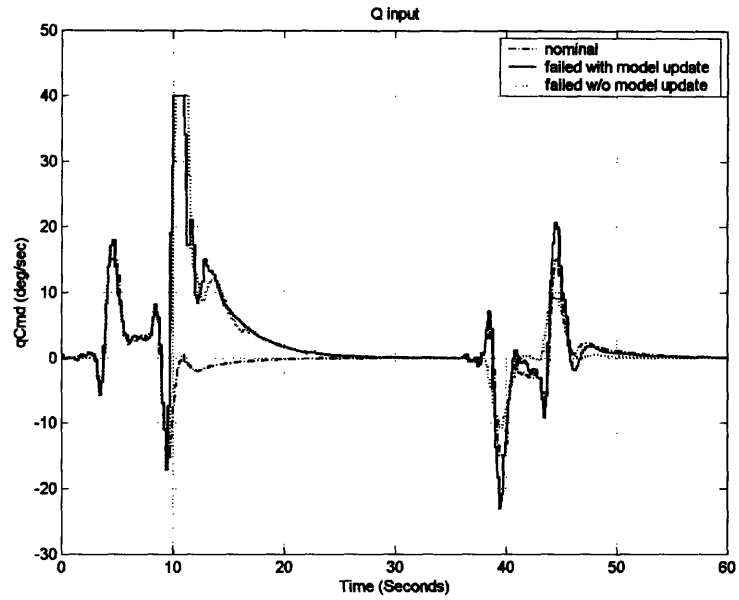


**Figure 77: Pitch Rate Command for Hardover Inboard Horizontal Tail Surface Controller Reconfiguration**

The higher controller spike at the point of failure allowed for the CAS to command a more aggressive horizontal tail surface movement so as to recover from the sudden hard over failure. This can be seen in Figure 78, with a deflection of almost -16 degrees shortly after the point of failure. This is compared to -10 degrees with the stronger input constraints.

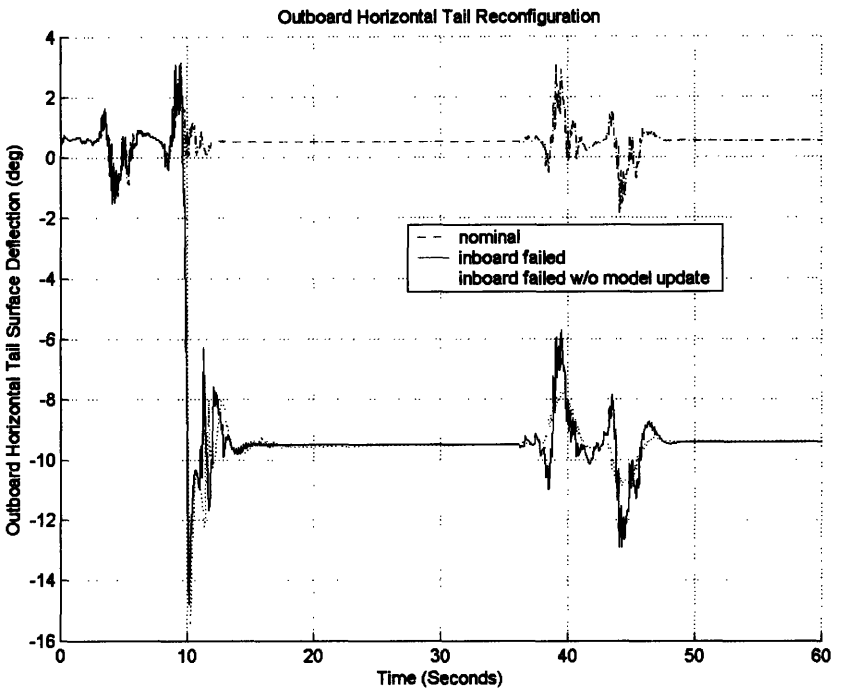


**Figure 78: Outboard Horizontal Tail Surface Reconfiguration**

Altitude Test Case 2

The previous test showed how the controller can hit its constraints and still recover the nominal trajectory after an initial transient. This test will show how the controller can recover the nominal trajectory with the control surface positions hitting their saturation levels as well. This is done by simulating a -10 degree hard under failure scenario in the two inboard horizontal tail surfaces at the same 10 second mark. These results can be seen in Figure 79. There is a huge transient caused not only by the controller saturation as in the last experiment but by the control surface saturation itself. The horizontal tail surfaces have a saturation level of 10.5 degrees and are shown to ride it while bringing the aircraft back to nominal. The controller and surface profiles can be seen in Figures 80 and 81 respectively.

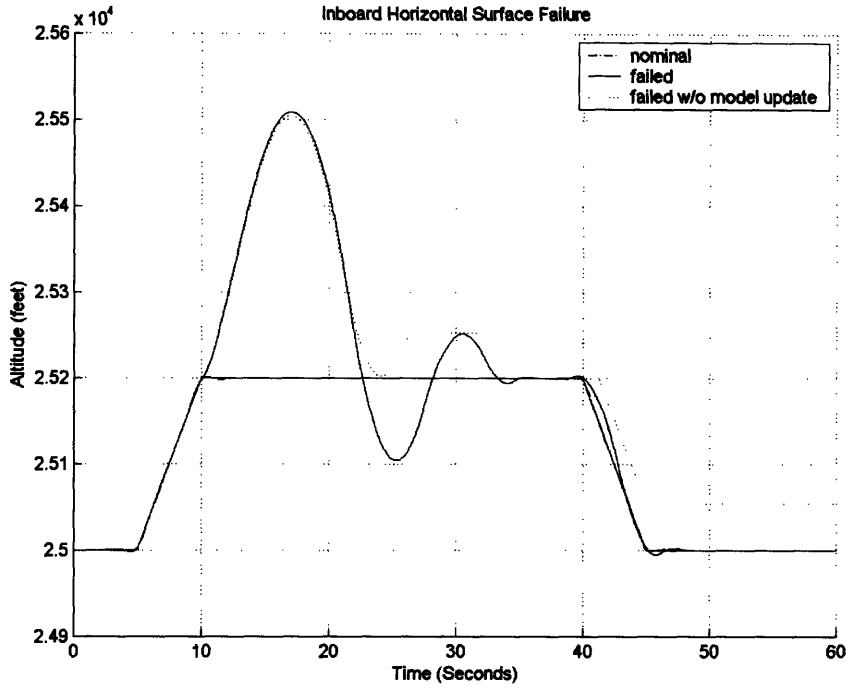


**Figure 79: Hardunder Inboard Horizontal Tail Surface Failure Reconfiguration**

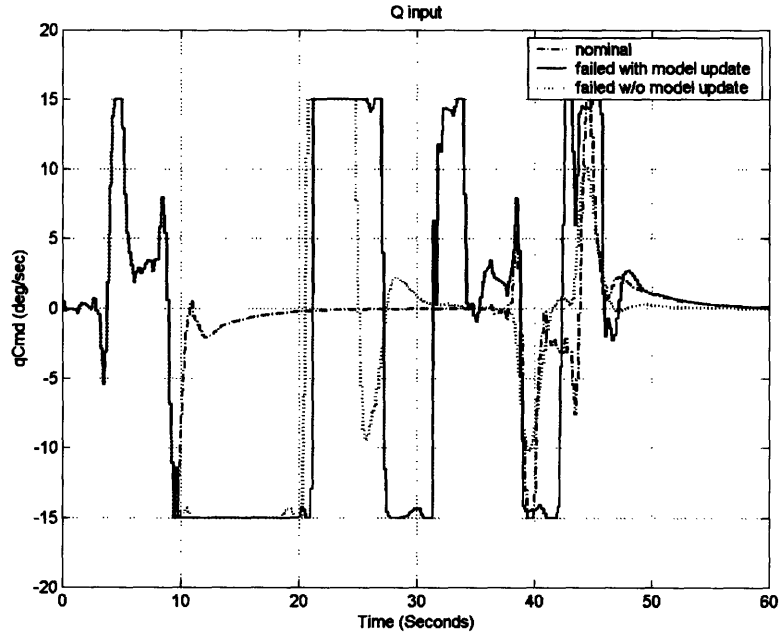


**Figure 80: Pitch Rate Command for Hardunder Inboard Horizontal Tail Surface Controller Reconfiguration**

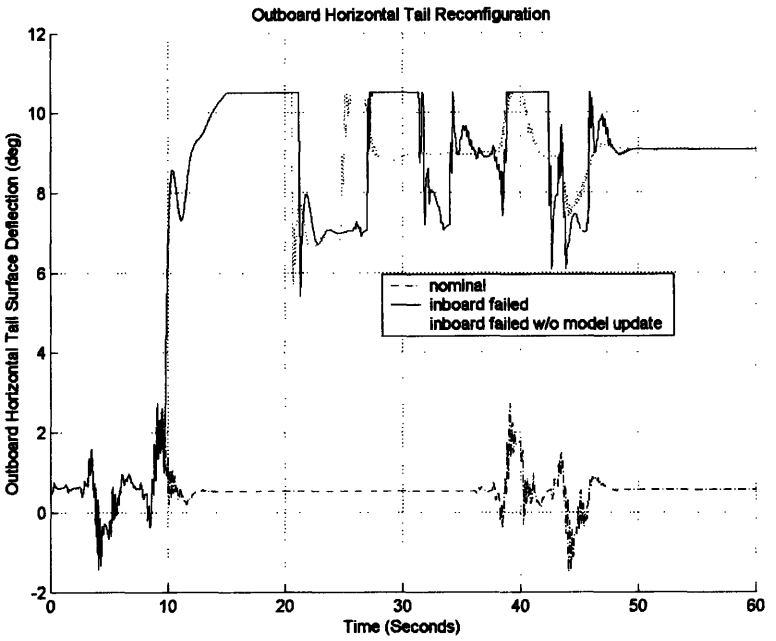


**Figure 81: Outboard Horizontal Tail Surface Reconfiguration**

Similar to the two previous autopilot modes, there was minimal controller reconfiguration in the frozen control surface scenario.

## 6.4 Altitude/Heading Mode Hold Failures

The altitude/heading hold autopilot mode has two new features. First it is a multiple input and multiple output controller, so both heading and altitude profiles can be tracked simultaneously. In addition, it has an additional redundant control feature which is the thrust vectoring system. It is this thrust vectoring system that will allow for more dramatic failure exercises. The maneuver is a climb by 400 feet in altitude and then a 15 degree heading change followed by a 400 foot descent to the original altitude. The controller update is 5 Hz with a prediction horizon of 4 seconds and 4 Laguerre bases are used.

Altitude/Heading Test Case 1

In the first failure exercise both rudders are failed hard over to +10 degrees at 5 seconds into the maneuver. This is well before any heading changes so the aircraft will be forced

to both track the heading profile, using only its thrust vectoring system, and counter the large amount of torque created by the hard over twin rudders. The resulting heading profile can be seen in Figure 82. It shows that in fact, with hard over full rudders, the vehicle can maintain control of its heading.

