VALIDATION OF LINEARIZED FLIGHT MODELS USING AUTOMATED SYSTEM-IDENTIFICATION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Aerospace Engineering

by

Keith Eric Rothman

May 2009

© 2009
Keith Eric Rothman
ALL RIGHTS RESERVED

Committee Membership

TITLE:	Validation of Linearized Flight Models using Automated System-Identification
AUTHOR:	Keith Eric Rothman
DATE SUBMITTED:	May 2009
COMMITTEE CHAIR:	Dr. Daniel J. Biezad Aerospace Engineering Professor
COMMITTEE MEMBER:	Dr. Eric Mehiel Aerospace Engineering Professor and Co-Chair
COMMITTEE MEMBER:	Dr. Mark B. Tischler Army Aeroflightdynamics Directorate
COMMITTEE MEMBER:	Kenny K. Cheung UARC

Abstract

Validation of Linearized Flight Models using

Automated System-Identification

Keith Eric Rothman

Optimization based flight control design tools depend on automatic linearization tools, such as Simulink®'s LINMOD, to extract linear models. In order to ensure the usefulness and correctness of the generated linear model, this linearization must be accurate. So a method of independently verifying the linearized model is needed. This thesis covers the automation of a system identification tool, CIFER®, for use as a verification tool integrated with CONDUIT®, an optimization based design tool. Several test cases are built up to demonstrate the accuracy of the verification tool with respect to analytical results and matches with LINMOD. Several common nonlinearities are tested, comparing the results from CIFER and LINMOD, as well as analytical results where possible. The CIFER results show excellent agreement with analytical results. LINMOD treated most nonlinearity as a unit gain, but some nonlinearities linearized to a zero, causing the linearized model to omit that path. Although these effects are documented within Simulink, their presence may be missed by a user. The verification tool is successful in identifying these problems when present. A section is dedicated to the diagnosis of linearization errors, suggesting solutions where possible.

Acknowledgements

The author would like to give thanks to Dr. Daniel J. Biezad, Professor at Cal Poly, San Luis Obispo, CA, Dr. Mark B. Tischer, Flight Control Group Leader, Army Aeroflightdynamics Directorate, Ames Research Center, CA, Kenny K. Cheung, Developer, UARC, Ames Research Center, CA, and Dexter Hermstad, Developer, UARC, Ames Research Center, CA, for the patience and hard work they have given to make this thesis possible; Thanks to Mohammadreza Hossein Mansur and Christy Ivler for making sense of strange questions. To everyone that has played a part in making this thesis possible, I thank you.

Table of Contents

Table of Figures viii

Table of Tables xi

Nomenclature xii

I. Introduction 1

II. Background Information 2

A. Optimization based Flight Control Design 2

B. Validation 3

C. System Identification 8

1. CIFER 8

2. Describing functions and CIFER 8

3. CIFER Process 9

III. CIFER command line interface 13

A. Legacy command line interface 13

B. Object-based command line interface 14

1. Goals 14

2. FRESPID Classes 15

3. MISOSA and COMPOSITE Classes 23

4. CIFER Frequency Response Class 25

IV. Analyzing simulation using system identification 31

A. Automating the process 31

1. Obtain time histories from the Simulink diagram 31

2. Packaging the data 35

3. Creating a FRESPID case 35

4. Generating the frequency responses 40

B. Integration with CONDUIT 41

C. Limitations 47

V. Comparison of system identification results with theoretical predictions and LINMOD for a simple open loop elements 49

A. Validations 50

1. Baseline 50

2. Hysteresis 52
3. Saturation 54
4. Dead zone 57
5. Lookup Table 59
6. Time delay 61
7. Memory block and unit delay 64
8. Rate limiting 66
B. Table of results 70
VI. Analysis of simple feedback flight control systems 71
A. Closed loop validation for simple 2nd order system 71
B. XV-15 79
VII. Simplification and verification of a comprehensive flight control system 84
A. Block diagram simplification 84
B. Comparisons between the simplified block diagram and flight test data 90
VIII. Validation of linearized model of a simplified flight control system model 95
IX. Conclusion 101
X. Appendix A: 1st over 2nd plots 103
XI. Appendix B: Analysis script examples 104
1. Flight test analysis example 104
2. XV-15 batch validation example 106
XII. Appendix C: Sweep equation 107
XIII. Appendix D: Second order elements metrics 108
XIV. Works Cited 110

Table of Figures

Figure 1: Example system 4
Figure 2: Bandwidth example 6
Figure 3: Broken loop setup 6
Figure 4: Broken loop metrics 7
Figure 5: Old FRESPID CLI 16
Figure 6: New FRESPID CLI 17
Figure 7: Time history plotting 19
Figure 8: Crossover plot 26
Figure 9: Error plot example 28
Figure 10: Sweep frequency 33
Figure 11: Sweep amplitude 33
Figure 12: Completed sweep without noise (1-10 rad/sec) 34
Figure 13: Frequency point distribution 37
Figure 14: CONDUIT Broken Loop Switch 42
Figure 15: CONDUIT Broken Loop Switch Off 43
Figure 16: CONDUIT Broken Loop Switch On (Ref 7.) 43
Figure 17: Linearization Validation Tool 44
Figure 18: Estimate correction example 46
Figure 19: Simple 2nd order setup 49
Figure 20: Baseline validation 51
Figure 21: Hysteresis placement 52
Figure 22: Hysteresis 52
Figure 23: Open loop Hysteresis validation 53
Figure 24: Saturation 54
Figure 25: Open loop Saturation validation 55
Figure 26: Saturation parameters 56
Figure 27: Dead zone 57
Figure 28: Open loop dead zone validation 58
Figure 29: Lookup table 59

Figure 30: Open loop lookup table validation 60
Figure 31: Open loop time delay validation with non-zero Pade order 62
Figure 32: Open loop time delay validation with Pade order zero 63
Figure 33: Open loop memory validation 65
Figure 34: Rate limiting effects 66
Figure 35: Open loop rate limit validation 68
Figure 36: Time domain effect of rate limiting case 69
Figure 37: Step response of the good design 72
Figure 38: Step response of the bad design 72
Figure 39: Closed loop (good design) 72
Figure 40: Hysteresis closed loop (good design) 73
Figure 41: Hysteresis closed loop (bad design) 73
Figure 42: Hysteresis broken loop (good design) 73
Figure 43: Hysteresis broken loop (bad design) 73
Figure 44: Dead zone closed loop (good design) 74
Figure 45: Dead zone closed loop (bad design) 74
Figure 46: Dead zone broken loop (good design) 74
Figure 47: Dead zone broken loop (bad design) 74
Figure 48: Lookup table closed loop (good design) 75
Figure 49: Lookup table closed loop (bad design) 75
Figure 50: Lookup table broken loop (good design) 75
Figure 51: Lookup table broken loop (bad design) 75
Figure 52: Delay closed loop (good design) 76
Figure 53: XV-15 Block diagram 79
Figure 54: Lateral broken loop 80
Figure 55: Direction broken loop 80
Figure 56: Lateral closed loop 81
Figure 57: Directional closed loop 81
Figure 58: Broken loop setup 81
Figure 59: Unstable broken loop 83
Figure 60: Full-up vs. Simplified Broken Loop 87

Figure 61: Step response in Pitch 89
Figure 62: Step response in Roll 89
Figure 63: Step response in Yaw 89
Figure 64: Pitch broken loop 90
Figure 65: Pitch broken loop error 90
Figure 66: Roll broken loop 91
Figure 67: Roll broken loop error 91
Figure 68: Yaw broken loop 92
Figure 69: Yaw broken loop error 92
Figure 70: Corrected pitch broken loop 94
Figure 71: Corrected roll broken loop 94
Figure 72: Corrected yaw broken loop 94
Figure 73: Pitch broken loop 95
Figure 74: Pitch broken loop error 95
Figure 75: Roll broken loop 96
Figure 76: Roll broken loop error 96
Figure 77: Yaw broken loop 97
Figure 78: Yaw broken loop error 97
Figure 79: Memory blocks in path 98
Figure 80: Pade order zero 99
Figure 81: Corrected pitch broken loop 99
Figure 82: Corrected pitch broken loop error 99

Table of Tables

Table 1: Windowing parameters 36
Table 2: Percent change between CIFER and LINMOD in open loop 70
Table 3: Simple 1/2 closed loop summary 71
Table 4: Summary of Percent metric changes between CIFER and LINMOD 77
Table 5: Block Summary 86
Table 6: Stability margin changes between diagrams 88
Table 7: Flight test margin summary 93
Table 8: Summary of found time delays 97

Nomenclature

A	=	Amplitude of oscillation
A_n	=	Fourier coefficients
AIL	=	Aileron control
ADS	=	Aeronautical Design Standards
AFDD	=	Army Aeroflightdynamics Directorate
B	=	Hysteresis dead band width
B_n	=	Fourier coefficients
bl_in	=	Broken loop input
bl_out	=	Broken loop output
$\hat{\gamma}^2_{xy}(f)$	=	Coherence function
C_1	=	Sweep constant 1
C_2	=	Sweep constant 2
CIFER®	=	Comprehensive Identification from FrEquency Responses
ciffrq	=	CIFER frequency response object
coh	=	Coherence
COMPOSITE	=	"CIFER program to combine multiple windows to achieve a final frequency response" (Ref 2.)
composite_obj	=	COMPOSITE case interface object
CONDUIT®	=	Control Designer's Unified Interface
CZT	=	Chirp-Z transform, zoom transform
δ	=	Actuator output
δ_{com}	=	Actuator command
δ_{noise}	=	Frequency sweep noise
δ_{pilot}	=	Pilot input
δ_{sweep}	=	Frequency sweep
D	=	Dead zone width
dB	=	Decibel
deg	=	Degrees
DERIVID	=	"CIFER program used to identify a state-space model" (Ref 2.)
DFT	=	Discrete Fourier Transform
dt	=	Time history step size (seconds)
e	=	Error signal
f	=	Feedback signal
f_{filter}	=	Filter frequency (Hz)
f_{sample}	=	Sample frequency (Hz)
FFT	=	Fast Fourier Transform
FRESPID	=	"CIFER program that calculates SISO frequency responses using a chirp z-transform" (Ref 2.)
frespid_obj	=	FRESPID case interface object
$\tilde{G}_{xx}(f)$	=	Rough input auto-spectrum
$\hat{G}_{xx}(f)$	=	Smooth input auto-spectrum
$\tilde{G}_{yy}(f)$	=	Rough output auto-spectrum

$\hat{G}_{yy}(f)$	=	Smooth output auto-spectrum
$\tilde{G}_{xy}(f)$	=	Rough cross spectrum
$\hat{G}_{xy}(f)$	=	Smooth cross spectrum
GUI	=	Graphical user interface
$\hat{H}_1$	=	Input frequency response estimate
$\hat{H}_2$	=	Output frequency response estimate
k_1	=	Lookup table inner slope
k_2	=	Lookup table outer slope
lat	=	Lateral control
LINMOD	=	Simulink's block diagram linearization function
LTI	=	Linear time invariant
mag	=	Magnitude (dB)
MATLAB®	=	MATrix LABoratory from The MathWorks™
MIMO	=	Multi-input/Multi-output
MISOSA	=	"CIFER program that determines frequency responses when multiple inputs are present" (Ref 2.)
misosa_obj	=	MISOSA case interface object
MUAD	=	Maximum unnoticeable added dynamics
NAVFIT	=	"CIFER program used to identify a transfer-function model" (Ref 2.)
p	=	Roll rate (radian/second)
ped	=	Pedal input
PIO	=	Pilot induced oscillation
r	=	Yaw rate (radian/second)
R	=	Rate limit
rad	=	Radian
S	=	Saturation limit
SAS	=	Stability Augmentation System
sec	=	Second
Simulink®	=	Block diagram modeling software from The MathWorks™
τ	=	Time constant (seconds)
SISO	=	Single-input/Single-output
$\theta(t)$	=	Sine argument (radians)
TIC	=	Theil inequality coefficient
$T_{\max}$	=	Maximum chosen window size (second)
$T_{\max_{\lim it}}$	=	Maximum allowable window size (second)
$T_{\min}$	=	Minimum chosen window size (second)
T_{rec}	=	Record length (second)
t_{fadein}	=	Frequency sweep fade-in length (second)
$t_{fadeout}$	=	Frequency sweep fade-out length (second)
thdt	=	Time history step size (second)
t_{park}	=	Amount of time spent at low frequency during sweep (second)
t_{trim}	=	Amount of time sweep held at zero for trim (second)
t_{zero}	=	Amount of time sweep held at zero (second)
VERIFY	=	"CIFER program (state-space model verification) used to check the time-domain

		predict accuracy of an identified model” (Ref 2.)
y	=	Output
$\hat{y}$	=	Measured output
W	=	Lookup table width
ω	=	Sine frequency (radian/second)
$\omega_{\lim}$	=	Rate limiting limit frequency (radian/second)
$\omega_{\max}$	=	Maximum frequency of interest (radian/second)
$\omega_{\max\ \mathrm{limit}}$	=	Maximum allowable frequency of interest (radian/second)
$\omega_{\min}$	=	Minimum frequency of interest (radian/second)
$\omega_{\min_{\lim it}}$	=	Minimum allowable frequency of interest (radian/second)
ω_{onset}	=	Rate limiting on-set frequency (radian/second)

I. Introduction

Modern flight control simulation diagrams are complicated to the point where automated linearization tools are important for efficient analysis. These tools are being integrated into smarter analysis tools and optimizers. However, the validation of such linearization tools is lacking.

This thesis utilizes a frequency domain system identification method, CIFER® (Ref 2.), to provide a truth model for the validation of linearization tools. The need and development of a more robust and flexible command line interface to CIFER for automated system identification is discussed. One integration method with Simulink® block diagrams is explained and implemented in a validation tool.

The validation tool is for an integrated control design suite called CONDUIT® (Ref 11.). CONDUIT uses the linearization method LINMOD, a linearization method for Simulink diagrams. In this thesis several validations are performed against examples of increasing complexity to demonstrate the validation tool's ability to provide a truth model and some potential problems with the LINMOD linearization.

II. Background Information

A. Optimization based Flight Control Design

With the increased power of flight computer systems, some modern flight control systems can now be complicated enough that manual gain selection is impractical. Optimization based control design is a method that uses computers to choose gain sets given a set of objectives and constraints. This thesis specifically focuses on CONDUIT, a program that uses this approach to control design (Ref 11.).

CONDUIT is a control design program that uses Simulink diagrams to express the system plant and control system. CONDUIT drives the optimization using results from the Simulink diagram.

CONDUIT utilizes time-domain and frequency-domain metrics to drive the optimization. For time-domain metrics, CONDUIT uses Simulink to perform a simulation given input time signals and it returns the output time responses. Simulink's simulation accurately accounts for all blocks present in the diagram, such as hysteresis or S-Functions. This can be used for time domain metrics based on step responses and impulses or looking for time-domain performance specifications, such as rise time.

For frequency-domain metrics, CONDUIT uses the functionality provided by the MATLAB® Control System Toolbox™ to extract a linearized state-space model from a Simulink diagram. In the linearization, some blocks have exact analytical forms, such as transfer functions or gains. However some blocks will not have exact analytical forms, so the linearization process will either perturb the block or it will use a representation chosen by the user. Given this state-space system, frequency responses based on

particular input-output pairs can be generated. CONDUIT then calculates frequency domain performance metrics such as gain and phase margin, crossover or bandwidth.

For piloted aircraft, CONDUIT uses handling quality specifications based on the appropriate literature (ADS 33 (Ref 13.), 9490 (Ref 12.), etc.) as constraints and objectives. For each of these specifications, a numerical score is assigned based on its ability to meet the specification. The optimizer uses a gradient-based algorithm to search for a new set of gains that satisfies the constraints and improves the worst score. CONDUIT repeats the process until it satisfies all constraints and has reached an optimized solution. The gains that CONDUIT generates are then based entirely on the quality of the time responses and linear models generated from the Simulink diagram. If the time responses or linear models from the diagram do not adequately match the aircraft response as measured in flight, then the gains will not generate the expected performance, requiring further refinement, or worse CONDUIT is not optimizing on a representative model of the aircraft.

B. Validation

In order to prevent mismatch between the aircraft and the model as represented in the Simulink diagram, the diagram must be validated. There are two types of validation that need to be performed. The first is validation between the aircraft and the Simulink diagram. The second is a validation between the Simulink diagram and the linearized state-space model extracted from the diagram.

The validation between the aircraft and the diagram requires flight testing to gather the data for analysis. For this thesis, a frequency domain-based validation procedure will be discussed that compares the frequency response attained from flight

test data with those attained from the simulation. The process of generating frequency responses from the time history data will be discussed later. This validation ensures the simulation is representative of the aircraft. Errors shown in this validation would indicate a modeling deficiency.

The second class of validation is between the Simulink simulation of the diagram and the generated linearized state-space model extracted from the diagram. This ensures that the dynamic response of the linearized model is an accurate characterization of the complete block diagram. This validation is between frequency responses data from the simulation and the linearized state-space model. Any differences between the frequency responses from the time history data and the linearized model are discrepancies in the linearization processes. The validation between the Simulink diagram and the linearized model of the simulation will be the focus of this thesis.

In both classes of validation there are three different frequency response pairs that should be checked. Each response will check characteristics of the model, so all three need to be done. The frequency response only needs to match in the frequency range of interest, which is near crossover. The rule of thumb for frequency range of interest is one third to three times the crossover frequency (Ref 10.).

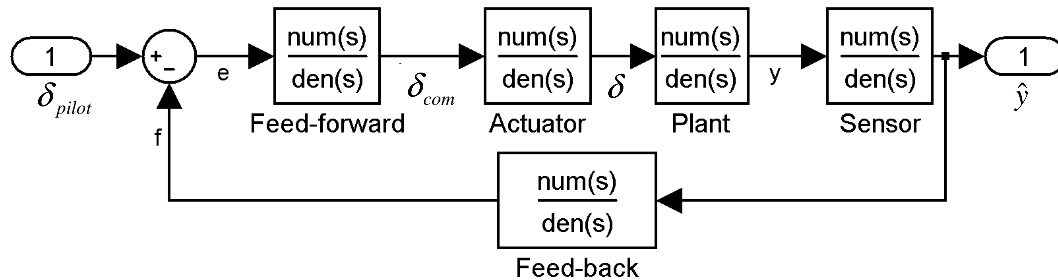

Figure 1: Example system

The first set of response pairs to check is the bare airframe responses. This is from either before or after the actuators to sensor output. In the example system shown

above, the bare airframe is either $\frac{\hat{y}}{\delta}$ or $\frac{\hat{y}}{\delta_{com}}$ depending on where the actuator sensing is located. These responses characterize the core behaviors of the aircraft in the absence of augmentation. The accuracy of these responses will determine the accuracy of the modeling of the bare airframe. If improvements in agreement between the aircraft and the Simulink diagram are needed, system identification techniques can be used to directly generate a model to match flight test data. When performing system identification to obtain a model, the match quality between the simulation and the aircraft is only a function of the identification method and flight data quality. When using physics-based modeling to generate the simulation plant, errors that show up in the validation will need to be tracked down in the physics-based modeling code.

The second set of frequency responses to check is the closed loop response pairs. This is from pilot stick to sensor output. In the system shown in the figure above, the closed loop pair is $\frac{\hat{y}}{\delta_{pilot}}$. These pairs will be used to determine bandwidth, and will be ultimately what the pilot experiences. Bandwidth is typically defined as the frequency where the magnitude drops 3 dB below the steady state value. The figure below shows a second order system and its bandwidth. Bandwidth represents the maximum frequency that the system can pass through and can be considered the speed of response of the system. However, just matching the closed loop response pairs is insufficient to validate the simulation because the loop closures will wash out the key features of the feedback system. For that reason, bare airframe and broken loop responses must also be checked.

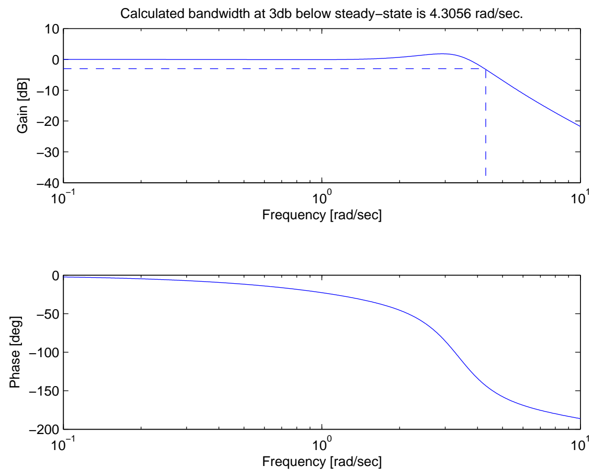

Figure 2: Bandwidth example

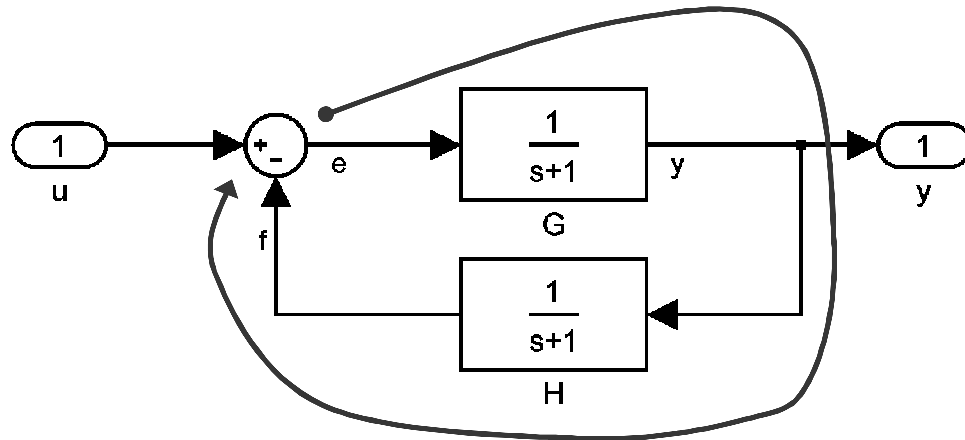

Figure 3: Broken loop setup

The last set of response pairs to check is the broken loop responses. This is from error to feedback $\left(\frac{f}{e}\right)$. This will capture all the important characteristics that determine closed loop behavior. It will include the characteristics of the bare airframe as well as time-delay or dynamics in the feedback system, such as computational delays. Errors in the broken loop responses can be from a large number of sources, including the bare airframe response. For this reason, conduct bare airframe validation before the broken loop will help eliminate math modeling errors. Ensuring the broken loop responses

match will provide good confidence that the CONDUIT analysis results are good representations of the aircraft.

The broken loop is used to generate gain margin, phase margin, and crossover frequency. Crossover frequency is the point where the system crosses 0 dB. Phase margin is the phase above -180 degrees at the crossover frequency. Gain margin is the magnitude in dB below 0 at the -180 degree phase crossing. These points are illustrated below. Crossover frequency is used as a prediction of closed loop bandwidth, and therefore system speed. Gain and phase margins are measures of stability and robustness against unknown plant changes. Have too little margin means unknown variation may cause the closed loop system to go unstable. Phase margin is also a measure of closed loop damping ratio. Having too little phase margin will cause many overshoots. Having too much phase margin may cause a first order response, which is slower than a well damped second order response. Pilots tend to expect second order responses in systems.

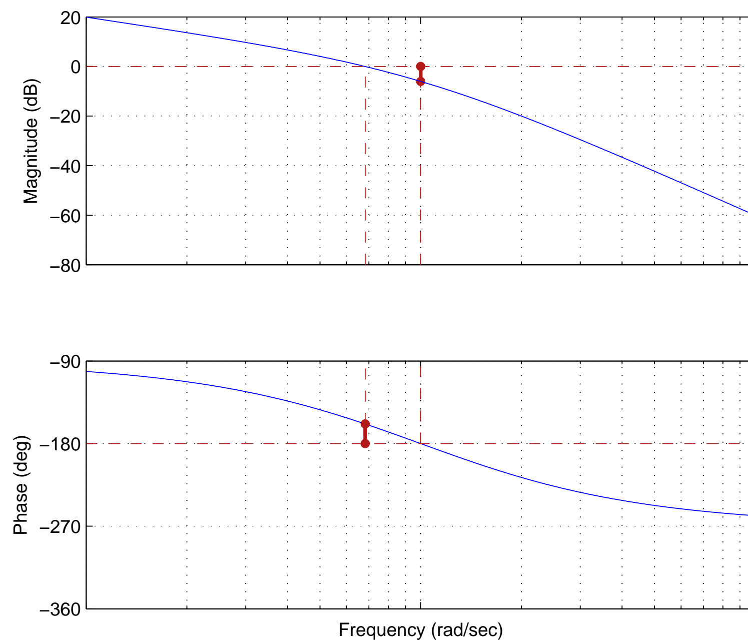

Figure 4: Broken loop metrics

C. System Identification

1. CIFER

The generation of frequency responses for validation from time history data can be achieved using a frequency domain system identification method. This thesis uses CIFER (Ref 2.) to perform frequency domain system identification. CIFER provides a set of tools that will filter time history data, perform a discrete Fourier transform on the time history data, and generate a frequency response. CIFER utilizes a multi-window averaging technique to get good resolution over the frequency range of interest and minimize noise. It has been used on many flight programs with success, proving it to be a reliable system identification tool (e.g. UH-60M (Ref 8.), XV-15 (Ref 9.), RMAX UAV (Ref 2.)). It also provides the ability for both parametric SISO and MIMO modeling, depending on the desired goal.

2. Describing functions and CIFER

Describing functions (DF) are methods for linearizing nonlinearities. The most common describing function class and the one that is going to be used through-out this thesis is one that assumes a sinusoidal input. The output is not necessarily a sinusoid; however we can approximate the output utilizing a Fourier series (Ref 3.):

$$
\begin{aligned}
&x(t) = A\,sin(\omega t) \\
&y(t) = f(x(t)) \cong A_0 + \sum_{n=0}^{\infty}(A_n\,cos(n\omega t) + B_n\,sin(n\omega t)) \\
&where \\
&A_n = \frac{1}{\pi}\int_0^{2\pi} y(t)cos(n\omega t)d(\omega t) \\
&B_n = \frac{1}{\pi}\int_0^{2\pi} y(t)sin(n\omega t)d(\omega t)
\end{aligned}
\quad \textbf{(1)}
$$

Describing functions only use the fundamental harmonic (i.e. drop terms n>1) and assumes a symmetric function $(A_0 = 0)$:

$$
N(A,\omega) = \frac{\sqrt{A_1^2 + B_1^2}}{A} \angle arctan\left(\frac{A_1}{B_1}\right) \quad \textbf{(2)}
$$

This form can be amplitude and frequency dependent. There are methods utilizing the describing functions for nonlinearities that can predict limit cycles that could arise due to the presence of nonlinear elements in feedback system.

CIFER utilizes a particular Fast Fourier transforms (FFT) to generate frequency responses. Effectively CIFER is computing a numerical describing function of the input to output response. This means CIFER can generate a frequency response from records with the presence of nonlinear elements, such as hysteresis. Active nonlinearities will be shown during validations and their effect on the system.

3. CIFER Process

Generation of frequency responses based on time history data is performed in three programs within CIFER. The three programs are FRESPID, MISOSA and COMPOSITE. FRESPID takes the time history data and generates frequency responses for each input-output pair utilizing a specialized DFT. MISOSA performs multi-input conditioning, and COMPOSITE performs multi-window averaging. Each of these

programs has a batch program that performs the actual calculations after the input data has been provided a case.

FRESPID is the most input intensive of these three programs. FRESPID takes a time history file and associates each channel of data to an input or output. The data in the channels can be filtered and decimated or interpolated. FRESPID also must remove bias and slope from the time history channels to meet the mathematical requirements of the transform used. FRESPID utilizes a specialized fast Fourier transform (FFT) called the chirp-Z transform (CZT) or zoom transform. The CZT has more flexibility than the FFT and this flexibility is controlled through window settings. The window settings are the number of input points, the number of output points, minimum and maximum frequency, and decimation ratio. Control over the minimum and maximum frequency allows a user to generate points only in the frequency range of interest and thereby increase frequency resolution in that range. FRESPID supports generation of up to five windows. This means that for each input-output pair, up to five frequency responses can be generated. The meaning and selection of the windows will be covered in detail later.

