Spring 5-2022

Development of a Fuzzy Logic Model-Less Aircraft Controller

Christopher M. Scott
Old Dominion University, Cscot002@odu.edu

DEVELOPMENT OF A FUZZY LOGIC MODEL-LESS AIRCRAFT CONTROLLER

by

Christopher M. Scott
B.S. Aerospace Engineering 2008, Virginia Polytechnic and State 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
May 2022

Approved by:

Oscar R. González (Director)

W. Steven Gray (Member)

Thomas E. Alberts (Member)

ABSTRACT

DEVELOPMENT OF A FUZZY LOGIC MODEL-LESS AIRCRAFT CONTROLLER

Christopher M. Scott
Old Dominion University, 2022
Director: Dr. Oscar R. González

The Modeling and Control for Agile Aircraft Development (MCAAD) group at NASA Langley Research Center(LaRC) is developing techniques for Real-Time Global Modeling (RTGM) and Robust Learning Control (RLC) for NASA's Transformational Tools and Technologies Project. This project seeks to develop a systematic approach to reduce the iterative nature of aircraft design by introducing a model-less control law and enabling in-flight aerodynamic modeling and controller design. The development of the flight control system without prior knowledge of the aircraft aerodynamic model makes use of Takagi-Sugeno-Kang fuzzy logic inference systems for pitch and roll controllers and are tested in various simulations and wind tunnel platforms. These fuzzy logic controllers are not based on a mathematical model but rather on a rule base of generic flight control laws generated from the designer's knowledge of aircraft flight mechanics. The controller architecture uses two channels to provide absolute and incremental controller commands as needed. The absolute channel is designed to reject disturbances and decrease rise time, while the incremental channel provides tracking and reduced steady state error. To provide controllers with acceptable performance without the need for tuning, a general method for selecting input and output scaling gains for the fuzzy inference systems is proposed. A performance and robustness comparison of similarly configured Type-1 and Interval Type-2 fuzzy logic controllers is made. The fuzzy logic controllers were implemented on an aircraft model in the NASA Langley 12-Foot low speed tunnel mounted on a free-to-pitch and free-to-roll rig. The development of the controller architectures and wind tunnel results are presented.

ACKNOWLEDGEMENTS

This work was partially supported by the NASA Langley Research Center under Cooperative Agreement NNL09AA00A (NIA Activity Number 201017-ODURF). The author would like to thank E. Heim, S. Riddick, D. Hatke, R. Weinstein, E. Viken, E. Harris and the rest of the NASA Langley Research Center 12-ft Wind Tunnel team, whose support made the controller development and testing possible.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Aircraft flight control design typically requires a mathematical aerodynamic model of the aircraft's dynamics, developed through extensive wind tunnel and flight testing as detailed by Morelli [1] [2]. This iterative approach is time consuming and costly, traditionally consisting of cyclical ground based aerodynamic modeling and flight control law development prior to in-flight testing. NASA's Modeling and Control for Agile Aircraft Development (MCAAD) group, within the Transformational Tools and Technologies (TTT) Project, is tasked with improving this process by developing ground-based and in-flight testing technologies to enable self-learning flight vehicles, as presented by Riddick [3]. Specifically, the ability to insert in-flight aerodynamic modeling more quickly into the aircraft design process allows control structures and control laws to be modified earlier and reduces the timeline of the overall aircraft development process.

The MCAAD group has focused on two technology improvements in the areas of real-time global modeling (RTGM) and Robust Learning Control (RLC). Traditional control design techniques are not effective to provide in flight stability without *a priori* knowledge of the aircraft's aerodynamics and can only be implemented after the aerodynamic model has been found within the tested flight envelope. The application of RTGM and RLC allows aircraft aerodynamic modeling to take place in-flight.

The RLC technology consists of model-less flight control law development and traditional model based control law implementation within the tested flight envelope. This approach develops an aircraft flight control law without *a priori* knowledge of the aircraft aerodynamic model to maintain stability throughout the RTGM learning process before transitioning to a learned aerodynamic model based control law. This approach allows aircraft aerodynamic modeling to take place in-flight, while the aircraft is stabilized using the model-less based aircraft control law. Additionally, the model-less control law may be implemented when the aircraft enters an unknown region of operation for a model-based controller. By using the model-less control law, the envelope of aerodynamic modeling could be expanded for the aircraft under test (AUT).

1.2 PROBLEM STATEMENT

The task of this thesis project is to develop model-less based flight control laws for free-to-pitch and free-to-roll configurations of an unknown aircraft model, and test the control design in the NASA Langley 12ft Low Speed Wind Tunnel. The controllers are to maintain stability of the AUT throughout the RTGM learning process, while a nonlinear model of the AUT and a linearized model for each degree of freedom is calculated for regions in the flight envelope. Once the RTGM process is complete, the model-less control law should transition to a proportional-derivative (PD) control law based on the learned nonlinear aerodynamics and calculated linear models.

Previous iterations of this project implemented a Fuzzy Logic proportional-integral-derivative (PID) controller, where each channel gain was scheduled with a unique Type-1 Fuzzy Inference System (FIS). These Type-1 FIS were designed to account for gain selection uncertainties, as the aircraft model was unknown and traditional PID tuning was not possible. Results of this approach are detailed by Benjamin [4]. While PID gain tuning was not required, the Type-1 input and output scaling gains were subject to tuning. Removing the aircraft specific tuning of the controller parameters should be addressed in the proposed control solution.

Constraints for controller development include no *a priori* knowledge of the AUT aerodynamic model, and no additional in-flight tuning of the control law. However, the control design may take into consideration known physical attributes of the AUT, such as actuator limitations, general knowledge of the control surfaces, such as lift direction, and basic understanding of aircraft flight mechanics and control. The proposed control structures are presented in the following section.

1.3 PROPOSED SOLUTION

This thesis investigates the design and implementation of Type-1 and Interval Type-2 (IT-2) fuzzy inference systems for an aircraft's pitch and roll controller architectures. A comparison of the performance and robustness characteristics between two similarly configured Type-1 and IT-2 fuzzy logic controllers for aircraft in free-to-pitch and free-to-roll configurations is presented. In many applications, Type-2 fuzzy logic controllers outperform similar Type-1 fuzzy logic controllers. For instance, in self-driving cars, it has been shown that Type-2 fuzzy logic controllers exhibit smoother trajectory tracking than similarly configured Type-1 controllers. Type-2 controllers may also require smaller rule base to achieve

similar performance and provide a smoother output in the presence of noisy input signals. These potential improvements are detailed extensively by Mendel [5] and may provide additional robustness to the unknown characteristics of the AUT. However, the benefits of the Type-2 controller must be weighed against increased computation time as compared to a similarly configured Type-1 controller. While the benefits of a Type-2 FIS are not guaranteed for an aircraft control system, they provide sufficient motivation to implement and compare to a similarly configured Type-1 FIS.

The proposed controller configuration consists of a two channel FIS architecture of absolute and incremental outputs for pitch control and a single channel FIS architecture of absolute output for roll control. The goal of this controller design is to maintain aircraft stability and provide adequate performance while the Real Time Global Modeling (RTGM) algorithm collects enough initial flight data to generate a global nonlinear aerodynamic model. Once the initial aerodynamic model is developed, flight control is switched to a model-based classically designed controller. The controller uses local linear models which approximate the global nonlinear model around the current states of the test aircraft and vary throughout the test envelope.

In many cases, inputs and outputs of a fuzzy inference systems must be scaled within the universe of discourse or domain and range of the function. These scaling values act as tunable parameters, which may vary significantly between types of aircraft. Due to the limited *a priori* knowledge of the vehicles under test, a general method is also presented to select the input and output signals scaling gains of the fuzzy inference system for a specific aircraft. This approach minimizes the tuning requirements of the model-less fuzzy logic controllers and is applied to both Type-1 and IT-2 controller configurations.

Simulation and wind tunnel test results of these controller designs are conducted to demonstrate the performance differences between these Type-1 and Type-2 controller configurations and the results are presented. The final configuration of the IT-2 fuzzy logic pitch controller is used to maintain stability of the AUT throughout the RTGM learning process, and the results of this wind tunnel test is also presented.

1.4 THESIS OUTLINE

The thesis is organized as follows: Chapter 2 presents a basic overview of Type-1 and Type-2 fuzzy logic inference systems. Chapter 3 provides a detailed description of the proposed Type-1 and IT-2 fuzzy inference systems for aircraft pitch and roll control, as well as the methodology of scaling gain selection. Chapter 4 provides wind tunnel experimental results to examine the controllers performance and is followed by concluding remarks in Chapter 5.

CHAPTER 2

REVIEW OF FUZZY INFERENCE SYSTEMS

Conventional control system design typically follows a standard procedure. A dynamic model for the system is defined by a set of differential equations, state equations, or a transfer function and a type of conventional controller design approach is chosen, such as a lag-lead compensator, PID control, or state feedback control. The controller is then designed to meet specific requirements, such as rise time, settling time, overshoot, steady state error, or gain and phase margins. Lastly, the connected controller-system is tested in a simulation to validate these performance specifications are met. These techniques provide a variety of tools to develop good control solutions based on information from a known or estimated mathematical model of the system. If a mathematical model for the system is unknown or poorly estimated, but heuristic expert knowledge of the system exists, then a fuzzy logic controller design approach may be considered.

The two most widely used types of fuzzy inference systems are the Mamdani FIS, developed by Mamdani and Assilian [5], and the Takagi-Sugeno (TS) or Takagi-Sugeno-Kang (TSK) FIS, developed by Takagi, Sugeno and Kang [6][7] . The Mamdani FIS is structured such that the input antecedents and the outputs consequents are both described in linguistic terms. For an aircraft control example, if the pitch error is described as positive and large, and the change in pitch error is described as negative and small, then the output consequent of the FIS rule is described as a negative and large deflection of the aircraft's elevator control surface. The TS and TSK FIS are structured such that the input antecedents are in linguistic terms but the output consequents of individual rules are mapped as a mathematical function of the input terms or as a constant value. For example, if the pitch error is described as positive and large, and the change in pitch error is described as negative and small, then the output consequent of the FIS rule can be defined by the constant function $u = -c_x$, where c_x is the consequent constant of the x^{th} fired rule.

Only the TSK FIS structures with singleton consequent membership functions will be discussed further, as this is the fuzzy inference system architecture developed in the two fuzzy logic controllers designed within this thesis.

2.1 TYPE-1 FUZZY INFERENCE SYSTEMS

A Type-1 fuzzy inference systems consist of four main components: Fuzzifier, Rule Base or Expert Knowledge Base, Inference System, and the Defuzzifier. The interconnection diagram of these components is shown in Fig. 1.

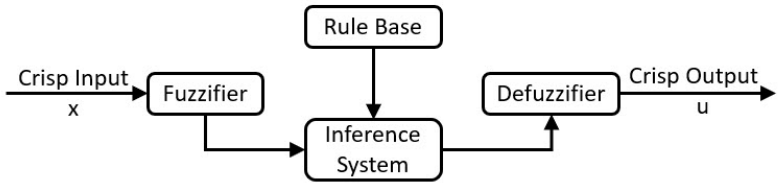

Fig. 1: Type-1 Fuzzy Inference System.

CRISP INPUTS

A crisp input to the FIS is a finite value, such as a sensor reading of pitch angle, roll rate, speed or altitude of an aircraft. The input typically defines a state or states of the plant, or has some heuristic correlation to a control input to the plant. In the case of aircraft attitude control, the pitch angle error may be corrected by changing the angle of the aircraft elevators, and this heuristic input-output relationship can be used to define an input to the FIS.

Input scaling factors normalize the crisp input signals to the universe of discourse, or the range of valid input values, before the inputs are sent to the Fuzzifier. This allows a standard universe of discourse to be used for all FIS, such as [-1,1], and only requires changing these scaling factors to modify the controller input ranges. Selection of these scaling factors are detailed in Section 3.4. After normalization, inputs to the fuzzy logic aircraft controllers within this thesis are defined by normalized error, $\hat{e}$, and normalized change in error, $\Delta\hat{e}$.

INPUT FUZZY SETS AND THE FUZZIFIER

The Fuzzifier determines which antecedent membership function(s) the crisp inputs belong to, and assigns a corresponding strength of membership, μ, for each function. In this way, the Fuzzifier maps crisp inputs into the antecedent fuzzy sets, as described further by Passino [8]. Normalized input signal values are evaluated for each membership function, and those with nonzero strength of membership are elements of the antecedent fuzzy set.

The shape and distribution of the antecedent membership functions within the fuzzy set are determined by the designer. In this thesis, interior membership functions are triangle functions within a universe of discourse of [-1,1], and are bookended with a Z function and an S function. The Type-1 input membership functions used for this thesis are shown in Figs. 2 and 3.

The selection of this particular set of membership functions are used to reduce the complexity of design, as the unknown aircraft dynamics do not provide additional information where more complex membership functions could be implemented. These membership functions are defined by Eqs. (1) - (3) and evaluated with the MATLAB Fuzzy Logic Toolbox [9]. The definitions for the membership functions are given next.

The Triangle Membership Function is defined as

$$f(x) = \begin{cases} (x-a)/(b-a), & \text{if } x \text{ is} < \text{b (Left Slope)} \\ 1, & \text{if } x \text{ is} = \text{b} \\ (c-x)/(c-b), & \text{if } x \text{ is} > \text{b (Right Slope)} \end{cases} \tag{1}$$

where, [a,b,c] are defined as the left break point, center break point and right break point, respectively.

The Z Membership Function is defined as

$$f(x) = \begin{cases} 1, & \text{if } x \leq \text{a} \\ 1 - 2 * ((x-a)/(a-b))^2, & \text{if } \text{a} < x \leq \text{(a+b)/2} \\ 2 * ((b-x)/(a-b))^2, & \text{if } \text{(a+b)/2} < x < \text{b} \\ 0, & \text{if } x \geq \text{b} \end{cases} \tag{2}$$

where, [a,b] are defined as the upper and lower break points, respectively.

The S Membership Function is defined as

$$f(x) = \begin{cases} 0, & \text{if } x \leq \text{a} \\ 2 * ((a-x)/(b-a))^2, & \text{if } \text{a} < x \leq (\text{a+b})/2 \\ 1 - 2 * ((x-b)/(b-a))^2, & \text{if } (\text{a+b})/2 < x < \text{b} \\ 1, & \text{if } x \geq \text{b} \end{cases} \tag{3}$$

where, [a,b] are defined as the lower and upper break points, respectively.

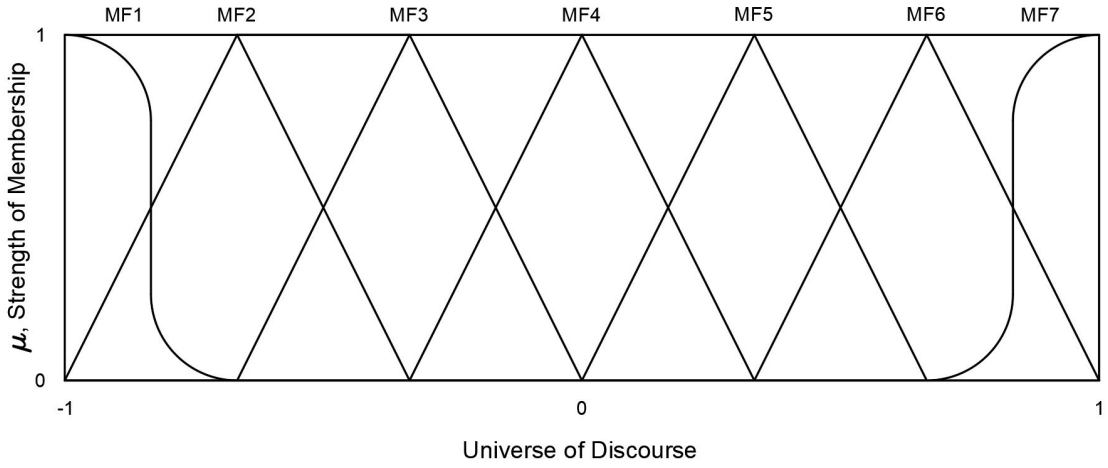

Fig. 2: Type-1 Input Membership Function Set for Free-to-Pitch Control.

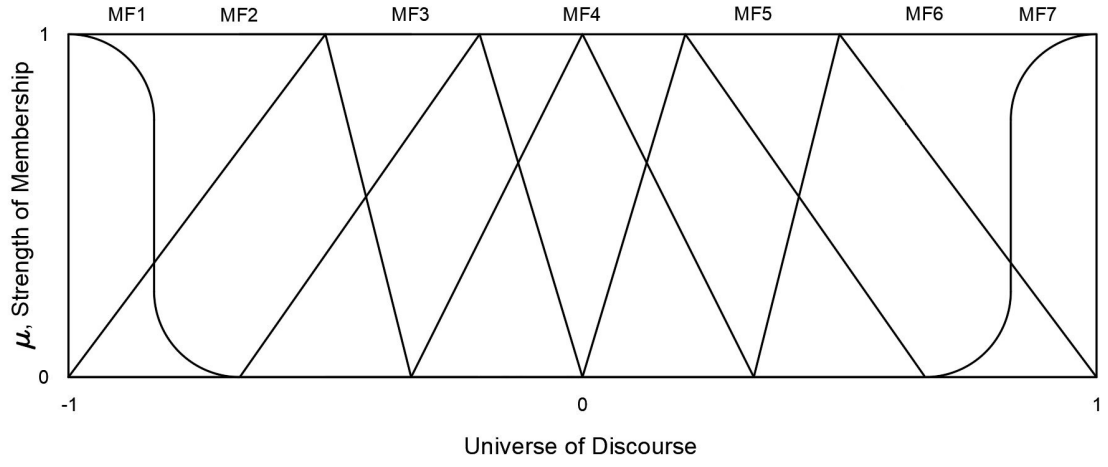

Fig. 3: Type-1 Input Membership Function Set for Free-to-Roll Control.

RULE BASE INFERENCE SYSTEM AND CONSEQUENT FUZZY SETS

The Rule Base is the set of if-then rules, which map the antecedent membership functions to the consequent membership functions. The Rule Base is determined by the controller designer, and may be based on "Expert Knowledge" of the process to be controlled, such as the AUT. The inference system maps the antecedent functions (1) - (3) with non-zero strength of membership to their corresponding consequent functions, from the Rule Base, to create the fuzzy set of all fired rules.

For a TSK FIS, these input membership functions represent linguistic terms, such as "Small, Medium, Large", and the mapped output functions are singleton values. In a Type-1 TSK FIS, the consequent membership function is defined by the function $u = c_x$, where c_x is the consequent constant of the x^{th} fired rule. For example, if the input membership functions within Fig. 2 are considered, and the output membership functions are singletons corresponding to the amount of normalized elevator deflection of an aircraft, then a single rule in the Rule Base may read as follows:

If $\hat{e}$ *is Negative Large*$(MF1)$ **And** $\Delta\hat{e}$ *is Negative Small*$(MF3)$ **Then** $u = c_1$.

The consequent fuzzy set of all fired rules is sent to the Defuzzifier, as described in the next section.

THE DEFUZZIFIER AND CRISP OUTPUTS

The final step is for the Defuzzifier to map the consequent fuzzy set from the inference engine to a crisp output. The crisp outputs of a fuzzy inference system are the control input to the controlled process, or in the case of this thesis the AUT. Many types of deffuzification methods exists for Type-1 fuzzy inference systems, however one of the most common is the center-average method. The center-average function,

$$Y = \frac{\sum_{i=1}^{R} b_i \mu_i}{\sum_{i=1}^{R} \mu_i}, \tag{4}$$

is used for deffuzification in the Type-1 TSK FIS architecture for the proposed controllers in Chapter 3.

In the center-average method, the consequent fuzzy set functions are weighted by their corresponding strength of membership, μ_i, and the output is normalized by the overall

strength of membership within the set. The output of this equation is the normalized crisp output of the fuzzy inference system. The defuzzification of all weighted consequent functions is output of the Eq. (4). Fig. 4 is a simplified representation of the mapping from antecedent membership functions to consequent membership functions.

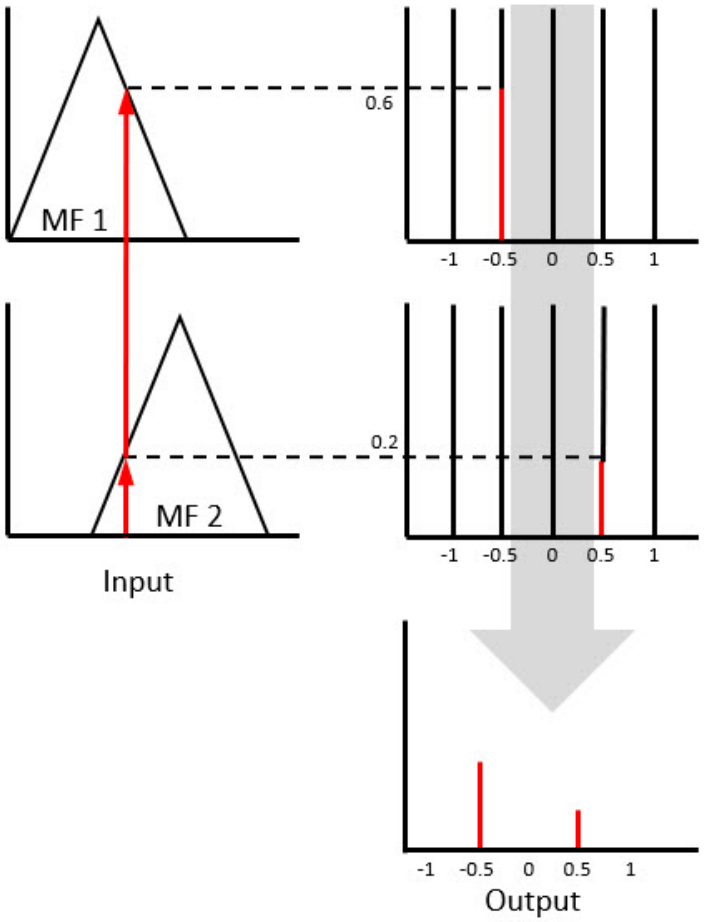

Fig. 4: Type-1 Defuzzification with Two Inputs.

2.2 TYPE-2 FUZZY INFERENCE SYSTEMS

This subsection will describe a Type-2 fuzzy inference system, and identify the differences from the previously defined Type-1 FIS. For this thesis, an Interval Type-2 (IT-2) fuzzy inference system is considered due to a reduced computation time for implementation over a more computationally demanding form such as the General Type-2 FIS. This section will also discuss potential advantages and disadvantages the IT-2 FIS may have as compared

to the Type-1 FIS.

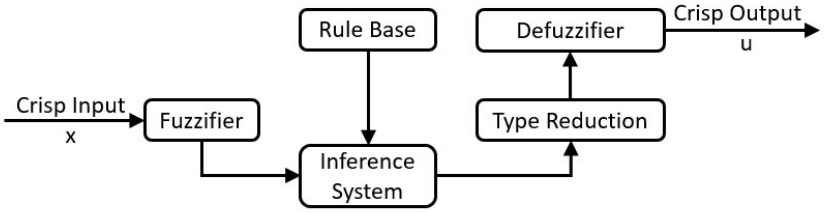

Fig. 5: Type-2 Fuzzy Inference System.

An IT-2 FIS consist of five main components: Fuzzifier, Rule Base or Expert Knowledge Base, Inference System, Type Reduction and the Defuzzifier. The flow diagram of interconnection of these components is shown in Fig. 5.

CRISP INPUTS, INPUT FUZZY SETS AND THE RULE BASE

As with a Type-1 FIS, crisp inputs are mapped to antecedent membership functions. In the case of a IT-2 FIS, each membership function consists of an upper membership and lower membership function. The area between the two is defined as the footprint of uncertainty (FOU), and represents all possible antecedent membership function shapes bounded by the upper and lower membership functions. This provides an additional degree of freedom to handle uncertainties with selection of the membership function shapes, as described by Mendel [10]. The FOU is shown as the shaded area in Figs. 6 and 7.

The inference engine for an IT-2 FIS similarly maps antecedent fuzzy sets to consequent fuzzy sets based on a Rule Base set. In many cases, a reduced Rule Base set can be used for IT-2 fuzzy logic controllers, when compared to a similarly designed Type-1 fuzzy logic controller, to achieve similar performance. This thesis uses identical Rule Base sets for comparable Type-1 and IT-2 controllers. Each fired rule within the Rule Base will contain a strength of membership value for the upper and lower antecedent functions, and will result in a Type-2 antecedent fuzzy set.

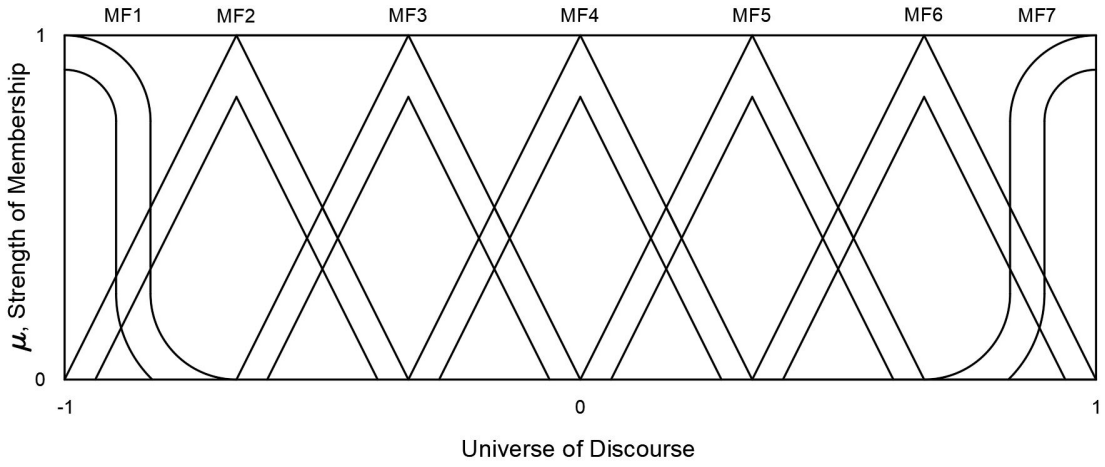

Fig. 6: Interval Type-2 Input Membership Function Set for Free-to-Pitch Control.

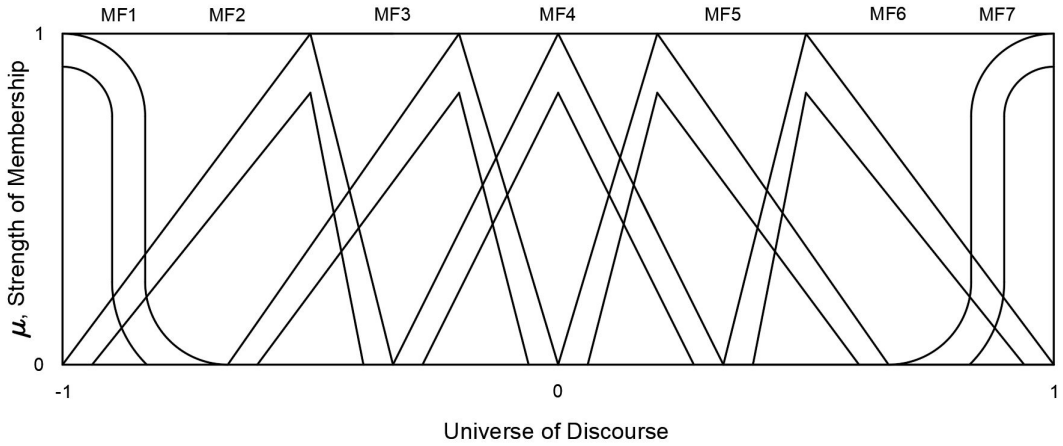

Fig. 7: IT-2 Input Membership Function Set for Free-to-Roll Control.