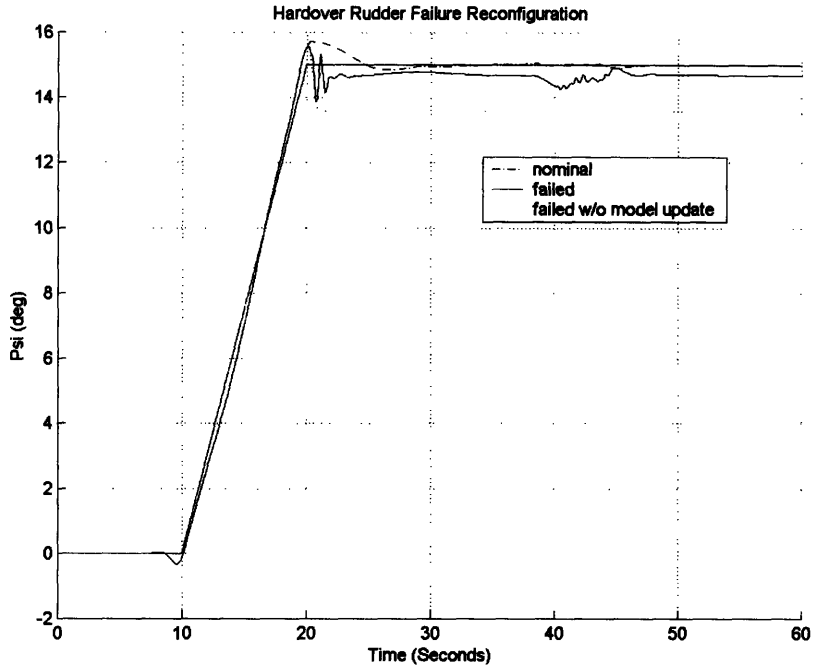


**Figure 82: Hardover Rudder Failure Reconfiguration**

The change in the MPC control output can be seen in Figure 83. The yaw rate input r lurches up as the failure is encountered and controls the aircraft through the entire maneuver. In Figure 84 below, the actual thrust vane deflections can be seen. It should also be mentioned that after the heading change the TVC vanes are kept to a constant deflection of -20 degrees. The TVC is counteracting the yawing moment generated by the failed rudders.

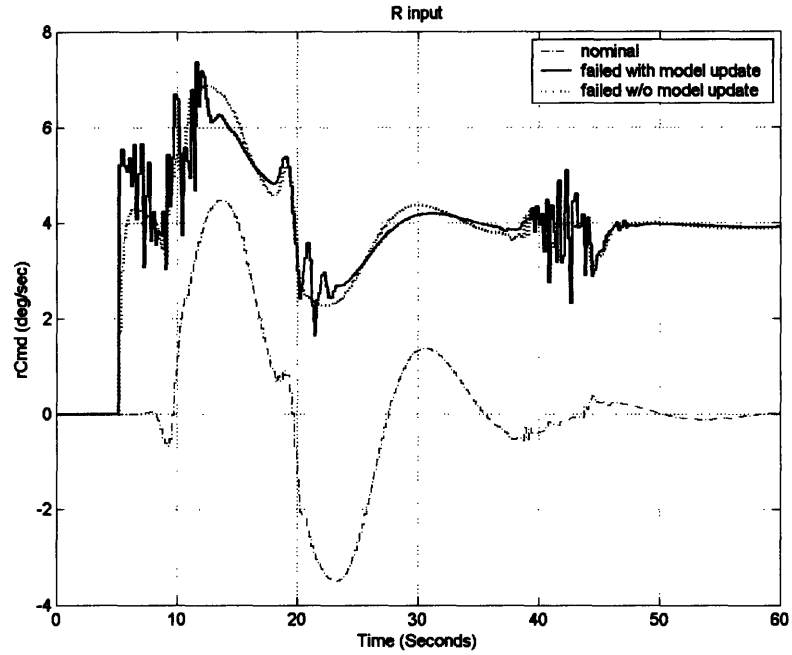


**Figure 83: Yaw Rate Command for Hardover Rudder Failure Controller Reconfiguration**

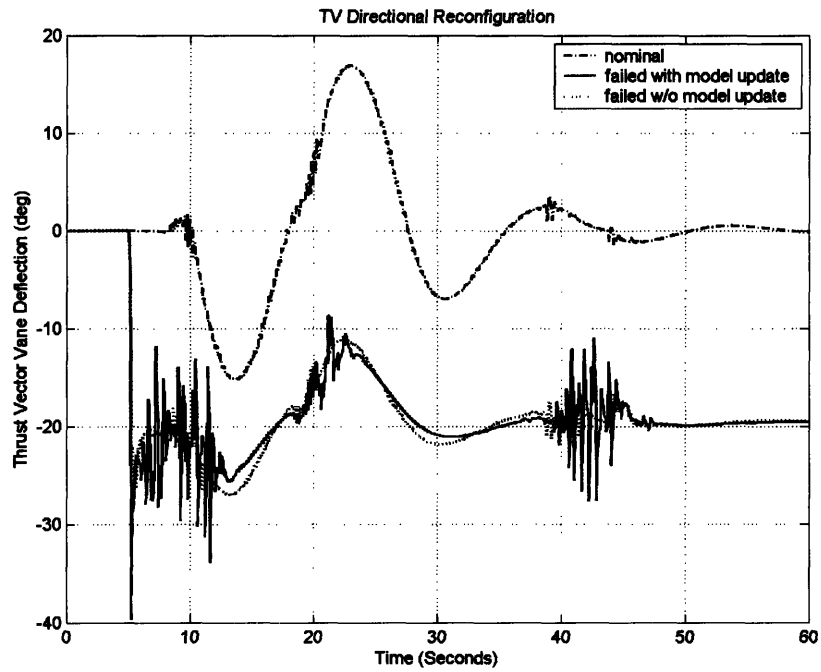


**Figure 84: Hardover Rudder Thrust Vane Reconfiguration**

Meanwhile in the longitudinal channel, the altitude track is near nominal despite the rudder failures. This can be seen below in Figure 85. Although nominal performance is

recovered, there is only a slight difference between the MPC longitudinal commands with and without the failure (See Figures 86 and 87).

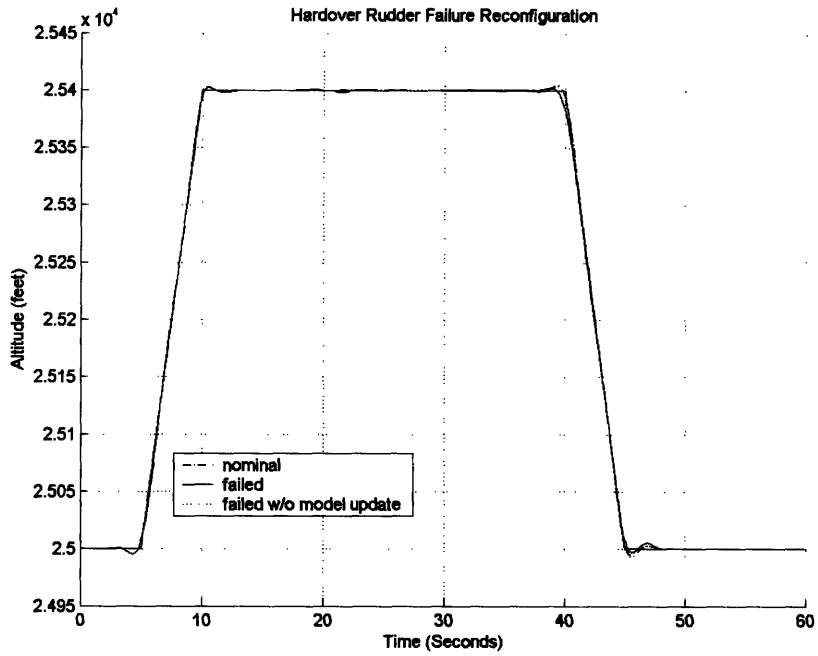


**Figure 85: Altitude/Rudder Hardover Reconfiguration**

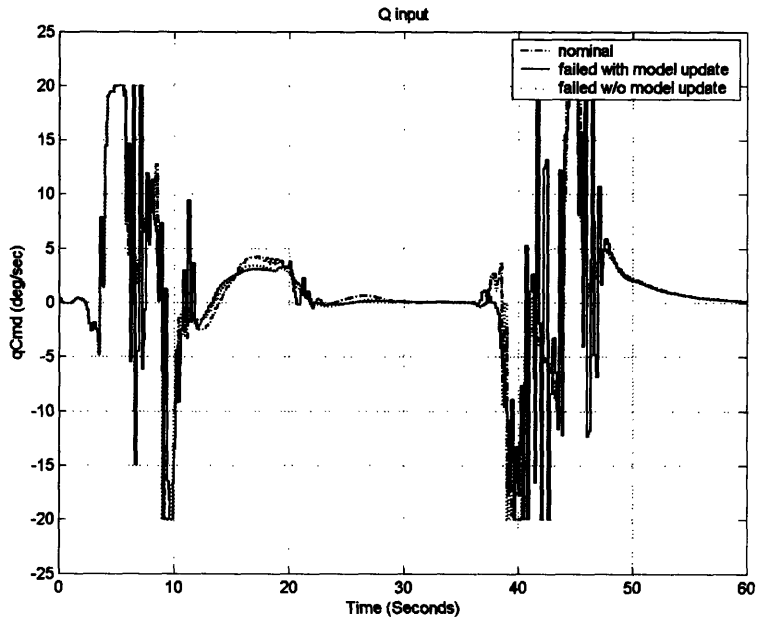


**Figure 86: Altitude Rudder Hardover Controller Reconfiguration**

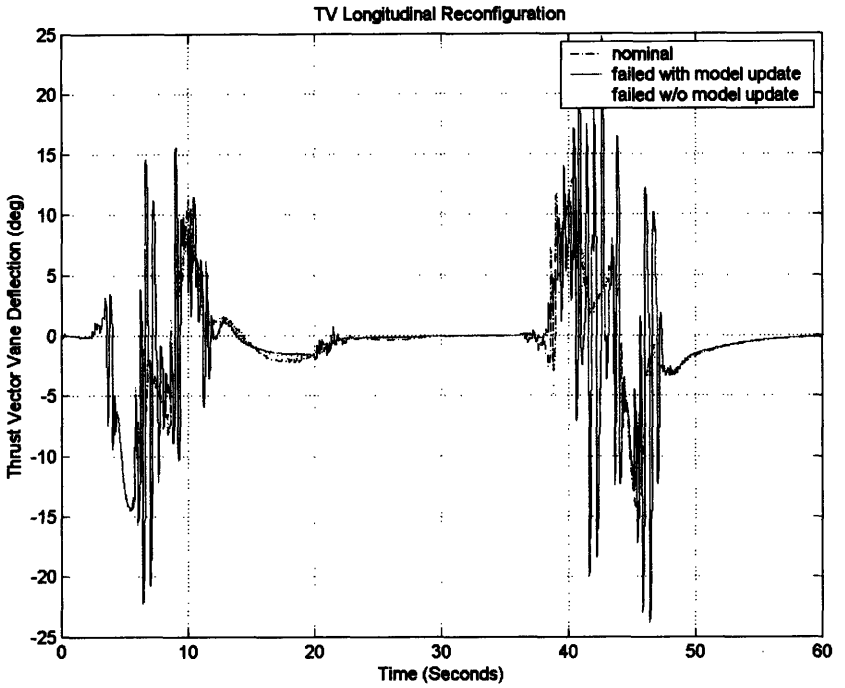


**Figure 87: Altitude/Rudder Hardover Thrust Vane Reconfiguration**

Altitude/Heading Test Case 2

The second failure experiment involves freezing the rudder position after 5 seconds into the flight. Since a maneuver has not been encountered, the rudder position stays at 0 for the duration of the flight. It can be seen in Figure 88 that the nominal is nearly completely recovered in the event of this failure.