The responses generated from the CZT are not immediately frequency responses; the CZT generates are the Fourier coefficients of the input $(X(f))$ and the output $(Y(f))$. These are complex valued arrays for each frequency point the CZT generated. To get to a frequency response, first three spectral functions must be generating from the Fourier coefficients, rough input autospectrum, the rough output autospectrum and the rough cross spectrum (Ref 2.).

$$\tilde{G}_{xx}(f) = \frac{2}{T_{win}} |X(f)|^2 \quad \textbf{(3)}$$

$$\tilde{G}_{yy}(f) = \frac{2}{T_{win}} |Y(f)|^2 \quad \textbf{(4)}$$

$$\tilde{G}_{xy}(f) = \frac{2}{T_{win}} \left[X^*(f)Y(f)\right] \quad \textbf{(5)}$$

These spectral functions will have errors, some of a deterministic nature and some of a non-deterministic (random) nature. If the number of input time history points to the CZT is less than the size of the time history record, multiple transforms can be taken into the data. Each CZT can then be averaged together. The random errors ideally average to zero, resulting in smooth estimates of the spectral functions (Ref 2.).

$$\hat{G}_{xx} = \left(\frac{1}{Un_r}\right)\sum_{k=1}^{n_r} \tilde{G}_{xx,k}(f) \quad \textbf{(6)}$$

$$\hat{G}_{yy} = \left(\frac{1}{Un_r}\right)\sum_{k=1}^{n_r} \tilde{G}_{yy,k}(f) \quad \textbf{(7)}$$

$$\hat{G}_{xy} = \left(\frac{1}{Un_r}\right)\sum_{k=1}^{n_r} \tilde{G}_{xy,k}(f) \quad \textbf{(8)}$$

From the smooth estimates, two frequency response estimates can be made, $\hat{H}_1$ and $\hat{H}_2$. The equations for these estimates are shown below (Ref 2.). These two estimates will only be equal in the absence of noise processes and nonlinear elements.

$$\hat{H}_1 = \frac{\hat{G}_{xy}(f)}{\hat{G}_{xx}(f)} \quad \textbf{(9)}$$

$$\hat{H}_2 = \frac{\hat{G}_{yy}(f)}{\hat{G}_{yx}(f)} \quad \textbf{(10)}$$

From the two frequencies response estimates the coherence function can be defined. The equation is shown below. The coherence is a representation of the linearity of the response. When the coherence is one the output is a linear function of the input.

The coherence function is essential in the CIFER system identification process because of its ability to indicate the quality of the response. In COMPOSITE and parametric analyses, coherence is used to weight responses.

$$\hat{\gamma}_{xy}^{2}(f)=\left|\frac{\hat{H}_1}{\hat{H}_2}\right|=\frac{\left|\hat{G}_{xy}(f)\right|\left|\hat{G}_{yx}(f)\right|}{\left|\hat{G}_{xx}(f)\right|\left|\hat{G}_{yy}(f)\right|}=\frac{\left|\hat{G}_{xy}(f)\right|^{2}}{\left|\hat{G}_{xx}(f)\right|\left|\hat{G}_{yy}(f)\right|} \quad \textbf{(11)}$$

When examining a MIMO system, the resulting frequency responses generated from FRESPID will detect output response due to inputs in all channels, not just due to the primary input. MISOSA utilizes the cross-correlation transfer functions between inputs to remove output frequency content due to off-axis inputs. The result is ideally a transfer function that corresponds to the SISO input-output relationship, in the absence of other inputs (Ref 2.).

COMPOSITE takes up to five windowed frequency responses and generates a single response by doing a weighted average of the frequency response points based on the random error and coherence function (Ref 2.). This new response has the best characteristics of each of the windows. The resulting response is now ready for further analysis.

At this point in the process, the user is free to conduct several analyses in CIFER. There are several non-parametric analyses available, such as computing stability margins, crossover frequency, or bandwidth. The user can plot or export the frequency responses as well. If the user wants a parametric model, two options are available. If only a SISO transfer function is required, NAVFIT can be used to fit a transfer function to the frequency response. If working with a MIMO system, DERIVID and VERIFY provide the ability to construct and validate a state-space representation.

III. CIFER command line interface

A. Legacy command line interface

Before the work of this thesis, sweeps of simulation diagrams were done manually within AFDD. The user would add a sweep block in Simulink to generate the sweep, add data recorders, etc. The user would then need to manually enter data into the CIFER program. The primary goal of this thesis is to streamline and automate this process, and demonstrate results based on several case studies. In order to automate this process, the CIFER system identification process must be scriptable. However, the CIFER program was originally developed with only an interactive mode of operation.

The nature of processing flight test data is slow and requires user knowledge about the parameters that drive the identification. This means ease of use was of primary importance when CIFER was originally designed. For this reason, CIFER utilizes a GUI to input data for processing. Automation was not done through the GUI, since it tends to be fragile, requiring updates every time the GUI changes.

Work to script the CIFER tools was started in an earlier Cal Poly San Luis Obispo Master Thesis (Ref 6.) wherein the author proposed and demonstrated porting CIFER from a text-based environment into a MATLAB based GUI environment with options for scripting provided by command line functions (Ref 6.). Rupnik developed MATLAB interfaces to the underlying CIFER methods, providing the first command line interface. This command line interface provided access to some of the functionality in CIFER through functions called with name-value pairs. In particular he provided interfaces into FRESPID, MISOSA and COMPOSITE, the three main utilities used in transforming time history data into frequency responses.

However, to automate the system identification process, the command line interface used needed to be robust, easy to use, internally consistent, well encapsulated, and extensible. The command line interface created by Rupnik does not meet these requirements. As a result a new command line interface was developed based on MATLAB classes.

B. Object-based command line interface

1. Goals

The primary goal of the new command line interface is consistency. At no point should the user be utilizing a data structure that is not internally consistent. This means at all times users are able to save the case and open it in the GUI and make changes.

This goal also means consistency with MATLAB error handling and calling conventions. The use of the exception handling allows stack traces and “stop on error” features to be used. The calling convention is now more like other toolboxes such as the Control System Toolbox. This allows better interfaces to loading, saving, batch and plotting. The functionality is exposed through the data structure fields similar to handle graphics. Lastly the new command line interface has more descriptive names for fields to reduce the need for excessive lookup from the documentation.

The new command line interface consists of 17 classes, each with member functions providing addition features as needed. The total source lines of code are 12415, across 422 files. The following sections will document some of the features and changes that went into the new command line interface.

2. FRESPID Classes

The legacy command line interface contained all the FRESPID case information in one structure, resulting in 44 fields at the top level, covering control, output and time history definitions, and windowing parameters, as well as other assorted options. Under the new object-based design, FRESPID now consists of three classes: frespid_obj, thfile and windows. The frespid_obj class contains thfile and windows objects. The frespid_obj deals with loading and saving to the database, assigning control and output channels, and executing FRESPID jobs. Time history and windowing parameters became their own individual classes, and the control and output definitions were folded into a structure. The end result was only 22 fields at the top level. Below are printouts from MATLAB of the old and new FRESPID structures. Notice that the new interface is much more concise.

```
fre = frespid('XVLATSWP',1)

fre =

         casename: 'XVLATSWP'
         comments: 'LATERAL FR SWP FOR XV15 HOVER'
         controls: {10x1 cell}
          outputs: {20x1 cell}
          caseout: 'XVLATSWP'
         crosscor: 'Y'
          savfile: 'N'
         froutdir: 'C:\CIFER_Pro\jobs\tfdata'
            savdb: 'Y'
             plot: 'N'
          evntnum: [883 884 0 0 0 0 0 0 0 0]
         flghtnum: [150 150 0 0 0 0 0 0 0 0]
          strttim: [0 0 0 0 0 0 0 0 0 0]
          stoptim: [0 0 0 0 0 0 0 0 0 0]
           source: 5
             thdt: 0.0040
         biasflag: 'Y'
           thfile: {10x1 cell}
          conchnl: {10x5 cell}
          conunit: {10x1 cell}
         conscfac: [10x5 double]
          outchnl: {20x5 cell}
          outunit: {20x1 cell}
         outscfac: [20x5 double]
            frall: 'N'
           frcalc: {20x10 cell}
          freqcut: 5.0266
          dtfinal: 0.0400
     conditioning: [2x10 double]
         condunit: {'Hz'  ''  ''  ''  ''  ''  ''  ''  ''  ''}
         savconth: 'N'
            winon: {'*'  '*'  '*'  '*'  '*'}
            winid: {'45 sec'  '35 sec'  '30 sec'  '20 sec'  '15 sec'}
           winlen: [45 35 30 20 15]
          wininpt: [1125 875 750 500 375]
         winoutpt: [923 149 274 524 137]
           windec: [1 1 1 1 1]
           minfft: [0.1396 0.1795 0.2094 0.3142 0.4189]
           maxfft: [12 12 12 12 12]
          plotopt: [12x1 double]
          plotdev: 'P'
             grid: 'Y'
          lrgplot: 'Y'
          plotdec: 'Y'
               ec: [1x1 struct]
```

Figure 5: Old FRESPID CLI

```
>> fre = frespid_obj('XVLATSWP')

fre =

                        name: 'XVLATSWP'
                    comments: 'LATERAL FR SWP FOR XV15 HOVER'
                     caseout: 'XVLATSWP'
                      db_out: 1
                    crosscor: 1
                 fr_file_out: 0
              fr_file_format: 'CIFER'
                 fr_file_dir: 'C:\CIFER_Pro\jobs\tfdata'
    th_file_out_unformatted: 0
          th_file_out_ascii: 0
                    controls: {'AIL'  'RUD'}
                     outputs: {'P'  'R'  'AY'  'VDOT'  'PHI'}
                     windows: [1x5 windows]
                     thfiles: [1x2 thfile]
                      frcalc: [5x2 logical]
                   gen_plots: 0
                       plots: [0 0 1 0 0 0 0 0 0 0 0 0]
                  heavy_grid: 1
                  large_plot: 1
               decimate_data: 1
                 plot_format: 'PostScript'
                     frnames: {5x2x5 cell}
```

Figure 6: New FRESPID CLI

By breaking the time history file into its own class, all time history error checking was removed from frespid_obj. The result is that thfile as a stand-alone class can be used to read, write, and plot time history files in a simple manner. This also means by the time the user is done setting up the thfile object, it has already checked the existence of the time history file before it is assigned within the frespid_obj.

In the example below, a CIFER Text time history file has been loaded using the thfile class. The time history step size (`"thdt"`) and `"channels"` fields are dynamic fields generated by examining the time history file. These fields are internally fairly cumbersome function calls, but with the new class system they are just another field in the structure.

```
>> thfil = thfile;
>> thfil.source = 5;
>> thfil.filename = 'Flt150_Event883.dat';
>> thfil.event = 883;
>> thfil.flight = 150;
>> thfil

thfil =

           source: 5
            event: 883
           flight: 150
       start_time: 0
        stop_time: 0
         filename: 'Flt150_Event883.dat'
             thdt: 0.0040
         channels: {'A300'  'D009'  'D022'  'D024'  'D284'  'D645'
'V012'  'V014'  'VDOT'}
     desired_rate: 0
    filter_cutoff: 0
          finaldt: 0.0040
```

FRESPID supports concatenating multiple time history files together, so the thfile object uses standard MATLAB concatenation syntax to represent this behavior. However, all the records must come from the same source and must undergo the same conditioning. For this reason, the thfile object enforces these restrictions when users create an array of thfile objects. If the user plots the concatenated object, the plot is of the concatenated record.

```
plot(thfil,'A300');
```

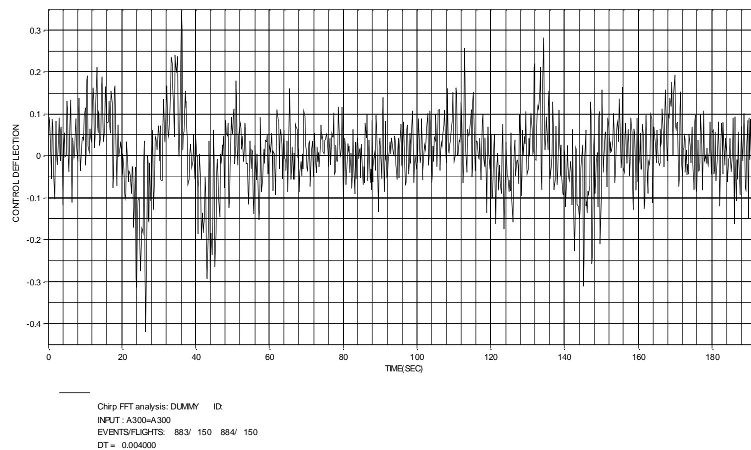

Figure 7: Time history plotting

Choosing windowing parameters has several constraints based on the time history step size. Because of the complicated nature of this process, windowing parameters were split into their own class that deals solely with window configuration. Depending on which field the user changes, the class behavior will also change dynamically. For example, changing the "`length`" or "`input_points`" fields will result in a recomputation of the window parameters. If the user changes the other fields it will only check that the new value is valid.

The code below shows how the legacy and new command line interfaces would change the window length and submit a batch job. Notice how the new command line interface splits the tasks into several smaller lines. Each of these lines has its own error checking. In the legacy command line interface, the second argument being a "`3`" indicates the batch job will be submitted after modifying the data structure. Notice the separate batch job call in the new command line interface, clarifying and separating the different actions. The legacy command line interface would also not return user control until the batch job was done. The new command line interface does not require waiting

for batch job completion. The "wait" call replicates the waiting that happens in the legacy command line interface, but it is not required.

```
frespid(fre,3,...
    'winlen',[25,20,15,10,5],...
    'maxfft',[30,30,30,30,30],...
    'wininpt',[0,0,0,0,0],...
    'winoutpt',[0,0,0,0,0],...
    'windec',[0,0,0,0,0],...
    'minfft',[0,0,0,0,0]);
```

```
win = fre.windows;
[win.length] = ...
     deal(25,20,15,10,5);
[win.max_freq] = deal(30);
fre.windows = win;
job = batch(fre);
wait(job);
```

Because FRESPID, MISOSA, and COMPOSITE cases are executed in a batch manner, the batch_job class was created to handle monitoring submitted jobs. It provides a way for the user to check if a job is done, wait until a job is done, and get locations of log files and plots.

With these components all combined, frespid_obj provides a complete interface into FRESPID, with better error handling, syntax, and consistency. Below is an example showing how a FRESPID case is defined and run between the legacy and the new command line interfaces. The new command line interface does take more lines than the old interface; however the additional lines add clarity to the actions taken. They will be explained in more detail later.

```
% Assign a blank frespid structure
f_in = frespid;
thename = 'XVLATSWP';

% Fill in all the necessary
information to make the case
f_in.casename = thename;
f_in.caseout  = thename;
f_in.crosscor = 'Y';
f_in.plot     = 'N';

% Time history selection
parameters:
f_in.source          = 5;
f_in.evntnum(1:2)  = [883,884];
f_in.flghtnum(1:2) = [150,150];
```

```
% Assign a blank frespid structure
fre = frespid_obj;




fre.name = 'XVLATSWP';
% In the new command line interface,
the caseout field is automatically
% set to the case name, as it is in
% the gui

% Set which time history file to use
fre.thfiles.source = 5;
fre.thfiles(1).filename = ...
   'Flt150_Event883.dat';
```

<pre>f_in.thfile(1:2) =
  {'Flt150_Event883.dat', ...
   'Flt150_Event884.dat'}</pre>
	<pre>fre.thfiles(1).event = 883;
	fre.thfiles(1).flight = 150;
	
	fre.thfiles(2).filename = ...
	   'Flt150_Event884.dat';
	fre.thfiles(2).event = 884;
	fre.thfiles(2).flight = 150;</pre>

<pre>% channel definition parameters:
f_in.controls(1:2) = {'AIL','RUD'};
f_in.outputs(1:4) = ...
    {'P','R','AY','VDOT' 'PHI'};</pre>
	<pre>% Set the control and output names
	fre.controls = {'AIL' 'RUD'};
	fre.outputs = {'P' 'R' 'AY' ...
	    'VDOT' 'PHI'};</pre>

<pre>f_in.conunit(1:2) = {'deg','deg'};
f_in.conchnl(1:2,1) = ...
    {'D645','D284'};
f_in.outunit(1:4) =
    {'rad/s', 'rad/s', ...
     'ft/sec2', 'ft/sec2'};
f_in.outchnl(1:4,1) = ...
    {'V012','V014','A300', ...
     'VDOT', 'D009'};
f_in.outscfac(1:4,1) = ...
    [0.0175,0.0175,32.174, ...
     0.0175,0.0175];</pre>
	<pre>% Associate control names with time
	history channels
	fre.controls('AIL').units = 'deg';
	fre.controls('AIL').channel = ...
	    'D645';
	
	fre.controls('RUD').units = 'deg';
	fre.controls('RUD').channel = ...
	    'D284';
	
	fre.outputs('P').units = 'rad/s';
	fre.outputs('P').channel = 'V012';
	fre.outputs('P').scale = 0.0175;
	
	fre.outputs('R').units = 'rad/s';
	fre.outputs('R').channel = 'V014';
	fre.outputs('R').scale = 0.0175;
	
	fre.outputs('AY').units = 'ft/sec2';
	fre.outputs('AY').channel = 'A300';
	fre.outputs('AY').scale = 32.174;
	
	% The channel defaults to the input
	name if
	% it exists in the time history
	fre.outputs('VDOT').units = ...
	    'ft/sec2';
	
	fre.outputs('PHI').units = 'RAD/S';
	fre.outputs('PHI').channel = 'D009';
	fre.outputs('PHI').scale = 0.0175;</pre>

<pre>% Frequency response selection
parameters
f_in.frcalc(1:4,1:2) = {'*'};</pre>
	<pre>% Logicals are used instead of cell
	% array of strings for flags
	fre.frcalc(:) = true;</pre>

<pre>% Conditioning parameters
f_in.conditioning(1,1:2) = [3,2];
f_in.conditioning(2,1:2) = [4,25];</pre>
	<pre>fre.thfiles.filter_cutoff = 4;
	fre.thfiles.desired_rate = 25;</pre>

<pre>% Window Parameters
f_in.winid   = {
    '45 sec'</pre>
	<pre>fre.windows(1).comments = '45 sec';
	fre.windows(1).length = 45;</pre>

'35 sec'
'30 sec'
'20 sec'
'15 sec'};
f_in.winlen = [45,35,30,20,15];
	fre.windows(2).comments = '35 sec';
	fre.windows(2).length = 35;
	fre.windows(3).comments = '30 sec';
	fre.windows(3).length = 30;
	fre.windows(4).comments = '20 sec';
	fre.windows(4).length = 20;
	fre.windows(5).comments = '15 sec';
	fre.windows(5).length = 15;

f_in.winon(1:5) = {'*'};
f_in.maxfft(:) = 12;
	for i = 1:5
	    fre.windows(i).on = true;
	    fre.windows(i).max_freq = 12;
	end

% Save the structure into the database frespid(f_in,2);	save(fre);

The channels assignment is one of the major changes between the two command line interfaces. Below is a line taken from the example above. In the legacy command line interface "conchnl" is the association between control names and channels. The row "1:2" is indicates "select the first and second channel", and the column "1" indicates the first of five channels for each control. In the new command line interface the particular channel is identified by name 'AIL', and the first channel is just the "channel" field. The association between the control and the channel is much clearer in the new command line interface.

f_in.conchnl(1:2,1) = ... {'D645','D284'};	fre.controls('AIL').channel = 'D645'; fre.controls('RUD').channel = 'D284';

In the legacy command line interface, the time history conditioning was specified with the array "conditioning", that has two rows. The first row is the action to take: 1 is interpolation, 2 is decimation, and 3 is filtering. The second row was the parameter, i.e. the filtering frequency or the new rate. The old-style conditioning allowed up to 10 different conditioning actions to be taken on the time history file. This was found to be completely unnecessary and unused, so it was simplified. The new command line interface takes advantage of this, by completely removing the "conditioning" array and replacing it with two parameters, the "filter_cutoff" frequency and the time history "desired_rate". Below is taken from the above example showing the same action in each command line interface. Notice how much clearer the intended action is. In this case, the new field name is much more specific to the action.

Legacy	New
`f_in.conditioning(1,1:2) = [3,2];`	`fre.thfiles.filter_cutoff = 4;`
`f_in.conditioning(2,1:2) = [4,25];`	`fre.thfiles.desired_rate = 25;`

3. MISOSA and COMPOSITE Classes

Both MISOSA and COMPOSITE only require a subset of information required for FRESPID. For this reason MISOSA and COMPOSITE cases can be inferred from the FRESPID case. The new command line interface provides a way to generate template MISOSA objects (misosa_obj) or COMPOSITE objects (composite_obj) from existing frespid_obj objects. Most users will only need the batchall command built into frespid_obj which generates the MISOSA and COMPOSITE cases from a frespid_obj case, and runs all the batch jobs. The only reason to directly use misosa_obj or composite_obj is to adjust plotting options or to gain finer control over the batch job processing. If the user needs to change plotting options, he is able to generate template MISOSA or COMPOSITE cases from the FRESPID case by calling the misosa_obj or

composite_obj constructor, respectively on a frespid_obj object. Below is an example showing how the old command line interface created MISOSA and COMPOSITE cases, and how simple it is to create them with the new command line interface.

<table>
<tr>
<td>

```
% Set up blank misosa case
m_in = misosa;

% Fill in appropriate values
m_in.casename        = thename;
m_in.casein          = thename;
m_in.caseout         = thename;
m_in.controls(1:2) = {'AIL','RUD'};
m_in.outputs(1:4)    =
{'P','R','AY','VDOT','PHI'};
m_in.winon(1:5)      = {'*'};
m_in.frcalc(1:5)     = {'*'};

% save case to database
misosa(m_in,2);

% Set up blank composite case
c_in = composite;

% Fill in appropriate values
c_in.casename        = thename;
c_in.casein          = thename;
c_in.caseout         = thename;
c_in.inpgm           = 'MIS';
c_in.controls(1)     = {'AIL'};
c_in.outputs(1:4)    =
{'P','R','AY','VDOT','PHI'};
c_in.winon(1:5)      = {'*'};
c_in.frcalc(1,1)     = {'*'};

% Save case into database
composite(c_in,2);

% Run the cases
frespid(thename,3);
misosa(thename,3);
composite(thename,3);
```

</td>
<td>

```
% Create a misosa case from
% the frespid_obj
mis = misosa_obj(fre);

save(mis);

% Create a composite case from
% the misosa_obj
com = composite_obj(mis);

save(com);

wait(batch(fre));
wait(batch(mis));
wait(batch(com));
```

</td>
</tr>
</table>

4. CIFER Frequency Response Class

The non-parametric analyses for CIFER are all available through the ciffrq object. Primarily it provides access to the frequency response database, giving the ability to load a frequency response from the database and access the data in the MATLAB workspace. The code below shows how easy it is to load a frequency response from the database. The freq, mag, phase and coh fields are the underlying frequency data as expected.

```
>> fr = ciffrq('XVLATSWP_FRE_A0000_AIL_P')

fr =

        name: 'XVLATSWP_FRE_A0000_AIL_P'
    comments: 'LATERAL FR SWP FOR XV15 HOVER'
        freq: [1x399 double]
         mag: [1x399 double]
       phase: [1x399 double]
         coh: [1x399 double]
        info: [1x1 struct]
```

The plot command is overridden for the ciffrq object, internally calling the underlying QPlot functionality available in the GUI. The other non-parametric analysis tools: RMS, bandwidth, crossover, and arithmetic are similarly available as functions.

Additional plotting methods are provided. The MATLAB Control System Toolbox Bode plot-style is available via the bode function. An interactive Bode plot with gain and phase margins is created via plot_perf.

```
fr = ciffrq('CLROLL_COM_AB000_LAT_P');
plot_perf(fr,'correct', ...
   corrset('spower', -1),'construct',false,'markers',true)
```

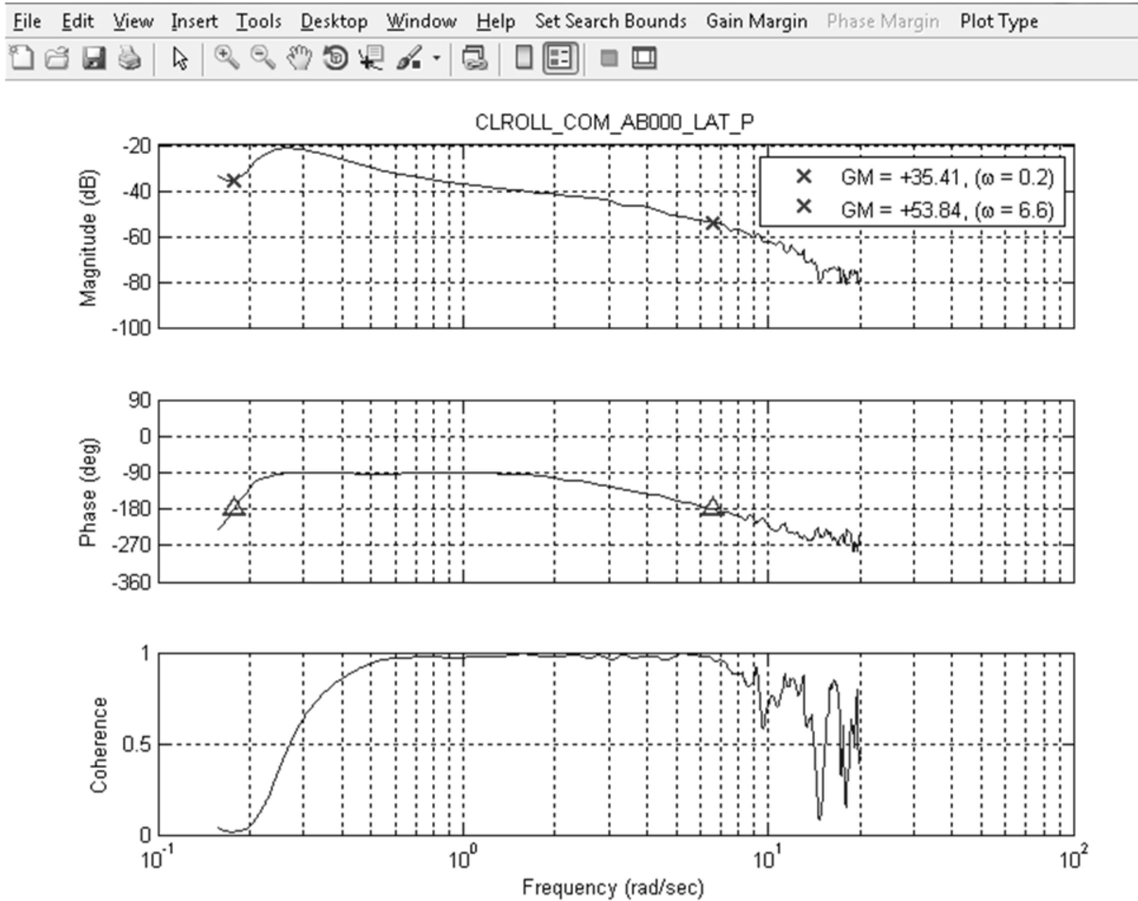

Figure 8: Crossover plot

For comparison of two frequency responses the command line interface provides a function to display the error between two responses. In this case the error function used is shown in the first equation below. The resulting magnitude and phase curves are the difference in magnitude (in dB) and difference in phase.

$$error = \frac{actual}{expected} \tag{12}$$

$$mag_{err} = mag_{actual} - mag_{expected} \quad \textbf{(13)}$$

$$phase_{err} = phase_{actual} - phase_{expected} \quad \textbf{(14)}$$

An example error plot is shown below. The actual response was the expected response multiplied by a gain and a phase delay. The positive magnitude curve indicates that the actual response has a higher magnitude curve than the expected response. The negative phase curve indicates that the actual response has a lower phase curve than the expect response. There are also two addition curves on each plot. These are the Maximum Unnoticeable Added Dynamics (MUAD) bounds developed based on handling qualities research. The bounds were created during a fixed-wing handling qualities survey. The bounds were developed by overlaying the variations in closed loop responses that did not result in a change in the Chopper-Harper pilot rating for that study (Ref 1.). The bounds are intended to represent the maximum amount of change an aircraft could undergo without the pilot changing the rating of the aircraft. The bounds are narrowest near crossover indicates that pilots are more sensitive to changes in that range. This agrees and explains why validation near the crossover region is important for model validation. For that reason, MUAD bounds will be used in this thesis to evaluate the quality of match between frequency responses (Ref 9.). The command line interface provides these plots via the plot_error function.

```
fr1 = ciffrq(1, logspace(-1,1.5,1000));
fr2 = correct(fr1,corrset('time_delay',.05))*1.2;
plot_error(fr2,fr1)
```

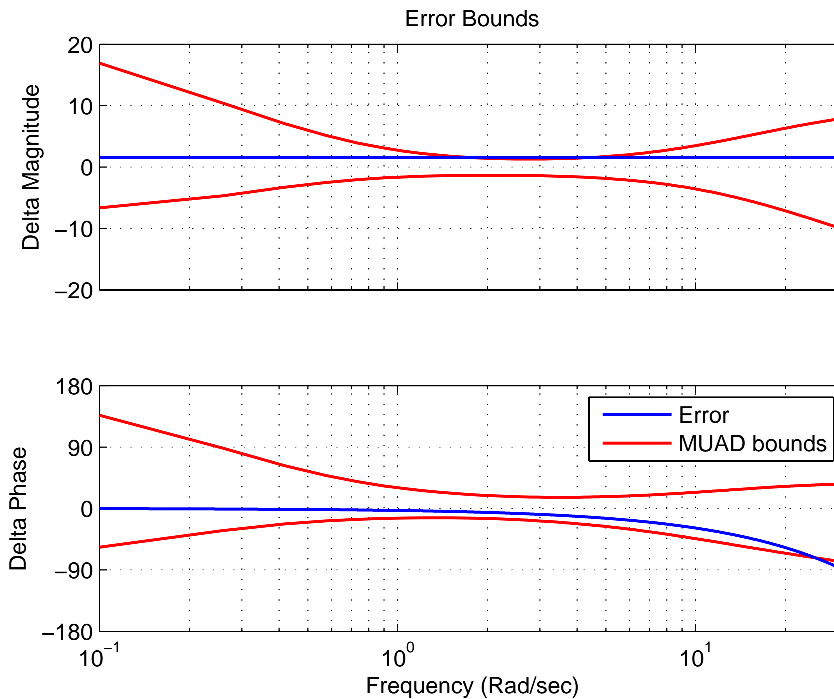

Figure 9: Error plot example

In addition to providing the functional call for arithmetic, the ciffrq class supports arithmetic operators such as + and -. The new response can then be named and saved back to the database. Below is an example showing multiplication of the frequency response by a scale factor.

```
>> fr2 = 5*fr

fr2 =

        name: ''
    comments: ''
        freq: [1x1000 double]
         mag: [1x1000 double]
       phase: [1x1000 double]
         coh: [1x1000 double]
        info: [1x1 struct]
```

The example below shows how to multiply by a power of s using the “correct” command.

```
>> fr3 = correct(fr,corrset('spower',-1))

fr3 =

        name: 'XVLATSWP_COM_ABCDE_AIL_P'
    comments: ''
        freq: [1x987 double]
         mag: [1x987 double]
       phase: [1x987 double]
         coh: [1x987 double]
        info: [1x1 struct]
```

Two frequency responses can be multiplied as shown below.

```
>> fr6 = fr*fr2

fr6 =

        name: ''
    comments: ''
        freq: [1x1000 double]
         mag: [1x1000 double]
       phase: [1x1000 double]
         coh: [1x1000 double]
        info: [1x1 struct]
```

The last major feature is compatibility with the built-in MATLAB Control System Toolbox LTI objects. The ciffrq class supports conversion to and from the frd object, the frequency response LTI object. frd objects are compatible with many of the LTI tools available in the MATLAB Control System Toolbox, so users can easily use these tools with frequency responses generated from CIFER. Any LTI object that can be converted to an frd object can also be converted to the ciffrq class. This feature allows users to multiply a frequency response by a transfer function object (tf class) or state-space object (ss class). This is shown below.

```
>> s = tf('s')

Transfer function:
s

>> fr5 = fr*(s+1)/(s-1)

fr5 =

        name: ''
    comments: ''
        freq: [1x1000 double]
         mag: [1x1000 double]
       phase: [1x1000 double]
         coh: [1x1000 double]
        info: [1x1 struct]
```

These classes provide a robust and user-friendly interface to the CIFER frequency domain system identification process. They provide the base to develop automated tools involving CIFER.