OUTPUT FUZZY SETS, TYPE REDUCTIONS METHODS AND DEFUZIFICATION

The IT-2 inference engine maps Type-2 antecedent fuzzy sets to Type-2 consequent fuzzy sets. To define a crisp output, the Type-2 consequent fuzzy set must first be reduced to a Type-1 consequent fuzzy set. There are many methods for this type reduction and discussion of each method is beyond the scope of this thesis. The implemented type-reduction method used for the IT-2 FIS in this thesis is the Nei-Tan (NT) method. The NT method

uses a first order approximation of the common Karnik and Mendel (KM) Method, where the vertical slices of the Type-2 consequent fuzzy set are averaged, producing a Type-1 consequent fuzzy set. This method was selected due to the fast computation requirements for implementation within the controller presented in the following chapters.

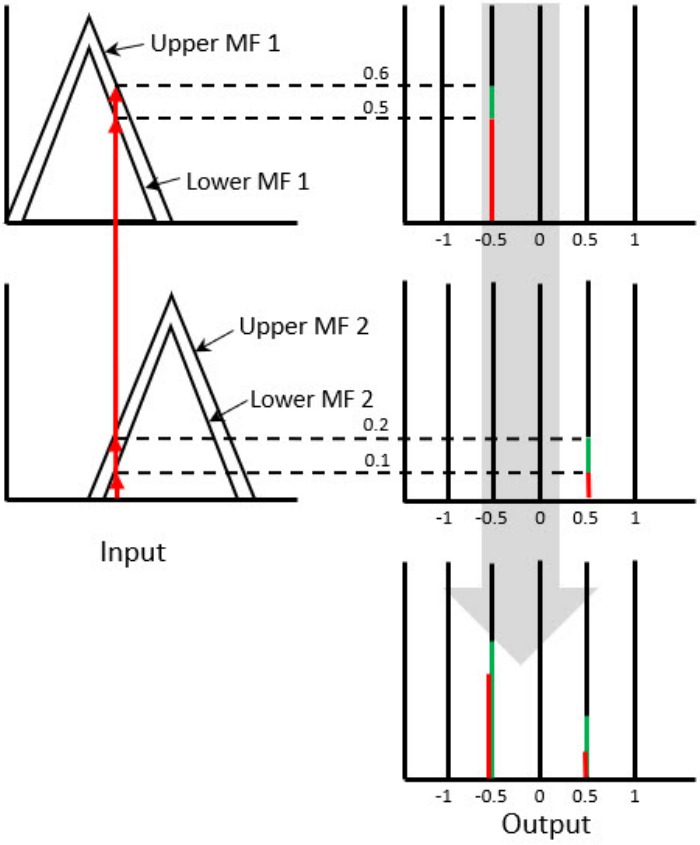

Fig. 8: Type-2 Defuzzification.

In an IT-2 TSK FIS, a consequent membership function is defined by the function $u = c_x$, where c_x is the consequent constant of the x^{th} fired rule. The defuzzification of the Type-2 consequent fuzzy set to a crisp output is produced by calculating the centroid of the Type-2 fuzzy set. This calculation is shown by Eq. (5), and pictorially shown in Fig. 8. Further details of the NT type reduction and defuzzification method are expanded by Mendel [10].

$$Y_{NT} = \frac{\sum_{i=1}^{N} u_i[\mu_{iL}(u_i|x)] + u_i[\mu_{iU}(u_i|x)]}{\sum_{i=1}^{R} \mu_{iL}(u_i|x)] + u_i[\mu_{iU}(u_i|x)} \tag{5}$$

The crisp outputs of a fuzzy inference systems are the control actions. For this thesis, the crisp outputs of the FIS are the desired elevator/aileron control surface angles of an aircraft.

2.3 AN ELECTRO-MAGNETIC BALL LEVITATION SYSTEM

The motivation for this example problem is to show that given some physical characteristics of a system, such as range in voltage of the electro-magnet and knowledge of the expected ranges for error and error rate, a fuzzy logic controller can be designed to stabilize the system without the use of a mathematical model. If these system characteristics are known or can be found experimentally, a simple fuzzy logic controller (FLC) can be designed and tested.

Suppose we have the electro-magnetic ball levitation system as provided by Kuo [11]. This system consists of an electro-magnet, which has an attractive force dependent on the current through the coil, and a metallic ball of mass m, which is free to move in the y-axis and assumed to remain stationary in the x and z axis. The electro-magnet circuit is defined by Eq. (6), consisting of an electro-magnet inductor, a resistance in the circuit, and a voltage source. The attractive force generated on the magnetic ball by the electro-magnet is defined by Eq. (7). The height of the ball is measured by a position sensor, and is assumed to be detectable through the entire range between the ground and the electro-magnet. The entire apparatus is shown in Fig. 9 below.

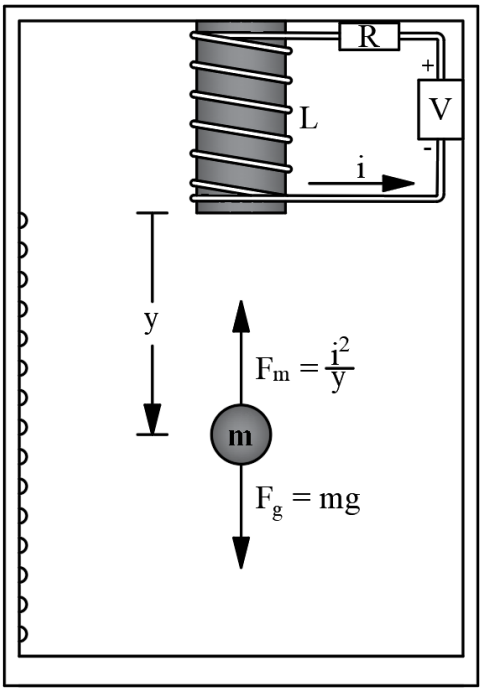

Fig. 9: An Electro-Magnetic Ball Levitation System.

The nonlinear equations of the electro-magnetic ball levitation system are defined as[1]

Electric Circuit Equation:

$$V = iR + L\frac{di}{dt}, \tag{6}$$

Magnetic Force Equation:

$$F_m = \frac{i^2}{y}, \text{ and} \tag{7}$$

Equation of Motion:

$$m\frac{d^2y}{dt^2} = mg - F_m, \tag{8}$$

where

V is the input voltage on the circuit,
i is the current in the circuit,
R is the resistance in the circuit,
L is the winding inductance,
F_m is the magnetic force applied to the ball,
y is the vertical position of the ball measured from the electro-magnet,
m is the mass of the ball, and
g is the constant of gravitational acceleration.

A nonlinear dynamic model of the electro-magnetic levitation system is found by combining Eqs. (6) - (8). This system consists of three state variables, position $x_1 = y$, position rate $x_2 = \dot{y}$, and current in the magnet coil $x_3 = i$. The derived nonlinear state equations,

$$\dot{x_1} = x_2, \tag{9}$$

$$\dot{x_2} = g - \frac{1}{m}\frac{x_3^2}{x_1}, \text{ and} \tag{10}$$

$$\dot{x_3} = -\frac{R}{L}x_3 + \frac{V}{L}, \tag{11}$$

for the electro-magnetic ball levitation system are found by combining Eqs. (6) - (8) and substituting the state variables. In order to describe the stability of the system and begin controller design, the nonlinear system described by Eqs. (9) - (11) is linearized around

[1]Nonlinear equations for the electro-magnetic ball levitation system are provided as published in [11].

an equilibrium point. The equilibrium states and input are derived from the following equations,

$$\begin{aligned} &\mathrm{x}_{10} = y_0,\ \mathrm{m}, \\ &\mathrm{x}_{20} = 0,\ \tfrac{m}{s}, \\ &\mathrm{x}_{30} = \sqrt{mgx_{10}},\ \mathrm{H}\ \text{and} \\ &\mathrm{V}_0 = Rx_{30},\ \mathrm{V}, \end{aligned} \tag{12}$$

where x_{10} is the desired position of the ball, x_{20} is the velocity of the ball, and x_{30} is the voltage required to hold the ball at the equilibrium position. A linear state-space representation of the system is defined by the linear state equation,

$$\frac{dx}{dt} = Ax + Bu, \tag{13}$$

and the linear output equation,

$$y = Cx + Du, \tag{14}$$

where A and B are the linear state matrix and linear input matrix, and C and D are the linear output matrices, respectively.

The linear state matrix terms are derived by taking the partial derivatives of the non-linear state equations with respect to each of the states,

$$A = \begin{bmatrix} \frac{d\dot{x_1}}{dx_1} & \frac{d\dot{x_1}}{dx_2} & \frac{d\dot{x_1}}{dx_3} \\ \frac{d\dot{x_2}}{dx_1} & \frac{d\dot{x_2}}{dx_2} & \frac{d\dot{x_2}}{dx_3} \\ \frac{d\dot{x_3}}{dx_1} & \frac{d\dot{x_3}}{dx_2} & \frac{d\dot{x_3}}{dx_3} \end{bmatrix}, \tag{15}$$

where partial derivatives are evaluated at the equilibrium states, x_{10}, x_{20}, x_{30}. The linearized state matrix for the electro-magnetic ball levitation system is found as

$$A = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{m}\frac{{x_{30}}^2}{{x_{10}}^2} & 0 & -2\frac{1}{m}\frac{x_{30}}{x_{10}} \\ 0 & 0 & -\frac{R}{L} \end{bmatrix}. \tag{16}$$

The linear input matrix is derived by taking the partial derivatives of the nonlinear state equations with respect to the input,

$$B = \begin{bmatrix} \frac{d\dot{x_1}}{dV} \\ \frac{d\dot{x_1}}{dV} \\ \frac{d\dot{x_1}}{dV} \end{bmatrix}, \tag{17}$$

where partial derivatives are evaluated at the equilibrium input, V_0. The linearized input matrix for the electro-magnetic ball levitation system is found as

$$B = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \end{bmatrix}. \tag{18}$$

The electro-magnetic ball levitation system has available state measurement for the position of the ball. The linear output equation matrices,

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \text{ and} \tag{19}$$

$$D = \begin{bmatrix} 0 \end{bmatrix}, \tag{20}$$

are derived based on the available state measurements and the lack of direct feed through from the input voltage.

The linearized system defined by Eqs. (16) and (18) - (20) is unstable, with two poles in the left half s-plane and one pole in the right half s-plane. This system requires a feedback controller to stabilize the system and provide desired response characteristics. In order to achieve this, two controllers will be designed and compared. The first controller is a PD controller, and is designed based on the linearized state-space model of the system. The second controller is a Type-1 fuzzy logic controller, and is designed using physical characteristics of the system and response characteristic data from simulations of the PD controller.

In order to design a PD controller around the equilibrium point, an input equation is defined as

$$V(t) = K_p e + K_d \dot{e}, \tag{21}$$

where $e = x_r - x_1$, x_r is the desired height of the ball, and x_1 is the measured height of the ball. Substituting Eq. (21) into the input, u(t), of Eq. (13), the closed loop equilibrium state and input matrices become

$$A = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{m}\frac{{x_{30}}^2}{{x_{10}}^2} & 0 & -2\frac{1}{m}\frac{x_{30}}{x_{10}} \\ -\frac{Kp}{L} & -\frac{Kd}{L} & -\frac{R}{L} \end{bmatrix} \text{ and} \tag{22}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ \frac{K_p}{L} \end{bmatrix}, \tag{23}$$

and the closed loop characteristic equation of the linearized system is found by

$$|sI - A_{CL}| = \det \begin{bmatrix} s & -1 & 0 \\ -\frac{1}{m}\frac{{x_{30}}^2}{{x_{10}}^2} & s & 2\frac{1}{m}\frac{x_{30}}{x_{10}} \\ \frac{Kp}{L} & \frac{Kd}{L} & s + \frac{R}{L} \end{bmatrix}$$

$$= s^3 + \frac{R}{L}s^2 - (\frac{1}{m}\frac{{x_{30}}^2}{{x_{10}}^2} + 2\frac{1}{mL}\frac{x_{30}}{x_{10}}K_d)s - (\frac{R}{mL}\frac{{x_{30}}^2}{{x_{10}}^2} + 2\frac{1}{mL}\frac{x_{30}}{x_{10}}K_p). \tag{24}$$

In order for the PD controller to provide stable closed loop poles, the roots of the characteristic equation must lie in the left half of the s-plane. To determine appropriate values for K_p and K_d, the Routh-Hurwitz criterion [12] is used on Eq. (24).

Table I: Routh-Hurwitz Criterion Matrix.

$$\begin{bmatrix} s^3 & 1 & -\frac{1}{m}\frac{{x_{30}}^2}{{x_{10}}^2} - 2\frac{1}{mL}\frac{x_{30}}{x_{10}}K_d \\ s^2 & \frac{R}{L} & -\frac{R}{mL}\frac{{x_{30}}^2}{{x_{10}}^2} - 2\frac{1}{mL}\frac{x_{30}}{x_{10}}K_p \\ s^1 & -2\frac{1}{mL}\frac{x_{30}}{x_{10}}K_d + 2\frac{1}{mR}\frac{x30}{x_{10}}K_p & 0 \\ s^0 & -\frac{R}{mL}\frac{{x_{30}}^2}{{x_{10}}^2} - 2\frac{1}{mL}\frac{x_{30}}{x_{10}}K_p & 0 \end{bmatrix}$$

For the closed loop system to be stable, the first column in Table I must not have any sign changes. To guarantee this criteria to be true, the following equalities must hold for the values of K_p and K_d:

$$K_p < -\frac{x_{30}}{2x_{10}}R \tag{25}$$

$$K_d < \frac{L}{R}K_p \tag{26}$$

Eqs. (25) and (26) are used to define the minimum gains of the PD controller that will guarantee stability for the linearized electro-magnetic ball levitation system around the defined equilibrium point. The following values from Section 6.4.1, pg 346 of Passino [8] are selected to describe the system for simulations and comparison with a similarly designed fuzzy logic controller.

The constants of the electro-magnetic circuit are the resistance in the circuit, R = 50 Ω, and the winding inductance, L = 0.5 H. The constants of the metallic ball dynamics are the ball mass, m = 0.1 kg, and the acceleration due to gravity, g = 9.81 $\frac{m}{s^2}$. An equilibrium height is selected, $y_0 = 0.15$ m, for which the system is linearized around, and the following equilibrium states and equilibrium input are found from Eq. (12) to get

$$\begin{aligned} x_{10} &= 0.15 \text{ m}, \\ x_{20} &= 0 \ \tfrac{m}{s}, \\ x_{30} &= 0.3836 \text{ H and} \\ V_0 &= 19.1801 \text{ V}. \end{aligned}$$

From these values, the linear state matrix for the open loop electro-magnetic ball levitation system are found from Eq. (22),

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 65.4 & 0 & -51.1468 \\ 0 & 0 & -100 \end{bmatrix},$$

and the linear input matrix is found from Eq. (23),

$$B = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}.$$

The characteristic polynomial for the open loop system is found from Eq. (24),

$$s^3 + 100s^2 - \tfrac{327}{5}s - 6540,$$

of which the poles for the open loop system are found to be 8.087, -8.087 and -100. This system is unstable, due to the single pole in the right half plane. The following PD controller gains for the closed loop electro-magnetic ball levitation system are found from Eqs. (25) and (26),

$$K_p < -8\frac{x_{30}}{2x_{10}}R = -383.6$$

$$K_d < 8\frac{L}{R}K_p = -30.7,$$

where a factor of 8 is used to move each gain further from the criterion point. The open loop phase and gain margins and the associated Nyquist plots, shown in Fig. 10 and 11, are found to determine the robustness of the controller design to deviations from the equilibrium states. Standard gain and phase margins for control design are typically greater than

a 6dB gain margin and a 45 degree phase margin. The selected K_p and K_d gains provide a gain margin of 15.6dB and a phase margin of 50.7 degrees, which are both acceptable values.

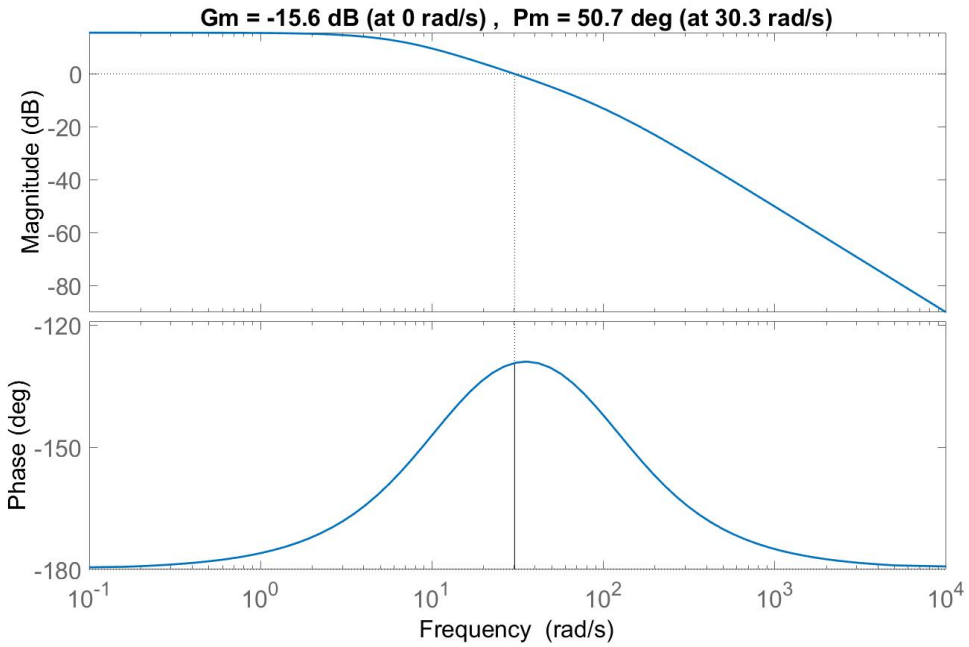

Fig. 10: Gain and Phase of a PD Controlled Electro-Magnetic Ball Levitation System.

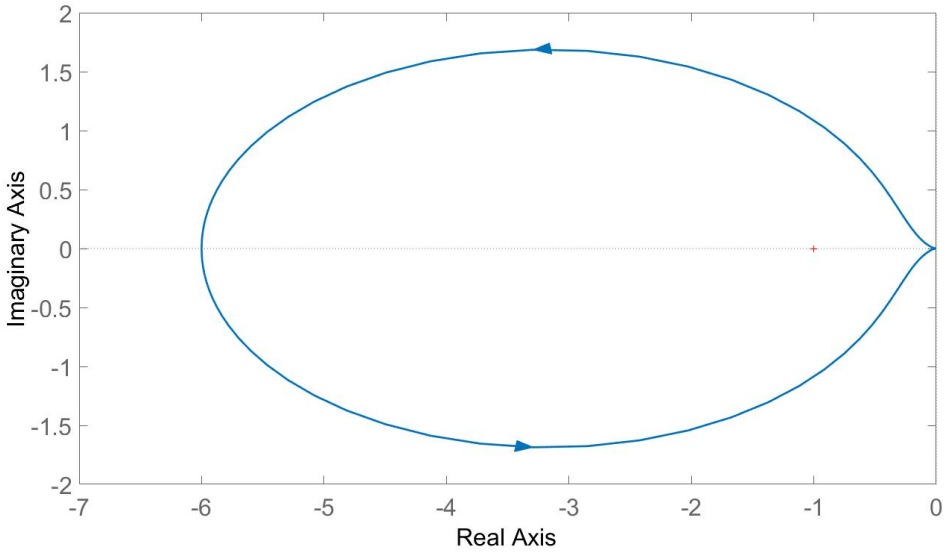

Fig. 11: Nyquist Plot of a PD Controlled Electro-Magnetic Ball Levitation System.

A simulation of the electro-magnet ball levitation system is setup. A sinusoidal input signal is provided, in addition to the equilibrium height, as the reference input to the system. This input signal is selected to test the controller robustness around the equilibrium point to a variable reference value. An error signal is generated by subtracting the reference input from the measured height of the ball at the current sample time. The PD controller is used to track the reference signal around the equilibrium point, by regulating the error of the system. The input to the voltage source of the electro-magnet circuit is the output of the PD controller in addition to the calculated equilibrium voltage, $V_0 = Rx_0$. Fig. 12 represents the closed loop PD controller and electro-magnetic ball levitation system.

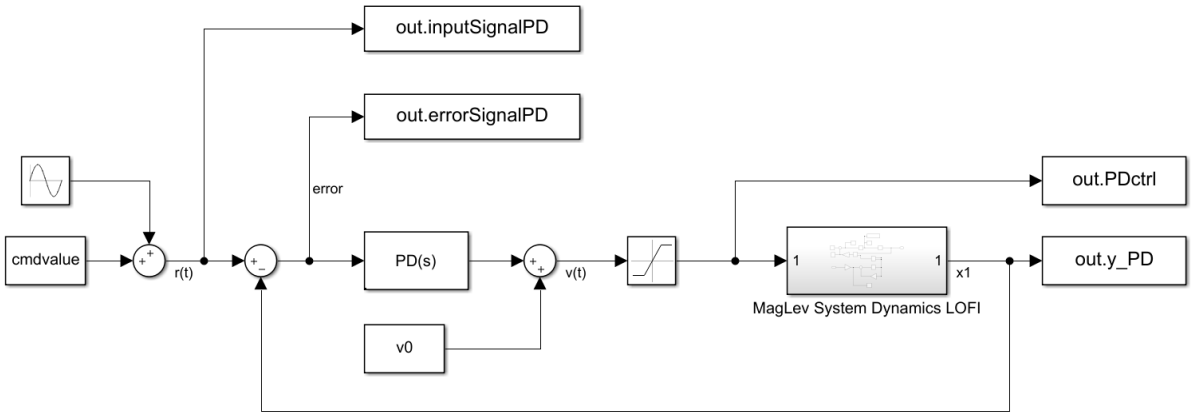

Fig. 12: PD Control Diagram of an Electro-Magnetic Ball Levitation System.

The integrators inside of the system are initialized from the equilibrium values calculated by Eq. (12), with the exception of the initial ball height. The location of the ball is initialized at 0.2 m, 0.05 m away from the selected equilibrium point of 0.15 m. The weight of the metal ball is increased by 10 percent to introduce additional uncertainty in the system dynamics model. Since the electro-magnetic ball levitation is an unstable system, the PD controller is needed to stabilize the system. The PD controller is assigned the proportional and derivative gains found from Eqs. (25) and (26). These values were specifically tuned to provide a stable system with desired response performance. The results of the PD control of the electro-magnetic ball levitation system simulation are shown in Fig. 13.

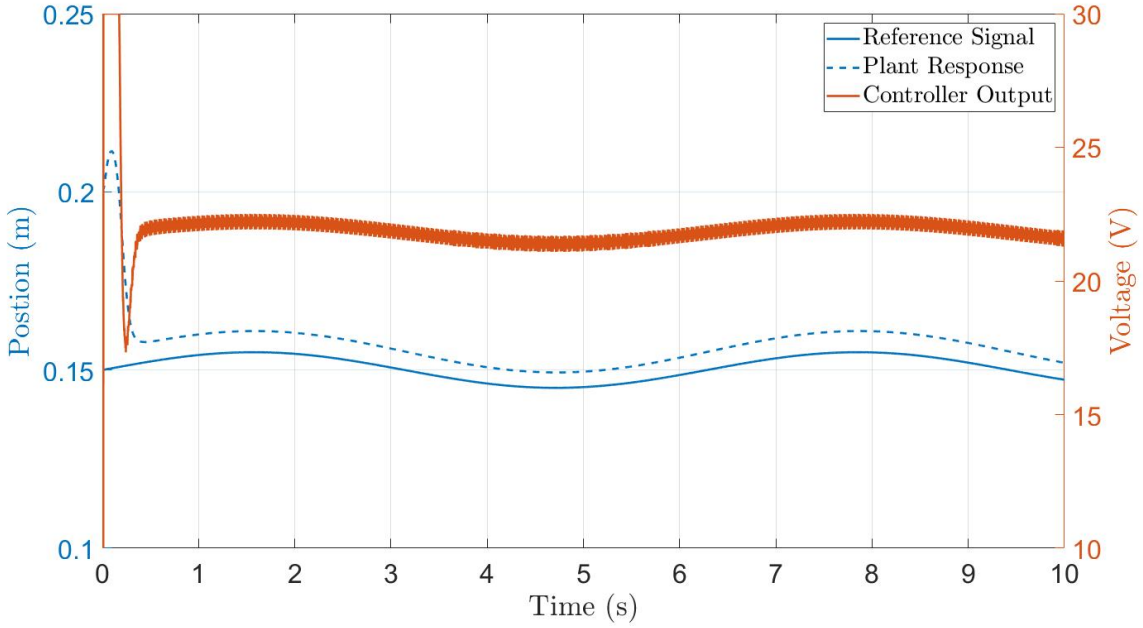

Fig. 13: PD Control of an Electro-Magnetic Ball Levitation System.

As shown in Fig. 13, the reference signal, error signal, controller output and plant outputs, in addition to the calculated error signal, are recorded. These signals are used in the development of a simple Type-1 fuzzy logic controller with a PD-like structure, to select the input and output scaling gains of the controller, K_e, K_{de} and K_o. Figure 14 represents the closed loop Type-1 fuzzy logic controller and electro-magnetic ball levitation system.

The same sinusoidal input signal is provided, in addition to the equilibrium height, as the reference input to the system. An error signal is generated by subtracting the reference input from the measured height of the ball at the current sample time, and the error rate signal is found by an estimate derivative of the error signal. The Type-1 fuzzy logic controller is used to track the reference signal around the equilibrium point, by regulating the error of the system. The input to the voltage source of the electro-magnet circuit is the crisp output of the Type-1 fuzzy logic controller in addition to the calculated equilibrium voltage, $V_0 = Rx_0$.

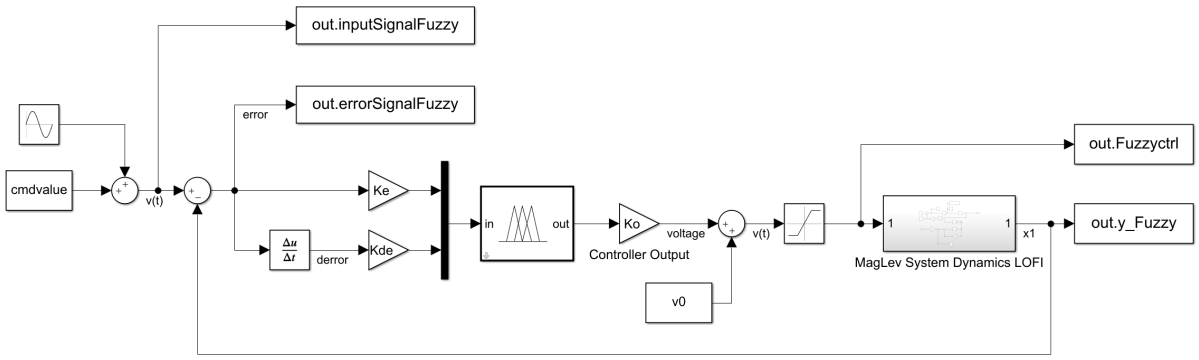

Fig. 14: Fuzzy Logic Control Diagram of an Electro-Magnetic Ball Levitation System.