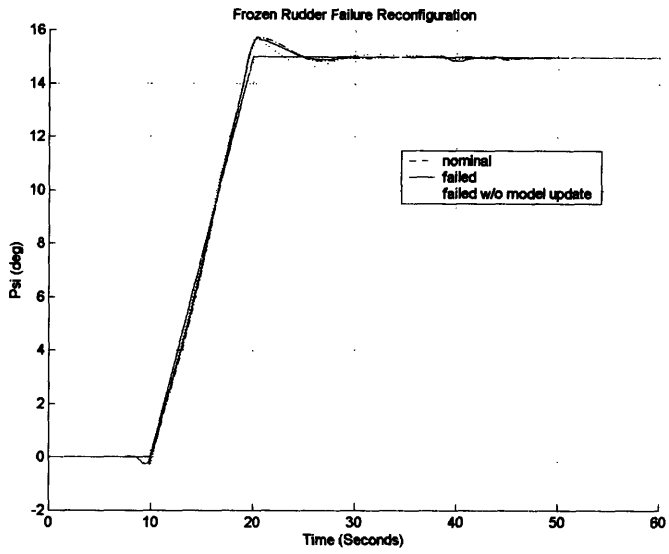


**Figure 88: Frozen Rudder Failure Reconfiguration**

There is only a slight undershoot when the MPC model is not updated. The MPC controller is still able to control the aircraft's heading with a faulty model of the frozen rudders. The effects of the controller reconfiguration can be seen in Figures 89 and 90. The resulting scheme may be counter intuitive because the nominal response goes slightly higher than the failed response. This is because the TVC can create very high moments but is operating at the slow rate of 5 Hz. This slow control rate is augmented by MPC by using the rudders, which operate at 50Hz. The rudders are used as fine tuning to counteract the large moments created by the TVC and keep following the tracking variable psi.

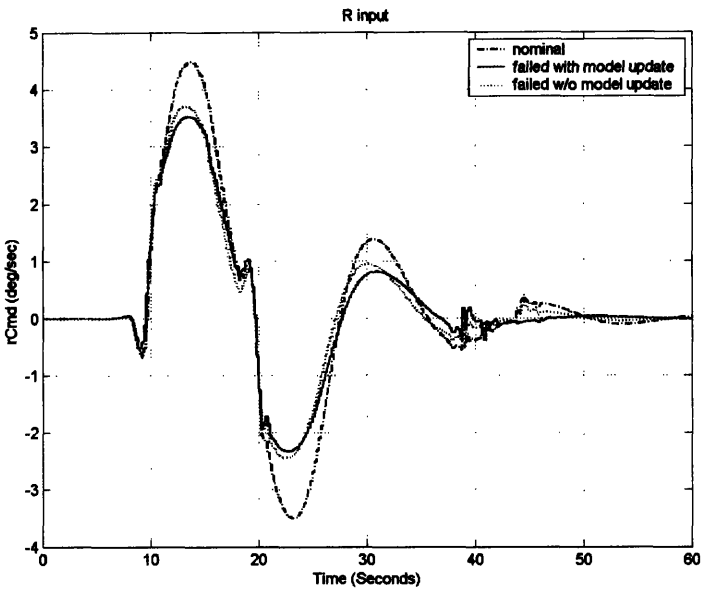


**Figure 89: Yaw Rate Command for Frozen Rudder Failure Controller Reconfiguration**

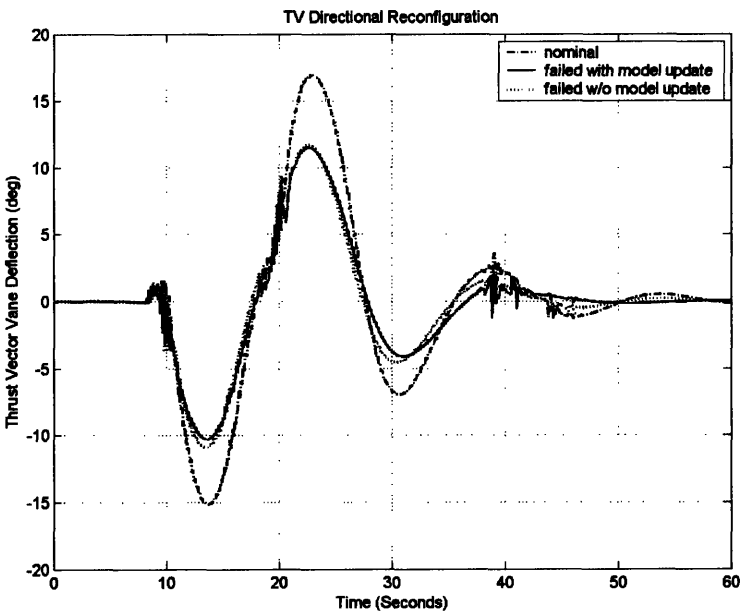


**Figure 90: Frozen Rudder Failure Thrust Vane Reconfiguration**

On the longitudinal channel, there is only a very slight amount of reconfiguration. The symmetry of the rudders and lack of deflection have virtually no effect on the longitudinal control sequences. This can be seen in altitude response in Figure 91 and control schemes in 92 and 93.

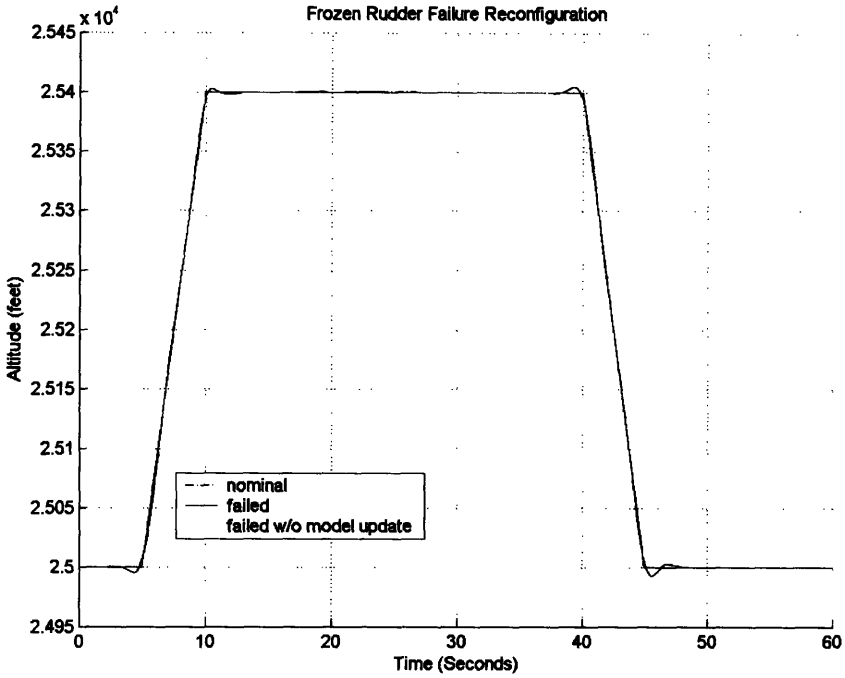


**Figure 91: Altitude/Rudder Frozen Reconfiguration**

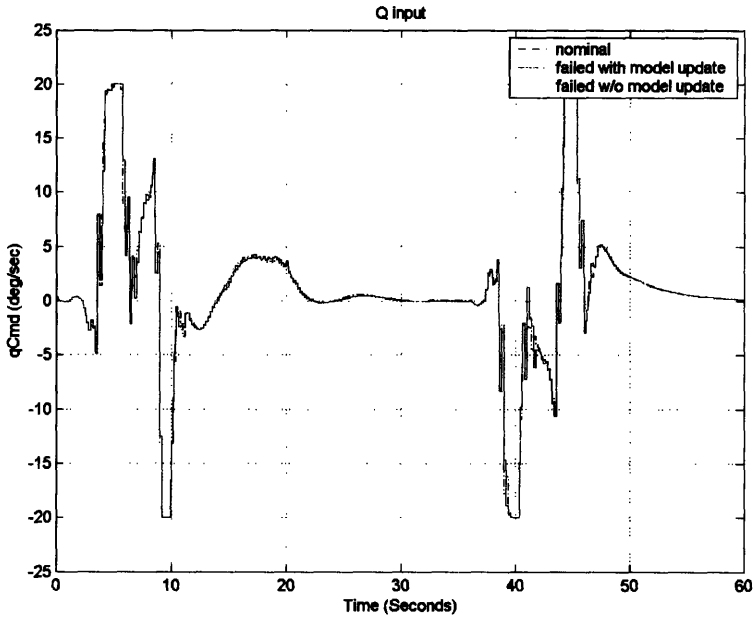


**Figure 92: Altitude/Rudder Frozen Controller Reconfiguration**

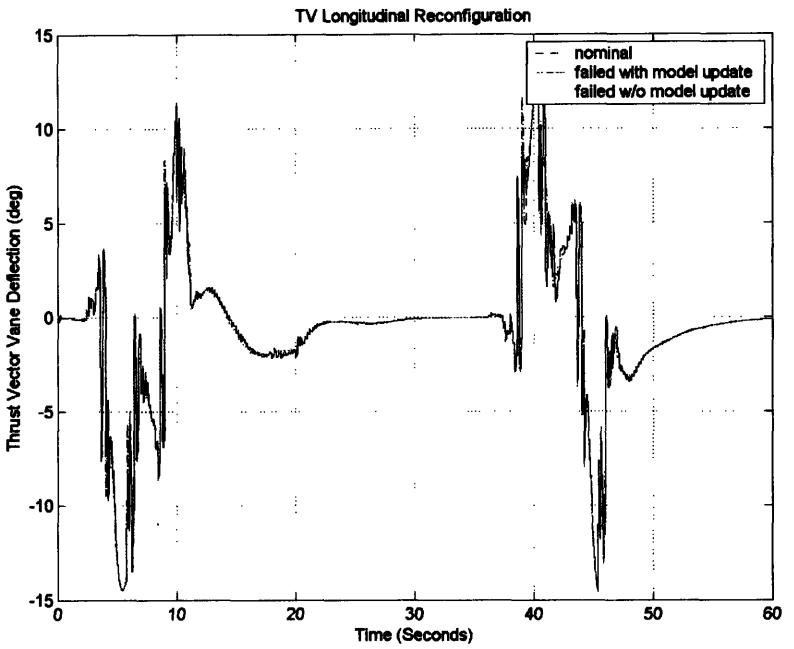


**Figure 93: Frozen Rudder Thrust Vane Reconfiguration**

Altitude/Heading Test Case 3

The final set of simulations is unique in the sense that it causes a loss of the aircraft when the MPC internal model is not updated. It is the same maneuver as before but with a failure in the two inboard horizontal tail surfaces at 10 seconds into the flight. In this first simulation the two inboard surfaces deflect to a +10.5 degrees hard over after 10 seconds. The response in the longitudinal channel can be seen below in Figure 94.