IV. Analyzing simulation using system identification

Now that a more robust and user-friendly command line interface was developed, an integrated simulation and system identification tool could be developed. Traditionally, generating frequency responses from a block diagram involves the user moving data from Simulink to CIFER and back again. The basic flow of the process follows:

1. Obtain time histories from the Simulink diagram
2. Package data into CIFER compatible format
3. Create a FRESPID case
4. Run FRESPID batch
5. Create a MISOSA case if needed
6. Run MISOSA batch if needed
7. Create a COMPOSITE case
8. Run COMPOSITE batch
9. Check frequency response and look for low coherence or abnormalities
10. Go back to step 1 if frequency response is not good enough

The new command line interface consolidates steps 4-8 into one command, "`batchall`". In the following sections, the details of the work flow will be explained within the context of an automated validation tool designed to sweep a block diagram.

A. Automating the process

1. Obtain time histories from the Simulink diagram

There are several steps involved in sweeping the block diagram. First, a data gathering mechanism needs to be added to the Simulink diagram. Root level output ports

are a good option because the SIM command returns the time histories from these ports. However, the “To Workspace” and “To File” blocks work as well. The sweep signal needs to be defined within the diagram, or computed before running the simulation and passed in via a root level input port or “From Workspace” block.

The simulation time step and duration need to be compatible with the frequency range of interest. The high frequency limit determines the maximum time step size, and the low frequency determines the minimum record size. The details of these parameters can vary. However the rules are thumb are what were used (Ref 2.). These are shown below. The filter frequency $\left(f_{filter}\right)$ is set such that frequency content before the maximum frequency of interest is not attenuated. The sample frequency $\left(f_{sample}\right)$ is set such that the filter can operate with sufficient data and not encounter effects of being near the Nyqist limit $\left(f_{Nyq} = 2f_{sample}\right)$. Step size (dt) is defined as the inverse sample frequency.

$$f_{filter} = \frac{5\omega_{max}}{2\pi} \tag{15}$$

$$f_{sample} = 5f_{filter} \tag{16}$$

$$dt = \frac{1}{f_{sample}} \tag{17}$$

The time history record size is the summation of several components. Because of the requirement from CIFER that the time record must begin and end in trim the sweep needs to have region of zero input at the beginning and the end for t_{zero} seconds, as illustrated in Figure 11.

In order to capture the low frequency data, the sine function is held for a full period t_{park} seconds at the low frequency. Following the full period, an exponential ramp up of frequency from the lower frequency limit to the upper frequency limit will excite the system across the entire frequency range. The length of the exponential ramp is five times the largest window size to be used, which is twice the t_{park} time. The sweep frequency with the park and exponential ramp are shown in Figure 10 below.

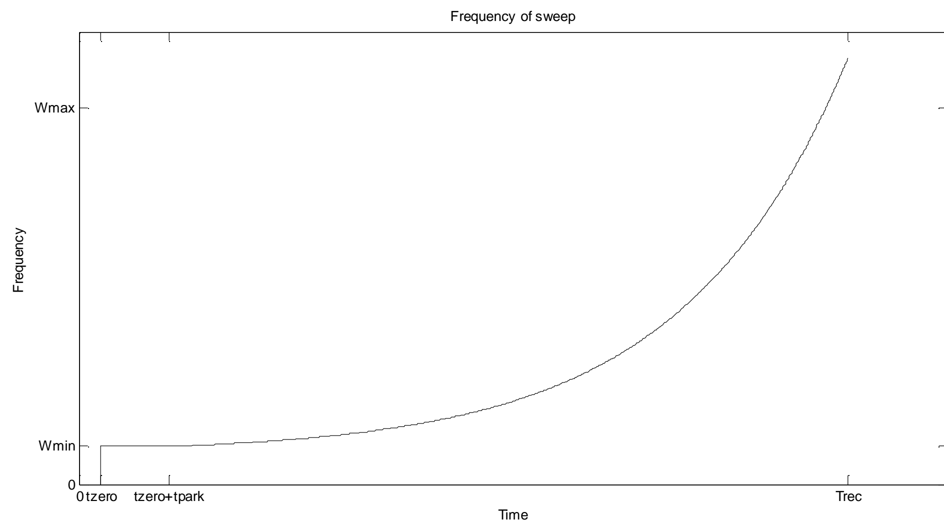

Figure 10: Sweep frequency

At the beginning of the sweep there is a ramp up for t_{fadein} seconds to full amplitude. Similarly, at the end of the sweep, there is a ramp down for $t_{fadeout}$ seconds. This is shown in Figure 11.

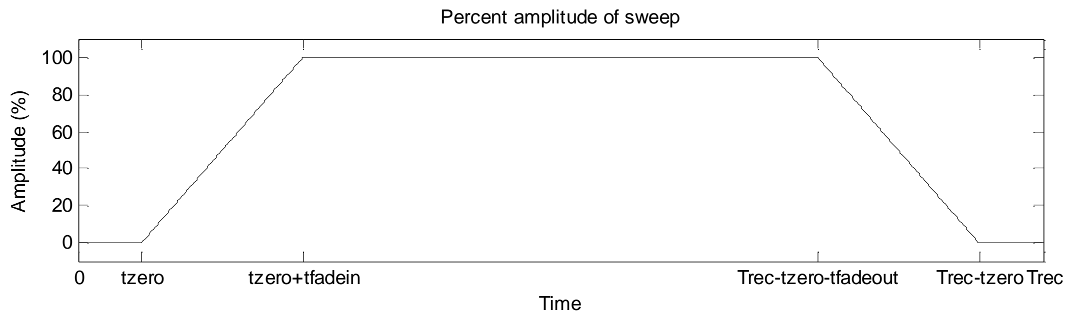

Figure 11: Sweep amplitude

The total length of the record then becomes the summation of each of these. It is defined below, along with its components. The maximum window size (T_{max}) is typically defined by the maximum period of interest, which is t_{park} in this case. The total record length (T_{rec}) should be five times the maximum window size to allow adequate data for identification. The zero time and fade in and fade out are added to that because they provide no content, and do not really count towards the five times maximum window size rule.

$$T_{rec} = 5T_{\max} + 2t_{zero} + t_{fadein} + t_{fadeout} \textbf{, where}$$

$$t_{park} = \frac{2\pi}{\omega_{\min}}$$

$$T_{\max} = 2t_{park} \tag{18}$$

$$t_{fadein} = t_{fadeout} = 2$$

$$t_{zero} = t_{trim} + 2$$

The last component of a computed sweep is noise. Injecting low amplitude white noise into the sweep aids the identification process by increasing high frequency input.

In Appendix C, the equations of the sweep are listed, in both integral form and analytic form. Below is an example sweep from 1 rad/sec to 10 rad/sec.

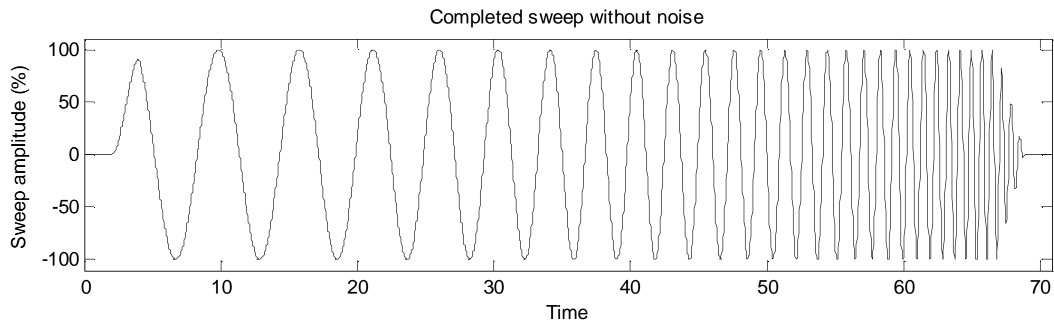

Figure 12: Completed sweep without noise (1-10 rad/sec)

With the input sweep defined, the simulation with the input sweep needs to be performed. MATLAB provides the SIM command to inject the sweep into a designated root level input port, and return the time histories at the designated root level output port.

2. Packaging the data

The time histories generated from the SIM command are going to be passed to the FRESPID program in CIFER as a MATLAB MAT-file. For SISO identifications, generic channel names such as 'IN' and 'OUT' can be used because there are only two channels. In the more general MIMO case, user name bindings will be required in order to construct the time history file.

3. Creating a FRESPID case

A FRESPID case requires four sets of information: 1) time history specification and conditioning, 2) control name and channels, 3) output name and channels, and 4) windowing parameters.

In the automated sweeping process, the program just wrote the time history file, so it knows all the specifications of the time history (format, filename, sample rate, and filtering). For SISO cases it will also have the generic control and output names, which can also be the channel names. For MIMO cases the user specified control and output names are also present, so the first three sets of information are known simply because of the integrated process. The only missing set of information for generating the FRESPID case from the sweeping process is the windowing parameters.

The windowing parameters have the most flexibility of the other FRESPID case parameters. The majority of the other parameters only have one correct input (the filename of the time history file, for example). However, the windowing parameters have

to be adjusted to suit the individual identification. There are six windowing parameters as shown in the table below.

Table 1: Windowing parameters

Parameter	Units
Window length	Seconds
Number of input points	Positive Integer
Number of output points	Positive Integer
Decimation Ratio	Positive Integer
Minimum Frequency	Radians per second
Maximum Frequency	Radians per second

The first three windowing parameters are coupled by algorithm requirements. The window length and the number of input points are related by the time history step size. For example, a window length of 10 seconds with a step size of .01 will have 1001 points in the window. The CZT used within CIFER requires the number of input points plus the number of output points to be a power of two.

The frequency response output points are the output of FRESPID, so there needs to be an adequate number of points over the frequency range. The frequency range for a given window is determined by the minimum frequency and maximum frequency parameters, as provided by the user. These are subject to limits provided by the window size and step size. The spacing of the output points in the frequency domain is always constant, so the points have even coverage on a linear frequency plot. However, on a logarithmic plot the number of points at low frequency is much fewer than that at high frequency as shown in Figure 13. So in order to increase the resolution in the low frequency, the user may need to decrease the upper frequency limit, which decreases the frequency increment. If too few points are located in the low frequency, the frequency limits may need to be adjusted to increase the frequency resolution at low frequencies.

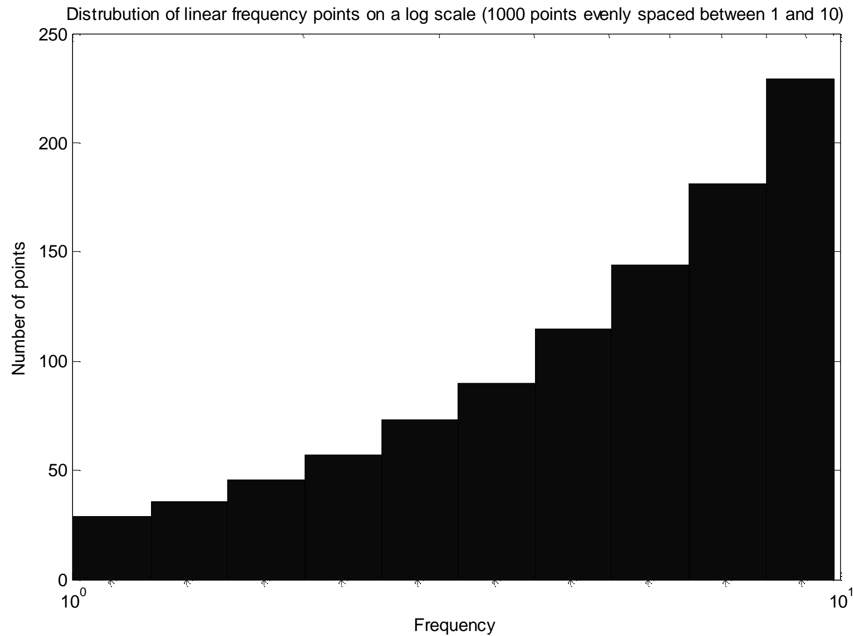

Figure 13: Frequency point distribution

Choosing the window lengths is primarily a trade between two objectives. The first is to capture low frequency data points. The second is minimizing random noise in the responses. In order to capture low frequency data points, long windows must be used. However, long windows means fewer transforms can be performed on the data. If the window length is equal to the record length, then only one transform can be performed on that data. Any random noise present in the frequency response is impossible to remove. However, if the window length is half that of the record length, then in the absence of overlap two transforms can be performed on the data, one on the first half and one on the second half. Now there are two sets of frequency responses, which should be identical assuming the plant remains constant in the frequency domain over time. If the set of points are averaged then the random variations will tend to zero. If more overlap is allowed, more transforms can be performed on the data, each time averaging the frequency points. The random errors cancel out and result in a smooth response with

fewer variations due to noise. Smaller windows allow for more overlapping transforms to be performed, small windows are less affected by random noise.

Given the trade between small and large windows, how should windows be picked? FRESPID allows for up to five windows, allowing the user to fill the trade space. The integrated tool uses five windows to span the trade space. When system identification is performed on simulation data, the windowing trade is less important because the data is collected in a perfectly controlled environment without external influences as opposed to collecting flight test data. In simulation, the only noise present is the noise added by the user, which is small compared with the sweep amplitude. The simulation will probably have fewer nonlinearities than an actual aircraft, which further simplifies the problem. In the end, only reasonable windows need to be chosen in order to get acceptable results.

Given a frequency range of interest (the target minimum and maximum frequencies), target minimum $(T_{\min})$ and maximum windows $(T_{\max})$ can be created. There are bounds for the window sizes that cannot be violated in either direction. For a hard limit on the minimum window size, there must be at least 5 points of data in each window $(\omega_{\max_{\text{limit}}})$. However, this is an impractical window size, because there would be no frequency range, as the minimum frequency point is the maximum frequency point for that window. For this reason the smallest window size is set to have a window minimum frequency one decade before the target maximum frequency. This ensures one decade of data is available in that window (Ref 2.). Anything smaller than that will provide insufficient frequency content for subsequent CIFER programs. This window size should always be larger than the hard lower limit because the tool should have chosen a step size

25 times the maximum frequency of interest. In the case where the target minimum and maximum frequencies are less than a decade apart, the integrated tool sets the target maximum frequency to be one decade above the target minimum frequency.

$$\omega_{\max_{\text{limit}}} = \frac{2\pi}{5dt} \quad \textbf{(19)}$$

$$T_{\min} = \begin{cases} \omega_{\max} \geq 12\omega_{\min} \Rightarrow \dfrac{20(2\pi)}{\omega_{\max}} \\ \omega_{\max} < 12\omega_{\min} \Rightarrow \dfrac{2\pi}{\omega_{\min}} \end{cases} \quad \textbf{(20)}$$

(Ref 2.)

For the target maximum window, the upper limit is the length of the time history records $\left(T_{\max_{\lim it}}\right)$. This is impractical as a limit because there will be no repeated transforms, so the random error will be high. For this reason, the maximum window size is half the record length, which determines the minimum frequency allowed $\left(\omega_{\min_{\lim it}}\right)$. The target maximum window size is twice the period of the target minimum frequency (Ref 2.). This allows the CZT to capture lower than the target minimum frequency.

$$T_{\max_{\lim it}} = \min\left(T_{rec}\right) \quad \textbf{(21)}$$

$$\omega_{\min_{\lim it}} = \frac{2(2\pi)}{T_{\max_{\lim it}}} \quad \textbf{(22)}$$

$$T_{\max} = \frac{2(2\pi)}{\omega_{\min}} \quad \textbf{(23)}$$

(Ref 2.)

The remaining three windows are distributed evenly between the minimum and maximum windows sizes. This automated window selection functionality is available in

the command line interface through the auto_window function provided for the frespid_obj class.

The code below demonstrates the auto_window function. Internally it checks for the input minimum and maximum frequencies based on step size and record length. It also adjusts the decimation and filtering based on equation 15 and 16. The output is a new frespid_obj with the modified windows and thfile objects. The user can then perform additional adjustments as needed or proceed to generate the frequency responses.

```
>> fre = auto_window(fre, .1, 30)
??? Error using ==> frespid_obj.auto_window at 60
Requested minimum frequency exceed minimum frequency possible with
given thfiles (0.133 rad/sec)

>> fre = auto_window(fre, .133, 30)

fre =

                     name: 'XVLATSWP'
                 comments: 'LATERAL FR SWP FOR XV15 HOVER'
                  caseout: 'XVLATSWP'
                   db_out: 1
                 crosscor: 1
              fr_file_out: 0
           fr_file_format: 'CIFER'
              fr_file_dir: 'C:\CIFER_Pro\jobs\tfdata'
    th_file_out_unformatted: 0
          th_file_out_ascii: 0
                 controls: {'AIL'  'RUD'}
                  outputs: {'P'  'R'  'AY'  'VDOT'  'PHI'}
                  windows: [1x5 windows]
                  thfiles: [1x2 thfile]
                   frcalc: [5x2 logical]
                gen_plots: 0
                    plots: [0 0 1 0 0 0 0 0 0 0 0 0]
               heavy_grid: 1
               large_plot: 1
            decimate_data: 1
              plot_format: 'PostScript'
                  frnames: {5x2x5 cell}
```

4. Generating the frequency responses

Once the FRESPID case is generated, the tool can proceed to generating the frequency responses with the “batchall” command. For SISO analysis, there is no need to

run MISOSA, so only a COMPOSITE case needs to be generated. At this point, the initial automation is done, and the frequency responses are returned to the user. The user is free to examine the generated frequency responses and go back and refine the FRESPID case if needed.

The command line interface now provides steps 2-4 in one function, cifer_siso, which conducts the SISO analysis as explained above. It requires a case name, control and output names, time, control and output time histories, and minimum and maximum target frequencies as inputs. It then packages the data, sets up the FRESPID case, and invokes the FRESPID and COMPOSITE batch jobs, and returns the final frequency response. A MIMO process was prototyped, but never fully developed, and was set aside for future work.

B. Integration with CONDUIT

With the command line interface providing a function that can perform SISO analysis with given data, the last step is to integrate the data gathering process with CONDUIT in order to perform model validations.

The CONDUIT problem format requires all input/output pairs to be connected to root level input and output ports. This provides access to all the time histories the validation tool needs. CONDUIT also utilizes the names of input and output ports when binding with specifications for the problem. This approach will be used as well in the validation tool.

There are three sets of validation: bare airframe, closed loop and broken loop. Each set of validations requires different input and output ports for the analysis. The closed loop is the simplest, and only requires the piloted input port and its corresponding

output port. The bare airframe has special requirements because it is a MIMO analysis, so for the scope of this thesis it will be ignored. The broken loop response validation requires additional considerations.

When performing broken loop response validation using CIFER, the loop must be closed in the case of unstable open loops. This is due to the fact that CIFER requires bounded time history records, as well as the fact that linear simulations are only valid near their trim point. For this reason, special input/output ports are required to capture this data, while maintaining the loop closure.

CONDUIT also requires special input/output ports for broken loop analysis. For LINMOD to return the broken loop response, the loop must be broken. CONDUIT provides a special Simulink switch block, called a "CONDUIT Gain/Phase Margin Switch" for use where a broken loop analysis is required. It has 2 inputs and 2 outputs. The first input and output connects to the control loop; the second input and output are the special input/output ports for broken loop analysis. Figure 14 shows an example wiring that uses the block.

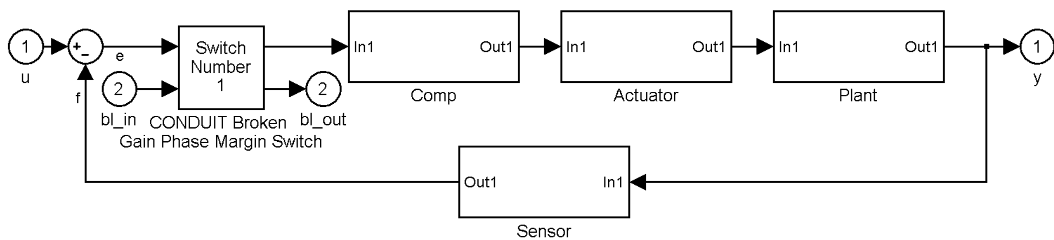

Figure 14: CONDUIT Broken Loop Switch

In the figure above is an example system with the CONDUIT Broken Loop Switch. The broken loop is from e→f. When the switch is off, it acts like the system below. If bl_in and bl_out are unused, it is as if the switch were not even there. When

sweeping the diagram using CIFER, the sweep is injected in bl_in. CIFER gathers the data on bl_out.

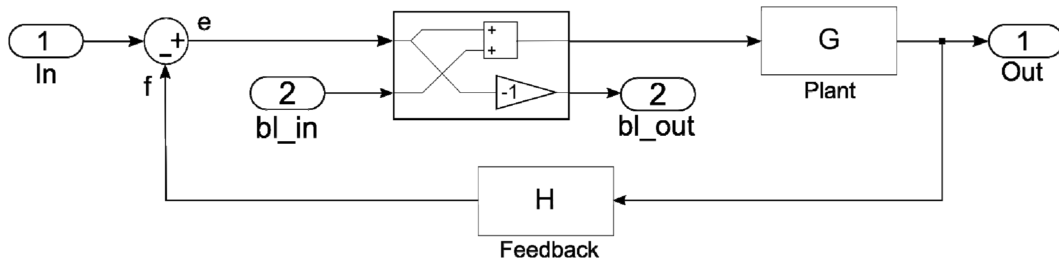

Figure 15: CONDUIT Broken Loop Switch Off

The data that needs to be passed to CIFER is e and f, not bl_in and bl_out. The math is rather simple to create e and f from bl_in and bl_out. These equations are shown below:

$$
\begin{aligned}
f &= bl_out \\
e &= bl_in - bl_out
\end{aligned}
\qquad \textbf{(24)}
$$

When performing linearization, the loop closure needs to be opened. The CONDUIT Broken Loop Switch is like the figure below. In this case bl_out is f and bl_in is e, so no changes are needed to get the broken loop response.

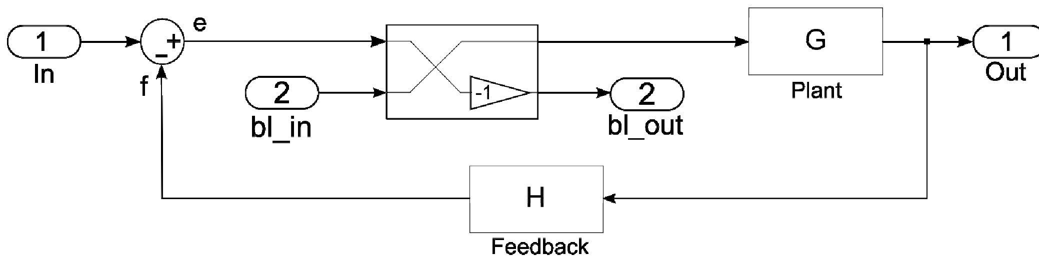

Figure 16: CONDUIT Broken Loop Switch On (Ref 7.)

When validating a broken loop response of a given axis with CIFER, the corresponding CONDUIT Broken Loop Switch must be activated during the LINMOD analysis, but deactivated during the CIFER analysis. This is different from most of the other CONDUIT tools, where the switches remain constant throughout an analysis.

With the details of the CONDUIT integration worked out, a GUI was developed to provide an interactive interface for a user to define the required inputs for validation as well as an interface to the results. The Linearization Validation Tool GUI is shown below. The top two-thirds is the results section, with plots display. The bottom third is the input section.

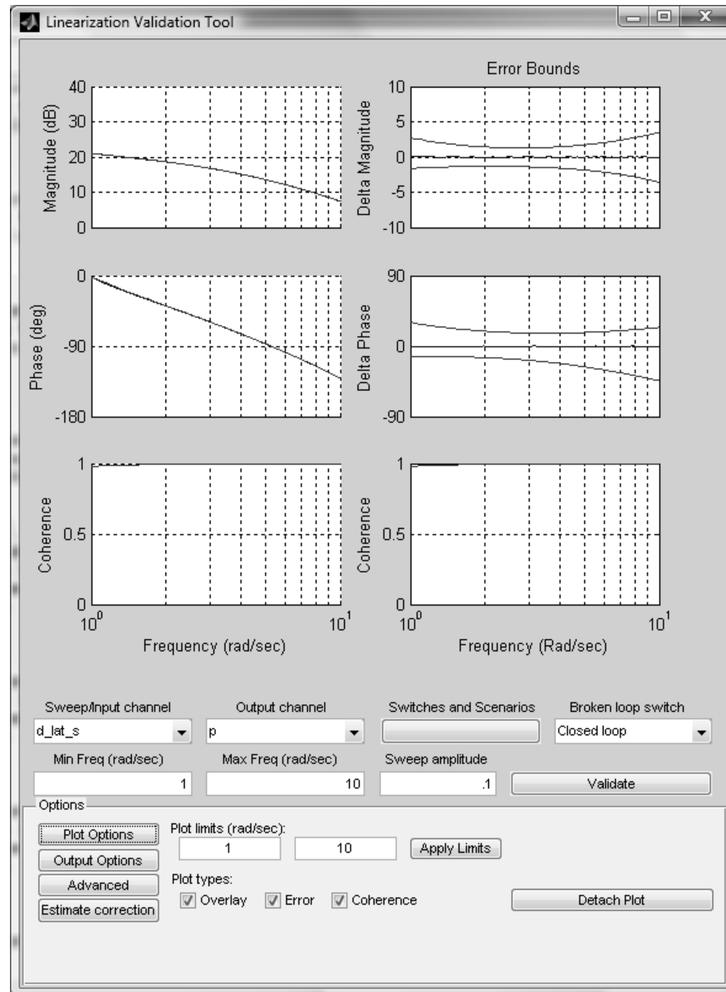

Figure 17: Linearization Validation Tool

The results section can have two different plots. On the left hand side is the overlay plot between the linearized model as extracted using LINMOD and the CIFER frequency responses as extracted for the simulation time history data. On the right hand

side are error plots that show the linearized model response divided by the CIFER response. The MUAD bounds are also present.