The fuzzy logic Rule Base is developed by the known physical interaction between the electro-magnet force on the ball, and how the strength of the magnet should change based on the error and error rate signals. The fuzzy logic Rule Base for the electro-magnetic ball levitation system is shown in Fig. 15.

E/dE	NB	NM	NS	Z	PS	PM	PB
NB	PB	PB	PB	PB	PM	PS	Z
NM	PB	PB	PB	PM	PS	Z	NS
NS	PB	PB	PM	PS	Z	NS	NM
Z	PB	PM	PS	Z	NS	NM	NB
PS	PM	PS	Z	NS	NM	NB	NB
PM	PS	Z	NS	NM	NB	NB	NB
PB	Z	NS	NM	NB	NB	NB	NB

Fig. 15: Fuzzy Logic Rule Base for an Electro-Magnetic Ball Levitation System.

The initial location and weight of the ball remains the same as the PD controller simulation. Since the electro-magnetic ball levitation is an unstable system, the fuzzy logic controller is needed to stabilize the system. The fuzzy logic controller is assigned the input gains by taking the inverse of the maximum error and error rate values from the PD controller simulation. This approach scales the expected range of error and error rate in the simulation to a universe of discourse of [-1, 1], as designed in the Type-1 fuzzy logic controller. The output scaling gain of the fuzzy logic controller is selected as 30, which represents a maximum voltage in the circuit as 30V. This is a reasonable value, where the current in the circuit will not exceed 0.3 amps. The results of the fuzzy logic control of the electro-magnetic ball levitation system simulation are shown in Fig. 16.

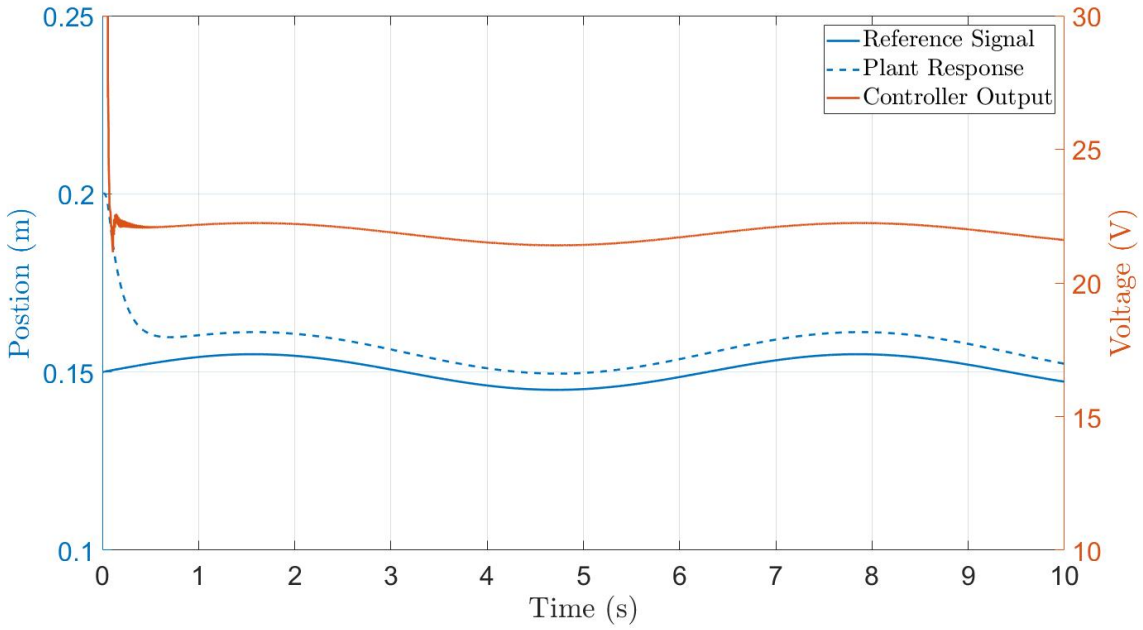

Fig. 16: Fuzzy Logic Control of an Electro-Magnetic Ball Levitation System.

2.4 CONCLUSIONS

Type-1 and IT-2 FIS controllers are considered for comparison, as IT-2 FIS has the potential to outperform similarly designed Type-1 FIS, as stated by Mendel [10]. Within an IT-2 FIS, each membership function consists of a set of type-1 fuzzy sets, which provides a wider range of outcomes. This is similar to having a "room full of expert opinions," instead of having "a single expert opinion", on a crisp output based on a crisp input. The IT-2 membership function set is a better representation of uncertainty to the linguistic meanings of each membership function shape, and may result in the IT-2 fuzzy sets having a smoother control output when Type-1 has the same number of rules. However, this increase in potential to handle uncertainties comes at the cost of increased computation time and must be taken into account during implementation. It is for these reasons that an IT-2 FIS is considered within this thesis for the controller design, as the AUT dynamic model is unknown, and the IT-2 FIS may provide additional robustness to controller design uncertainties.

As shown in the electro-magnetic ball levitation system example, control design for a well known system model can be accomplished with PD control techniques and provide robust performance. However, where there is significant uncertainty within the system model, or the model is completely unknown, a fuzzy logic controller may provide the benefit of using heuristic terms to design a controller, where expert knowledge of the AUT can be leveraged when dynamic models are not available.

The following chapter will provide a review of fuzzy logic control, and implementation of the Type-1 and IT-2 fuzzy inference systems described within this chapter for control of an aircraft in free-to-pitch and free-to-roll configurations.

CHAPTER 3

FUZZY LOGIC CONTROL REVIEW

3.1 IMPLEMENTATION OF FUZZY LOGIC FOR AIRCRAFT CONTROL

A fuzzy logic controller is developed to provide in-flight stability of the AUT, as traditional control design techniques are not effective and can only be implemented after the aerodynamic model has been found within the tested flight envelope. This FLC contains a set of flight control laws for free-to-pitch and free-to-roll configurations of an unknown aircraft model, and has the purpose of maintaining stability of the aircraft through the RTGM learning process before a transition to a learned aerodynamic model based control law can take place.

A fuzzy logic system generalizes an expert's knowledge of how an aircraft is controlled, and provides proper and adequate control actions, with heuristic and physical-meaning input-output associations. While there is no *a priori* knowledge of the aerodynamic model of the AUT, the control design takes into account known physical attributes of the AUT. Knowledge of actuator limitations, general knowledge of the control surfaces, such as lift direction, and basic understanding of aircraft flight mechanics and control can be implemented into the controller design.

The AUT is assumed to have a coordinate system such that a positive pitch angle is considered "nose up" motion in the theta axis, and a positive roll angle is considered clockwise rotation in the phi axis. The lifting surfaces have a coordinate system such that a positive angle of the lifting surface provides a force to generate a positive motion in the aircraft's pitch or roll axis. The aircraft coordinate system is shown in Fig. 17.

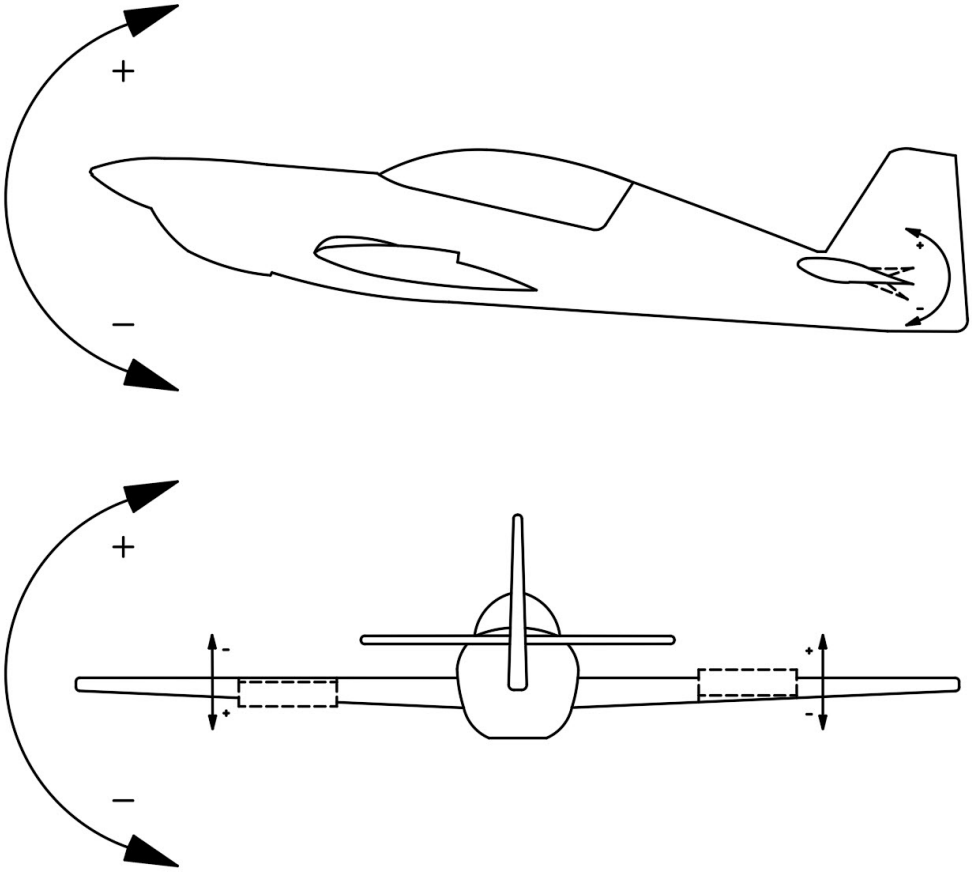

Fig. 17: Aircraft Attitude and Control Surface Coordinate System for Pitch and Roll Axis.

The proposed controllers consists of a two channel FIS architecture of absolute and incremental outputs for pitch control, and a single channel FIS architecture of absolute output for roll control. This type of control architecture was preferred to reduce the tuning requirements for the controllers, as a model for the aircraft is unknown prior to testing. Three tunable parameters are identified as the two input scaling factors and the output scaling factor for each channel, and selection of these parameters are detailed in Section 3.4. The Type-1 and IT-2 controllers were implemented in MATLAB using the Interval Type-2 Fuzzy Logic Systems Toolbox developed by Taskin [13], where the Type-1 controller uses identical upper and lower Type-2 membership functions to simulate a Type-1 membership function.

3.2 TYPE-1 AND IT-2 FUZZY PITCH CONTROLLER

The free-to-pitch FLC is based on the knowledge a pilot would use to control an aircraft in a pitching up and down movement. The fuzzy controller receives pitch angle error and change

in error signals, which would be available to a pilot through instrumentation readings. The Rule Base of the pitch controller is adapted from Napolitano's [14] Type-1 fuzzy logic system, for the Type-1 and Interval Type-2 (IT-2) pitch controllers. The Rule Base represents the pilot's heuristic knowledge of flight control in the pitching moment. Reference signals are generated by feeding the desired aircraft pitch angle, theta, through the second order reference model,

$$\theta_{ref}(s) = \frac{6.25}{s^2 + 4.25s + 6.25}\theta_{cmd}(s), \tag{27}$$

with a natural frequency, ω_n, of 2.5 rad/s, and a damping ratio, ζ, of 0.85. This reference model defines the desired closed loop characteristics of the controller and AUT. The reference model (27) is discretized at a sample rate of 50 Hz.

The fuzzy logic controller is designed to minimize the error between the closed loop system and the reference model. An error signal, e, is generated from the difference of the sampled reference signal and the sampled theta value, from the aircraft sensors, at a frequency of 50 Hz. The change in error signal, Δe, is estimated by the difference of error signals over each sample period. These two signals are the inputs into the FLC to generate a control signal which operates the aircraft elevators. The FLC attempts to track the reference signal by minimizing the error signal while maintaining reasonable pitch rates. This process is outlined in Fig. 18.

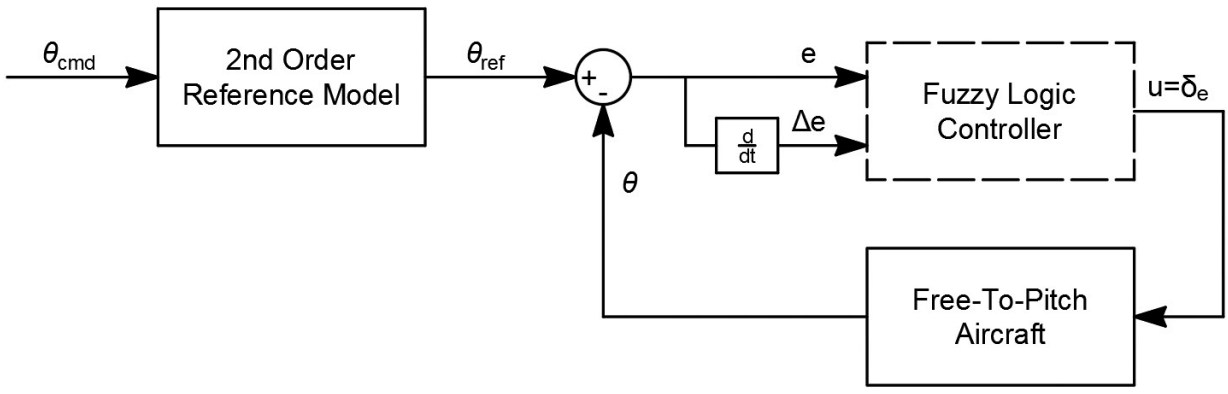

Fig. 18: Free-to-Pitch Control Architecture.

During simulations and testing phases, a single channel was implemented as proposed by Napolitano [14]. The output of this channel provided the absolute deflection angle command to the aircraft elevators, as opposed to an incremental change in deflection angle, based on the pitch error and error rates. This channel is referred to as the "Absolute Controller" channel within this thesis. As the aircraft would pitch towards the reference angle, the error approaches zero, and the elevators deflection angle approaches zero. Due to the controller inputs correlating directly to the deflection angle of the aircraft's elevators, it was found that this single channel configuration did not provide reasonable steady state errors.

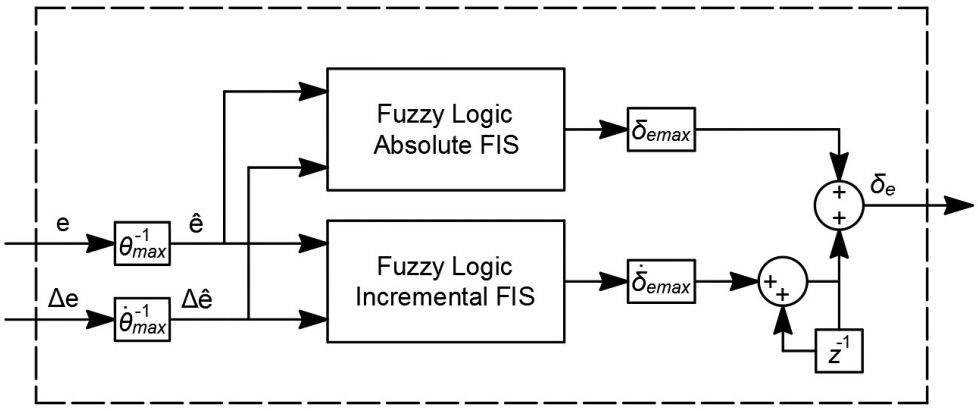

Fig. 19: Free-to-Pitch Fuzzy Logic Controller.

It was found that a second channel was required to provide a trimming output, and this channel is referred to as the "Incremental Controller" channel. The combination of these two channels, shown in Fig. 19, make up the architecture of the FLC Pitch Controller and will be discussed further in the following sections.

ABSOLUTE CHANNEL

The absolute channel uses the FIS developed by Napolitano [14], and outputs an absolute elevator angle deflection. For example, when a negative large theta, θ, error exists, the controller attempts to pitch the aircraft nose up by deflecting the elevators in a large negative direction. Conversely, when a positive small theta error exists, the controller attempts to pitch the aircraft nose down by deflecting the elevators in a small positive direction. There

is a dead zone around zero error, where the contribution of the absolute controller will be a zero deflection of the elevators. This dead zone was added to the controller Rule Base to provide a smoother steady-state response, by reducing control surface deflections due to small changes in error or signal noise once the aircraft has settled at small pitch angle error and error rates.

ΔE / E	NB	NM	NS	Z	PS	PM	PB
NB	NB	NB	NB	NM	NS	PS	PM
NM	NB	NB	NM	NM	NS	PS	PM
NS	NB	NB	NM	NS	Z	PM	PB
Z	NB	NM	NS	Z	PS	PM	PB
PS	NB	NM	Z	PS	PM	PB	PB
PM	NM	NS	PS	PM	PM	PB	PB
PB	NM	NS	PS	PM	PB	PB	PB

Fig. 20: Free-to-Pitch Absolute Controller Rule Base.

The Rule Base table for the free-to-pitch absolute controller FIS is shown in Fig. 20, and maps the input membership functions to the output membership functions by using the Rule Base as described in Chapter 2. This channel is designed for disturbance rejection and improved transient response to step commands by providing an absolute angle input to the aircraft elevators. This channel can provide rapid changes in pitching moment but does not provide adequate reduction in steady state error.

INCREMENTAL CHANNEL

To supplement the absolute control output and address potential large steady state errors, an incremental channel was developed. The incremental channel uses a FIS developed within this thesis, and outputs an incremental change in the elevator angle deflection. Where negative large theta, θ, error exists, the controller will attempt to pitch the aircraft nose up

by deflecting the elevators in a large negative increment from the previous elevator position value. Conversely, where a positive small theta error exists, the controller will attempt to pitch the aircraft nose down by deflecting the elevators in a small positive increment from the previous elevator position value. There is a dead zone around zero error, where the controller will not attempt to change the trim of the elevators. Similar to the absolute channel, the incremental channel provides a smoother steady-state response when small changes in error or signal noise does not correlate to additional changes in elevator deflection when the aircraft has settled near the desired pitch angle.

ΔE / E	NB	NM	NS	Z	PS	PM	PB
NB	PB	PB	PB	PB	PM	PS	PS
NM	PB	PM	PM	PS	PS	Z	Z
NS	PB	PS	PS	Z	Z	Z	NS
Z	PB	PS	Z	Z	Z	NS	NB
PS	PS	Z	Z	Z	NS	NS	NB
PM	Z	Z	NS	NS	NM	NM	NB
PB	NS	NS	NM	NB	NB	NB	NB

Fig. 21: Free-to-Pitch Incremental Controller Rule Base.

The Rule Base table for for the free-to-pitch incremental controller FIS is shown in Fig. 21, and maps the input membership functions to the output membership functions by using the Rule Base as described in Chapter 2. This channel is designed based on generic aircraft pitching control law for reduction of steady state error response to step commands by providing a trimming input to the aircraft elevators which only changes incrementally over sample periods.

3.3 TYPE-1 AND IT-2 FUZZY ROLL CONTROLLER

The free-to-roll FLC is based on the knowledge an aircraft pilot would use to control an aircraft in a rolling left and right movement. The controller receives roll angle error and change in error signals, which would be available to a pilot through instrumentation readings. The Rule Base of the roll controller is adapted from Napolitano's [14] Type-1 fuzzy logic system, for the Type-1 and IT-2 roll controllers. The Rule Base represents the pilots heuristic knowledge of flight control in the rolling moment. Reference signals are generated by feeding the desired aircraft roll angle, phi, through the second order reference model (27). An error signal is generated from the difference of the reference signal and the measured phi value from the aircraft sensors. The change in error signal is estimated by the difference of the error signals over each sample period. These two signals are fed into the FLC to generate a control signal which operates the aircraft ailerons. The FLC attempts to track the reference signal by minimizing the error signal while maintaining reasonable roll rates. This process is outlined within Figs. 22 and 23.

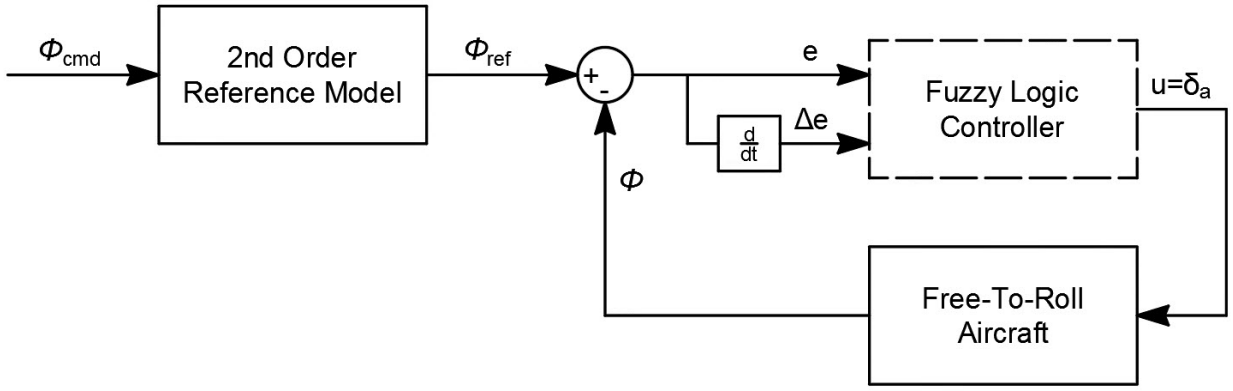

Fig. 22: Free-to-Roll Control Architecture.

During simulation and testing phases a single channel was implemented as proposed by Napolitano [14], and this configuration was adequate to maintain reasonable steady state errors. This was due to the nature of the input-output relationship of the aircraft roll mechanics. The inputs of error and change in error correlate to the absolute deflection angle of the aircraft's ailerons, and as the aircraft would roll towards the reference angle, the error approaches zero, and the ailerons deflection angle would approach zero.

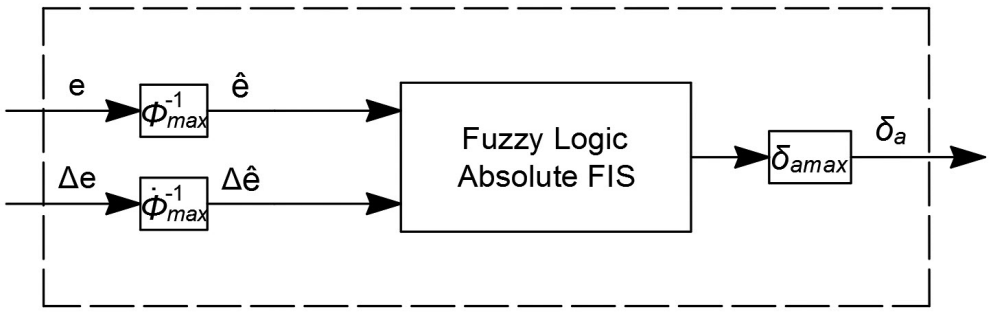

Fig. 23: Free-to-Roll Fuzzy Logic Controller.

While addition of a second "Incremental" channel was not needed during the simulations and wind tunnel testing presented, it may be useful for extreme regions of operation, with large side slip angle, beta, and where high pitch and roll coupling may occur. This additional channel could be designed based on generic aircraft rolling control law for reduction of steady state error response to step commands by providing a trimming input to the aircraft ailerons, which only changes incrementally over sample periods.

ABSOLUTE CHANNEL

The free-to-roll FLC consists of one channel, an absolute channel which uses the FIS developed by Napolitano [14], and outputs an absolute aileron angle deflection. The FLC receives the error and change in error signals as inputs. For example, where negative large phi, ϕ, error exists, the controller will attempt to roll the aircraft left by deflecting the right aileron in a large negative direction and the left aileron in a large positive direction. Conversely, where a positive small phi error exists, the controller will attempt to roll the aircraft right by deflecting the right aileron in a small positive direction and the left aileron in a small negative direction. There is a dead zone around zero error, where the contribution of the absolute controller will be a zero deflection of the ailerons.

E \ ΔE	NB	NM	NS	Z	PS	PM	PB
NB	NB	NB	NB	NM	NS	PS	PM
NM	NB	NB	NM	NM	NS	PS	PM
NS	NB	NB	NM	NS	Z	PM	PB
Z	NB	NM	NS	Z	PS	PM	PB
PS	NB	NM	Z	PS	PM	PB	PB
PM	NM	NS	PS	PM	PM	PB	PB
PB	NM	NS	PS	PM	PB	PB	PB

Fig. 24: Free-to-Roll Absolute Controller Rule Base.

The Rule Base table for the free-to-roll absolute controller FIS is shown in Fig. 24, and maps the input membership functions to the output membership functions by using the Rule Base as described in Chapter 2. This channel is designed for disturbance rejection, improved transient response and adequate steady state error to step commands, by providing an absolute angle deflection input to the aircraft elevators. This channel can provide rapid changes in rolling moment.

3.4 INPUT AND OUTPUT SCALING GAIN SELECTION

In many controllers, there are values that must first be tuned prior to achieving desired system performance. For model based controllers, this could be a state feedback gain matrix in a linear quadratic controller, or the three proportional-integral-derivative gains of a PID controller. In these cases the poles of the combined controller and system model are placed to match a desired linear reference model, such as Eq. (27), and can be thought of as "tuning knobs" of the system. In the case of fuzzy inference system, the input channel and output channel scaling gains are one of the means of tuning the controller to achieve a desired performance from the controlled system.

Each component of the FIS has an effect on performance of the controller, and up to this point, several of these design components have been generalized in the design. The set of seven input and output membership functions are generally shaped and spaced within the universe of discourse, as they could not be optimized for a specific aircraft without a model.

Similarly, the Rule Base is based on generic flight control laws. The intent of this approach was to minimize the amount of tuning required for the Pitch and Roll controller structures and use only general *a priori* knowledge of aircraft dynamics and physical restrictions of the aircraft.

Table II: Pitch and Roll Controller Scaling Gains.

	FIS Input and Output Scaling Gains		
	θ_{max}, Φ_{max}	$\dot{\theta}_{max}$, $\dot{\Phi}_{max}$	δ_{emax}, $\dot{\delta}_{emax}$, δ_{amax}
Absolute Pitch FIS	30	60	24
Incremental Pitch FIS	3	10	2
Absolute Roll FIS	10	150	10.75

The last design component of the FIS are the input and output scaling gains. These scaling gains determine how to scale input signals to fit within the general universe of discourse of [-1,1]. This determines what is considered a large input to the FIS. In this thesis, a large input to the pitch and roll FIS is considered a large error and a large change in error as previously defined. The controller designer is tasked with defining what value is appropriate for a large error and large change in error. This selected value is the input scaling gain for each input channel of the absolute and incremental FIS. For the output scaling gain, this determines how to scale an output from the general universe of discourse of [-1,1] to a change in a control surface angle such as elevator or aileron deflection angles. For the absolute FIS, the output scaling gain is selected based on the desired maximum range of the elevators and ailerons. For the incremental controller, the desired maximum rate of change of the actuators for the elevators and ailerons was selected for the output scaling gain. A table of the scaling gains for both the pitch and roll controllers is shown in Table II. While these values are not tuned to achieve optimal performance of the aircraft, there is enough *a priori* knowledge of the simulation or test aircraft to heuristically select these values from designer or expert experience for the task of providing adequate performance.

3.5 SIMULATIONS

A series of simulations were performed to provide motivation for wind tunnel testing of the proposed controller configurations detailed in the previous sections. Each of the Type-1 and IT-2 fuzzy logic controllers are used to "pilot" a nonlinear F16 simulation through a series of free-to-pitch and free-to-roll step input commands. The F16 nonlinear model will be presented, and the simulations results for each of controller and test configurations are compared.

