Old Dominion University
ODU Digital Commons

Electrical & Computer Engineering Theses & Disssertations

Electrical & Computer Engineering

Summer 2018

Model-Less Fuzzy Logic Control for the NASA Modeling and Control for Agile Aircraft Development Program

Keith A. Benjamin
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/ece_etds

Part of the Electrical and Computer Engineering Commons

Recommended Citation
Benjamin, Keith A.. "Model-Less Fuzzy Logic Control for the NASA Modeling and Control for Agile Aircraft Development Program" (2018). Master of Science (MS), thesis, Electrical/Computer Engineering, Old Dominion University, DOI: 10.25777/8e62-nc73
https://digitalcommons.odu.edu/ece_etds/36

This Thesis is brought to you for free and open access by the Electrical & Computer Engineering at ODU Digital Commons. It has been accepted for inclusion in Electrical & Computer Engineering Theses & Disssertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.

MODEL-LESS FUZZY LOGIC CONTROL FOR THE NASA MODELING AND CONTROL FOR AGILE AIRCRAFT DEVELOPMENT PROGRAM

by

Keith A. Benjamin
B.S. Computer Engineering 2016, Old Dominion University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

ELECTRICAL AND COMPUTER ENGINEERING

OLD DOMINION UNIVERSITY
August 2018

Approved by:

Oscar R. González (Director)

W. Steven Gray (Member)

Dimitrie C. Popescu (Member)

ABSTRACT

MODEL-LESS FUZZY LOGIC CONTROL FOR THE NASA MODELING AND CONTROL FOR AGILE AIRCRAFT DEVELOPMENT PROGRAM

Keith A. Benjamin
Old Dominion University, 2018
Director: Dr. Oscar R. González

The NASA Modeling and Control for Agile Aircraft Development (MCAAD) program seeks to develop new ways to control unknown aircraft to make the aircraft development cycle more efficient. More specifically, there is a desire to control an aircraft with an unknown mathematical model using only first principles of flight. In other words, rather than using a rigorously developed mathematical model combined with wind-tunnel tests, a controller is sought which would allow one to bypass the development of a rigorous mathematical model and enter wind-tunnel testing more directly. This paper presents the design of a fuzzy PID controller, governed by a fuzzy supervisory system which incorporates knowledge of first principles of flight, to control a model-less aircraft's pitch dynamics in a free-to-pitch wind-tunnel environment. This hybrid structure is implemented using a PID controller constructed from independent fuzzy inference systems and augmented in real time by a supervisory system also constructed of independent fuzzy inference systems. Experimental results of the pitch control performance and real-time adaptivity capabilities are presented for both aerodynamically stable and unstable aircraft models.

Copyright, 2018, by Keith A. Benjamin, All Rights Reserved.

ACKNOWLEDGMENTS

This work was partially supported by the NASA Langley Research Center under Cooperative Agreements NNL09AA00A and 80LARC170004 (NIA Grants C15-2B00, Activity 2B51 and C15-2B00-ODURF, Activity 201017). The author would like to thank A. Mekky for his support in developing the non-linear F-16 simulations used for developing and testing the implemented controller and J. Brandon, M. Croom, and E. Viken and the rest of the NASA Langley Research Center 12-ft Wind Tunnel team, whose guidance and support made the development and testing of this experiment possible.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES ix

Chapter

1. INTRODUCTION 1
   - 1.1 MOTIVATION 1
   - 1.2 PROBLEM DESCRIPTION 2
   - 1.3 OUTLINE 3
2. BACKGROUND 5
   - 2.1 FUZZY LOGIC PRIMER 5
   - 2.2 APPLICATION 12
3. METHODOLOGY 14
   - 3.1 OVERVIEW 14
   - 3.2 DESIGN 14
   - 3.3 CONTROL SYSTEM ARCHITECTURE 15
   - 3.4 CONTROLLER DESIGN 17
   - 3.5 SUPERVISOR DESIGN 20
   - 3.6 ADAPTIVITY 22
   - 3.7 BRINGING IT ALL TOGETHER 23
4. SIMULATION 24
   - 4.1 OVERVIEW 24
   - 4.2 SIMULATION 24
   - 4.3 SIMULATION MODELS 25
   - 4.4 CONTROLLER EVALUATION 29
   - 4.5 EXPERIMENTATION HYPOTHESIS 36
5. EXPERIMENTATION 38
   - 5.1 EXPERIMENTAL DATA COLLECTION 38
   - 5.2 PERFORMANCE CHARACTERISTICS 42
6. SUPPLEMENTAL WORK 47
   - 6.1 MOTIVATION 47
   - 6.2 METHODOLOGY 47
   - 6.3 SIMULATION 49

Chapter Page

7. CONCLUSION 54
7.1 OVERVIEW OF FINDINGS 54
7.2 RESEARCH IMPLICATIONS 54
7.3 SUMMARY 54

REFERENCES 56

APPENDICES

A. MATLAB CODE 58
A.1 SIMULATION DRIVER 58
A.2 INITIALIZATION 63
A.3 LINEAR SIMULATION MODELS 69
A.4 CUSTOM QUEUE CLASS 72
A.5 DATA LOGGING CLASS 74
A.6 FUZZY LOGIC CONTAINER CODE 78
A.7 CONTROLLER CODE 90
A.8 SUPERVISOR CODE 99

VITA 114

LIST OF TABLES

Table Page

1. Simulation Configurations Table 25
2. FLC Initialization Parameters 26
3. Linear F16 Model Parameters 27
4. F16 Linear Model 1 and F16 Non-Linear Model Controller Simulation Performance Characteristics 33
5. F16 Linear Model 2 Controller Simulation Performance Characteristics 35
6. Boeing 747 Linear Model Controller Simulation Performance Characteristics 37
7. Controller Initial Values 42
8. Controller Output Adaptation Parameters 50
9. Controller Input Adaptation Parameters 50
10. Adaptive Controller Simulation Performance Characteristics 51

LIST OF FIGURES

Figure Page

1. Model Plane on Free-to-Pitch Rig . . . 3
2. Example Input (Output) Membership Functions . . . 7
3. Example Membership Functions . . . 9
4. Output Distribution Functions – Singletons . . . 11
5. Example Output . . . 13
6. Conceptual Architecture of the Model-Less Fuzzy Logic Controller and Supervisor 15
7. Input Membership Functions . . . 17
8. K_p Rule-base Surface Representation . . . 21
9. K_i Rule-base Surface Representation . . . 21
10. K_d Rule-base Surface Representation . . . 22
11. Simulink Simulation Diagram . . . 25
12. Fuzzy Logic and PID Controller Response to Linear F16 Model 1 . . . 30
13. Controller Performance Comparison Based on Instantaneous RMS Error for Linear Models . . . 30
14. Fuzzy Logic and PID Controller Response to Non-Linear F16 Model . . . 31
15. Non-Linear F16 Model Controller Performance Comparison Based on Instantaneous RMS Error . . . 31
16. PID Performance on Non-Linear Plant Demonstrating Tracking During Long Duration Doublets . . . 32
17. F16 Linear Model 1 and Non-Linear F16 Model Controller Performance Comparison Based on Instantaneous RMS Error . . . 34
18. Fuzzy logic and PID response to the F16 Linear Model 2 . . . 34
19. Fuzzy logic and PID response to the Boeing 747 Linear Model . . . 36
20. Hardware Stack . . . 39

Figure Page

21. Run-time Program Flowchart 40

22. LEX Aircraft Configuration 42

23. Stock Aircraft Response to Doublet Input (Open Loop) 43

24. Stock Aircraft Response to Doublet Input (Closed Loop) 44

25. Stock Aircraft Initial Self-Tuning Time Frame 44

26. LEX Aircraft Response to Doublet Input (Open Loop) 45

27. LEX Aircraft Response to Doublet Input (Closed Loop) 46

28. Input Adaptation Algorithm 48

29. Output Adaptation Algorithm 49

30. F16 Linear Model 1 and Non-Linear F16 Model Fuzzy Logic Controller Response With Adaptive Routines 52

31. F16 Linear Model 1 and Non-Linear F16 Model Controller Performance with Adaptivity Comparison Based on Instantaneous RMS Error 52

32. Fuzzy Logic and Linear PID Controller Response to Non-Linear F16 Model with Adaptivity Simulation Shown 53

33. Simulated Performance Comparison Based on Instantaneous RMS Error 53

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Historically, the time from aircraft design to implementation is protracted due to the iterative nature of design analysis and testing [1]. In this process, mathematical approximations are made to form an aerodynamic model of a theoretical aircraft design and are used to develop a flight control system [2]. After simulated analysis, an aircraft model is constructed for use in a wind tunnel for further development and refinement of the mathematical model [3]. Lessons learned from wind-tunnel experimentation are used to tune the mathematical models, making them more accurate in order to further refine the control system. This process is repeated until sufficient confidence is gained with the theoretical models to move to full-scale testing and production [1].

Computational fluid dynamics (CFD), finite-element (FE) models, and other modeling and simulation techniques are common approaches to the aircraft design flow [4, 5]. These approach aircraft design through a robust, albeit incomplete, understanding of the mathematical underpinnings of aerodynamics and fluid dynamics. However, even though modern modeling and simulation offers an increasingly complete simulation environment, wind-tunnel testing of scaled models is still required in order to fully validate an aerodynamic model [2,3]. Thus, an iterative process, whereby wind-tunnel experiments are followed by CFD model refinement, is used to update mathematical models. This process is repeated until sufficient confidence in the CFD models is achieved such that they closely match wind-tunnel experimental dynamics [1]. This is, however, not without its own limitations. For example, a full six degree-of-freedom (6-DOF) mathematical model cannot be validated in a wind tunnel, as this would require an aircraft suspended in mid-air and operating under its own power. Instead, lower order degrees of freedom are investigated independently by fixing the degrees-of-freedom not under investigation. In this way, a sufficiently large amount of the stable flight envelope is exercised to assure confidence such that full-scale aircraft test flights may commence. [2]

A more streamlined approach calls for the use of a generalized controller, tuned not for

performance but stable flight.[1] Ideally, such a controller would be tuned solely against first principles of flight so that it would apply to the broadest class of aircraft. This would provide a method to control a new aircraft design in a wind-tunnel environment with minimal *a priori* knowledge yet still obtain accurate mathematical models. In this way, a scaled model can be exercised throughout the entire flight envelope while performing System Identification (SID) in order to obtain accurate CFD model parameters for further off-line development, thereby shortening development time [6, 7].

The path to achieve this goal requires a degree of adaptability in order to maintain stable flight. Model Reference Adaptive Control (MRAC) [8], model-less control utilizing on-line SID [9], and machine learning [10] are a few approaches which have been tried with various degrees of success. These approaches suffer from three main detractors: 1) a baseline model must be supplied *a priori*, 2) solution convergence is slow with 8.5 seconds being a representation of *fast* convergence [11] and 3) stable convergence is not necessarily guaranteed due to unpredictable *in-situ* aircraft conditions.

Therefore, a controller capable of stable control without the need for a baseline model, off-line pre-tuning, and minimal *a priori* aircraft specifications;[2] stable control achievement at least as fast as [11]; and some measure of guaranteed convergence would be of great value in shortening aircraft development time.

1.2 PROBLEM DESCRIPTION

The focus of this study is to develop a One Degree-of-Freedom (1-DOF) fuzzy logic pitch controller capable of tracking the pitch command of an aircraft with unknown dynamics. The controller must meet the following specifications:

1. Use only first principles of flight for controller development. More specifically, a basic understanding of free-body mechanics. Control limits are assumed known.

2. The controller must track pitch commands in a 1-DOF wind-tunnel experiment. Adequate tracking will be considered achieved when the instantaneous root-mean-square error converges to a value less than five degrees.

3. Controller adaptivity convergence must occur in less than 8.5 seconds.[3]

[1] For the purposes of this experiment, stable flight is determined to be any time the aircraft's change-in-pitch converges within a pre-determined flight envelope.

[2] For example, control surface and actuator limits.

[3] A literature search found this to be a high performance benchmark for adaptive aircraft. [11]

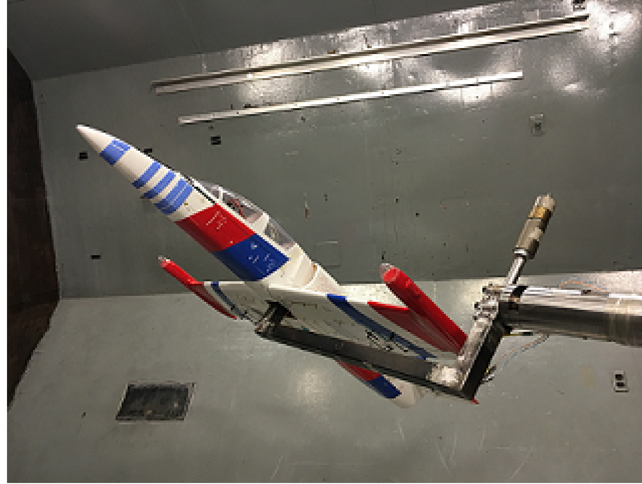

FIG. 1: Model Plane on Free-to-Pitch Rig

4. Stability must be achieved for differing linear and nonlinear plants in simulation without controller pre-tuning.

Given the abstract nature of the design requirements, fuzzy logic control is selected as the proposed control solution for its ability to handle both nonlinear and abstract control regimes providing mathematical machinery which allows for the embedding of *expert knowledge*. Thus, control is obtained using a qualitative approach similar to a pilot rather than the quantitative approach required of classic control methodologies.

This controller uses only *expert knowledge* and a basic understanding of flight principles to adapt in real time to a new airplane model. A two-part hybrid fuzzy logic controller approach comprised of a separate supervisor and controller was adopted under the assumptions that the aircraft is controllable and the aircraft obeys the standard 6-DOF equations-of-motion (EOM). The controller was evaluated in the NASA Langley 12-foot wind tunnel using a scaled model of an Aero L-59 Super Albatros. A free-to-pitch One Degree-of-Freedom (1-DOF) rig, as shown in Fig. 1, was obtained by locking the longitudinal and vertical axes so that only rotation about the lateral axis is obtained. This aircraft was treated as a "black-box" with unknown aerodynamics.

1.3 OUTLINE

Chapter 2 provides a brief background to fuzzy inference systems, their design, and implementation. Standard fuzzy logic terminology and apparatus necessary for the understanding

of this approach will be introduced using a simple example. Chapter 3 describes the controller developed in this study. Chapter 4 discusses the simulation of multiple aircraft models in a virtual environment. Chapter 5 discusses the controller's performance by analyzing data collected during wind-tunnel experimentation. Chapter 6 reviews the implications of this work and future research pathways. The paper will be concluded in Chapter 7.

CHAPTER 2

BACKGROUND

2.1 FUZZY LOGIC PRIMER

A thorough treatment of the theory, design, and implementation of fuzzy inference systems is beyond the scope of this paper. However, this is to serve as a brief review of the fuzzy logic topics used in this study. This chapter reviews the topics of fuzzy logic systems used in this work, while leaving further study of the topic to the reader.

2.1.1 TERMINOLOGY

- Fuzzy

  A qualitative value rooted in conceptual ideas and abstract constructs used to approximate a *crisp* value.

- Crisp

  A quantitative value rooted in mathematical ideas carrying specific values represented by real or complex numbers.[1]

- Universe-of-Discourse

  The set of values over which an input or output is valid. This includes real or complex numbers, a finite set of integers, a closed-infinite set, or any other group of values. Continuity is not required.

- Linguistic Values and Variables

  Abstract terms used to describe the value of an input or output. A useful example is human height. To say one is *tall* or *short* is not concretely descriptive. Suppose a poll is conducted whereby it is determined that a short person is 5 feet in height and a tall person is 7 feet in height. What does this say about the person who is 6 feet tall? Would this person be average height? Suppose further that it is known that 85 percent of the population is between 5 feet 4 inches and 5 feet 10 inches tall.

[1] Complex numbers are not required in this experiment but are included here for completeness.

Statistically speaking, one would correctly determine that an average person is in this range of values, but using merely the concrete definition of short equals 5 feet in height and tall equals 7 feet in height, one would determine average height to be 6 feet tall.

Humans intuitively correct for these ambiguities with continuous observation of the environment and use terms like *short*, *tall*, *small*, *big*, *little*, *large*, etc. to describe ranges of values meant only to be loose approximations. These constitute the *descriptive* terms of a fuzzy logic system; they are *linguistic variables.* A *linguistic value* is the numerical value associated with a *linguistic variable.* In this case *tall* equals 7 feet, *average* equals 5 feet 10 inches, and *short* equals 5 feet tall.

- Membership Function
  A distribution function which maps linguistic values to crisp values.

- Fuzzification
  The conversion of crisp values into fuzzy values.

- Defuzzification
  The conversion of fuzzy values into crisp values.

- Rule-base
  A set of IF-THEN rules used to map input values to output values as part of the inference system.

- Inference system
  "The system which emulates the expert's decision making in interpreting and applying knowledge about how best to control the plant." [12]

2.1.2 UNIVERSE-OF-DISCOURSE

The universe-of-discourse effectively carries the same meaning as a mathematical *set* and is typically referred to by scripted variables, such as $\mathcal{U}$. In terms of control, this directly relates to the set of operating values for an input or output. For example, the elevator of an aircraft has physical limits of operation, say $\pm 30\,\text{deg}$. The universe-of-discourse for this parameter is therefore $\mathcal{U} \subseteq [-30, 30]$. It is important to note, however, that a complete fuzzy inference system will have a universe-of-discourse for each input and output – all of which are independent. Additionally, there is no requirement that a universe-of-discourse be a finite set; any set of real or complex numbers is valid.

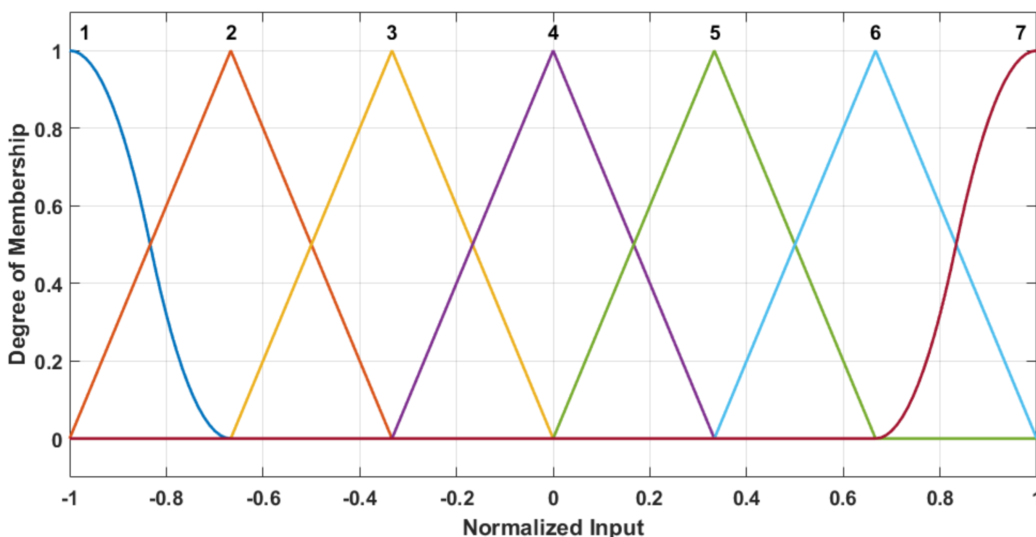

FIG. 2: Example Input (Output) Membership Functions

2.1.3 MEMBERSHIP FUNCTIONS

Membership functions, also known as *distribution functions* and *confidence functions*, are a way of representing the degree to which one may be confident that a value is represented by a linguistic variable. Linguistic variables are named using conceptual ideas like "positive small" or "negative big" and are assigned to individual membership functions. The degree-of-membership is the confidence one has that the input/output value being evaluated belongs to a represented membership function. Note that the name *distribution function* is somewhat unfortunate as it might lead one to relate these functions to *probabilistic distribution functions*. This is, however, not the case as there is no requirement for the integral of a given function to equal 1; hence the name *membership function*, which is an attempt to communicate a *degree* of membership. Consequently, *membership function*, *distribution function*, and *confidence function* are used interchangeably and are not probability distributions.

For example, Fig. 2 represents a normalized fuzzy set – all distribution functions for an input/output over a universe-of-discourse. Take the distribution function 5 to represent the linguistic value "positive small" and the distribution function 6 to represent the linguistic value "positive medium," supposing this represents a fuzzy set for an input. Now suppose that an input value of 0.25 is to be evaluated. By visual inspection, one will notice that one can be approximately 80% confident that the input value represents a *positive small* number, approximately 20% certain the value belongs to membership function 4, but may

be absolutely sure that it does not belong to any other membership function.

Many different membership functions exist but only four are used in this work, triangle, z, s, and singleton, with one additional function, Gaussian, used in examples. Membership functions are defined below as used in the *Matlab*® *Fuzzy Logic Toolbox* ^{TM}. [13] The triangle membership function is defined in (1).

$$f_{tri}(x;a,b,c) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & z \leq x \leq b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & c \leq x \end{cases} \tag{1}$$

The z membership function is defined in (2).

$$f_z(x;a,b) = \begin{cases} 1, & x \leq a \\ 1 - 2\left(\frac{x-a}{b-a}\right)^2, & a \leq x \leq \frac{a+b}{2} \\ 2\left(\frac{x-b}{b-a}\right)^2, & \frac{a+b}{2} \leq x \leq b \\ 0, & x \geq b \end{cases} \tag{2}$$

The s membership function is defined in (3).

$$f_s(x;a,b) = \begin{cases} 0, & x \leq a \\ 2\left(\frac{x-a}{b-a}\right)^2, & a \leq x \leq \frac{a+b}{2} \\ 1 - 2\left(\frac{x-b}{b-1}\right)^2, & \frac{a+b}{2} \leq x \leq b \\ 1, & x \geq b \end{cases} \tag{3}$$

The Gaussian membership function is defined in (4).

$$f_{gauss}(x;\sigma,c) = \exp\left\{\frac{-(x-c)^2}{2\sigma^2}\right\} \tag{4}$$

The singleton function, pertaining only to outputs, is defined in (5).

$$f_{singleton}(x;a) = \begin{cases} 1, & x = a \\ 0, & otherwise \end{cases} \tag{5}$$

These point values of 1, which characterize the *Takagi-Sugeno* type fuzzy system, can be distributed anywhere in the fuzzy output space. Instead of harnessing the concept of *degree-of-certainty*, these systems require different techniques, discussed in Sec. 2.1.6, to develop output values.

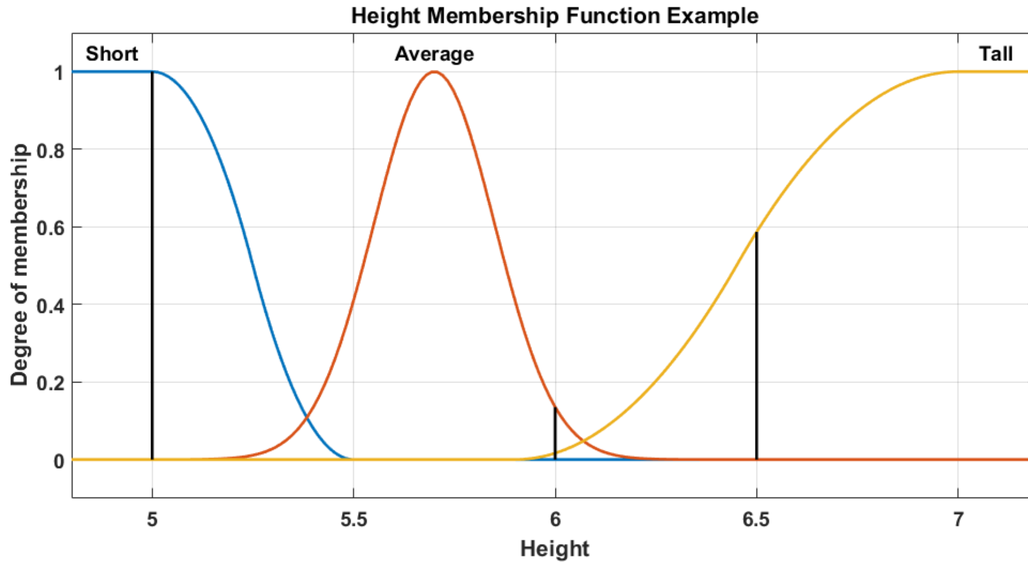

FIG. 3: Example Membership Functions

2.1.4 FUZZIFICATION

Fuzzification is the method by which an input value is converted to a fuzzy value – a quantified value is converted to a qualified value. Take the height example of Sec. 2.1.1, three different heights are examined: 5 feet, 6 feet, and 6.5 feet. To develop this example further, consider the membership function distribution in Fig. 3, where the fuzzy set, the distribution functions associated with an input variable, is defined axiomatically as:

1. SHORT $\equiv f_z(x; 5, 5.5)$
2. AVERAGE $\equiv f_{gauss}(x; 0.15, 5.7)$
3. TALL $\equiv f_s(x; 5.9, 7)$

An explanation of the fuzzification process follows using the inputs 5 feet, 6 feet, and 6.5 feet.

Input: 5 Feet

To evaluate the input of 5 feet to the fuzzy set, each distribution function is individually evaluated. The SHORT membership function will evaluate to 1, indicating that there is 100% confidence that someone who is 5 feet tall is SHORT, which matches the axiomatic

system description. Similarly, the AVERAGE and TALL membership functions will evaluate to 0, implying that one may be completely confident that someone who is 5 feet tall is neither AVERAGE nor TALL.

- SHORT: $f_1 \equiv f_z|_{x=5} = 1.0 \equiv 100\%$
- AVERAGE: $f_2 \equiv f_{gauss}|_{x=5} = 0.0 \equiv 0\%$
- TALL: $f_3 \equiv f_s|_{x=5} = 0.0 \equiv 0\%$

Input: 6 Feet

Continuing the example, evaluating the fuzzy set at 6 feet yields the following:

- SHORT: $f_1 \equiv f_z|_{x=6} = 0.0 \equiv 0\%$
- AVERAGE: $f_2 \equiv f_{gauss}|_{x=6} = 0.1353 \equiv 13.53\%$
- TALL: $f_3 \equiv f_s|_{x=6} = 0.0165 \equiv 1.65\%$

Graphically, from Fig. 3, it is seen that one who is 6 feet tall ought to be on the high side of the AVERAGE distribution and the low side of the TALL distribution. Additionally, the evaluation of the system ought to yield a higher confidence that this person would be AVERAGE rather than TALL, which is precisely what is seen. One may be completely confident that a 6 foot individual is not SHORT, but only partially confident as to the person's AVERAGE or TALL classification. Interestingly, it is possible, as in this case, that one may not be particularly confident at all as to one's height classification.

Input: 6.5 Feet

Lastly, consider one who is 6.5 feet tall. Colloquially, one may suggest a classification of TALL. Evaluation of the distribution functions yields the following results:

- SHORT: $f_1 \equiv f_z|_{x=6.5} = 0.0 \equiv 0\%$
- AVERAGE: $f_2 \equiv f_{gauss}|_{x=6.5} = 0.0 \equiv 0\%$
- TALL: $f_3 \equiv f_s|_{x=6.5} = 0.5868 \equiv 58.68\%$

Thus, it is determined that though someone who is 6.5 feet tall is not TALL, since they are less than 7 feet tall, intuition will say that the person is indeed TALL because they are of

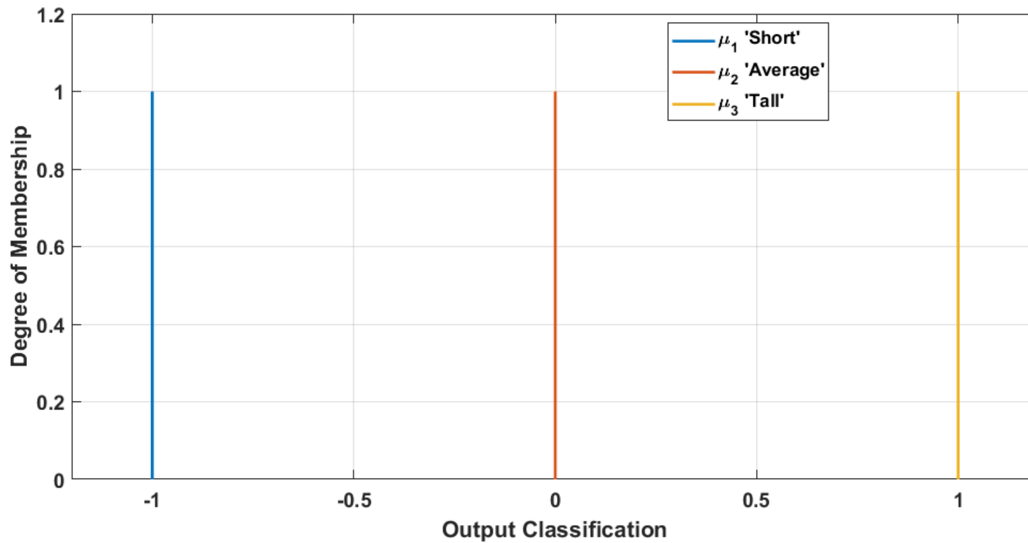

FIG. 4: Output Distribution Functions – Singletons

neither AVERAGE nor SHORT height. It should be noted that by changing membership function type, location, parameters, and quantity, different classification outputs may be obtained.