The seven inputs in the input section above the options pane define the validation to be performed. The “Sweep/Input channel” and “Output channel” fields select the input and output ports used for CIFER and LINMOD. The “Switches and Scenarios” field is a standard CONDUIT feature that allows for different configurations to be tested. The “Broken loop switch” drop down allows the user to specify the corresponding broken loop switch. The user selects either “Closed loop”, which means during both analyses the switches and scenarios configuration are the same, or a particular switch number to facilitate a broken loop response, which means during LINMOD analysis the switch is “on” and during CIFER analysis the switch is “off”. “Min Freq” and “Max Freq” control the automatic windowing scheme, the minimum step size, and the sweep that is generated. The “Sweep amplitude” field controls the sweep amplitude.

Most of the other optional features in the GUI are output or plot adjustments. The "Estimate correction" button utilizes the transfer function fitting program within CIFER called NAVFIT. It fits a gain and time delay to the error response.

In the figure below is an example system. In the titles of the error plots the gain and delay estimations, as well as the "cost", are shown. The cost is the measure of how good the transfer function fit is. A cost of less than 50 is considered an excellent model. In this case the cost is .1, which implies the fit is basically perfect. The gain is very close to 1, but the time delay is -.0144 seconds. This is about 1.5 time steps in this 100 Hz simulation.

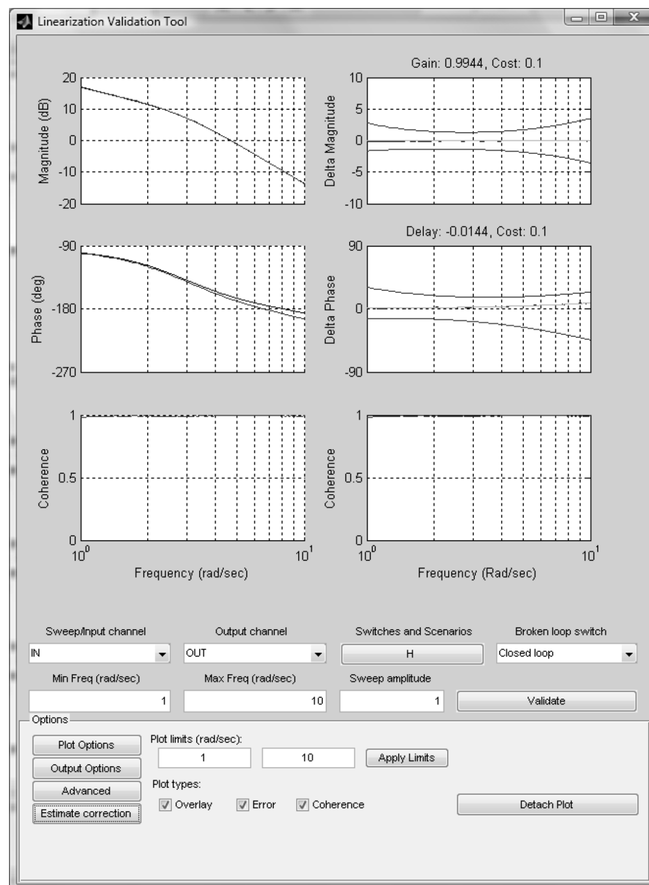

Figure 18: Estimate correction example

C. Limitations

It is known that this automation process has some drawbacks. The primary drawback is the lack of sanity checking by a human. The time history data is passed directly into CIFER and the CIFER output is checked against LINMOD. Broken loop responses are particularly sensitive to well-behaved simulations. As will be seen later, one case has been encountered with rate-limiter saturation where the feedback loop response dominated the sweep and CIFER identifies a -1 instead of the broken loop which will be discussed later. Since then a check has been added and warnings that help identify problems such as unstable loops, but it is still important to examine the time history data.

Besides the time history problems, choosing small windows could create difficulties in capturing lightly damped roots. The general rule of thumb is that the user needs a target minimum frequency of one tenth the frequency of the lightly damped root (Ref 2.). In the case of the validation tool, the tool can use the LINMOD results to warn about lightly damped roots and suggest a new minimum frequency. The problem with lowering minimum frequency is that it requires long time histories. So besides taking longer to run the simulation, this leads into another problem with windowing.

The auto-windowing mechanism works best when the frequency range is around two decades. At less than a decade separation, the small window criterion is violated, and the maximum frequency is automatically increased to one decade separation. At very large decade separations the time history records become very long and time steps become very small. The result is the small windows have plenty of points, but the large windows have too many. When decimating, the lower frequency points get removed much faster than the high frequency points. This means the large windows have poor

resolution in the low frequency, the region where they are supposed to have good resolution. Tangentially, the combination of long records with small time steps means that the number of points in the records can exceed the internal CIFER limits.

In general, the automation attempts to remove the requirement for the user to understand the underlying principles behind CIFER before he can perform model validation. This means the user will be getting results from a very streamlined process. The entire point of utilizing CIFER as the truth model was that it is transparent and very robust. The automation removes some of that transparency and if the automation proves to be flawed, validations that expose a flaw will return spurious results. However, the next three examples should demonstrate the ability of the validation tool to perform its intended task, and ability to identify when it is not working.

V. Comparison of system identification results with theoretical predictions and LINMOD for a simple open loop elements

The first example case is a simple first over second analytic transfer function, with an actuator in the open loop. Different nonlinearities are placed after the actuator model. Both LINMOD and CIFER will be used to identify the transfer function for the system. In the open loop, the magnitude of the input to the nonlinearities is going to be fairly constant, so describing function analysis will be available to compare against the two numerical results. The following nonlinearities were evaluated:

1. Hysteresis
2. Saturation
3. Dead zone
4. Lookup Table
5. Memory block
6. Time delay
7. Rate limiter

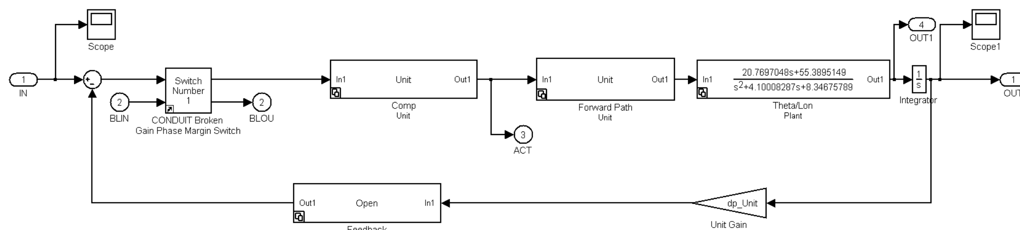

Figure 19: Simple 2nd order setup

The sweep amplitudes and nonlinearity parameters were intentionally chosen to demonstrate when CIFER and LINMOD will diverge, and to confirm that CIFER is correctly capturing the describing function of the system and can be serve as the truth model for the validation process.

A. Validations

1. Baseline

Before adding any nonlinearities, the frequency response of the actuator and plant were generated to serve as the baseline of this validation exercise. Because there are no nonlinearities, it is expected that the validation will show an exact match between CIFER and LINMOD. As can be seen below, the first column shows the two frequency responses overlaid. The CIFER curve in the magnitude and phase plots cannot be seen because the LINMOD curve is directly on top of it. Third row is the coherence plot which is an indication of the quality of the CIFER response. In this case it is always at 1, indicating a very good response over this frequency range. In the right column are error plots. The error plots are the LINMOD response divided by the CIFER response. The red lines in the error plots are MUAD bounds. Errors within these bounds will not be noticed by pilots in flight. As expected, it can be seen that the CIFER and LINMOD curves agree exactly and the error curves are constant at 0 dB and 0 deg.

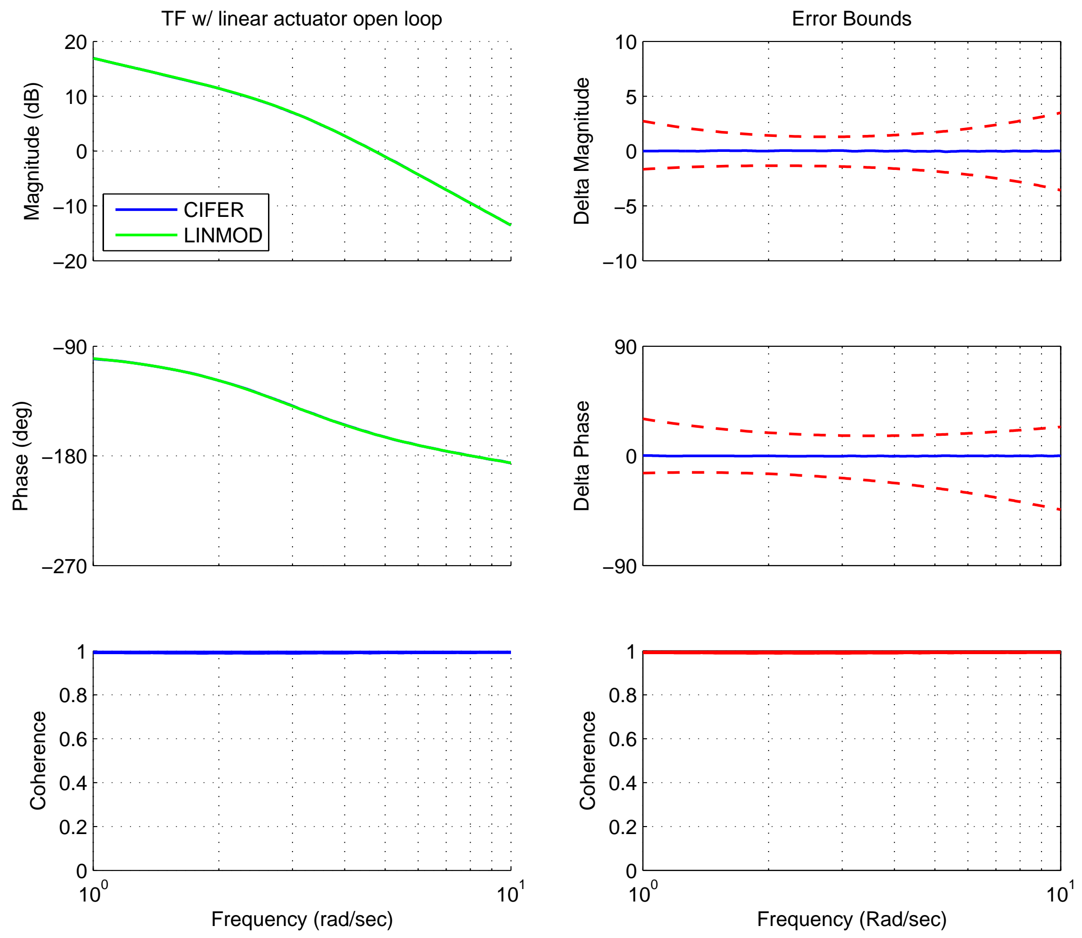

Figure 20: Baseline validation

2. Hysteresis

A hysteresis or backlash is a nonlinear behavior due to directional "stickness". It could represent loose linkages connecting an actuator to a swash plate. For the validation it was modeled in between the actuator and the plant, as shown below.

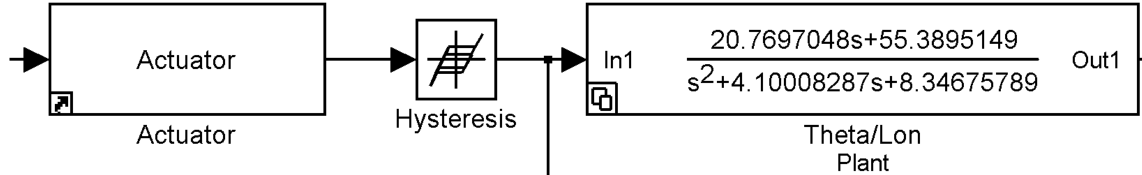

Figure 21: Hysteresis placement

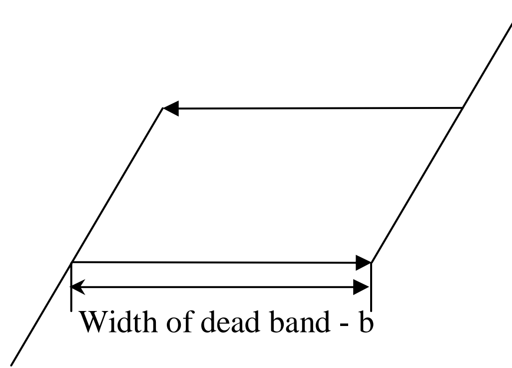

Figure 22: Hysteresis

In describing function analysis, the hysteresis effect is only a function of the width of the dead band (b shown above) and the input amplitude (A). The describing function is (Ref 5.):

$$N_{backlash}(A,b)=\frac{1}{\pi}\left[\frac{\pi}{2}+arcsin\left(1-\frac{b}{A}\right)+\left(1-\frac{b}{A}\right)\sqrt{2\frac{b}{A}-\frac{b^2}{A^2}}\right]-\frac{i}{\pi}\left(\frac{2b}{A}-\frac{b^2}{A^2}\right) \quad \textbf{(25)}$$

For open loop cases, the input amplitude is constant, so the effect on the baseline case is a scalar multiplication. In this case, because there is an imaginary term as well as a real component, there is an effect a drop in magnitude and an increased phase loss.

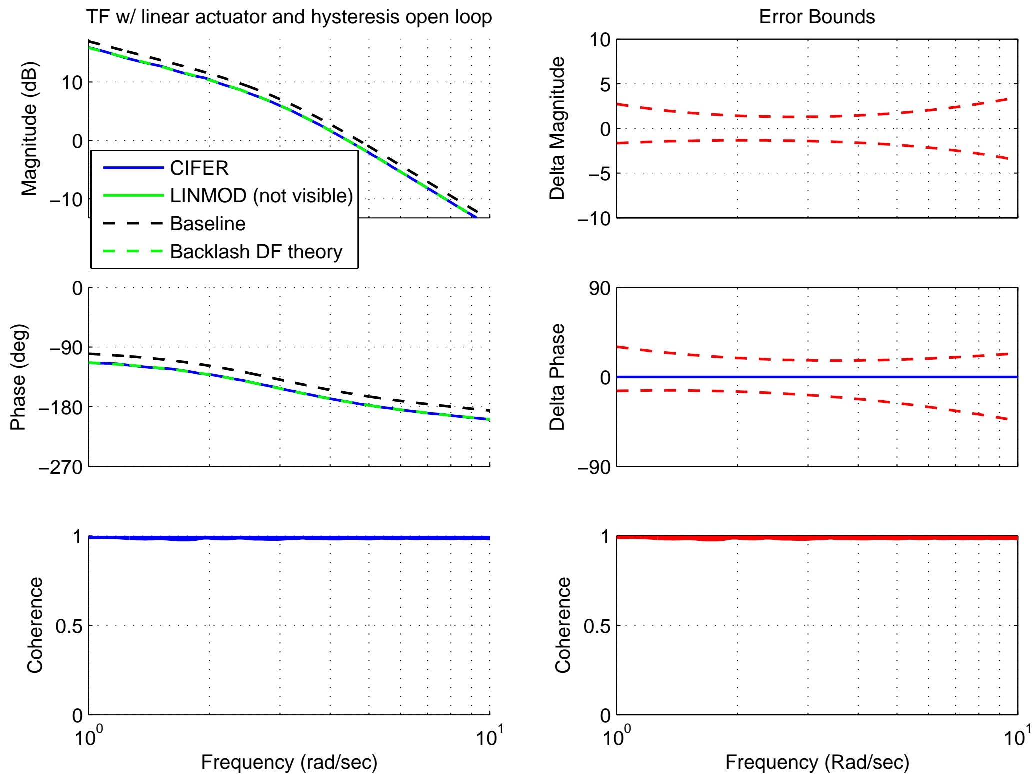

Figure 23: Open loop Hysteresis validation

In the figure above is the validation of the open loop response. There are four curves, CIFER results, LINMOD results, the baseline results, and the baseline results with the theoretical hysteresis describing function multiplied. In this case, LINMOD linearizes the system with hysteresis to zero, resulting in the LINMOD magnitude curve being at –Inf, so there is no LINMOD curve shown in the above figure. The magnitude error plot is also at –Inf, so it is not visible. On the other hand, the CIFER and theoretical hysteresis describing function curves are exactly overlaid. This confirms CIFER's ability to capture describing functions of systems with hysteresis. The CIFER and DF curves show a constant magnitude shift down and phase loss, which agrees with the form of the equation. Comparing against the baseline, CIFER shows a gain margin drop of ~6.5 db and a phase margin drop of ~10 degrees. The crossover frequency dropped by .3 rad/sec.

The reason LINMOD linearizes hysteresis blocks as a zero is because of the method LINMOD uses to linearize blocks. Some blocks, like state-space or transfer function, have closed form linearization. All other blocks are linearized utilizing a perturbation method. Apply a perturbation to a hysteresis block by any amount smaller than the dead band width, the output is not affected and the linearized gain is zero.

For linearization hysteresis blocks should either be removed, or bypassed specially when using LINMOD for linearization. For this example, the amplitude was chosen that would demonstrate the nonlinearity. For larger inputs, the effect of nonlinearity is minimized.

If a control system is operating in the amplitude where the hysteresis has a large effect, the effect of hysteresis will need to be accounted for in some other way.

3. Saturation

Saturation is an element that exists in all flight systems due to the nature of control surfaces. There are limits to the amount of deflection a control surface can move, and saturation provides a way to express the limitation in a model.

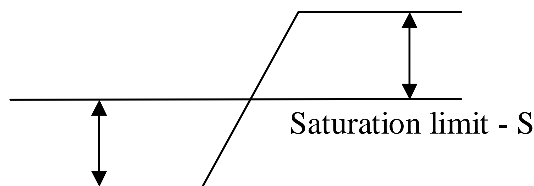

Figure 24: Saturation

The saturation describing function is a function of the saturation limit (S) and the input amplitude (A). Saturation is active only if the input amplitude is greater than saturation limit. For the validation shown, the sweep amplitude was internationally chosen to be slightly larger than the saturation limit. The describing function for saturation is (Ref 3.):

$$N_{saturation}(S,A)=\begin{cases} A>S \Rightarrow \dfrac{2}{\pi}\left[\arcsin\left(\dfrac{S}{A}\right)+\dfrac{S}{A}\sqrt{1-\left(\dfrac{S}{A}\right)^2}\right] \\ A\leq S \Rightarrow 1 \end{cases} \quad \textbf{(26)}$$

The saturation describing function has no imaginary part, so it will only affect magnitude in the open loop response.

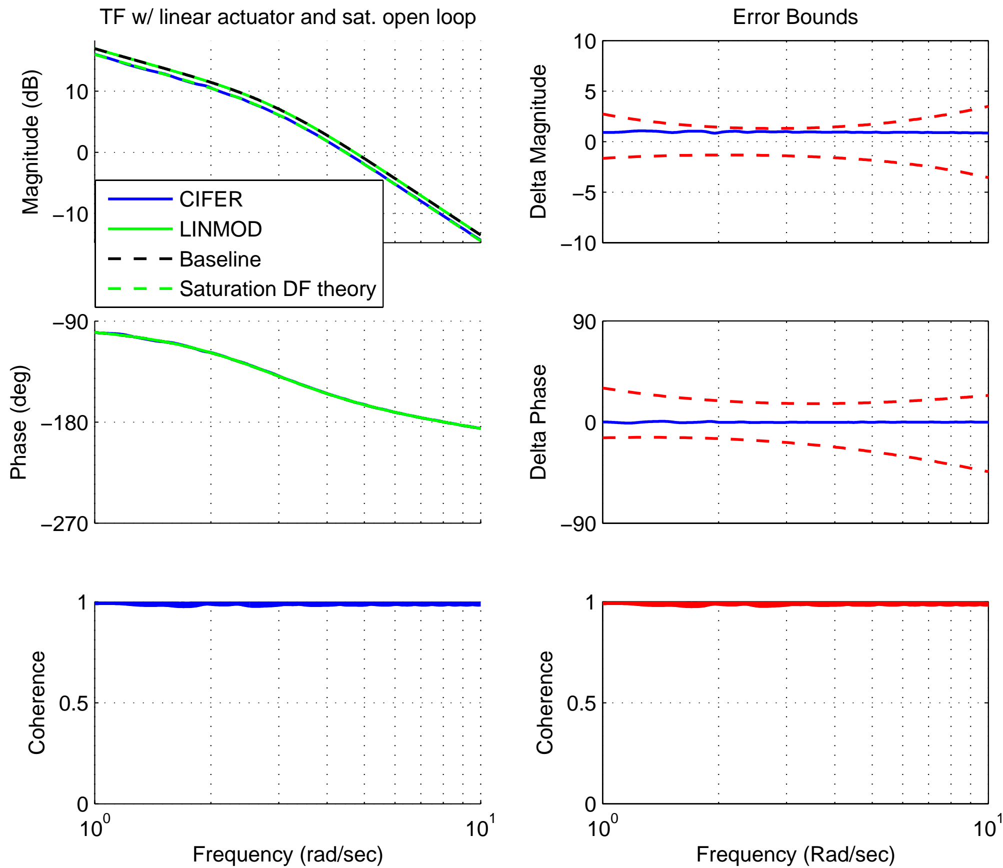

Figure 25: Open loop Saturation validation

Like the hysteresis validation plots there are four curves in the figure above. Again the CIFER and the describing function curves are overlaid, confirming CIFER's validity. The effect of the saturation is seen as a constant magnitude drop with no phase loss as expected by the form of the equation. Compared against LINMOD, CIFER

shows a 1 dB increase in gain margin, a 2.5 degrees increase in phase margin, and a .25 rad/sec decrease in crossover frequency.

This time the LINMOD result is non-zero. However, it can be seen that the LINMOD and the baseline curves are directly on top of each other. This means that LINMOD is treating the saturation as a unit gain. Examine the Simulink Saturation block parameters shown in the figure below, it can be seen that there is an option "Treat as gain when linearizing" which controls the behavior. This option is turned on by default. If this option is unchecked, LINMOD will perform a perturbation analysis. When linearizing about trim the saturation will not be active, so the effect of the saturation will linearize as a one. However, when linearizing near saturation limit, the perturbation analysis may result in a zero, hence the option to treat it as a unit gain.

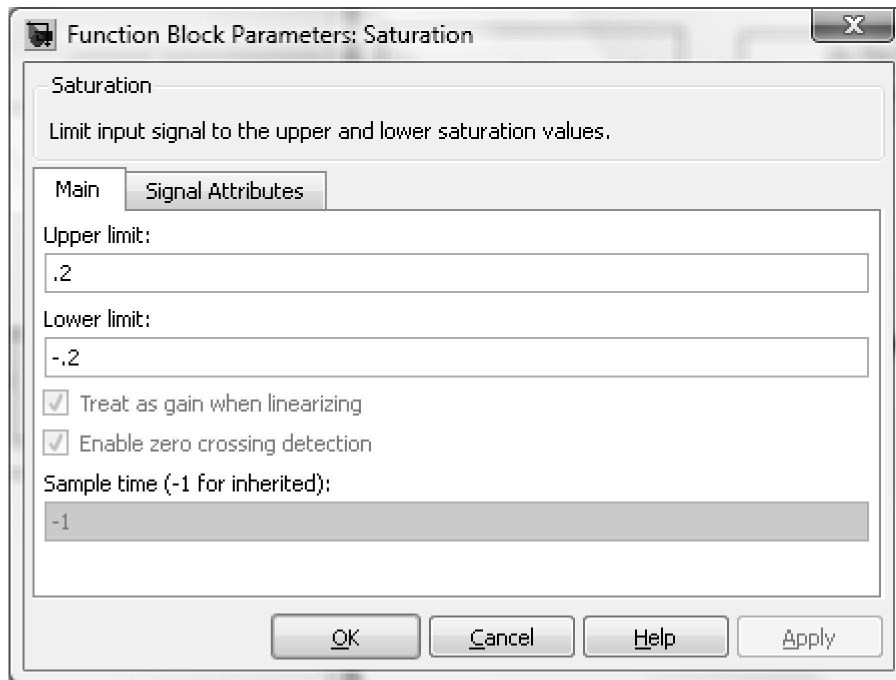

Figure 26: Saturation parameters

From a linearization perspective, saturation should not be an issue because the "Treat as gain when linearizing" option is on by default, and even if it is unchecked most linearization's are not done near a saturation point.

4. Dead zone

Dead zone is a zeroing of the output near the origin of a signal. It can be used to minimize sign changes or reduce noise at the center of a pilot control. Physically, joysticks have dead zones when the spring tension switches near the center.

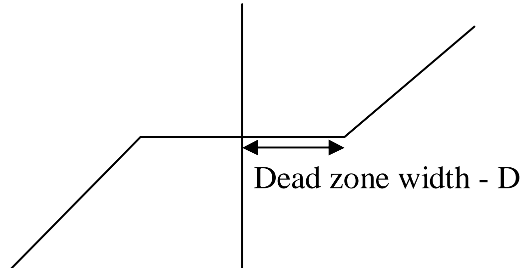

Figure 27: Dead zone

The dead zone describing function is only a function of dead zone width (D) and input amplitude (A). The dead zone describing function can be expressed using the saturation describing function as (Ref 3.):

$$N_{deadzone}(D,A) = 1 - N_{saturation}(D,A) \quad \textbf{(27)}$$

Again, dead zone only affects the magnitude of the response in the open loop. Unlike saturation, dead zone is always an active element. If the input amplitude is smaller than the dead zone width, the output is always zero. If the input amplitude is greater than the dead zone width, the describing function of the dead zone asymptotically approaches 1.

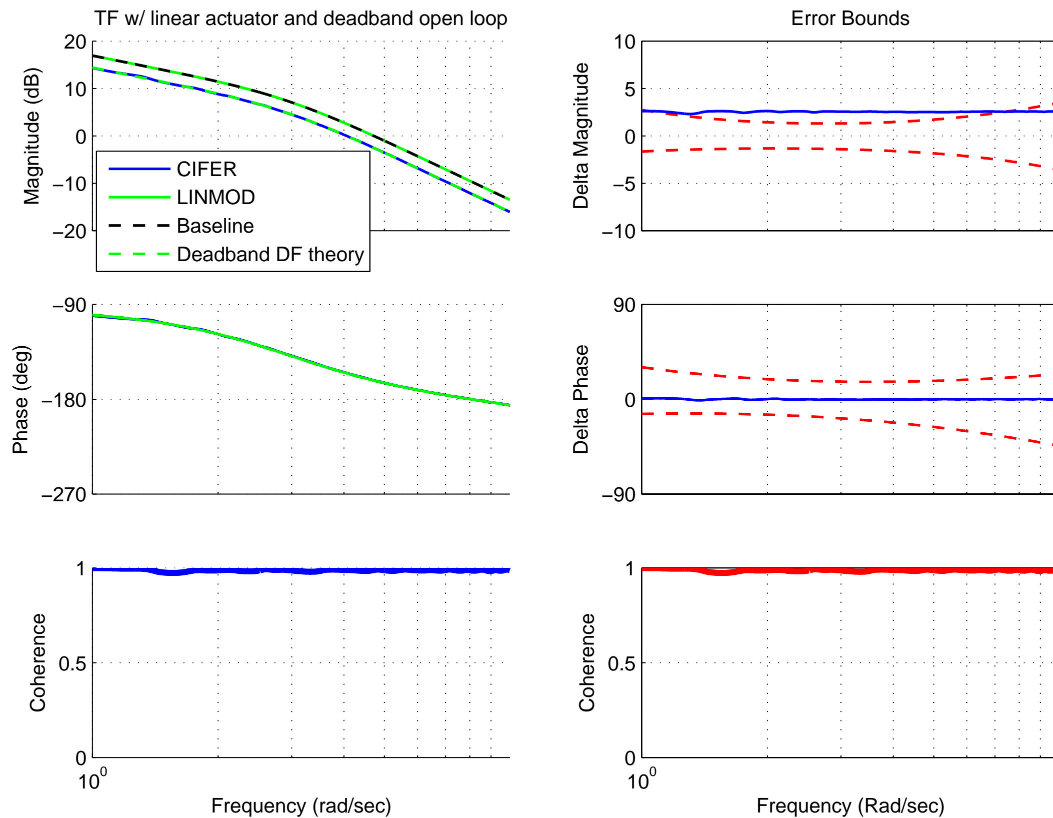

Figure 28: Open loop dead zone validation

Figure 28 shows the results of the validation. As in the other cases, there are four curves again. CIFER and the theoretical describing function theory agree very well. Like saturation, the effect is a constant magnitude drop with no phase loss. This agrees with the fact that the dead zone expression is very similar to the saturation expression. Compared against LINMOD, CIFER shows a 2.6 dB increase in gain margin, a 7 degree increase in phase margin, and a .6 rad/sec decrease in crossover frequency. As expected the direction of change of the frequency domain metrics due to the dead zone is the same as saturation.