NONLINEAR F16 MODEL

A nonlinear F16 simulation provided by the University of Minnesota [15] is used to simulate the AUT. This simulation is based on a report by Stevens and Lewis [16], and is described in the simulation manual [15] as the low fidelity model. This model has six degrees of freedom, three position and three attitude, and provides four input parameters: thrust, elevator, aileron and rudder deflection angles.

The F16 plant has twelve states: north position, east position, altitude, roll angle, pitch angle, yaw angle, total velocity, angle of attack, angle of side-slip, roll rate, pitch rate and yaw rate. The outputs of the plant consist of the twelve state derivatives, along with normalized accelerations in the three positional axis, Mach number, free stream dynamic pressure and static pressure.

The simulation is run in a Simulink and MATLAB environment at a sampling rate of 50 Hz. The sampling rate was chosen to match the available system hardware in the NASA Langley 12-ft low speed wind tunnel. The Simulink model, shown in Ref. [15], is initialized with the current aircraft states and control inputs, and after the simulation runs for a sample period, the final aircraft states are stored and sent to the MATLAB based controllers. These final aircraft state values are used as feedback inputs to the fuzzy logic controllers, as shown in Figs. 18 and 22.

Additionally, measurements of the nonlinear F16 simulation final state values, θ, $\dot{\theta}$, ϕ, and $\dot{\phi}$, for each sample period are simulated by adding a generated white noise signal. The motivation for adding this term was to explore robustness of the controller configurations

to uncertainties in sensor measurement data. The simulated measured state values,

$$\begin{aligned}
\hat{\theta}(t) &= \theta(t) + \beta_{\theta}\overline{W}(t), \\
\dot{\hat{\theta}}(t) &= \dot{\theta}(t) + \beta_{\dot{\theta}}\overline{W}(t), \\
\hat{\phi}(t) &= \phi(t) + \beta_{\phi}\overline{W}(t) \text{ and} \\
\dot{\hat{\phi}}(t) &= \dot{\phi}(t) + \beta_{\dot{\phi}}\overline{W}(t),
\end{aligned} \tag{28}$$

combine the actual state values from the Simulink model with a band-limited white noise signal with unity variance, $\overline{W}$, generated by the Simulink model. The Signal-to-Noise Ratio (SNR) for the aircraft attitude and rate sensors is defined as the ratio of signal power to signal noise power [17]. To achieve the desired SNR for each state signal in the simulation, a scalar is added to the SNR equation,

$$SNR = \frac{SignalPower}{\beta^2 * NoisePower},$$

where β is a scalar which allows the Signal-to-Noise ratio to be fixed for each simulation run. The following equations for each of the states,

$$\beta_{\theta}^2 = \frac{\sum \theta_{ref}^2}{SNR * \sum \overline{W}^2}, \quad \beta_{\dot{\theta}}^2 = \frac{\sum \dot{\theta}_{ref}^2}{SNR * \sum \overline{W}^2}, \quad \beta_{\phi}^2 = \frac{\sum \phi_{ref}^2}{SNR * \sum \overline{W}^2}, \quad \beta_{\dot{\phi}}^2 = \frac{\sum \dot{\phi}_{ref}^2}{SNR * \sum \overline{W}^2},$$

are used to find the scalar value for each of the the measured states in Eq. (28).

Signal Power is defined as the square of the reference signal. While the reference signal does not represent the actual signal, it represents the desired signal value and is also available before the simulations are conducted. This is an appropriate approximation of the signal power to calculate the scaling terms, β, prior to running the simulations. The final noise signal for each of the measured states of the F16 Nonlinear simulations is calculated and added to the final state value by the set of Eq. (28) before they are sent to the pitch and roll controllers.

TEST CONFIGURATIONS

Four independent simulations are conducted with the nonlinear F16 model testing the Type-1 and IT-2 pitch and roll controllers. To simulate a one degree of freedom attitude configuration, the attitude axis not under test is commanded to a zero value and controlled by its corresponding controller. For example, in the free-to-pitch Simulations, the roll axis is commanded to zero degrees, and the roll controller is tasked with maintaining this position while the aircraft is commanded through the pitch command sequence. The one degree of freedom constraint will simulate the wind tunnel test configuration in the following section. In all simulations, north position, east position, altitude and yaw angle are left free to vary.

FREE TO PITCH SIMULATION RESULTS

The free-to-pitch simulations were run through a command sequence of step inputs between positive eight and negative eight degrees of pitch angle. Each of the step inputs are held at the desired pitch angle for ten seconds before moving to the next command value. During this simulation, the aircraft roll angle is commanded to zero degrees, and the roll controller is used to maintain this configuration for a one degree of freedom simulation. Tables III and IV contains the test and controller parameters.

Table III: Free to Pitch Simulation Parameters

Command Sequence (deg): 10 Second Steps	0, 8, 0, -8, 0, 8, 0, -8, 0, 8, 0, -8
Velocity (ft/s)	700
Test Duration (sec)	120
Signal to Noise Ratio	20
Altitude (ft)	15,000

Each of the controller types, Type-1 and IT-2, are run through identical simulations with identical parameters, and only the type of fuzzy inference system is varied. The two inputs to the pitch controller are error and change in error, and each are passed through the the Type-1 and IT-2 fuzzy inference systems with input membership functions as shown in

Fig. 6. The Rule Base shown in Figs. 20 and 21, map the input membership functions to the output membership functions for the absolute and incremental channels, respectively.

Table IV: Free-to-Pitch Controller Parameters

Scaling Gain	Input 1, e	Input 2, Δe	Output
Absolute	30 deg	60 deg/s	24 deg
Incremental	3 deg	10 deg/s	2 deg

Figs. 25 and 26 represent the F16 simulation results for a Type-1 and IT-2 fuzzy logic control architecture, respectively, in the free-to-pitch configuration. As previously discussed, the motivation for using the IT-2 fuzzy inference system is to reduce the impact of noise in the input signals on the output control.

Four measurements were estimated for each of the free-to-pitch controller architecture system outputs to compare the performance over the test sequences: Mean Average Error (MAE), Percent Overshoot, Rise Time and Settling Time. The Mean Average Error,

$$MAE = \frac{\sum_{k=1}^{n} |\hat{\theta}_k - \theta_{ref}|}{n}, \tag{29}$$

averages differences between the simulated measured system output, $\hat{\theta}$, and the desired reference value, $\hat{\theta}_{ref}$, over the entire command sequence. Percent Overshoot,

$$\%OS = \frac{|\theta_{ref} - \hat{\theta}_{max}|}{|\hat{\theta}_{ref}|} * 100\%, \tag{30}$$

calculates the maximum system output, $\hat{\theta}_{max}$, which exceeds the desired reference value, $\hat{\theta}_{ref}$, for each of the step commands. Rise time is estimated as the time required for the simulated measured system output, $\hat{\theta}$, to reach 90% of the desired reference value, $\hat{\theta}_{ref}$, and Settling Time is estimated when the simulated measured system output, $\hat{\theta}$, remains within a signal error band of $\pm$ 7.5%.

Table V: Free-to-Pitch Controller Performance.

Measurement	Type-1 Average	IT-2 Average
Mean Average Error (deg)	0.31	.42
Percent Overshoot (%)	10.48	16.73
Rise Time (s)	1.35	1.31
Settling Time (s)	6.25	5.64

Table V provides the average performance measurements over all step inputs of the command sequence in Table III. The Type-1 controller has better MAE and Percent Overshoot as compared to the IT-2 Controller, while providing similar Rise Time and a slower Settling Time. The performance measurements for both the Type-1 and IT-2 controllers are acceptable.

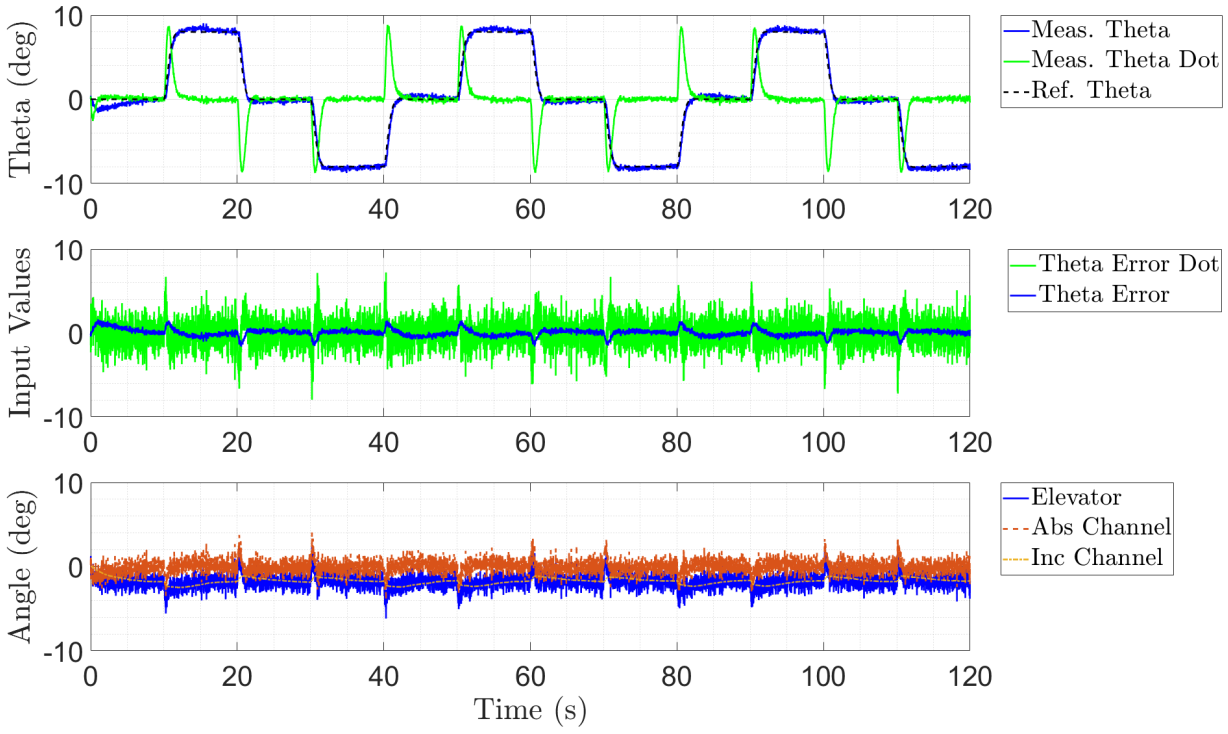

Fig. 25: Type-1 FLC in Free-to-Pitch Nonlinear F16 Simulation.

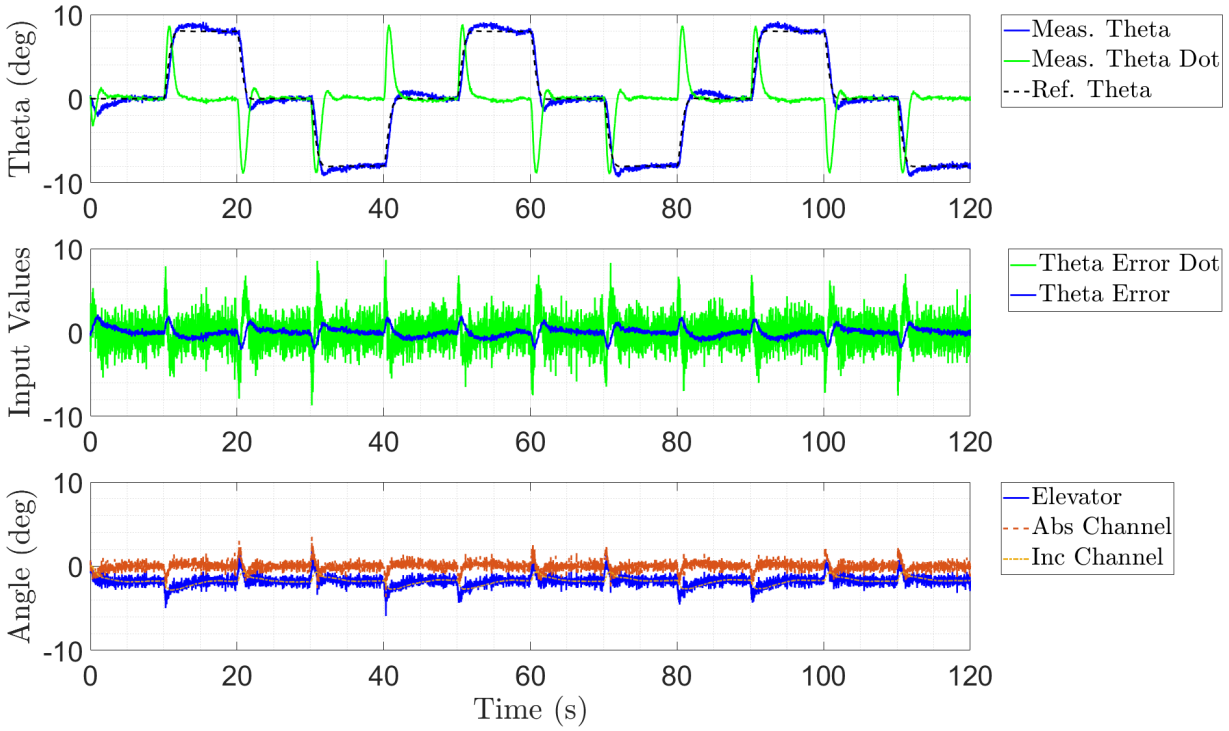

Fig. 26: Type-2 FLC in Free-to-Pitch Nonlinear F16 Simulation.

The Type-1 controller exhibited acceptable reference tracking throughout the simulation, but the controller output was significantly affected by the additional noise signal. When compared to the Type-1, the IT-2 controller provided similarly acceptable reference tracking but was less sensitive to the additional noise parameter.

FREE TO ROLL SIMULATION RESULTS

The free-to-roll simulations were run through a command sequence of step inputs between +20 and -20 degrees of roll angle. Each of the step inputs are held at the desired roll angle for 10 seconds before moving to the next command value. During this simulation, the aircraft pitch angle is commanded to zero degrees, and the pitch controller is used to maintain this configuration for a one degree of freedom simulation. Tables VI and VII contains the test and controller parameters.

Table VI: Free-to-Roll Simulation Parameters.

Command Sequence (deg): 10 Second Steps	0, 20, 0, -20, 0, 20, 0, -20, 0, 20, 0, -20
Velocity (ft/s)	700
Test Duration (sec)	120
Signal to Noise Ratio	40
Altitude (ft)	15,000

Each of the controller types are run through identical simulations with identical parameters, and only the type of fuzzy inference system is varied. The two inputs to the roll controller is error and change in error, and each are passed through the the Type-1 and IT-2 fuzzy inference systems with input membership functions as shown in Fig. 7. The Rule Base, shown in Fig. 24, maps the input membership functions to the output membership functions for the absolute channel.

Table VII: Free to Roll Controller Parameters.

Scaling Gain	Input 1, e	Input 2, Δe	Output
Absolute	10 deg	150 deg/s	10.75 deg

Figs. 27 and 28 represent the F16 simulation results for a Type-1 and IT-2 fuzzy logic control architecture, respectively, in the free-to-roll configuration. As previously discussed, the motivation for using the IT-2 fuzzy inference system is to reduce the impact of noise in the input signals on the output control.

Table VIII: Free-to-Roll Controller Performance.

Measurement	Type-1 Average	IT-2 Average
Mean Average Error (deg)	0.43	0.45
Percent Overshoot (%)	6.79	6.98
Rise Time (s)	1.75	1.77
Settling Time (s)	2.67	3.7

The same four measurements, MAE, Percent Overshoot, Rise Time of 90% and Settling Time of ± 7.5%, were estimated for each of the free-to-roll controller architecture system outputs to compare the performance over the test sequence. Table VIII provides the average performance measurements over all step inputs of the command sequence in Table VI. The Type-1 controller has similar MAE, Percent Overshoot, Rise Time and Settling Time as compared to the IT-2 Controller and the performance measurements for both the Type-1 and IT-2 controllers are acceptable.

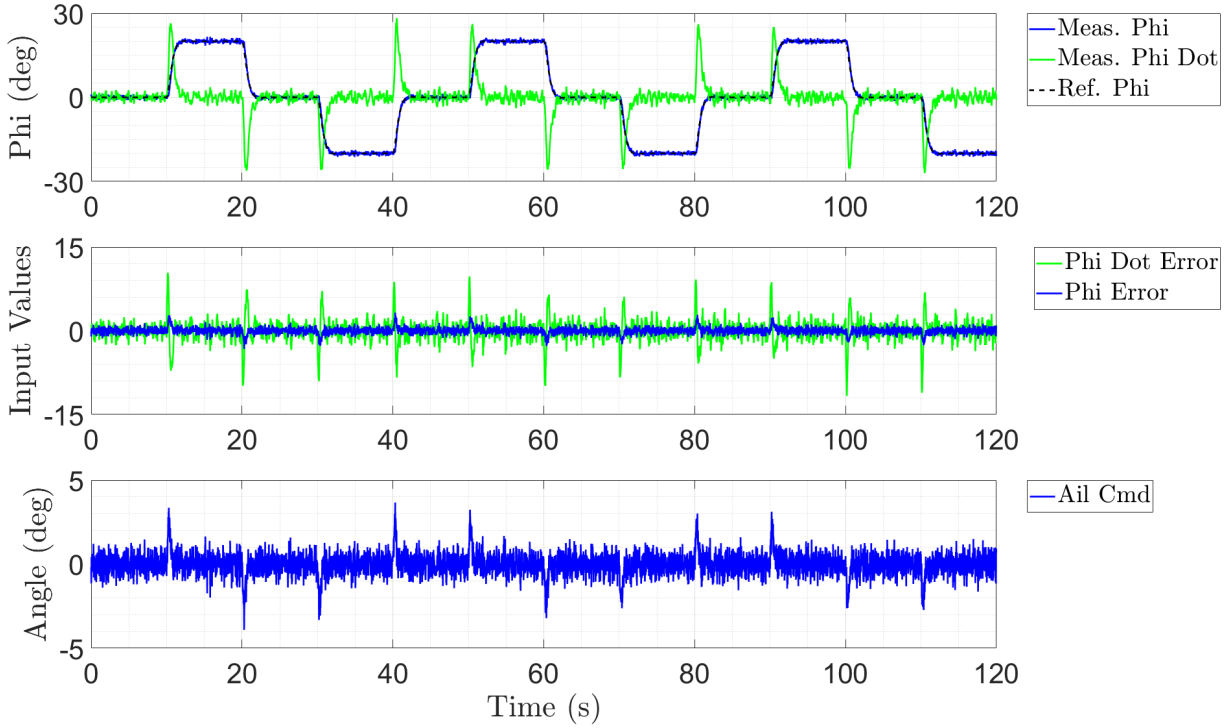

Fig. 27: Type-1 FLC in Free-to-Roll Nonlinear F16 Simulation.

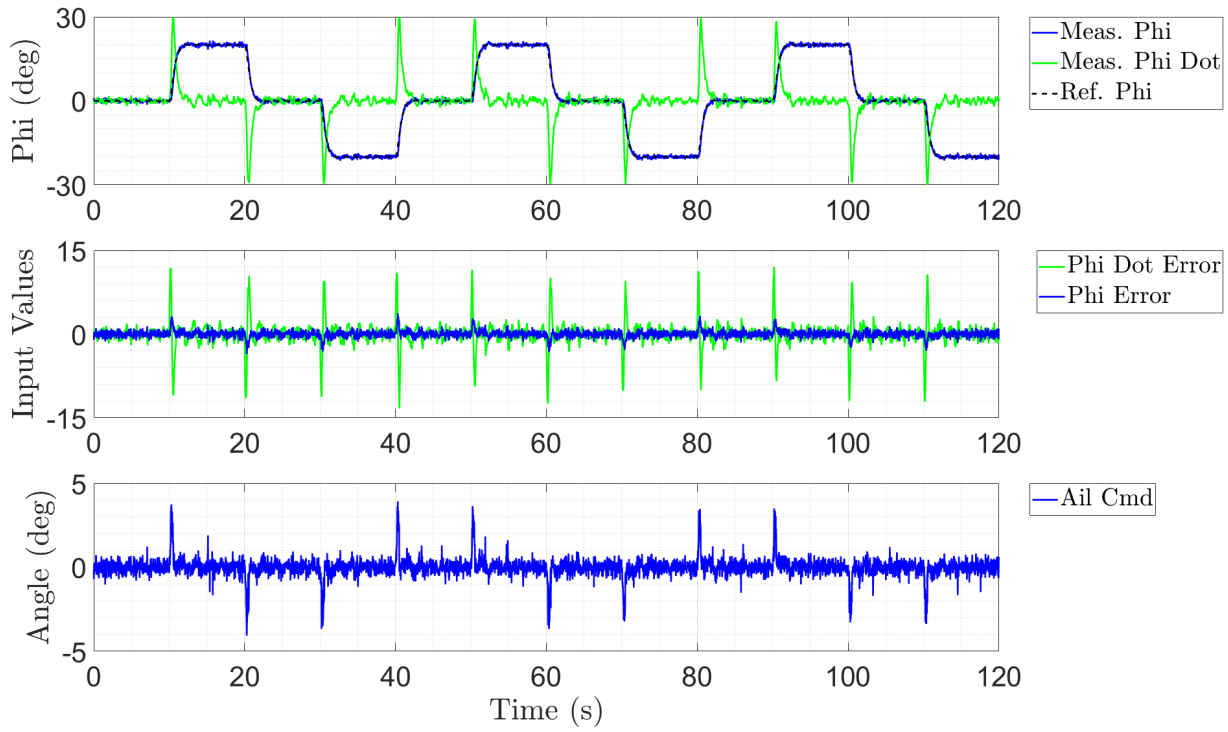

Fig. 28: Type-2 FLC in Free-to-Roll Nonlinear F16 Simulation.

The Type-1 controller exhibited acceptable reference tracking throughout the simulation, and the controller output exhibited significant affects by the additional noise signal. When compared to the Type-1, the IT-2 controller provided similarly acceptable reference tracking and was less sensitive to the additional noise parameter. In both simulation configurations the controller output is acceptable.

In both of the free-to-pitch and free-to-roll simulation configurations, the Type-1 and IT-2 fuzzy logic controller provided acceptable reference tracking. The IT-2 fuzzy inference system was less sensitive to the additional noise signal when applied to the inputs, however this appeared to have less of an impact on the aircraft dynamics and more of an impact on the input to the control surface actuators. These result are as expected based on review of the IT-2 literature, and further comparison between the Type-1 and IT-2 controllers is shown from testing in the wind tunnel and is presented in the following chapter.

TWO DEGREES OF FREEDOM SIMULATION RESULTS

In addition to the independent pitch and roll simulations, both controllers were combined in a two degrees of freedom simulation, where each of the previous simulations ran in parallel with the same independent command sequences from Sections 3.5.3 and 3.5.4. The purpose of this final simulation is to show that the pitch and roll controllers were able to run simultaneously without causing a processing bottleneck with the IT-2 fuzzy logic structures and that the controllers are able to maintain control of the aircraft in a coupled state. As with the one degree of freedom (DOF) simulations, each of the controller types are run through identical simulations with identical parameters, and only the type of fuzzy inference system is varied.

Table IX: Free-to-Pitch two DOF Controller Performance.

Measurement	Type-1 Average	IT-2 Average
Mean Average Error (deg)	0.32	0.43
Percent Overshoot (%)	10.90	17.91
Rise Time (s)	1.37	1.32
Settling Time (s)	6.31	5.43

Table X: Free-to-Roll two DOF Controller Performance.

Measurement	Type-1 Average	IT-2 Average
Mean Average Error (deg)	0.45	0.50
Percent Overshoot (%)	7.51	8.31
Rise Time (s)	1.74	1.76
Settling Time (s)	3.32	4.12

The same four measurements, MAE, Percent Overshoot, Rise Time of 90% and Settling Time of ± 7.5%, were estimated for each of the free-to-pitch and free-to-roll controller architecture system outputs to compare the performance over the coupled test sequences. Tables IX and X provide the average performance measurements over all step inputs of the command sequences in Tables III and VI. The performance metric results of the two DOF simulation has similar MAE, Percent Overshoot, Rise Time and Settling Time as compared to the one DOF independent simulations, and the performance measurements for both the Type-1 and IT-2 controllers are still acceptable.

Figs. 29 - 32 represent the nonlinear F16 simulation results for the Type-1 and IT-2 fuzzy logic control architectures in the two degrees of freedom configuration. As previously discussed, the motivation for using the IT-2 fuzzy inference system is to reduce the impact of noise in the input signals on the output control. During the two DOF simulations, both axis of movement are experiencing the same additional noise signal use in the one DOF simulations.

The Type-1 and Type-2 controller output profiles for both Pitch and Roll in two DOF are very similar to their corresponding one DOF simulations. Both Pitch and Roll controllers exhibited acceptable reference tracking throughout the simulation and the controller outputs exhibited similar effects by the additional noise signal as shown by the one DOF simulations. This simulation showed that combining the simulations simultaneously in pitch and roll did not have a significant additional effect on the system performance or controller outputs within the tested region.

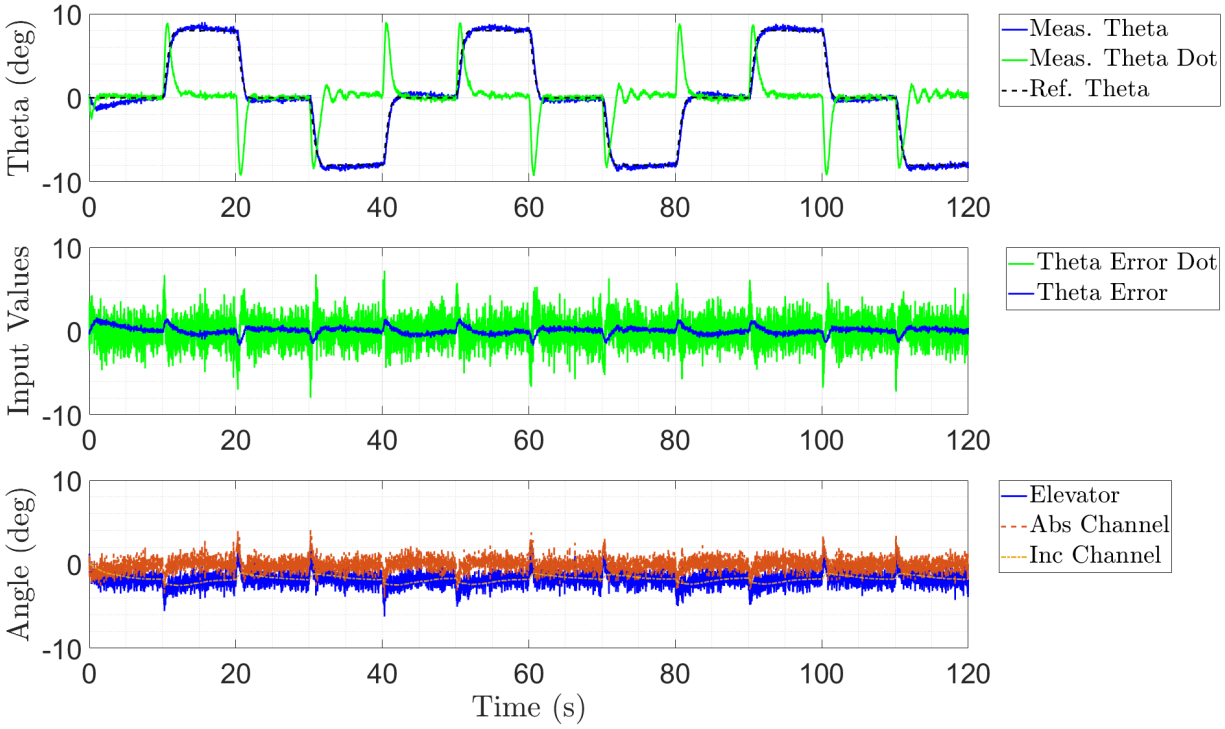

Fig. 29: Type-1 FLC in Free-To-Pitch-and-Roll Nonlinear F16 Simulation.