2.1.5 THE RULE-BASE AND INFERENCE SYSTEMS

This simple, one dimensional example yields a very simplistic rule-base – IF-THEN statements are used by the inference system to *infer* the output response. For this example, singletons, Fig. 4, will be used such that the output -1 will mean *SHORT*, 0 will mean *AVERAGE*, and 1 will mean *TALL*. Therefore, the rules, μ_i, for the system are:

$$\begin{aligned}
\mu_1 &= \text{IF SHORT} \quad \text{THEN } \textit{SHORT} \\
\mu_2 &= \text{IF AVERAGE} \quad \text{THEN } \textit{AVERAGE} \\
\mu_3 &= \text{IF TALL} \quad \text{THEN } \textit{TALL}
\end{aligned}$$

Inference occurs by determining the amount each rule contributes to the final output. For example, evaluating an input of 5.43 for each input membership function results in the following confidence value from each rule:

$$\begin{aligned}
\mu_1|_{x=5.43} &= 0.0392 \\
\mu_2|_{x=5.43} &= 0.1979 \\
\mu_3|_{x=5.43} &= 0
\end{aligned}$$

These values represent the confidence with which each rule contributes to the output. In other words, each membership function will evaluate to a single *fuzzy* number during fuzzification. Since every membership function should be assigned to one or more rules, the confidence of a membership function is the confidence in the rule to which it is associated. Complex situations, such as rules utilizing more than one membership function, are beyond the scope of this example. Their use, however, is a natural extension of this membership function to rule mapping.

2.1.6 DEFUZZIFICATION

The fuzzification and inference processes described in Sections 2.1.4 and 2.1.5 resulted in numerous outputs with values ranging from 0 to 1 – one output for each rule. Defuzzification is the process by which these results are converted to a single, usable, *crisp* output.

Numerous algorithms are available; however, the one appropriate for this study, center-of-gravity, is a simple sum of the products of each rule and its associated output value divided by the sum of the confidence levels. A *Takagi-Sugeno* system is characterized as a fuzzy inference system using *singleton* outputs and *center-of-gravity* defuzzification. [12] The formula presented in (6) describes the defuzzification process mathematically where y_{crisp} is the *crisp* output, μ_i is a specific inference rule, and y_i is the output singleton associated with the given rule.

$$y_{crisp} = \frac{\sum_{i=1}^{n} \mu_i y_i}{\sum_{i=1}^{n} \mu_i} \tag{6}$$

Fig. 5 demonstrates the selection capability of this system graphically. The output values of the fuzzy inference system, the output of (6), lie along the blue line labeled *Output Surface*. The classification regions are defined by the dominant confidence function at a particular input in Fig. 3. This makes sense in the simple example presented but may be more complicated based on a specific system configuration. As this line enters the various classification regions, the associated classification becomes SHORT, AVERAGE, or TALL as appropriate. In this example, all input values less than 5.38 ft are designated as SHORT, all inputs greater than or equal to 6.07 ft are classified as TALL, and all others are AVERAGE. The *Fuzzy Output* crossover points lie at ±0.5 due to the even distribution of output singletons, Fig. 4, and the center-of-gravity defuzzification function, (6).

2.2 APPLICATION

The example presented in this chapter is a simple system which demonstrates the ability of fuzzy logic to make classification decisions. This example is meant merely to describe the

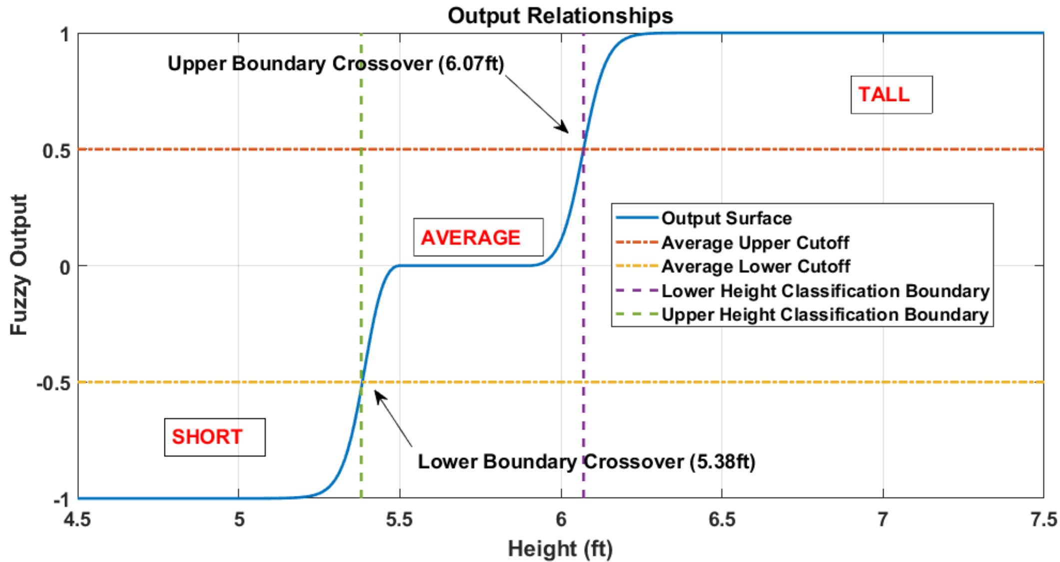

FIG. 5: Example Output

operation and implementation of fuzzy logic as far simpler techniques could have been used for this manner of classification. What is presented, however, is the ability of fuzzy logic to incorporate *expert* knowledge in a relational system to provide a quantified output based on quantified inputs using qualified connections.

The classification example provides one of three outputs, SHORT, AVERAGE, and TALL for a range of inputs, namely $(-\infty, \infty)$. The power of the fuzzy logic system, however, is not limited to merely three outputs; an infinite range of values can be produced based on the system design. Next, a fuzzy logic controller will be presented for aircraft pitch control using the same techniques presented here.

CHAPTER 3

METHODOLOGY

3.1 OVERVIEW

The goal for this controller was to obtain stable flight for a general class of aircraft based on a fundamental understanding of first principles-of-flight, rather than tuning a controller to a mathematical model. To achieve this, design and verification was carried out in three phases: development, simulation, and experimentation. The design phase, Chapter 3, focuses on developing a fuzzy logic control system for linear and non-linear models. The simulation phase, Chapter 4, focuses on refining the controller structure to increase performance and expanding its capability to a wider range of linear and non-linear models. Finally, the experimentation phase, Chapter 5, puts the proposed design into a real-world wind-tunnel application.

3.2 DESIGN

In the design phase, parameter and actuator limits for each model were considered as boundary values for simulation. This motivated an architecture, Section 3.3, constructed around fuzzy logic systems that are inherently bounded by design, support performance evaluation, and aid fault detection for conditions such as controller input-output saturation and ineffective control. Fig. 6 depicts the system architecture comprised of a controller and supervisor built from fuzzy logic controllers.

A single generic fuzzy inference system, Section 3.3.1, applicable to all desired control points was created in order to ease system development and integration. Specifically, it provides a uniform input/output interface as a means of simplifying programmatic design and future usability in other projects. Additionally, it reduces the run-time memory space via code reuse and aids computational efficiency, a necessity for meeting design requirement 3 of Section 3.3.

The proposed controller was simulated using four different plants: two linear F-16 models [14,15], a linear Boeing 747 model [16], and a non-linear F-16 model [14,17]. The models are presented in detail in Section 4.3. These simulations were utilized to develop a fuzzy logic

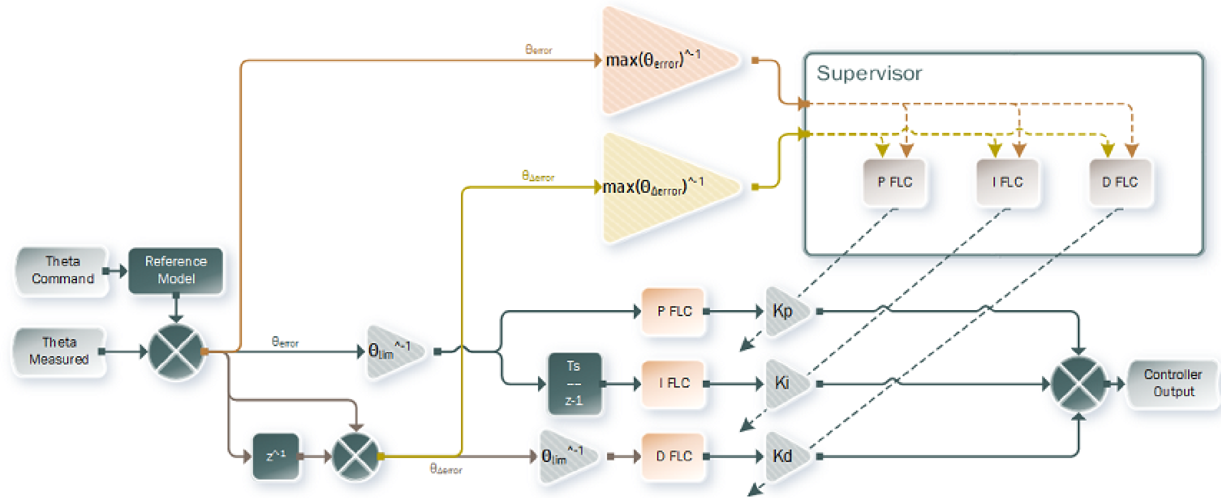

FIG. 6: Conceptual Architecture of the Model-Less Fuzzy Logic Controller and Supervisor

controller architecture capable of meeting stability performance specifications for a 1-DOF plant.

3.3 CONTROL SYSTEM ARCHITECTURE

In this project, fuzzy inference systems convert *pilot experience* into *crisp* control outputs through a series of fuzzification and defuzzification techniques. The overall controller architecture consists of two fuzzy subsystems: a fuzzy PID controller and fuzzy supervisor, each composed of three individual fuzzy inference systems (FIS). The controller FISs are shown as P.FLC, I.FLC, D.FLC in Fig. 6, while the supervisor FISs are shown as P.FLC, I.FLC, D.FLC within the *Supervisor* block. The MATLAB® Fuzzy Logic Toolbox was used to generate and implement all FISs.

A PID style controller design was chosen due to its proven closed-loop performance characteristics. The PID functionality was performed with three separate fuzzy logic systems, one corresponding to each of the classical *proportional*, *integral*, and *derivative* control paths. The magnitudes of these paths were scaled by the gains Kp, Ki, and Kd, shown as triangular blocks in Fig. 6. Controller output gains were updated in real time by a fuzzy supervisor subsystem.

The supervisor is composed of three separate FISs, each mapped one-to-one to a PID controller output gain. That is, each FIS internal to the supervisor is specifically constructed to adapt its associated PID gain to reduce the errors (8)-(10) found in Sec. 3.4.

The FISs of the PID and supervisor subsystems were created using a single generalized FIS designed for application to any two input, one output system. Section 3.3.1 presents a Takagi-Sugeno type FIS with normalized inputs and outputs useful for the creation of the PID and supervisor subsystems. Section 3.4.1 describes the PID subsystem and Section 3.5.1 describes the supervisor subsystem.

3.3.1 GENERAL FUZZY INFERENCE SYSTEM

A general fuzzy inference system was created using seven membership functions for each of the two inputs and one output. The input membership functions were placed on a normalized universe-of-discourse, $[-1,\ 1]$, as shown in Fig. 7, which allowed easy scaling of inputs using external multiplication [12]. Membership functions 2 through 6 were symmetric triangle distribution functions with centers evenly spaced at

$$\left\{-\frac{2}{3}, -\frac{1}{3}, 0, \frac{1}{3}, \frac{2}{3}\right\},$$

respectively, and widths set such that the outer endpoints of each membership function were coincident with the center of the adjacent membership function. These overlaps served to prevent *dead–zones* from appearing at the output by allowing smooth transitions between adjacent firing rules.

The outer two membership functions, 1 and 7, were z and s distribution functions, respectively. Unlike the triangle membership functions, which yielded a degree-of-membership of 0 at $\pm\infty$, these gave a degree-of-membership of 1 when an input saturated. Generating a non-zero value during saturation allows the controller to create a meaningful output. Without this capability, (6) would produce a zero output since no distribution functions would be firing. Saturation of a z distribution function occurred on the interval $\{(-\infty, -1]$ while the s distribution function saturated on the interval $[1, \infty)\}$.

As per Takagi-Sugeno type systems, the output membership functions were singletons rather than distributions. Their outputs were selected to take the same normalized locations as did the centers of the input membership functions; that is, the outputs are located at

$$\left[-1, -\frac{2}{3}, -\frac{1}{3}, 0, \frac{1}{3}, \frac{2}{3}, 1\right].$$

A uniform distribution of membership functions, spanning negative and positive values, was chosen to facilitate application to a general class of aircraft. Each supervisor output had a scaling weight initially selected during the simulation phase of the development and identified as a reasonable starting location to begin adaptive control.

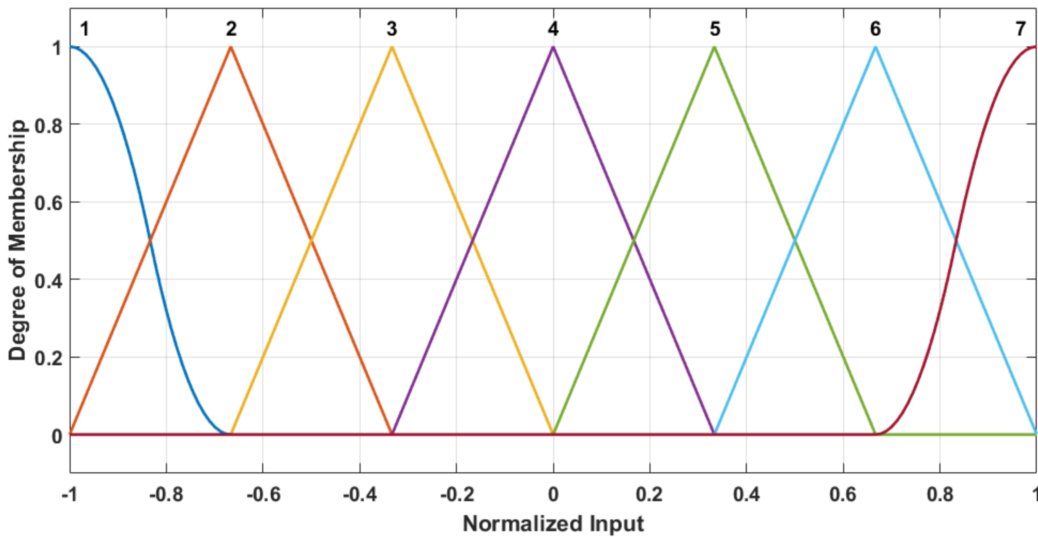

FIG. 7: Input Membership Functions

The controller's response depended on its rule-base definitions but was fine tuned by adjusting its output gains via the supervisor. The rule-bases, which mapped inputs to outputs were uniquely applied to each FIS. Sections 3.4 and 3.5 outline the structure of the forty-nine rules necessary to fully define each FIS's input-output relationships.

3.4 CONTROLLER DESIGN

The fuzzy logic control system was developed to track pitch angle commands. Pitch angle command tracking assumes that:

1. Measurements of the controlled angle are available,
2. The limits of operation for the controlled angle are known, and
3. The angular limits on the control surfaces are known.

Pitch angle, defined as the angle between an aircraft's longitudinal axis and the ground, is controlled via elevator deflection commands [7]. Intuitively, from a pilot's perspective, a small change in elevator angle should induce a small change in pitch angle. Likewise, large, fast, and slow elevator changes should result in large, fast, and slow pitch angle responses, respectively, taking an aircraft centric approach. Additionally, performance was characterized using the concepts of quick and accurate where *quick* is the speed of the

aircraft's response and *accurate* is satisfaction of the aircraft's final pitch error taking a pilot centric approach.

Translating the nebulous concepts above into concrete, quantifiable ideas required the identification of some mathematical constraints. Therefore, tracking accuracy was specified by selecting a reference model that defined the desired transient and steady-state error characteristics. The primary purpose of the reference command filter was to prevent large derivative terms from saturating outputs, but it also established instantaneous tracking performance while maintaining the spirit of a *model-less* approach by making no assumptions of the aircraft's mathematical model. The selected reference-model was

$$\theta_{\text{ref}}(s) = \frac{6.25}{s^2 + 4.25s + 6.25}\theta_{\text{cmd}}(s), \tag{7}$$

a second-order transfer function in the s-domain with a natural frequency $\omega_n = 2.5\,\text{rad/s}$ and damping ratio $\zeta = 0.85$, where *quickness* of the system is determined by ω_n. Additionally, this transfer function produced a settling time $T_{settle} = 1.68s$ and an overshoot $\%OS = 0.63$ percent.

The reference model (7) was discretized for a sampled-data system running at 50 Hz. For *accuracy*, the goal for the controller was to reduce the following errors: tracking error (8), change in error (9), and the integral of the error (10), where $T_s = \frac{1}{50}$ s is the sample period, $k = 0, 1, 2, \ldots$ were the sample instants, and the integrator's initial condition was zero.

$$\theta_{\text{error}}(kT_s) = \theta_{\text{ref}}(kT_s) - \theta_{\text{meas}}(kT_s), \tag{8}$$

$$\theta_{\Delta\,\text{error}}(kT_s) = \theta_{\text{error}}(kT_s) - \theta_{\text{error}}((k-1)T_s), \tag{9}$$

$$\theta_{\sum\text{error}}(kT_s) = T_s \sum_{n=1}^{k} \theta_{\text{error}}(kT_s). \tag{10}$$

3.4.1 PID OVERVIEW

A parallel fuzzy PID controller was developed using the general FIS template described in Section 3.3.1 with one FIS created for each PID control path. The inputs to the three channels of the fuzzy PID controller were $\theta_{error}(kT_s)$, $\theta_{\sum error}(kT_s)$, and $\theta_{\Delta error}(kT_s)$ for the proportional, integral, and derivative paths, respectively. The inputs to each controller FIS were scaled by $1/\theta_{\text{lim}}$, where θ_{lim} was the assumed given pitch limit of the aircraft.

The PID subsystem FISs have been restricted to single-input single-output systems by holding one of the inputs constant. This created a linear mapping where the output is the negative of the input. This negation was a result of the sign convention used for aircraft

equations of motion [14]. This served to scale the response of each controller path and assign the correct output elevator angular position, δ_e. More specifically, the body axis aircraft model defines negative δ_e as an elevator position pitched toward the top of the aircraft. Based on (8), a negative error means the current pitch exceeds the desired pitch, so the elevator must move in the positive, that is the downward, direction in order to correct the error. However, these linear mappings required the oversight of the supervisor outputs as they did not constitute a PID controller in the strictest sense.

The PID structure was only fully implemented when the fuzzy controller was combined with the supervisor described in Section 3.5.1. When combined with the supervisor described in Section 3.5.1, the three fuzzy PID controller output channels were scaled by the weights set by the supervisor: Kp, Ki, and Kd. The weighted outputs were summed to create the control output to the elevators for θ control. To prevent over-driving the actuators, the final control output was limited to $[\min(\delta_e), \max(\delta_e)]$, where $\delta_{e_{\mathrm{lim}}}$ denotes the angular displacement limit of the aircraft's elevators.

3.5 SUPERVISOR DESIGN

3.5.1 OVERVIEW

The supervisor consisted of three FISs of the design presented in Section 3.3.1 to control the proportional, integral, and derivative output gains in the fuzzy PID subsystem. Each FIS took θ_{error} and $\theta_{\Delta error}$ as inputs with $1/\max(\theta_{error})$ and $1/\max(\theta_{\Delta error})$ as input normalizing factors, where $\max(\theta_{error})$ was the difference between the upper and lower θ limits. The weight $\max(\theta_{\Delta_{error}})$ was set to four times $\max(\theta_{error})$ as a general starting location for simulation and wind-tunnel testing. This did not change during simulation or wind-tunnel experiments.

The supervisor rule-bases were developed so as to specifically accentuate different features of the separate PID constructs given the aircraft's performance with respect to θ_{error} and $\theta_{\Delta_{error}}$. For example, the supervisory FIS associated with integral PID action is set to output larger values when θ_{error} is small in order to reduce steady-state error. Each rule-base is described Sections 3.5.2–3.5.4.

3.5.2 PROPORTIONAL SUPERVISION

The proportional input-output rule-base mapping is shown in Fig. 8. The given distribution increased the proportional gain of the controller when either θ_{error} or $\theta_{\Delta error}$ was small and reduced it when both θ_{error} and $\theta_{\Delta error}$ were large. This allowed the controller to respond to conditions when θ_{error} was large and unchanging by increasing proportional gain, while a situation where θ_{error} and $\theta_{\Delta_{error}}$ were large and in the same direction results in a small change because the system is already moving toward $\theta_{error} = 0$. Additionally, the system responded to θ_{error} and $\theta_{\Delta_{error}}$ being large in opposite directions, driving $\theta_{error} \rightarrow \pm\infty$, by driving outputs in a direction opposite to the current trend so that $\theta_{error} \rightarrow 0$. Proportional gains were high near zero so that the system was able to quickly respond to step input changes.

3.5.3 INTEGRAL SUPERVISION

The integral input-output rule-base mapping is shown in Fig. 9. To reduce the steady state error, the rule-base output is maximum when θ_{error} is small. Large θ_{error} also demands a maximum output in order to return the system to steady-state quickly. Note that when θ_{error} and $\theta_{\Delta_{error}}$ are small, the distributions of Fig. 8 and Fig. 9 reinforce one another to create a

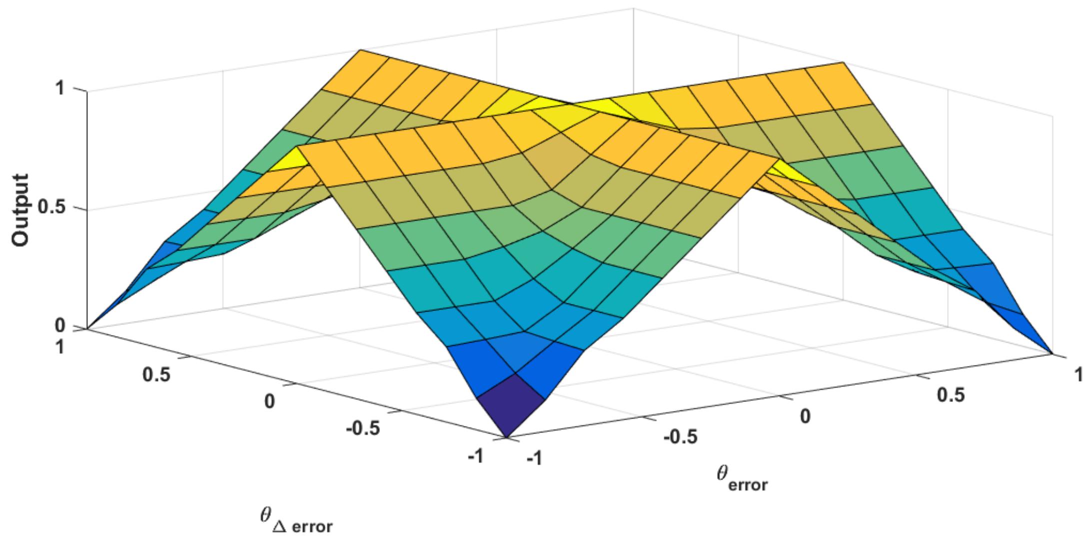

FIG. 8: K_p Rule-base Surface Representation

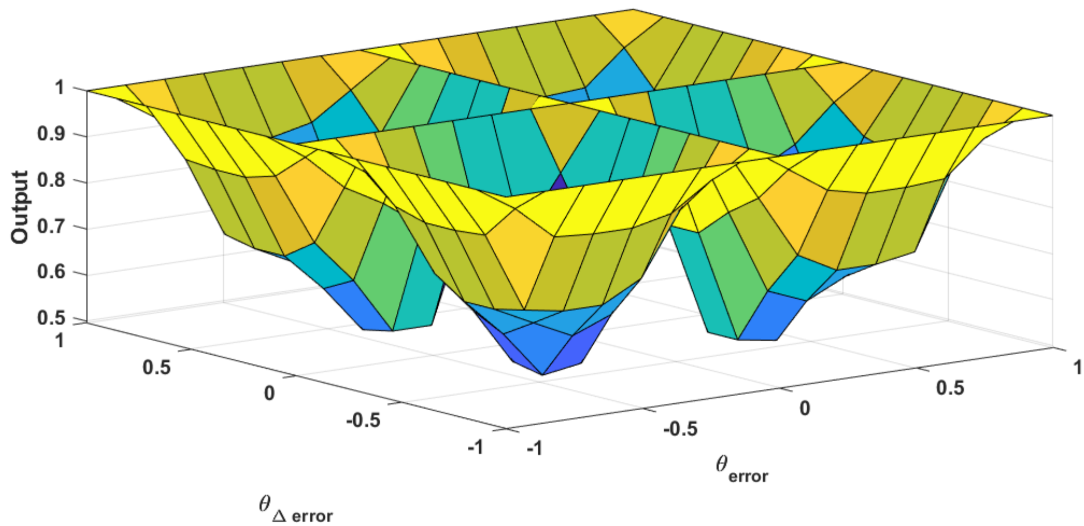

FIG. 9: K_i Rule-base Surface Representation

stronger control action; however, their outer ranges differ in order to specifically tailor the output response. The low points near $(\pm 0.5, \pm 0.5)$ help to prevent integrator windup by limiting its operating region and allow the *proportional* and *derivative* supervisory systems to operate as the primary actors. Should those be insufficient to maintain control, the system inputs will naturally gravitate to a region where the *integral* supervisor has more control.

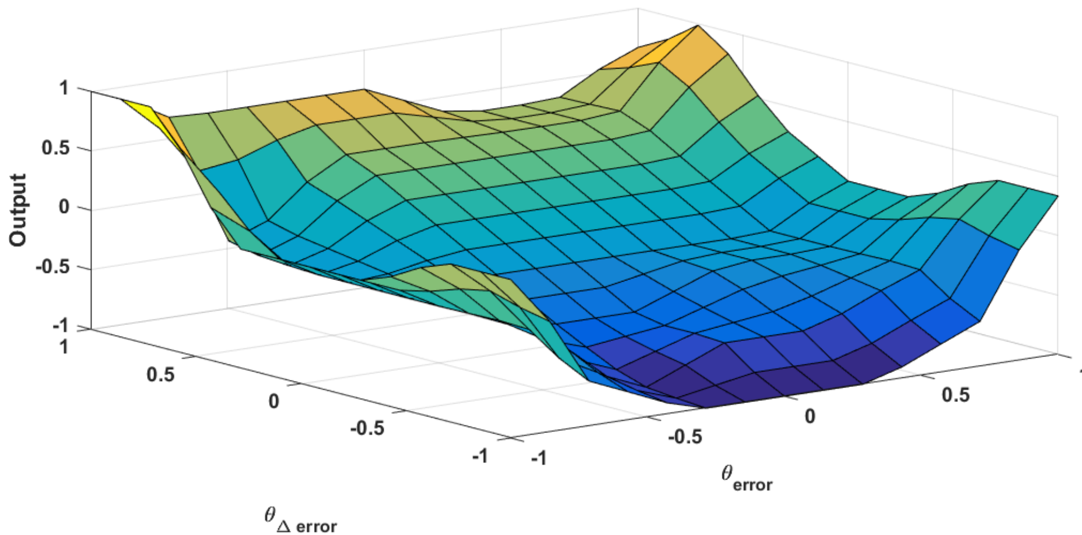

FIG. 10: K_d Rule-base Surface Representation

3.5.4 DERIVATIVE SUPERVISION

Finally, the derivative input-output rule-base mapping is shown in Fig. 10. This distribution was proposed in [18] based on the results presented in [19] as a way to prevent oscillation and over-saturation. Oscillations are suppressed by increasing rate feedback as the rate magnitude increases toward $\theta_{\Delta error} = \pm 1$. Over-saturation is achieved by minimizing the number of points where the output is maximal.

The magnitudes of the supervisor output channels were scaled by considering the *function* of the individual FISs. Since the proportional supervisor system is setting the proportional gain of the fuzzy PID controller, its output scale was set to $\delta_{e_{\text{lim}}} = \max|\delta_e|$ so that the controller could access the full range of δ_e values. The scaling factors for the derivative and integral channels were chosen as fractions or multiples of elevator angular position limit, $\delta_{e_{\text{lim}}}$; these are given in Chapters 4 and 5.

3.6 ADAPTIVITY

The unknown nature of the system indicates that some amount of adaptivity was required. Thus, a supervisory fuzzy logic system is constructed to auto-tune the fuzzy controller parameters so that it could meet the control system specifications. Equations (8) and (9) were used as inputs to the supervisor with the goal of driving $\theta_{error}(kT_s) \to 0$ and $\theta_{\Delta error}(kT_s) \to 0$ as $k \to \infty$. Intuitive principles were employed for adaptivity, for example, the presence of

steady-state error requires either more or less control input in order to minimize error. These have been programmed implicitly into the rule-bases of the fuzzy inference systems.

Presented here is a different type of adaptivity than is typically presented. Rather than using a series of algorithms or techniques to converge to an operating point, the presupposition was made that if an aircraft is controllable, then its operating point exists within the confines of the combined fuzzy logic controllers of the proposed system. Stated another way, the combination of the proposed supervisor-controller system generates a six-dimensional operating space wherein it is assumed that, if the aircraft is stabilizable, stable pitch control exists and is reachable.[1] Section 6 discusses additional work in adaptation, not empirically tested, designed to provide better performance.