The LINMOD curve is overlaying the baseline curve, meaning it is treating the dead zone as a unit gain. Like saturation, the Simulink dead zone blocks have the "Treat as unit gain" option that is checked by default. Unlike saturation, a linearization about

the trim point for a dead zone results in a zero response. If a user unchecks the "Treat as unit gain" option, LINMOD results of a response with a dead zone will probably be a zero.

The effect of the dead zone all depends on its amplitude. Even if a system is not operating within the dead zone, its affect will still be seen.

5. Lookup Table

A lookup table is a very flexible element, but in this case the focus is on a sensitivity change near the origin. This element is similar to a dead zone, because a dead zone can be interpreted as a lookup table with an inner slope (k_1) of zero.

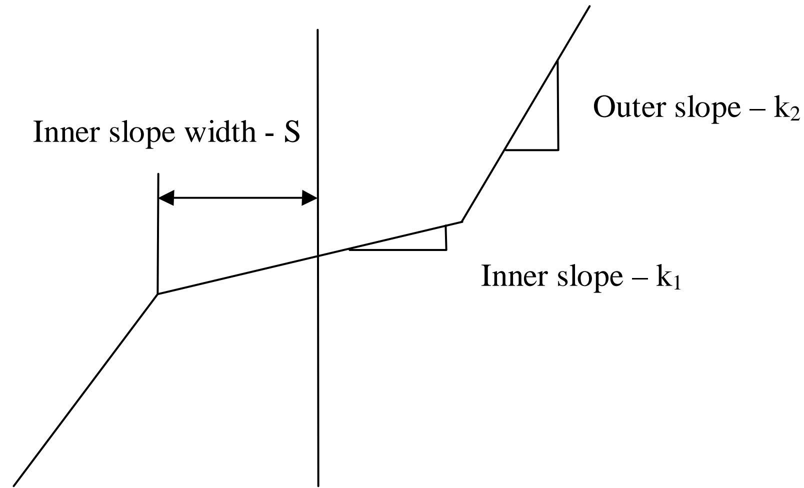

Figure 29: Lookup table

The inner slope (k_1) does not have to be less than the outer slope (k_2). This describing function is only a function of the lookup table parameters (k_1, k_2, S) and amplitude (A). The describing function for this setup is (Ref 3.):

$$N_{lookup}(S, k_1, k_2, A) = \begin{cases} A < S \Rightarrow k_1 \\ A \geq S \Rightarrow k_2 + 2\dfrac{(k_1 - k_2)}{\pi}\left(\arcsin\left(\dfrac{S}{A}\right) + \dfrac{S}{A}\sqrt{1 - \left(\dfrac{S}{A}\right)^2}\right) \end{cases} \quad \textbf{(28)}$$

This nonlinearity only affects magnitude in the open loop.

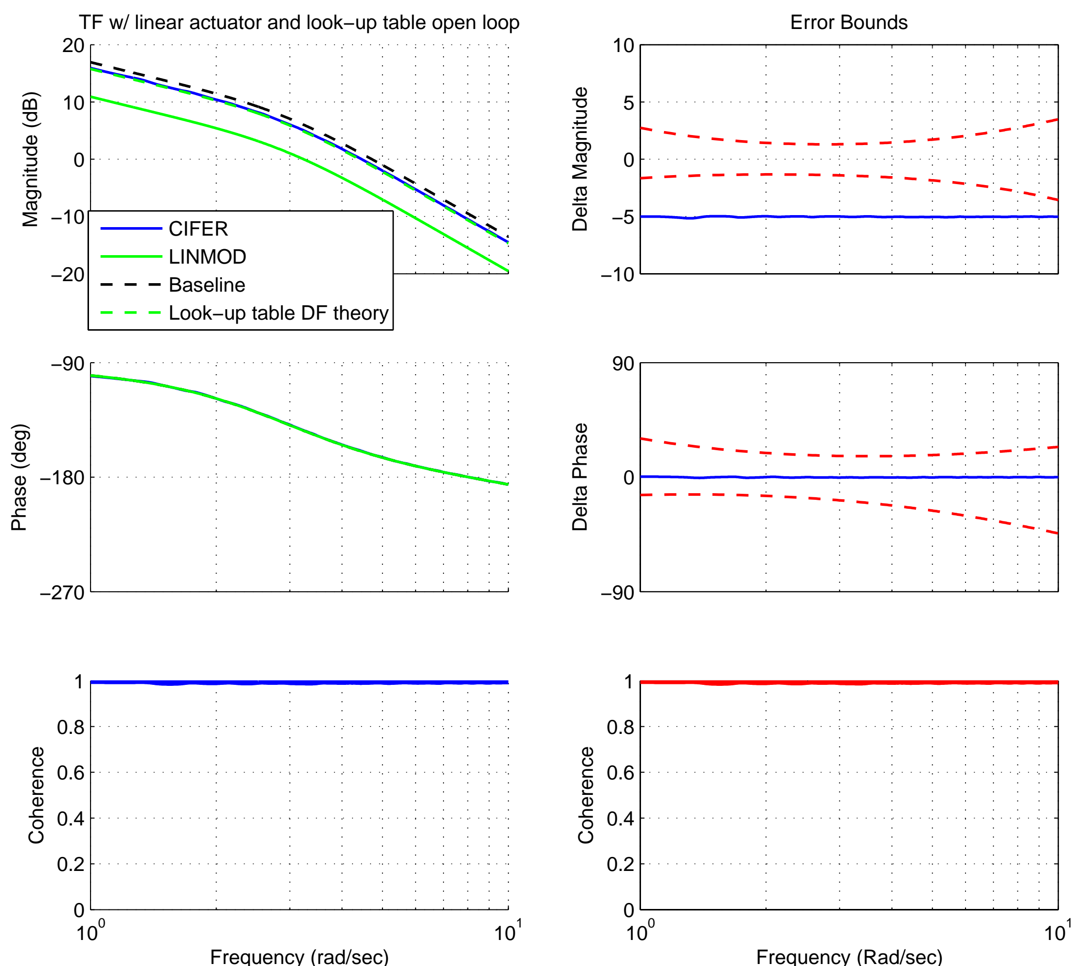

Figure 30: Open loop lookup table validation

For this case both CIFER and LINMOD register a change. CIFER and the theoretical describing function curves match very well. The magnitude drop from the baseline is very small because the effective gain is some weighted average of the two slopes. Compared with LINMOD, CIFER shows a 5 dB decrease in gain margin, a 16 degrees decrease in phase margin, and a 1.2 rad/sec increase in crossover frequency.

LINMOD however shows a greater decrease in magnitude because the gain it uses is only the k_1 slope. This is due to the perturbation analysis. A larger perturbation would raise the LINMOD magnitude curve up closer to the CIFER and baseline curves. The

key to note here is the perturbation analysis by default will only capture the slope near the linearization point. If it is expected that the linearized model to represent the simulation over larger amplitudes than the default LINMOD perturbation size, then the effect needs to be accounted for separately during linearization.

6. Time delay

Equivalent time delays can be included to model high frequency dynamics and computational delays that are not explicitly simulated. Time delay is a critical element to be modeled correctly because phase errors due to time delay grow with frequency. For flight control, time delays must be accurate because the phase loss due to time delay has large effect on phase and gain margins.

There is a linear relationship between the frequency and the phase drop. The complex value for time delay is defined as:

$$N_{timedelay}(\tau, \omega) = e^{-\tau \cdot i\omega} \quad (29)$$

In Simulink, time delay blocks have the option to approximate it with a Pade approximation when performing linearization. The Pade approximation is a re-ordering of a Taylor series into a rational expression. The Taylor series is of $N_{timedelay}$. For the purpose of this validation, a second order Pade approximation was used. Higher order approximations are valid over a longer frequency range, and second order is enough in this case.

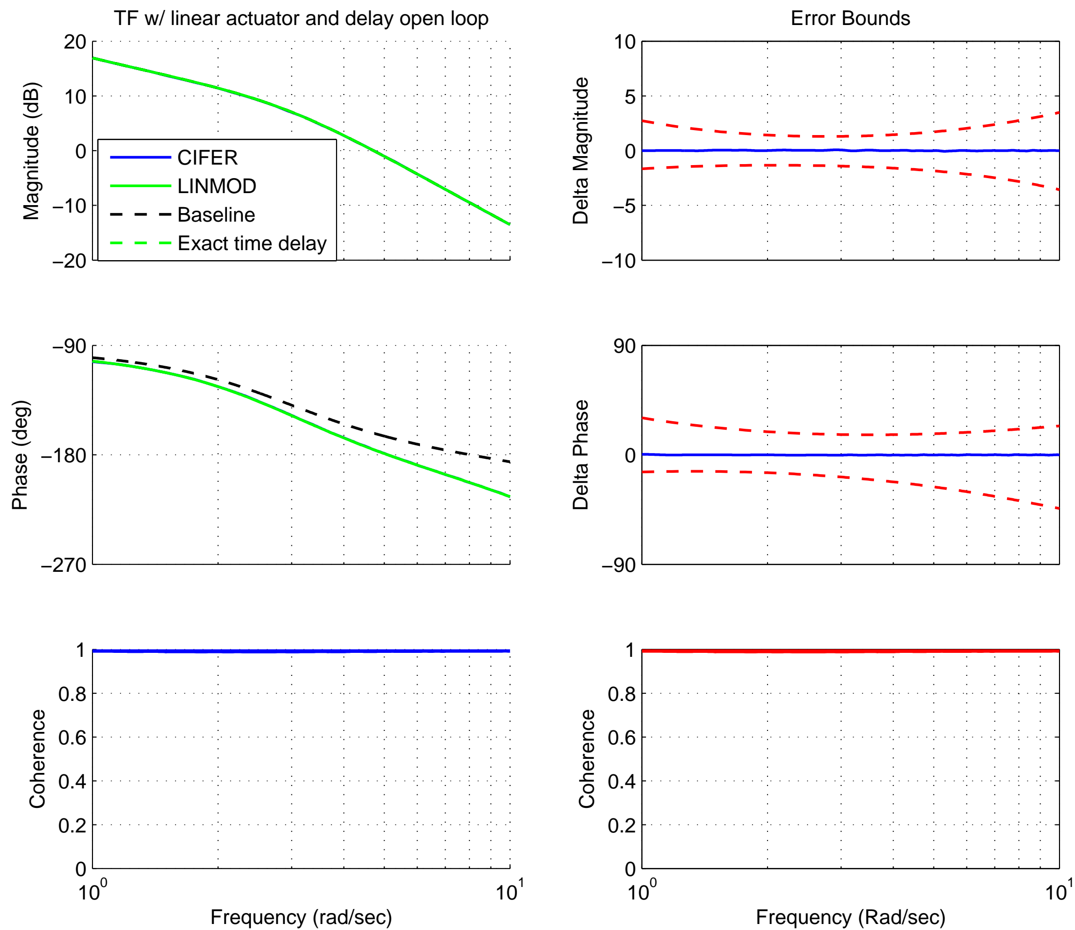

Figure 31: Open loop time delay validation with non-zero Pade order

In the figure above it can be seen that the magnitude curves for all cases agree very well. The phase curves from CIFER, LINMOD and the analytical equation given above all show the growing phase loss with no magnitude drop. Looking over at the error plots, there is no phase error between the CIFER and LINMOD responses. Because of the good agreement between the curves, the gain margin, phase margin, and crossover frequency agree very well between the CIFER and LINMOD responses.

This agreement indicates that time delay elements are safe for linearization if and only if the Pade order is greater than zero. However in Simulink, the default Pade order

is zero. Below is a validation result when the Pade order is set to zero. This removes the time delay from the linearization.

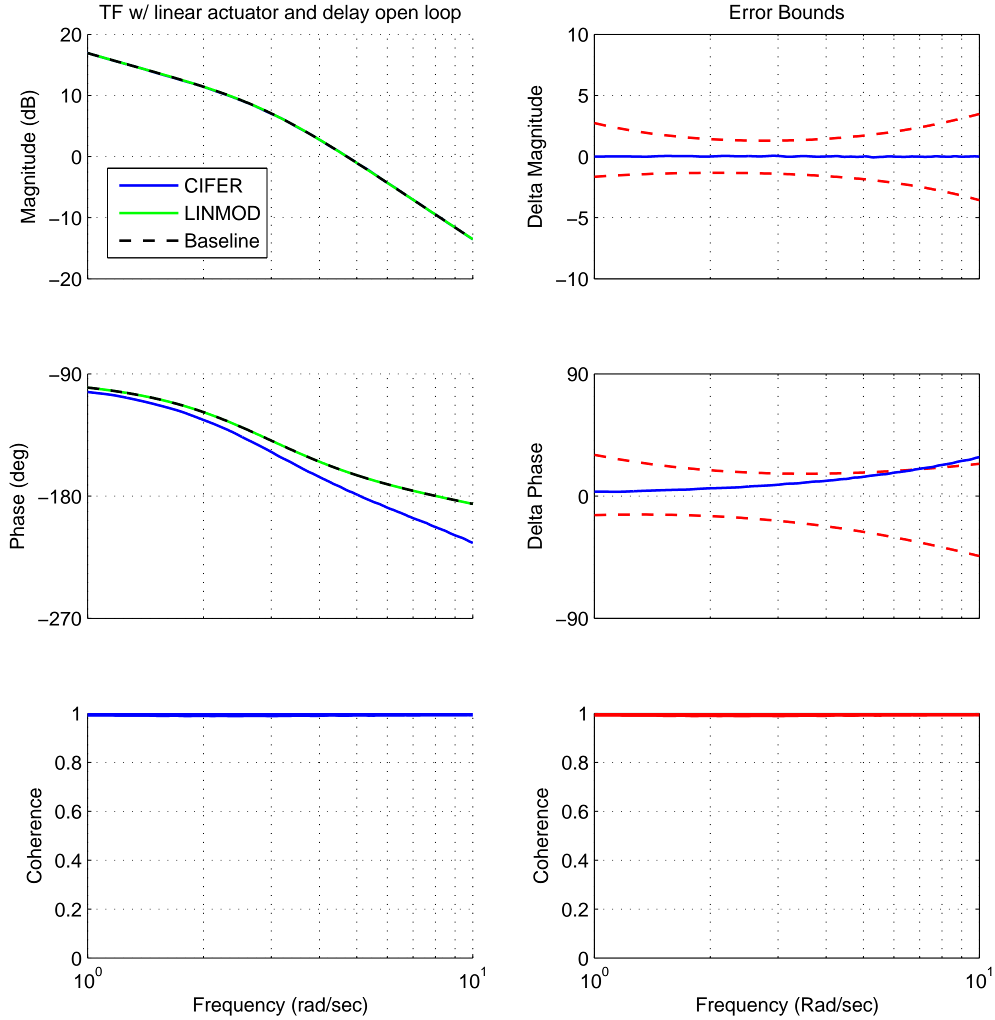

Figure 32: Open loop time delay validation with Pade order zero

The figure above shows phase error between LINMOD and CIFER due to the missing delay. The baseline and LINMOD curves are directly overlaid, indicating that LINMOD is treating the time delay as a unit gain. Around 6 rad/sec the phase error is great enough to exceed the MUAD bounds. Therefore, it is very important for the user to ensure that the Pade order is not set to zero in order to have agreement between the time-

domain simulation and the linearized model. Future work might include making a diagram scanner to check that the Pade order is non-zero.

7. Memory block and unit delay

The memory block and unit delay are elements that can be used in the simulation model to represent computation delays or to solve algebraic loop errors. They delay the output one integration step from the input, so they function like time delays. The difference between a time delay and a memory block or a unit delay is that the amount of delay is a function of the sample time. The difference between memory blocks and unit delays are how Simulink treats their sample time. Memory blocks also have the option "Treated as unit delays" during linearization.

Because memory block and unit delays are like time delays their effect on the open loop frequency response is only a function of the time step and frequency. The function for these elements is based on the time delay equation:

$$N_{memory}(dt, \omega) = e^{-dt \cdot i\omega} \qquad (30)$$

Below is the validation result with a memory block modeled after the actuator. The CIFER curve shows a phase delay behind the LINMOD and baseline responses. Because the "Treat as unit delay" option is unchecked in the memory block, LINMOD is treating it as a unit gain. The delay difference between CIFER and LINMOD using the estimate correction feature in the validation tool shows a delay of .015 seconds, around 1.5 dt.

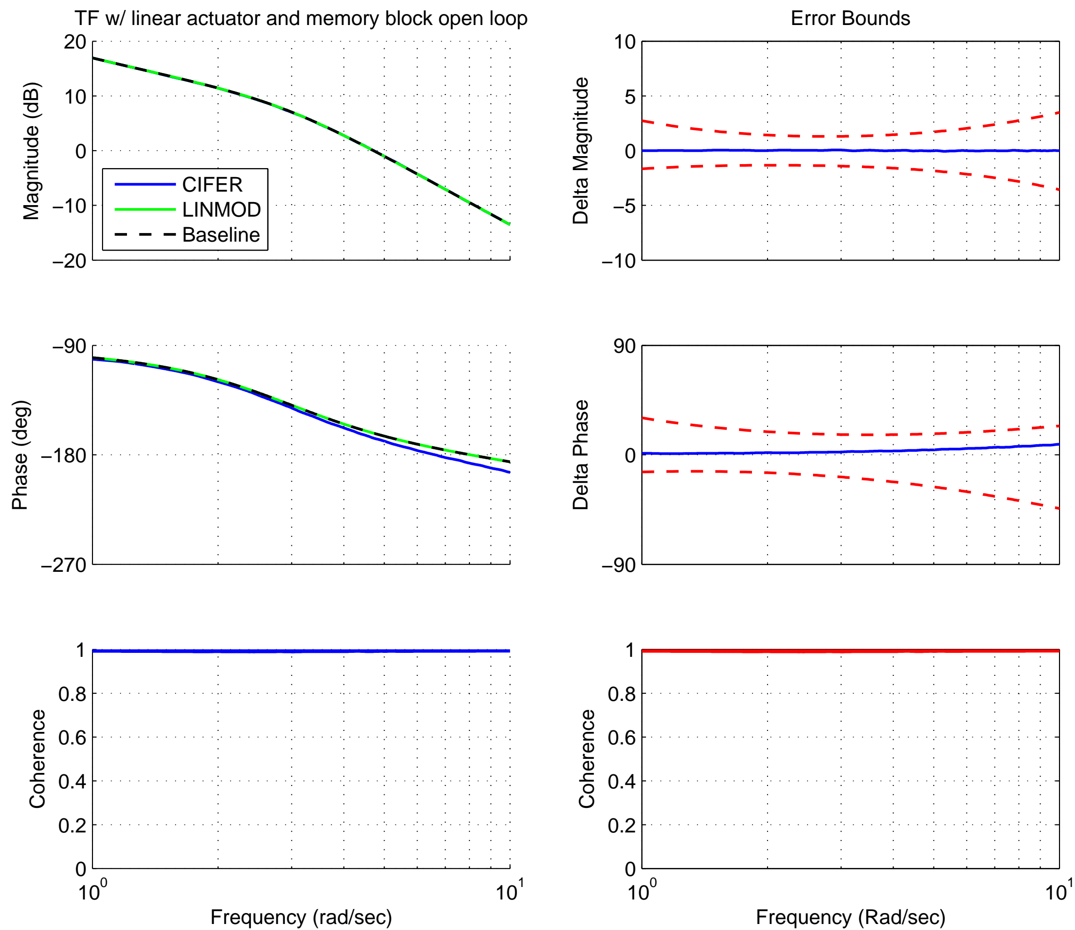

Figure 33: Open loop memory validation

Comparing against LINMOD, CIFER shows 3.5 dB decrease in gain margin, and a 4 degree decrease in phase margin. Because there was no change in the magnitude curve, there was no change in the crossover frequency.

This particular discrepancy in linearization can be resolved, because memory blocks behave similarly to time delays, but the latter have the Pade approximation option for linearization. Time delays of 1 dt can replace memory blocks to make the two responses agree. If the memory block was being used to prevent an algebraic loop error, time delays can fulfill that function as well.

8. Rate limiting

Another nonlinearity that is intrinsic to real systems is rate limiting. No actuator can slew infinitely fast, so rate limiters are used to account for this. The describing function of a rate limiter is a function of the rate limit, frequency and input amplitude.

Before a certain frequency, called the onset frequency, the rate limiter has no effect. Beyond the onset frequency there is a transition region where the phase will roll off faster until the limit frequency is reached. Beyond the limit frequency the phase loss asymptotically approaches 90 degrees. Below are an example effect of rate limiting and the describing function for rate limiting (Ref 4.):

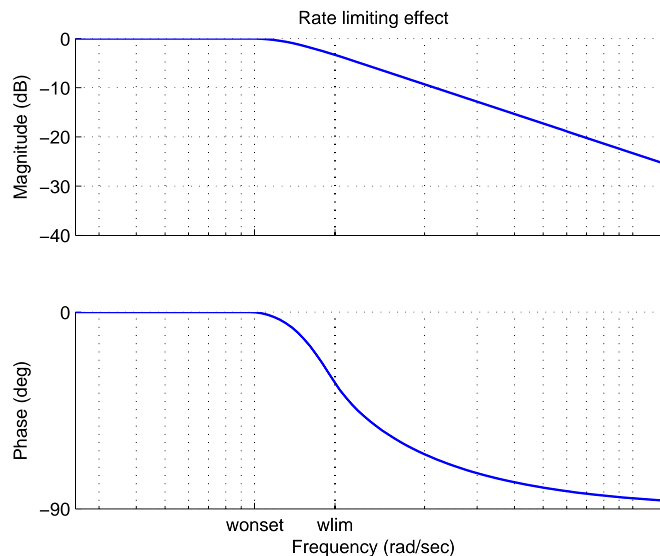

Figure 34: Rate limiting effects

$$w_{onset} = \frac{\mathrm{R}}{\mathrm{A}}$$

$$w_{\lim} = \frac{\sqrt{\pi^2 + 4}}{2} w_{onset}$$

$$N_{rate_\lim}(R, w, A) = \begin{cases} w \geq w_{\lim} \Rightarrow \dfrac{4 w_{onset}}{\pi \cdot w} e^{\left[\frac{-j \arccos\left(\frac{\pi}{2} w_{onset}\right)}{w}\right]} \\ w \leq w_{onset} \Rightarrow 1 \\ w_{\lim} > w > w_{onset} \Rightarrow spline\left(N_{rate_\lim}(R, w_{onset}, A) \rightarrow N_{rate_\lim}(R, w_{\lim}, A), w\right) \end{cases}$$

(31)

Beyond the limit frequency, the phase will have rolled off by about 90 degrees, a fairly significant amount. Most systems do not carry more than 40-50 degrees of phase margin, so a loss of ~90 degrees of phase at the crossover frequency is unacceptable.

Below is the validation result with the rate limiter. The LINMOD curve is directly over the baseline curve, implying that LINMOD is treating the rate limiter as a unit gain. The CIFER and rate limit theoretical describing function curves are directly overlaid, so CIFER is correctly capturing the describing function of the system. For the chosen amplitude of the sweep, the onset frequency is ~5.5 rad/sec. In the error plots both the magnitude and phase exceed the MUAD bounds by ~6 rad/sec.

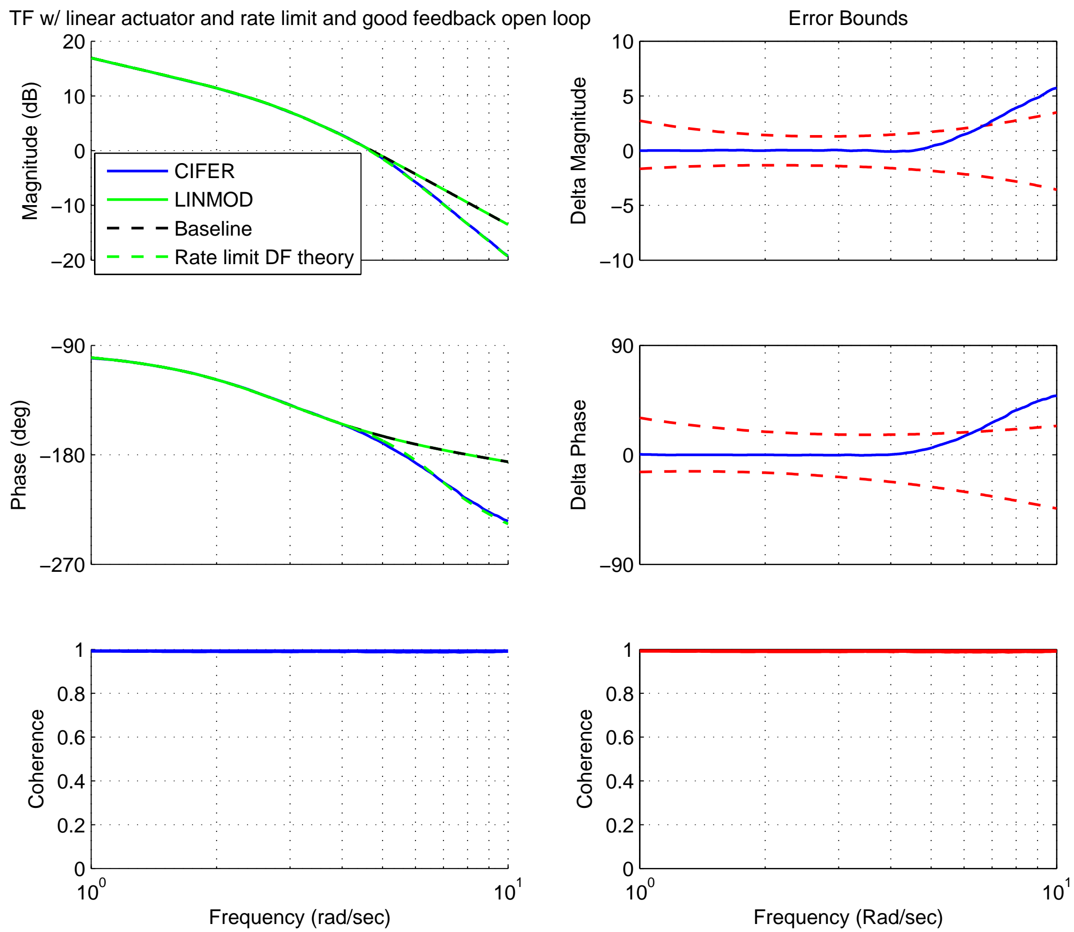

Figure 35: Open loop rate limit validation

Compared against LINMOD, the CIFER response shows a 5.5 dB decrease in gain margin, and a 3 degree decrease in phase margin. The crossover frequency did not change much because the onset frequency was just after crossover.

Below is the time history record before and after the rate limiter from the CIFER sweep used to generate the frequency responses above. At around 50 seconds the rate limiting effect becomes active.

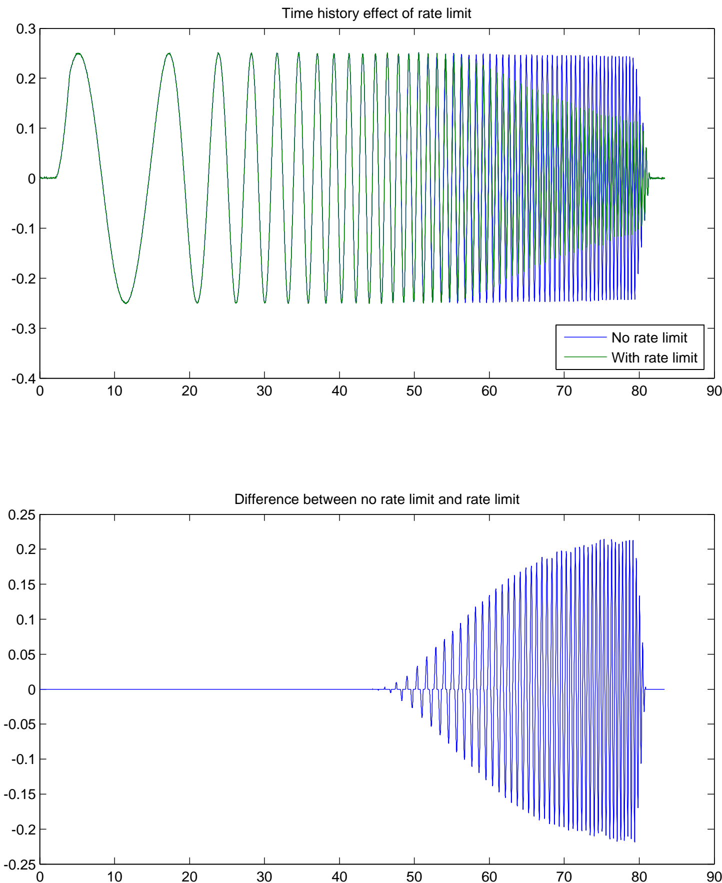

Figure 36: Time domain effect of rate limiting case

LINMOD is not able to handle rate limiting correctly. The validation tool can indicate that rate limiting is happening, but the effect of this nonlinearity needs to be analyzed in some other way.