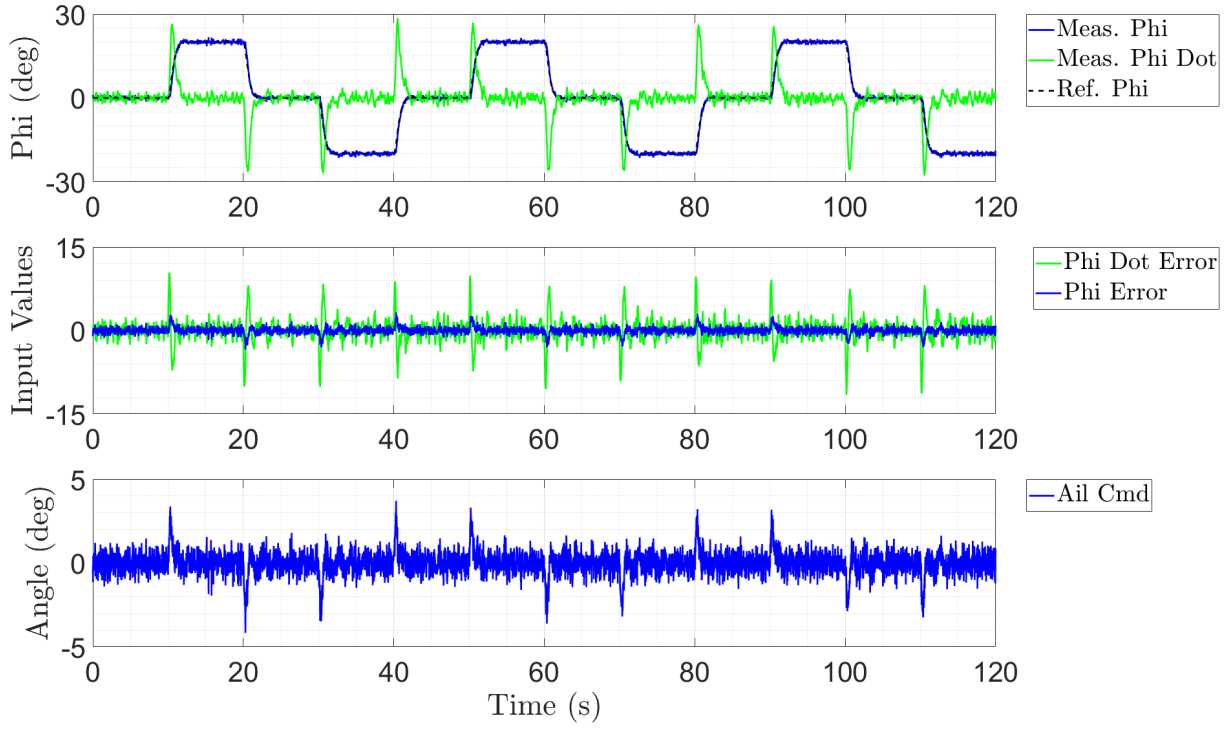

Fig. 30: Type-1 FLC in Free-to-Pitch-and-Roll Nonlinear F16 Simulation.

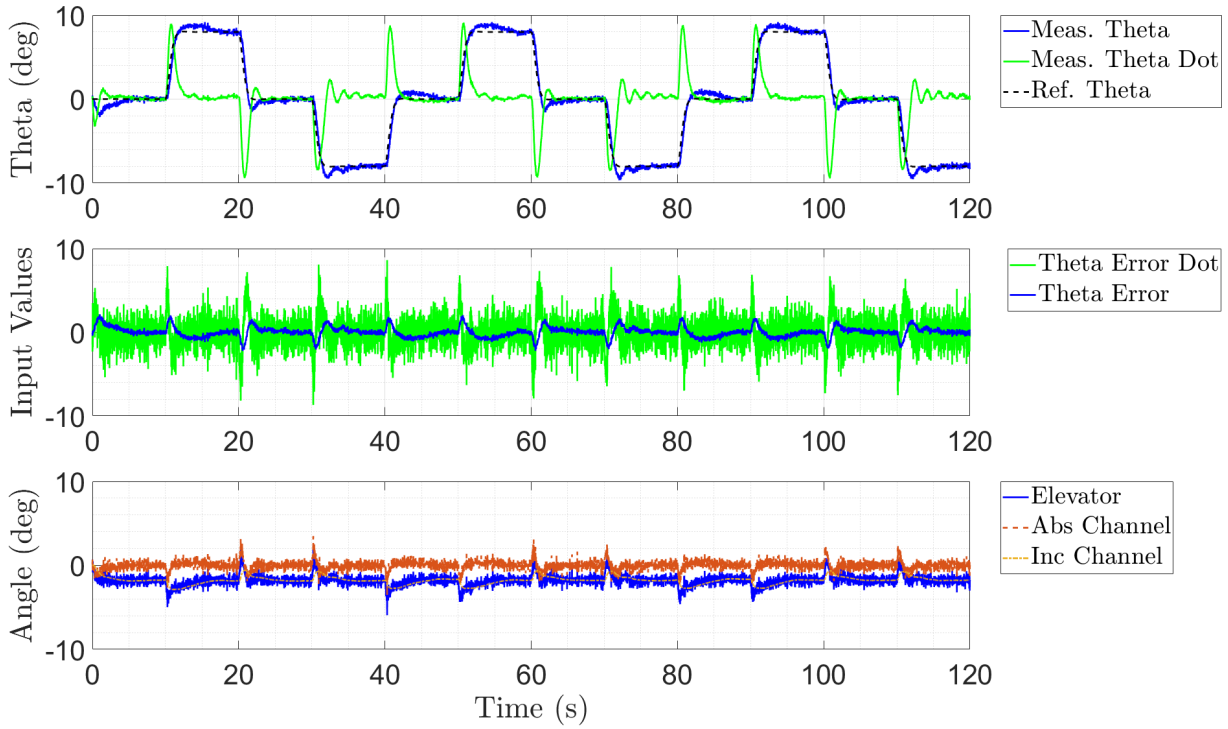

Fig. 31: Type-2 FLC in Free-To-Pitch-and-Roll Nonlinear F16 Simulation.

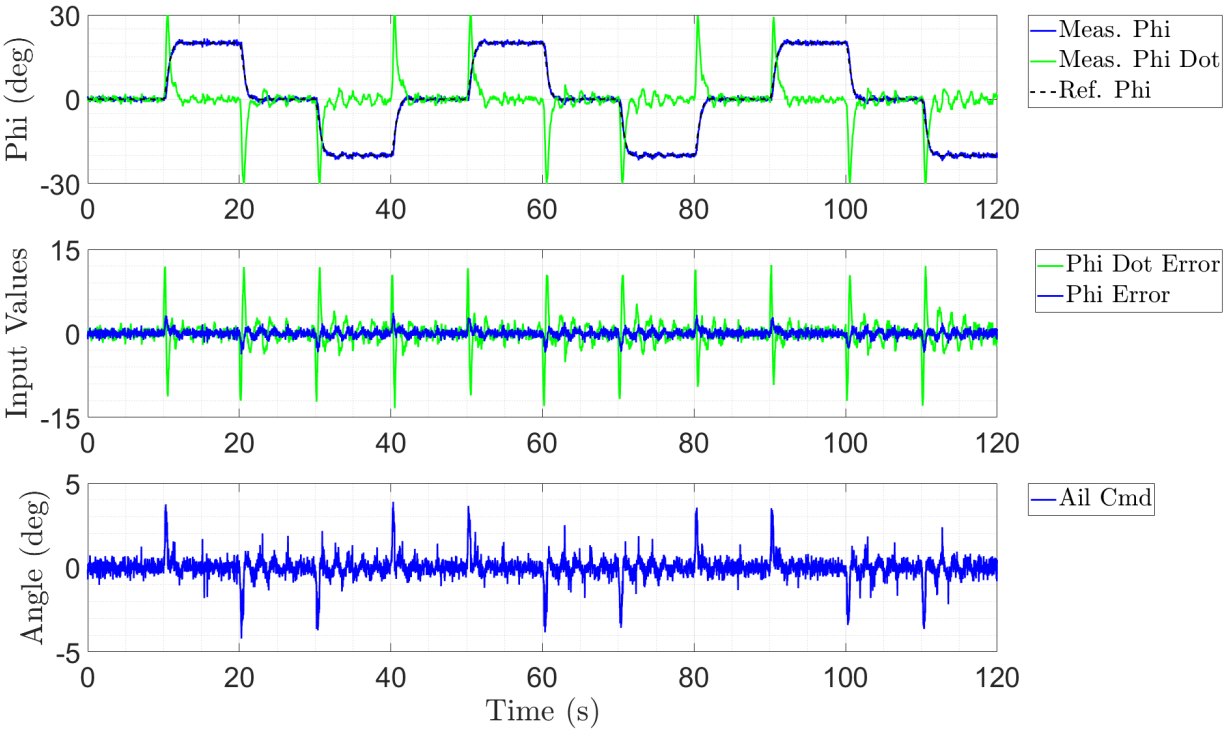

Fig. 32: Type-2 FLC in Free-to-Pitch-and-Roll Nonlinear F16 Simulation.

3.6 CONTRIBUTION OF CONTROL ARCHITECTURE

A fuzzy logic system was presented which generalizes an expert's knowledge of how an aircraft is controlled and defines heuristic and physical-meaning input-output associations in order to provide adequate control actions. The proposed controllers consisted of a two channel FIS architecture of absolute and incremental outputs for pitch control, and a single channel FIS architecture of absolute output for roll control.

The free-to-pitch controller provided disturbance rejection and improved transient response by providing an absolute deflection angle output from the absolute channel, and reduced steady state error by providing a trimming output from the incremental channel. The combination of these two channels provided the control signal for the aircraft elevators. The free-to-roll controller provided disturbance rejection, improved transient response and adequate steady state error by providing an absolute deflection angle output from the signal absolute channel. This channel provided the control signal for the aircraft ailerons.

The controller Rule Base and membership functions were selected generically to minimize the amount of tuning required for the pitch and roll controller structures. This allowed only the input and output scaling gains to require specific selection criteria. The input scaling gain for each input channel of the absolute and incremental FIS were selected based on expected values for a large error and large change in error, allowing the FIS to activate through the full range of the universe of discourse. Estimated values can be found through simulation or designer knowledge of similar aircraft platform. The absolute channels output scaling gain was selected based on the desired maximum range of the elevators and ailerons, to provide full range of motion of the control surfaces without exceeding the hardware limits. The incremental controller output scaling gain was selected based on the desired maximum rate of change of the actuators for the elevators and ailerons in order to provide maximum rate of deflection without exceeding the hardware limits.

A nonlinear simulation of an F16 aircraft in flight was presented as motivation and proof of concept for a wind tunnel experiment. The simulation provided free-to-pitch and free-to roll configurations of the nonlinear F16 model by controlling and holding the "off axis" to zero degrees. A two DOF simulation was also presented to show that both controller configurations would run simultaneously and provide similar performance to the one DOF simulations. The two-channel pitch controller and one-channel roll controller provided adequate stability and performance to motivate an experimental demonstration at the NASA Langley 12ft low speed wind tunnel. The setup and experiment for free-to-pitch and free-to-roll will be presented in the next chapter.

CHAPTER 4

EXPERIMENTAL RESULTS

The goal of this project is to provide independent one DOF stable flight control of the AUT in pitch and roll axis, as proof of concept for future two and three DOF testing. The 12ft low-speed wind tunnel in Hampton, Virginia is equipped with a mounting rig that can be configured in one degree of freedom at a time. Wind tunnel testing of the proposed controllers for pitch and roll, which have been outlined in the previous chapter, are tested in each of their corresponding configurations. In addition to providing stability through a series of controlled maneuvers, the pitch controller is tested with the use of the RTGM Algorithm, to develop a nonlinear model of the AUT. The proposed controller is tasked with maintaining control of the aircraft in pitch movement, through a series of global maneuvers, while the RTGM algorithm provides planned test inputs to the corresponding control surfaces and estimates a global nonlinear model of the aircraft.

Additional wind tunnel testing of similarly designed fuzzy logic based pitch and roll controllers was also conducted. The testing results are presented to compare the performance of the Type-1 and IT-2 fuzzy logic pitch and roll controllers discussed in Chapter 3. The performance between Type-1 and IT-2 is compared in terms of transient and steady state errors. The controller outputs for each fuzzy set type are also considered. This section will be broken into free-to-pitch and free-to-roll subsections, as these wind tunnel tests were run independently.

4.1 EXPERIMENT SETUP

The testing of the proposed fuzzy logic controllers was performed in the NASA Langley 12-ft low speed wind tunnel in Hampton, VA. The AUT is a commercial 40 percent scaled model of the Extra 330 SC aircraft , which has been modified by the NASA Langley MCAAD group to be mounted onto a one DOF rig, either in the free-to-pitch or free-to-roll configuration. The AUT lifting surfaces available for control are the elevators for free-to-pitch, and the ailerons for free-to-roll. The rudder is fixed and not controlled.

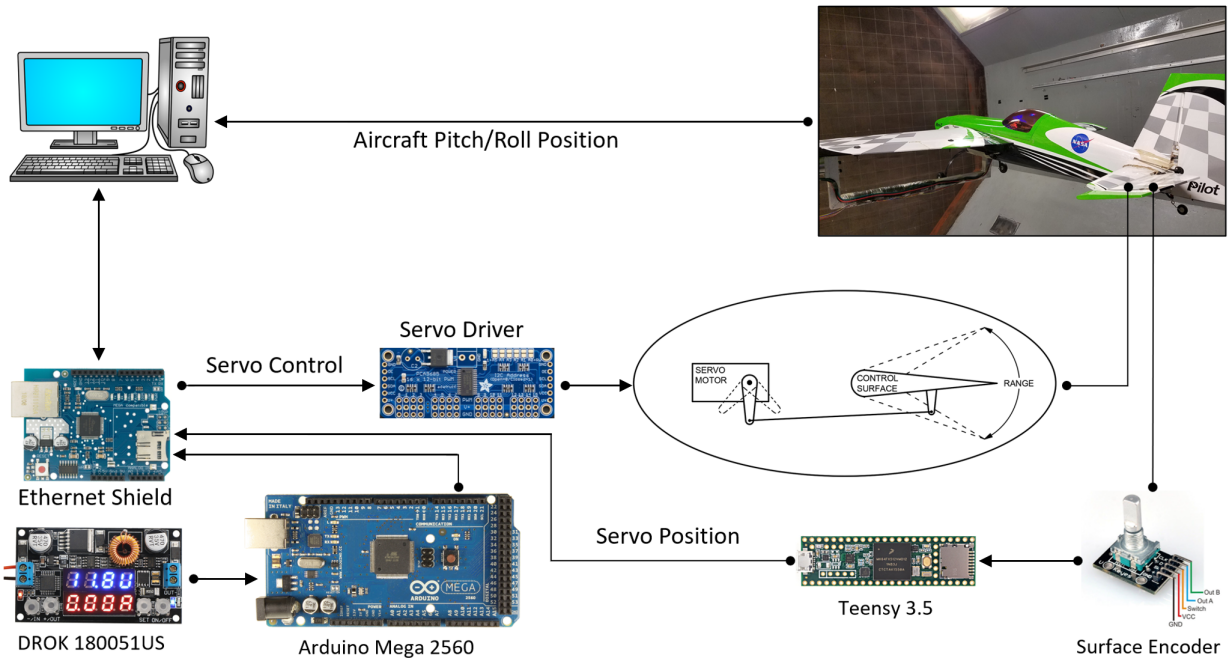

Fig. 33: 12-ft Low Speed Wind Tunnel Hardware Configuration.

The hardware suite used for the AUT includes an Arduino Mega 2560 [18] with a W5500 Ethernet Shield [19] and Encoder Board. This microprocessor is used to control all of the lifting surface servo motors through an Adafruit PCA9685 16x12 Bit PWM Servo Shield [20]. Controller output is sent through these units to change the deflection of the AUT lifting surfaces. Each control surface has an encoder, which is read by a Teensy 3.5 [21], and the data is sent to the Arduino stack and back to the controlling computer station. All of the control hardware is powered by two DROK 180051US Numerical Control Voltage Regulators [22]. The AUT pitch or roll position is read with US Digital MA3 12-bit PWM Magnetic Encoders [23], filtered with a Krohn-Hite Model 3364 4-pole Butterworth Filter [24] using a cutoff frequency of 10 Hz. This data is sent back to the controlling computer station through the wind tunnel network system. The AUT hardware suite configuration is shown in Fig. 33.

This hardware suite configuration provides an interface between the MATLAB control software running on the controlling computer station and the aircraft in the wind tunnel. The control software runs at a rate of 50 Hz, sampling the current pitch or roll state of the aircraft and the current angle of the elevator or aileron lifting surface. The pitch or roll

error is calculated and sent to the controller where a new elevator or aileron deflection angle command is determined. The controlling computer station sends the updated command to each lifting surface servo motor through the aircraft hardware interface. A time series of the aircraft pitch, roll and lifting surface angles is recorded from this data for each test.

4.2 F2P TYPE-1 AND IT-2 FUZZY LOGIC CONTROL COMPARISON

The controller described in Chapter 3.2 is tested in a free-to-pitch configuration. In this configuration, the roll, ϕ, and yaw, ψ, axis are locked out while the aircraft is free to rotate on the pitch, θ, axis. The controller output is sent to each of the elevator surfaces on the aircraft, and the remaining aircraft surfaces are set to zeros. The aircraft controller is provided readings from the aircraft encoder for pitch angle. Pitch error and pitch error rates are calculated based on the difference from the encoder reading and the desired reference pitch angle, θ_{ref}, which is generated from the second order reference model (27).

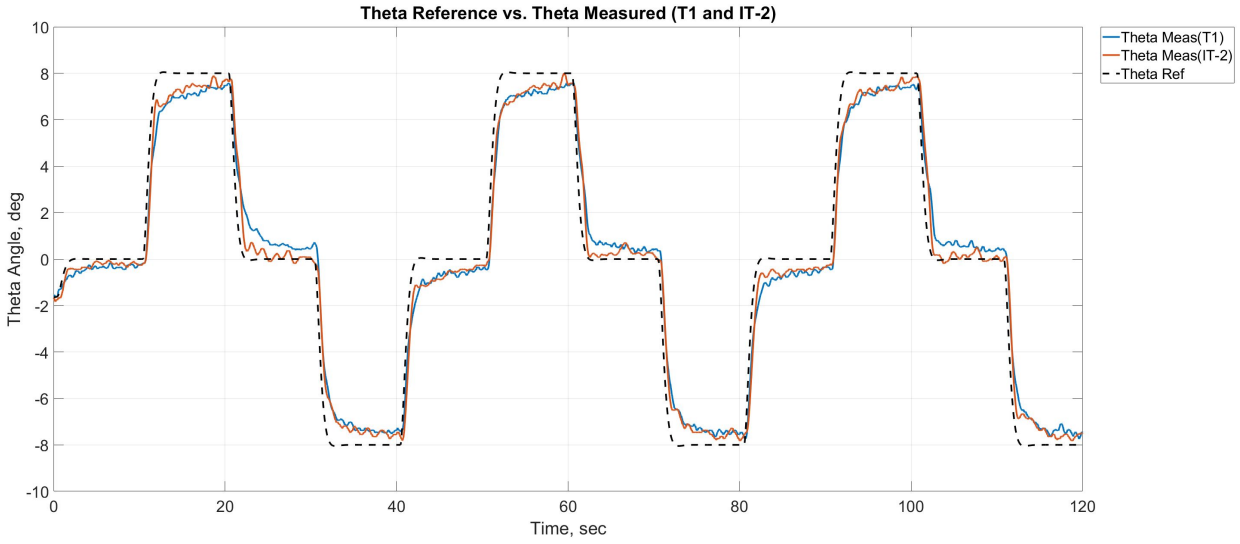

Fig. 34: Free-to-Pitch Step Response Comparison of Type-1 and IT-2 FIS Structures.

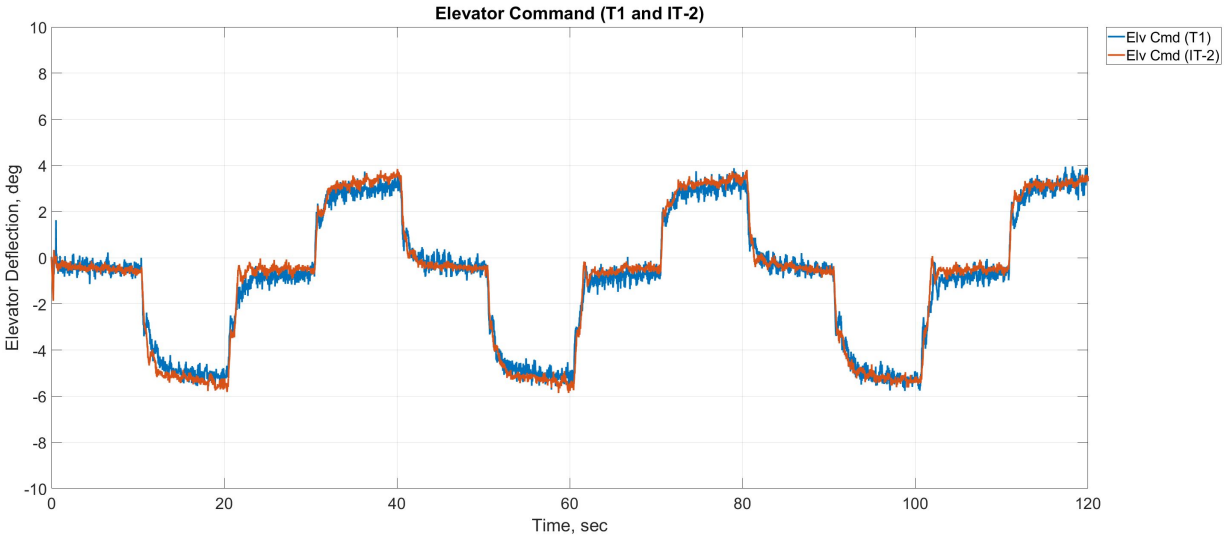

Fig. 35: Free-to-Pitch Controller Output to Elevator Surfaces for Type-1 and IT-2 FIS Structures.

The Type-1 and IT-2 controllers are tested independently by subsequent identical test runs. The step commands of desired pitch angle, θ, are sent to the controller after filtering through the reference model (27), and the corresponding step responses are shown in Fig. 34. The IT-2 controller exhibits slightly better steady state error. The controller output for each controller type is shown in Fig. 35. The IT-2 controller output signal has reduced high frequency oscillations as compared to the Type-1 controller. The goal of this test was to show that each of the controller configurations is able to maintain stable control of the aircraft through a range of pitch angles. This controller will be tested with the RTGM algorithm, and detailed results will be presented in Section 4.5.

4.3 F2R TYPE-1 AND IT-2 FUZZY LOGIC CONTROL COMPARISON

The controller described in Chapter 3.3 is tested in a free-to-roll configuration. In this configuration, the pitch, θ, and yaw, ψ, axis are commanded through the cross box sequence shown in Fig. 36, where the aircraft is starting from position 0 and transitions sequentially to position 33. The cross box is used to test the robustness of the roll controller through a range of pitch and yaw sting angles. The aircraft is free to rotate on the roll axis, ϕ, throughout the test. The controller output is sent to each of the aileron surfaces on the aircraft, and

the remaining aircraft surfaces are set to zeros. The aircraft controller is provided readings from the aircraft sensor for roll angle, ϕ, and the roll error and error rates are calculated based on the desired reference roll angle, ϕ_{ref}, which is generated from the second order reference model.

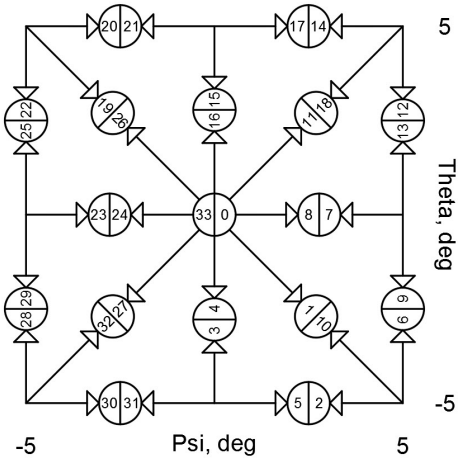

Fig. 36: Cross Box Sequence for Psi and Theta during Free-to-Roll Testing.

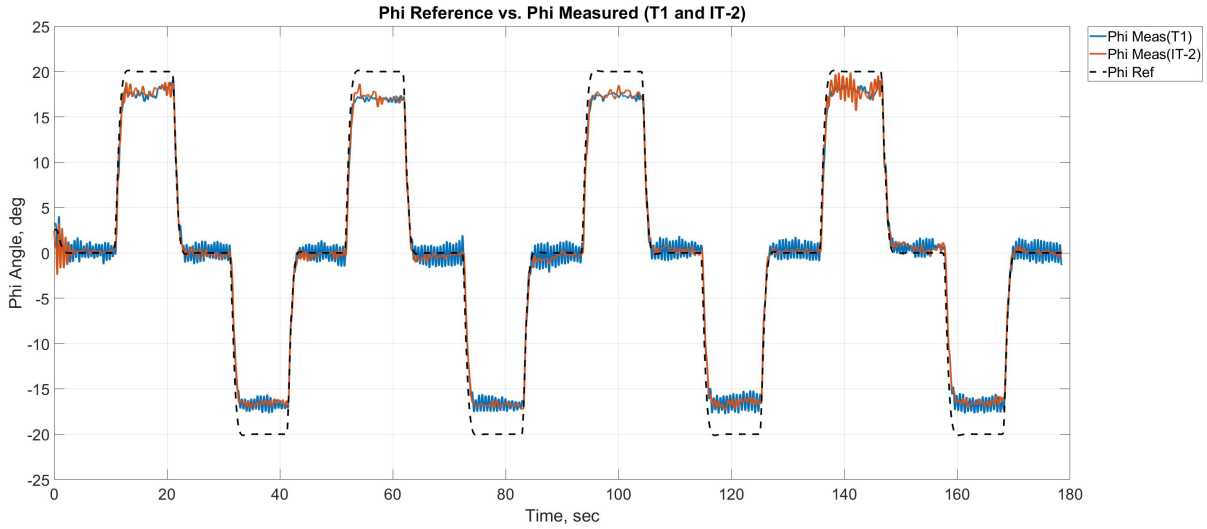

Fig. 37: Free-to-Roll Step Response Comparison of Type-1 and IT-2 FIS Structures.