3.7 BRINGING IT ALL TOGETHER

The presented controller is an amalgamation of a number of control strategies: PID control, fuzzy logic control, and supervision. Sections 3.2–3.6 serve to answer the practical question of *how* the controller operates, but this section seeks to answer *what* the controller does during each sample period.

Recall Fig. 6. A command θ_{cmd} is issued and filtered through a reference model to generate commands which, if tracked perfectly, will provide a desirable rise-time and overshoot. This signal is compared with the measured pitch, θ_{meas} to develop the error signal, θ_{error}.

The error signal, θ_{error} is then processed to obtain the difference, $\theta_{\Delta error}$, and cumulative sum, $\theta_{\Sigma error}$. These signals are then supplied to the PID and supervisor subsections of the controller.

The supervisor normalizes the input signals θ_{error} and $\theta_{\Delta error}$ which are then supplied to the proportional, integral, and derivative portions of the supervisor. The outputs of these supervisor modules are supplied to the PID controller to become the output scaling gains.

Similarly, the PID subsection normalizes its inputs and passes those values to the proportional, integral, and derivative sections of the controller. The PID subsection outputs, now scaled by the supervisor's supplied output scaling gains, are summed to create a control output. Not shown in Fig. 6 is a final block which limits the control outputs so as not to overdrive the physical actuators.

[1] Reachability here describes the ability to maintain an aircraft's stability by keeping it within the confines of the stable flight envelope.

CHAPTER 4

SIMULATION

4.1 OVERVIEW

Simulation is an important part of this project since wind-tunnel experimentation will be limited to a single aircraft with limited aerodynamic variations. The virtual environment, however, allows one to perform any number of control schemes to get a sense of techniques that merit implementation. Additionally, development iterations are quickened as the bounds of time may also be simulated and the cost of virtual testing is negligible compared to wind-tunnel operating costs.

Four models were chosen for the simulation environment to both match the wind-tunnel unit-under-test (UUT) and gauge performance in dissimilar aircraft. Using an Aero L-59 Super Albatros as the wind-tunnel UUT, models for the F16 Fighting Falcon were chosen as *like* aircraft. Simulation was performed using two different linear models, trimmed at different operation points, and a full 6 degree-of-freedom (6-DOF) non-linear model. A Boeing 747 linear model was chosen as a dissimilar aircraft. Each model was simulated with both a simple PID controller and the proposed FLC for comparison. One PID controller was selected for use in simulation of the non-linear model as a comparison. It is understood that this is not a wholly appropriate approach since no attempt at gain scheduling at multiple operating points was made, but it is expected to have an operational range over which some comparison may be made. A controller's ability to track input pitch commands and reach a steady-state condition while remaining within the flight envelope was the primary performance consideration.

4.2 SIMULATION

The simulation environment was created using Matlab in which various models and controllers could be easily switched. A diagram of the closed-loop pitch simulation can be seen in Fig. 11. Working from left to right, a simple doublet command routine is employed as the system input due to its historical use in simulation. [20] Commands were filtered by the discretized reference model in (7) to produce a command sequence with more manageable

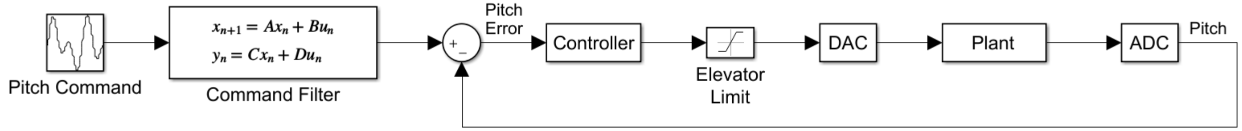

FIG. 11: Simulink Simulation Diagram

TABLE 1: Simulation Configurations Table

	F16 Model			Boeing 747
Controller	*Linear-1 FLC*	*Linear-2 FLC*	*Non-Linear FLC*	*FLC*
	Linear-1 PID	*Linear-2 PID*	*Non-Linear PID*	*PID*

rise times and minimize large control commands induced by the sharp differentials created by true step commands. One of the two controllers and one of the four plants were switched in for the appropriate simulation according to the configuration list in Table 1. A saturation block was placed in-line to simulate the real physical travel limits of an aircraft elevator. [1]

4.3 SIMULATION MODELS

Three linear models were used for simulation. Two F16 models trimmed about different operation points were used to study controller performance on similar models with different characteristics. A third model, a Boeing 747, was used to study controller performance to a wider set of aircraft, namely comparing an agile fighter aircraft to a transport aircraft.

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 \\ \frac{\bar{c}S\bar{q}}{I_y C_{ma}} & \frac{\bar{c}S\bar{q}}{I_y C_{mq}} \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{\bar{c}S\bar{q}}{I_y C_{me}} \end{bmatrix} \\ C &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned} \tag{11}$$

The generalized model in (11) presents a reduced order model for the state-space representation $\dot{x} = Ax + bu$, $y = Cx + du$ where $\bar{c}$ is the mean aerodynamic chord length, $\bar{q}$ is

[1] The analog-to-digital converter (ADC) and digital-to-analog converter (DAC) in Fig. 11 have no significance in simulation because inputs and outputs were not quantized. They are included for completeness as their presence is necessary in wind-tunnel experiments.

TABLE 2: FLC Initialization Parameters

Experiment Parameter	Value
$\delta_{e_{\text{lim}}}$	40 deg
θ_{lim}	25 deg
$\max\lvert\theta_{error}\rvert$	50 deg
$\max\lvert\theta_{\Delta_{error}}\rvert$	100 deg
Supervisor Proportional Output Scale	40
Supervisor Integral Output Scale	10
Supervisor Derivative Output Scale	640

dynamic pressure, S is the area of the wing, I_y is the pitching moment of inertia, C_{ma} is the pitching moment coefficient contribution due to the angle-of-attack (α), C_{mq} is the pitching moment coefficient contribution due to pitch rate (q), and C_{me} is the pitching moment coefficient contribution due to elevator deflection (δ_e). [21] The input $u \triangleq \delta_e$ where δ_e is aircraft elevator angular position in degrees, $x \triangleq [\alpha, \dot{\alpha}]^T$ where α is the aircraft's angle of attack and $\dot{\alpha}$ is the first derivative of α.

The four-state models in (14) and (16) represent a second F16 model and a Boeing 747 model, respectively. They take the states $x \triangleq [\Delta u, \alpha, q, \theta]^T$ where θ is the pitch angle and q is the pitch rate. For the purposes of this study, $\alpha \equiv \theta$, as testing was performed in a wind tunnel in which $\alpha = \theta$ as a constrained parameter and all future references to α will be discussed at θ. These simulations were utilized to develop a fuzzy logic controller architecture capable of meeting stability performance specifications for a 1-DOF plant. The FLC initialization parameters are provided in Table 2. Lastly, the continuous linear models in (12)-(16) were discretized, not shown, with a zero-order-hold equivalent of $T_s = 0.02$ sec.

Linear F16 Model 1 The reduced order model in (11) is described more specifically by the state-space system in (12) obtained by substituting the values in Table 3. [21]

The model used the linearized parameters C_{ma}, C_{mq}, and C_{me} for an equilibrium point of $\alpha = \theta = 0\,\text{deg}$, $V = 600\,\text{ft/s}$, and $h = 25\,000\,\text{ft}$. The parameters for (12) and equilibrium points are provided in Table 3. [21] Note that the values for C_{ma}, C_{mq}, and C_{me} are dimensionless.

TABLE 3: Linear F16 Model Parameters

Parameter	**Value**
I_y	$55\,814\,\text{slug}\cdot\text{ft}$
$\bar{c}$	$11.32\,\text{ft}$
S	$300\,\text{ft}^2$
$\bar{q}$	$192.21\,\text{psf}$
C_{ma}	-0.0376
C_{mq}	-0.0584
C_{me}	-0.6073
α	$0\,\text{deg}$
δ_e	$0\,\text{deg}$
V	$600\,\text{ft/s}$
h	$25000\ \text{ft}$

$$
\begin{aligned}
A &= \begin{bmatrix} 0 & 1 \\ -0.4403 & -0.6830 \end{bmatrix} & B &= \begin{bmatrix} 0 \\ -7.1030 \end{bmatrix} \\
C &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & D &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned} \tag{12}
$$

The plant is controllable due to its full-rank controllability matrix. Extracting θ from (12) yields the transfer function (13) with two stable poles at $s = -0.3415 \pm 0.5689i$.

$$
H_\theta(s) = \frac{-7.103}{s^2 + 0.683s + 0.4403} \tag{13}
$$

Linear F16 Model 2 The four-state model in (14) was taken from [15, p. 128]. This represents a model linearized about an operating point where $\delta_e = 0\,\text{deg}$ and velocity $V = 203.87\,\text{ft/s}$.

$$A = \begin{bmatrix} -0.0507 & -3.861 & 0 & -32.2 \\ -0.00117 & -0.5164 & 1 & 0 \\ -0.000129 & 1.4168 & -0.4932 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ -0.0717 \\ -1.645 \\ 0 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{14}$$

The plant is controllable due to its full-rank controllability matrix. Extracting θ from (14) yields the transfer function (15) with four poles at $s = \{-1.7036,\ 0.7310,\ -0.0438 \pm 0.2066i\}$.

$$H_{sys}(s) = \frac{-0.0717s^3 - 1.684s^2 - 0.08519s - 0.06168}{s^4 + 1.06s^3 - 1.115s^2 - 0.0658s - 0.05552} \tag{15}$$

Linear Boeing 747 Model The four-state model in (16) was taken from [16, p. 92]. This represents a model linearized about an operating point where $\delta_e = 0\,\text{deg}$ and velocity $V = 278.67\,\text{ft/s}$.

$$A = \begin{bmatrix} -0.0188 & 11.5959 & 0 & -32.2 \\ -0.0007 & -0.5357 & 1 & 0 \\ 0.000048 & -0.4944 & -0.4935 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ -0.5632 \\ 0 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{16}$$

The plant is controllable due to its full-rank controllability matrix. Extracting θ from (16) yields the transfer function (17) with four poles at $s = \{-0.5221 \pm 0.7029i,\ -0.0019 \pm 0.1250i\}$.

$$H_{sys}(s) = \frac{-0.5632s^2 - 0.01059s - 0.01269}{s^4 + 1.048s^3 + 0.7862s^2 + 0.01926s + 0.01197} \quad (17)$$

Non-Linear F16 Model The non-linear F16 model is a 6-DOF model based the equations-of-motion and parameters presented in [14, 17]. Model dynamics are solved via a fourth order Runge-Kutta method.

Simple PID Controller A simple PID controller was used for performance comparison. It is understood that PID is fully capable of controlling a linear plant very accurately. The transfer function in (18) was used as the PID controller for all linear models. The lack of a *derivative* term in (18) is due to the desire to use the simplest controller possible which yielded an adequate response. Since performance was adequate using only *proportional* and *integral* terms, the *derivative* term is left out to avoid complications due to large derivative inputs.

$$H_{PID}(s) = \frac{-0.5s + 0.05}{s} \quad (18)$$

4.4 CONTROLLER EVALUATION

Controller performance is measured as a factor of how fast the controller demonstrates a convergence to zero in the instantaneous root mean square error (19), where k is the sample index. So long as error, the base of the exponent term in the radicand, has a tendency to decrease over time, then as $k \to \infty$, $RMS_{error}(kT_s) \to 0$.

$$RMS_{error}(kT_s) = \sqrt{\frac{1}{k}\Sigma_{i=0}^{k}\left(\theta_{ref}(i) - \theta_{meas}(i)\right)^2} \quad (19)$$

The four combinations of F16 Linear Model 1 and F16 Non-Linear models and controllers in Table 1 are presented along with a graphical representation of the actual simulation and the instantaneous RMS error. Summaries of the F16 Linear Model 2 and Boeing 747 follow.

F16 Linear Model 1 and Non-Linear Model In the linear simulations, the linear F16 model presented in (12) was placed in the plant section of Fig. 11, and separate simulation runs were conducted with either the linear PID or fuzzy logic controller (FLC) placed in the controller section. Fig. 12 depicts the side-by-side comparison of the PID and FLC performance along with the input command reference which the controllers are meant to track where the initial command and pitch are both $\theta_{ref} = \theta_{meas} = 0\,\text{deg}$.

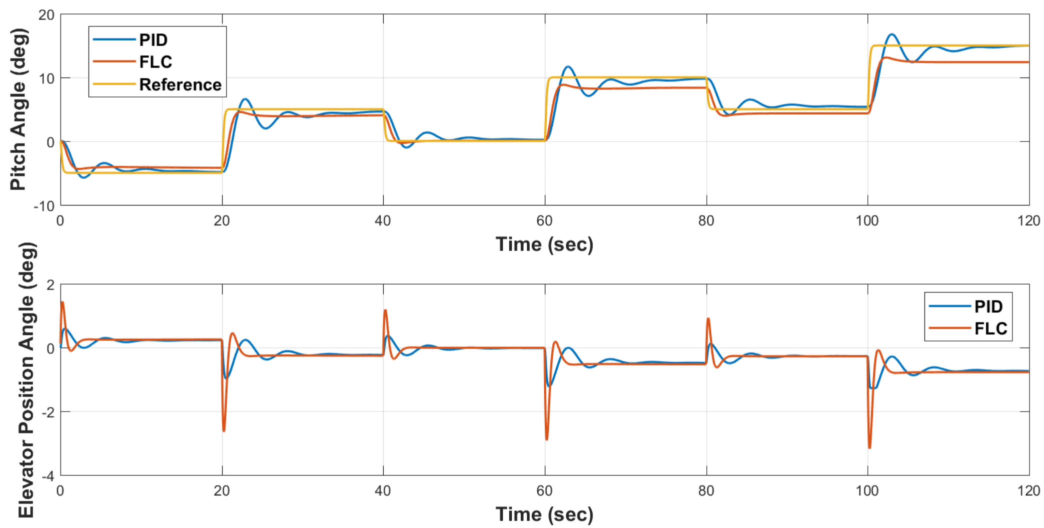

FIG. 12: Fuzzy Logic and PID Controller Response to Linear F16 Model 1

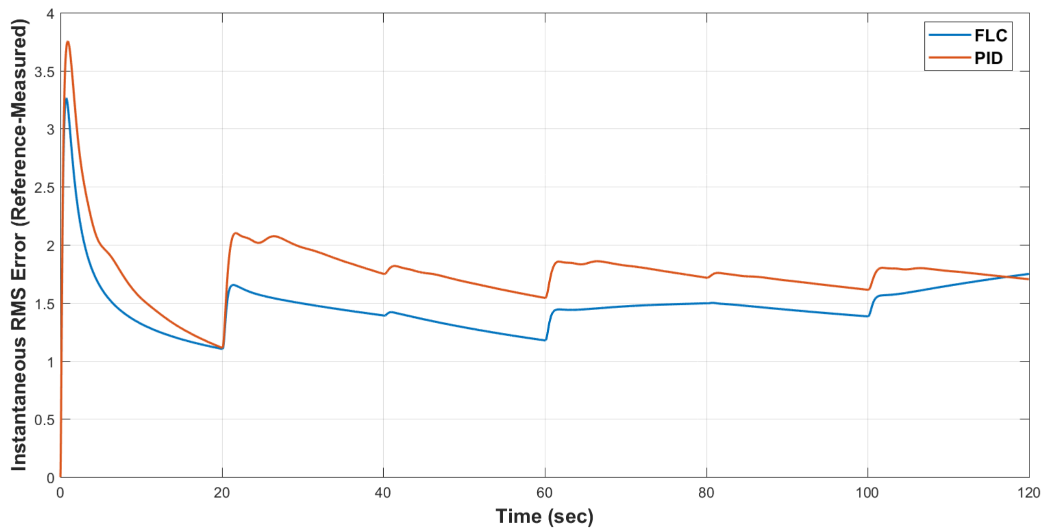

FIG. 13: Controller Performance Comparison Based on Instantaneous RMS Error for Linear Models

This graphic clearly depicts the superiority of the FLC with respect to steady-state time and tracking the reference command transient yet apparently lacking in overshoot and steady-state error. Fig. 13 shows the RMS performance of the two controllers. As one might expect, the PID controller is able to track commands with a decreasing RMS error over time.

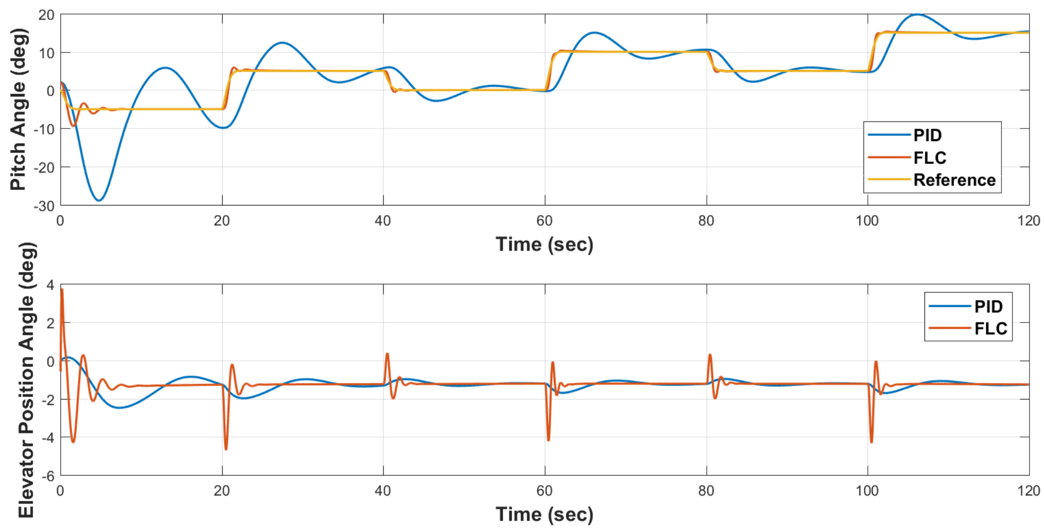

FIG. 14: Fuzzy Logic and PID Controller Response to Non-Linear F16 Model

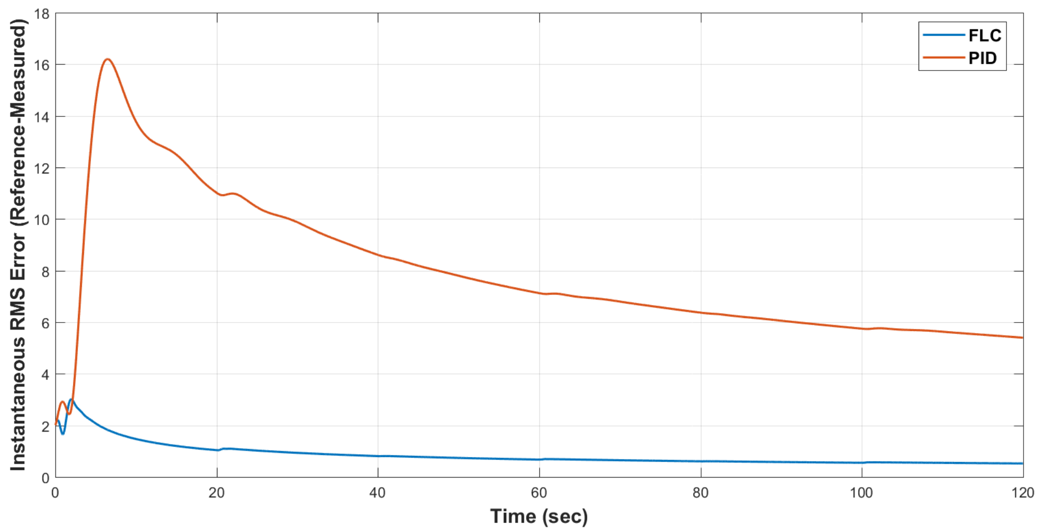

FIG. 15: Non-Linear F16 Model Controller Performance Comparison Based on Instantaneous RMS Error

The FLC RMS error begins to accumulate commands distant from the equilibrium condition due to the general constraints placed on the controller. This limitation is not considered significant here because RMS error performance increases in non-linear simulations. Specific performance numbers can be found in Table 4.

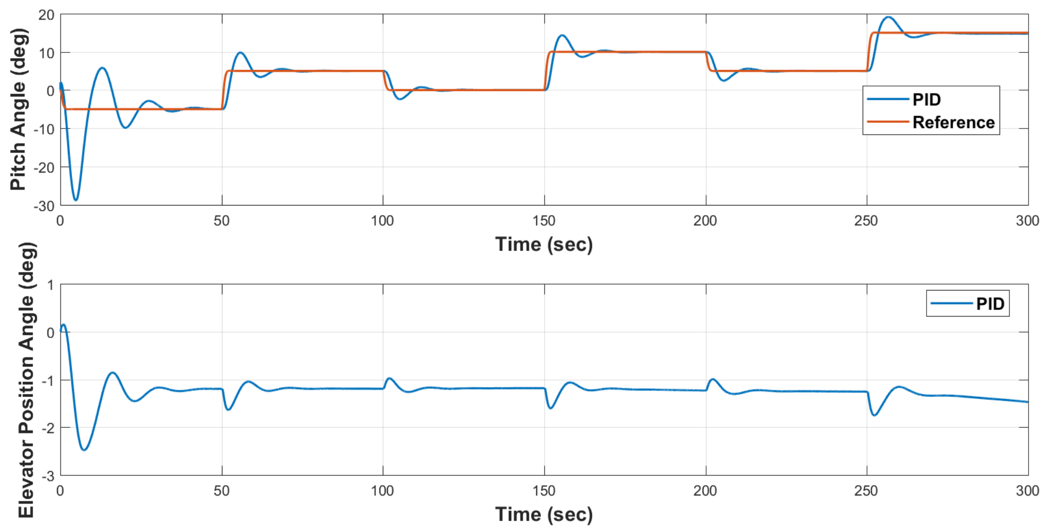

FIG. 16: PID Performance on Non-Linear Plant Demonstrating Tracking During Long Duration Doublets

The performance differences of the non-linear simulations are, as expected, considerably different. Long period simulations, Fig. 16, confirmed the ability of the PID controller to track the reference input and are substantiated by the apparent monotonic decrease of RMS error, Fig 15. It is not considered as an acceptable *in situ* controller due to the large overshoot at all command changes, but its simulation is provided here for continuity and comparison. Specific performance numbers can be found in Table 4.

The previous simulations used a static controller with preset bounds and no adaptivity. This resulted, as expected, in a fuzzy logic controller of better performance to a non-linear plant than that of a linear PID controller due to its inherent non-linearity; yet steady-state error still remained. Specific performance numbers can be found in Table 4 where *overshoot*, *undershoot*, and rise-time are calculated using the standard definitions and *steady-state error* is calculated at the last sample instance for a time period.

F16 Linear Model 2 The simulation present in Fig. 18 was performed using the same input criteria as the F16 Linear Model 1 and F16 Non-Linear model. Specific performance characteristics are presented in Table 5 for comparison. It is interesting to note that the FLC model performs quite poorly; large overshoot and sustained oscillations appear to nearly grow unbounded during the interval $60\,\text{sec} \leq t < 80\,\text{sec}$, but it recovers at the last moment. However, once this time period passed, the FLC controller performed with overshoot and

TABLE 4: F16 Linear Model 1 and F16 Non-Linear Model Controller Simulation Performance Characteristics

		Time Period (sec)					
		0-20	20-40	40-60	60-80	80-100	100-120
F16 Linear 1 PID	Over(under)shoot (%)	15.37	14.77	13.24	15.01	13.13	14.03
	Rise Time (sec)	1.30	1.34	1.44	1.34	1.38	1.44
	Steady-State Error (deg)	-0.13	0.37	-0.19	0.21	-0.37	0.01
F16 Linear 1 FLC	Over(under)shoot (%)	-11.98	-12.23	-12.54	-12.01	-12.21	-12.23
	Rise Time (sec)	4.58	1.44	1.32	4.58	1.06	4.48
	Steady-State Error (deg)	-0.78	0.96	-0.04	1.64	0.66	2.62
F16 Non-Linear PID	Over(under)shoot (%)	596.12	123.19	76.44	53.33	67.82	50.92
	Rise Time (sec)	0.56	1.52	1.68	1.86	1.64	1.86
	Steady-State Error (deg)	4.91	-0.74	0.29	-0.56	0.29	-0.34
F16 Non-Linear FLC	Over(under)shoot (%)	130.65	9.13	10.71	3.35	6.18	3.27
	Rise Time (sec)	0.36	0.62	0.58	0.62	0.54	0.58
	Steady-State Error (deg)	-0.00	-0.01	0.01	-0.00	0.01	0.01

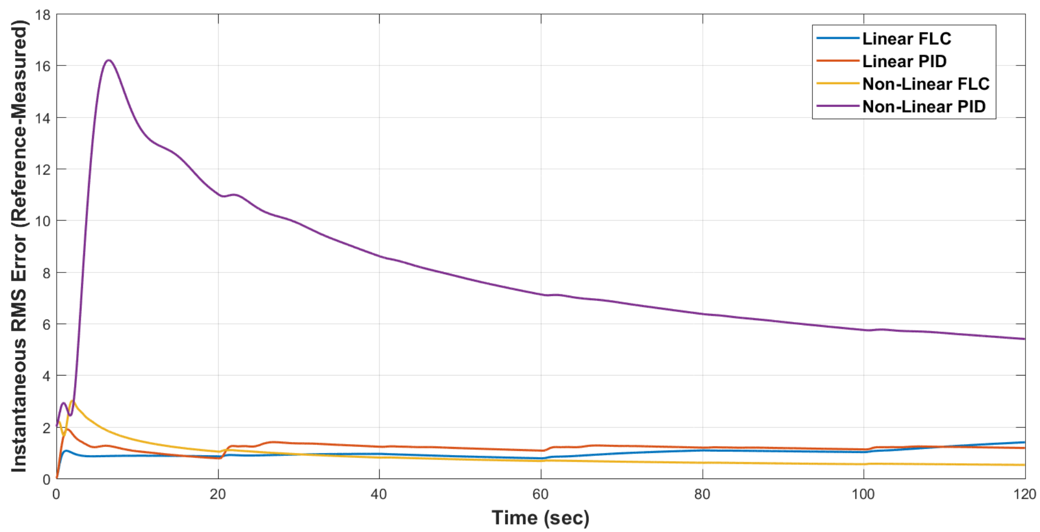

FIG. 17: F16 Linear Model 1 and Non-Linear F16 Model Controller Performance Comparison Based on Instantaneous RMS Error

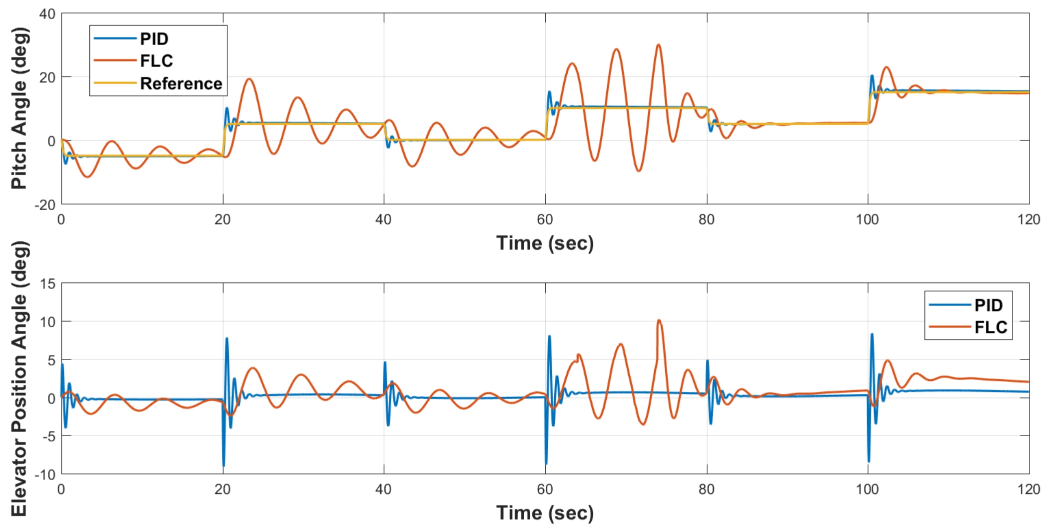

FIG. 18: Fuzzy logic and PID response to the F16 Linear Model 2

steady-state performance comparable to that of the PID controller but with a longer rise time.