The reason it is so important to capture the effects of rate limiting is it has been observed in almost all PIO cases (Ref 14.). To capture this within CONDUIT, the Open Loop Operating Point (OLOP) specification is used, which is a predication of pilot

induced oscillations due to rate limiting, allowing the user to capture the rate limiting effect (Ref 4.). It utilizes the onset frequency given maximum stick deflection to determine if pilot induced oscillation (PIO) is likely.

B. Table of results

Table 2: Percent change between CIFER and LINMOD in open loop (+%: LINMOD>CIFER)

Case	Gain Margin	Phase Margin	Crossover frequency
Baseline	0.19%	0.36%	-0.08%
Hysteresis	N/A	N/A	N/A
Dead zone	-21.57%	-28.44%	16.16%
Saturation	-8.92%	-12.85%	5.42%
Lookup table	47.21%	77.79%	-27.40%
Memory block	58.17%	29.84%	-0.05%
Time delay	-2.03%	2.20%	-0.03%
Rate limiting	140.28%	22.45%	0.62%

For the open loop elements, CIFER captured every describing function almost identically, showing that it is a good truth model for the validation. LINMOD was only able to match CIFER for the case with nonlinearities and time delay. Most of the nonlinearities were ignored (treated as a unit gain). Hysteresis was linearized as a zero gain, making it an element that must be bypassed before linearization. Memory blocks were the only nonlinearity that could be replaced to allow LINMOD to match the CIFER responses. The lookup table element was linearized about the trim flight condition, which may or may not be the correct effect, depending on the need.

VI. Analysis of simple feedback flight control systems

The previous section dealt solely with open loop systems. In this section, two closed loop examples will be examined using the validation tool. The first example is a follow on to previous section, where the loop is closed around the simple 2nd order transfer function with two different compensators. The same nonlinearities as in the previous section will be tested and examined using the validation tool. Because of the loop closure there will be no theoretical describing function analysis to accompany the CIFER and LINMOD results. The second example is based on a simplified XV-15 control system. Closed and broken loop validations will be performed for the on-axis channels.

A. Closed loop validation for simple 2nd order system

To demonstrate and gather some more baseline results for comparison, closed and broken loop results were generated for the nonlinearities used in the previous section. In order to show the effect of different feedbacks, two loop closures were designed, a "Good design" with ~44 degrees of phase margin and a "Bad design" with ~18 degrees of phase margin. Below is a summary of the gain and phase margins for the two feedbacks.

Table 3: Simple 1/2 closed loop summary

	Good	Bad
Gain Margin (dB)	17.35	9.57
Phase Margin (deg)	44.17	17.89
Crossover freq (rad/sec)	2.6	4.7

By looking at the step responses shown below, the good design has a better behaved response.

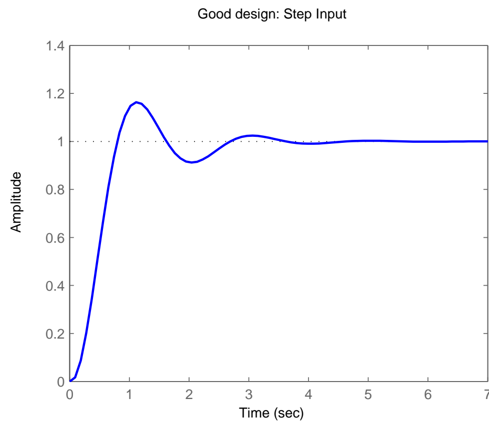

Figure 37: Step response of the good design

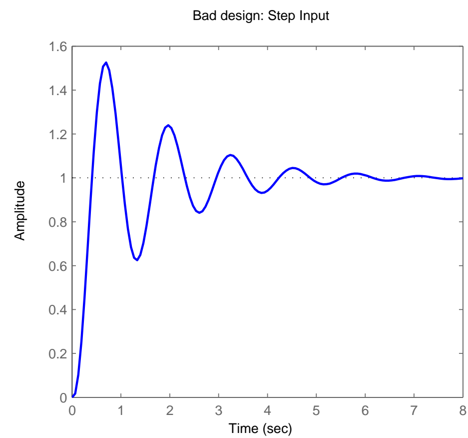

Figure 38: Step response of the bad design

The majority of the validation results are similar to the open loop results. All validation results with no nonlinearities show excellent agreement between CIFER and LINMOD. The closed loop response for the good design is shown below. The complete set of plots of the validation results are in appendix A.

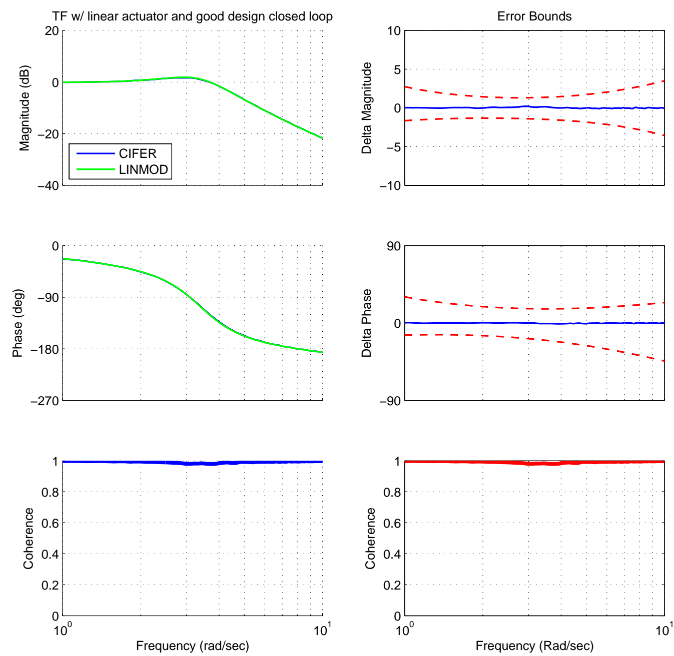

Figure 39: Closed loop (good design)

Shown below are the closed and broken loop responses with hysteresis. Like the open loop case, LINMOD linearizes the hysteresis as a zero and so that response is not visible in the figures below. The baseline response is the response in the absence of nonlinearities. In this case based on the CIFER analysis, the good design shows the effect of the hysteresis more than the bad design because the amplitude of the feedback is smaller.

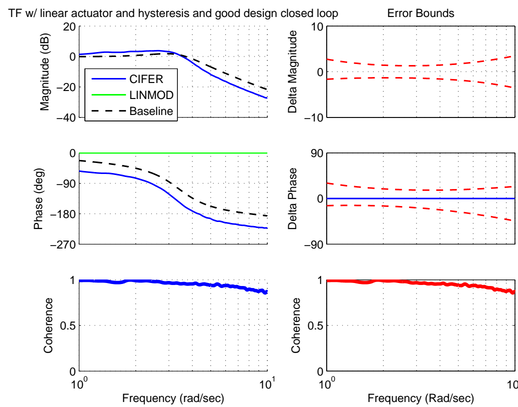

Figure 40: Hysteresis closed loop (good design)

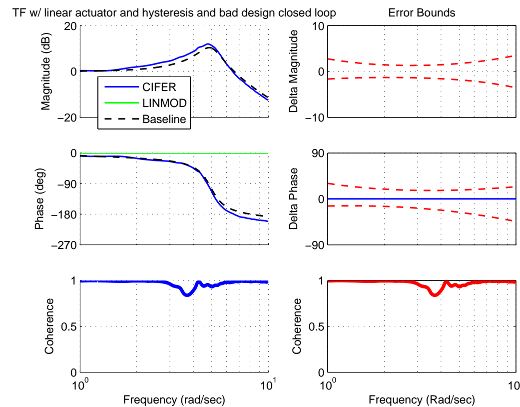

Figure 41: Hysteresis closed loop (bad design)

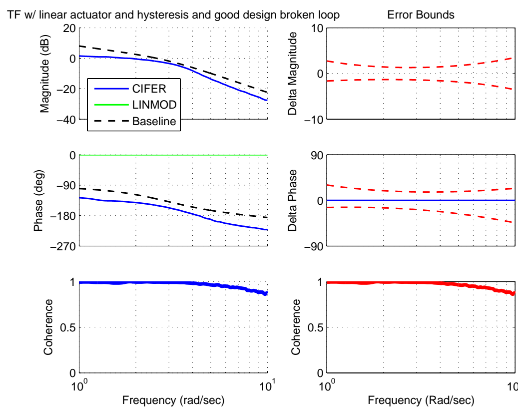

Figure 42: Hysteresis broken loop (good design)

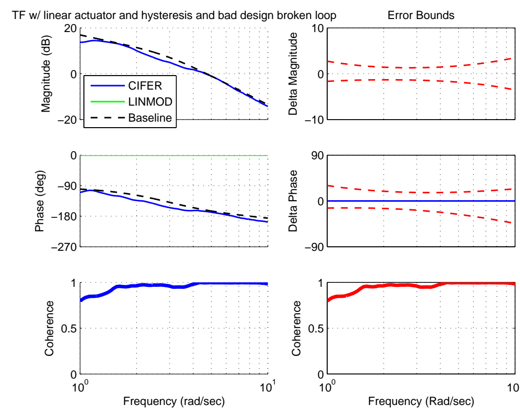

Figure 43: Hysteresis broken loop (bad design)

The dead zone results are shown below. Like in the open loop results, dead zone is treated as a unit gain by LINMOD, so the results overlap directly with the baseline results. LINMOD treats saturation, rate limiters, and memory blocks in the same way, so these plots are not shown here. See appendix A for the remainder of the plots.

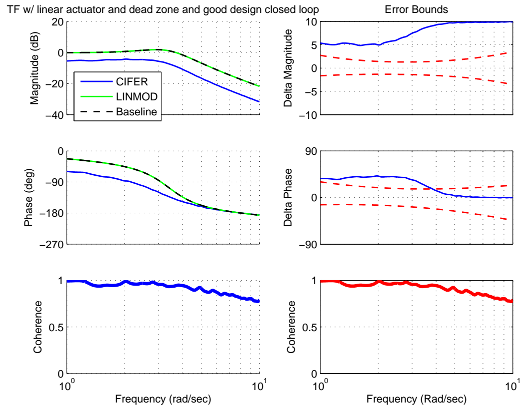

Figure 44: Dead zone closed loop (good design)

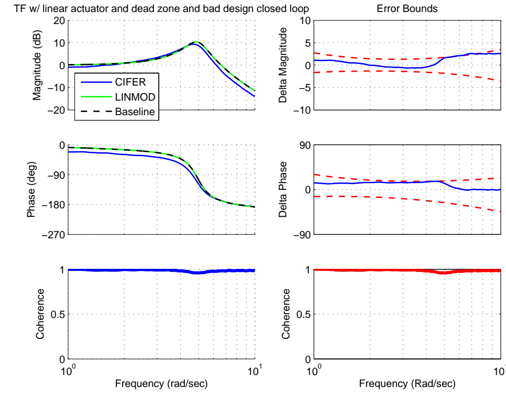

Figure 45: Dead zone closed loop (bad design)

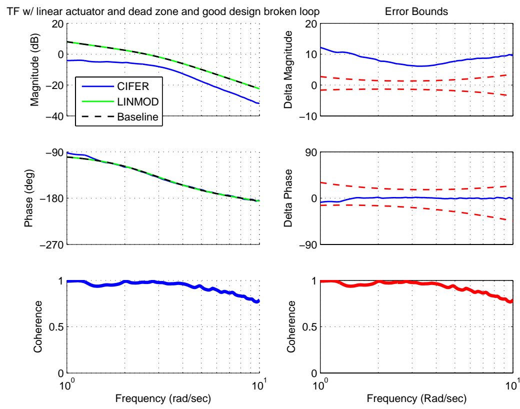

Figure 46: Dead zone broken loop (good design)

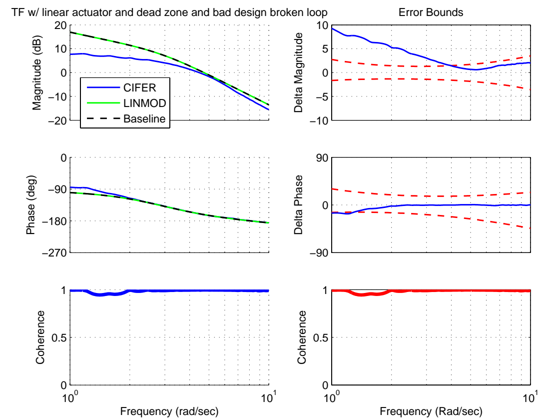

Figure 47: Dead zone broken loop (bad design)

Shown below are the results with lookup tables. Unlike the other nonlinearities besides time delay, the LINMOD response changes from the baseline. Here CIFER is closer to the baseline response because the effective gain of the lookup table is close to one. LINMOD on the other hand only sees the smaller gain near the origin, hence the shift in magnitude seen in the broken loop response and the large change in the closed loop response. This is an example where validation tool would generate valuable data

because some lookup tables will not need special treatment, and the validation tool will quickly show if the lookup table needs to be replaced by a gain for analysis. In cases where the lookup has a dramatic effect, the system might need to be optimized at both gain levels (either simultaneously or sequentially), depending on the setup (i.e. gain scheduling, in-detent/out-detent mode specs, etc).

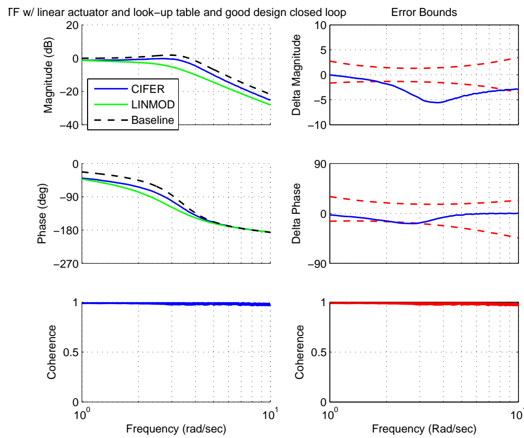

Figure 48: Lookup table closed loop (good design)

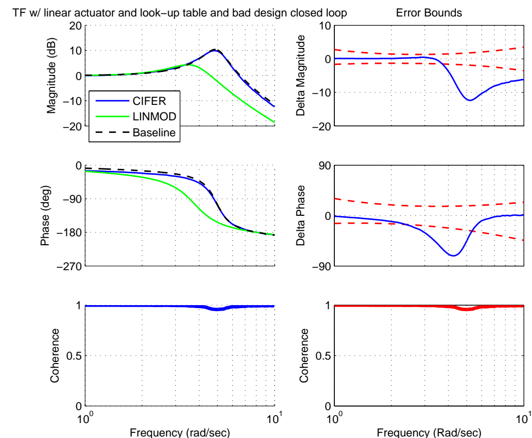

Figure 49: Lookup table closed loop (bad design)

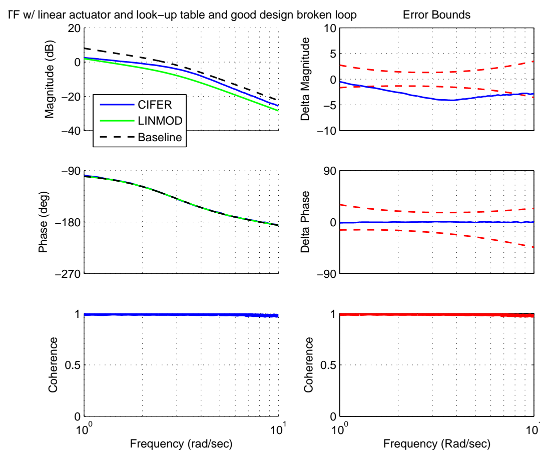

Figure 50: Lookup table broken loop (good design)

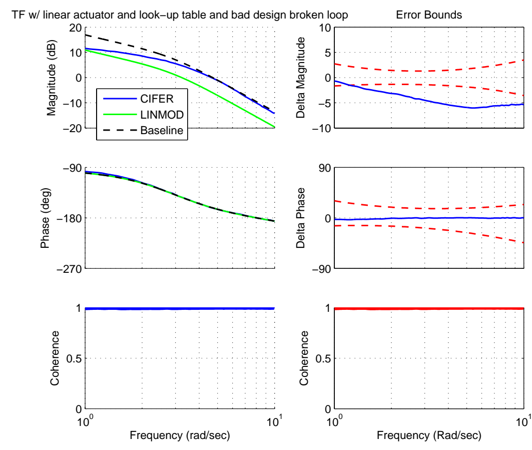

Figure 51: Lookup table broken loop (bad design)

Below are the results for the good design system with a time delay in closed loop. As seen, CIFER and LINMOD agree very well in the frequency range of interest. Like

the results with no nonlinearity, only the good design closed loop result is shown because the other plots all show similar good agreement.

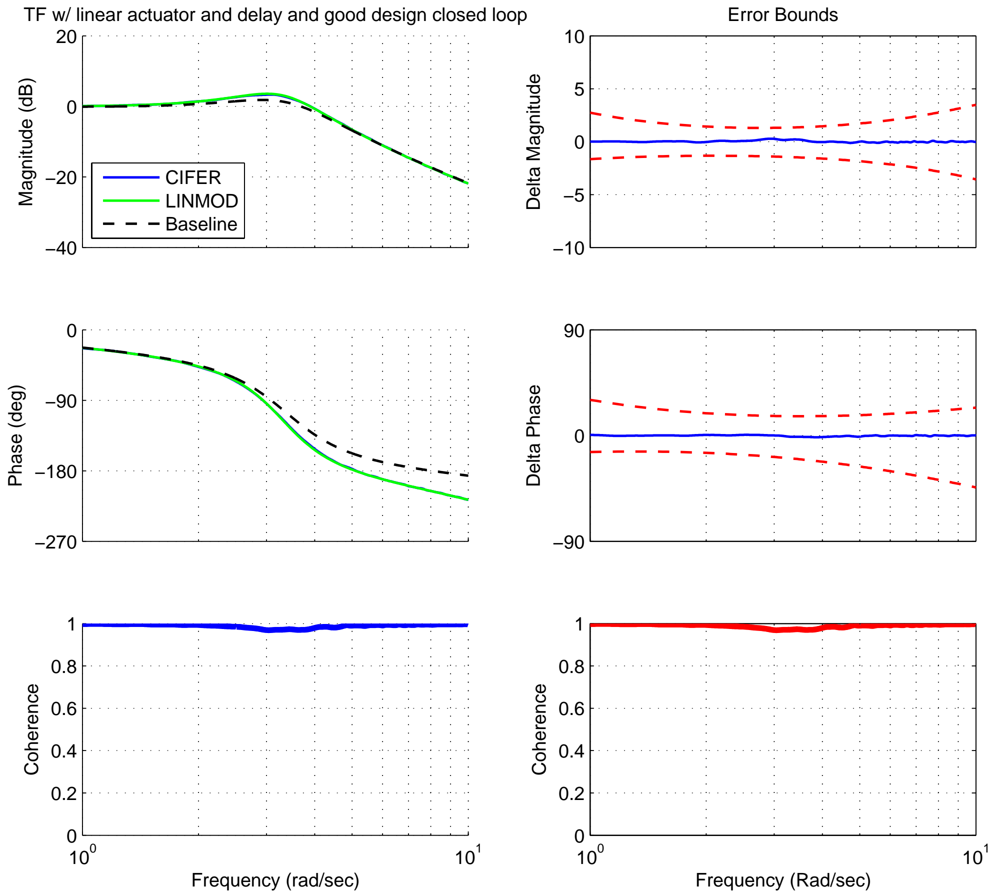

Figure 52: Delay closed loop (good design)

Below is a table summarizing the percent differences between frequency domain metrics as computed from the CIFER frequency responses and the LINMOD frequency responses. The complete table with original values can be found in appendix D. Note that a positive percentage means LINMOD has a higher value than CIFER, or is more liberal. A negative percentage means LINMOD has a lower value than CIFER, or is more conservative.

The baseline errors are all within 1.5%, so consider less than 5% change a match. The hysteresis rows are all N/A because the LINMOD responses were all zeros, so no metrics were found.

Table 4: Summary of Percent metric changes between CIFER and LINMOD (+%: LINMOD>CIFER)

Case	Gain Margin	Phase Margin	Crossover frequency	Bandwidth
Baseline (good design)	-1.32%	-0.27%	0.91%	-0.01%
Baseline (bad design)	0.29%	-0.25%	0.41%	-0.19%
Hysteresis (good design)	N/A	N/A	N/A	N/A
Hysteresis (bad design)	N/A	N/A	N/A	N/A
Dead zone (good design)	-33.06%	N/A	N/A	18.72%
Dead zone (bad design)	-16.24%	-10.68%	5.71%	4.81%
Saturation (good design)	2.21%	1.01%	4.17%	14.64%
Saturation (bad design)	-20.85%	-48.17%	28.64%	31.01%
Lookup table (good design)	14.21%	9.69%	-23.41%	-30.42%
Lookup table (bad design)	53.83%	101.65%	-30.33%	-27.62%
Memory block (good design)	22.37%	4.38%	0.88%	-0.78%
Memory block (bad design)	56.22%	29.13%	0.55%	-0.23%
Time delay (good design)	-0.74%	-0.44%	1.07%	-0.13%
Time delay (bad design)	-1.04%	-1.42%	1.47%	-0.72%
Rate limit (good design)	34.55%	-1.59%	1.48%	4.00%
Rate limit (bad design)	3.61%	18.61%	0.33%	-0.37%

For both of the dead zone cases the magnitude loss causes the gain margin in the CIFER responses to increase. The good design's magnitude dropped completely below 0 dB, so the phase margin and crossover frequency metrics were not found.

For saturation, the good design did not saturate as much as the bad design, so the margins did not change much between CIFER and LINMOD. The bad design saturated a lot, causing a large magnitude drop, resulting in LINMOD showing lower gain and phase margins, but higher crossover frequency and bandwidth.

The lookup table cases showed dramatic variations, especially in the bad design. The LINMOD responses showed higher gain and phase margins, but lower crossover frequency and bandwidth than the CIFER responses.

The memory block cases shows little change in crossover frequency, which is expected given that it should only have a phase effect in the broken loop. However effect on gain and phase margin is pretty significant for the bad design. The effect on bandwidth is minimal in this case because the magnitude changes were small.

The time delay cases show very little metric changes, which is to be expected because the Pade approximation used by LINMOD for time delays was seen to accurately capture the effect on phase.

The rate limiting cases show changes in gain and phase margins, but little change in crossover frequency and bandwidth. Most the margins are larger in LINMOD, except for phase margin in the good design.

Ultimately in this exercise, only two cases were correctly captured by LINMOD, the case without nonlinearities and the case with time delay. Hysteresis resulted in a zero in the linearized model, so these elements must be removed or by-passed. Lookup tables are accounted for, but LINMOD may exaggerate the effect due to the perturbation analysis, so care must be taken. The lookup table may need to be removed and different scenarios may be needed to capture the different gains provided by the lookup table.

Saturation, dead zone, memory blocks and rate limiters are completely ignored, so any effect they have on the system will be neglected. As mentioned earlier, memory blocks can be replaced with time delays, so the delay effect can be captured in the linearized model. Rate limiters will be examined in more details in the next example.

B. XV-15

The XV-15 is a tilt-rotor aircraft with which AFDD has many research experiences. The block diagram used is of the XV-15 in hover with a stability augmentation system (SAS) for lateral and directional axes only. The bare airframe model is a plant identified from flight test data processed using CIFER (Ref 2.). The actuator models are 2nd order transfer functions with rate limiting and saturation block elements. The two input axes in the model, lateral cyclic and directional cyclic, each have a PID SAS. The lateral stick is limited to 4.8 stick inches of throw, and the pedals are limited to 2.4 stick inches of throw. A representation of the block diagram is shown below.

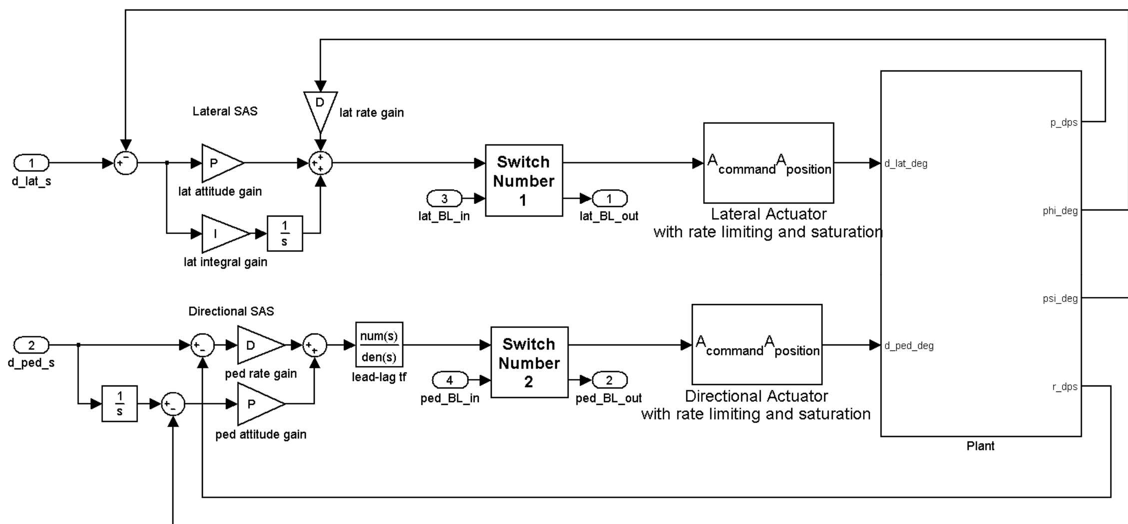

Figure 53: XV-15 Block diagram

The two modeled nonlinearities, saturation and rate limiting, are more active with large amplitudes, so small inputs should minimize any differences between CIFER and LINMOD. Saturations have no effect unless the input amplitude is greater than the saturation limit. Rate limiting is both amplitude and frequency dependent. The onset frequency of the rate limiter is inversely proportional to the input amplitude. Because the validation tool only looks at a specific frequency range, a small sweep size should eliminate any differences between CIFER and LINMOD responses.

Below are the broken loop frequency responses in lateral and directional axes. The amplitudes are specified in percentage of full throw. For the smallest amplitude (0.2%), the CIFER frequency response does not show the effect of the rate limiter and agrees with LINMOD. However, the second (15%) and third (50%) amplitude show the characteristic magnitude and phase loss beyond the onset frequency of the rate limiter. As the amplitude of the input increases, the onset frequency continues to decrease, as expected.

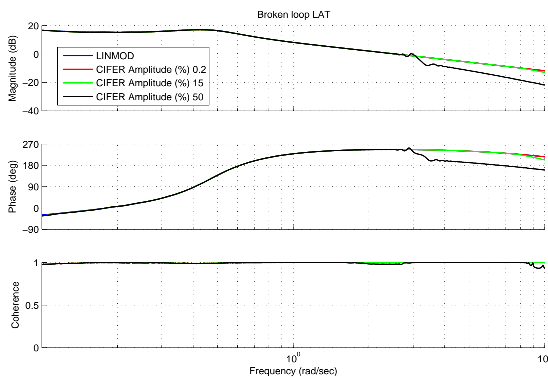

Figure 54: Lateral broken loop

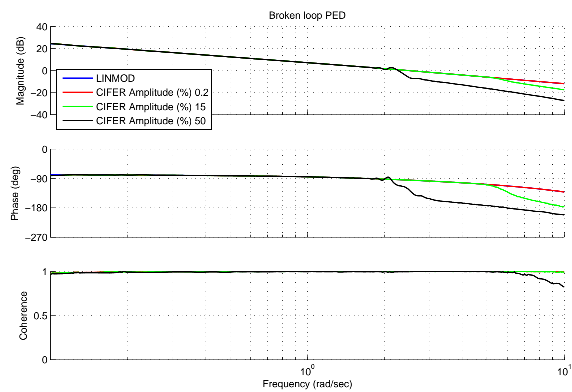

Figure 55: Direction broken loop

In the closed loop responses shown below, the onset frequency of the rate limiting is basically the same because the relative amplitude sweep sizes are the same. However, in the closed loop response the phase loss is more pronounced than in the broken loop.