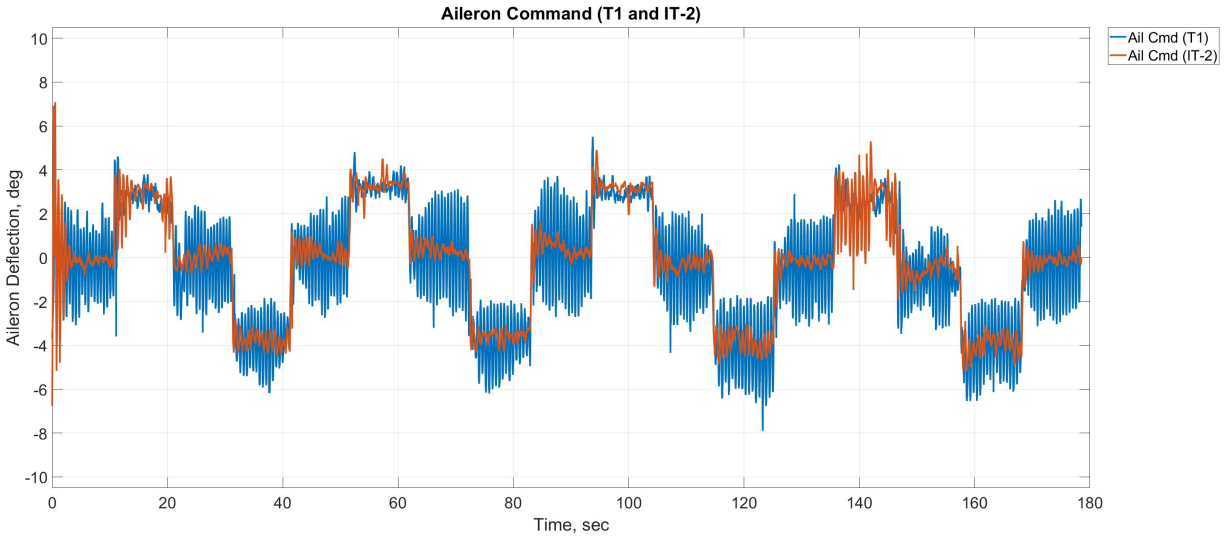

Fig. 38: Free-to-Roll Controller Output to Aileron Surfaces for Type-1 and IT-2 FIS Structures.

The Type-1 and IT-2 controllers are tested independently by subsequent identical test runs. The step commands of desired roll angle, ϕ, are sent to the controller after filtering through the reference model (27), and the corresponding step responses are shown in Fig. 37. The IT-2 controller exhibits better steady state error, minimizing high frequency osculations as compared to the Type-1 controller response. The controller output for each controller type are shown in Fig. 38. The IT-2 controller output signal has reduced high frequency oscillations as compared to the Type-1 controller, which contributed to the better system response.

4.4 REAL-TIME GLOBAL MODELING (RTGM)

A key component of the Learn-To-Fly concept is real-time global nonlinear aerodynamic modeling based on flight data, and has been extensively detailed by Morelli [2][25][26]. This section will briefly cover the importance of this technology for the Learn-to-Fly program and how it is used with the fuzzy logic controllers detailed in this paper. The author encourages a thorough examination of the topic using the referenced literature.

The RTGM method injects automatic control surface perturbations into the command signals for each of the control surfaces by summing an excitation signal with the control commands from the flight controllers. These excitation signals are automated, orthogonally

optimized and multi-sine perturbations, called planned test inputs (PTI). These PTI signals are designed to excite the aircraft's dynamic response in all six degrees of freedom of the aircraft, however for the one degree of freedom testing, PTIs are only applied to one control surface. This results in a global nonlinear aerodynamic model that can be used to design a linear PD controller to be implemented in place of the fuzzy logic controllers.

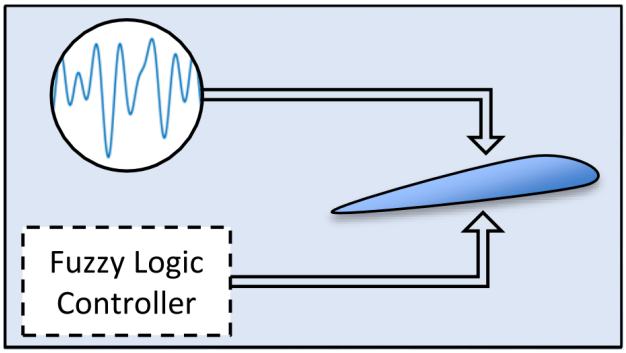

Fig. 39: Control Surface with PTI.

The PTI signals, $\boldsymbol{u}_j$, are calculated using the equation presented by Morelli [2]. These PTI signals are determined prior to experimental testing of the AUT from the following equation,

$$\boldsymbol{u}_j = \sum_{k\in(1,2,\ldots,M)} A_k \sin\left(\frac{2\pi \boldsymbol{t}}{T} + \Phi_k\right), \tag{31}$$

where M is the total number of harmonic frequencies, T is the length of the test in seconds, and A_k is the amplitude of the k^{th} sinusoidal component. This sum of sinusoidal inputs provides orthogonality in the time and frequency domains, due to the unique harmonic frequencies with the same base period, T, and unique frequencies for the component sinusoids [2].

To prevent the summation amplitude from becoming exceedingly large, the following relative peak factor (RFP) cost function,

$$RFP(\boldsymbol{u}_j) = \frac{\left[max(\boldsymbol{u}_j) - min(\boldsymbol{u}_j)\right]/2}{2\sqrt{2}\, rms(\boldsymbol{u}_j)},$$

is applied to select the phase frequency of each sinusoidal component of Eq. (31). The RFP represents the the effectiveness of the PTI signal to develop dynamic responses, while minimizing the amount of energy input to the system. By applying the PTI signals to the AUT in the 12ft low speed wind tunnel, while the aircraft is commanded through a variety of pitch angle commands, a nonlinear dynamic model is calculated.

The following section details the application of fuzzy logic control of the aircraft through an initial period where the aircraft dynamic model is unknown. After a short period, a nonlinear model for the control axis is calculated and linear approximation is used to automatically apply proportional-derivative feedback control in place of the fuzzy logic controllers.

4.5 F2P CONTROL WITH RTGM MODELING

The goal of the Robust Learning Control team is to maintain aircraft stability and adequate performance while the RTGM algorithm collects enough initial flight data to provide an aerodynamic model. Once the initial aerodynamic model is developed, flight control is switched to a PD model-based controller, with gains calculated to place the closed loop poles in the same location of a desired reference model. The model-less fuzzy logic controller is used to control the aircraft within areas of the flight envelope which have not been previously modeled. Figure 40 is the wind tunnel test result of the free-to-pitch configuration, where the initial phase of flight is controlled with the IT-2 fuzzy logic controller described within Chapter 3.2. The IT-2 fuzzy logic controller maintains stable flight with the insertion PTI signals added to the controller output. The developed model is implemented into a PD model-based controller, and the aircraft is commanded through the explored flight envelope.

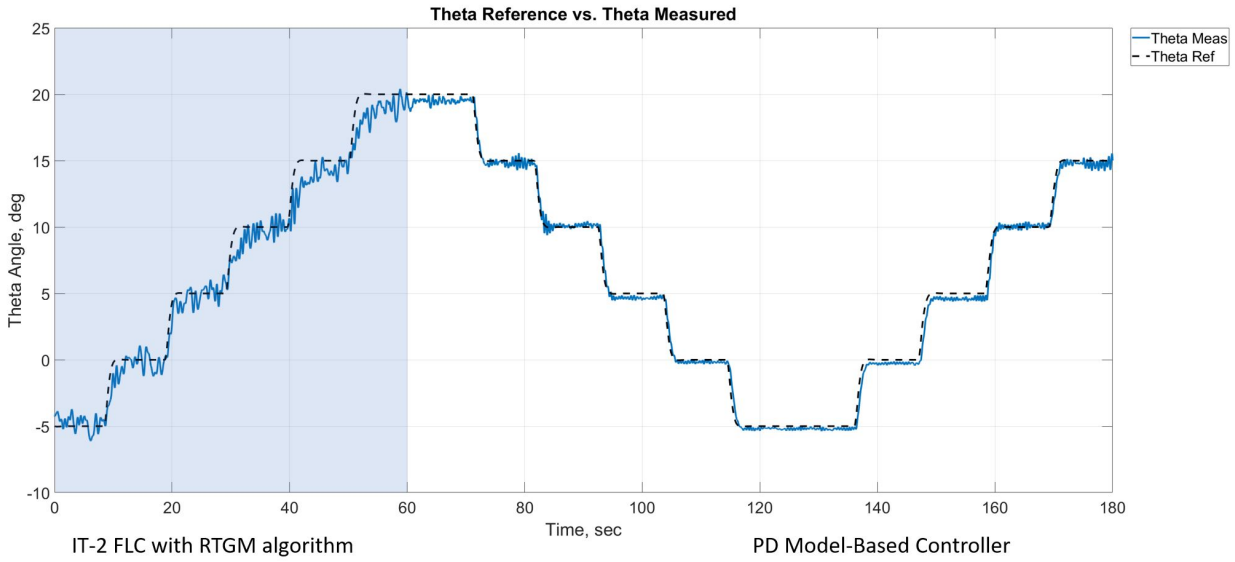

Fig. 40: IT-2 Fuzzy Logic Control with RTGM Algorithm and Transition to PD Control.

The application of a IT-2 model-less aircraft controller to maintain stability during the RTGM algorithm learning process was successfully demonstrated in a one DOF wind tunnel rig configuration. The proposed controllers for pitch and roll were also capable of running in parallel, while only providing control in one DOF, without causing process bottle necks. This provided validation that this controller configuration can be used for future two DOF wind tunnel rig configurations. The nonlinear global model generated from the RTGM algorithm was used to approximate local linear models around the varying tested pitch angles, and the proportional and derivative gains were automatically generated by the controller. Using the PD controller, the aircraft was able to track the test step inputs throughout the entire explored flight envelope.

4.6 DISCUSSION OF FINDINGS

Type-1 and Interval Type-2 TSK fuzzy logic controllers were developed to provide robust flight control system without prior complete knowledge of the aircraft mathematical model and aerodynamics. The performance of these controllers were compared in free-to-pitch and free-to-roll configurations in the 12ft Low Speed Wind Tunnel at NASA Langley Hampton, VA. Through the use of knowledge of general flight control law, reasonable maximum error and change in error values for pitch and roll, and the expected actuator ranges and rates, a

rule base was developed with input and output scaling factors for a generic set of membership functions. This approach simplified the design process, minimizing the need to tune the controllers and could be applied to many aircraft. From the test results, it is observed that the Interval Type-2 fuzzy logic controllers outperforms the similarly designed Type-1 fuzzy logic controllers in free-to-pitch and free-to-roll configurations. The steady state errors are reduced, and the controller outputs contain less high frequency oscillations. Both controller configurations maintained the goal of adequately stabilizing the aircraft, and it is shown that the IT-2 pitch controller maintains adequate stability while the RTGM algorithm collected initial flight data prior to transitioning to a model-based PD controller.

CHAPTER 5

CONCLUSION

The MCAAD group at NASA Langley Research Center is developing techniques for Real-Time Global Modeling and Robust Learning Control for NASA's Transformational Tools and Technologies Project. The group is tasked with developing a systematic approach to reduce the iterative nature of aircraft design by introducing a model-less control law and enabling in-fight aerodynamic modeling and control to take place. A model-less fuzzy logic control law was developed using heuristic expert knowledge and known physical attributes of the tested aircraft. The developed fuzzy logic controller allowed modelling of the aircraft to take place and transition to a model-based proportional-derivative control law based on the learned nonlinear aerodynamics and a calculated linear model.

The proposed controller consisted of a two channel FIS architecture of absolute and incremental outputs for pitch control and a single channel FIS architecture of absolute output for roll control. This thesis investigated the design and implementation of Type-1 and Interval Type-2 (IT-2) fuzzy inference systems to the pitch and roll controller architectures. Both controllers utilized a rule base generated from heuristic expert knowledge of aircraft control laws, and scaling gains selected based on known physical ranges and attributes of the tested aircraft. This controller design reduced the amount of parameters required to tune the controllers prior to testing. A comparison of the performance and robustness characteristics between two similarly configured fuzzy logic controllers for aircraft in free-to-pitch and free-to-roll configurations was presented.

Nonlinear F16 simulations and wind tunnel testing were preformed to compare similarly designed Type-1 and Type-2 fuzzy logic controllers for free-to-pitch and free-to-roll configurations. Each test consisted of a series of step inputs, which commanded the aircraft through a wide range of the flight envelope in the pitch and roll axis. These tests compared the system response as well as the controller outputs. It was found that both types of controllers had similar system response characteristics, and the Type-2 controller output exhibited reduced high frequency oscillations as compared to the Type-1 controller. A final test was preformed in the free-to-pitch configuration, where the RTGM algorithm was used to calculate a nonlinear model of the pitching aircraft dynamics. This model was used to approximate linear models at various trim angles within the test envelope and was used to

calculate the gains for a classical PD controller. It was demonstrated that the IT-2 fuzzy logic controller was able to maintain stable flight conditions throughout the test envelope during the learning and transition process.

Future configuration of the aircraft controller will require multiple degrees of freedom tested simultaneously, when the aircraft is installed in a 2-DOF and 3-DOF mounting rig. The presented pitch and roll controllers will be used in parallel to accomplish 2-DOF control. In addition to extending the test configurations, additional work is required to understand the envelope of stability for these type of model-less flight controllers.

REFERENCES

[1] E. A. Morelli and V. Klein, *Aircraft System Identification - Theory and Practice*. Williamsburg, VA: Sunflyte Enterprises, 2 ed., December 2016.

[2] E. Morelli, "Autonomous real-time global aerodynamic modeling in the frequency domain," AIAA Atmospheric Flight Mechanics Conference, AIAA SciTech Forum, 2020.

[3] S. E. Riddick, "An overview of nasa's learn-to-fly technology development," AIAA Atmospheric Flight Mechanics Conference, AIAA SciTech Forum, 2020.

[4] K. A. Benjamin, "Model-less fuzzy logic control for the nasa modeling and control for agile aircraft development program," Master's thesis, Old Dominion University, Norfolk, Virginia USA, 2018.

[5] E. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies 7, No. 1, pp. 1–139*, January 1975.

[6] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," *IEEE Transactions on Systems, Man, and Cybernetics, Vol. 15, pp. 116-132*, 1985.

[7] M. Sugeno and G. Kang, "Structural identification of fuzzy model," *Fuzzy Sets and Systems, Vol. 26, pp. 15-33*, 1988.

[8] K. M. Passino and S. Yurkovich, *Fuzzy Control*, ch. 1, 2. California: Addison Wesley Longman, Inc, 1 ed., 1998.

[9] The MathWorks, Inc., *MATLAB and Fuzzy Logic Toolbox Release 2018b*, Natick, Massachusetts, United States.

[10] J. M. Mendel, H. Hagras, W.-W. Tan, W. W. Melek, and H. Ying, *Introduction To Type-2 Fuzzy Logic Control*, ch. 1. New Jersey: John Wiley & Sons, Inc, 1 ed., 2014.

[11] B. C. Kuo, *Automatic Control Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1 ed., 1991.

[12] N. S. Nise, *Control Systems Engineering*. Hoboken, NJ: John Wiley & Sons, 7 ed.

[13] A. Taskin and T. Kumbasar, "An open source matlab/simulink toolbox for interval type-2 fuzzy logic systems," *IEEE Symposium Series on Computational Intelligence*, 2015.

[14] M. R. Napolitano, J. L. Casanova, and M. Innocenti, "On-line learning neural and fuzzy logic controllers for actuator failure accommodation in flight control systems," *AIAA-97-3539*, 1997.

[15] R. S. Russell and University of Minnesota, *Non-linear F-16 Simulation using Simulink and Matlab, Version 1.0*, June 22, 2003.

[16] B. Stevens and F. Lewis, *Aircraft Control and Simulation*. New York: Wiley Interscience, 1 ed., 1992.

[17] B. Carlson, *Communication Systems: an Introduction to Signals and Noise in Electrical Communications*. New York: McGraw-Hill, 5 ed., 1968.

[18] Arduino, *Arduino Mega 2560*.

[19] Seed Technology, *W5500 Ethernet Shield*.

[20] Adafruit, *PCA9685 16x12 Bit PWM Servo Shield*.

[21] PJRC LLC., *Teensy 3.5 USB Development Board*.

[22] DROK, *180051US Numerical Control Voltage Regulator*.

[23] US Digital, *MA3 Miniature Absolute Magnetic Shaft Encoder*.

[24] Krohn-Hite Corporation, *0.1Hz to 200kHz Four Channel 4–Pole Filter*, Model 3364 datasheet.

[25] E. Morelli, "Real-time global nonlinear aerodynamic modeling for learn-to-fly," AIAA Atmospheric Flight Mechanics Conference, AIAA SciTech Forum, 2016.

[26] E. Morelli, "Practical aspects of real-time modeling for the learn-to-fly concept," 2018 Atmospheric Flight Mechanics Conference, AIAA AVIATION Forum, 2018.

APPENDIX A

ELECTRO-MAGNETIC BALL LEVITATION SYSTEM

MATLAB SIMULATION

```
warning('off','all')
clear all;
%Magnetic Levitation Problem

%Equations of Motion:

%1) M*d^2y(t)/dt^2 = M*g - i^2(2)/y(t)
%2) v(t) = R*i(t) + L *di(t)/d(t)

% y(t) is the ball position (plant output, state)
M = 0.1; % M is the mass of the ball, 0.1 kg
g = 9.81; % g is the gravitational acceleration constant, 9.8 m/s^2
R = 50; % R is the winding resistance, 50 Ohms
L = 0.5; % L is the winding inductance, 0.5 Henrys
% v(t) is the applied voltage (plant input)
% i(t) is the winding current (plant state)
H0 = 0.15; %Equilibrium position of the ball

%Additional Plant State: di(t)/dt

%% State Equations:
%x1 = y(t), x3 = i(t)

%1) dx1/dt = x2
%2) dx2/dt = g - x3^2/(M*x1)
%3) dx3/dt = -(R/L)x3 + v(t)/L

%State Space Model
%dx/dt = Ax +Bu
%y = Cx + Du

ep = H0; %midway point
```

```
x10 = ep;
x20 = 0;
x30 = sqrt(g*M*x10);
v0 = R*x30;

%    [df1/dx1 df1/dx2 df1/dx3]
%A = [df2/dx1 df2/dx2 df2/dx3]
%    [df3/dx1 df3/dx2 df3/dx3]
%Partial derivatives are evaluated at the equilibrium point values

A = [0 1 0;...
    (x30^2)/(M*x10^2) 0 -2*x30/(M*x10);...
    0 0 -R/L];

%B = [df1/dv;df2/dv;df3/dv]
%Partial derivatives are evaluated at the equilibrium input value
B = [0;0;1/L];

%Output Equation (sensor for state of the ball height (x1)
C = [1 0 0];
D = 0;

%Build state space system
sys = ss(A,B,C,D);

%% Design PD Controller

% v(t) = Kp*e + Kd*ed

%where e = R - H, R is the reference signal, H is the measured height of
    the ball

%Closed Loop State Equations

% dx1/dt = x2
% dx2/dt = g - x3^2/(M*x1)
% dx3/dt = -(R/L)x3 + (Kp*e + Kd*ed)/L
% ...
% dx3/dt = -(R/L)x3 + (Kp*(xr-x1) + Kd*(-x2))/L
% ...
% dx3/dt = -(R/L)x3 + Kp*xr/L - Kp*x1/L - Kd*x2/L
```

```
73
74 % Equillibrium Linearization with PD controller:
75
76 % det(SI - A) = 0
77
78 % det(SI - A) = [s -1 0;...
79 %    -(x30^2)/(M*x10^2) s 2*x30/(M*x10);...
80 %    Kp/L Kd/L s+R/L];
81
82 % = s^3 + (R/L)s^2 - (x30^2/(M*x1e^2) - 2*x30/(L*M*x10)*Kd*s
83 %    - 2*Kp*x30/(L*M*x10) - x30^2*R/m*x10^2*L = 0
84
85 % ROUTH-HURWITZ CRITERION
86
87 % s^3   1                        -x30^2/(M*x10^2) - 2*x30/(M*L*x10)*Kd
88 % s^2   R/L                   -x30^2*R/(m*L*x10^2) -2*Kp*x30/(M*L*x10)
89 % s^1   -2*x30/(L*M*x10)*Kd + 2(Kp*x30/(M*R*x10)                    0
90 % S^0   -x30^2*R/(m*L*x10^2) - 2*x30/(M*L*x10)*Kp                   0
91
92 % For stable closed loop stability, the first column must be positive,
93 % therefor
94
95 %Kd < (L/R)*Kp
96 %Kp < (-x30*R)/(2*x10)
97
98 scale = 8; %scalar to move away from stability critical points
99 Kp = (-x30*R)/(2*x10)*scale;
100 Kd = (L/R)*Kp*scale;
101 N = 1000; %PD controller filter coefficient
102
103 %% Stability
104 psys = tf(sys); %Plant transfer function
105 pdsys = tf([Kd Kp],1); %PD controller transfer function
106 olsys = series(pdsys, psys); %Open Loop
107
108 %Nyquist Plot
109 figure (3);
110 set(gcf, 'Position',  [100, 100, 1500, 800])
111 nyquist(olsys);
112 set(gcf,'color','w');
113
```

```
%Gain and Phase Margins
figure (4);
set(gcf, 'Position',  [100, 100, 1500, 800])
margin(olsys);
set(gcf,'color','w');

%% PD Control Simulations
cmdvalue = ep;

%Set Initial Start point away from Equilibrium Point
x10 = ep + .05; %Modify starting position
M = 1.25*M; %Modify bearing weight from model
LinSimPD = sim('MagLevLoFiSim');

%% PD Control Plots
figure (1)
grid;
set(gcf, 'Position',  [100, 100, 1500, 800])
yyaxis left
plot(LinSimPD.inputSignalPD.time, LinSimPD.inputSignalPD.signals.values,
    LinSimPD.y_PD.time, LinSimPD.y_PD.signals.values,'LineWidth',2);
axis([0 max(LinSimPD.PDctrl.time) 0.10 .25]);
yl = ylabel('Postion (m)');
set(yl, 'Interpreter', 'latex');

yyaxis right
plot(LinSimPD.PDctrl.time, LinSimPD.PDctrl.signals.values,'LineWidth',2);
axis([0 max(LinSimPD.PDctrl.time) 10 30]);
y2 = ylabel('Voltage (V)');
set(y2, 'Interpreter', 'latex');

l1 = legend('Reference Signal', 'Plant Response','Controller Output');
set(l1, 'Interpreter', 'latex');
set(gcf,'color','w');
% Get handle to current axes.
ax = gca;
ax.FontSize = 24;
xl = xlabel('Time (s)');
set(xl, 'Interpreter', 'latex');
%% Output Max Error / Rate Data For Fuzzy Controller Design
%This section is analysing the PD control response data, to find maximum
```

```
%rates of the plan response. This data will be used to select the scaling
%gains for the fuzzy logic controller.
errorRate = diff(LinSimPD.errorSignalPD.signals.values,1);
errorTimeRate = diff(LinSimPD.errorSignalPD.time,1);

Ke = 1/max(abs(LinSimPD.errorSignalPD.signals.values)); % Input 1 Gain
Kde = 1/max(abs(errorRate./errorTimeRate)); % Input 2 Gain
Ko = max(LinSimPD.PDctrl.signals.values);  % Output Gain

%% Fuzzy Logic Control Simulations
fis = readfis('PDFuzzyController3');
LinSimFuzzy = sim('MagLevLoFiSimFuzzy');

%% Fuzzy Plots
figure (2)
set(gcf, 'Position',  [100, 100, 1500, 800])
yyaxis left
plot(LinSimFuzzy.inputSignalFuzzy.time, LinSimFuzzy.inputSignalFuzzy.
   signals.values, LinSimFuzzy.y_Fuzzy.time, LinSimFuzzy.y_Fuzzy.signals.
   values,'LineWidth',2);
axis([0 max(LinSimFuzzy.inputSignalFuzzy.time) 0.1 .25]);
yl = ylabel('Postion (m)');
set(yl, 'Interpreter', 'latex');

yyaxis right
plot(LinSimFuzzy.Fuzzyctrl.time, LinSimFuzzy.Fuzzyctrl.signals.values,'
   LineWidth',2);
axis([0 max(LinSimFuzzy.inputSignalFuzzy.time) 10 30]);
y2 = ylabel('Voltage (V)');
set(y2, 'Interpreter', 'latex');

l2 = legend('Reference Signal', 'Plant Response','Controller Output');
set(l2, 'Interpreter', 'latex');
set(gcf,'color','w');
ax = gca;
ax.FontSize = 24;
xl = xlabel('Time (s)');
set(xl, 'Interpreter', 'latex');
grid;
```

APPENDIX B

F16 SIMULATION MATLAB PROGRAM

B.1 MAIN PROGRAM

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% F16 Simulation Main Program
% Main.m
% The Main file initializes the simulation, runs the call functions for
% sampling the F16 simulation and calculating controller outputs, and
% stores the simulation data at a 50 Hz frequency.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear; clc;
warning('off','all')
%Add Initialization File

%% Set Simulation Control Input Delay & Noisy Data
delay = 0.02; %Simulate Actuator Delay
noisyData = 1; %Noise Toggle

%% Set Model Initial Parameters
%Model Input Parameters
dt = 0.02;   %Time step @50 hz
x = [0 0 0 0 0 0 0 0 0 0 0 0]';%Initial State Vector
u  = [0 0 0 0 0 0]';%Initial Input Vector [ele ail rud elecmd ailcmd
   rudcmd]
i = 1; % sets iteration counter
t = 0; %Current Time
time = 0;

open('F16Block');

% Initial Trim Values
velocity = 700;
altitude = 15000;
```

```
%F16 Input Parameters: 'LF' || 'HF', 'level' || 'pullup' || 'roll'
%F16 Function Output: structure with 'trim_state','trim_thrust','
    trim_control','fi_flag_Simulink', 'type'
F16 = plantParams( velocity, altitude, 'LF', 'level' );

%Set for zero initial starting pitch
F16.trim_state = [0 0 altitude 0 0 0 velocity 0 0 0 0 0];

% Initial Disturbance Values
DisEle = [0, 0, 0];% Elevator
DisAil = [0, 0, 0];% Aileron
DisRud = [0, 0, 0];% Rudder

%Initial State Values
elevator = F16.trim_control(1);
aileron = F16.trim_control(2);
rudder = F16.trim_control(3);

%Leading Edge Flag Initial State
dLEF = 0;%Low Fidelity Case

%% System Input delay (seconds)
u_pitch = elevator*ones(1,delay/dt+1); %Pitch plant input from Pitch
    Controller
u_roll = aileron*ones(1,delay/dt+1); %Roll plant input from Roll
    Controller

%% Command Sequence
%SignalGenerator([PAmp], [IAmp], [PSeq], Resolution, CommandLength,
    Saturation)
% PAmp - Perturbation Amplitudes
% IAmp = Initial Sequence Amplitudes
% PSeq = Perturbation Sequence (i.e. [0 -1 1]
% Resolution = Sample Period
% CommandLength = Hold Time at Command

%Step Commands
CommandSequencePitch = SignalGenerator([0 8 0 -8 0 8 0 -8 0 8 0 -8], [0],
    [1], dt, 10, 90);
%CommandSequenceRoll = SignalGenerator([0 20 0 -20 0 20 0 -20 0 20 0 -20],
     [0], [1], dt, 10, 90);
```