TABLE 5: F16 Linear Model 2 Controller Simulation Performance Characteristics

		Time Period (sec)					
		0-20	20-40	40-60	60-80	80-100	100-120
F16 Linear 2 PID	Over(under)shoot (%)	53.26	53.30	53.41	53.31	53.40	53.30
	Rise Time (sec)	0.20	0.18	0.18	0.20	0.20	0.20
	Steady-State Error (deg)	0.11	-0.11	-0.02	-0.19	-0.12	-0.28
F16 Linear 2 FLC	Over(under)shoot (%)	134.30	146.48	192.46	299.62	83.64	76.22
	Rise Time (sec)	0.90	0.78	0.58	0.86	1.44	0.80
	Steady-State Error (deg)	0.10	0.20	-0.73	1.70	-0.29	0.33

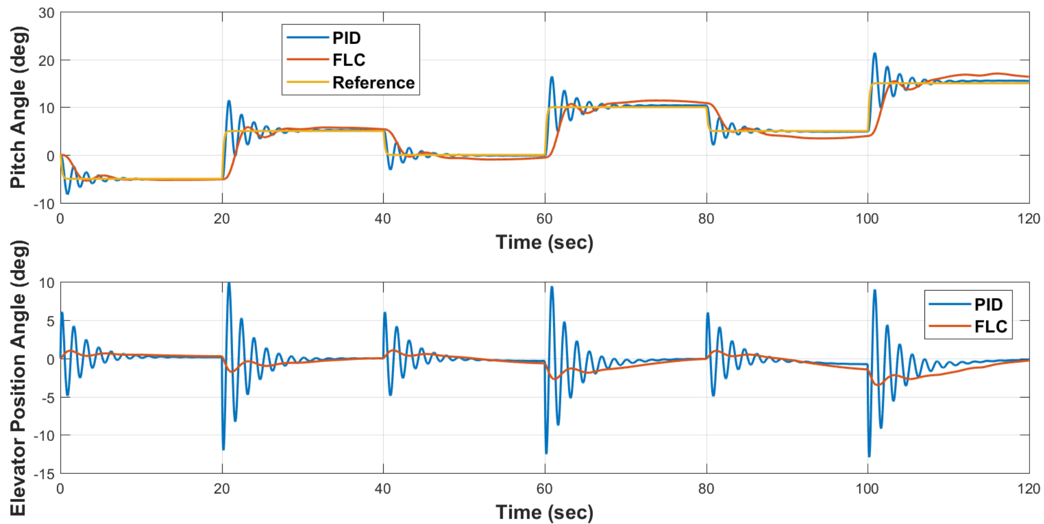

FIG. 19: Fuzzy logic and PID response to the Boeing 747 Linear Model

Boeing 747 Linear Model The simulation present in Fig. 19 was performed using the same input criteria as the F16 Linear Model 1 and F16 Non-Linear model. Specific performance characteristics are presented in Table 6 for comparison. The FLC controller in this simulation performs quite well with regard to overshoot but suffers steady-state error compared to the PID controller.

4.5 EXPERIMENTATION HYPOTHESIS

From the simulations presented, particularly the non-linear model simulations, one should expect the fuzzy logic control scheme to work well. Overshoot is expected to diminish over time; rise time should match the command reference within an order of magnitude, and steady-state error should be negligible. One should also expect to tune the controller and supervisor initialization parameters corresponding to the maximum control boundaries of the supervisor and controller based on data gathered during experimentation. More specifically, the output scaling gains from the supervisor will likely need adjustment to obtain the performance requirements as defined in Section 1.2.

TABLE 6: Boeing 747 Linear Model Controller Simulation Performance Characteristics

		Time Period (sec)					
		0-20	20-40	40-60	60-80	80-100	100-120
B747 Linear PID	Over(under)shoot (%)	65.24	65.25	65.36	65.37	65.52	65.42
	Rise Time (sec)	0.30	0.30	0.30	0.30	0.28	0.30
	Steady-State Error (deg)	0.10	-0.15	0.11	-0.33	0.10	-0.49
B747 Linear FLC	Over(under)shoot (%)	8.06	9.26	26.93	19.42	47.80	28.96
	Rise Time (sec)	1.52	1.48	1.42	1.46	1.40	1.44
	Steady-State Error (deg)	0.12	-0.38	0.56	-0.85	1.04	-1.35

CHAPTER 5

EXPERIMENTATION

5.1 EXPERIMENTAL DATA COLLECTION

5.1.1 EXPERIMENTAL APPARATUS

The performance of the model-less fuzzy logic control system was tested with two different aircraft model configurations, a known stable *stock* configuration and a known unstable *Leading Edge Extension (LEX)* configuration, to demonstrate its ability to control aircraft with significantly different aerodynamics. The model under test was a commercially available scaled model of the Aero L-59 Super Albatros. The model was operated in the 12-foot wind-tunnel located at the NASA Langley Research Center in Hampton, VA. The pitch control configuration, shown in Fig. 1, was obtained by fixing the model such that only 1-DOF was obtained. The stock configuration was comprised of the standard L-59 model aircraft outfitted with control actuators and micro-controller stack affixed to a sting providing Free-to-Pitch functionality. The LEX configuration was obtained by attaching aluminum leading edge extensions, as shown in Fig. 22. This aircraft is capable of differential elevator movement through independent articulation, control servos, and instrumentation; however, they were controlled as a single unit in this experiment.

The aircraft was instrumented with US Digital MA3 12-bit PWM Magnetic Encoders [22] filtered with a Krohn-Hite Model 3364 4-pole Butterworth filter [23], using a cutoff frequency of 10 Hz, to measure pitch. Control surface movement was accomplished using an Arduino DUE micro-controller [24], Seeed Technology W5200 Ethernet shield [25], and SparkFun Ludus Protoshield Wireless motor shield [26] stack attached to Futaba S9650 [27] digital servo motors. Elevator feedback measurement was accomplished via US Digital MA3 12-bit PWM Magnetic Encoders connected to the control surfaces.

The Arduino DUE micro-controller served as an interface between the primary control program run on an external computer via Matlab and the servo motors and the angle position sensors. Fig. 20 shows the controller hardware stack where the *Arduino Stack* region is taken as a single unit to which the digital encoders and servos connect. Each sample period

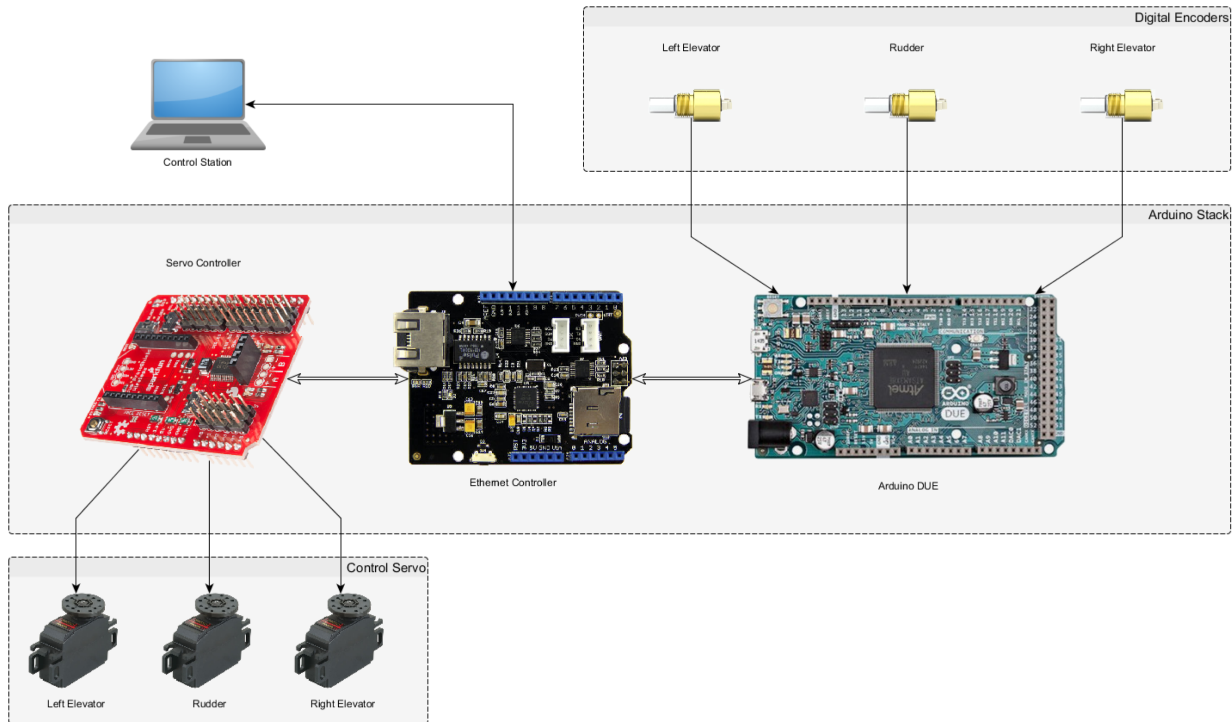

FIG. 20: Hardware Stack

follows the programmatic flow depicted in Fig. 21 where the Matlab program read the current elevator angle measure from the micro-controller, read the current pitch angle measurement from the Butterworth filter, calculated the control output, issued the control command to the micro-controller, and logged data from the sample period of off-line analysis. Fixed-point to floating-point conversions were handled where appropriate by the Matlab program.

The entire experiment was driven by a proprietary Matlab program developed by NASA Langley. This program set up all appropriate interface requirements for the data acquisition interfaces, generated the experiment control commands, and connected to the application program interface of the controller. The flowchart depicted in Fig. 21 depicts the run-time procedural loop.

5.1.2 EXPERIMENTAL PROCEDURE

The experimental procedures are as follows:

1. Setup

    (a) Install aircraft.

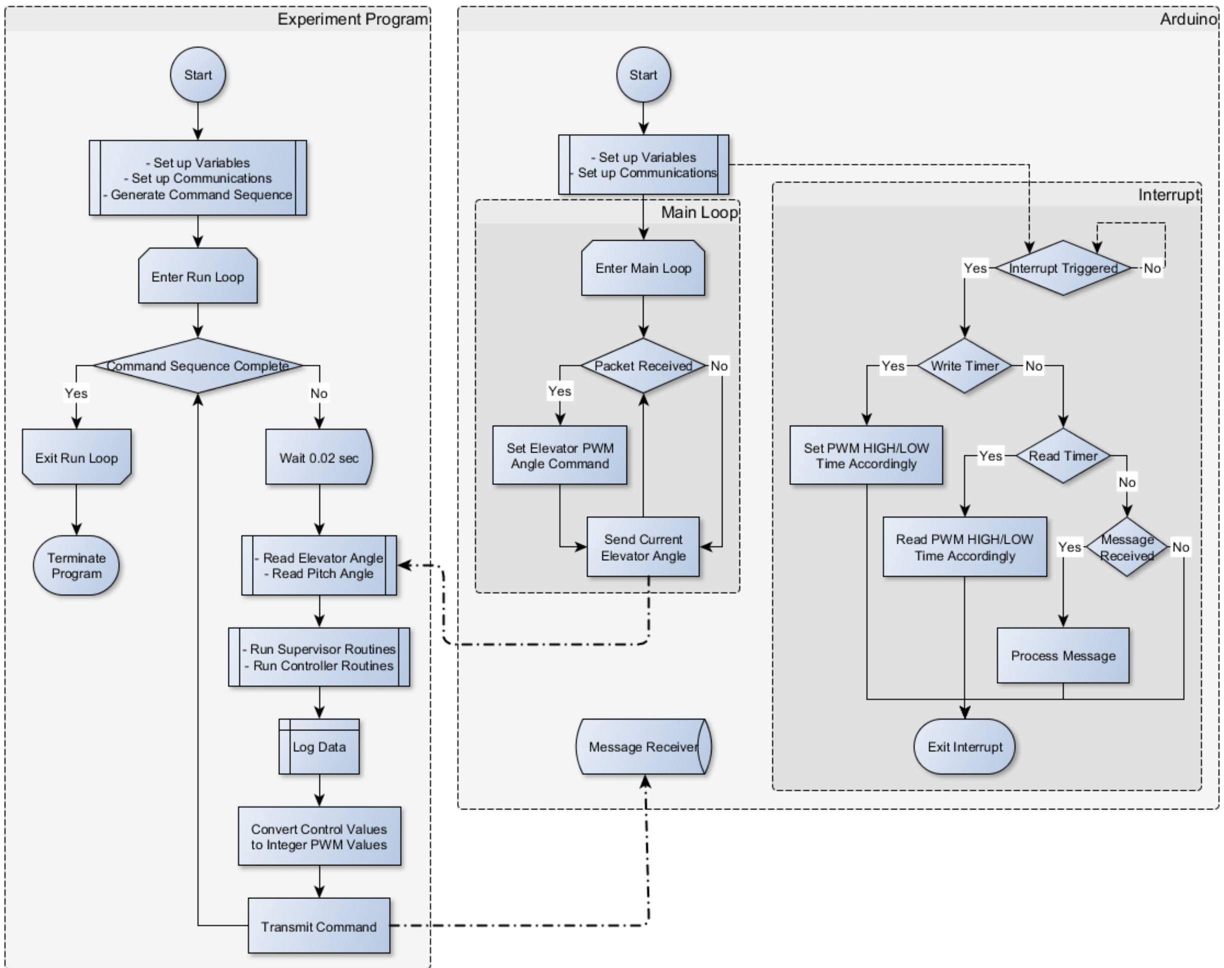

FIG. 21: Run-time Program Flowchart

(b) Ensure hard stops are placed such that the aircraft cannot exceed its expected pitch limits.

(c) Install instrumentation.

(d) Check sensor connectivity by reading sample data through a dummy Matlab interface.

(e) Check actuator connectivity by issuing dummy commands to the servo motors.

(f) Calibrate servo motors.

i. Repeatedly issue port elevator commands until the chord line of the elevator is collinear with the chord line of the horizontal stabilizer.

ii. Record data as $\delta_e = 0\,\text{deg}$.

iii. Using a block with a 30 deg angle mounted on the ventral side horizontal stabilizer, repeatedly issue port elevator commands until the elevator is flush with block.

iv. Record data as $\delta_e = 30\,\text{deg}$.

v. Repeat Step 1(f)iii with the block mounted on the dorsal side of the horizontal stabilizer.

vi. Record data as $\delta_e = -30\,\text{deg}$.

vii. Repeat Steps 1(f)iii–1(f)vi with a 45 deg block.

viii. Repeat Steps 1(f)iii–1(f)vii for the starboard elevator.

(g) Input calibration values into the control sequence to map control values to actual elevator commands.

(h) Verify calibration by issuing control commands in degrees and verifying accuracy with calibration blocks.

2. Experimental Run

(a) Generate command sequence.

(b) Set wind-tunnel airspeed as a function of dynamic pressure, q.

(c) With the wind-tunnel at the desired q, run program.

(d) At program conclusion, save the Matlab workspace data for post analysis and archival purposes.

(e) Determine any necessary program changes.

(f) Update control algorithms as necessary.

(g) Repeat Steps 2a–2f as necessary.

(h) Set wind-tunnel $q = 0$.

Open-loop runs were performed by issuing a series of sequential step commands over a large portion of the flight envelope. Open-loop runs were conducted to verify stability or instability of a configuration prior to closed-loop runs in order to draw performance comparisons. Open-loop and closed-loop mathematical characterization of the plant configurations are not a goal of this study and were not performed.

TABLE 7: Controller Initial Values

Experiment Parameter	**Value**
$\delta_{e_{\text{lim}}}$	40 deg
θ_{lim}	25 deg
$\max\lvert\theta_{error}\rvert$	50 deg
$\max\lvert\theta_{\Delta_{error}}\rvert$	100 deg
Supervisor Proportional Output Scale	40
Supervisor Integral Output Scale	10
Supervisor Derivative Output Scale	800

5.2 PERFORMANCE CHARACTERISTICS

The proposed controller was tested on both the stock aircraft configuration, Fig. 1, and the LEX aircraft configuration, Fig. 22. Both configurations were tested under open-loop and closed-loop conditions to compare controller effectiveness. Controller initialization parameters for the runs shown here are provided in Table 7.

The stock configuration received doublet commands for both open-loop and closed-loop test runs. In the open-loop case, pretest calibration was performed to find the steady-state trim condition of the aircraft, that is, elevator commands $\delta_e \in [-10\,\text{deg}, 10\,\text{deg}]$ were issued, the aircraft was allowed to reach a steady-state operating point, and the pitch angle was

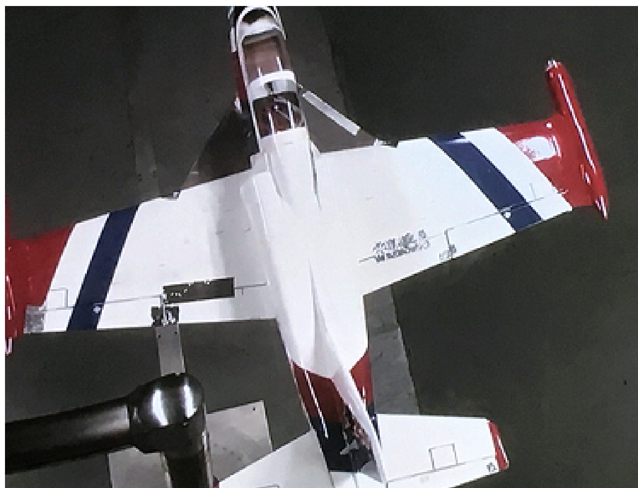

FIG. 22: LEX Aircraft Configuration

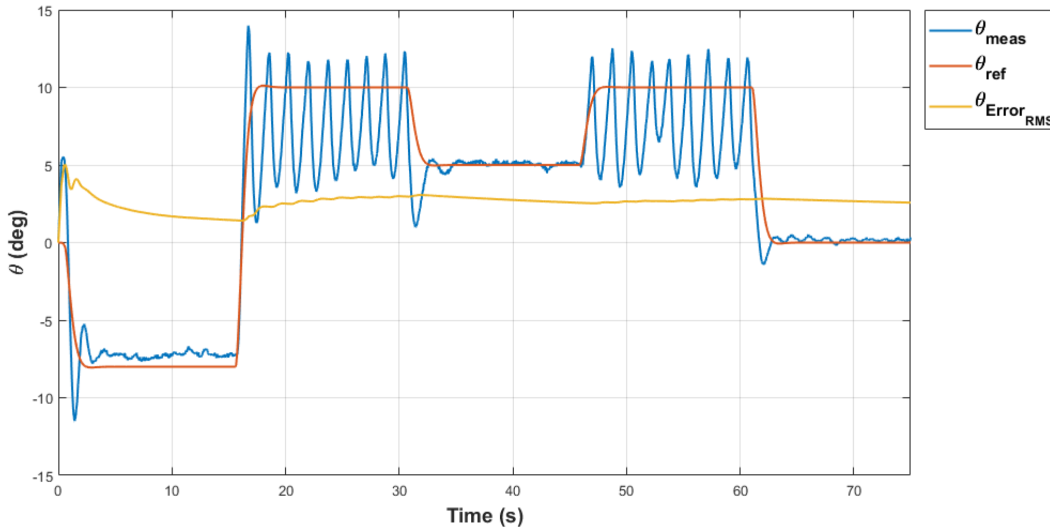

FIG. 23: Stock Aircraft Response to Doublet Input (Open Loop)

recorded; this is the steady-state trim value. These values were then used to command open-loop pitch by issuing the appropriate elevator command associated with desired pitch; these values were only used for open-loop testing. Fig. 23 shows the open-loop response of the stock configuration. The open-loop aircraft is unable to maintain trim near $\theta = 10\,\mathrm{deg}$, overshoot is present during transitions, and steady-state error is noted over the interval $(5\,\mathrm{s} < t < 15\,\mathrm{s})$.

The closed-loop stock configuration was tested with a series of doublet commands in the range $[-10\,\mathrm{deg}, 10\,\mathrm{deg}]$ as θ_{cmd}. These commands were filtered to generate θ_{ref}. Fig. 24 shows the performance of the controller with no plant knowledge. The instantaneous RMS error given in (19) has been overlain to show the RMS error asymptotically approaching zero. Observe that the closed-loop controller maintains trim at $\theta = 10\,\mathrm{deg}$ with no oscillations or overshoot. The overshoots at step changes have been eliminated and steady-state error approaches $\theta_{error} \to 0$ asymptotically.

The period of initial learning is shown in Fig. 25. During this time frame, the controller's initial output of $\delta_e = 0\,\mathrm{deg}$ caused the aircraft to pitch down to $-13\,\mathrm{deg}$. At $t = 0.5\,\mathrm{s}$, the controller's FIS structures have recovered and began corrective action by $t = 1\,\mathrm{s}$.

The LEX aircraft configuration is shown in Fig. 22. Aluminum sheets were added to the leading edge of the wings to create aerodynamic instability by shifting the aircraft's aerodynamic center forward with respect to its center of mass. Open-loop LEX configuration testing was performed by sweeping the elevator angle in step commands over the range

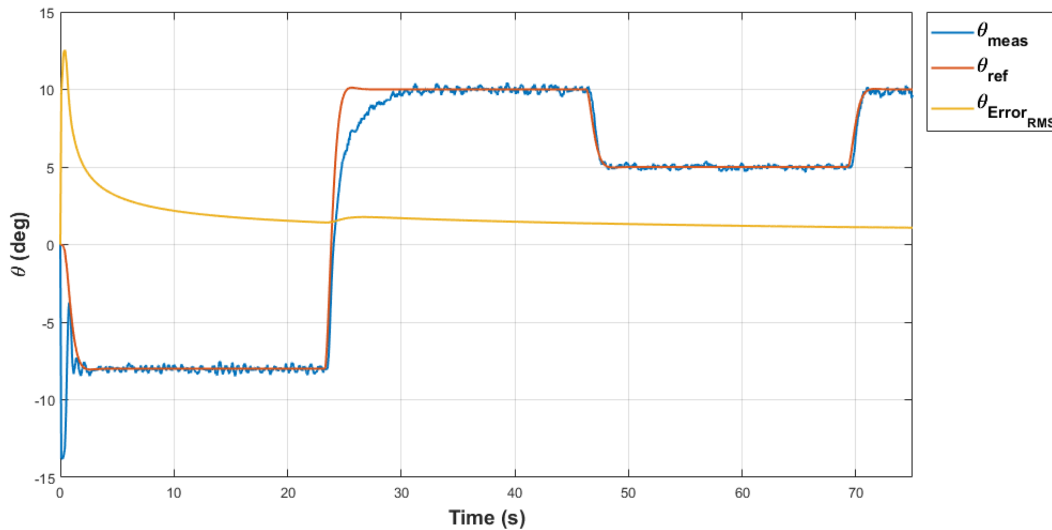

FIG. 24: Stock Aircraft Response to Doublet Input (Closed Loop)

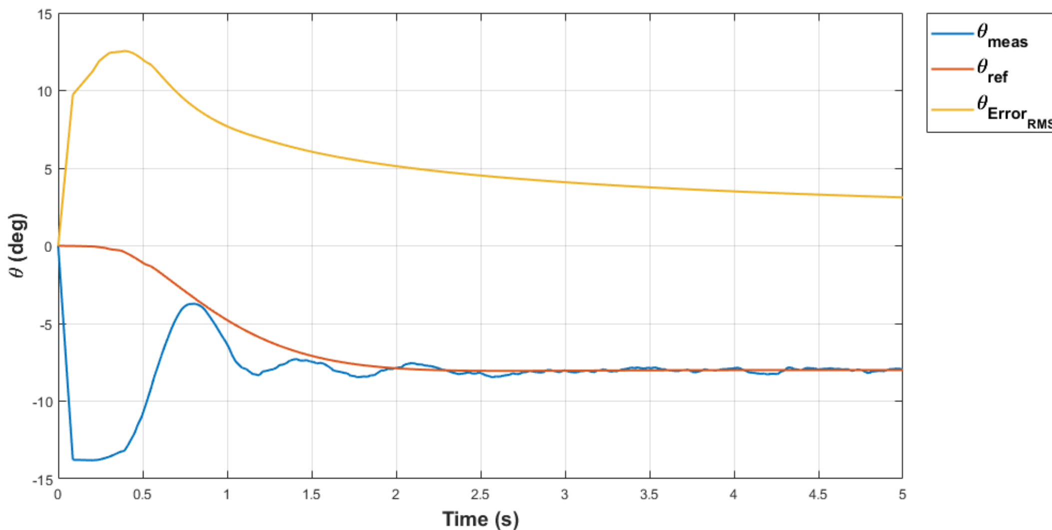

FIG. 25: Stock Aircraft Initial Self-Tuning Time Frame

$\delta_e \in [-10\,\text{deg}, 10\,\text{deg}]$. Fig. 26 shows the uncontrollable nature of this configuration where the pitch response is maintained above $\theta = 10\,\text{deg}$ for $\delta_{e_{cmd}} < 2\,\text{deg}$ at which point it sharply transitioned to $\theta = -35\,\text{deg}$.

The closed-loop response of the LEX configuration shown in Fig. 27 demonstrates control recovery. While control performance does not match the stock closed-loop response, stable

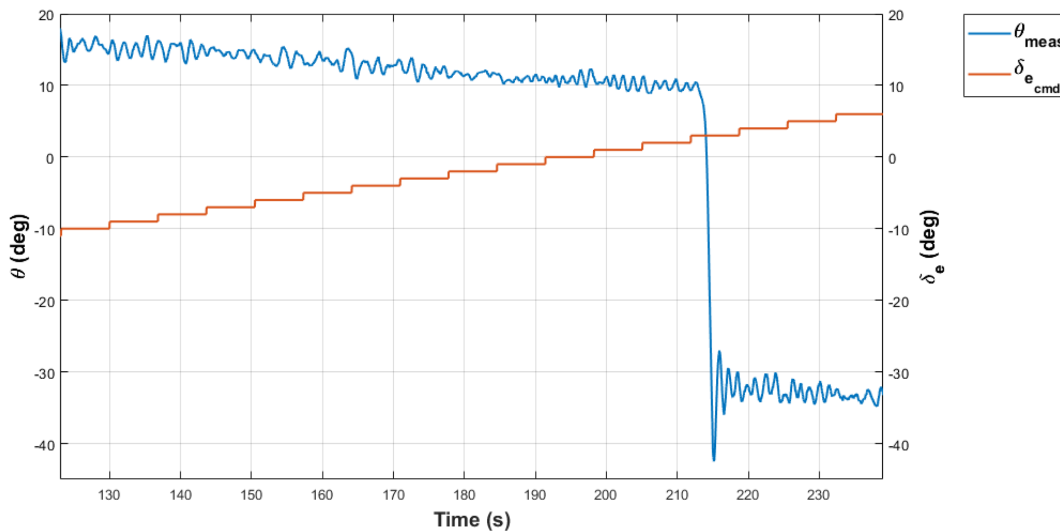

FIG. 26: LEX Aircraft Response to Doublet Input (Open Loop)

flight, the goal of this study, has been achieved. Note that $\theta_{cmd} = 5\,\text{deg}$ contains slow oscillatory convergence. This is evidenced graphically in the reduction of oscillation amplitude peak values over time.

Initial parameters pertaining to the sensitivity of the supervisory system required manual adjustments between test runs to achieve the results presented here. The initial parameters were set based on simulation performance. To reduce the steady-state error in wind-tunnel experiments, the supervisor's scaling factor for the integral path was increased from $\frac{1}{24}$ to $\frac{1}{4}$ of the scaling factor of the proportional path. These fixed parameters represent the maximum gains available for the system to create a range of possible values for the controller to choose. This resulted in the characteristic performance presented above and points to how to improve the controller in the future, for example, adding adaptive subroutines to adapt the scaling weights, which were fixed, based on detected performance.

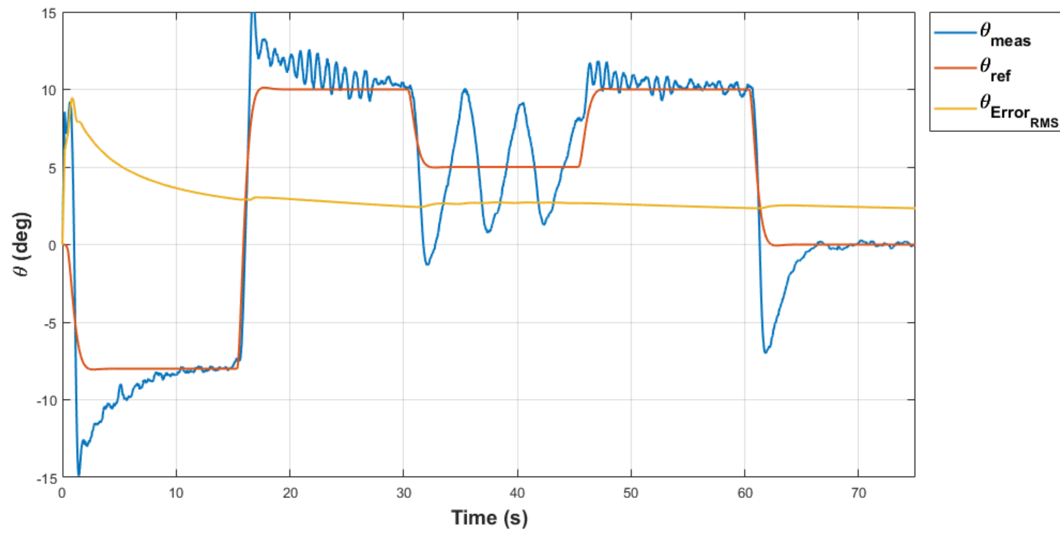

FIG. 27: LEX Aircraft Response to Doublet Input (Closed Loop)

CHAPTER 6

SUPPLEMENTAL WORK

6.1 MOTIVATION

The technique presented thus far is helpful under the conditions presented. However, visual analysis of Figs. 14 and 27 alone are enough to suggest a need to further refine the presented technique. Thus, the adaptivity theory was expanded to incorporate on-line tuning of the FLC. This was done by tracking the operating point within the six-dimensional structure of a normalized FLC and adjusting the dimensional boundaries according to certain criteria. In general, a given system input or output is allowed to operate near its boundary for a limited period of time, but if it is operating near the boundary too often, there the controller is in danger of operating in saturation.

6.2 METHODOLOGY

If, in general, one needs to make supervisor-controller adjustments based on the six-dimensional operating point, the question becomes, "When, where, and how often are these changes to be made?" In short, adjustments should be made when "error along a given supervisor or controller path is too great," where the "operating point is too close to or exceeds a boundary," and "not too often."