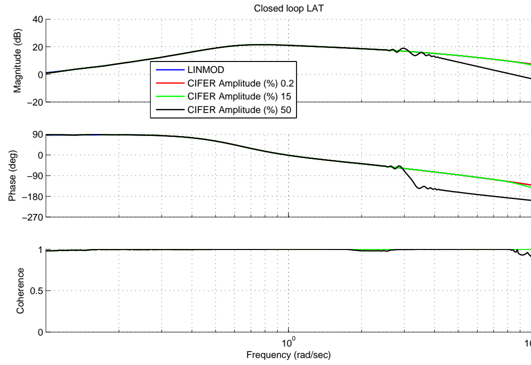

Figure 56: Lateral closed loop

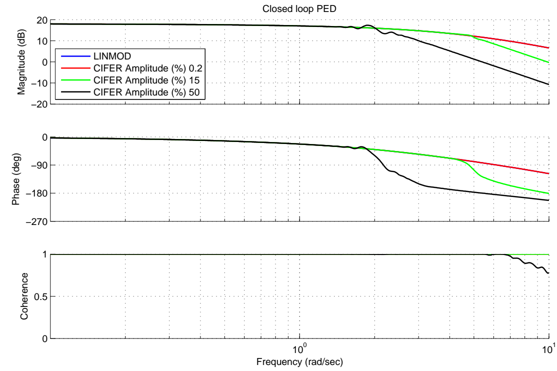

Figure 57: Directional closed loop

In both the broken and closed loop responses, the effect of the rate limiter on the frequency response is dramatic. In the broken loop there is ~70 degrees phase loss after onset, and in the closed loop ~90 degrees phase loss after onset. As seen in the simple 2nd order case, LINMOD treats the rate limiter as a unit gain in the buildup of the linear model. However, due to the large effect on phase, the rate limiter is an effect that must be accounted for in the analysis.

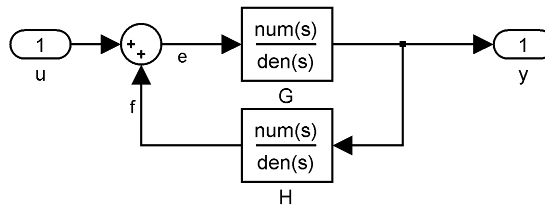

Figure 58: Broken loop setup

In a pathological case, the sweep amplitude was intentionally increased to 200% maximum deflection to demonstrate a weakness in the validation tool. When performing broken loop identification, the response to identify is feedback over error, $\frac{f}{e}$. When

using CIFER, this is done by feeding an input sweep into u and measuring e and f. Ideally the result is $\frac{f}{e} = GH$, which is the broken loop. However $\frac{f}{e}$ is really:

$$\frac{f}{e} = \frac{1}{\frac{e}{u}} - 1 \tag{32}$$

For $\frac{f}{e} = GH$ to be true, $\frac{1}{\frac{e}{u}}$ must not be zero. When the actuator encounters rate limiting the system self excites in a limit cycle. $\frac{1}{\frac{e}{u}}$ is effectively zero because there is no relationship between the external input and actual response. The result is CIFER identifies $\frac{f}{e} = -1$, with a coherence of 1. The math explains why this answer is correct, but this is not the broken loop frequency response. Because this result is clearly not the correct response, when using the validation tool it should be noticeable when this error occurs. The validation tool also has a check for trim at the beginning and end of the time history record that could catch a problem like this.

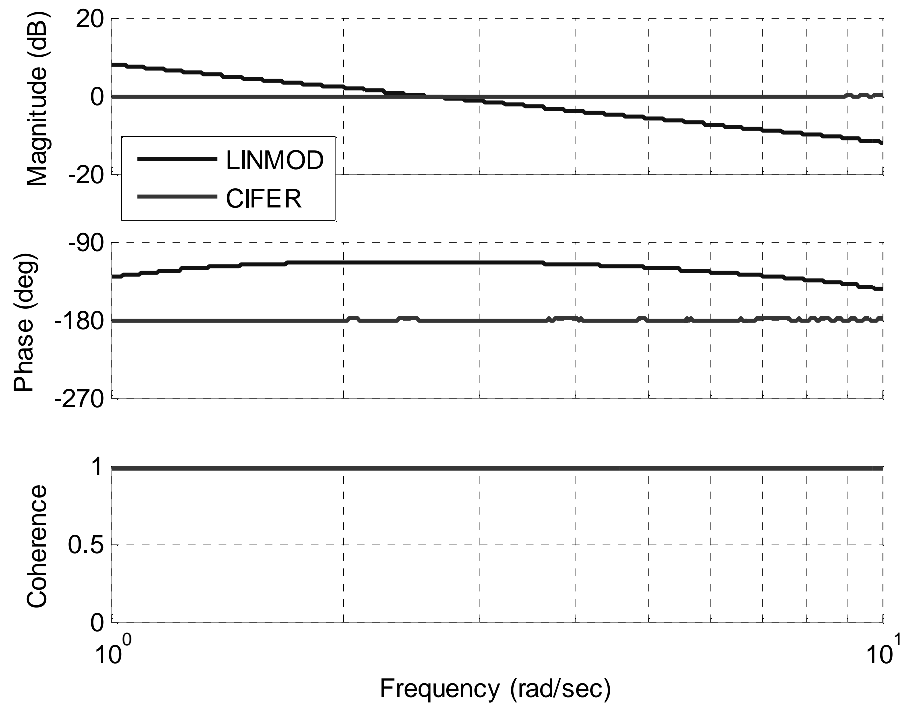

Figure 59: Unstable broken loop

This error is made possible because of the simplified validation process that is utilized by the validation tool. In the normal interactive operation, the time history files are checked for strange behavior before the identification process begins and therefore this issue would have been caught early in the validation process.

The results from these two closed loop example validations have confirmed what was seen in the open loop example. First, the CIFER and LINMOD agree very well in the absence of nonlinearities or a time delay. Second, CIFER is capable of capturing the describing functions of systems with nonlinearities. And third, it can be seen that LINMOD either ignores or miscomputes the majority of the nonlinearities tested.

VII. Simplification and verification of a comprehensive flight control system

In the previous examples, the systems were intentionally chosen to demonstrate the validation tool in environments where the results were known to ensure that the tool was functioning properly. The next example is based on the highly complex RASCAL fly-by-wire flight control system (Ref 8.). The RASCAL helicopter is a UH60 used for experimental fly-by-wire control systems and in-flight simulation at AFDD.

The RASCAL block diagram ultimately is converted to flight code to go on the aircraft, so it represents a level of complexity found in real-world systems. However, the diagram's complexity made it prohibitive to work with. This section will mainly deal with the simplified version of the diagram and its verification. In the end, both block diagrams were used in this example, the original more complicated diagram is denoted as the "full-up" diagram, and the simplified diagram is denoted as the "simplified" diagram.

A. Block diagram simplification

The full-up RASCAL block diagram has 33627 blocks, with multiple modes and complex switch networks to account for different flight conditions. The bare airframe model in the analysis is either an identified model from flight test data using CIFER or a physics based model. The physics based airframe model is a 25 state state-space respresntation. Of the 33000, 21937 of the blocks are just organizational element, SubSystems, Inports, and Outports.

The reason for all this complexity is that the block diagram is design to be auto-coded using The MathWorks Real-Time Workshop®. The flight ready code from Real-Time Workshop has been installed on the aircraft and flight tested. The block diagram

has been validated by the RASCAL team to behave exactly as the final flight control system will, so ideally it provides accurate results during analysis.

The drawback to all this complexity is added cost of computational speed. The block diagram takes almost a minute to load, several seconds to start a time domain simulation, and over 11 minutes on a high-end machine for a single performance evaluation in CONDUIT. There are Stateflow® logics to handle gain scheduling and mode transition, which requires special handling in the CONDUIT case to get the diagram into different modes for analysis. The linear analysis requires the use of a more advanced linearization algorithm in MATLAB called LINEARIZE, that allows the model to "fly" to the trim point before performing the linearization.

The idea behind the simplified diagram is ultimately to reduce the high block count due to both organization and control logic that are unnecessary from a control design perspective. If the broken loop responses of both models are close in the frequency range of interest, then from an analysis stand-point the diagrams can be used interchangeably for control design purposes. The full-up diagram can still be auto-coded, but the optimized set of gains is obtained from the CONDUIT analysis based on the simplified diagram.

The simplified block diagram was provided by AFDD for the study. This thesis will present the simplified model's improvements and the verification of the simplified diagram against both the full-up model and flight test data. The core of the validation tool was used to generate broken loop frequency responses for the full-up and simplified diagrams, and the new CIFER command line interface was used to process the flight test data. The scripts for these analyses are in Appendix B.

The simplified diagram managed to capture the key dynamics present in the full-up model with a total of 1886 blocks, more than an order of magnitude of reduction. The amount of organizational overhead dropped as well, with only 706 SubSystem, Inports and Outports, less than half of the total count. Below is a table comparing block counts between the full-up and simplified diagram.

Table 5: Block Summary

	Full-up	(%)	Simplified	(%)
Total	33627	100.00%	1886	100.00%
Abs	38	0.11%	-	0.00%
BusCreator	41	0.12%	-	0.00%
BusSelector	100	0.30%	-	0.00%
Concatenate	3	0.01%	-	0.00%
Constant	1679	4.99%	188	9.97%
DataTypeConversion	484	1.44%	-	0.00%
Demux	52	0.15%	12	0.64%
Display	67	0.20%	4	0.21%
EnablePort	77	0.23%	-	0.00%
From	1491	4.43%	179	9.49%
Gain	487	1.45%	103	5.46%
Goto	1008	3.00%	96	5.09%
Ground	346	1.03%	26	1.38%
Inport	9330	27.75%	310	16.44%
Integrator	20	0.06%	40	2.12%
Logic	521	1.55%	10	0.53%
Lookup	40	0.12%	-	0.00%
Lookup2D	10	0.03%	-	0.00%
Math	21	0.06%	8	0.42%
Memory	5	0.01%	1	0.05%
MinMax	82	0.24%	-	0.00%
Mux	39	0.12%	8	0.42%
Outport	5108	15.19%	297	15.75%
Product	833	2.48%	136	7.21%
RelationalOperator	427	1.27%	-	0.00%
Rounding	9	0.03%	-	0.00%
Scope	16	0.05%	2	0.11%
Selector	2	0.01%	2	0.11%
SignalConversion	5	0.01%	-	0.00%
SignalSpecification	400	1.19%	-	0.00%
Signum	2	0.01%	-	0.00%
Stateflow	12	0.04%	-	0.00%
Step	103	0.31%	25	1.33%
SubSystem	7499	22.30%	99	5.25%
Sum	876	2.61%	119	6.31%
Switch	1567	4.66%	98	5.20%

	Full-up	(%)	Simplified	(%)
Terminator	394	1.17%	37	1.96%
TransferFcn	2	0.01%	36	1.91%
TransportDelay	4	0.01%	25	1.33%
Trigonometry	12	0.04%	25	1.33%
TruthTable	3	0.01%	-	0.00%
UnitDelay	412	1.23%	-	0.00%

The simplified diagram is smaller and has a significant speed improvement over the full-up diagram. The Stateflow logic was removed, so the system was loaded at trim and there was no longer a need to "fly" to a trim point while the Stateflow logic initialized. For this reason, LINMOD can now be used instead of LINEARIZE.

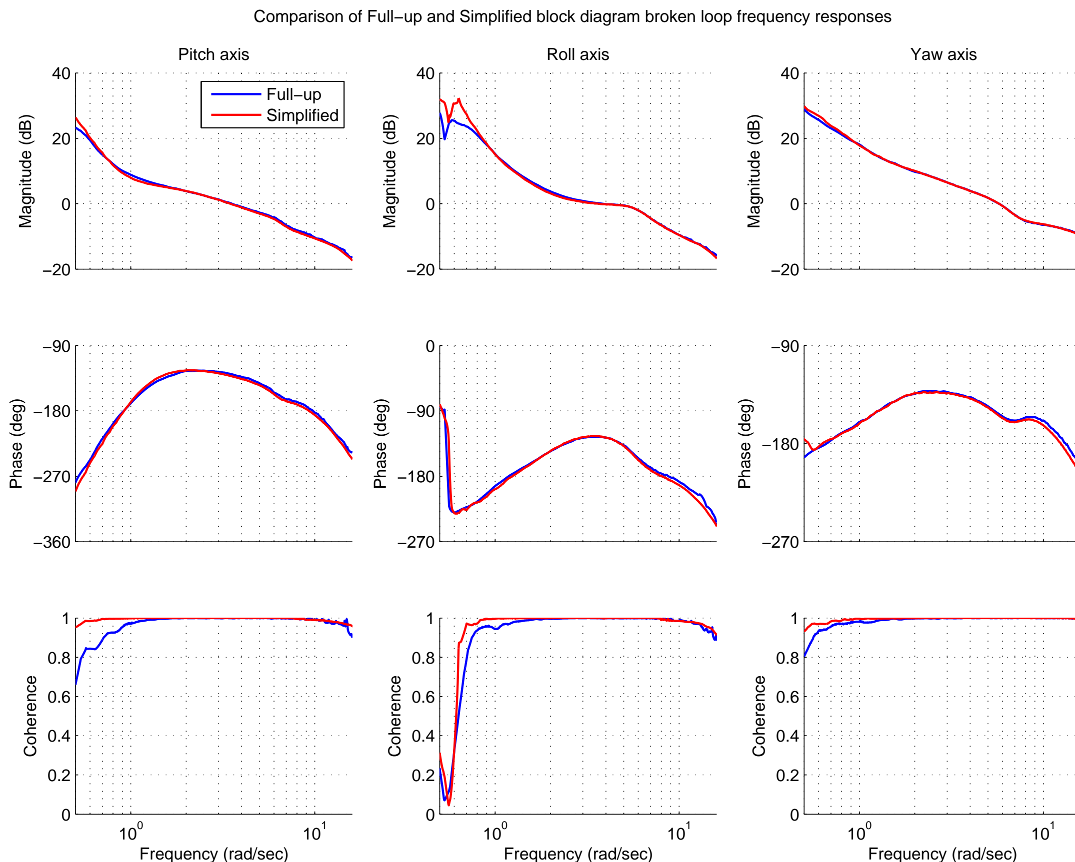

Figure 60: Full-up vs. Simplified Broken Loop

Figure 60 above shows the broken loop frequency responses generated from sweeping both the full-up and simplified block diagrams. Both systems are in good agreement from .5 to 16 rad/sec. At high frequency, the simplified diagram has a higher phase roll-off. There is a small phase delay difference of ½ time step present due to the fact that the simplified diagram is continuous, and the full-up diagram is hybrid. The phase delay makes the simplified diagram more conservative in most axes. The small difference in crossover frequency in the roll axis is due to the flatness near crossover, allowing very small changes in magnitude to have an effect on crossover frequency. The changes in stability margins are summarized in the table below.

Table 6: Stability margin changes between diagrams

Axis	Diagram	Crossover Frequency (rad/sec)	Phase Margin (deg)	Gain Margin (dB)	180 Crossing (rad/sec)
Pitch	Full-up	3.5	50.29	-9.86, 10.03	0.9, 9.8
	Simplified	3.5	48.00	-9.05, 9.97	0.9, 9.3
Roll	Full-up	3.8	53.83	-11.51, 7.80	1.2, 8.7
	Simplified	3.6	55.06	-10.25, 6.68	1.2, 8.0
Yaw	Full-up	5.7	26.96	-23.89, 7.89	0.7, 13.2
	Simplified	5.7	25.54	-26.15, 7.54	0.6, 12.4

To compare both systems in time domain, step inputs were injected into the three attitude axes. In the figures below, the rate responses to the corresponding step inputs are shown. In the title of each plot, the RMS cost, as well as Theil inequality coefficient (TIC) for the error between the two diagrams are displayed. A cost less than 1 and a TIC less than .25 are considered excellent for model matching (Ref 2.).

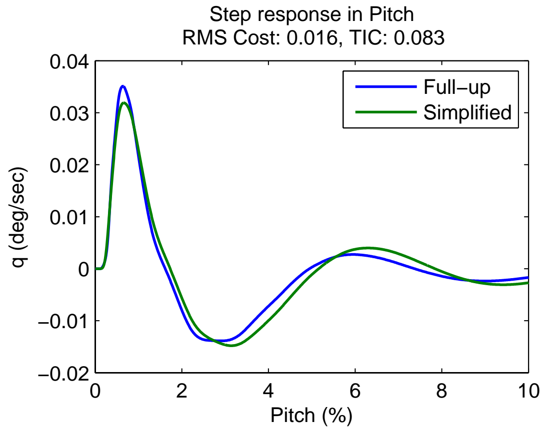

Figure 61: Step response in Pitch

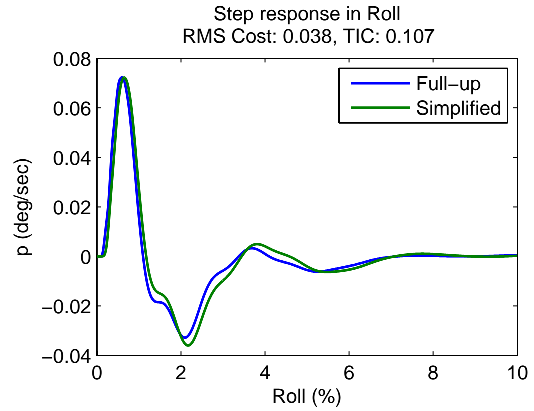

Figure 62: Step response in Roll

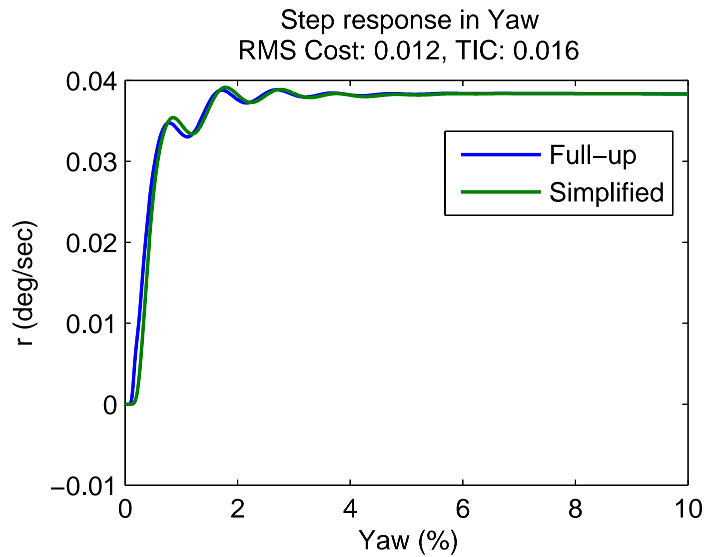

Figure 63: Step response in Yaw

Given the good agreement from the frequency responses, stability margins, and step responses, the simplified diagram can be considered as a valid simplification of the full-up diagram. With the reduced computation cost the new CONDUIT case is easier to work with and was optimized in a reasonable amount of time.

B. Comparisons between the simplified block diagram and flight test data

Another check that needs to be done is the validation between the simplified diagram and the aircraft based on flight test data. These results are based on a new set of gains, so the stability margins have changed. Below are the Bode plots of the pitch axis broken loop response, overlaying the two cases on the left and error plot on the right. It can be seen that the simplified diagram is representative of the aircraft from about 2 rad/sec to 14 rad/sec in the pitch axis. The crossover frequency is near 3 rad/sec, so the low frequency magnitude needs improvement. The phase curves are in good agreement across the frequency range.

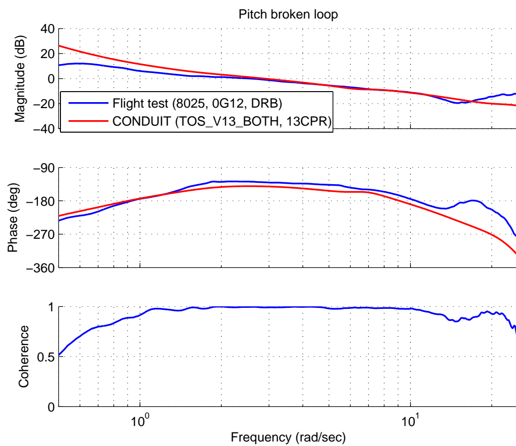

Figure 64: Pitch broken loop

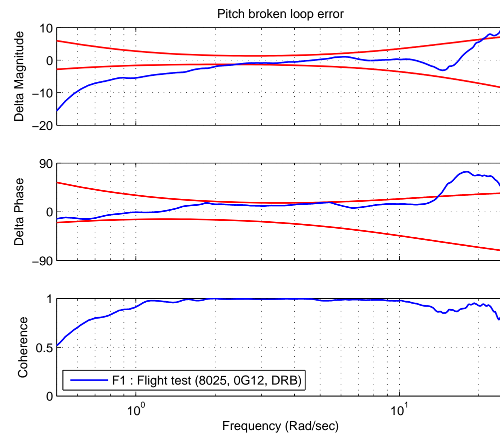

Figure 65: Pitch broken loop error

Below are the roll broken loop responses. Similar to the pitch axis, the high frequency region has a better match, with magnitude errors at low frequency. The good fit range is from 3 rad/sec to 15 rad/sec. The fact that the errors near crossover frequency are exceeding the MUAD bounds is problematic, so the low frequency magnitude needs improvement.

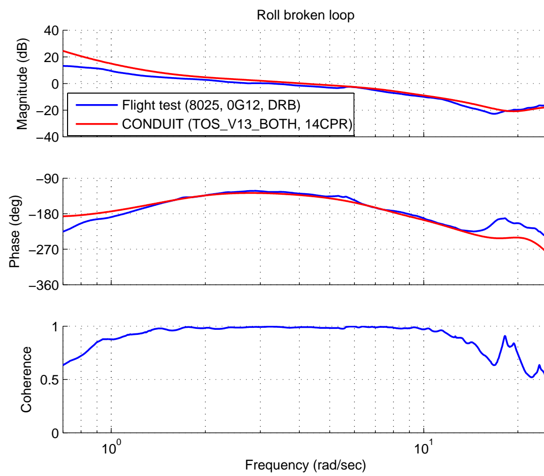

Figure 66: Roll broken loop

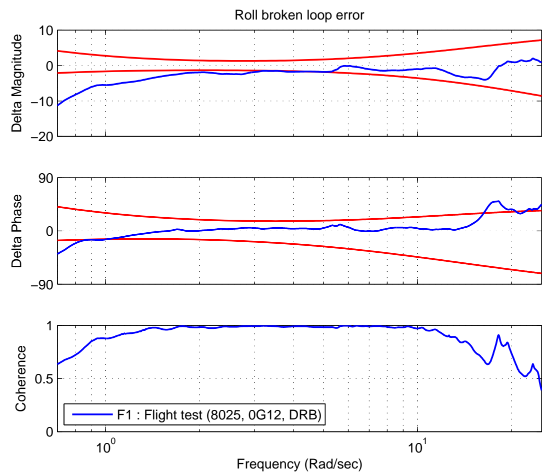

Figure 67: Roll broken loop error

Below is the yaw broken loop results. Unlike pitch and roll, the yaw low frequency matchs up to 1.6 rad/sec, but the phase and magnitude errors at high frequency are much more dramatic than the pitch and roll results. The crossover frequency of the yaw channel is also at a higher frequency, 4.39 rad/sec, so the high frequency errors are going to affect the accuracy of the analysis of the yaw axis more.

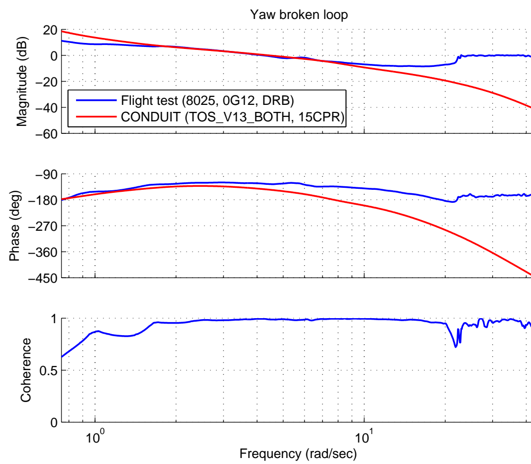

Figure 68: Yaw broken loop

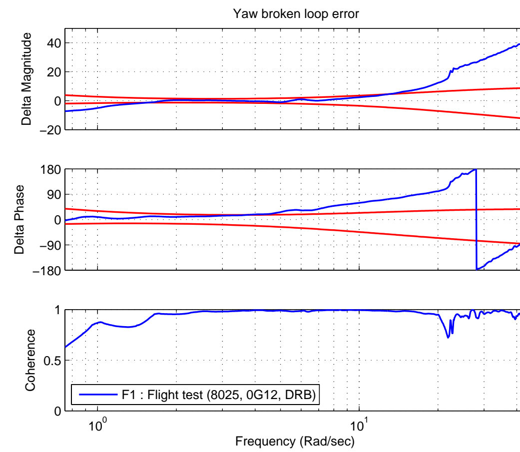

Figure 69: Yaw broken loop error

Below is a table comparing the stability margins obtained from CIFER against that from the flight test data. The stability margin differences are significant in all of the axes. This indicates that the current model is insufficient in some way. Because of the close agreement between the full-up and simplified diagrams, the control system portion of the block diagram is probably not the problem. Both the full-up and the simplified diagrams rely on the same math models, so the simplification would not have affected that portion of the diagram. That indicates that the problem is a math model issue.

Table 7: Flight test margin summary

Axis	Diagram	Crossover (rad/sec)	Phase Margin (deg)	Gain Margin (dB)	180 Crossing (rad/sec)
Pitch	Flight	2.5	51.77	-7.07, 11.67	0.9, 10.5
	Simplified	2.9	38.94	-13.12, 10.35	0.9, 9.1
Roll	Flight	3.3	55.19	-7.71, 8.55	1.1, 8.9
	Simplified	4.1	46.65	-17.94, 6.22	0.9, 8.3
Yaw	Flight	4.2	56.05	-11.04, 3.5	0.8, 22.1
	Simplified	4.4	35.17	N/A, 5.6	N/A, 7.7

The discrepancies seen here were documented and provided to the principle investigator, who confirmed that the math modeling inaccuracies accounted for the discrepancies. These errors were corrected by adding a gain and time delay to the block diagram. Below are some of the new broken loop response overlays based on the updated block diagram. Further flight test data from the recently published AHS paper shows a much better match between the aircraft with two flight test records (9009 and 8025) and the updated CONDUIT block diagram (Ref 8.).

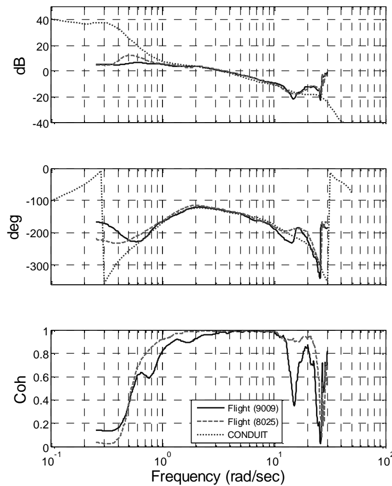

Figure 70: Corrected pitch broken loop

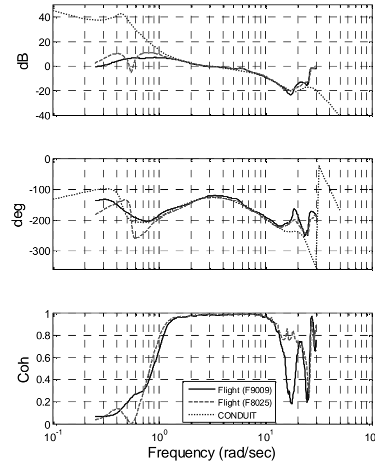

Figure 71: Corrected roll broken loop

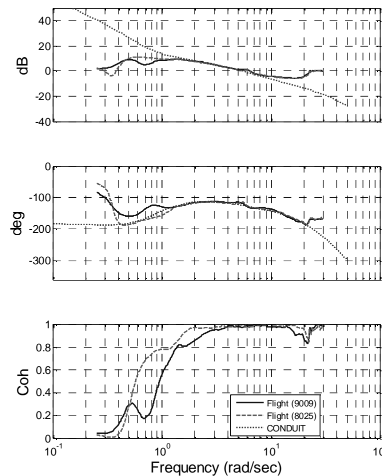

Figure 72: Corrected yaw broken loop

VIII. Validation of linearized model of a simplified flight control system model

The previous section was completely focused on frequency responses generated from CIFER based on frequency sweeps. After correcting the problems found between the simplified model and the aircraft, it is necessary to check that the linearization of the new diagram is correct. Like the previous sections, the comparisons are between frequency responses generated from LINMOD to those generated from CIFER, to ensure the integrity of the linear model obtained with LINMOD.