```
67
68 %CommandSequencePitch = zeros(1,length(CommandSequenceRoll));
69 CommandSequenceRoll = zeros(1,length(CommandSequencePitch));
70 CommandSequenceYaw = zeros(1,length(CommandSequencePitch));
71 CommandSequence = [CommandSequencePitch;CommandSequenceRoll;
       CommandSequenceYaw];
72
73 %% Initialize Controller Structure
74 [pitchData, rollData, yawData, Pitch_FISa, Pitch_FISi, Roll_FIS, Log,
       refModel, PitchOn, RollOn, controlDirection] = INITIALIZATION( dt,
       elevator, aileron, rudder, length(CommandSequence), CommandSequence,
       0);
75
76 %Create Matrix FIS
77 matPitch_FISi = struc2mat(Pitch_FISi);
78 matPitch_FISa = struc2mat(Pitch_FISa);
79 matRoll_FIS = struc2mat(Roll_FIS);
80
81 %% Model Simulation
82 fprintf('\nBeginning Simulation:\t%d Iterations\n', length(CommandSequence
       ));
83
84 %% Create Sensor Noise
85 noisePower = 0.02;
86 tlength = length(CommandSequencePitch);
87
88 % Generate White Noise with Variance of 1
89 noiseSig = sim('noiseSignal');
90
91 % 2.5% Signal Noise for Rate and Attitudes (SNR = 20)
92 SNR = 20;
93
94 %Beta scales the noise signal power for a SNR of 1600
95 beta1 = sqrt(sum(refModel.sysRefThetaDTY(1,:).^2) / sum(noiseSig.Noise.
       Data.^2) / SNR);
96 beta2 = sqrt(sum(refModel.sysRefThetaDTY(2,:).^2) / sum(noiseSig.Noise.
       Data.^2) / SNR);
97 beta3 = sqrt(sum(refModel.sysRefPhiDTY(1,:).^2) / sum(noiseSig.Noise.Data
       .^2) / SNR);
98 beta4 = sqrt(sum(refModel.sysRefPhiDTY(2,:).^2) / sum(noiseSig.Noise.Data
       .^2) / SNR);
```

```
99
100 %% Simulation Loop
101 tic; % Start Timer
102
103 while (1)
104     try
105         u(4) = CommandSequence(1,i);
106         u(5) = CommandSequence(2,i);
107         u(6) = CommandSequence(3,i);
108     catch
109         break
110     end
111
112     TStart = time;
113     TFinal = TStart + dt;
114
115     % Use for speed up time
116     time = toc+dt;
117
118
119     %% F16 Model Plant
120     %Run F16 Simulation for period dt
121     sim('F16Block');
122     timer = tic;
123     %Assign Final State Values
124     F16.trim_state = y_sim_rad(end,1:12);
125
126     %Store State Variables as Controller Input Variables
127     x = y_sim(end,:)';
128     % x(5) Pitch (Theta)
129     % x(4) Roll (Phi)
130     % x(6) Yaw (Psi)
131     % x(11) Pitch Rate (ThetaDot)
132     % x(10) Role Rate (PhiDot)
133     % x(12) Yaw Rate (PsiDot)
134
135
136     elevator = F16.trim_control(1);
137     aileron = F16.trim_control(2);
138     rudder = F16.trim_control(3);
139
```

```
    %% Added Simulation Noise
    if noisyData(1) == 1
        pitchData.noiseValue = noiseSig.Noise.Data(i);
        rollData.noiseValue = noiseSig.Noise.Data(i);

        xhat(5) = x(5) +  pitchData.noiseValue * beta1; %Theta
        xhat(11) =  x(11) + pitchData.noiseValue * beta2; %ThetaDot
        xhat(4) = x(4) + rollData.noiseValue * beta3; %Phi
        xhat(10) = x(10) + rollData.noiseValue * beta4; %PhiDot
        try
            noise = [noiseSig.Noise.Data(i-1), noiseSig.Noise.Data(i),
    beta1, beta2];
        catch
            noise = [0, noiseSig.Noise.Data(i), beta1, beta2];
        end

    else
        xhat(5) = x(5) ; %Theta
        xhat(11) =  x(11); %ThetaDot
        xhat(4) = x(4); %Phi
        xhat(10) = x(10); %PhiDot
        noise = [0, 0, 0, 0];
    end

    %% Calculate Error
    [refModel, pitchData, rollData, yawData] = mainController(...
        refModel,...%Reference Model Data
        pitchData,...%Pitch Data Queues
        rollData,...%Roll Data Queues
        yawData,...%yaw Data Queues
        xhat(5),...%Theta
        xhat(11),...%ThetaDot
        xhat(4),...%Phi
        xhat(10),...%PhiDot
        x(6),...%Psi
        x(12),...%PsiDot
        u,...%Input Vector
        time,...%Current Time (was timeVar)
        elevator,...%Previous Elevator Command
        aileron,...%Previous Aileron Command
        rudder,...%Previous Rudder Command
```

```
180         i,...%index
181         noise);%Simulate Noisy Sensor Data
182 
183     %% Controller Architecture
184     if PitchOn == 1
185         [D_El, pitchData, Pitch_FISa, matPitch_FISa, Pitch_FISi,
    matPitch_FISi] = fuzzyPitchController(...
186             pitchData, ... %Pitch Data Queues
187             Pitch_FISa,...%Incremental Pitch FIS Data
188             matPitch_FISa,...%Incremental Pitch FIS Data
189             Pitch_FISi,...%Absolute Pitch FIS Data
190             matPitch_FISi,...%Absolute Pitch FIS Data
191             controlDirection);
192     else
193         D_El = elevator;
194     end
195 
196     if RollOn == 1
197         [D_Ail, rollData] = fuzzyRollController(...
198             rollData, ... %Roll Data Queues
199             Roll_FIS,...
200             matRoll_FIS); %Roll FIS Data
201     else
202         D_Ail = aileron;
203     end
204 
205     u_pitch = [u_pitch(2:end) D_El]; %Pitch Controller Output with delay
206     u_roll = [u_roll(2:end) D_Ail]; %Roll Controller Output with delay
207 
208     %Assign Trim Control
209     F16.trim_control(1) = u_pitch(1);%Pitch Control
210     F16.trim_control(2) = -u_roll(1);%Roll Control
211 
212     Log = DataLog(Log, refModel, x, xhat,u, time, elevator, elevator,
    aileron, rudder, pitchData, rollData, yawData, u_pitch, u_roll,
    Pitch_FISa, Pitch_FISi, Roll_FIS, D_El, D_Ail, i);
213 
214     i = i+1; %March the counter
215     time = time + dt;
216     if time>0 % Prevent double zero at start
217         t = [t time];
```

```
    end
    %End Loop
end

%Store Matrix FIS Values Into Structures
Pitch_FISi = mat2struc(matPitch_FISi);
Pitch_FISa = mat2struc(matPitch_FISa);
Roll_FIS = mat2struc(matRoll_FIS);

% End Program
```

B.2 F16 AIRCRAFT FLIGHT INITIALIZATION

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% F16 Plant Parameter Settings
% plantParams.m
% This function sets the plant trim paramters based on the desired
% simulation parameters. This is used in the intialization of the F16
% simulation.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ Plant ] = plantParams( speed, altitude, type, flight )

Plant = struct( ...
    'trim_state', {}, ...
    'trim_thrust', {}, ...
    'trim_control', {}, ...
    'fi_flag_Simulink', {}, ...
    'type', {});

switch type
    case 'LF'
        Plant(1).fi_flag_Simulink = 0;
        switch flight
            case 'level'
                %All rates are zero
                switch speed
                    case 400
                        Plant(1).trim_state = [0 0 altitude 0 0.1358 0 400
    0.1358 0 0 0 0];
                        Plant(1).trim_thrust = 2370.6;
```

```
28                      Plant(1).trim_control = [-3.5079 0 0];
29
30                  case 500
31                      Plant(1).trim_state = [0 0 altitude 0 0.0779 0 500
    0.0779 0 0 0 0]';
32                      Plant(1).trim_thrust = 2120.6;
33                      Plant(1).trim_control = [-2.4607 0 0]';
34
35                  case 600
36                      Plant(1).trim_state = [0 0 altitude 0 0.0465 0 600
    0.0465 0 0 0 0];
37                      Plant(1).trim_thrust = 2164.0;
38                      Plant(1).trim_control = [-2.0282 0 0];
39
40                  case 700
41                      Plant(1).trim_state = [0 0 altitude 0 0.0274 0 700
    0.0274 0 0 0 0];
42                      Plant(1).trim_thrust = 2584.5;
43                      Plant(1).trim_control = [-1.7675 0 0];
44
45                  case 800
46                      Plant(1).trim_state = [0 0 altitude 0 0.0151 0 800
    0.0151 0 0 0 0];
47                      Plant(1).trim_thrust = 3265;
48                      Plant(1).trim_control = [-1.5986 0 0];
49              end
50          case 'pullup'
51              %Pitch Rate is nonzero
52              switch speed
53                  case 400
54                  case 500
55                  case 600
56                  case 700
57                  case 800
58              end
59          case 'roll'
60              %Roll Rate is nonzero
61              switch speed
62                  case 400
63                  case 500
64                  case 600
```

```
65                      case 700
66                      case 800
67                  end
68
69          end
70      case 'HF'
71          Plant(1).fi_flag_Simulink = 1;
72          switch flight
73              case 'level'
74                  %All rates are zero
75                  switch speed
76                      case 400
77                      case 500
78                          Plant(1).trim_state = [0 0 altitude 0 0.0791 0 500
    0.0791 0 0 0 0];
79                          Plant(1).trim_thrust = 21094;
80                          Plant(1).trim_control = [-2.2441 -0.0936 0.0945];
81
82                      case 600
83                      case 700
84                      case 800
85                  end
86              case 'pullup'
87                  %Pitch Rate is nonzero
88                  switch speed
89                      case 400
90                      case 500
91                      case 600
92                      case 700
93                      case 800
94                  end
95              case 'roll'
96                  %Roll Rate is nonzero
97                  switch speed
98                      case 400
99                      case 500
100                     case 600
101                     case 700
102                     case 800
103                 end
104         end
```

```
end
Plant.type = type;
end
```

B.3 SIGNAL GENERATOR

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Signal Generator
% SignalGenerator.m
% This function generates a test input signal based on desired paramters.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function output = SignalGenerator(PAmp, IAmp, PSeq, Resolution,
    CommandLength, Saturation)
%%
% PAmp - Perturbation Amplitudes
% IAmp = Initial Sequence Amplitudes
% PSeq = Perturbation Sequence (i.e. [0 -1 1]
% Resolution = Sample Period
% CommandLength = Hold Time at Command
output = zeros(1, length(PAmp) * length(IAmp) * length(PSeq) *
    CommandLength);
counter = 1;

for indexX = 1:length(PAmp)
    for indexY = 1:length(IAmp)
%         PSeq(1) = IAmp(indexY);
        for indexZ = 1:length(PSeq)
            for index = 1:CommandLength/Resolution
%                 output(counter) = min(PAmp(indexX) * PSeq(indexZ), 25);
                output(counter) = min( ...
                    max( ...
                    PSeq(indexZ) * PAmp(indexX) + IAmp(indexY), -
    Saturation), Saturation);
                counter = counter + 1;
            end
        end
    end
end

t = 0:.02:length(PAmp) * length(IAmp) * length(PSeq) * CommandLength;
```

```
33
34 end
```

B.4 MAIN INITIALIZATION FILE

```
1  % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Initialization File for Fuzzy Logic Controllers
3  % INITIALIZATION.m
4  % This function is the main initialization of the reference model,
5  % controller parameters and controller structures.
6  % Chris Scott 3/11/21
7  % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8  function [ pitchData, rollData, yawData, Pitch_FISa, Pitch_FISi, Roll_FIS,
       Log, refModel, PitchOn, RollOn, controlDirection, selfOrg,
      learningFactor, learningDelay, linkConsequents, meshFIS,
      inputConsequentArray] = INITIALIZATION7( dt, elevator, aileron, rudder
      , dataLength, CommandSequence, feedback )
9  %% Controller Scaling Gains
10 %Pitch Controller FIS Input scalers [incremental gain, absolute gain]
11 errorLimitPitch = [3 30]; %Pitch Input 1 (Error)
12 errorChangeLimitPitch = [10 60];  %Pitch Input 2 (Error Rate)
13 outputGainPitch = [2 24]; %(Inc) max elevator change rate per cycle (50
      cycles/sec, i.e. 0.2 is 10 deg per second) / (Abs) Max elevator
      deflection
14
15 %Roll Controller FIS Input scalers
16 errorLimitRoll =10; %Roll Input 1 (Error)
17 errorChangeLimitRoll = 150; %Roll Input 2 (Error Rate)
18 outputGainRoll = 10.75; % (Abs) Max aileron deflection
19
20 %% Set Control Modes
21 %1-On, 0-Off
22 PitchOn = 1; %Pitch Angle Tracking Control
23 RollOn = 1; %Roll Angle Tracking Control
24 controlDirection = -1; %Negative Up (-1), Negative Down (1)
25 windtunnel = 0;
26
27 %% Initial Trim Parameters
28     initialPitch = 0;
29     initialPitchRate = 0;
30
```

```
if windtunnel == 1 %(Wind Tunnel)
    initialPitch = feedback.aoa;
    initialPitchRate = feedback.q;

    elevator = 0;
    aileron  = 0;
    rudder   = 0;
end

%% Build Reference Model Structure

[refModel, refModelParams] = referenceModel(); %Sets reference model
   structure and parameters
refModel = referenceSetup(refModel, refModelParams, CommandSequence,
   dataLength, initialPitch, initialPitchRate);

%% Calculate Reference Model
fprintf('Building Reference Models... \n');


%% Build Fuzzy Architecture for Pitch and Roll
[pitchData, rollData, yawData, Pitch_FISa, Pitch_FISi, Roll_FIS ] =
   initializeFuzzySets(dt, elevator, aileron, rudder, refModel,
   errorLimitPitch, errorChangeLimitPitch, outputGainPitch,
   errorLimitRoll, errorChangeLimitRoll, outputGainRoll);


%% Build Data Log
Log = initializeDataLog(PitchOn, RollOn, controlDirection, dataLength,
   errorLimitPitch, errorChangeLimitPitch, outputGainPitch);


end
```

B.5 REFERENCE MODEL

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reference Model
% referenceModel.m
% This function creates the reference model structure, which is used to
% filter the input commands before the error and change in error signals
```

```
% are calculated and passed to the FIS controllers.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ refModel, refModelParams ] = referenceModel()
%% Reference Model Characteristics
zeta = [0.85, 0.85, 0.85];%Pitch, Roll, Yaw
wn = [2.5, 2.5, 2.5];%Pitch, Roll, Yaw
referenceFunction = [[zeta(1), wn(1)];[zeta(2), wn(2)];[zeta(3), wn(3)]];%
    Pitch, Roll, Yaw
dt = 0.02;

refModelParams = struct();

refModelParams(1).zeta = zeta;
refModelParams.wn = wn;
refModelParams.referenceFunction = referenceFunction;
refModelParams.dt = dt;

refModel = struct();

    refModel(1).sysRefThetaDT = [];
    refModel.sysRefThetaDTX = [];
    refModel.sysRefThetaDTY = [];
    refModel.sysRefThetaTs = [];
    refModel.sysRefPhiDT = [];
    refModel.sysRefPhiDTX = [];
    refModel.sysRefPhiDTY = [];
    refModel.sysRefPhiTs = [];
    refModel.sysRefPsiDT = [];
    refModel.sysRefPsiDTX = [];
    refModel.sysRefPsiDTY = [];
    refModel.sysRefPsiTs = [];
end
```

B.6 REFERENCE SETUP

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reference Setup
% referenceSetup.m
% This function calculates and stores the reference model for the current
% test run.
```

```
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [refModel] = referenceSetup(refModel, params, CommandSequence,
    dataLength, initialTheta, initialPitchRate)

WN    = params.wn; % deg/s
zeta = params.zeta;
samplePeriod = params.dt;

%%Pitch Reference Model

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

sysRefThetaCT = ss([sys1; sys2; sys3]);
sysRefThetaCT.StateName = {'Theta' 'ThetaDot'};
sysRefThetaCT.StateUnit = {'deg' 'deg/s'};
sysRefThetaCT.InputName = 'Theta Command';
sysRefThetaCT.InputName = 'Theta Com';
sysRefThetaCT.OutputName = {'Theta' 'ThetaDot' 'ThetaDotDot'};

%Find P from the ARE
syslm = ss(sys1)
%Define Q:
Q = [1 0;0 1];
%Am'*P + P*Am - PBB'P = -Q
refModel.P = care(syslm.A,syslm.B,Q);


refModel.sysRefThetaDT = c2d(sysRefThetaCT, samplePeriod);
%refModel.sysRefThetaDTX = [[0;initialTheta/refModel.sysRefThetaDT.c(1,2)]
    zeros(2,dataLength-1)]; %zeros(2,1);
%refModel.sysRefThetaDTY = zeros(3,dataLength); %zeros(3,1);
refModel.sysRefThetaTs = stepInfo.SettlingTime;

[refModel.sysRefThetaDTY,time,refModel.sysRefThetaDTX] = lsim(refModel.
    sysRefThetaDT,CommandSequence(1,:),[],[0,initialTheta/refModel.
    sysRefThetaDT.c(1,2)]);
refModel.sysRefThetaDTY = refModel.sysRefThetaDTY';
```

```
refModel.sysRefThetaDTX = refModel.sysRefThetaDTX';


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

sysRefPhiCT = ss([sys1; sys2; sys3]);
sysRefPhiCT.StateName = {'Phi' 'PhiDot'};
sysRefPhiCT.StateUnit = {'deg' 'deg/s'};
sysRefPhiCT.InputName = 'Phi Command';
sysRefPhiCT.InputName = 'Phi Com';
sysRefPhiCT.OutputName = {'Phi' 'PhiDot' 'PhiDotDot'};

refModel.sysRefPhiDT = c2d(sysRefPhiCT, samplePeriod);
% refModel.sysRefPhiDTX = zeros(2,1);
% refModel.sysRefPhiDTY = zeros(3,1);
refModel.sysRefPhiTs = stepInfo.SettlingTime;

[refModel.sysRefPhiDTY,time,refModel.sysRefPhiDTX] = lsim(refModel.
    sysRefPhiDT,CommandSequence(2,:),[],[0,0]);%Update for future 3DOF
    initial conditions
refModel.sysRefPhiDTY = refModel.sysRefPhiDTY';
refModel.sysRefPhiDTX = refModel.sysRefPhiDTX';

%% Yaw Reference Model
sys1 = tf([WN(3)^2],      [1 2*zeta(3)*WN(3) WN(3)^2]); %#ok
sys2 = tf([WN(3)^2 0],    [1 2*zeta(3)*WN(3) WN(3)^2]);
sys3 = tf([WN(3)^2 0 0], [1 2*zeta(3)*WN(3) WN(3)^2]);
stepInfo = stepinfo(sys1);

sysRefPsiCT = ss([sys1; sys2; sys3]);
sysRefPsiCT.StateName = {'Psi' 'PsiDot'};
sysRefPsiCT.StateUnit = {'deg' 'deg/s'};
sysRefPsiCT.InputName = 'Psi Command';
sysRefPsiCT.InputName = 'Psi Com';
sysRefPsiCT.OutputName = {'Psi' 'PsiDot' 'PsiDotDot'};

refModel.sysRefPsiDT = c2d(sysRefPsiCT, samplePeriod);
```

```
refModel.sysRefPsiDTX = zeros(2,1);
refModel.sysRefPsiDTY = zeros(3,1);
refModel.sysRefPsiTs = stepInfo.SettlingTime;

[refModel.sysRefPsiDTY,time,refModel.sysRefPsiDTX] = lsim(refModel.
    sysRefPsiDT,CommandSequence(3,:),[],[0,0]);%Update for future 3DOF
    initial conditions
refModel.sysRefPsiDTY = refModel.sysRefPsiDTY';
refModel.sysRefPsiDTX = refModel.sysRefPsiDTX';
end
```

B.7 INITIALIZE FUZZYSETS

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize Data Log
% initializeFuzzySets.m
% This function initializes a controller fuzzy inference system data
% structure for each of the controllers.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ pitchData, rollData, yawData, Pitch_FISa, Pitch_FISi, Roll_FIS
    ] = initializeFuzzySets( dt, pTrim, rTrim, yTrim, refModel,
    errorLimitPitch, errorChangeLimitPitch, outputGainPitch,
    errorLimitRoll, errorChangeLimitRoll, outputGainRoll)
%% Initialization File for Type-2 Fuzzy Logic Controller
%For pitch and roll control. CMS 3-8-2018

warning('off','Fuzzy:evalfis:InputOutOfRange');
warning('off','all')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Aircraft Model Physical Stops %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alphaLimit = 25;    %'CS' Sets +- Angle of Attack Limit
elevLimit = 30;    % Sets +- elevator limit % Was 40, CMS Changed to 24
    for F16 model

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Build Fuzzy Logic Inference Systems %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Pitch_FISa = initializeFIS('PitchFISa', errorLimitPitch(2),
    errorChangeLimitPitch(2), outputGainPitch(2), dt);%For absolute;
    errorLimitPitch, errorChangeLimitPitch, 24, dt)
Pitch_FISi = initializeFIS('PitchFISi', errorLimitPitch(1),
    errorChangeLimitPitch(1), outputGainPitch(1), dt);%For Incremental
Roll_FIS = initializeFIS('RollFIS', errorLimitRoll, errorChangeLimitRoll,
    outputGainRoll, dt);

%% Load Fuzzy Infernce System From External Files
%Pitch_FISa.FLC = readt2fis('IT1 FIS Pitch - Absolute FIS.t2fis');%Added
    for Fuzzy Only Control (Self Org Absolute) Type-1
Pitch_FISa.FLC = readt2fis('IT2 FIS Pitch - Absolute FIS.t2fis');%Added
    for Fuzzy Only Control (Self Org Absolute) (2-19-19)

%10/8/19 Testing
%Pitch_FISi.FLC = readt2fis('IT1 FIS Pitch - Incremental FIS.t2fis');%Self
     Org Incremental Controller v4 Type-1
Pitch_FISi.FLC = readt2fis('IT2 FIS Pitch - Incremental FIS.t2fis');%Self
    Org Incremental Controller v4 (2-19-19)

% Roll FIS Files 10/8/19 Testing
%Roll_FIS.FLC = readt2fis('IT1 FIS Roll - Absolute FIS.t2fis');%Added for
    Fuzzy Only Control Type 1
Roll_FIS.FLC = readt2fis('IT2 FIS Roll - Absolute FIS.t2fis');%Added for
    Fuzzy Only Control

%% Set Fuzzy Logic Pitch Controller Variable Structure
pitchData = FuzzyControllerConstructor( alphaLimit, elevLimit, dt, pTrim,
    refModel.sysRefThetaTs);

%% Set Fuzzy Logic Roll Controller Variable Structure
rollData = FuzzyControllerConstructor( alphaLimit, elevLimit, dt, rTrim,
    refModel.sysRefPhiTs);

%% Set Fuzzy Logic Roll Controller Variable Structure
yawData = FuzzyControllerConstructor( alphaLimit, elevLimit, dt, yTrim,
    refModel.sysRefPsiTs);
end
```

B.8 INITIALIZE FIS

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize Data Log
% initializeFIS.m
% This function initializes a fuzzy inference system data structure.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ FISSTRUCT ] = initializeFIS( NAME, gainIn1, gainIn2, gainOut,
    dt)
%Fuzzy Inference System Constructor

FISSTRUCT = struct();

FISSTRUCT(1).input1Name = 'e';
FISSTRUCT.input2Name = 'ec';
FISSTRUCT.outputName = 'u';

FISSTRUCT.input1Gain = gainIn1;
FISSTRUCT.input2Gain = gainIn2;
FISSTRUCT.outputGain = gainOut;

FISSTRUCT.NAME = NAME;

FISSTRUCT.FLC = [];

FISSTRUCT.firedRule1 = [];
FISSTRUCT.firedRule2 = [];
FISSTRUCT.firedRule3 = [];
FISSTRUCT.firedRule4 = [];
FISSTRUCT.pFIS = [];
FISSTRUCT.ruleCorrection = [];

FISSTRUCT.scaleInput1Array = FISSTRUCT.input1Gain * ones(1,nearest(1/dt));
    %Scalar array to enable smooth transition
FISSTRUCT.scaleInput2Array = FISSTRUCT.input2Gain * ones(1,nearest(1/dt));
    %Scalar array to enable smooth transition
FISSTRUCT.scale1Index = 1;
FISSTRUCT.scale2Index = 1;