With MPC operating with an updated model, the response has only an overshoot but soon regains the reference trajectory. If the MPC internal model is not updated then the controller fails to control the aircraft as seen in Figure 95. The altitude plummets abruptly at around 27 seconds into the flight. This happens soon after the controller breaks out of its 30 deg/sec input constraints and begins to assign unreachable control commands which the CAS cannot possibly track. Figure 96 shows this dramatic failure by showing turning angles well over 90 degrees.

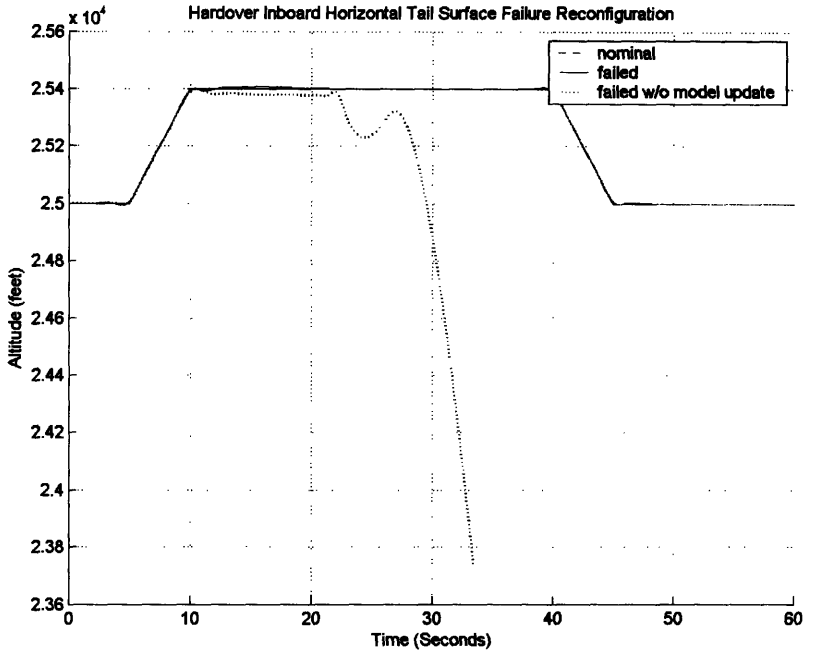


**Figure 94: Hardover Inboard Horizontal Tail Surface Failure Reconfiguration**

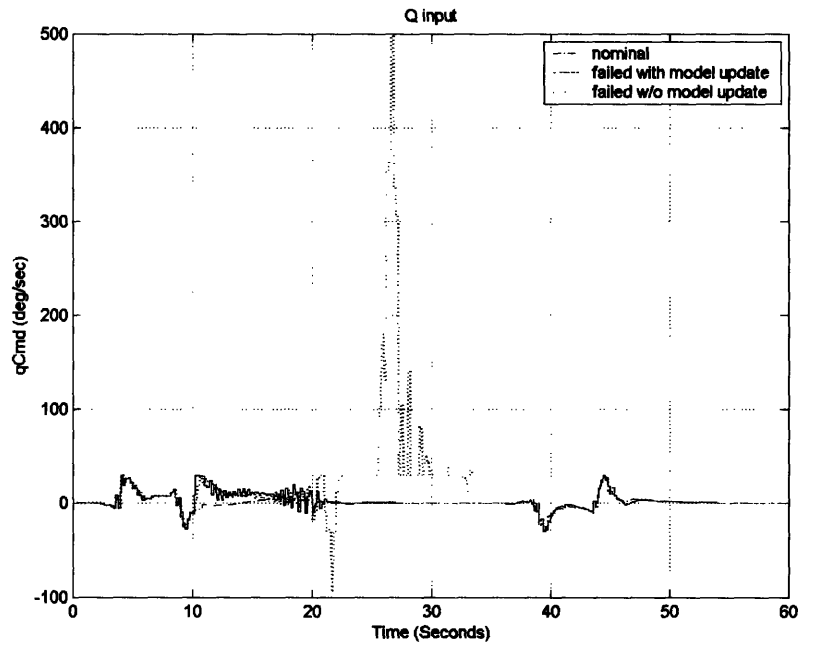


**Figure 95: Pitch Rate Command for Hardover Inboard Horizontal Tail Surface Failure Controller Reconfiguration**

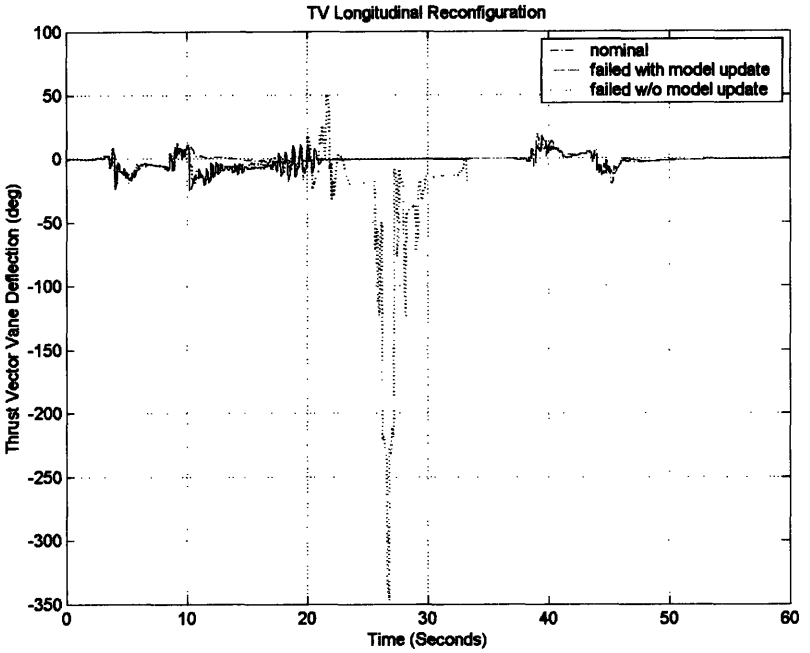


**Figure 96: Hardover Inboard Horizontal Tail Surface Failure Thrust Vector Reconfiguration**

Figure 97 below shows the new reconfigured control surface positions for the maneuver as well. The MPC controller without an updated model shows erratic behavior

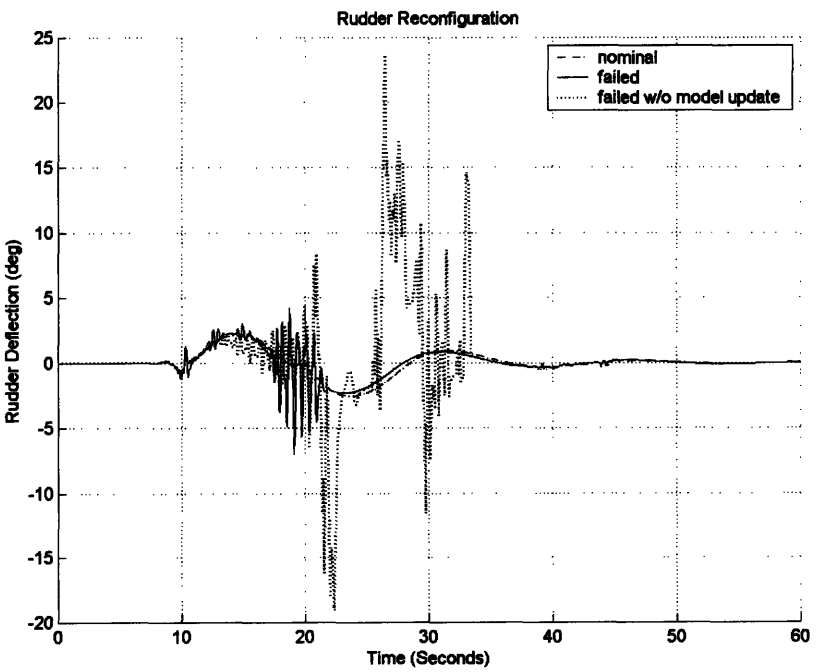


**Figure 97: Hardover Inboard Horizontal Tail Surface Failure Rudder Reconfiguration**

The MPC controller without the updated model starts losing control of the heading angle after about 20 seconds as shown in Figure 98. The heading plummets at 25 seconds into the flight as the controller hits the constraints and then breaks out of them. This sends the aircraft into a spin as the thrust vectoring tries to compensate. Eventually the simulation breaks down with erratic thrust vane angles and r input commands (See Figures 99 and 100).