The adaptivity routines are applied uniformly to all inputs and outputs. Fig. 28 depicts the decision flow for input adaptation. Specifically, if an input is within a certain percentage of the boundary of the associated FLC, then an accumulator increments up to a certain threshold, with credit given when within the *normal* range, where the *normal* range is defined as a certain percentage of the total range. For example, if an input registers 15 consecutive boundary excursions followed by three *normal* range values, the final accumulator value for this time would be 12 – assuming an initial accumulator value of zero. This is to allow brief excursions near the controller's operating boundary, but long-term excursions occur when, presumably, the controller is ill-conditioned for the plant; thus, adjustment is required. Once the threshold is exceeded, the boundary is increased by a specific multiplier. Additionally, in this model, inputs completely outside the input boundary prompted an immediate change so that out-of-range values may be handled.

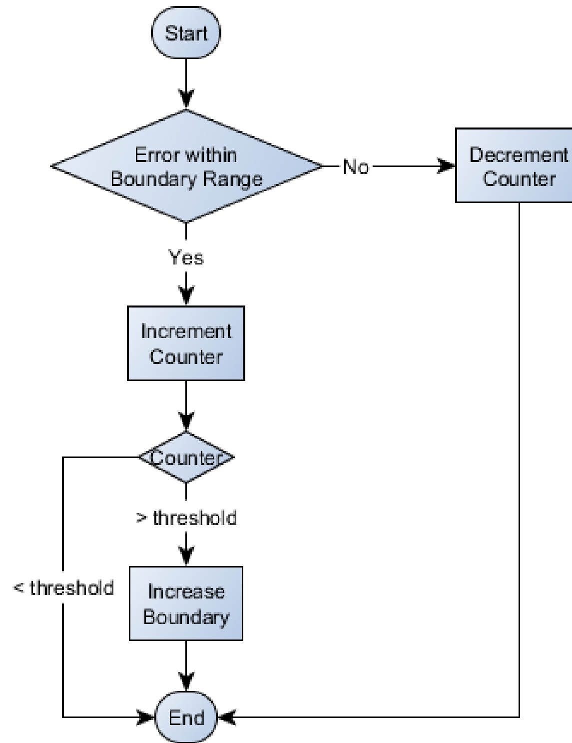

FIG. 28: Input Adaptation Algorithm

The output adaptation algorithm depicted in Fig. 29 utilizes a similar boundary detection scheme as the inputs. Here it is assumed that the output performance is strictly coupled to the output range of the supervisor-controller combination related to a certain PID control path. Specifically, *Error Magnitude* in Fig. 29 relates to $\theta error$ for the proportional path, $\dot{\theta}_{error}$ for the derivative path, and $\Sigma(\theta_{error})$ for the integral path and a change along the respective control path effects change in the associated output.

Specifically, if the specific error or average error over a given sample window exceeds a certain threshold, a series of decisions is made. First, a check is performed to make sure that the controller has not been updated too recently; this is accomplished using a stand-down counter which decrements every sample period after a change in addition to an accumulator which keeps track of the number of recent error excursions.

Once a change event is initiated, the mode of the error is determined. If the operating point of the controller is within a certain distance of, or over the boundary, the error is likely due to an aggressive controller regime, so the output boundaries are reduced in order to reduce the magnitude of the outputs. Conversely, if the controller is operating within the

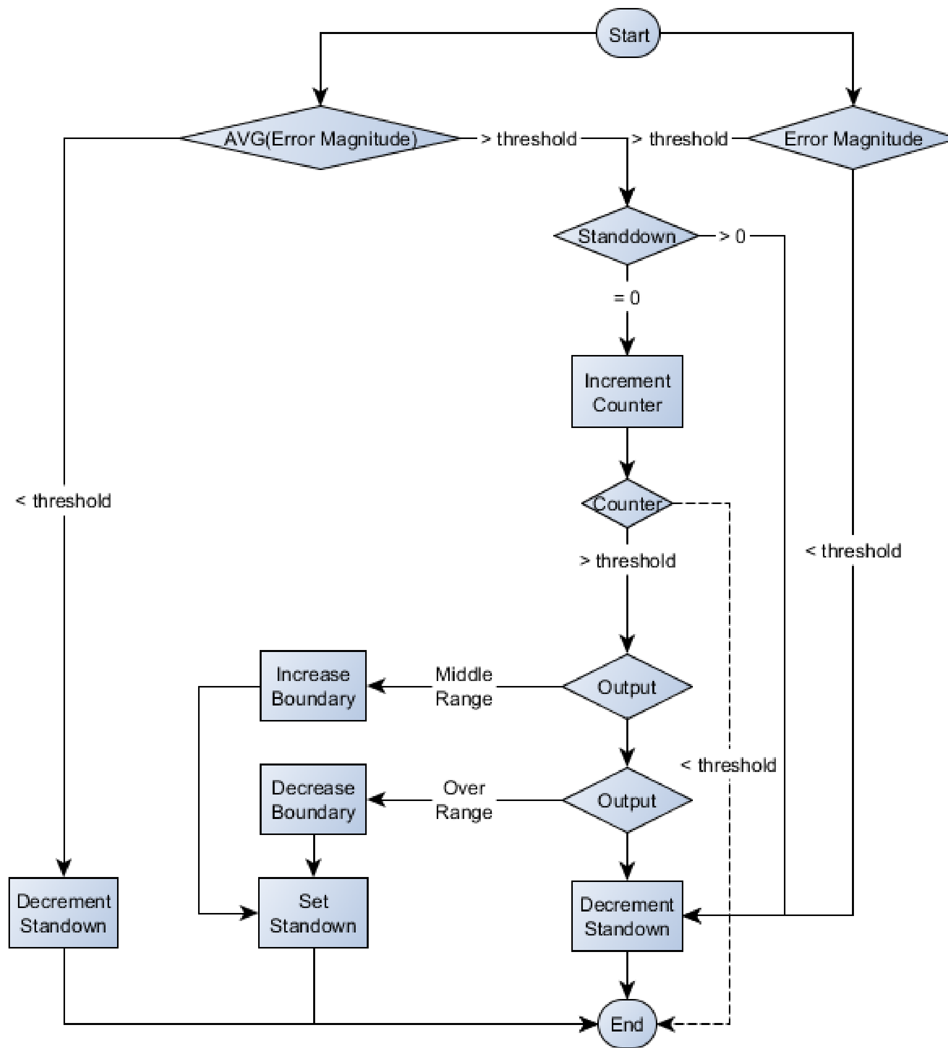

FIG. 29: Output Adaptation Algorithm

normal region, then the boundaries must be expanded to provide more control authority.

6.3 SIMULATION

Simulation of the adaptation algorithms was performed using the same analysis technique described in Chapter 4 while the adaptive controller was used to control both a linear and non-linear F16 model plant. The adaptation parameters described in the previous section are given in Tables 8 and 9.

Surprisingly, the adaptive supervisor-controller structure performed substantially better against a non-linear plant than a linear plant. Fig. 30 shows ringing throughout the entire

TABLE 8: Controller Output Adaptation Parameters

	Output		
	K_p	K_i	K_d
Counter	10	10	10
Stand-down	25	50	25
Increase Multiplier	2	1.5	4
Decrease Multiplier	0.5	0.67	0.25
Boundary Region	25%	25%	25%
Window Size	20	20	20
Error Threshold	5	25	5
Avg. Error Threshold	15	15	15

TABLE 9: Controller Input Adaptation Parameters

	Input					
	P	I	D	K_p	K_i	K_d
Counter	10	10	10	10	10	10
Increase Multiplier	2	1.5	4	2	1.5	4
Decrease Multiplier	0.5	0.67	0.25	0.5	0.67	0.25
Boundary Region	25%	25%	25%	25%	25%	25%

linear simulation, with the largest amount during the first 20 seconds. This oscillation was quickly damped but was not completely eliminated and contributed to the larger RMS error shown in Fig. 31 over the non-linear simulation.

However, against the non-linear plant, the adaptive routines corrected rather directly to a final suitable condition. It is true that approximately 18-percent overshoot occured at each transition; however, steady-state error is effectively zero, Table 10, and met the design criteria for stability. Lastly, a comparison is given in Figs. 32 and 33 showing the performance of all non-linear controllers against one another.

TABLE 10: Adaptive Controller Simulation Performance Characteristics

		Time Period (sec)					
		0-20	20-40	40-60	60-80	80-100	100-120
Linear FLC Adaptive	Overshoot(%)	224.79	119.52	51.22	43.12	45.48	40.68
	Rise Time(sec)	7.50	7.22	7.32	7.30	7.38	7.32
	Steady-State(deg)	5.47	-0.24	0.09	-0.10	0.04	-0.08
NonLinear FLC Adaptive	Overshoot(%)	238.16	26.10	0.94	31.69	4.54	33.03
	Rise Time(sec)	0.26	3.50	1.36	2.86	1.26	2.24
	Steady-State(deg)	-0.78	0.05	-0.24	-0.08	-0.06	-0.01

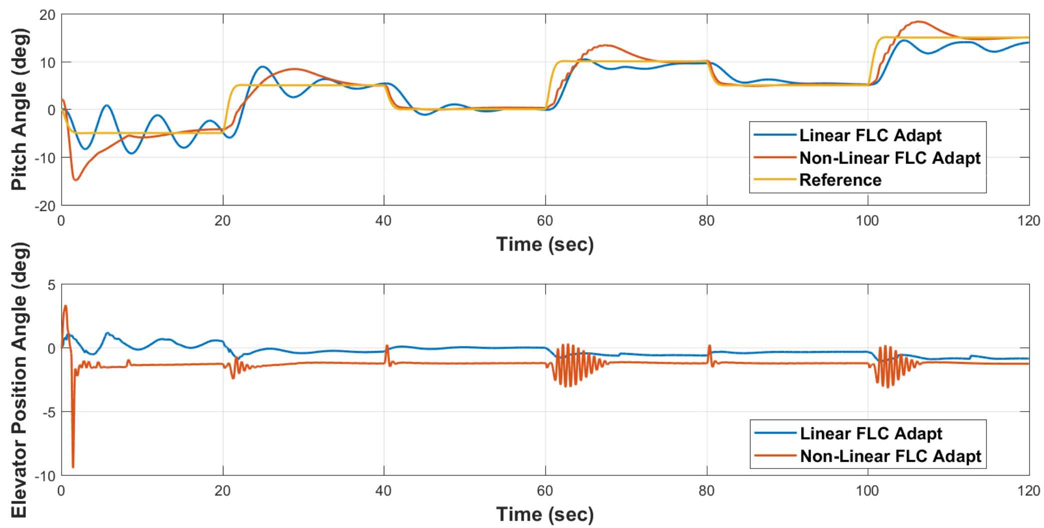

FIG. 30: F16 Linear Model 1 and Non-Linear F16 Model Fuzzy Logic Controller Response With Adaptive Routines

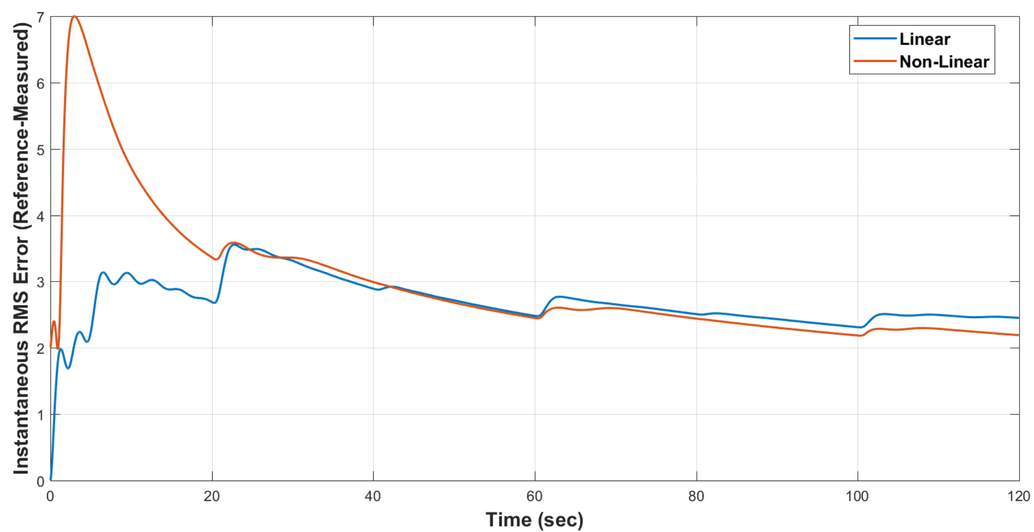

FIG. 31: F16 Linear Model 1 and Non-Linear F16 Model Controller Performance with Adaptivity Comparison Based on Instantaneous RMS Error

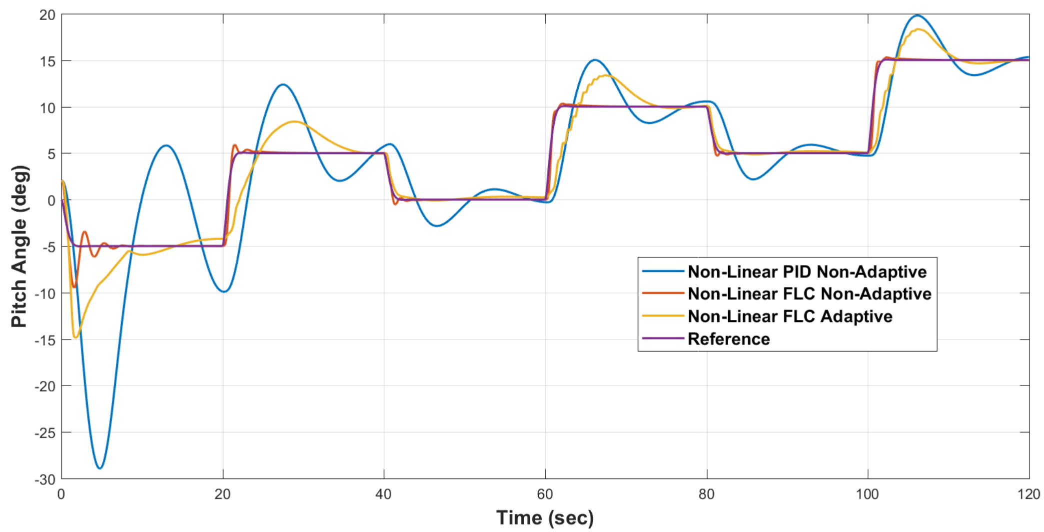

FIG. 32: Fuzzy Logic and Linear PID Controller Response to Non-Linear F16 Model with Adaptivity Simulation Shown

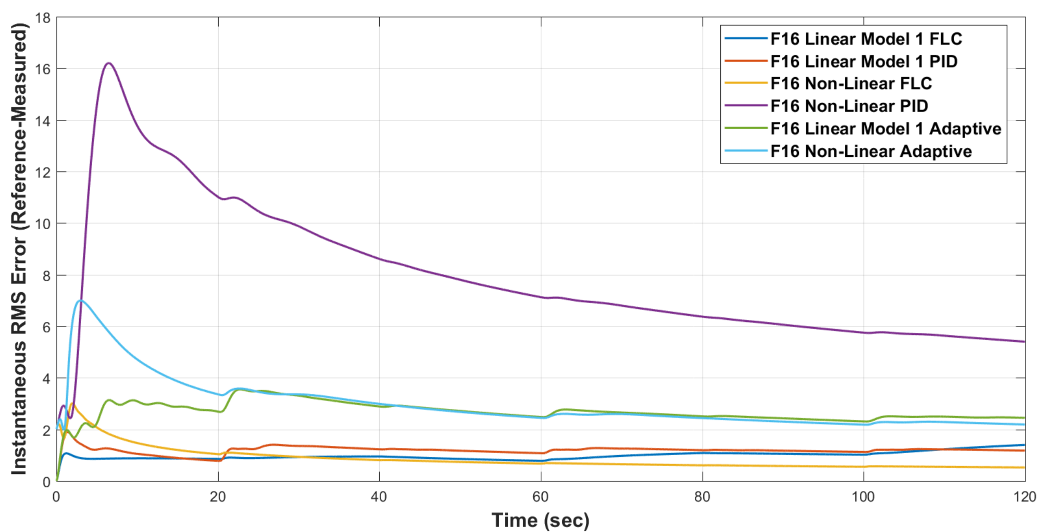

FIG. 33: Simulated Performance Comparison Based on Instantaneous RMS Error

CHAPTER 7

CONCLUSION

7.1 OVERVIEW OF FINDINGS

In general, the proposed control scheme operated well and met the primary design goal of creating stable flight for an unknown plant of a certain type – a fixed-wing aircraft obeying the standard aerodynamic equations of motion. Without further augmentation, this technique appears to be quite useful for control of open-loop stable plants but suffers from performance degradation for open-loop unstable plants. However, when used as a tool to provide stability for the development of robust plant models via SID in order to yield more powerful control schemes, its usefulness is apparent in the reduction of development time gained.

The primary lesson to be learned is that while general success was obtained, it would be foolish to consider this technique fully mature. The performance change from stable to unstable plants alone is enough to demonstrate that the static fuzzy logic controller presented is probably not robust enough to handle a wide variety of unstable plants. Additionally, the need to tune scaling gains proves that, in order to make this technique available for a wider variety of plants, on-line adaptive routines could be utilized to great effect.

7.2 RESEARCH IMPLICATIONS

Traditionally, classical and modern control methodologies are rooted in mathematical models, and for good reason. Accurate mathematical models allow design and simulation iterations to evolve in systematic progressions without costly manufacturing or risky real world testing. Furthermore, mathematics provides the analysis tools to understand how those design and simulation iterations point to desired results in the form of tools like trend-lines. Yet, in certain circumstances this approach may itself be too costly if only a quick general answer is required or if the time to develop models requires too many hours.

Conversely, the presented approach demonstrates that it is possible to create a real-world solution which abstracts mathematical models, allows for stable control of a plant, and supplies a qualified result which may be used to obtain a further quantified result. Furthermore, by utilizing wind-tunnel data logging and control systems, the development of mathematical models via SID can be automated, resulting in lower overall prototyping cost.

7.3 SUMMARY

In this paper, a fuzzy logic controller was developed for the Modeling and Control for Agile Aircraft Development Program. The proposed controller required no *a priori* mathematical aircraft model, and only a second-order reference model was used for response shaping. Control was obtained by creating a fuzzy inference system such that general *expert knowledge* was embedded in the input-output space of the FIS to sufficiently cover the operational flight envelope.

Real-time adjustments were performed by a supervisory system created with a second set of FISs. This subsystem proved to adapt to unknown aircraft configurations to control both aerodynamically stable and unstable aircraft configurations semi-autonomously. While certain parameters required tuning to set up parameters for the beginning of an experimental run, this yielded the necessary intuition required to fully automate parameter tuning in future systems.

Wind-tunnel tests with this control design were performed using a one-degree-of-freedom test apparatus for an L-59 model aircraft at the NASA Langley Research Center 12-ft wind-tunnel. The results showed the usefulness and capability of the proposed controller. The controller operated based on general first principles-of-flight, tracked an input command within an RMS error less than 5 deg, converged in under 8.5 sec, and achieved the notion of stability described in this work.

The proposed controller will be extended to multiple degree-of-freedom aircraft experiments, and additional adaptivity systems will be added. While the specific purpose of this experiment was to create a controller using abstract concepts of aircraft dynamics, future work is necessary to understand mathematically the range of aircraft classes for which this approach is applicable and what performance may be achieved.

REFERENCES

[1] D. J. Diston, *Computational Modelling and Simulation of Aircraft and the Environment Platform Kinematics and Synthetic Environment*, pp. 1–24. Hoboken, NJ, USA: Wiley, 2009.

[2] C. B. de Mendonça, E. T. da Silva, M. Curvo, and L. G. Trabasso, "Model-based flight testing," *J. Aircraft*, vol. 50, pp. 176–186, 2013.

[3] S. Nagai and H. Iijima, "Uncertainty identification of supersonic wind tunnel testing," *J. Aircraft*, vol. 48, Mar. 2011.

[4] A. Abbas-Bayoumi and K. Becker, "An industrial view on numerical simulation for aircraft aerodynamic design," *J. Math. Ind.*, vol. 1, Dec. 2011.

[5] S. L. Kukreja, "Data-driven model development for the supersonic semispan transport," *AIAA J.*, vol. 51, pp. 1333–1341, Jun. 2013.

[6] A. Mekky and O. R. González, "LQ control for the NASA learn-to-fly free-to-roll project," in *2016 IEEE Nat. Aerospace and Electron. Conf. (NAECON) and Ohio Innovation Summit (OIS)*, pp. 173–178, Jul. 2016.

[7] E. Morelli, "Real-time global nonlinear aerodynamic modeling for learn-to-fly," in *AIAA Atmospheric Flight Mechanics Conf., AIAA SciTech Forum*, Jan. 2016.

[8] D. Choe and J.-H. Kim, "Pitch autopilot design using model-following adaptive sliding mode control," *J. Guidance, Control, and Dynamics*, vol. 25, pp. 826–829, Jul. 2002.

[9] I. Rusnak, A. Guez, I. Bar-Kana, and M. Steinberg, "Online identification and control of linearized aircraft dynamics," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 7, pp. 56–60, Jul. 1992.

[10] H.-J. Rong, S. Han, and G.-S. Zhao, "Adaptive fuzzy control of aircraft wing-rock motion," *Appl. Soft Computing J.*, vol. 14, pp. 181–193, Jan. 2014.

[11] K. Lu and Y. Xia, "Adaptive attitude tracking control for rigid spacecraft with finite-time convergence," *Automatica*, vol. 49, pp. 3591–3599, Dec. 2013.

[12] K. M. Passino and S. Yurkovich, *Fuzzy Control.* Boston, MA, USA: Addison-Wesley Longman, Inc., 1997.

[13] "Fuzzy logic toolbox, user's guide," 2017a. The MathWorks, Natick, MA, USA.

[14] V. Klein and E. Morelli, *Aircraft System Identification: Theory and Practice.* AIAA education series, Reston, VA: Amer. Inst. of Aeronautics and Astronautics, 2006.

[15] B. Friedland, *Control System Design: An Introduction to State-Space Methods.* Dover Publications, 2012.

[16] C. Rohrs, J. Melsa, and D. Schultz, *Linear Control System.* McGraw-Hill, 1993.

[17] L. Nguyen, M. Ogburn, W. Gilbert, K. S. Kibler, P. W. Brown, and P. L. Deal, "Simulator study of stall/post-stall characteristics of a fighter airplane with relaxed longitudinal static stability," Tech. Paper 1538, NASA, Dec. 1979.

[18] N. Beygi, M. Beigy, and M. Siahi, "Design of fuzzy self-tuning PID controller for pitch control system of aircraft autopilot," *CoRR*, vol. abs/1510.02588, Oct. 2015.

[19] Y. Ma, Y. Liu, and C. Wang, "Design of parameters self-tuning fuzzy PID control for dc motor," in *2010 The 2nd Int. Conf. Ind. Mechatronics and Automation*, vol. 2, pp. 345–348, May 2010.

[20] E. Morelli, "Flight test maneuvers for efficient aerodynamic modeling," *J. Aircraft*, vol. 49, pp. 1857–1867, Nov. 2012.

[21] Y. Huo, "Model of F-16 fighter aircraft." University of Southern California, 2018.

[22] US Digital, *MA3 Miniature Absolute Magnetic Shaft Encoder*, Aug. 2016.

[23] Krohn-Hite Corporation, *0.1Hz to 200kHz Four Channel 4–Pole Filter*, Feb. 2016.

[24] Arduino, *Arduino DUE*, 2018.

[25] Seeed Technology, *W5200 Ethernet Shield*, 2017.

[26] SparkFun Electronics, *SparkFun Ludus Protoshield Wireless*, 2017.

[27] Hobbico, *Futaba S9650 Digital Mini Servo.* Hobbico, Jun. 2003.

APPENDIX A

MATLAB CODE

A.1 SIMULATION DRIVER

The following code is the primary linear model simulation driver. It requires the relative path packages and is capable of running all linear simulations given appropriate parameters.

```
1 % Primary Simulation Driver
2 clear; clear;
3 clc;
4 profile on;
5
6 addpath('.\Plants');
7 addpath('.\Tools');
8 addpath('.\Functions');
9 addpath('..\..\Mekky\F2P_F16');
10 addpath('..\..\Mekky\F_16_control');
11
12 warning('off','Fuzzy:evalfis:InputOutOfRange');
13
14 %% Simulation Information
15 STEPTIME = 0.02;
16 STOPTIME = 20;
17
18 AMPLITUDE = 1;
19 x_cg = .3;
20
21 X_cg = [0.3];
22 Amplitude = [1];
23 PERIOD    = 2;
24 %% Create Input Commands
25 InputCommand;
26
27 U = zeros(numel(X_cg), numel(alphaCommand));
28
29 for plantID = 1:3
30     for ctrlID = 1:2
31         %% Call Plant
32         if plantID == 1
33             PlantF16_1;
34         elseif plantID == 2
35             PlantF16_2;
36         elseif plantID == 3
37             Plant747;
```

```
38          end
39
40          %% Call Controller
41          if ctrlID == 1
42              ctrlname = 'PID';
43          elseif ctrlID == 2
44              ctrlname = 'FLC';
45          end
46
47          %% Set Adaptivity
48          disableAdaptivity = 1;
49          presets = 0;
50
51          if disableAdaptivity == 1
52              adaptName = 'NO_Adapt';
53          else
54              adaptName = 'Adapt';
55          end
56          %% Generate PID Systems
57          if strcmp(name, 'F16-1')
58              Kp = .5;
59              Ki = .05;
60              Kd = 0;
61              alphaPID = pid(-Kp, -Ki, -Kd, .02,.02);
62          elseif strcmp(name, 'F16-2')
63              Kp = .5;
64              Ki = .05;
65              Kd = 0;
66              alphaPID = pid(-Kp, -Ki, -Kd, .02,.02);
67          elseif strcmp(name, 'B747')
68              Kp = .5;
69              Ki = .05;
70              Kd = 0;
71              alphaPID = pid(-Kp, -Ki, -Kd, .02,.02);
72          end
73
74          sysPID = ss(alphaPID);
75
76          Ysave = zeros(numel(X_cg), numel(alphaCommand));
77          %%
78          for xIndex = 1:numel(X_cg)
79              for aIndex = 1:numel(Amplitude)
80                  x_cg = X_cg(xIndex);
81                  AMPLITUDE = Amplitude(aIndex);
82
83                  %% Call Controller
84                  INITIALIZATION;
85
86                  %% Create Simulation Variables
87                  X    = zeros(size(sys.a, 2), numel(alphaCommand)+1);
88                  Xref = zeros(size(sysRef.a, 2), numel(alphaCommand)+1);
89
```

```
90              Y    = zeros(size(sys.c, 1), numel(alphaCommand));
91              Yref = zeros(size(sysRef.c, 1), numel(alphaCommand));
92
93              XPID = zeros(size(sysPID.a, 2), numel(alphaCommand)+1);
94              YPID = zeros(size(sysPID.c, 2), numel(alphaCommand));
95
96              Position_Output         = zeros(1, numel(alphaCommand));
97              Velocity_Output         = zeros(1, numel(alphaCommand));
98              Accel_Output            = zeros(1, numel(alphaCommand));
99              ErrorCorrection_Output = zeros(1, numel(alphaCommand));
100
101             alphaError              = zeros(1, numel(alphaCommand));
102             dAlphaError             = zeros(1, numel(alphaCommand));
103             d2Alpha                 = zeros(1, numel(alphaCommand));
104
105             FFTSize = 2^nextpow2((1/STEPTIME)/2)+1;
106             AlphaFFT_Output = zeros(numel(alphaCommand), FFTSize);
107             ElevFFT_Output  = zeros(numel(alphaCommand), FFTSize);
108             FFT_FFT_Output  = zeros(numel(alphaCommand), FFTSize);
109
110             check = 0;
111
112             threshold = exp(-1/16.*[0:32]);
113             ARRPrev = 0;
114             %% Start Simulation Clock
115             tic;
116
117             %% Primary Simulation Loop
118             for index = 1:numel(alphaCommand)
119                 if floor((index/numel(alphaCommand))*100) > check
120                     fprintf('.');
121                     check = check + 10;
122                 end
123
124                 % Calculate Error
125                 try
126                     alphaError(index) = ...
127                         Yref(1, index-1) - X(alphaLoc, index);
128                 catch
129                     alphaError(index) = ...
130                         Yref(1, index)   - X(alphaLoc, index);
131                 end
132
133                 try
134                     dAlphaError(index) = ...
135                         Yref(2, index-1) - X(alphaDotLoc, index);
136                 catch
137                     dAlphaError(index) = ...
138                         Yref(2, index)   - X(alphaDotLoc, index);
139                 end
140
141                 if strcmp(ctrlname, 'FLC')
```

```
142                 MainController.setSampleTime(STEPTIME);
143                 MainSupervisor.super(...
144                     MainController, ...
145                     X(alphaLoc, index), ...
146                     X(alphaDotLoc, index), ...
147                     alphaCommand(index), ...
148                     STEPTIME, ...
149                     0);
150
151                 U(xIndex, index) = MainController.control(...
152                     MainSupervisor, ...
153                     X(alphaLoc, index), ...
154                     X(alphaDotLoc, index), ...
155                     V);
156             elseif strcmp(ctrlname, 'PID')
157                 XPID(index+1)   = sysPID.a * XPID(index) ...
158                     + sysPID.b ...
159                     * (Xref(alphaLoc,index)-X(alphaLoc,index));
160                 YPID(index)     = sysPID.c * XPID(index) ...
161                     + sysPID.d ...
162                     * (Xref(alphaLoc,index)-X(alphaLoc,index));
163                 U(index) =  YPID(index);
164             end
165
166             % Limiting Elevator Rate
167             try
168                 min(max((U(index) - U(index)-1) ...
169                     / STEPTIME, -RATELIMIT), RATELIMIT);
170             catch
171             end
172
173             % Limiting Elevator Position
174             U(index) = min(...
175                 max(...
176                 U(index), -POSITIONLIMIT*180/pi), ...
177                 POSITIONLIMIT*180/pi);
178
179             % Calculate Reference Model
180             Xref(:, index+1) = sysRef.a * Xref(:, index) ...
181                 + sysRef.b * alphaCommand(index);
182             Yref(:, index)   = sysRef.c * Xref(:, index) ...
183                 + sysRef.d * alphaCommand(index);
184
185             % Simulate Plant
186             if strcmp(name, 'F16-1')
187                 U(index) = U(index)/(pi);
188                 X(:, index+1) = ...
189                     sys.a * X(:, index) + sys.b * U(index);
190                 Y(:, index)   = ...
191                     sys.c * X(:, index) + sys.d * U(index);
192
193                 if strcmp(ctrlname, 'PID')
```

```
194                             Y(:, index) = Y(:, index)/pi;
195                         end
196
197                     % Simulate Plant
198                     elseif strcmp(name, 'F16-2')
199                         U(index) = U(index)*180/pi;
200                         X(:, index+1) = ...
201                             sys.a * X(:, index) + sys.b * U(index);
202                         Y(:, index)   = ...
203                             sys.c * X(:, index) + sys.d * U(index);
204
205                         if strcmp(ctrlname, 'PID')
206                             U(index) = U(index)*pi/180;
207                             Y(:, index) = Y(:, index)./pi;
208                         end
209
210                     elseif strcmp(name, 'B747')
211                         U(index) = U(index)*180/pi;
212                         X(:, index+1) = sys.a * X(:, index) + sys.b * U(index);
213                         Y(:, index)   = sys.c * X(:, index) + sys.d * U(index);
214
215                         if strcmp(ctrlname, 'PID')
216                             U(index) = U(index)*pi/180;
217                             Y(:, index) = Y(:, index)./pi;
218                         end
219                     end
220                 end
221                 fprintf('\n');
222
223                 %% Stop Simulation Clock
224                 toc;
225
226             end
227             Ysave(xIndex,:)=Y(alphaLoc,:);
228         end
229
230         saveFile = sprintf('%s_%s_%s', name, ctrlname, adaptName);
231         saveFile = strrep(saveFile, '-', '_');
232         eval(['Ysave_' saveFile '=Ysave;']);
233         eval(['U_' saveFile '=U;']);
234         save(saveFile, sprintf('Ysave_%s', saveFile), ...
235             sprintf('U_%s', saveFile), 'Yref');
236
237     end
238 end
239 %% Plot Simulation
240 SimPlots;
```