Broken loop responses in the three attitude channels were checked. Below is the pitch broken loop. Agreement is excellent across the entire frequency range, but there is a small phase error between the two responses, which will be discussed later.

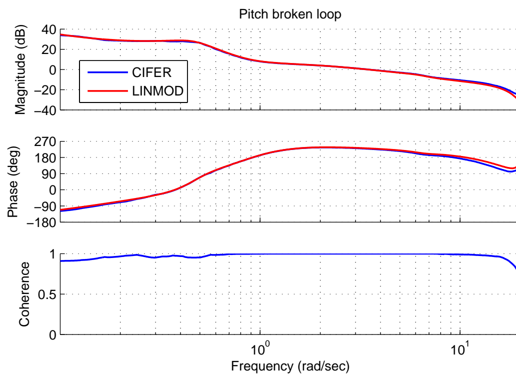

Figure 73: Pitch broken loop

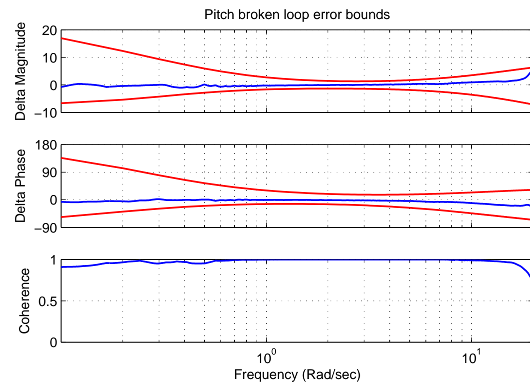

Figure 74: Pitch broken loop error

Figure 75 and Figure 76 show the roll broken loop comparisons between CIFER and LINMOD. Unlike pitch, there appears to be some errors at ~.5 rad/sec. This is the location of a lightly damped mode. Lightly damped modes typically require much long window sizes to capture, which is impractical. The location of the root is very close in both CIFER and LINMOD, so there is no reason to expect linearization errors there. At frequencies below the lightly damped mode the magnitude curve matches again. However, the phase appears to have a large error. This is just due to phase wrapping, which is confirmed on the error plot that there is no real problem. At high frequency there is a similar phase roll off as found in pitch.

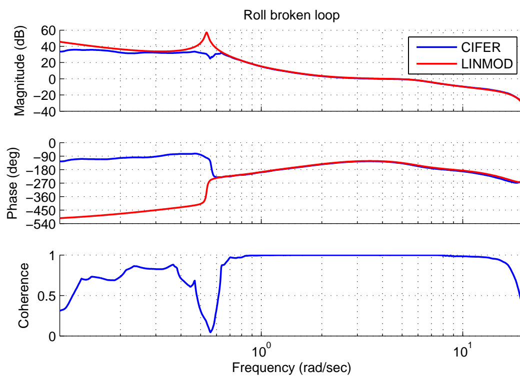

Figure 75: Roll broken loop

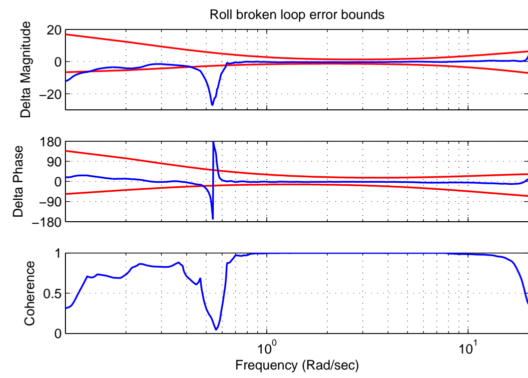

Figure 76: Roll broken loop error

In the yaw axis, the same small high frequency phase roll-off is present as in pitch and roll. Additionally, in these figures at the low frequency errors there are some problems with the CIFER frequency responses as indicated by the drop in coherence. This is limitation of the validation tool due to drift in the system. However, the range of good identification is from .4 to 20 rad/sec, which is more than enough to compare validation results against CONDUIT.

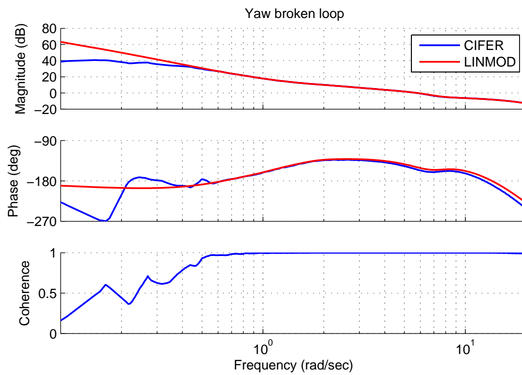

Figure 77: Yaw broken loop

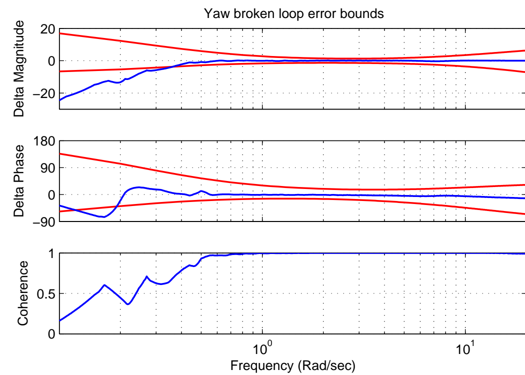

Figure 78: Yaw broken loop error

Since each of the axes has a additive phase delay, there must be some elements in the block diagram that is causing the discrepancy. Below is a table summarizing the estimated time delay errors in each channel. The fact that the delays are close to 1-2 time step is the first hint to the problem. Looking back to the validation work for the open loop simple 2nd order system, this error looks very much like an error due to a memory block.

Table 8: Summary of found time delays

Axes	Time delay (msec)	N*dt
Pitch	20.8	2.56
Roll	10.0	1.28
Yaw	11.2	1.4

Looking through the simplified block diagram, memory blocks were indeed found in front of the system plant, as show in the figure below. As discussed earlier, if the memory block is replaced with a time delay of 1 time step, the time domain simulation will be unaffected and the linearized model will capture the delay correctly.

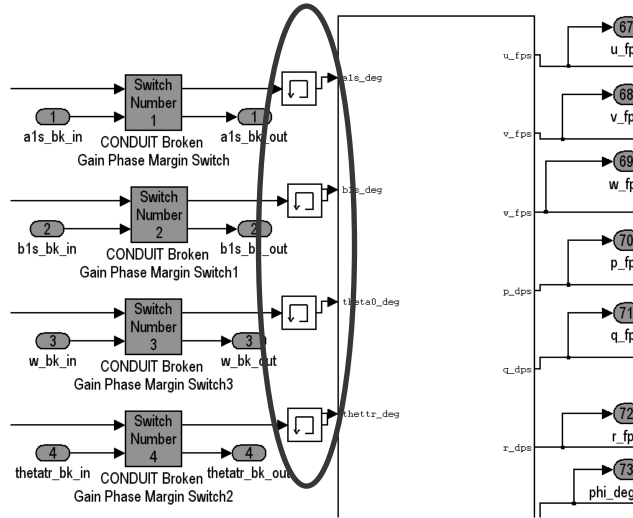

Figure 79: Memory blocks in path

After replacing the memory blocks with time delays, the estimated delay error decreased by ~10 ms, although each channel still had phase errors at high frequency. After examining the diagram further it was found that several time delays in the feedback path had their Pade approximation order set to zero. This means that LINMOD was completely ignoring the effect of the time delay, and hence the error showed up during the validation.

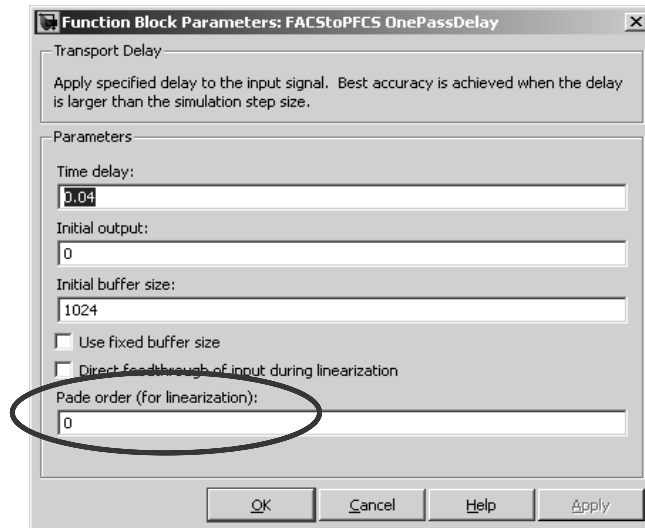

Figure 80: Pade order zero

It was not clear why the Pade order was left at the default value of zero. Perhaps the engineer that developed the Simulink block diagram forgot to set the order to two, or there may have been some other engineering reason for leaving the order at zero. Regardless, the setting caused LINMOD to ignore the time delay. This error introduces a small, but present difference between the linearized model and the full simulation model. By replacing the memory blocks with time delays and setting the Pade order to non-zero in the existing time delays, the CIFER and LINMOD results agree perfectly in the frequency range of interest, as seen in the figures below.

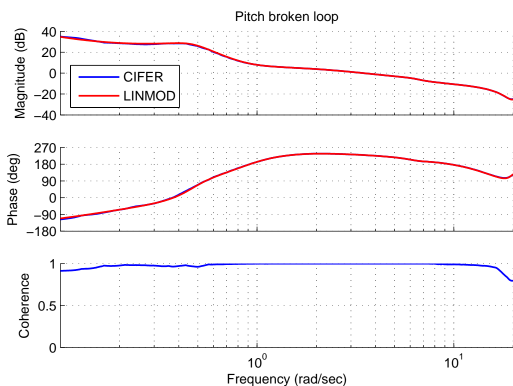

Figure 81: Corrected pitch broken loop

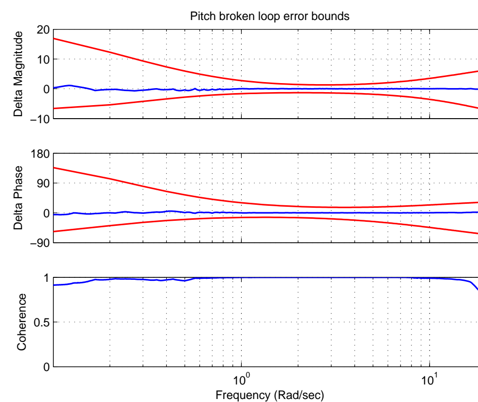

Figure 82: Corrected pitch broken loop error

The amount of phase margin lost due to the discrepancies was around 4 degrees. Differences in gain margins were also seen because the 180 degree point was reached sooner. The biggest problem with these discrepancies would show up when tuning correction factors. If tuned to the LINMOD frequency responses, the corrected model will be more liberal in margins than indicated by CIFER frequency response or the model in the time domain. Now that the LINMOD and CIFER frequency responses match, tuning done to the model will achieve the correct effect in both time and frequency domains.

IX. Conclusion

The new CIFER command line interface has provided a reliable and stable platform to create the validation tool. The MATLAB class structure allows for reduced complexity of the out-lying code by encapsulating and streamlining the CIFER system identification process. The new frequency response class also allows the retrieval frequency responses from the CIFER database and frequency response plots to be generated with little work.

The process for sweeping block diagrams is now easy enough to be performed on any CONDUIT block diagram with little to no setup. The validation tool GUI provides a simple push-button functionality to perform validation work. The ease of the validation tool should benefit users by reducing the time it takes to validate the linearization process.

The simple 2nd order and XV-15 examples give confidence that CIFER is providing an accurate truth model from the time domain simulation. CIFER matches the analytic describing function for all of the tested nonlinearities. In the few cases where the validation process broke down, the coherence provided enough information to determine the valid range of the validation. In the observed case where CIFER did not correctly identify the broken loop response due to rate limiting and the missing indication from the coherence data, the CIFER results were obviously not the broken loop response and a warning was added to notify the user of the potential problem.

The RASCAL example showed how a complex block diagrams can be validated using the CIFER command line interface for more than the linearization process. The RASCAL simplified diagram was validated utilizing the CIFER command line interface

against both the full-up diagram and flight test data. Additionally the linearization validation showed some small linearization problems, and they were fixed. These validations proved that the automated sweeping process can handle complex diagrams.

The validation of linearization method against system identification has proven to be effective. Some mistakes were found and fixed, and the validation tool allows for re-validation to be performed very quickly.

Thanks to this work, system identification of Simulink simulations can now be done quickly and effectively when associated with a CONDUIT case. Part of the future work will be to generalize the Simulink integration to allow system identification of any Simulink diagram in an automated fashion.

X. Appendix A: 1st over 2nd plots

See Supplementary File simple_1_2.pdf

XI. Appendix B: Analysis script examples

1. Flight test analysis example

This example shows the processing of flight test data from the 8025 flight test shown in the thesis. This is a typical analysis setup, just done with the new command line.

run_cases.m

```
clear; clc; close all

% Specify flight number
flight = '8025';

% Choose windows
window = [36,20,15,10,5];

% Link time history events with channel names and min/max frequencys
events = {
    3   'VIPPITCH'  'VIP PITCH AXIS CHRIP'          'EP' 'FP' [35,20,15,10,5]    .5   25
    4   'VIPROLL'   'VIP ROLL AXIS CHRIP'           'ER' 'FR' window             .8   25
    5   'VIPYAW'    'VIP YAW AXIS CHRIP'            'EY' 'FY' window             .5   45
    6   'VIPVERT'   'VIP VERTICAL AXIS CHRIP'       'EC' 'FC' window            0   20
    7   'DRBPITCH'  'HIGH DRB PITCH AXIS CHRIP'     'EP' 'FP' window             .5   25
    8   'DRBROLL'   'HIGH DRB ROLL AXIS CHRIP'      'ER' 'FR' window             .7   25
    9   'DRBYAW'    'HIGH DRB YAW AXIS CHRIP'       'EY' 'FY' window             .75   43
    10  'DRBVERT'   'HIGH DRB VERTICAL AXIS CHRIP'  'EC' 'FC' window             .45   20
    11  'BASPITCH'  'BASELINE PITCH AXIS CHRIP'     'EP' 'FP' window             .65   25
    12  'BASROLL'   'BASELINE ROLL AXIS CHRIP'      'ER' 'FR' window             .6   25
    13  'BASYAW'    'BASELINE YAW AXIS CHRIP'       'EY' 'FY' window             .6   40
    14  'BASVERT'   'BASELINE VERTICAL AXIS CHRIP'  'EC' 'FC' window             .3   20
    };

% Prepare the command line interface for use
setup_CIFER;

warning('off','UH60M_CIFER:Field_missing')
% Interate through each event
for i = 1:size(events,2)
    % Perform some transformation and unit conversions on the flight test
    % data
    [filename, data, uh60_2_cifer, cifer_2_uh60] =  ...
        uh60m_cifer(['.' filesep 'data\'],flight,events{i,1});

    % Generate a blank frespid_obj
    fre = frespid_obj;
    % Set the case name and comments
    fre.name = events{i,2};
    fre.comments = events{i,3};

    % These responses are SISO, so no need for cross correlation
    fre.crosscor = false;

    % Set the input and output names
    fre.controls{1} = events{i,4};
    fre.outputs{1} = events{i,5};

    % Setup the time history file, in this case it is from a MATLAB .MAT
    % file, source 4
    fre.thfiles(1).source = 4;
    fre.thfiles(1).flight = str2double(flight);
    fre.thfiles(1).event = events{i,1};
```

```
    fre.thfiles(1).filename = filename;

    % Compute all response pairs
    fre.frcalc(:) = true;

    window_length = events{i,6};
    min_freq = events{i,7};
    max_freq = events{i,8};

    % Set the windows
    for j = 1:numel(window_length)
        fre.windows(j).on = true;
        fre.windows(j).length = window_length(j);
        fre.windows(j).min_freq = min_freq;
        fre.windows(j).max_freq = max_freq;
    end

    % Create the composite case
    com = composite_obj(fre);

    clean_BATDIR(fre.name)

    % Submit COMPOSITE FRESPID job
    job = batch(fre);
    wait(job);

    if(~check(job))
        error('FRESPID job %d failed', i)
    end

    % Submit the COMPOSITE job
    job = batch(com);
    wait(job);

    if(~check(job))
        error('COMPOSITE job %d failed', i)
    end

    frname{i} = [events{i,2} '_COM_ABCDE_' events{i,4} '_' events{i,5}];

    fr{i} = ciffrq(frname{i});

    % Compute gain and phase margins, and bandwidth
    [phase{i}, gain{i}] = crossover(fr{i});
    bw{i} = bandwidth(fr{i});
end

for i = 1:numel(fr)
    % Write the frequency responses to .mat files for overlaying
    writeFRfile(fr{i}.name,[fr{i}.name '.mat'], 'M', int32([1 -1 1]));
end
```

2. XV-15 batch validation example

The plots for the XV-15 example were generated using the following script. The 2nd order elements and RASCAL validations were done via similar scripts.

```
close all;clc

fields = {'id', 'in', 'out','bl','stick_fac'};

outputs = {'v' 'p' 'r' 'ay' 'phi' 'psi'};

% Maximum stick amounts
lat_max = 4.8;
ped_max = 2.5;

% Specific the input, output and switches for the validation
data = {
    'Closed loop LAT'  'd_lat_s' 'p'             [] lat_max
    'Closed loop PED'  'd_ped_s' 'r'             [] ped_max
    'Broken loop LAT'  'd_lat_c_in' 'd_lat_c_out'  1  lat_max*1.5
    'Broken loop PED'  'd_ped_c_in', 'd_ped_c_out' 2  ped_max*2.8};

cases = cell2struct(data, fields, 2);

% Percent throw
Amps = [.002 .15 .5];
% Iterate over each percentage
for j = 1:numel(Amps)
    minfreq = .1;
    maxfreq = 10;
    Amp = Amps(j);
    scenarios = {};
    switches = [];

    % Iterate over each case
    for i = 1:numel(cases)
        fprintf('%.3f\n',100*(i/numel(cases)))

        % This function calls LINMOD and does the CIFER sweeping
process
        % and returns the frequency response, the frespid and composite
        % case, and the time history data
        [linmod_tf{i,j}, cifer_fr{i,j}, fre, com, thdata{i,j}] = ...
            validate_short(cases(i).in, cases(i).out, switches, ...
            cases(i).bl, scenarios, minfreq, maxfreq, ...
            Amp*cases(i).stick_fac);
    end

end

% Save the results for plotting
save results linmod_tf cifer_fr cases thdata Amps
```

XII. Appendix C: Sweep equation

$$\delta_{sweep} = A(t)\sin[\theta(t)] + \delta_{noise}$$

$$A(t) = \begin{cases} t < t_{zero} \Rightarrow 0 \\ t_{zero} \le t < t_{zero} + t_{fadein} \Rightarrow \dfrac{t - t_{zero}}{t_{fadein}} \\ t_{zero} + t_{fadein} \le t < T_{rec} - t_{fadeout} - t_{zero} \Rightarrow 1 \\ T_{rec} - t_{fadeout} - t_{zero} \le t < T_{rec} - t_{zero} \Rightarrow 1 - \dfrac{t - T_{rec} + t_{zero} + t_{fadeout}}{t_{fadeout}} \\ T_{rec} - t_{zero} < t \Rightarrow 0 \end{cases}$$

$$\theta(t) = \int \omega(t)\,dt$$

$$\omega(t) = \begin{cases} t < t_{zero} \Rightarrow 0 \\ t_{zero} \le t < t_{zero} + t_{park} \Rightarrow \omega_{\min} \\ t \ge t_{zero} + t_{park} \Rightarrow \omega_{\min} + K(t)(\omega_{\max} - \omega_{\min}) \end{cases}$$

$$K(t) = C_2\left(e^{\frac{C_1(t - t_{park} - t_{zero})}{T_{rec} - t_{park} - 2t_{zero}}} - 1 \right)$$

$$C_1 = 4$$

$$C_2 = \frac{1.0023}{e^{C_1} - 1} = .0187$$

$$\theta(t) = \begin{cases} t < t_{zero} \Rightarrow 0 \\ t_{zero} \le t < t_{zero} + t_{park} \Rightarrow \omega_{\min}(t - t_{zero}) \\ t_{zero} + t_{park} \le t \Rightarrow C_2(\omega_{\max} - \omega_{\min})\left[\dfrac{(T_{rec} - t_{park} - 2t_{zero})}{C_1}\left(e^{\frac{C_1(t - t_{park} - t_{zero})}{T_{rec} - t_{park} - 2t_{zero}}} - 1 \right) + t_{zero} + t_{park} - t \right] + \omega_{\min}(t - t_{zero}) \end{cases}$$

XIII. Appendix D: Second order elements metrics

Case	Source	Gain Margin (dB)	Phase Margin (deg)	Crossover frequency (rad/sec)	Bandwidth (rad/sec)
Baseline (open loop)	CIFER	9.553	17.725	4.717	N/A
	LINMOD	9.571	17.788	4.713	N/A
	Difference	0.188%	0.358%	-0.078%	N/A
Hysteresis (open loop)	CIFER	2.994	7.396	4.428	N/A
	LINMOD	N/A	N/A	N/A	N/A
	Difference	N/A	N/A	N/A	N/A
Dead zone (open loop)	CIFER	12.203	24.858	4.058	N/A
	LINMOD	9.571	17.788	4.713	N/A
	Difference	-21.567%	-28.443%	16.161%	N/A
Saturation (open loop)	CIFER	10.509	20.411	4.471	N/A
	LINMOD	9.571	17.788	4.713	N/A
	Difference	-8.924%	-12.852%	5.420%	N/A
Lookup table (open loop)	CIFER	10.591	20.570	4.457	N/A
	LINMOD	15.591	36.571	3.236	N/A
	Difference	47.213%	77.792%	-27.396%	N/A
Memory block (open loop)	CIFER	6.051	13.699	4.716	N/A
	LINMOD	9.571	17.788	4.713	N/A
	Difference	58.171%	29.844%	-0.053%	N/A
Time delay (open loop)	CIFER	1.401	4.193	4.715	N/A
	LINMOD	1.372	4.286	4.713	N/A
	Difference	-2.031%	2.205%	-0.032%	N/A
Rate limit (open loop)	CIFER	3.983	14.527	4.684	N/A
	LINMOD	9.571	17.788	4.713	N/A
	Difference	140.276%	22.445%	0.622%	N/A
Baseline (good design)	CIFER	18.732	49.471	2.554	4.274
	LINMOD	18.483	49.339	2.577	4.274
	Difference	-1.324%	-0.268%	0.912%	-0.006%
Baseline (bad design)	CIFER	9.543	17.833	4.694	7.210
	LINMOD	9.571	17.788	4.713	7.196
	Difference	0.289%	-0.254%	0.405%	-0.194%
Hysteresis (good design)	CIFER	9.898	40.530	1.805	3.732
	LINMOD	N/A	N/A	N/A	N/A
	Difference	N/A	N/A	N/A	N/A
Hysteresis (bad design)	CIFER	6.376	14.316	4.648	6.914
	LINMOD	N/A	N/A	N/A	N/A
	Difference	N/A	N/A	N/A	N/A
Dead zone (good design)	CIFER	27.611	N/A	N/A	3.600
	LINMOD	18.483	49.339	2.577	4.274
	Difference	-33.059%	N/A	N/A	18.721%

Dead zone (bad design)	CIFER	11.427	19.915	4.459	6.866
	LINMOD	9.571	17.788	4.713	7.196
	Difference	-16.244%	-10.679%	5.710%	4.813%
Saturation (good design)	CIFER	18.083	48.848	2.474	3.728
	LINMOD	18.483	49.339	2.577	4.274
	Difference	2.212%	1.006%	4.170%	14.638%
Saturation (bad design)	CIFER	12.091	34.318	3.664	5.493
	LINMOD	9.571	17.788	4.713	7.196
	Difference	-20.846%	-48.167%	28.641%	31.011%
Lookup table (good design)	CIFER	21.455	68.796	1.689	3.939
	LINMOD	24.504	75.460	1.294	2.741
	Difference	14.212%	9.686%	-23.411%	-30.423%
Lookup table (bad design)	CIFER	10.136	18.136	4.645	7.045
	LINMOD	15.591	36.571	3.236	5.099
	Difference	53.828%	101.650%	-30.330%	-27.617%
Memory block (good design)	CIFER	15.105	47.269	2.554	4.307
	LINMOD	18.483	49.339	2.577	4.274
	Difference	22.369%	4.378%	0.876%	-0.784%
Memory block (bad design)	CIFER	6.126	13.775	4.687	7.213
	LINMOD	9.571	17.788	4.713	7.196
	Difference	56.223%	29.130%	0.553%	-0.232%
Time delay (good design)	CIFER	10.361	42.141	2.549	4.352
	LINMOD	10.285	41.957	2.577	4.346
	Difference	-0.736%	-0.437%	1.073%	-0.130%
Time delay (bad design)	CIFER	1.387	4.347	4.645	7.131
	LINMOD	1.372	4.286	4.713	7.080
	Difference	-1.036%	-1.418%	1.466%	-0.722%
Rate limit (good design)	CIFER	13.737	50.138	2.539	4.109
	LINMOD	18.483	49.339	2.577	4.274
	Difference	34.552%	-1.593%	1.484%	3.996%
Rate limit (bad design)	CIFER	8.847	14.998	4.698	7.223
	LINMOD	9.571	17.788	4.713	7.196
	Difference	3.611%	18.606%	0.327%	-0.370%

XIV. Works Cited

[1] Hodgkinson, John. Aircraft Handling Qualities. Reston, VA: American Institute of Aeronautics and Astronautics, 1999. Page 68.

[2] Tischler, Mark B. and Robert K. Remple. Aircraft and Rotorcraft System Identification. Reston, VA: American Institute of Aeronautics and Astronautics, 2006. Page 89-90, 169-177

[3] Ogata, Katsuhiko. Modern Control Engineering. Englewood Cliffs, N. J.: Prentice-Hall, 1970. Page 540-545

[4] Duda, Holger. "Effects of Rate Limiting Elements in Flight Control Systems - A New PIO-Criterion." AIAA Guidance, Navigation and Control. 1995. DLR, German Aerospace Research Establishment.

[5] Gelb, Arthur and Wallace E. Vander Velde. "Sinusoidal-Input Describing Function (DF)." Multi-input Describing Functions and Nonlinear System Design. New York, NY: McGraw-Hill, 1968.

[6] Rupnik, Brian K. CIFER-MATLAB Interfaces: Development and Application. California Polytechnic State University, 2005. Aerospace Engineering - Flight Simulator - Cal Poly San Luis Obispo. 2007. California Polytechnic State University. 1 May 2009 <http://aerosim.calpoly.edu/library.html>.

[7] Cheung, Kenny. "CONDUIT Setup and Run Modes." CONDUIT Training Course. 2009

[8] Mansur, Mohammadreza H., Jeff A. Lusardi, Mark B. Tischler, and Tom Berger. "Achieving the Best Compromise between Stability Margins and Disturbance Rejection Performance." American Helicopter Society 65th Annual Forum. 2009 May. 2009. Aeroflightdynamics Directorate (AMRDEC), U.S. Army RDECOM, UC Santa Cruz (UARC).

[9] Tischler, Mark B., "System Identification Methods for Aircraft Flight Control Development and Validation," NASA TM-110369, 1995.

[10] Ivler, Christy. "Simulation Requirements." CONDUIT Training Course. 2009

[11] Tischler, Mark B., Jason D. Colbourne, Mark R. Morel, Daniel J. Biezad, William S. Levine, and Veronica Moldoveanu. CONDUIT - A New Multidisciplinary Integration Environment for Flight Control Development. 1997. Moffet Field: NASA and US Army, 1997.

[12] Anon, "Flight Control Systems - Design, Installation And Test Of Piloted Aircraft, General Specification For," MIL-DTL-9490E, Department of Defense, April 2008.

[13] Anon, "Aeronautical Design Standard, Performance Specification, Handling Qualities Requirement for Military Rotorcraft," US Army Aviation and Missile Command, USAAM-COM, ADS-33E-PRF, March 2000.

[14] National Research Council (U.S.). Aviation Safety and Pilot Control. Washington: National Academy Press, 1997.