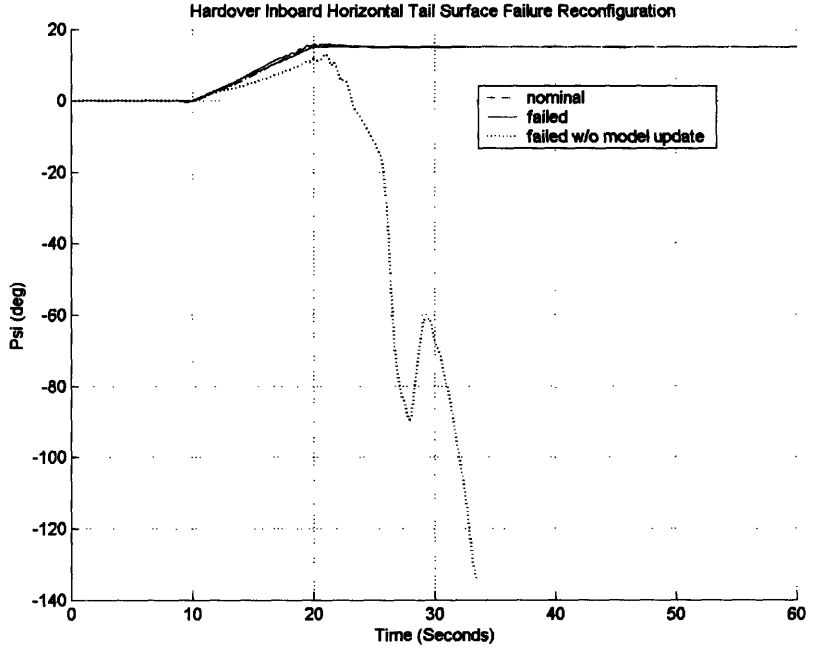


**Figure 98: Hardover Inboard Horizontal Tail Surface Failure Reconfiguration**

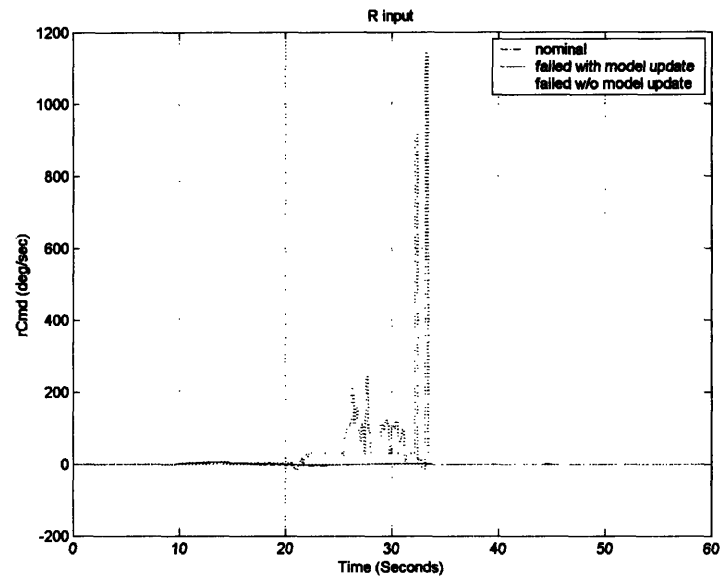


**Figure 99: Yaw Rate Command for Hardover Inboard Horizontal Tail Surface Failure Controller Reconfiguration**

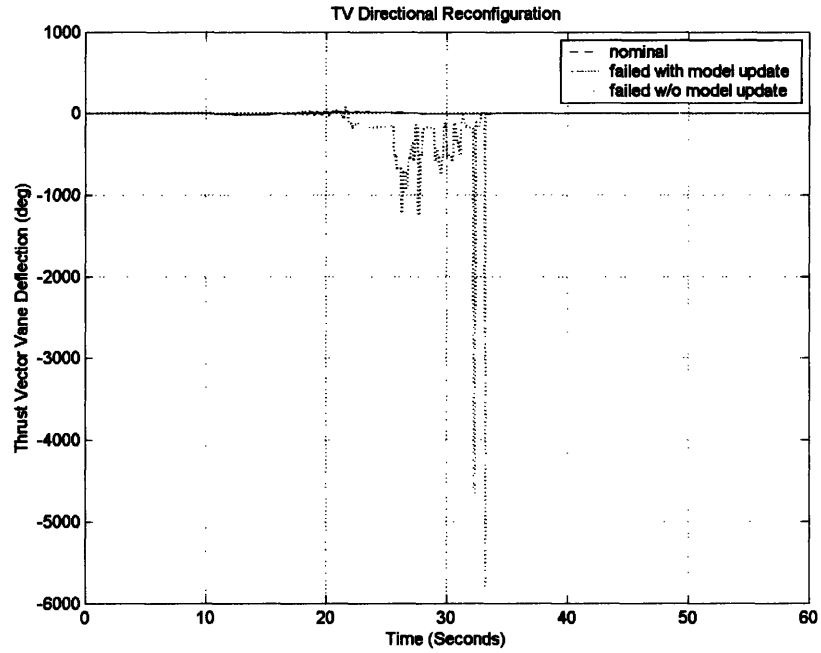


**Figure 100: Hardover Inboard Horizontal Tail Surface Failure Thrust Vector Reconfiguration**

Altitude/Heading Test Case 4

The hard under scenario shows similar results. The two inboard horizontal tails surfaces deflect to -10 degrees at 10 seconds into the flight. Again, with an updated model of the

failed plant, nominal performance is recovered. Although the craft is not lost, the simulation shows that there is great difficulty if the model is not updated. The simulation does not go numerically unstable but still produces unsatisfactory results. For instance, the simulation has the thrust vectoring vanes maintaining a sustained oscillation of upwards of 40 degrees just to maintain stable flight. These high amplitude, high frequency oscillations are occurring on both the longitudinal and directional channels of the thrust vanes. Clearly this is a case where, although the simulation results do not show a loss of the aircraft, there is a serious problem. In Figure 101 the MPC controller with the updated model can be shown to adapt quickly to the failure. The controller without an updated model again has tremendous difficulties. Figures 102-104 show the control schemes for this maneuver. Again, with the MPC controller without an updated model the high frequency and large amplitude oscillations can be seen throughout.