A.2 INITIALIZATION

The following code is the primary constructor for all controller and supervisor components. Settable options include specifications for gain pre-initialization and activating adaptive algorithms.

```
1  warning('off','Fuzzy:evalfis:InputOutOfRange');
2
3  % Initialize Controller Variables
4  CTRL_errorGain = 1;
5  CTRL_positGain = 1;
6  CTRL_velocGain = 1;
7  CTRL_accelGain = 1;
8
9  alphaLimit = 25;
10 elevLimit = 40;     % Sets +- elevator limit
11 lowerOffset = 0;    % limit = -elevLimit + lowerOffset
12 upperOffset = -10; % limit = elev + upperOffset
13
14 if disableAdaptivity == 1
15     % set nonzero to disable autogain adjustment changing
16     disableGainSense = 1;
17
18     % this means individual I/O gains must be established
19     % for all FLCs below
20     % (0 = Enable | 1 = Disable)
21     %
22     % NOTE: This does not disable the original supervisor
23     % gain adjustment techniques that control the
24     % controller output gains. This will (en/dis)able
25     % the automatic detection of FLC I/O gains based on
26     % evalutation within 10% of an FLC I/O boundary or
27     % for out of bounds evaluation
28 else
29     disableGainSense = 0;
30 end
31
32 windtunnel = 0; % set nonzero for windtunnel usage
33 % (0 = Simulation | 1 = Windtunnel Usage)
34
35 errorLimit = 2*alphaLimit;
36 errorChangeLimit = 4*errorLimit;
37
38 zeta = 0.85;
39 wn = 2.5;
40 referenceFunction = [zeta, wn];
41 dt = 0.02;
42
43 ssAnalyzeTime = 1;  % sec -> Window length for detecting Steady-State
44 ssSensitivity = .1; % deg -> Sensitivity threshold below which the system is
45 % considered in steady state if the sample amplitude spread
```

```
46 % is maintained for the ssAnalyeTime window
47
48 % FIS Initalizations
49 FISStruct = struct( ...
50     'distribution', {}, ...
51     'overlap', {}, ...
52     'inputRange1', {}, ...
53     'inputRange2', {}, ...
54     'outputRange', {}, ...
55     'MFs', {}, ...
56     'inputType', {}, ...
57     'outputType',  {}, ...
58     'ENABLE', {}...
59     );
60 CTRL_A_E   = FISStruct;
61 CTRL_AI_EI = FISStruct;
62 CTRL_AD_ED = FISStruct;
63 SUPER_Kp   = FISStruct;
64 SUPER_Ki   = FISStruct;
65 SUPER_Kd   = FISStruct;
66
67 % CAREFUL: no error checking is performed on FIS parameters. Heed option
68 % comments on CTRL_A_E
69 CTRL_A_E(1).distribution = 'linear'; % linear, square, cube, quad, pent
70
71 % 1 -> 200 (100 means 100 percent overlap with adjacent leg
72 CTRL_A_E(1).overlap = 100;
73 if disableGainSense == 0
74     CTRL_A_E(1).inputRange1 = 1;            % Only positive numbers
75     CTRL_A_E(1).inputRange2 = 1;            % Only positive numbers
76
77     CTRL_A_E(1).outputRange = 1;            % Only positive numbers
78 else
79     CTRL_A_E(1).inputRange1 = alphaLimit;   % Only positive numbers
80     CTRL_A_E(1).inputRange2 = alphaLimit;   % Only positive numbers
81
82     if strcmp(name, 'F16-1')
83         CTRL_A_E(1).outputRange = 1;        % Only positive numbers
84     elseif strcmp(name, 'F16-2')
85         CTRL_A_E(1).outputRange = 1;        % Only positive numbers
86     elseif strcmp(name, 'B747')
87         CTRL_A_E(1).outputRange = 2;        % Only positive numbers
88     end
89 end
90
91 CTRL_A_E(1).MFs = 7;                    % DO NOT CHANGE
92 CTRL_A_E(1).inputType = 'trimf';        % trimf, constant
93 CTRL_A_E(1).outputType = 'constant'; %trimf, constant
94 CTRL_A_E(1).ENABLE = 1;                 % 1->ON | 0->OFF
95
96 %%
97 CTRL_AI_EI(1).distribution = 'linear';
```

```
98  CTRL_AI_EI(1).overlap = 100;
99  if disableGainSense == 0
100     CTRL_AI_EI(1).inputRange1 = 1;
101     CTRL_AI_EI(1).inputRange2 = 1;
102 
103     CTRL_AI_EI(1).outputRange = 1;
104 else
105     CTRL_AI_EI(1).inputRange1 = alphaLimit;
106     CTRL_AI_EI(1).inputRange2 = alphaLimit;
107 
108     if strcmp(name, 'F16-1')
109         CTRL_AI_EI(1).outputRange = 1;
110     elseif strcmp(name, 'F16-2')
111         CTRL_AI_EI(1).outputRange = 1;
112     elseif strcmp(name, 'B747')
113         CTRL_AI_EI(1).outputRange = 1.5;
114     end
115 end
116 CTRL_AI_EI(1).MFs = 7;              % DO NOT CHANGE
117 CTRL_AI_EI(1).inputType = 'trimf';
118 CTRL_AI_EI(1).outputType = 'constant';
119 CTRL_AI_EI(1).ENABLE = 1;              % 1->ON | 0->OFF
120 
121 %%
122 CTRL_AD_ED(1).distribution = 'linear';
123 CTRL_AD_ED(1).overlap = 100;
124 if disableGainSense == 0
125     CTRL_AD_ED(1).inputRange1 = 1;
126     CTRL_AD_ED(1).inputRange2 = 1;
127 
128     CTRL_AD_ED(1).outputRange = 1;
129 else
130     CTRL_AD_ED(1).inputRange1 = alphaLimit;
131     CTRL_AD_ED(1).inputRange2 = alphaLimit;
132 
133     if strcmp(name, 'F16-1')
134         CTRL_AD_ED(1).outputRange = 1;
135     elseif strcmp(name, 'F16-2')
136         CTRL_AD_ED(1).outputRange = 1;
137     elseif strcmp(name, 'B747')
138         CTRL_AD_ED(1).outputRange = 2;
139     end
140 end
141 
142 CTRL_AD_ED(1).MFs = 7;              % DO NOT CHANGE
143 CTRL_AD_ED(1).inputType = 'trimf';
144 CTRL_AD_ED(1).outputType = 'constant';
145 CTRL_AD_ED(1).ENABLE = 1;              % 1->ON | 0->OFF
146 
147 %%
148 SUPER_Kp(1).distribution = 'linear';
149 SUPER_Kp(1).overlap = 100;
```

```
150 if disableGainSense == 0
151     SUPER_Kp(1).inputRange1 = 1;
152     SUPER_Kp(1).inputRange2 = 1;
153     SUPER_Kp(1).outputRange = 1;
154 else
155     SUPER_Kp(1).inputRange1 = errorLimit;
156     SUPER_Kp(1).inputRange2 = errorChangeLimit;
157
158     if strcmp(name, 'F16-1')
159         if presets == 0
160             SUPER_Kp(1).outputRange = elevLimit;
161         else
162             SUPER_Kp(1).outputRange = elevLimit;
163         end
164
165     elseif strcmp(name, 'F16-2')
166         if presets == 0
167             SUPER_Kp(1).outputRange = elevLimit*pi/180;
168         else
169             SUPER_Kp(1).outputRange = 2;
170         end
171
172     elseif strcmp(name, 'B747')
173         if presets == 0
174             SUPER_Kp(1).outputRange = elevLimit*pi/180;
175         else
176             SUPER_Kp(1).outputRange = 3.7713;
177         end
178     end
179 end
180 SUPER_Kp(1).MFs = 7;                % DO NOT CHANGE
181 SUPER_Kp(1).inputType = 'trimf';
182 SUPER_Kp(1).outputType = 'constant';
183 SUPER_Kp(1).ENABLE = 1;               % 1->ON | 0->OFF
184
185 %%
186 SUPER_Ki(1).distribution = 'linear';
187 SUPER_Ki(1).overlap = 100;
188 if disableGainSense == 0
189     SUPER_Ki(1).inputRange1 = 1;
190     SUPER_Ki(1).inputRange2 = 1;
191     SUPER_Ki(1).outputRange = 1;
192 else
193     SUPER_Ki(1).inputRange1 = errorLimit;
194     SUPER_Ki(1).inputRange2 = errorChangeLimit;
195
196     if strcmp(name, 'F16-1')
197         if presets == 0
198             SUPER_Ki(1).outputRange = elevLimit/2^2;
199         else
200             SUPER_Ki(1).outputRange = elevLimit/2^2;
201         end
```

```
202
203     elseif strcmp(name, 'F16-2')
204         if presets == 0
205             SUPER_Ki(1).outputRange = elevLimit/2^2*pi/180;
206         else
207             SUPER_Ki(1).outputRange = 1.5;
208         end
209
210     elseif strcmp(name, 'B747')
211         if presets == 0
212             SUPER_Ki(1).outputRange = elevLimit/2^2*pi/180;
213         else
214             SUPER_Ki(1).outputRange = 1.5;
215         end
216     end
217
218 end
219 SUPER_Ki(1).MFs = 7;                  % DO NOT CHANGE
220 SUPER_Ki(1).inputType = 'trimf';
221 SUPER_Ki(1).outputType = 'constant';
222 SUPER_Ki(1).ENABLE = 1;                 % 1->ON | 0->OFF
223
224 %%
225 SUPER_Kd(1).distribution = 'linear';
226 SUPER_Kd(1).overlap = 100;
227 if disableGainSense == 0
228     SUPER_Kd(1).inputRange1 = 1;
229     SUPER_Kd(1).inputRange2 = 1;
230     SUPER_Kd(1).outputRange = 1;
231 else
232     SUPER_Kd(1).inputRange1 = errorLimit;
233     SUPER_Kd(1).inputRange2 = errorChangeLimit;
234     if strcmp(name, 'F16-1')
235         if presets == 0
236             SUPER_Kd(1).outputRange = elevLimit*2^4;
237         else
238             SUPER_Kd(1).outputRange = elevLimit*2^4;
239         end
240
241     elseif strcmp(name, 'F16-2')
242         if presets == 0
243             SUPER_Kd(1).outputRange = elevLimit*2^2*pi/180;
244         else
245             SUPER_Kd(1).outputRange = 6.7048;
246         end
247
248     elseif strcmp(name, 'B747')
249         if presets == 0
250             SUPER_Kd(1).outputRange = elevLimit*2^2*pi/180;
251         else
252             SUPER_Kd(1).outputRange = 273.3817;
253         end
```

```
254     end
255 end
256 SUPER_Kd(1).MFs = 7;                    % DO NOT CHANGE
257 SUPER_Kd(1).inputType = 'trimf';
258 SUPER_Kd(1).outputType = 'constant';
259 SUPER_Kd(1).ENABLE = 1;                 % 1->ON | 0->OFF
260 
261 %%
262 ctrlFIS = [CTRL_A_E, CTRL_AI_EI, CTRL_AD_ED];
263 superFIS = [SUPER_Kp, SUPER_Ki, SUPER_Kd];
264 
265 MainSupervisor = Supervisor( ...
266     1, .05, ...
267     ssAnalyzeTime, ssSensitivity, ...
268     dt, ...
269     alphaLimit, ...
270     elevLimit, ...
271     referenceFunction, ...
272     superFIS, ...
273     disableGainSense, ...
274     windtunnel);
275 
276 MainController = Controller(...
277     CTRL_errorGain, ...
278     CTRL_positGain, ...
279     CTRL_velocGain, ...
280     CTRL_accelGain, ...
281     alphaLimit, ...
282     elevLimit, ...
283     lowerOffset, ...
284     upperOffset, ...
285     ctrlFIS, ...
286     disableGainSense, ...
287     windtunnel);
288 MainDataLog = DataLog();
```

A.3 LINEAR SIMULATION MODELS

A.3.1 F16 LINEAR MODEL 1

The following code initializes the continuous-time and discrete-time F16 Linear Model 1. [21]

```
%% Linear F16-1 Model
name = 'F16-1';

h = 25000;
V = 600;
rho = (2.377e-3)*(1-(.703e-5)*h)^4.14;

POSITIONLIMIT = 25;                % (deg)
RATELIMIT     = POSITIONLIMIT*4; % (deg/s)

% Convert to Radians
POSITIONLIMIT = POSITIONLIMIT * (pi/180); % (rad)
RATELIMIT     = RATELIMIT     * (pi/180); % (rad/s)

% Incorporation of additional models for simulation
Iy   = 55814;   % slug-ft
cBar = 11.32;   % ft
S    = 300;     % ft^2
qBar = (1/2)*rho*V^2;

% Set Initial Flight CM Input Parameters
alphaInitial =  0; % (deg)
elevInitial  =  0; % (deg)

% Set SS Location for Alpha/AlphaDot
alphaLoc    = 1;
alphaDotLoc = 2;

% F16 State-space Realization -> E. Morelli's System Identification Book
syms Cma Cmq Cme;
sysA = [...
    0                       1;...
    (cBar*S*qBar)/Iy*Cma (cBar*S*qBar)/Iy*Cmq];
sysB = [...
    0,...
    (cBar*S*qBar)/Iy*Cme].';
sysC = [...
    1  0;... % Alpha    (rad)
    0  1];   % AlphaDot (rad/s)
sysD = [0  0]';

%% Actuator Model
sysAct = tf([-3.028  130.8], [1  10.26  132.5]);
```

```
44
45 %% Recalculate Plant with Current Pitching Moment Coefficients
46 % I/O's for F16_Aero_Lin are (deg)
47 [Cma, Cme, Cmq] = F16_Aero_Lin(V, alphaInitial, 0, 0, 0, 0, ...
48     elevInitial, 0, 0, 0, 0, x_cg );
49
50 % Convert pitching moments to rad
51 Cma = Cma * 180/pi;
52 Cme = Cme * 180/pi;
53 Cmq = Cmq * 180/pi;
54
55 sysCT = ss(eval(sysA), eval(sysB), sysC, sysD);
56 sysCT.OutputName={'Alpha', 'AlphaDot'};
57 sysCT.InputName = {'Command'};
58 sysCT.StateName={'Alpha', 'AlphaDot'};
59
60 sys = c2d(sysCT, STEPTIME);
```

A.3.2 F16 LINEAR MODEL 2

The following code initializes the continuous-time and discrete-time F16 Linear Model 2. [15, p. 128]

```
1 %% Linear F16-2 Model
2 name = 'F16-2';
3 V=203.867;
4
5 sysA = [...
6     -0.0507 -3.861 0 -32.2;...
7     -0.00117 -.5164 1 0;...
8     -.000129 1.4168 -.4932 0;...
9     0 0 1 0];
10 sysB = [0 -.0717 -1.645 0]';
11 sysC = [...
12     1 0 0 0;...
13     0 1 0 0;...
14     0 0 1 0;...
15     0 0 0 1];
16 sysD = 0;
17
18 POSITIONLIMIT = 25;
19 RATELIMIT     = POSITIONLIMIT*16;
20
21 % Set Initial Flight CM Input Parameters
22 alphaInitial = 0;
23 elevInitial  = 0;
24
25 % Set SS Location for Alpha/AlphaDot
26 alphaLoc    = 2;
27 alphaDotLoc = 3;
28
```

```
29 %% Establish State-Space Model
30 sysCT = (ss(sysA, sysB, sysC, sysD));
31
32 %% Actuator Model
33 sysAct = tf([-3.028 130.8], [1 10.26 132.5]);
34
35 sys = c2d(sysCT, STEPTIME);
```

A.3.3 BOEING 747 LINEAR MODEL

The following code initializes the continuous-time and discrete-time Boeing 747 Linear Model. [16, p. 92]

```
1 %% Linear 747 Model
2 name = 'B747';
3 V=278.667;
4
5 POSITIONLIMIT = 15;
6 RATELIMIT     = POSITIONLIMIT*4;
7
8 %Incorporation of additional models for simulation
9 sysA = [...
10     -.0188   11.5959   0.0    -32.2;...
11     -.0007    -.5357   1.0      0.0;...
12     .000048   -.4944   -.4935  0.0;...
13     0          0.0      1.0      0.0];
14 sysB = [0 0 -.5632 0]';
15 sysC = [...
16     1 0 0 0;...
17     0 1 0 0;...
18     0 0 1 0;...
19     0 0 0 1];
20 sysD = 0;
21
22 % Set Initial Flight CM Input Parameters
23 alphaInitial = 0;
24 elevInitial  = 0;
25
26 % Set SS Location for Alpha/AlphaDot
27 alphaLoc    = 2;
28 alphaDotLoc = 3;
29
30 %% Establish State-Space Model
31 sysCT = ss(sysA, sysB, sysC, sysD);
32 sys = c2d(sysCT, STEPTIME);
```

A.4 CUSTOM QUEUE CLASS

The following code is a simple queue class useful for maintaining running statistics. The queue size is set upon instantiation. The queue is implemented with First-In-First-Out (FIFO) methodology where the lowest queue index represents the oldest value.

```
1  classdef CustomQueue < handle
2      properties
3          data
4          maximum
5          minimum
6          spread
7          size
8          last
9          previous
10         dc
11         average
12         change
13         windowMean
14     end
15
16     methods
17         function obj = CustomQueue(size)
18             obj.data = zeros(1, size);
19             obj.maximum = 0;
20             obj.minimum = 0;
21             obj.spread = 0;
22             obj.size = size;
23             obj.last = 0;
24             obj.previous = 0;
25             obj.dc = 0;
26             obj.average = 0;
27             obj.change = 0;
28             obj.windowMean = 0;
29         end
30
31         function Reset(obj)
32             obj.data = zeros(1, obj.size);
33             obj.maximum = 0;
34             obj.minimum = 0;
35             obj.spread = 0;
36             obj.last = 0;
37             obj.previous = 0;
38             obj.dc = 0;
39             obj.average = 0;
40             obj.change = 0;
41             obj.windowMean = 0;
42         end
43
44         function Push(obj, data)
45             obj.data = [obj.data(2:end) data];
```

```
46              obj.maximum = max(obj.data);
47              obj.minimum = min(obj.data);
48              obj.spread = obj.maximum - obj.minimum;
49              obj.previous = obj.last;
50              obj.last = data;
51              obj.dc = obj.spread / obj.size;
52              obj.average = mean(obj.data);
53              obj.change = obj.last - obj.previous;
54              obj.windowMean = ...
55                  mean(obj.data(end, end-min(length(obj.data)-1, 20)));
56          end
57      end
58  end
```

A.5 DATA LOGGING CLASS

The following code is a container class useful for storing run-time data. It was designed to maintain a snapshot of the current system state for each sample period.

```
classdef DataLog < handle
    properties
        alpha
        alphaDot
        alphaDotDot
        dE

        alphaRef
        alphaDotRef
        alphaDotDotRef

        alphaError
        alphaDotError

        Kp
        Ki
        Kd

        POutput
        DOutput
        IOutput

        PGain
        DGain
        IGain

        ctrlOutput

        errorIntegration
        errorDotIntegration
        errorDotDotIntegration
        errorSum

        errorDifferentiation
        errorDotDifferentiation
        errorChange

        time

        alphaCommand

        KpOutputGain
        KiOutputGain
        KdOutputGain
        AEOutputGain
        AI_EIOutputGain
        AD_EDOutputGain
```

```
48    end
49
50    methods
51        function obj = DataLog()
52            obj.alpha = 0;
53            obj.alphaDot = 0;
54            obj.alphaDotDot = 0;
55            obj.dE = [0,0];
56
57            obj.alphaRef = 0;
58            obj.alphaDotRef = 0;
59            obj.alphaDotDotRef = 0;
60
61            obj.alphaError = 0;
62            obj.alphaDotError = 0;
63
64            obj.Kp = 0;
65            obj.Ki = 0;
66            obj.Kd = 0;
67
68            obj.POutput = 0;
69            obj.DOutput = 0;
70            obj.IOutput = 0;
71
72            obj.PGain = 0;
73            obj.DGain = 0;
74            obj.IGain = 0;
75
76            obj.ctrlOutput = 0;
77
78            obj.errorIntegration = 0;
79            obj.errorDotIntegration = 0;
80            obj.errorDotDotIntegration = 0;
81            obj.errorSum = 0;
82
83            obj.errorDifferentiation = 0;
84            obj.errorDotDifferentiation = 0;
85            obj.errorChange = 0;
86
87            obj.time = 0;
88
89            obj.alphaCommand = 0;
90
91            obj.KpOutputGain = 0;
92            obj.KiOutputGain = 0;
93            obj.KdOutputGain = 0;
94            obj.AEOutputGain = 0;
95            obj.AI_EIOutputGain = 0;
96            obj.AD_EDOutputGain = 0;
97        end
98
99        function LogData(obj, SUPERVISOR, CONTROLLER, time, dE)
```

```
100             obj.alpha(end + 1)         = SUPERVISOR.alphaHistory.last;
101             obj.alphaDot(end + 1)      = SUPERVISOR.alphaDotHistory.last;
102             obj.alphaDotDot(end + 1) = SUPERVISOR.alphaDotDotHistory.last;
103
104             obj.dE(end + 1,:) = dE;
105
106             obj.alphaRef(end + 1)         = SUPERVISOR.sysRefDTY(1, end);
107             obj.alphaDotRef(end + 1)      = SUPERVISOR.sysRefDTY(2, end);
108             obj.alphaDotDotRef(end + 1) = SUPERVISOR.sysRefDTY(3, end);
109
110             obj.alphaError(end + 1)       = SUPERVISOR.alphaError.last;
111             obj.alphaDotError(end + 1)   = SUPERVISOR.alphaDotError.last;
112
113             obj.Kp(end + 1) = CONTROLLER.Kp.last;
114             obj.Ki(end + 1) = CONTROLLER.Ki.last;
115             obj.Kd(end + 1) = abs(CONTROLLER.Kd.last);
116
117             obj.POutput(end + 1) = CONTROLLER.POutput.last;
118             obj.IOutput(end + 1) = CONTROLLER.IOutput.last;
119             obj.DOutput(end + 1) = CONTROLLER.DOutput.last;
120
121             obj.PGain(end + 1) = CONTROLLER.Kp.last;
122             obj.IGain(end + 1) = CONTROLLER.Ki.last;
123             obj.DGain(end + 1) = CONTROLLER.Kd.last;
124
125             obj.ctrlOutput(end + 1) = CONTROLLER.ctrlOutput.last;
126
127             obj.errorIntegration(end + 1)      = ...
128                 SUPERVISOR.errorIntegrator.last;
129             obj.errorDotIntegration(end + 1) = ...
130                 SUPERVISOR.errorDotIntegrator.last;
131             obj.errorDotDotIntegration(end + 1) = ...
132                 SUPERVISOR.errorDotDotIntegrator.last;
133             obj.errorSum(end + 1) = SUPERVISOR.errorSum.last;
134
135             obj.errorDifferentiation(end + 1) = ...
136                 SUPERVISOR.errorDifferentiator.last;
137             obj.errorDotDifferentiation(end + 1) = ...
138                 SUPERVISOR.errorDotDifferentiator.last;
139             obj.errorChange(end + 1) = SUPERVISOR.errorChange.last;
140
141             obj.time(end + 1) = time;
142
143             obj.alphaCommand(end + 1) = ...
144                 SUPERVISOR.alphaCommandHistory.last;
145
146             obj.KpOutputGain(end + 1) = SUPERVISOR.Kp.outputGain.last;
147             obj.KiOutputGain(end + 1) = SUPERVISOR.Ki.outputGain.last;
148             obj.KdOutputGain(end + 1) = SUPERVISOR.Kd.outputGain.last;
149             obj.AEOutputGain(end + 1) = CONTROLLER.A_E.outputGain.last;
150             obj.AI_EIOutputGain(end + 1) = ...
151                 CONTROLLER.AI_EI.outputGain.last;
```

```
152                 obj.AD_EDOutputGain(end + 1) = ...
153                     CONTROLLER.AD_ED.outputGain.last;
154             end
155         end
156 end
```

A.6 FUZZY LOGIC CONTAINER CODE

A.6.1 FUZZY LOGIC CONTAINER CLASS

The following class code defines a wrapper for a two-input one-output fuzzy logic system.

```
1  classdef FIS_2X1 < handle
2      properties
3          % I/O Gains
4          input1Gain
5          input2Gain
6          outputGain
7
8          % FLC Parameters
9          input1Name
10         input2Name
11         outputName
12
13         input1MFNum
14         input2MFNum
15         outputMFNum
16
17         input1MFType
18         input2MFType
19         outputMFType
20
21         input1Overlap
22         input2Overlap
23         outputOverlap
24
25         % Fuzzy Logic Controller
26         FLC
27         LUT
28
29         absRange
30         absRangeSize
31
32         NAME
33         ENABLE
34         IRR
35
36         boundaryCount1
37         boundaryCount2
38         disableGainSense
39
40         alphas
41         gains
42     end
43
44     methods
45         % Constructor
```

```
46        function obj = FIS_2X1(...
47                nameIn1, gainIn1, mfNumIn1, mfTypeIn1, overlapIn1,...
48                nameIn2, gainIn2, mfNumIn2, mfTypeIn2, overlapIn2,...
49                nameOut, gainOut, mfOutNum, mfTypeOut, overlapOut,...
50                orientation, distribution, NAME, ENABLE, ...
51                disableGainSense, varargin)
52            obj.disableGainSense=disableGainSense;
53            obj.input1Name = nameIn1;
54            obj.input2Name = nameIn2;
55            obj.outputName = nameOut;
56
57            obj.input1Gain = gainIn1;
58            obj.input2Gain = gainIn2;
59            obj.outputGain = CustomQueue(100);
60            obj.outputGain.Push(gainOut);
61
62            obj.input1MFNum = mfNumIn1;
63            obj.input2MFNum = mfNumIn2;
64            obj.outputMFNum = mfOutNum;
65
66            obj.input1MFType = mfTypeIn1;
67            obj.input2MFType = mfTypeIn2;
68            obj.outputMFType = mfTypeOut;
69
70            obj.input1Overlap = overlapIn1;
71            obj.input2Overlap = overlapIn2;
72            obj.outputOverlap = overlapOut;
73
74            obj.NAME = NAME;
75
76            INMF = {...
77                obj.input1Name, obj.input1MFNum, obj.input1MFType, ...
78                obj.input1Overlap, [-1 1];...
79                obj.input2Name, obj.input2MFNum, obj.input2MFType, ...
80                obj.input2Overlap, [-1 1]};
81
82            OUTMF = {...
83                obj.outputName, obj.outputMFNum, obj.outputMFType, ...
84                obj.outputOverlap, [-1 1]};
85
86            if nargin == 18
87                obj.FLC = obj.Fuzzy(...
88                    'INMF', INMF,...
89                    'OUTMF', OUTMF,...
90                    'TYPE', 'sugeno',...
91                    'ORIENTATION', orientation,...
92                    'CENTERS', distribution,...
93                    'VERBOSE', 1);
94            else
95                TYPE = 'sugeno';
96                for index = 1:nargin
97                    try
```

```
98                          switch varargin{index}
99                              case 'TYPE'
100                                 TYPE = varargin{index + 1};
101                             otherwise
102                         end
103                     catch
104                     end
105                 end
106
107
108                 obj.FLC = obj.Fuzzy(...
109                     'INMF', INMF,...
110                     'OUTMF', OUTMF,...
111                     'TYPE', TYPE,...
112                     'ORIENTATION', orientation,...
113                     'CENTERS', distribution,...
114                     'VERBOSE', 1, ...
115                     varargin{:} );
116             end
117
118             obj.absRange = [-1 : 0.01 : 1];
119             obj.absRangeSize = length(obj.absRange);
120
121             obj.ENABLE = 1; % Force enable to calculate LUT
122             obj.ENABLE = ENABLE; % User option ENABLE/DISABLE
123
124             obj.IRR = zeros(2, 49);
125             obj.boundaryCount1 = 0;
126             obj.boundaryCount2 = 0;
127
128             obj.alphas = -25:.1:25;
129             obj.gains = zeros(1, length(obj.alphas));
130         end
131
132         % FIS Evaluation
133         output = evalFLC(obj, Input1, Input2, CONTROLLER, varargin)
134
135         PushGain(obj, gain)
136         LUT_Calc(obj)
137
138         output = Lookup(obj, Input1, Input2)
139         output = Fuzzy(obj, varargin)
140         output = mfGen(obj, Number, PrimaryType)
141         output = ruleGen(obj, X, varargin)
142         output = outputMatrix(obj, X, varargin)
143         output = ruleSpace(obj, varargin)
144         output = generatefis(obj, FIS_TYPE, FIS_INPUT, ...
145             FIS_OUTPUT, varargin )
146     end
147 end
```