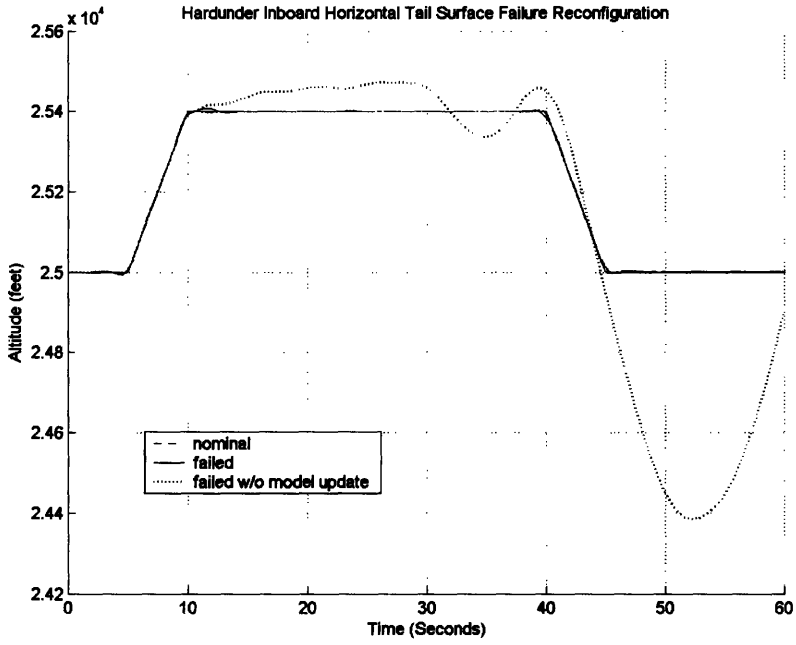


**Figure 101: Hardounder Inboard Horizontal Tail Surface Failure Reconfiguration**

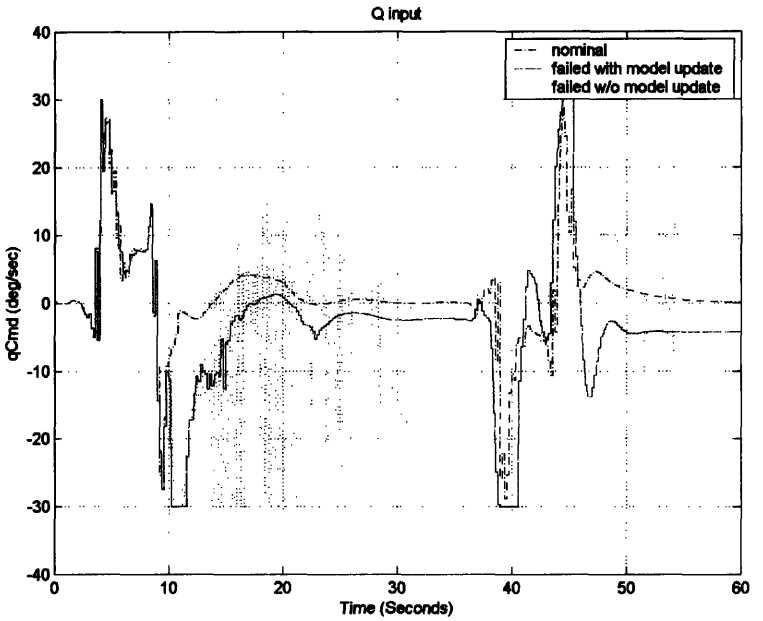


**Figure 102: Pitch Rate Command for Hardunder Inboard Horizontal Tail Surface Failure Controller Reconfiguration**

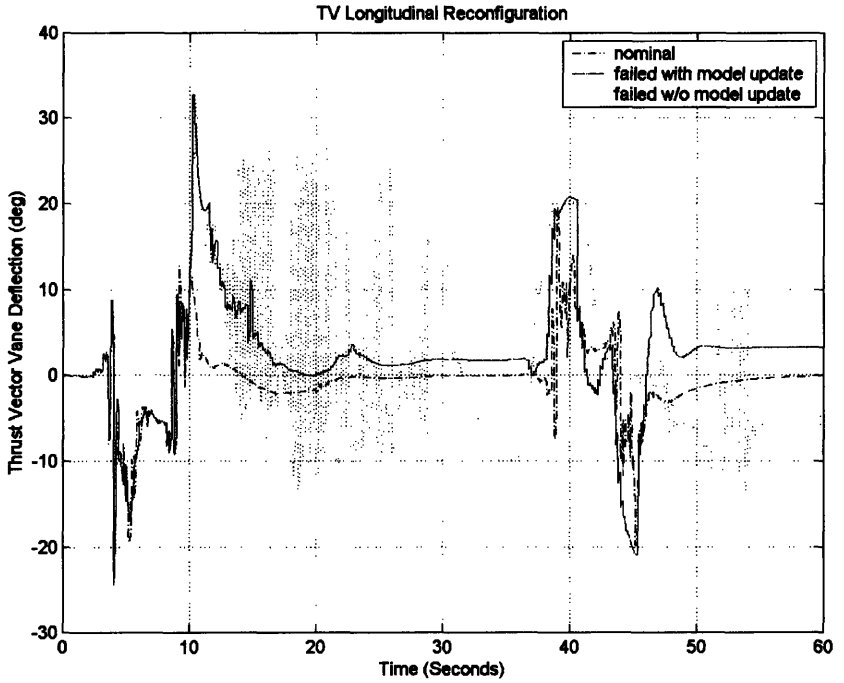


**Figure 103: Hardunder Inboard Horizontal Tail Surface Failure Thrust Vector Reconfiguration**

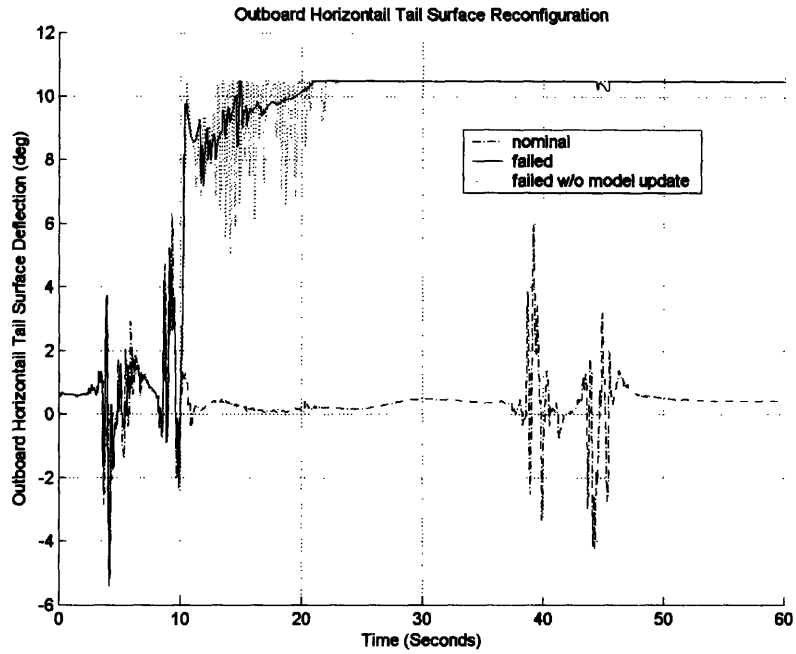


**Figure 104: Outboard Horizontal Tail Surface Reconfiguration**

The heading channel experiences similar problems of erratic oscillations in the controller. The heading angle varies by 10 degrees as the rudder and directional thrust vanes vary wildly. The heading response can be seen in Figure 105 with the control schemes in Figures 106-108.

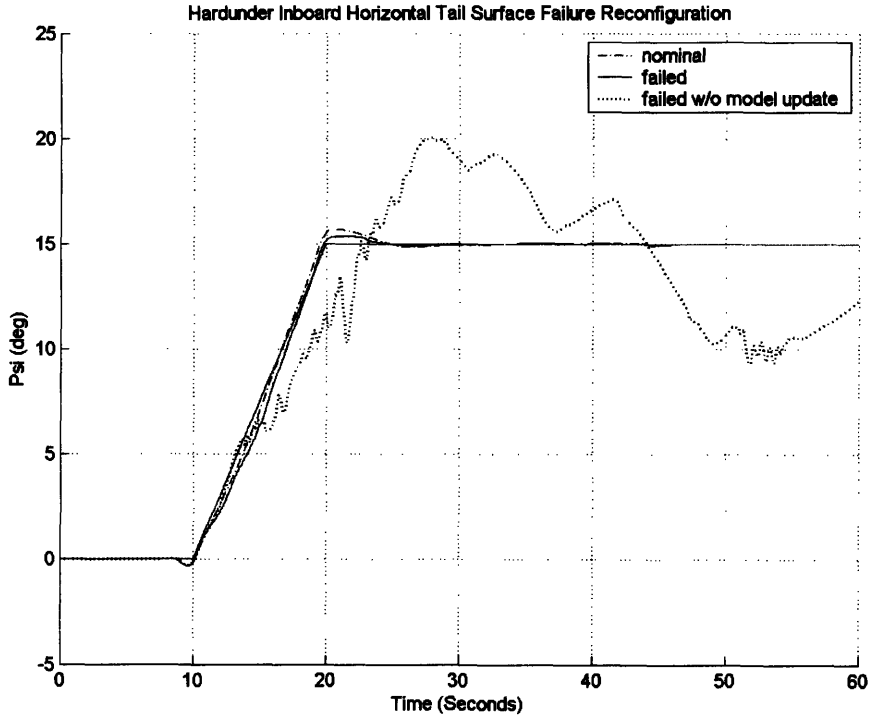


**Figure 105: Hardover Inboard Horizontal Tail Surface Failure Reconfiguration**

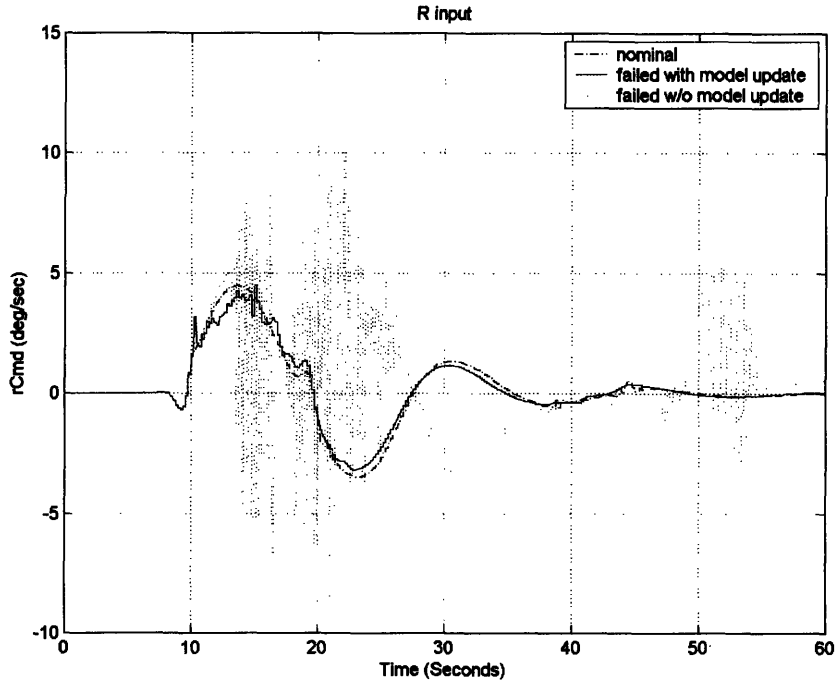


**Figure 106: Yaw Rate Command for Hardunder Inboard Horizontal Tail Surface Failure Controller Reconfiguration**

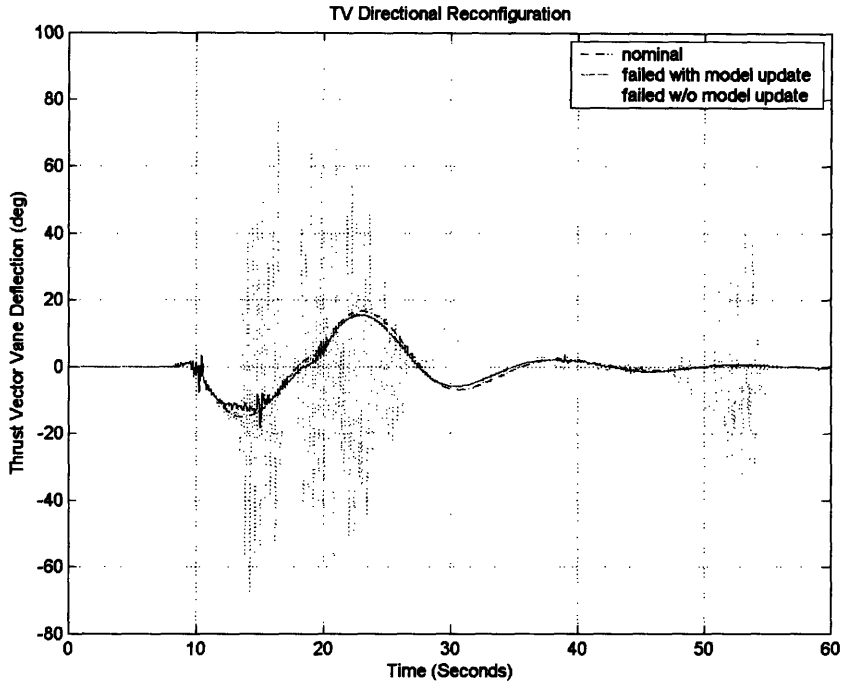


**Figure 107: Hardunder Inboard Horizontal Tail Surface Failure Thrust Vector Reconfiguration**

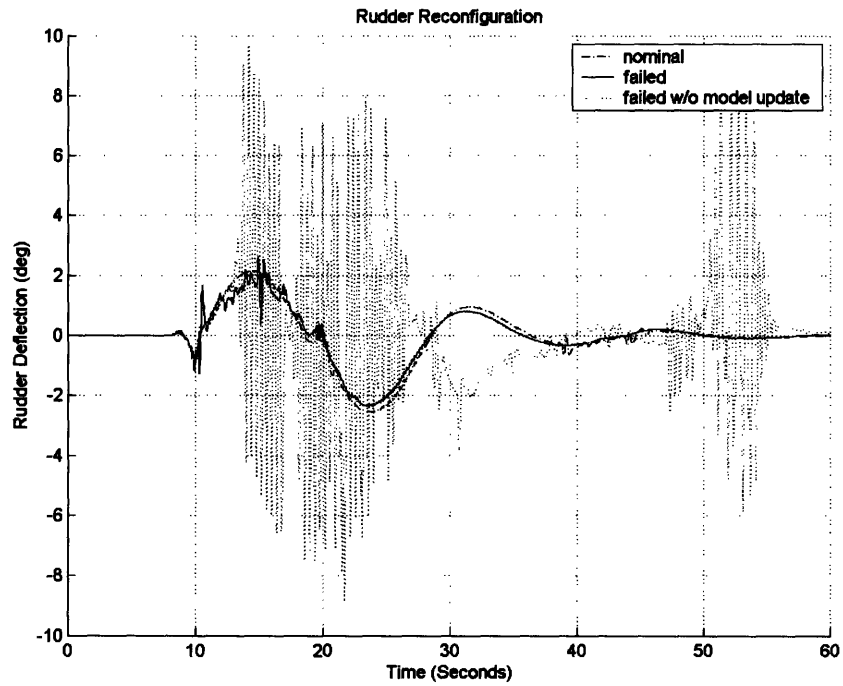


**Figure 108: Rudder Reconfiguration**

[This Page Intentionally Left Blank]

# Chapter 7

# Conclusions and Recommendations

## 7.1 Conclusions

This thesis provides an approach to the design of a reconfigurable autopilot by updating the internal model of an MPC controller. It is shown that in the face of substantial control surface failures MPC has the ability to reconfigure its control law and maintain acceptable performance by using an internal model of the failed plant. In the case when the internal model is not updated with the plant failure, the MPC autopilot has difficulty tracking the reference signal. In most cases a larger transient is seen with a steady state error throughout the rest of the maneuver. In the case of the altitude/heading autopilot however, the use of an un-updated model, while experiencing a partial horizontal tail failure, resulted in the loss of the aircraft.

A principal advantage of the MPC controller presented in this work is the fact that it uses a nonlinear model of the plant to predict the outputs over the time horizon. In case of a known failure the MPC internal model can be updated during flight. This is opposed to state space methods where the entire linear model would have to be re-calculated in case of a failure at each point in the flight envelope. The nonlinear model can also be reused in all autopilot modes. Finally, the MPC controller can impose constraints on the inputs to the CAS and the vehicle states as a failure occurs. For example, constraints can be used to limit the performance of the vehicle in case of a failure in order to safeguard against unforeseen dangers.

In order to make MPC a real time process it was developed as an outer loop and a perturbational method was used. By placing MPC around a stabilizing inner loop, the MPC controller rate could be lowered to as much as 5 Hz while maintaining acceptable performance. The dynamic of the plant required a time horizon of four seconds. The rate

of the propagator in the MPC controller was 50 Hz because the inner loop CAS was included in the model. The total number of independent variables in this case was 200 (4*50) times the number of inputs. In order to reduce the size of the QP problem basis functions were introduced as perturbations to the nominal input. This reduced the number of independent variables to only 4 when orthogonal Laguerre polynomials were used as the basis set. The Laguerre polynomials only required slight tuning of the weighting matrices Q and R to account for the oscillatory transient response of the system.

Several autopilot modes and failures were tested in this work. The pitch hold autopilot mode was found to recover suitable performance after a very short duration. The controller proceeded with an aggressive input spike to regain its nominal as fast as possible. In the case of an un-updated model, there was a steady state bias and tracking of the reference was therefore lost.

The bank hold autopilot mode proved to be extremely time sensitive and so the controller rate had to be increased relative to the other autopilot modes. The failure updated MPC controller proved to be able to bring the aircraft back to wings level. The MPC controller without an updated model had a larger transient and reached a steady state bias, unable to reach wings level.

The altitude capture autopilot mode with failure updated MPC controller was found to regain the altitude reference with a transient proportional to the amount of constraints put on the input or pitch rate. The un-updated model had a large steady state bias but proved to be stable.

The heading/altitude hold mode showcased the failure updated MPC controller's ability to reconfigure itself on both the longitudinal and directional channels. Furthermore, the entire rudder was failed and the thrust vectoring was shown to fully compensate for it. As for the MPC controller without failure update, the discrepancy between the true failed model proved to be too great resulting in the loss of the aircraft.

MPC shows many advantages over other approaches to reconfigurable control. The automated gain schedule is the most straightforward of all methods and though may

obviously work for failures foreseen, is cumbersome and time consuming. The amount of time creating the controllers is extended greatly as each is tuned for its particular failure. MPC's advantage to this is that only one controller is designed and the algorithm itself solves for the new strategy as the failure is encountered.

Another class of reconfigurable control systems reacts to a failure reallocating the commanded moments to the remaining un-failed surfaces. In our scheme, the moment allocation problem is kept confined to the CAS which assigns control surface positions as prescribed by its inputs from MPC. MPC is explicitly aware of how the inner loop CAS will respond because it uses a model of the entire inner loop when it predicts ahead. The CAS itself however is not updated as a result of the reconfiguration. The other more exotic techniques are lumped as general constrained optimizations and are all striving for most of what MPC can actually do. MPC can explicitly take a failed model of the plant and use it to find the optimal control strategy as defined by its cost function for a given failure. This is opposed to implicit model following where certain key measurements that are deemed fundamental to aircraft performance are monitored and used to develop a new control strategy.

There is great promise in MPC as faster computers make highly computational control algorithms easier to implement. The design of next generation controllers for aerospace vehicles will then shift to creating higher fidelity models to place inside the MPC engine. This will drastically reduce the amount of time spent designing a control system and reconfiguration capabilities would be possible as long as the model was able to incorporate failure information.

## 7.2 Recommendations for Further Work

MPC has the capability to create a reconfigurable controller by means of an updatable model. The process of updating the internal model has many applicable consequences and lends itself to many other types of reconfiguration. A degree of payload reconfiguration can be added when the internal model is simply updated with the new payload capacity for a mission. The amount of time tuning a new controller just for this slight augmentation would be enormous.

Furthermore MPC can be used to provide a degree of mission reconfiguration. This is because the MPC algorithm is virtually identical for all autopilots. In the event of a failure and a re-planning of the entire mission, the controller can be updated relatively easily. This is because all that needs to be updated are the inputs and outputs that MPC is tracking and creating.

Perhaps the most pressing issue to be investigated relative to this research is the addition of engine throttle as a control input to MPC in the event of failure. The engines are an often overlooked redundant control effecter for the aircraft. They are capable of controlling the aircraft in all three principal axes. The addition of this input would reduce the need for redundant surface controls. Furthermore the algorithm could be placed on existing aircraft without the costly addition of redundant control surfaces or thrust vectoring.

# Appendix A

## Internal Model Generation

The internal model used by the MPC controller in this thesis was created by using Real Time Workshop (RTW) from MathWorks Inc. This enabled the entire simulink diagram, complete with S functions, to be converted into stand alone C code. This C code can then be fully integrated into the larger MPC controller algorithm making it entirely in C. The simulink diagram must explicitly have inputs and outputs labeled for the internal class structure assigned by RTW. A simulink snapshot of the aircraft model explicitly in this state before it is auto coded can seen in Figure 109.

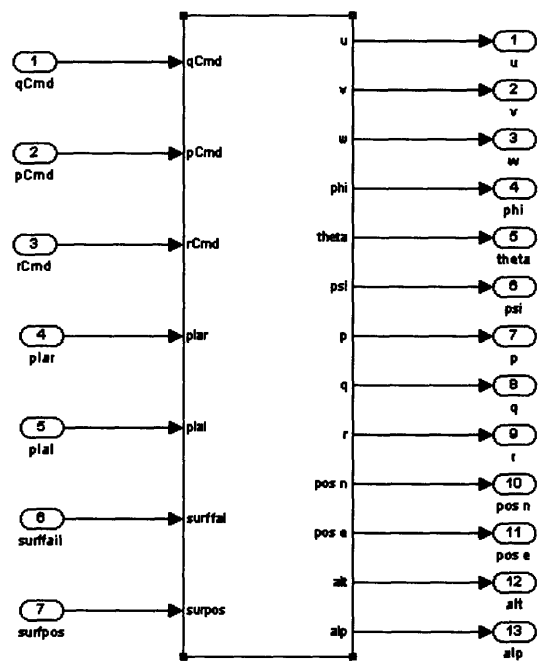


**Figure 109: Aircraft Simulink Diagram for RTW**

RTW will then generate the necessary C files complete with headers and an internal class structure. The key components created are an initialization function and a step function. The initialization function must be run before executing the step function and loads all gain, trim and other data. The step function is created to be run at the rate at which the entire model is discretized. The propagator works by simply calling this step function for the appropriate amount of points in the prediction horizon. The actual propagator function can be seen below.

```
void propagator_calc(propagator *me) {

        int i, j;

        me->out.numPoints = me->in.numPoints;

        for(j=0; j<me->out.numPoints; j++) {

                if (j == 0) {

                        Aircraft_propr_initialize(1);

                        /* assign initial states */

                        for (i=0; i<11; i++)
                                Aircraft_propr_DWork.Enginput_DSTATE[i] = me->state.Enginput.value[i];

                        for (i=0; i<6; i++)
                                Aircraft_propr_DWork.FMstates_DSTATE[i] = me->state.Fmstates.value[i];

                        for (i=0; i<2; i++)
                                Aircraft_propr_DWork.Engdynstates_DSTATE[i]=me>state.Engdynstates.value[i]
                        ;

                        for (i=0; i<37; i++)
                                Aircraft_propr_DWork.Aeroinput_DSTATE[i] = me->state.Aeroinput.value[i];

                        Aircraft_propr_DWork.Derivative_DSTATE = me->state.Derivative.value[0];

                        for (i=0; i<12; i++)
                                Aircraft_propr_DWork.xState_DSTATE[i] = me->state.xStates.value[i];

                        Aircraft_propr_DWork.LongCas1_DSTATE = me->state.LongCas1.value[0];
                        Aircraft_propr_DWork.CASint_DSTATE = me->state.CASint.value[0];

                        Aircraft_propr_DWork.LatCas1_DSTATE = me->state.LatCas1.value[0];
                        Aircraft_propr_DWork.LatCas2a_DSTATE = me->state.LatCas2a.value[0];
                        Aircraft_propr_DWork.LatCas2b_DSTATE = me->state.LatCas2b.value[0];
                        Aircraft_propr_DWork.LatCas3a_DSTATE = me->state.LatCas3a.value[0];
                        Aircraft_propr_DWork.LatCas3b_DSTATE = me->state.LatCas3b.value[0];
                        Aircraft_propr_DWork.DirCas1_DSTATE = me->state.DirCas1.value[0];
                        Aircraft_propr_DWork.DirCas2_DSTATE = me->state.DirCas2.value[0];
                        Aircraft_propr_DWork.DirCas3_DSTATE = me->state.DirCas3.value[0];
                        Aircraft_propr_DWork.DirCas4a_DSTATE = me->state.DirCas4a.value[0];
                        Aircraft_propr_DWork.DirCas4b_DSTATE = me->state.DirCas4b.value[0];
                        Aircraft_propr_DWork.DirCas5_DSTATE = me->state.DirCas5.value[0];

                }
```

```
            Aircraft_propr_U.plal   = me->in.plal.value[j];
            Aircraft_propr_U.plar   = me->in.plar.value[j];
            Aircraft_propr_U.qCmd   = me->in.qCmd.value[j];
            Aircraft_propr_U.pCmd   = me->in.pCmd.value[j];
            Aircraft_propr_U.Surffail[0] = me->in.surffail1.value[j];
            Aircraft_propr_U.Surffail[1] = me->in.surffail2.value[j];
            Aircraft_propr_U.Surffail[2] = me->in.surffail3.value[j];
            Aircraft_propr_U.Surffail[3] = me->in.surffail4.value[j];
            Aircraft_propr_U.Surffail[4] = me->in.surffail5.value[j];
            Aircraft_propr_U.Surffail[5] = me->in.surffail6.value[j];
            Aircraft_propr_U.Surffail[6] = me->in.surffail7.value[j];
            Aircraft_propr_U.Surffail[7] = me->in.surffail8.value[j];
            Aircraft_propr_U.Surffail[8] = me->in.surffail9.value[j];
            Aircraft_propr_U.Surffail[9] = me->in.surffail10.value[j];
            Aircraft_propr_U.Surffail[10] = me->in.surffail11.value[j];
            Aircraft_propr_U.Surffail[11] = me->in.surffail12.value[j];
            Aircraft_propr_U.Surffail[12] = me->in.surffail13.value[j];
            Aircraft_propr_U.Surffail[13] = me->in.surffail14.value[j];
            Aircraft_propr_U.surfpos[0] = me->in.surfpos1.value[j];
            Aircraft_propr_U.surfpos[1] = me->in.surfpos2.value[j];
            Aircraft_propr_U.surfpos[2] = me->in.surfpos3.value[j];
            Aircraft_propr_U.surfpos[3] = me->in.surfpos4.value[j];
            Aircraft_propr_U.surfpos[4] = me->in.surfpos5.value[j];
            Aircraft_propr_U.surfpos[5] = me->in.surfpos6.value[j];
            Aircraft_propr_U.surfpos[6] = me->in.surfpos7.value[j];
            Aircraft_propr_U.surfpos[7] = me->in.surfpos8.value[j];
            Aircraft_propr_U.surfpos[8] = me->in.surfpos9.value[j];
            Aircraft_propr_U.surfpos[9] = me->in.surfpos10.value[j];
            Aircraft_propr_U.surfpos[10] = me->in.surfpos11.value[j];
            Aircraft_propr_U.surfpos[11] = me->in.surfpos12.value[j];
            Aircraft_propr_U.surfpos[12] = me->in.surfpos13.value[j];
            Aircraft_propr_U.surfpos[13] = me->in.surfpos14.value[j];

            Aircraft_propr_step();

            me->out.u.value[j]     = Aircraft_propr_Y.u;
            me->out.v.value[j]     = Aircraft_propr_Y.v;
            me->out.w.value[j]     = Aircraft_propr_Y.w;
            me->out.pitch.value[j] = Aircraft_propr_Y.pitch;
            me->out.phi.value[j]   = Aircraft_propr_Y.phi;
            me->out.psi.value[j]   = Aircraft_propr_Y.psi;
            me->out.pitchrate.value[j]   = Aircraft_propr_Y.pitchrate;
            me->out.p.value[j]   = Aircraft_propr_Y.p;
            me->out.r.value[j]   = Aircraft_propr_Y.r;
            me->out.pos_n.value[j] = Aircraft_propr_Y.pos_n;
            me->out.pos_e.value[j] = Aircraft_propr_Y.pos_e;
            me->out.alt.value[j]   = Aircraft_propr_Y.alt;
            me->out.alp.value[j] = Aircraft_propr_Y.alp;

        }
}
```