A.6.2 FUZZY LOGIC EVALUATOR

The following code provides an evaluation interface for the fuzzy logic system to simplify wider system integration.

```
function output = evalFLC(obj, Input1, Input2, CONTROLLER, varargin)
% Identify boundary operations and expand if necessary
countThreshold = 10;
sensitivity = 0.25;

switch obj.NAME
    case 'A_E'
        multiplier = 2;
    case 'AI_EI'
        multiplier = 1.5;
    case 'AD_ED'
        multiplier = 4;

    case 'Kp'
        multiplier = 2;
    case 'Ki'
        multiplier = 1.5;
    case 'Kd'
        multiplier = 4;
    otherwise
        multiplier = 1;
end

if obj.disableGainSense == 0
    if abs(Input1 - obj.input1Gain)/obj.input1Gain < sensitivity ...
            || abs(Input1)>obj.input1Gain
        obj.boundaryCount1 = obj.boundaryCount1 + 1;
        if obj.boundaryCount1 > countThreshold
            obj.input1Gain = obj.input1Gain * multiplier;
            fprintf('Increasing Input1 Boundary in %s to %1.2f\n', ...
                obj.NAME, obj.input1Gain);
        elseif abs(Input1)>obj.input1Gain
            obj.input1Gain = 1/abs(Input1);
        end
    else
        if obj.boundaryCount1 > 0
            obj.boundaryCount1 = obj.boundaryCount1 - 1;
        end
    end

    if abs(Input2 - obj.input2Gain)/obj.input2Gain < sensitivity ...
            || abs(Input2)>obj.input2Gain
        obj.boundaryCount2 = obj.boundaryCount2 + 1;
        if obj.boundaryCount2 > countThreshold
            obj.input2Gain = obj.input2Gain * multiplier;
            fprintf('Increasing Input2 Boundary in %s to %1.2f\n', ...
                obj.NAME, obj.input2Gain);
```

```
48          elseif abs(Input2)>obj.input2Gain
49              obj.input2Gain = 1/abs(Input2);
50          end
51      else
52          if obj.boundaryCount2 > 0
53              obj.boundaryCount2 = obj.boundaryCount2 - 1;
54          end
55      end
56  end
57
58  % Evaluate
59  [output, temp] = evalfis( ...
60      [Input1 / obj.input1Gain, Input2 / obj.input2Gain], obj.FLC);
61
62  % Identify output boundary operations and expand if necessary
63  if abs(output - obj.outputGain.last)/obj.outputGain.last < sensitivity ...
64          && CONTROLLER.ctrlOutput.last ...
65          < (CONTROLLER.elevLimit + CONTROLLER.upperOffset)*0.9 ...
66          && CONTROLLER.ctrlOutput.last ...
67          > (-CONTROLLER.elevLimit + CONTROLLER.lowerOffset)*0.9
68      obj.outputGain.Push(obj.outputGain.last * multiplier);
69      fprintf('Increasing Output Boundary in %s to %1.2f\n', ...
70          obj.NAME, obj.outputGain.last);
71  end
72
73  switch obj.NAME
74
75      otherwise
76          outputscale = 1;
77  end
78
79  output = output * obj.outputGain.last * obj.ENABLE * outputscale;
80
81  % Store input rule calculations
82  obj.IRR = obj.IRR + temp';
83  end
```

A.6.3 FUZZY LOGIC CONSTRUCTOR

The following code is useful for constructing the fuzzy logic system. It contains low-level instantiation as well as option specification for creating a wide variety of systems.

```
1  function [ varargout ] = Fuzzy(obj, varargin )
2  %Creates a Fuzzy Inference System based on supplied inputs
3  %% Primary Fuzzy Inference System
4  %    Version:     0.1
5  %    Date:        07 April 2016
6  %    Author:      Keith Benjamin
7  %
8  %
9  %
```

```
10 %   Inputs
11 %       - INMF {name, #MFs, type, %overlap, n-d range}
12 %       - OUTMF {name, #MFs, type, %overlap, n-d range}
13 %       - Step = 'double' < range
14 %       - 'LUT' to calculate LUT
15
16 INMF = {...
17     'alphaCommandError', 15, 'trimf', 1.5, [-1 1];...
18     'alpha',             15, 'trimf', 1.5, [-1 1];...
19     'alphaDot',          15, 'trimf', 1.5, [-1 1];...
20     };
21 OUTMF = {'u', 15, 'constant', 1.5, [-1 1]};
22 Step = 0.2;
23 LUT = -1;
24 TYPE = 'sugeno';
25
26 if nargin ~= 0
27     for index = 1:nargin
28         try
29             switch varargin{index}
30                 case 'STEP'
31                     Step = varargin{index+1};
32                 case 'INMF'
33                     INMF = varargin{index+1};
34                 case 'OUTMF'
35                     OUTMF = varargin{index+1};
36                 case 'LUT'
37                     LUT = 0;
38                 case 'TYPE'
39                     TYPE = varargin{index+1};
40                 otherwise
41             end
42         catch
43         end
44     end
45 end
46
47 %% 3x1 FLC System Definition
48 warning('off','MATLAB:colon:nonIntegerIndex'); % Suppress warnings
49 inputFunctions = INMF(:, [1:2, 4:5]);
50
51 for index = 1:size(INMF, 1)
52     eval(sprintf('%sMF = obj.mfGen(INMF{%d,2}, INMF{%d,3});', ...
53         INMF{index, 1}, index, index));
54
55     if index == 1
56         inputFunctions{index, 2} = eval(sprintf('%sMF', INMF{index, 1}));
57     else
58         inputFunctions{index, 2} = eval(sprintf('%sMF', INMF{index, 1}));
59     end
60
61 end
```

```
62
63 % Define output membership functions
64 outputFunctions = OUTMF(:, [1:2, 4:5]);
65 for index = 1:size(OUTMF, 1)
66     eval(sprintf('%sMF = obj.mfGen(OUTMF{%d,2}, OUTMF{%d,3});', ...
67         OUTMF{index, 1}, index, index));
68
69     if index == 1
70         outputFunctions{index, 2} = eval(sprintf('%sMF', OUTMF{index, 1}));
71     else
72         outputFunctions{index, 2} = eval(sprintf('%sMF', OUTMF{index, 1}));
73     end
74 end
75
76 % Generate FIS using 'generatefis'
77 fprintf('Generating the following FIS\n')
78 fprintf('Fuzzy Inference Type: %s\n', TYPE);
79 fprintf('\tInputs:\t%d\n', size(INMF, 1));
80 tracker = 1;
81 for index = 1:size(INMF, 1)
82     fprintf('\t\tInput %d - MFs: %d\tPrimary Type: %s\t\tOverlap: %0.2f\tRange: [%d %d]\n'
       ,...
83         index, INMF{index, 2}, INMF{index, 3}, ...
84         INMF{index, 4}, INMF{index, 5});
85     tracker = tracker * INMF{index, 2};
86 end
87
88 fprintf('\n\tOutputs:\t%d\n', size(OUTMF, 1));
89 for index = 1:size(OUTMF, 1)
90     fprintf('\t\tOutput %d - MFs: %d\tPrimary Type: %s\t\tOverlap: %0.2f\tRange: [%d %d]\n'
       ,...
91         index, OUTMF{index, 2}, OUTMF{index, 3}, ...
92         OUTMF{index, 4}, OUTMF{index, 5});
93 end
94 fprintf('\n\tRule Num:\t%d\n\n', tracker);
95
96
97 FLC = obj.generatefis(...
98     TYPE,...
99     inputFunctions,...
100     outputFunctions,...
101     varargin{:}...
102     );
103
104 %% Construct 3x1 FLCLUT
105 if LUT ~= -1
106     fprintf('Generating Lookup Table with Breakpoint Step Divisor: %0.3f\n', Step);
107     [FLCLUT, LUTSize] = calcLUT(inputFunctions, FLC, 'STEP', Step);
108     FLCLUT = single(reshape(FLCLUT, LUTSize));
109
110 end
111
```

```
112 for index = 1:nargout
113     switch index
114         case 1
115             varargout{1} = FLC;
116         case 2
117             varargout{2} = FLCLUT;
118         otherwise
119     end
120 end
121 end
```

A.6.4 MEMBERSHIP FUNCTION GENERATOR

The following code is useful for describing, in detail, input and output membership functions.

```
1  function [ output_args ] = ...
2      generatefis(obj, FIS_TYPE, FIS_INPUT, FIS_OUTPUT, varargin )
3  %generatefis Creates a new Fuzzy Inference System with normalized I/O
4  %   FIS_TYPE => Fuzzy Logic TYPE - ('mamdani','sugeno') default:'sugeno'
5  %   FIS_INPUT => Cell Array of Format {Name, [Labels; MFTYPE]}
6  %
7  %   Version:    0.1
8  %   Date:       04 March 2016
9  %   Author:     Keith Benjamin
10 %
11 %   Notes:  Currently only supports trimf,zmf,smf inputs and constant
12 %           outputs.
13 %           Mamdani systems not supported.
14
15 %% System Parameters
16 % Set Defaults
17 if strcmp(FIS_TYPE, 'sugeno')
18     RANGE = [-1 1];
19     AND = 'prod';
20     OR = 'probor';
21     ORIENTATION = 'left';
22     IMP = 'prod';
23     AGG = 'max';
24     DEFUZZ = 'wtaver';
25     RULES = -1;
26     FIS_CENTERS = 'linear';
27 elseif strcmp(FIS_TYPE, 'mamdani')
28     RANGE = [-1 1];
29     AND = 'min';
30     OR = 'max';
31     ORIENTATION = 'left';
32     IMP = 'min';
33     AGG = 'max';
34     DEFUZZ = 'centroid';
35     RULES = -1;
```

```
36     FIS_CENTERS = 'linear';
37 else
38     error('Type must be (sugeno | mamdani)')
39 end
40
41 % Override appropriate system parameters based on supplied options
42 for index = 4:nargin
43     try
44         switch varargin{index}
45             case 'ORIENTATION'
46                 ORIENTATION = varargin{index+1};
47                 %             case 'TYPE'
48                 %                 FIS_TYPE = varargin{index+1};
49             case 'RANGE'
50                 RANGE = varargin{index+1};
51             case 'AND'
52                 AND = varargin{index+1};
53             case 'OR'
54                 OR = varargin{index+1};
55             case 'IMP'
56                 IMP = varargin{index+1};
57             case 'AGG'
58                 AGG = varargin{index+1};
59             case 'DEFUZZ'
60                 DEFUZZ = varargin{index+1};
61             case 'RULES'
62                 RULES = varargin{index+1};
63             case 'CENTERS'
64                 FIS_CENTERS = varargin{index+1};
65             otherwise
66         end
67     catch
68     end
69 end
70
71 %% Create New Fuzzy Inference System
72 flc = newfis('flc', FIS_TYPE, AND, OR, IMP, AGG, DEFUZZ);
73
74 if ~isempty(FIS_INPUT)
75     % Add default range to input parameter if missing
76     if ~(size(FIS_INPUT, 2) == 4)
77         temp = cell(size(FIS_INPUT, 1), size(FIS_INPUT, 2) + 1);
78         for index = 1:size(FIS_INPUT, 1)
79             temp(index, :) = [FIS_INPUT(index, :), RANGE];
80         end
81     else
82         temp = FIS_INPUT;
83     end
84
85     % Add inputs to FIS
86     flc = batchAdd(flc, 'input', temp, FIS_CENTERS);
87 end
```

```
88
89 if ~isempty(FIS_OUTPUT)
90     % Add default range to output parameter if missing
91     if ~(size(FIS_OUTPUT, 2) == 4)
92         for index = 1:size(FIS_OUTPUT, 1)
93             temp = {FIS_OUTPUT{index, :}, RANGE};
94         end
95     else
96         temp = FIS_OUTPUT;
97     end
98
99     % Add output to FIS
100    flc = batchAdd(flc, 'output', temp, FIS_CENTERS);
101 end
102
103 % clear temp;
104 if RULES ~= -1
105     % Add Rules from custom Rules Matrix
106     flc = addrule(flc, RULES);
107 else
108     % Add auto-generated rulebase
109     flc = addrule(flc, obj.ruleGen(getfis(flc,'inmfs'), ...
110         'ORIENTATION', ORIENTATION));
111 end
112
113 % Return Fuzzy Logic System
114 output_args = flc;
115 end
116
117 function back = batchAdd(flc, FIS_TYPE, FIS_INPUT, FIS_CENTERS)
118 %% Add Input Membership Functions
119
120 for index_input = 1:size(FIS_INPUT)
121     % Create Fuzzy Logic I/O
122     RANGE = FIS_INPUT{index_input, 4};
123
124     % Add new input
125     flc = addvar(...
126         flc,...
127         char(FIS_TYPE),...
128         char(FIS_INPUT(index_input)),...
129         RANGE...
130         );
131
132     mfNumber = size(FIS_INPUT{index_input,2},2);
133
134     if strcmp(FIS_CENTERS, 'linear')
135         normalizedMFCenters = linspace(-1, 1, mfNumber);
136     elseif strcmp(FIS_CENTERS, 'square');
137         normalizedMFCenters = linspace(-1, 1, mfNumber);
138         temp = sign(normalizedMFCenters);
139         normalizedMFCenters = normalizedMFCenters .^ 2;
```

```
140         normalizedMFCenters = normalizedMFCenters .* temp;
141     elseif strcmp(FIS_CENTERS, 'cube');
142         normalizedMFCenters = linspace(-1, 1, mfNumber) .^ 3;
143     elseif strcmp(FIS_CENTERS, 'quad');
144         normalizedMFCenters = linspace(-1, 1, mfNumber);
145         temp = sign(normalizedMFCenters);
146         normalizedMFCenters = normalizedMFCenters .^ 4;
147         normalizedMFCenters = normalizedMFCenters .* temp;
148     elseif strcmp(FIS_CENTERS, 'pent');
149         normalizedMFCenters = linspace(-1, 1, mfNumber);
150         temp = sign(normalizedMFCenters);
151         normalizedMFCenters = normalizedMFCenters .^ 5;
152         normalizedMFCenters = normalizedMFCenters .* temp;
153     else
154         error('FIS center distribution must be defined.');
155     end
156 
157     mfCenter = normalizedMFCenters * max(abs(RANGE));
158     OVERLAP = FIS_INPUT{index_input, 3};
159 
160     % Add Membership Functions
161     for index_mf = 1:mfNumber
162         % Calculate Left | Right | Center points -> required for trimf
163         % Center used for Constant FIS_OUTPUT location for Sugeno systems
164         try
165             mfRight = ((mfCenter(index_mf+1) - mfCenter(index_mf))/2);
166         catch
167             mfRight = ((mfCenter(index_mf-1) - mfCenter(index_mf))/2);
168         end
169 
170         try
171             mfLeft = ((mfCenter(index_mf-1) - mfCenter(index_mf))/2);
172         catch
173             mfLeft = ((mfCenter(index_mf+1) - mfCenter(index_mf))/2);
174         end
175 
176         %TODO pass MF overlapping to here from higher functions
177         % Setting Left/Right Membership Function bounds
178         %          mfCenter
179         %          mfLeft
180         %          mfRight
181         %          OVERLAP
182         %          index_mf
183         %          mfCenter(index_mf)
184         %          (mfCenter(index_mf)) + (mfLeft * (1 + OVERLAP/100))
185 
186         temp = min([abs(mfLeft* (1 + OVERLAP/100)) ...
187             abs(mfRight* (1 + OVERLAP/100))]);
188         mfLeft  = mfCenter(index_mf) - temp;
189         mfRight = mfCenter(index_mf) + temp;
190 
191         % Adds appropriate membership function based on FIS_INPUT criteria
```

```
192         switch char(FIS_INPUT{index_input, 2}(2, index_mf))
193             case 'trimf'
194                 flc = addmf(...
195                     flc,...
196                     FIS_TYPE,...
197                     index_input,...
198                     char(FIS_INPUT{index_input,2}(1,index_mf)),...
199                     char(FIS_INPUT{index_input,2}(2,index_mf)),...
200                     [mfLeft, mfCenter(index_mf), mfRight]...
201                     );
202             case 'zmf'
203                 flc = addmf(...
204                     flc, ...
205                     FIS_TYPE, ...
206                     index_input, ...
207                     char(FIS_INPUT{index_input,2}(1,index_mf)), ...
208                     char(FIS_INPUT{index_input,2}(2,index_mf)), ...
209                     [mfCenter(index_mf), mfRight]...
210                     );
211             case 'smf'
212                 flc = addmf(...
213                     flc, ...
214                     FIS_TYPE, ...
215                     index_input, ...
216                     char(FIS_INPUT{index_input,2}(1,index_mf)), ...
217                     char(FIS_INPUT{index_input,2}(2,index_mf)), ...
218                     [mfLeft, mfCenter(index_mf)]...
219                     );
220             case 'constant'
221                 flc = addmf(flc, ...
222                     FIS_TYPE, ...
223                     index_input, ...
224                     char(FIS_INPUT{index_input,2}(1,index_mf)), ...
225                     char(FIS_INPUT{index_input,2}(2,index_mf)), ...
226                     mfCenter(index_mf)...
227                     );
228         end
229     end
230 end
231
232 % Return updated Fuzzy Logic System
233 back = flc;
234
235 end
```

A.7 CONTROLLER CODE

A.7.1 CONTROLLER CLASS

The following code describes the controller class. This class covers the entire controller section of the system architecture.

```
classdef Controller < handle
    %CONTROLLER Advanced Data Structure for aircraft controller
    %   Defines a 3-Level Parallel Fuzzy Logic Controller with internal
    %   integration and AlphaDotDot estimation.
    properties
        POutput
        DOutput
        errOutput
        IOutput
        trmOutput
        ctrlOutput

        % Fuzzy Logic Controllers
        errorFLC
        velocFLC
        accelFLC
        positFLC

        controllerIndex

        % Fuzzy Logic Output Gains
        errorGain
        DGain
        IGain
        PGain

        % Controller Trim
        trim
        trim_calculated
        trim_gain

        % Controller Direction
        sysDirection
        errDirection
        velDirection
        accDirection
        posDirection

        sampleTime

        % Aircraft Parameters
        alphaLimit
        alphaDotLimit
```

```
44
45          alpha_dot
46          alpha_dot_dot
47
48          elevLimit
49          lowerOffset
50          upperOffset
51
52          Kp
53          Ki
54          Kd
55
56          A_E
57          AD_ED
58          AI_EI
59
60          AirspeedKp
61          AirspeedKi
62          AirspeedKd
63
64          disableGainSense
65          windtunnel
66
67          PIDX
68          PIDY
69          sysPID
70      end
71
72
73
74      % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
75      % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
76      % Public Methods
77      % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
78      % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
79      methods (Access = public)
80          % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
81          % Constructor
82          % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
83          function obj = Controller(errorGain, PGain, DGain, IGain, ...
84                  alphaLimit, elevLimit, lowerOffset, upperOffset, ...
85                  ctrlFIS, disableGainSense,windtunnel)
86              % Fuzzy Logic Output Gains
87              % Controller Directions
88              obj.windtunnel = windtunnel;
89              obj.disableGainSense = disableGainSense;
90              obj.DOutput = CustomQueue(3);
91
92              obj.IOutput = CustomQueue(3);
93              obj.POutput = CustomQueue(3);
94
95              obj.ctrlOutput = CustomQueue(3);
```

```
96
97            obj.sysDirection =  1;
98            obj.errDirection = -1;
99            obj.velDirection = -1;
100           obj.accDirection = -1;
101           obj.posDirection = -1;
102
103           obj.DGain = DGain;
104           obj.IGain = IGain;
105           obj.PGain = PGain;
106
107           obj.alphaLimit = alphaLimit;
108           obj.alphaDotLimit  = 4 * alphaLimit;
109
110           % Fuzzy Logic Controllers Initialization
111           obj.InitializeFLC(ctrlFIS);
112
113           % Controller Trim
114           obj.trim = ...
115               containers.Map('KeyType', 'double', 'ValueType', 'double');
116           obj.trim_calculated = ...
117               containers.Map( ...
118               single(alphaLimit), single(zeros(1, length(alphaLimit))));
119           obj.trim_gain = ...
120               containers.Map('KeyType', 'double', 'ValueType', 'double');
121
122           obj.controllerIndex = 1;
123
124           obj.alpha_dot = CustomQueue(3);
125           obj.alpha_dot_dot = CustomQueue(3);
126
127           obj.elevLimit = elevLimit;
128           obj.lowerOffset = lowerOffset;
129           obj.upperOffset = upperOffset;
130
131
132           obj.Kp = CustomQueue(3);
133           obj.Ki = CustomQueue(3);
134           obj.Kd = CustomQueue(3);
135
136           obj.AirspeedKp = polyfit([150 300 600], [1 .75 .5], 2);
137           obj.AirspeedKi = polyfit([150 300 600], [2 1.5 1], 2);
138           obj.AirspeedKd = polyfit([150 300 600], [2 1.5 1], 2);
139
140
141           obj.PIDX = 0;
142           obj.PIDY = 0;
143           obj.sysPID = ...
144               evalin('base', 'ss(pidtune(sys(alphaLoc),''PI''))');
145       end
146
147       % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
148         % Set Sample Time Interval
149         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
150         function setSampleTime(obj, interval)
151             obj.sampleTime = interval;
152         end
153 
154         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
155         % Set New Trim Condition and Calculate Estimated Trim Values
156         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
157         stored = setTrim(obj, alpha)
158 
159         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
160         % Generate Control Output
161         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
162         output = control(obj, SUPERVISOR, alpha, alphaDot, Speed)
163         output = Error2(obj, SUPERVISOR, Speed)
164         output = testPID(obj, SUPERVISOR)
165         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
166         % Reset Controller Gains
167         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
168         ResetGains(obj)
169     end
170 
171 
172     % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
173     % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
174     % Protected Methods
175     % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
176     % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
177     methods (Access = protected)
178         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
179         % AlphaDotDot Estimator
180         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
181         AlphaDotDotEst(obj, alphaDot)
182 
183         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
184         % Initialize Fuzzy Logic Controllers
185         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
186         InitializeFLC(obj, params)
187     end
188 end
```

A.7.2 ESTIMATOR – $\ddot{\alpha}$

The following code estimates angle-of-attack acceleration.

```
1 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % AlphaDotDot Estimator
3 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function AlphaDotDotEst(obj, alphaDot)
5 obj.alpha_dot.Push(alphaDot);
6 obj.alpha_dot_dot(obj.controllerIndex + 1) = ...
```

```
7       (-obj.alpha_dot.data(3) ...
8       + 4 * obj.alpha_dot.data(2) ...
9       - 3 * obj.alpha_dot.data(1)) ...
10      / (2 * obj.sampleTime);
11  end
```

A.7.3 CONTROL SELECTION

The following code selects the control regime – either PID or FLC. Both may be used during simulation; however, only *Error2* is useful in real-world experimentation.

```
1 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Generate Control Output
3 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function output = ...
5     control(obj, SUPERVISOR, alpha, alphaDot, Speed)
6 
7 % output = obj.Error2(SUPERVISOR, Speed);
8 output = obj.testPID(SUPERVISOR.alphaError.last);
9 end
```

A.7.4 FUZZY LOGIC INITIALIZER

The following code initializes the fuzzy logic systems embedded in the supervisor.

```
1 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Initialize Fuzzy Logic Controllers
3 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function InitializeFLC(obj, params)
5 params_A_E = params(1);
6 params_AI_EI = params(2);
7 params_AD_ED = params(3);
8 
9 NB = 1;
10 NM = 2;
11 NS = 3;
12 ZO = 4;
13 PS = 5;
14 PM = 6;
15 PB = 7;
16 temp1 = cell(7,1);
17 temp1(:) = {1:7};
18 temp1 = cell2mat(temp1);
19 
20 temp2 = temp1';
21 
22 a_e = [
23     PB  PB  PB  PB  PB  PB  PB  ;
24     PM  PM  PM  PM  PM  PM  PM  ;
25     PS  PS  PS  PS  PS  PS  PS  ;
```

```
26      ZO  ZO  ZO  ZO  ZO  ZO  ZO  ;
27      NS  NS  NS  NS  NS  NS  NS  ;
28      NM  NM  NM  NM  NM  NM  NM  ;
29      NB  NB  NB  NB  NB  NB  NB
30      ];
31
32  rule = [temp1(:) temp2(:) a_e(:) ones(49,2)];
33
34  obj.A_E = FIS_2X1(...
35      'e',  params_A_E.inputRange1, params_A_E.MFs, ...
36      params_A_E.inputType, params_A_E.overlap,...
37      'ec', params_A_E.inputRange2, params_A_E.MFs, ...
38      params_A_E.inputType, params_A_E.overlap,...
39      'u',  params_A_E.outputRange, params_A_E.MFs, ...
40      params_A_E.outputType, params_A_E.overlap,...
41      'left', ...
42      params_A_E.distribution, ...
43      'A_E', ...
44      params_A_E.ENABLE, obj.disableGainSense, ...
45      'RULES', rule);
46
47  ad_ed = [
48      PB  PB  PB  PB  PB  PB  PB  ;
49      PM  PM  PM  PM  PM  PM  PM  ;
50      PS  PS  PS  PS  PS  PS  PS  ;
51      ZO  ZO  ZO  ZO  ZO  ZO  ZO  ;
52      NS  NS  NS  NS  NS  NS  NS  ;
53      NM  NM  NM  NM  NM  NM  NM  ;
54      NB  NB  NB  NB  NB  NB  NB
55      ];
56
57  rule = [temp1(:) temp2(:) ad_ed(:) ones(49,2)];
58
59  obj.AD_ED = FIS_2X1(...
60      'e',  params_AD_ED.inputRange1, params_AD_ED.MFs, ...
61      params_AD_ED.inputType, params_AD_ED.overlap,...
62      'ec', params_AD_ED.inputRange2, params_AD_ED.MFs, ...
63      params_AD_ED.inputType, params_AD_ED.overlap,...
64      'u',  params_AD_ED.outputRange, params_AD_ED.MFs, ...
65      params_AD_ED.outputType, params_AD_ED.overlap,...
66      'left', ...
67      params_AD_ED.distribution, ...
68      'AD_ED', ...
69      params_AD_ED.ENABLE, obj.disableGainSense, ...
70      'RULES', rule);
71
72  ai_ei = [
73      PB  PB  PB  PB  PB  PB  PB  ;
74      PM  PM  PM  PM  PM  PM  PM  ;
75      PS  PS  PS  PS  PS  PS  PS  ;
76      ZO  ZO  ZO  ZO  ZO  ZO  ZO  ;
77      NS  NS  NS  NS  NS  NS  NS  ;
```

```
78      NM   NM   NM   NM   NM   NM   NM   ;
79      NB   NB   NB   NB   NB   NB   NB
80      ];
81
82  rule = [temp1(:) temp2(:) ai_ei(:) ones(49,2)];
83
84  obj.AI_EI = FIS_2X1(...
85      'e',  params_AI_EI.inputRange1, params_AI_EI.MFs, ...
86      params_AI_EI.inputType, params_AI_EI.overlap,...
87      'ec', params_AI_EI.inputRange2, params_AI_EI.MFs, ...
88      params_AI_EI.inputType, params_AI_EI.overlap,...
89      'u',  params_AI_EI.outputRange, params_AI_EI.MFs, ...
90      params_AI_EI.outputType, params_AI_EI.overlap,...
91      'left', ...
92      params_AI_EI.distribution, ...
93      'AI_EI', ...
94      params_AI_EI.ENABLE, obj.disableGainSense,...
95      'RULES', rule);
96  end
```

A.7.5 FUZZY LOGIC CONTROLLER

The following code implements the fuzzy logic control regime.

```
1  function output = Error2(obj, SUPERVISOR, Speed)
2  % Establish individual control arguments
3  obj.POutput.Push(...
4      obj.Kp.last * ...
5      polyval(obj.AirspeedKp, Speed) * ...
6      obj.A_E.evalFLC(0, SUPERVISOR.alphaError.last, obj));
7
8  obj.DOutput.Push( ...
9      obj.Kd.last * ...
10     polyval(obj.AirspeedKd, Speed) * ...
11     obj.AD_ED.evalFLC(0, SUPERVISOR.errorChange.last, obj));
12
13 obj.IOutput.Push( ...
14     obj.Ki.last * ...
15     polyval(obj.AirspeedKi, Speed) * ...
16     obj.AI_EI.evalFLC( 0, SUPERVISOR.errorIntegrator.last, obj));
17
18 % Assemble output control value
19 obj.ctrlOutput.Push(1 ...
20     * (obj.POutput.last...
21     +  obj.DOutput.last...
22     +  obj.IOutput.last...
23 ));
24
25 obj.ctrlOutput.Push( ...
26     min( ...
27     max(obj.ctrlOutput.last, -obj.elevLimit+obj.lowerOffset), ...
28     obj.elevLimit+obj.upperOffset) ...
```

```
29     );
30
31 output = obj.ctrlOutput.last;
32 obj.controllerIndex = obj.controllerIndex + 1;
33 end
```

A.7.6 GAIN CLEARING

The following code is useful for reseting gain parameters at run-time, if desired.

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Reset Controller Gains
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function ResetGains(obj)
5 obj.accelFLC.PushGain(obj.accelGain);
6 obj.velocFLC.PushGain(obj.velocGain);
7 obj.errorFLC.PushGain(obj.errorGain);
8 end
```

A.7.7 SET TRIM CONDITION

The following code is useful for recalling stored trim conditions, if desired.

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Set New Trim Condition and Calculate Estimated Trim Values
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function stored = setTrim(obj, alpha)
5 ALPHA = round(alpha, 1, 'decimal'); % Use 0.1 degree resolution
6
7 % Average new trim value with previous value otherwise store new value
8 if isKey(obj.trim, ALPHA)
9     temp = obj.trim(ALPHA);
10    remove(obj.trim, ALPHA);     % Old key must be removed first
11    obj.trim(ALPHA) = (temp + obj.ctrlOutput.last) / 2;
12    updated = 1;
13 else
14    obj.trim(ALPHA) = obj.ctrlOutput.last;
15    updated = 1;
16 end
17
18 % Interpolate/Extrapolate trim values for the entire flight envelope
19 if updated == 1
20    polyX = cell2mat(obj.trim.keys);
21    polyY = cell2mat(obj.trim.values);
22    calcX = cell2mat(obj.trim_calculated.keys);
23
24    if obj.trim.length ~= 1
25        % Interpolate using a cubic function - necessary for zero crossing
26        calcY = interp1(polyX, polyY, calcX, 'pchip', 'extrap');
27
28        % Set saturation points to the first and last actual detected trim
```

```
29          % values. Without this, trim estimates outside the previously
30          % measured area can cause instability
31          lowerStop =  find(calcX < polyX(1),    1, 'last');
32          upperStart = find(calcX > polyX(end), 1, 'first');
33
34          calcY(1:lowerStop) = polyY(1);
35          calcY(upperStart:end) = polyY(end);
36      else
37          calcY = ones(1, length(calcX)) * polyY;
38      end
39
40      obj.trim_calculated = containers.Map(calcX, calcY);
41  end
42
43  stored = updated;
44  end
```

A.7.8 PID CONTROLLER

The following code implements the PID control regime.

```
1 function output = testPID(obj, error)
2 % Establish individual control arguments
3 obj.PIDX = obj.sysPID.a * obj.PIDX + obj.sysPID.b * error;
4 obj.PIDY = obj.sysPID.c * obj.PIDX + obj.sysPID.d * error;
5
6 output = obj.PIDY;
7 end
```

A.8 SUPERVISOR CODE

A.8.1 SUPERVISOR CLASS

The following code describes the supervisor class. This class covers the entire supervisor section of the system architecture.

```
classdef Supervisor < handle
    properties
        fftSensitivity
        fftLength
        ssLength

        % Used by FFT
        velFreq
        accFreq
        errFreq

        % Threshold for SteadyState Detection
        steadyStateSensitivity

        % Parameter History Queue
        alphaHistory
        alphaDotHistory
        alphaDotDotHistory
        alphaCommandHistory

        % Error History Queues
        alphaError
        alphaDotError
        alphaDotDotError

        % SteadyState Indicator
        ssTrigger

        % Controller Gain Adjustment FLCs
        positGainFLC
        velocGainFLC
        accelGainFLC
        errorGainFLC

        % Variables for Reference Model Calculation
        sysRefCT
        sysRefDT

        sysRefCTX
        sysRefCTY

        sysRefDTX
        sysRefDTY
```

```
44
45          sysRefLUT
46          sysRefLUTx
47          sysRefTime
48
49          % Integrator Queues
50          errorIntegrator
51          errorDotIntegrator
52          errorDotDotIntegrator
53
54          % Differentiator Queues
55          errorDifferentiator
56          errorDotDifferentiator
57
58          % Sample Period of System
59          samplePeriod
60          commandTime
61
62          referenceCommand
63          lastCommand
64
65          superFilter
66
67          errorSum
68          errorChange
69
70          Kp
71          Ki
72          Kd
73
74          errorLimit
75          errorChangeLimit
76          elevLimit
77          elevHistory
78          elevDotHistory
79
80          tester
81          changeStanddown
82          decreaseTrigger
83          disableGainSense
84          windtunnel
85          KiChangeStanddown
86          KpChangeStanddown
87      end
88
89      % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
90      % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
91      % Public Methods
92      % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
93      % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
94      methods
95          % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
96          % Constructor
97          % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
98          function obj = Supervisor( fftLength, fftSensitivity, ...
99                  ssLength, ssSensitivity, samplePeriod, alphaLimit, ...
100                 elevLimit, reference, superFIS, disableGainSense, ...
101                 windtunnel)
102
103             obj.windtunnel=windtunnel;
104             obj.tester = polyfit([-(pi/180)*25 0 ...
105                 (pi/180)*25], [-8 0 8], 2);
106             obj.changeStanddown = 0;
107             obj.disableGainSense = disableGainSense;
108
109             obj.samplePeriod = CustomQueue(2);
110             obj.commandTime = 0;
111             obj.errorLimit = 2*alphaLimit;
112             obj.errorChangeLimit = 4*obj.errorLimit;
113             obj.elevLimit = elevLimit;
114
115             obj.fftLength = fftLength / samplePeriod;
116             obj.ssLength  = ssLength  / samplePeriod;
117             obj.fftSensitivity = fftSensitivity / samplePeriod;
118
119             obj.steadyStateSensitivity = ssSensitivity;
120
121             obj.alphaHistory        = CustomQueue(obj.ssLength);
122             obj.alphaDotHistory     = CustomQueue(obj.ssLength);
123             obj.alphaDotDotHistory  = CustomQueue(obj.ssLength);
124             obj.alphaCommandHistory = CustomQueue(obj.ssLength);
125
126             obj.alphaError       = CustomQueue(obj.ssLength);
127             obj.alphaDotError    = CustomQueue(obj.ssLength);
128             obj.alphaDotDotError = CustomQueue(obj.ssLength);
129
130             obj.elevHistory = CustomQueue(obj.ssLength);
131             obj.elevDotHistory = CustomQueue(obj.ssLength);
132
133             obj.ssTrigger = 0;
134
135             obj.ReferenceSetup(reference(1), reference(2), samplePeriod);
136
137             obj.errorIntegrator    = CustomQueue(obj.ssLength);
138             obj.errorDotIntegrator = CustomQueue(obj.ssLength);
139             obj.errorDotDotIntegrator = CustomQueue(obj.ssLength);
140
141             obj.errorDifferentiator    = CustomQueue(obj.ssLength);
142             obj.errorDotDifferentiator = CustomQueue(obj.ssLength);
143
144             obj.referenceCommand = 0;
145             obj.lastCommand = [0; 0; 0;];
146
147             obj.errorSum = CustomQueue(5);
```

```
148             obj.errorChange = CustomQueue(5);
149
150             obj.InitializeFLC(superFIS);
151             obj.decreaseTrigger = 0;
152
153             obj.KiChangeStanddown = 0;
154             obj.KpChangeStanddown = 0;
155         end
156
157         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
158         % Primary Supervisor Driver
159         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
160         super(obj, CONTROLLER, alpha, alphaDot, ...
161             alphaCommand, sampleTime, elevator)
162     end
163
164
165
166
167     % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
168     % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
169     % Private Methods
170     % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
171     % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
172     methods (Access = private)
173         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
174         % Store Variable Histories in Queue
175         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
176         StoreHistories(obj, alpha, alphaDot, alphaCommand, elevator)
177
178         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
179         % Controller Output Gain Updates
180         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
181         GainUpdates(obj, CONTROLLER, ~, ~)
182
183         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
184         % Error Calculation
185         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
186         CalculateError(obj, alpha, alphaDot)
187
188         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
189         % New Command Detection
190         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
191         NewCommand(obj, CONTROLLER)
192
193         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
194         % Steady State Detector
195         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
196         SteadyState(obj, CONTROLLER, alpha)
197
198         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
199         % Frequency Suppressor
```

```
200         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
201         output = FreqSuppress(obj, signal)
202 
203         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
204         % Reference Calculation
205         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
206         ReferenceCalculation(obj, alphaCommand)
207 
208         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
209         % Reference Model Setup
210         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
211         ReferenceSetup(obj, zeta, wn, samplePeriod)
212 
213         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
214         % Error / Error_Dot Integrator
215         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
216         Integrate(obj, CONTROLLER)
217 
218         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
219         % Error / Error_Dot Differentiator
220         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
221         Differentiate(obj)
222 
223         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
224         % Append ZERO to end of Integrator Chain (resets Integrator)
225         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
226         ZeroIntegrators(obj)
227 
228         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
229         % Initialize Fuzzy Logic Controllers
230         % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
231         InitializeFLC(obj, params)
232     end
233 end
```

A.8.2 SUPERVISOR DRIVER

The following code is the primary supervisor abstraction layer. It calls all necessary supervisory actions when called.

```
1 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Primary Supervisor Driver
3 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function super(obj, CONTROLLER, alpha, alphaDot, ...
5                 alphaCommand, sampleTime, elevator)
6 obj.samplePeriod.Push(sampleTime);
7
8 if obj.windtunnel == 0
9     obj.samplePeriod.Push(obj.samplePeriod.last + 0.02);
10 end
11
12 if obj.alphaCommandHistory.last ~= obj.alphaCommandHistory.previous
13     obj.referenceCommand = ...
14                 obj.alphaCommandHistory.last...
15                 -obj.alphaCommandHistory.previous;
16     obj.lastCommand = obj.sysRefDTY(:, end);
17     obj.commandTime = obj.samplePeriod.last;
18 end
19
20 % Store Histories
21 obj.StoreHistories(alpha, alphaDot, alphaCommand, elevator);
22
23 % Calculate Reference Model
24 obj.ReferenceCalculation(alphaCommand);
25
26 % Calculate Alpha(Dot) Error
27 obj.CalculateError(alpha, alphaDot);
28
29 % Differentiate Alpha(Dot) Error
30 obj.Differentiate();
31
32 % Integrate Alpha(Dot) Error
33 obj.Integrate(CONTROLLER);
34
35 % Detect New Input Command
36 obj.NewCommand();
37
38 % Steady State Detection
39 obj.SteadyState(CONTROLLER, alpha);
40
41 % Update Controller Output Gains
42 obj.GainUpdates(CONTROLLER, alpha, alphaDot)
43
44 end
```

A.8.3 ERROR CALCULATOR

The following code calculates and stores the current error state of the system.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Error Calculation
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function CalculateError(obj, alpha, alphaDot)
obj.alphaError.Push(obj.sysRefDTY(1, end) - alpha);
obj.alphaDotError.Push(obj.sysRefDTY(2, end) - alphaDot);
obj.alphaDotDotError.Push( obj.sysRefDTY(3, end) ...
    - obj.alphaDotDotHistory.last);
end
```

A.8.4 DIFFERENTIATOR

The following code computes and stores the discrete derivative, finite-difference, of system parameters.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Error / Error_Dot Differentiator
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Differentiate(obj)
temp = (obj.alphaError.last - obj.alphaError.previous) ...
    / obj.samplePeriod.spread;
obj.errorDifferentiator.Push(temp);

temp = (obj.alphaDotError.last - obj.alphaDotError.previous) ...
    / obj.samplePeriod.spread;
obj.errorDotDifferentiator.Push(temp);

obj.errorChange.Push(obj.alphaError.last - obj.alphaError.previous);
end
```

A.8.5 GAIN UPDATER

The following code computes the output gains distributed to the controller output scaling gains.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller Output Gain Updates
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function GainUpdates(obj, CONTROLLER, ~, ~)
if obj.disableGainSense == 0
    if ((abs(obj.alphaError.last) > 5) ...
            || (abs(obj.alphaError.windowMean) > 15)) ...
            && obj.KpChangeStanddown == 0
        obj.KpChangeStanddown = 25;

        if CONTROLLER.ctrlOutput.last ...
```

```
12                  > (CONTROLLER.elevLimit + CONTROLLER.upperOffset) * 0.9 ...
13                  && CONTROLLER.ctrlOutput.last ...
14                  < (-CONTROLLER.elevLimit + CONTROLLER.lowerOffset) * 0.9
15
16              obj.Kp.outputGain.Push(obj.Kp.outputGain.last * .95);
17              fprintf('Decreasing Supervisor Output Kp Gains to %1.4f\n', ...
18                  obj.Kp.outputGain.last);
19          elseif CONTROLLER.ctrlOutput.last ...
20                  < (CONTROLLER.elevLimit + CONTROLLER.upperOffset) * 0.9 ...
21                  && CONTROLLER.ctrlOutput.last ...
22                  > (-CONTROLLER.elevLimit + CONTROLLER.lowerOffset) * 0.9
23              obj.Kp.outputGain.Push(obj.Kp.outputGain.last * 1.05);
24              fprintf('Increasing Supervisor Output Kp Gains to %1.4f\n', ...
25                  obj.Kp.outputGain.last);
26          end
27      else
28          if obj.KpChangeStanddown ~= 0
29              obj.KpChangeStanddown = obj.KpChangeStanddown - 1;
30          end
31      end
32
33      if ((abs(obj.errorIntegrator.last) > 25) ...
34              || (abs(obj.errorIntegrator.windowMean) > 15)) ...
35              && obj.KiChangeStanddown == 0
36          obj.KiChangeStanddown = 50;
37
38          if CONTROLLER.ctrlOutput.last ...
39                  > (CONTROLLER.elevLimit + CONTROLLER.upperOffset) * 0.9 ...
40                  || CONTROLLER.ctrlOutput.last ...
41                  < (-CONTROLLER.elevLimit + CONTROLLER.lowerOffset) * 0.9
42
43              obj.Ki.outputGain.Push(obj.Ki.outputGain.last * .95);
44              fprintf('Decreasing Supervisor Output Ki Gains to %1.4f\n', ...
45                  obj.Ki.outputGain.last);
46          elseif CONTROLLER.ctrlOutput.last ...
47                  < (CONTROLLER.elevLimit + CONTROLLER.upperOffset) * 0.9 ...
48                  || CONTROLLER.ctrlOutput.last ...
49                  > (-CONTROLLER.elevLimit + CONTROLLER.lowerOffset) * 0.9
50              obj.Ki.outputGain.Push(obj.Ki.outputGain.last * 1.01);
51              fprintf('Increasing Supervisor Output Ki Gains to %1.4f\n', ...
52                  obj.Ki.outputGain.last);
53          end
54      else
55          if obj.KiChangeStanddown ~= 0
56              obj.KiChangeStanddown = obj.KiChangeStanddown - 1;
57          end
58      end
59
60      if (abs(obj.alphaDotError.average) > 5 ...
61              || abs(obj.alphaDotError.windowMean) > 15) ...
62              && obj.changeStanddown == 0
63          obj.changeStanddown = 25;
```

```
64
65          if CONTROLLER.ctrlOutput.last ...
66                  > (CONTROLLER.elevLimit + CONTROLLER.upperOffset) * 0.9 ...
67                  && CONTROLLER.ctrlOutput.last ...
68                  < (-CONTROLLER.elevLimit + CONTROLLER.lowerOffset) * 0.9
69
70              obj.Kd.outputGain.Push(obj.Kd.outputGain.last * .95);
71              fprintf('Decreasing Supervisor Output Kd Gains to %1.4f\n', ...
72                  obj.Kd.outputGain.last);
73          elseif CONTROLLER.ctrlOutput.last ...
74                  < (CONTROLLER.elevLimit + CONTROLLER.upperOffset) * 0.9 ...
75                  && CONTROLLER.ctrlOutput.last ...
76                  > (-CONTROLLER.elevLimit + CONTROLLER.lowerOffset) * 0.9
77              obj.Kd.outputGain.Push(obj.Kd.outputGain.last * 1.05);
78              fprintf('Increasing Supervisor Output Kd Gains to %1.4f\n', ...
79                  obj.Kd.outputGain.last);
80          end
81      else
82          if obj.changeStanddown ~= 0
83              obj.changeStanddown = obj.changeStanddown - 1;
84          end
85      end
86  end
87
88  CONTROLLER.Kp.Push( ...
89      abs(obj.Kp.evalFLC( ...
90      obj.alphaError.last, ...
91      obj.errorChange.last, CONTROLLER, obj.alphaHistory.change, ...
92      obj.elevHistory.change, obj.alphaHistory.last)));
93
94  CONTROLLER.Ki.Push( ...
95      abs(obj.Ki.evalFLC( ...
96      obj.alphaError.last, ...
97      obj.errorChange.last, CONTROLLER)));
98
99  CONTROLLER.Kd.Push( ...
100     abs(obj.Kd.evalFLC( ...
101     obj.alphaError.last, ...
102     obj.errorChange.last, CONTROLLER, obj.alphaDotHistory.change, ...
103     obj.elevDotHistory.change, obj.alphaHistory.last)));
104 end
```

A.8.6 FUZZY LOGIC INITIALIZER

The following code initializes the fuzzy logic systems embedded in the supervisor.

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Initialize Fuzzy Logic Controllers
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function InitializeFLC(obj, params)
5 params_Kp = params(1);
6 params_Ki = params(2);
```

```
7  params_Kd = params(3);
8
9  NB = 1;
10 NM = 2;
11 NS = 3;
12 ZO = 4;
13 PS = 5;
14 PM = 6;
15 PB = 7;
16 temp1 = cell(7,1);
17 temp1(:) = {1:7};
18 temp1 = cell2mat(temp1);
19
20 temp2 = temp1';
21
22 %% Kd Options
23 KdRule = [      % Original
24       PS NM NB NB NB NM PS  ;
25       PS NS NB NM NM NS PS  ;
26       ZO NS NM NM NS NS ZO  ;
27       ZO NS NS NS NS NS ZO  ;
28       ZO NS ZO ZO ZO ZO PS  ;
29       PB NS PS PS PS PS PB  ;
30       PB PM PM PM PS PS PM
31       ];
32
33 %% Kp Options
34 KpRule = [        % Original
35       ZO   PS   PM   PB   PM   PS   ZO   ;
36       PS   PS   PM   PB   PM   PS   PS   ;
37       PM   PM   PM   PB   PM   PM   PM   ;
38       PB   PB   PB   PB   PB   PB   PB   ;
39       PM   PM   PM   PB   PM   PM   PM   ;
40       PS   PS   PM   PB   PM   PS   PS   ;
41       ZO   PS   PM   PB   PM   PS   ZO
42       ];
43
44 %% DO NOT CHANGE THESE RULES
45 KiRule = [
46       PB   PB   PB   PB   PB   PB   PB   ;
47       PB   PM   PM   PB   PM   PM   PB   ;
48       PB   PM   PS   PB   PS   PM   PB   ;
49       PB   PB   PB   PB   PB   PB   PB   ;
50       PB   PM   PS   PB   PS   PM   PB   ;
51       PB   PM   PM   PB   PM   PM   PB   ;
52       PB   PB   PB   PB   PB   PB   PB
53       ];
54
55 KpRule = [temp1(:) temp2(:) KpRule(:) ones(49,2)];
56 KiRule = [temp1(:) temp2(:) KiRule(:) ones(49,2)];
57 KdRule = [temp1(:) temp2(:) KdRule(:) ones(49,2)];
58
```

```
59 obj.Kp = FIS_2X1(...
60     'e',  params_Kp.inputRange1, params_Kp.MFs, ...
61     params_Kp.inputType, params_Kp.overlap,...
62     'ec', params_Kp.inputRange2, params_Kp.MFs, ...
63     params_Kp.inputType, params_Kp.overlap,...
64     'u',  params_Kp.outputRange, params_Kp.MFs, ...
65     params_Kp.outputType, params_Kp.overlap,...
66     'left', ...
67     params_Kp.distribution, ...
68     'Kp', ...
69     params_Kp.ENABLE, obj.disableGainSense, ...
70     'RULES', KpRule);
71
72 obj.Ki = FIS_2X1(...
73     'e',  params_Ki.inputRange1, params_Ki.MFs, ...
74     params_Ki.inputType, params_Ki.overlap,...
75     'ec', params_Ki.inputRange2, params_Ki.MFs, ...
76     params_Ki.inputType, params_Ki.overlap,...
77     'u',  params_Ki.outputRange, params_Ki.MFs, ...
78     params_Ki.outputType, params_Ki.overlap,...
79     'left', ...
80     params_Ki.distribution, ...
81     'Ki', ...
82     params_Ki.ENABLE, obj.disableGainSense, ...
83     'RULES', KiRule);
84
85 obj.Kd = FIS_2X1(...
86     'e',  params_Kd.inputRange1, params_Kd.MFs, ...
87     params_Kd.inputType, params_Kd.overlap,...
88     'ec', params_Kd.inputRange2, params_Kd.MFs, ...
89     params_Kd.inputType, params_Kd.overlap,...
90     'u',  params_Kd.outputRange, params_Kd.MFs, ...
91     params_Kd.outputType, params_Kd.overlap,...
92     'left', ...
93     params_Kd.distribution, ...
94     'Kd', ...
95     params_Kd.ENABLE, obj.disableGainSense, ...
96     'RULES', KdRule);
97 end
```

A.8.7 INTEGRATOR

The following code computes and stores the discrete integral, right Riemann sum, for system parameters.

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Error / Error_Dot Integrator
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function Integrate(obj, CONTROLLER)
5 temp = obj.errorIntegrator.last ...
6     + obj.alphaError.last * obj.samplePeriod.spread;
```

```
7
8  if (CONTROLLER.ctrlOutput.last ...
9          < (CONTROLLER.elevLimit + CONTROLLER.upperOffset)*0.9)
10     if (CONTROLLER.ctrlOutput.last ...
11             > (-CONTROLLER.elevLimit + CONTROLLER.lowerOffset)*0.9)
12         obj.errorIntegrator.Push(temp);
13     end
14 else
15     fprintf('Restricting integrator\n');
16     if (temp <= obj.errorIntegrator.last)
17         obj.errorIntegrator.Push(temp);
18     else
19         obj.errorIntegrator.Push(obj.errorIntegrator.last);
20     end
21 end
22
23 temp = obj.errorDotIntegrator.last ...
24     + obj.alphaDotError.last * obj.samplePeriod.spread;
25 obj.errorDotIntegrator.Push(temp);
26
27 temp = obj.errorDotDotIntegrator.last ...
28     + obj.alphaDotDotError.last * obj.samplePeriod.spread;
29 obj.errorDotDotIntegrator.Push(temp);
30
31 obj.errorSum.Push(obj.alphaError.last + obj.alphaError.previous);
32 end
```

A.8.8 NEW INPUT COMMAND DETECTION

The following code detects new input commands regardless of the filter state.

```
1  % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % New Command Detection
3  % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  function NewCommand(obj)
5  if obj.alphaCommandHistory.previous ~= ...
6          obj.alphaCommandHistory.last;
7      fprintf('New Command (Alpha-> %1.2f) Detected at Time: %1.2f\n', ...
8          obj.alphaCommandHistory.last, obj.samplePeriod.last);
9
10     % Reset Appropriate Data Structures
11     obj.ssTrigger = 0;
12     obj.decreaseTrigger = 0;
13
14     obj.changeStanddown = 150;
15     obj.KiChangeStanddown = 75;
16 end
17 end
```

A.8.9 REFERENCE COMMAND FILTER

The following code filters the input command. It is useful for mitigating large derivative actions as well as establishing a performance tracking goal.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reference Calculation
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ReferenceCalculation(obj, alphaCommand)
obj.sysRefDTX(:, end+1) = ...
    obj.sysRefDT.a * obj.sysRefDTX(:, end) ...
    + obj.sysRefDT.b * alphaCommand;

obj.sysRefDTY(:, end+1) = ...
    obj.sysRefDT.c * obj.sysRefDTX(:, end) ...
    + obj.sysRefDT.d * alphaCommand;
end
```

A.8.10 REFERENCE FILTER CONSTRUCTOR

The following code initializes the reference filter.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reference Model Setup
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ReferenceSetup(obj, zeta, wn, samplePeriod)
WN   = wn; % deg/s

sys1 = tf([WN^2],      [1 2*zeta*WN WN^2]); %#ok
sys2 = tf([WN^2 0],    [1 2*zeta*WN WN^2]);
sys3 = tf([WN^2 0 0],  [1 2*zeta*WN WN^2]);

obj.sysRefCT = ss([sys1; sys2; sys3]);
obj.sysRefCT.StateName = {'Alpha' 'AlphaDot'};
obj.sysRefCT.StateUnit = {'deg' 'deg/s'};
obj.sysRefCT.InputName = 'Alpha Command';
obj.sysRefCT.InputName = 'Alpha Com';
obj.sysRefCT.OutputName = {'Alpha' 'AlphaDot' 'AlphaDotDot'};

obj.sysRefDT = c2d(obj.sysRefCT, samplePeriod);

obj.sysRefCTX = zeros(2,1);
obj.sysRefCTY = zeros(3,1);
obj.sysRefDTX = zeros(2,1);
obj.sysRefDTY = zeros(3,1);

[obj.sysRefLUT, obj.sysRefTime, obj.sysRefLUTx] = step(obj.sysRefDT);
obj.sysRefLUT = obj.sysRefLUT.';
end
```

A.8.11 STEADY STATE DETECTION

The following code detects when the system is operating in the steady-state. Steady-state limits are set externally.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Steady State Detector
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function SteadyState(obj, CONTROLLER, alpha)
if obj.alphaCommandHistory.spread < obj.steadyStateSensitivity
    if obj.alphaDotHistory.spread < obj.steadyStateSensitivity
        if abs(obj.alphaError.last) < obj.steadyStateSensitivity
            obj.ssTrigger = 1;

            stored = CONTROLLER.setTrim(alpha);

            if stored == 1
                fprintf('\t\tSaving Trim: %1.4f to Alpha: %1.1f and Time Index: %1.2fs\n',
    ...
                    CONTROLLER.ctrlOutput.last, ...
                    round(alpha, 1, 'decimal'), ...
                    (CONTROLLER.controllerIndex + 1) ...
                    * obj.samplePeriod.spread);
            end
        end
    end
end
end
```

A.8.12 HISTORY STORAGE

The following code is a convenience function useful for storing system state parameter histories.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Store Variable Histories in Queue
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function StoreHistories(obj, alpha, alphaDot, alphaCommand, elevator)
obj.alphaHistory.Push(alpha);
obj.alphaDotHistory.Push(alphaDot);
obj.alphaCommandHistory.Push(alphaCommand);

obj.alphaDotDotHistory.Push( ...
    (obj.alphaDotHistory.previous - obj.alphaDotHistory.last) ...
    / obj.samplePeriod.spread);

obj.elevHistory.Push(elevator);
obj.elevDotHistory.Push(obj.elevHistory.change);
end
```

A.8.13 INTEGRATOR RESET

The following code is useful in resetting the integrator state, if desired. This may be necessary in conditions where integrator wind-up is detected.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Append ZERO to end of Integrator Chain (resets Integrator)
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ZeroIntegrators(obj)
obj.errorIntegrator.Reset();
obj.errorDotIntegrator.Reset();
obj.errorDotDotIntegrator.Reset();
end
```

VITA

Keith A. Benjamin
Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA 23529

Keith Benjamin was born in California, USA in 1984. He received a B.S. degree in Computer Engineering from Old Dominion University in 2016. His studies have focused on Control Theory.

He is currently a Firmware Engineer with Micron Technology where he develops and maintains Non-Volatile Dual In-Line Memory Modules (NVDIMM) for enterprise data-center applications. He is an eight year United States Navy veteran.

EDUCATION

Bachelor of Science in Computer Engineering, May 2016, Old Dominion University

PROFESSIONAL EXPERIENCE

- Micron Technology, Feb. 2018 – Present
  Firmware Engineer Primary duties include the development and maintenance of firmware for Non-Volatile Dual In-line Memory Module (NVDIMM) enterprise server application.

- Tek Fusion Global, Inc., Sep. 2016 – Feb. 2018
  Electrical Engineer Primary duties include the development and maintenance of avionics equipment for light helicopter and small fixed-wing military aircraft.

ACADEMIC HONORS

- Tau Beta Pi Engineering Honor Society

Typeset using LaTeX.