The simulink model itself was very complicated consisting of multiple inner loops, second and first order transfer functions, and C S-Functions. The transfer functions and delays created additional states that had to be initialized before the propagator could simulate ahead. These all had to be dealt with separately to satisfy both MPC's need to have the propagator resetable and for RTW to have the model discrete.

In order for the transfer functions in the CAS to be made fully resetable they had to be put into state space form. After this they could be implemented in simulink to bring the

integrators outside. Once this was done, the integrator simply had to be reassigned a new initial value each time using feedback state information from the truth model. An example of this involving a 44 Hz filter in the Lateral CAS system is given.

The transfer function for this filter is the following.

$$\frac{(1/44)^2 s^2 + 2*0.07/44s + 1}{(1/44)^2 s^2 + 2*0.7/44s + 1} \tag{41}$$

This transfer function was then put into state space format with the following matrices.

$$A = \begin{bmatrix} -61.6 & -60.5 \\ 32 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 8 \\ 0 \end{bmatrix} \tag{42}$$

$$C = [-6.93 \quad 0]$$

$$D = 1$$

This leads to a single input, single output, two state system that can be put into simulink in the following way (see Figure 110).

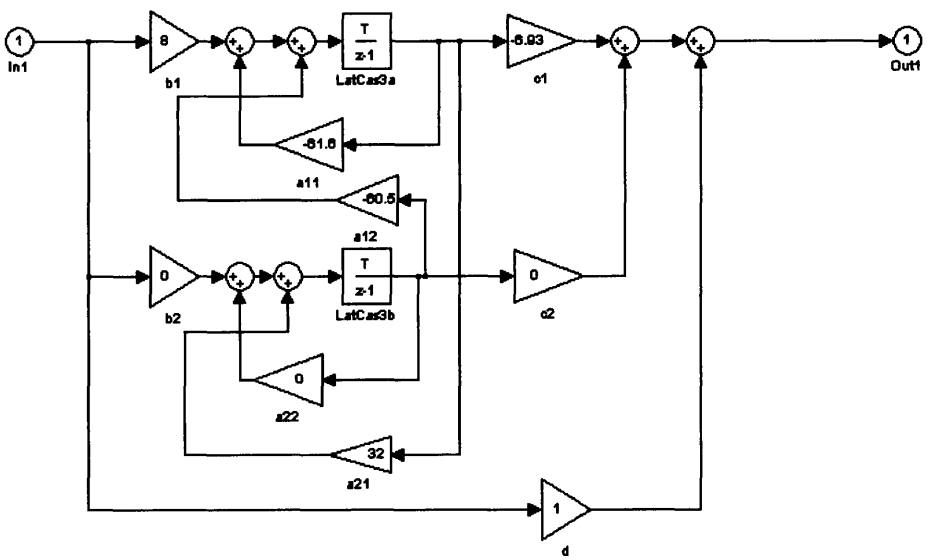


**Figure 110: Filter with Explicit Integrators**

The integrators labeled LatCas3a and LatCas3b can now be reset each time the propagation sequence starts. This leads to feedback from all 13 integrator states that are in the entire CAS systems.

Furthermore, certain delays had to be added to prevent algebraic loops from forming in the resulting code. Figure 111 shows the aircraft airframe simulation which further clarifies this technique.

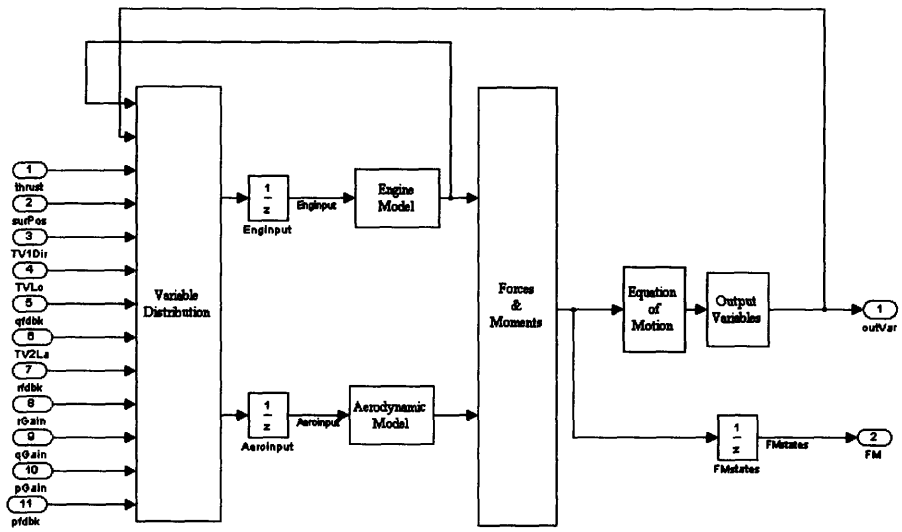


**Figure 111: Aircraft Airframe with Delays**

The vectors Enginput, Aeroinput and FMStates are all delayed in order to prevent these algebraic loops from forming. As a result all must be initialized by feedback from the truth model as well leading to 55 additional states.

Finally, S-functions which solve differential equations must be put into an alternate format in order for the integration to become resetable. This can be seen in the 6DOF equations of motion simulation below (see Figure 112).

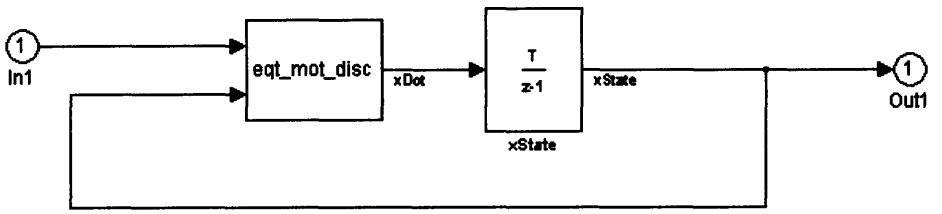


**Figure 112: S-Functions with Explicit Integrator**

The integration step is removed to the outside resulting in feedback to the original equations. This process leads to 14 integrator states that must be fedback to the propagator. The total feedback to initialize the propagator comes to 82 states. There is also however additional feedback from the control surface positions which is used to update the internal model.

One additional step taken in order to generate the C code was to add corresponding .tlc files for each S-Function. These files make autocoding S-Functions possible by incorporating the C function calls found in them into the resulting code.

# Appendix B

# MPC_PREP S Function

The MPC_PREP S function assembles the A, B, D, S, H, f matrices and the nominal input and output vectors. This was done using a linear algebra utility package developed in house. The utility sets up a matrix object and allows the user to easily transpose, multiply, scale and add matrices as well as a host of other things. This was used extensively as the proper matrices were stacked up and then integrated into the final H, f and A, B matrices. This can be seen in the following code which assembles the H and f matrices for a single input/output situation without input weighting.

```
void mpc_set_H_F(mpc *me) {

        /* H = S'*Q*S */
        Matrix_matrixMultiplication(&me->Q, &me->tmp.QS, &me->S);
        Matrix_transpose(&me->S, &me->tmp.S_Transp);
        Matrix_matrixMultiplication(&me->tmp.S_Transp, &me->H, &me->tmp.QS);

        /* F = -ETA'*Q*S */
        Matrix_transpose(&me->referenceOutput, &me->tmp.ETA_Transp);
        Matrix_matrixMultiplication(&me->tmp.ETA_Transp, &me->F, &me->tmp.QS);
        Matrix_negative(&me->F);

}
```

The insert function from this package was also used as vectors from the propagation sequence had to be assembled to form the matrices D and S. Furthermore, the identity and scaling functions were used to create the Q and R weighting matrices.

[This Page Intentionally Left Blank]

# Bibliography

[1] "Application of Multivariable Control Theory to Aircraft Control Laws," Wright Laboratory, WL-TR-96-3099, May 1996.

[2] Cotting, C., and Burken, J., "Reconfigurable Control Design for the Full X-33 Flight Envelope," NASA-TM-2001-210396. August 2001.

[3] Kassapakis E.G., and Warwick K., "Predictive Algorithm for Autopilot Design," Berkshire, UK, 1994.

[4] Lapp, Tiffany. "Guidance and Control using Model Predictive Control for Low Altitude Real Time Terrain Following Flight," MIT Master's Thesis, 2004, pp.56.

[5] Maciejowski, J.M., *Predictive Control with Constraints*, Pearson Education Limited, Essex, England, 2002.

[6] Military Standard, "Flying Qualities of Piloted Vehicles," MIL-STD-1797, 31 March 1987.

[7] Shertzer, R., Zimpfer, D., Brown, P., "Control Allocation for the Next Generation of Entry Vehicles." AIAA Paper 2002-4849, Aug. 2002.

[8] Van Den Boom, J., Backx, C., *Model Predictive Control*, Delft, The Netherlands, Nov. 2001.

[9] Ward, D, Monaco, J., "Development and Flight Testing of a Parameter Identification Algrithm for Reconfigurable Control," *Journal of Guidance, Control and Dynamics*, Vol. 21, No. 6, 1998, pp.948-956.