FISSTRUCT.ruleMemory = ones(1,49);

end
```

B.9 READ T2 FIS

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% READFIS Load FIS from disk.
% readt2fis.m
% FISMAT=READFIS('filename') creates a FIS matrix in the
% workspace corresponding to the FIS file 'filename' on disk.
%
% FISMAT=READFIS brings up a UIGETFILE dialog box to assist
% with the name and directory location of the file.
%
% The extension '.t2fis' is assumed for 'filename' if it is not
% already present.
%
% See also WRITEFIS.
% Ned Gulley, 5-10-94, Kelly Liu 4-15-96
% Copyright 1994-2004 The MathWorks, Inc.
% $Revision: 1.36.2.4 $  $Date: 2007/03/28 21:37:12 $
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [out,errorStr]=readt2fis(fileName,pathName)

out=[];
if nargin<1, %This section if is no file is specified when the readt2fis
   function is called. (CMS)
    [fileName,pathName]=uigetfile('*.t2fis','Read FIS');
    if isequal(fileName,0) || isequal(pathName,0)
        % If fileName is zero, "cancel" was hit, or there was an error.
        errorStr='No file was loaded';
        if nargout<2,
            error(errorStr);
        end
        return
    end
    fileName = fullfile(pathName, fileName);
else %This section if is a file name is specified when the readt2fis
   function is called. (CMS)
    if ischar(fileName)
        [pathstr, name, ext] = fileparts(fileName);
        if nargin==2
            pathstr=pathName;
```

```
38          end
39
40          if ~strcmp(ext,'.t2fis') %adds .t2fis extension if this was not
    included in the function input. (CMS)
41              name = [name ext];
42              ext = '.t2fis';
43          end
44          if isempty(name) %error if there was no file name in the input
45             errorStr = 'Empty file name: no file was loaded';
46             if nargout<2,
47                error(errorStr);
48             end
49             return
50          end
51          fileName = fullfile(pathstr,[name ext]);    %Builds full file name
    . (CMS)
52       else
53          error('File name must be specified as a string.')
54       end
55  end
56
57  [fid,errorStr]=fopen(fileName,'r'); %Opens file for reading only
58  if fid<0,
59     if nargout<2,
60        error(errorStr); %Output error if the file was not opened. (CMS)
61     end
62     return
63  end
64
65  % Structure
66  nextLineVar=' ';
67  topic='[System]';
68  while isempty(findstr(nextLineVar,topic)),
69     nextLineVar=LocalNextline(fid); %LocalNextLine function returns the
    next line in the text file.
70  end
71
72  % These are the system defaults in case the user has omitted them
73  Name='Untitled';
74  Type='mamdani';
75  AndMethod='min';
```

```
76 OrMethod='max';
77 ImpMethod='min';
78 AggMethod='max';
79 DefuzzMethod='centroid';
80 TypeRedMethod='Karnik-Mendel';
81 outputType='crisp';
82
83 nextLineVar=' ';
84 % Here we are evaluating everything up till the first "[" bracket
85 % The lines we're eval-ing contain their own variable names, so
86 % a lot of variables, like "Name" and so on, are getting initialized
87 % invisibly
88 while isempty([findstr(nextLineVar,'[Input') findstr(nextLineVar,'[Output'
   )
89       findstr(nextLineVar,'[Rules')]),
90    eval([nextLineVar ';']);
91    nextLineVar=LocalNextline(fid); %LocalNextLine function returns the
   next line in the text file.
92 end
93
94 if strcmp(Type,'sugeno')
95     ImpMethod = 'prod';
96     AggMethod = 'sum';
97 end
98 out.name=Name;
99 out.type=Type;
100 out.andMethod=AndMethod;
101 out.orMethod=OrMethod;
102 out.defuzzMethod=DefuzzMethod;
103 out.impMethod=ImpMethod;
104 out.aggMethod=AggMethod;
105 out.typeRedMethod=TypeRedMethod;
106 out.outputType=outputType;
107
108 % I have to rewind here to catch the first input. This is because
109 % I don't know how long the [System] comments are going to be
110 frewind(fid)
111
112 % Now begin with the inputs
113 for varIndex=1:NumInputs,%Moves through text file to each input
114    nextLineVar=' ';
```

```
115   topic='[Input';
116   while isempty(findstr(nextLineVar,topic)),
117      nextLineVar=LocalNextline(fid); %LocalNextLine function returns the
      next line in the text file.
118   end
119
120   % Input variable name
121   Name=0;
122   eval([LocalNextline(fid) ';']) %LocalNextLine function returns the next
       line in the text file.
123   if ~Name,
124      error(['Name missing or out of place for input variable ' ...
125            num2str(varIndex)]);
126   end
127
128   out.input(varIndex).name=Name; %Assigns input name to out structure
129   % Input variable range
130   Range=0;
131   eval([LocalNextline(fid) ';']) %LocalNextLine function returns the next
       line in the text file.
132   if ~Range,
133      error(['Range missing or out of place for input variable ' ...
134            num2str(varIndex)]);
135   end
136   out.input(varIndex).range=Range; %Assigns universe of discourse (range)
       to out structure
137
138   % Number of membership functions
139   eval([LocalNextline(fid) ';']); %LocalNextLine function returns the
      next line in the text file.
140
141   for MFIndex=1:NumMFs*2,
142       MFIndex2=round(MFIndex/2);
143       if ~( isempty(MFIndex/2) || ~isa(MFIndex/2,'double') || (round(
      MFIndex/2)~=MFIndex/2)  )
144          MFIndex1=1;
145       else
146          MFIndex1=2;
147       end
148      MFStr=LocalNextline(fid); %LocalNextLine function returns the next
      line in the text file.
```

```
149       nameStart=findstr(MFStr,'=');
150       nameEnd=findstr(MFStr,':');
151       MFName=eval(MFStr((nameStart+1):(nameEnd-1))); %Break off Membership
      Name
152       typeStart=findstr(MFStr,':');
153       typeEnd=findstr(MFStr,',');
154       MFType=eval(MFStr((typeStart+1):(typeEnd-1))); %Break off Membership
      Type
155       MFParams=eval(MFStr((typeEnd+1):length(MFStr))); %Break off
    Membership Parameters
156       out.input(varIndex).mf(MFIndex1,MFIndex2).name=MFName; %Assign
    Membership Name to out struct
157       out.input(varIndex).mf(MFIndex1,MFIndex2).type=MFType; %Assign
    Membership Type to out struct
158       out.input(varIndex).mf(MFIndex1,MFIndex2).params=MFParams; %Assign
    Membership Parameters to out struct
159    end
160 end
161
162 % Now for the outputs
163 for varIndex=1:NumOutputs,
164    nextLineVar=' ';
165    topic='Output';
166    while isempty(findstr(nextLineVar,topic)),
167       nextLineVar=LocalNextline(fid); %LocalNextLine function returns the
    next line in the text file.
168    end
169
170    % Output variable name
171    varName=LocalNextline(fid); %LocalNextLine function returns the next
    line in the text file.
172    varName=strrep(varName,'Name','');
173    varName=eval(strrep(varName,'=',''));
174    out.output(varIndex).name=varName; %Assign Output Name to out struct
175
176    % Output variable range
177    rangeStr=LocalNextline(fid); %LocalNextLine function returns the next
    line in the text file.
178    rangeStr=strrep(rangeStr,'Range','');
179    rangeStr=strrep(rangeStr,'=','');
```

```
180   out.output(varIndex).range=eval(['[' rangeStr ']']); %Assign Output
      Universe of Discourse (Range) to out struct
181
182   NumMFsStr=LocalNextline(fid); %LocalNextLine function returns the next
      line in the text file.
183   NumMFsStr=strrep(NumMFsStr,'NumMFs','');
184   NumMFsStr=strrep(NumMFsStr,'=','');
185   NumMFs=eval(NumMFsStr); %Read number of Output Membership Functions
186
187   for MFIndex=1:NumMFs,
188      MFStr=LocalNextline(fid); %LocalNextLine function returns the next
      line in the text file.
189      nameStart=findstr(MFStr,'=');
190      nameEnd=findstr(MFStr,':');
191      MFName=eval(MFStr((nameStart+1):(nameEnd-1))); %Read Output
      Membership Function Name
192
193      typeStart=findstr(MFStr,':');
194      typeEnd=findstr(MFStr,',');
195      MFType=eval(MFStr((typeStart+1):(typeEnd-1))); %Read Output
      Membership Function Type
196
197      %MFParams=eval(MFStr((typeEnd+1):length(MFStr)));
198      oneOutMFParam = findstr(MFStr,' '); %Added by CMS 1/1/19 to remove
      duplicate output MF value
199      MFParams=eval([MFStr((typeEnd+2):(oneOutMFParam-1))]);%Added by CMS
      1/1/19 to remove duplicate output MF value
200
201      if ~strcmpi(MFType,'linear')
202          out.output(varIndex).mf(MFIndex).name=MFName; %Assign Output
      Membership Function Name to out struct
203          out.output(varIndex).mf(MFIndex).type=MFType; %Assign Output
      Membership Function Type to out struct
204          out.output(varIndex).mf(MFIndex).params=MFParams; %Assign Output
       Membership Function Parameters to out struct *** Use this line to
      modify for hypersurface***
205      else
206          out.output(varIndex).mf(MFIndex).name=MFName;
207          out.output(varIndex).mf(MFIndex).type=MFType;
208          out.output(varIndex).mf(MFIndex).params=reshape(MFParams,[length
      (MFParams)/2,2])';
```

```
209       end
210
211    end
212 end
213
214 % Now assemble the whole FIS data matrix
215
216
217
218 % If NumInputs or NumOutputs is zero, we need a space holder for the MF
    indices
219 % Otherwise they'll just be the empty set
220 %if isempty(NumInputMFs), NumInputMFs=0; end
221 %if isempty(NumOutputMFs), NumOutputMFs=0; end
222
223
224
225 % Now for the rules
226 nextLineVar=' ';
227 topic='Rules';
228 while isempty(findstr(nextLineVar,topic)),
229    nextLineVar=LocalNextline(fid); %LocalNextLine function returns the
    next line in the text file.
230 end
231
232
233 % out.rule=[];
234 % ruleIndex=1;
235 % txtRuleList='';
236 %     while ~feof(fid)
237 %         ruleStr=LocalNextline(fid);
238 %         if ischar(ruleStr)
239 %             txtRuleList(ruleIndex,1:length(ruleStr))=ruleStr; %#ok<*
    AGROW>
240 %             ruleIndex=ruleIndex+1;
241 %         end
242 %     end
243
244
245 ruleIndex=1;
246 txtRuleList='';
```

```
out.rule=[];
while ~feof(fid) %Test for end-of-file
   ruleStr=LocalNextline(fid); %LocalNextLine function returns the next
   line in the text file.
   if ischar(ruleStr)
    %txtRuleList(ruleIndex,1:length(ruleStr))=ruleStr;
    txtRuleList(ruleIndex,1:length(ruleStr))=ruleStr; %#ok<*AGROW>
    ruleIndex=ruleIndex+1;
   end
end


if ~isempty(txtRuleList)& isfield(out, 'input') & isfield(out, 'output')
%            & isfield(out.input, 'mf') & isfield(out.output, 'mf') ...
%            & isfield(out.input.mf, 'name') & isfield(out.output.mf, '
   name')
  outRules=parsrule(out,txtRuleList,'format','indexed'); %2018 parsrule is
    causing the problem, extracting only parsed rules
end

out.rule = outRules.rule; %Extract parsed rules (CMS 1/1/19)

fclose(fid);



function outLine=LocalNextline(fid)
%LOCALNEXTLINE Return the next non-empty line of a file.
% OUTLINE=LOCALNEXTLINE(FID) returns the next non-empty line in the
% file whose file ID is FID. The file FID must already be open.
% LOCALNEXTLINE skips all lines that consist only of a carriage
% return and it returns a -1 when the end of the file has been
% reached.
%
% LOCALNEXTLINE ignores all lines that begin with the % comment
% character (the % character must be in the first column)

% Ned Gulley, 2-2-94

outLine=fgetl(fid);

```

```
stopFlag=0;
while (~stopFlag),
   if length(outLine)>0,
      if (~strcmp(outLine(1),'%') | (outLine ==-1)),
         % This line has real content or the end of the file; stop and
   return outLine
         stopFlag=1;
      else
         % This line must be a comment; keep going
         outLine=fgetl(fid);
      end
   else
      % This line is of length zero
      outLine=fgetl(fid);
   end
end;
```

B.10 FUZZY CONTROLLER CONSTRUCTOR

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fuzzy Controller Constructor
% FuzzyControllerConstructor.m
% This function creates the data structure for a fuzzy logic controller.
% This is used to create the Pitch and Roll fuzzy logic controllers.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ FLCStruct ] = FuzzyControllerConstructor( alphaLimit, elevLimit
   , dt, trim, Ts)
%Constructor for Fuzzy Logic Controller Variable Structure

FLCStruct = struct([]);

ssLength  = nearest(1/dt);
FLCStruct(1).samplePeriod = queueCreate(50);
FLCStruct.commandTime = 0;
FLCStruct.ctrlSurfaceLimit = elevLimit;
FLCStruct.angleLimit = alphaLimit;
FLCStruct.Ts = Ts;
FLCStruct.dt = dt;
FLCStruct.noiseValue = 0;

```

```
22 FLCStruct.angleHistory        = queueCreate(ssLength);
23 FLCStruct.angleHistoryFiltered = queueCreate(ssLength);
24 FLCStruct.angleDotHistory     = queueCreate(ssLength);
25 FLCStruct.angleDotDotHistory  = queueCreate(ssLength);
26 FLCStruct.angleCommandHistory = queueCreate(ssLength);
27 FLCStruct.angleErrorHistoryStdScalar = [];
28 FLCStruct.angleErrorDotHistoryStdScalar = [];
29 FLCStruct.angleRef = queueCreate(100);
30
31 FLCStruct.angleError       = queueCreate(ssLength);%FIS input 1
32 FLCStruct.angleErrorDot    = queueCreate(ssLength);%FIS input 2
33 FLCStruct.angleDotError    = queueCreate(ssLength);
34 FLCStruct.angleDotErrorDot    = queueCreate(ssLength);
35 FLCStruct.angleDotDotError = queueCreate(ssLength);
36
37 FLCStruct.ctrlSurfaceHistory = queueCreate(ssLength);
38 FLCStruct.ctrlSurfaceDotHistory = queueCreate(ssLength);
39
40 FLCStruct.dp = queueCreate(3); %damping ratio
41
42 FLCStruct.ssError = queueCreate(3);
43 FLCStruct.direction = 0;
44
45 FLCStruct.triggeredRules = zeros(1,49);
46
47 FLCStruct.ssCount = 0;
48 FLCStruct.dcControl = trim;
49
50 %Performance Data
51 FLCStruct.smallPeaks = []; %Small peaks for noise detection
52 FLCStruct.largePeaks = []; %Large peaks for oscillation detection
53 FLCStruct.maxPeaks = []; %Max peaks for frequency detection
54 FLCStruct.minPeaks = []; %Min peaks for frequency detection
55
56 %Data Smoothing
57 FLCStruct.smoothYData = []; % (not used currently)
58 FLCStruct.smoothElData = []; % (not used currently)
59
60 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
61 %%%%% Command Specific Container%%%%%
62 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
63 FLCStruct.keyMap = trim*[0;1];%Sets initial trim value for 0 command
64 FLCStruct.trimArray = trim*ones(1,20); %Trim array to enable transition
       between key values
65 FLCStruct.trimIndex = 0; %Index for Trim Array
66
67 %Smooth Data for steadystate detection
68 FLCStruct.smoothAngle = [];
69 FLCStruct.smoothAngleDot = [];
70
71 %FIS Rule Firing Data
72 FLCStruct.FR1a = queueCreate(50);
73 FLCStruct.FRS1a = queueCreate(50);
74 FLCStruct.FR2a = queueCreate(50);
75 FLCStruct.FRS2a = queueCreate(50);
76 FLCStruct.FR3a = queueCreate(50);
77 FLCStruct.FRS3a = queueCreate(50);
78 FLCStruct.FR4a = queueCreate(50);
79 FLCStruct.FRS4a = queueCreate(50);
80
81
82 FLCStruct.FR1i = queueCreate(50);
83 FLCStruct.FRS1i = queueCreate(50);
84 FLCStruct.FR2i = queueCreate(50);
85 FLCStruct.FRS2i = queueCreate(50);
86 FLCStruct.FR3i = queueCreate(50);
87 FLCStruct.FRS3i = queueCreate(50);
88 FLCStruct.FR4i = queueCreate(50);
89 FLCStruct.FRS4i = queueCreate(50);
90
91 FLCStruct.FRI = queueCreate(50); %State Feedback Consequent Index
92
93 %Adaptive FIS
94 FLCStruct.ruleChangeSign = zeros(1,49);
95 FLCStruct.lastRuleChangea = []; %Store last rule consequent changes for
       history.
96 FLCStruct.lastRuleChangei = []; %Store last rule consequent changes for
       history.
97
98 %Absolute + Incremental Data
99 FLCStruct.AbsOut = 0;
100 FLCStruct.IncOut = 0;
```

```
end
```

B.11 QUEUE CREATE

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Queue Create
% queueCreate.m
% This function creates a queue structure to store data and calculate
% properties of the data over a time window.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ varStruct ] = queueCreate( size )
%Create Queue Struct

varStruct = struct( ...
    'data', {}, ...
    'maximum', {}, ...
    'minimum', {}, ...
    'spread', {}, ...
    'size', {}, ...
    'last', {}, ...
    'previous', {}, ...
    'dc', {}, ...
    'average', {}, ...
    'change', {}, ...
    'windowMean', {}, ...
    'count', {});

varStruct(1).data = zeros(1, size);
varStruct.maximum = 0;
varStruct.minimum = 0;
varStruct.spread = 0;
varStruct.size = size;
varStruct.last = 0;
varStruct.previous = 0;
varStruct.dc = 0;
varStruct.average = 0;
varStruct.change = 0;
varStruct.windowMean = 0;
varStruct.count = 0;
end
```

B.12 INITIALIZE DATALOG

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize Data Log
% initializeDataLog.m
% This function initializes the Data Log structure.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ DataLog ] = initializeDataLog(PitchOn, RollOn, controlDirection
    , dataLength, errorLimitPitch, errorChangeLimitPitch, outputGainPitch)
%Creates system DataLog for parameter history

DataLog = struct();

DataLog(1).theta = zeros(1,dataLength);%Pitch / Alpha Measured
DataLog.thetaDot = zeros(1,dataLength);%Pitch Rate / AlphaDot Measured
DataLog.Relevator = zeros(1,dataLength);%Right Elevator Angle Measured (
    with delay)
DataLog.Lelevator = zeros(1,dataLength);%Left Elevator Angle Measured (
    with delay)
DataLog.phi = zeros(1,dataLength);%Roll Measured
DataLog.phiDot = zeros(1,dataLength);%Roll Rate Measured
DataLog.aileron = zeros(1,dataLength);%Aileron Angle Measured (with delay)
DataLog.psi = zeros(1,dataLength);%Yaw Measured
DataLog.psiDot = zeros(1,dataLength);%Yaw Rate Measured
DataLog.rudder = zeros(1,dataLength);%Rudder Angle Measured (with delay)
DataLog.thetaRef = zeros(1,dataLength);%Pitch / Alpha Reference
DataLog.thetaDotRef = zeros(1,dataLength);%Pitch Rate / AlphaDot Reference
DataLog.thetaDotDotRef = zeros(1,dataLength);%PitchDot Rate / AlphaDotDot
    Reference
DataLog.phiRef = zeros(1,dataLength);%Roll Reference
DataLog.phiDotRef = zeros(1,dataLength);%Roll Rate Reference
DataLog.phiDotDotRef = zeros(1,dataLength);%RollDot Rate Reference
DataLog.psiRef = zeros(1,dataLength);%Yaw Reference
DataLog.psiDotRef = zeros(1,dataLength);%Yaw Rate Reference
DataLog.psiDotDotRef = zeros(1,dataLength);%YawDot Rate Refernce

DataLog.thetahat = zeros(1,dataLength); %Pitch / Alpha (noisy sensor)
DataLog.thetaDothat = zeros(1,dataLength); %Pitch Rate (noisy sensor)
DataLog.phihat = zeros(1,dataLength); %Roll (noisy sensor)
DataLog.phiDothat = zeros(1,dataLength); %Roll Rate (noisy sensor)
```

```
36
37
38 DataLog.thetaError = zeros(1,dataLength);%Pitch Error // Pitch FIS input1
39 DataLog.thetaErrorDot = zeros(1,dataLength);%Pitch Error rate // Pitch FIS
       input2
40 DataLog.thetaDotError = zeros(1,dataLength);%Pitch Rate Error
41
42 DataLog.phiError = zeros(1,dataLength);%Roll Rate Error // Roll FIS input1
43 DataLog.phiErrorDot = zeros(1,dataLength);%Roll Error Rate // Roll FIS
      input2
44 DataLog.phiDotError = zeros(1,dataLength);%Roll Rate Error
45
46 DataLog.psiError = zeros(1,dataLength);%Yaw Error
47 DataLog.psiErrorDot = zeros(1,dataLength);%Yaw Error Rate
48 DataLog.psiDotError = zeros(1,dataLength);%Yaw Rate Error
49
50 DataLog.pitchCTRL = zeros(1,dataLength);%Pitch Controller Output
51 DataLog.rollCTRL = zeros(1,dataLength);%Roll Controller Output
52 DataLog.yawCTRL = zeros(1,dataLength);%Yaw Controller Output
53 DataLog.time = [];%Time Log
54 DataLog.thetaCommand = zeros(1,dataLength);%Commanded Pitch Angle
55 DataLog.phiCommand = zeros(1,dataLength);%Commanded Roll Angle
56 DataLog.psiCommand = zeros(1,dataLength);%Commanded Yaw Angle
57 DataLog.thetaFISaInput1 = zeros(1,dataLength);%Pitch FIS Input(Normalized
      Pitch Error)
58 DataLog.thetaFISaInput2 = zeros(1,dataLength);%Pitch Rate FIS Input (
      Normalized Pitch Rate Error)
59 DataLog.thetaFISaOutput = zeros(1,dataLength);%Pitch FIS Normalized Output
       (Pre-scaled)
60 DataLog.thetaFISiInput1 = zeros(1,dataLength);%Pitch FIS Input(Normalized
      Pitch Error)
61 DataLog.thetaFISiInput2 = zeros(1,dataLength);%Pitch Rate FIS Input (
      Normalized Pitch Rate Error)
62 DataLog.thetaFISiOutput = zeros(1,dataLength);%Pitch FIS Normalized Output
       (Pre-scaled)
63 DataLog.phiFISInput1 = zeros(1,dataLength);%Roll FIS Input(Normalized Roll
       Error)
64 DataLog.phiFISInput2 = zeros(1,dataLength);%Roll Rate FIS Input (
      Normalized Roll Rate Error)
65 DataLog.phiFISOutput = zeros(1,dataLength);%Roll FIS Normalized Output (
      Pre-scaled)
```

```
DataLog.psiFISInput1 = zeros(1,dataLength);%Yaw FIS Input(Normalized Yaw
    Error)
DataLog.psiFISInput2 = zeros(1,dataLength);%Yaw Rate FIS Input (Normalized
     Yaw Rate Error)
DataLog.psiFISOutput = zeros(1,dataLength); %Yaw FIS Normalized Output (
    Pre-scaled)
DataLog.DEl = zeros(1,dataLength);% Elevator Control Output
DataLog.DAil = zeros(1,dataLength);% Aileron Control Output
DataLog.DRud = zeros(1,dataLength);%Rudder Control Output

%F16 States
DataLog.x1 = zeros(1,dataLength);%npos
DataLog.x2 = zeros(1,dataLength);%epos
DataLog.x3 = zeros(1,dataLength);%altitude
DataLog.x4 = zeros(1,dataLength);%roll
DataLog.x5 = zeros(1,dataLength);%pitch
DataLog.x6 = zeros(1,dataLength);%yaw
DataLog.x7 = zeros(1,dataLength);%speed
DataLog.x8 = zeros(1,dataLength);%angle of attack
DataLog.x9 = zeros(1,dataLength);%side slip
DataLog.x10 = zeros(1,dataLength);%roll rate
DataLog.x11 = zeros(1,dataLength);%pitch rate
DataLog.x12 = zeros(1,dataLength);%yaw rate

% Initialization Parameters
DataLog.PitchOn = PitchOn;
DataLog.RollOn = RollOn;
DataLog.controlDirection = controlDirection;

%Controller FIS Input scalers [incremental gain, absolute gain]
DataLog.errorLimitPitch = errorLimitPitch;
DataLog.errorChangeLimitPitch = errorChangeLimitPitch;
DataLog.outputGainPitch = outputGainPitch;

%Absolute + Incremental Output Data
DataLog.AbsOut = zeros(1,dataLength);
DataLog.IncOut = zeros(1,dataLength);

end
```

B.13 STRUCTURE TO MATRIX

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Structure to Matrix
% struc2mat.m
% This function takes the fuzzy logic structure and creates a multiple
% paged matrix which contains all of the data. This will significantly
% improve the speed of the controller where fuzzy evaluation and self
% organizing is used.
% The goal here is to move all of the values used to evaluate the Fuzzy
% Inference System, without all of the labels and character strings. Any
% of these which are required for FIS evaluation, will be converted into a
% number sequence (i.e. "MF Type" [1,2,3]). At the end of the simulation
% or test run, the final values of the matrix will be saved back into the
% data structure for organization and ease of reference.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ matFIS ] = struc2mat( FIS )


%% The following is the current (3-17-19) data structure format.
%If any other parameters are added to the structure, these will need to be
%added here and added to the matrix.

% Main FIS Structure (Sheet 1)
    %FIS Name [char] -- Matrix(1,1,1,1)
    %input1Name [char] -- Not used within the matrix
    %input2Name [char] -- Not used within the matrix
    %outputName [char] -- Not used within the matrix
    %input1Gain [double] - Matrix(2,1,1,1)
    %input2Gain [double] - Matrix(3,1,1,1)
    %outputGain [double] - Matrix(4,1,1,1)
    %firedRules 1-4 [4x3 matrix] - Matrix([4x3],2,1)
    %ruleMemory [1x49 array] - Matrix([1x49],[2xlength],3,1)
    %variableConsequent [1x3 array] - Matrix(1,[1x3],4,1) [Array, Range,
    %Resolution]
    %pFIS ---> Placed on 4th matrix dimension
    %scaleInput1Array [1x50 array] - Matrix(13,[1x50],1,1)
    %scaleInput2Array [1x50 array] - Matrix(14,[1x50],1,1)
    %scale1Index (double) - Matrix(15,1,1,1)
    %scale1Index (double) - Matrix(16,1,1,1)

    %FLC [multiple sheet matrix]
```

```
42        %Name [char] -- Not used within the matrix
43        %type [char] - Matrix(5,1,1,1) - Convert char to number [sugeno =
    1,mumdani = 2]
44        %andMethod [char] - Matrix(6,1,1,1) - Convert char to number [min
    = 1, product = 2]
45        %orMethod [char] - Matrix(7,1,1,1) - Convert char to number [max =
     1, probor = 2]
46        %defuzzMethod [char] - Matrix(8,1,1,1) - Convert char to number [
    sugeno = 1,mumdani = 2]
47        %impMethod [char] - Matrix(9,1,1,1) - Convert char to number [min
    = 1]
48        %aggMethod [char] - Matrix(10,1,1,1) - Convert char to number [max
     = 1]
49        %typeRedMethod [char] - Matrix(11,1,1,1) - Convert char to number
    ['KM' = 1,'EKM' = 2,'IASC' = 3,'EIASC' = 4,'EODS' = 5,'WM' = 6,'NT' =
    7,'BMM' = 8]
50        %FLC.outputType [char] - Matrix(12,1,1,1) - Convert char to number
     ['icrisp' = 1]
51        %FLC.input [2 matrix sheets]
52            %name -- Not used within the matrix
53            %range -
54                % input 1: Matrix(1,[1x2],5,1)
55                % input 2: Matrix(2[1x2],6,1)
56            %MF [3x14 matrix] [name (char), type (char), params (1x3 array
    )]- Upper (2-8) Lower (9-15)
57                % input 1: type [char] - Matrix([2-15],1,5,1) - Convert
    char to number [trimf = 1,zmf = 2, smf = 3]
58                % input 1: params [1x3] - Matrix([2-15],[2-4],5,1)
59                % input 2: type [char] - Matrix([2-15],1,6,1) - Convert
    char to number [trimf = 1,zmf = 2, smf = 3]
60                % input 2: params [1x3] - Matrix([2-15],[2-4],6,1)
61        %FLC.output [1 sheet matrix]
62            %name -- Not used within the matrix
63            %range - Matrix(1,[1x2],7,1)
64            %MF [3x14 matrix] [name (char), type (char), params (1
    xvariable size array)]
65                % type [char] - Matrix([2-50],1,7,1) - Convert char to
    number ['icrisp' = 1]
66                % params [(1xvariable size array] - Matrix([2-50],[2-size
    ],7)
67        %rule [1 matrix sheet]
```

```
68              % [antecent] [consequent] [weight] [connection] x 49 rules
69                  %antecent [1x2] - Matrix([1-49],[1x2],8,1)
70                  %consequent [double] - Matrix([1-49],3,8,1)
71                  %weight [double] - Matrix([1-49],4,8,1)
72                  %connection [double] - Matrix([1-49],5,8,1)
73
74 %% Construct Main FIS Matrix
75 %Set 1, Sheet 1
76 if strcmp(FIS.NAME, 'PitchFISa')
77     matFIS(1,1,1,1) = 1;
78 elseif strcmp(FIS.NAME,'PitchFISi')
79     matFIS(1,1,1,1) = 2;
80 end
81
82     matFIS(2,1,1,1) = FIS.input1Gain;
83     matFIS(3,1,1,1) = FIS.input2Gain;
84     matFIS(4,1,1,1) = FIS.outputGain;
85
86 %type
87 if strcmp(FIS.FLC.type, 'sugeno')
88     matFIS(5,1,1,1) = 1;
89 elseif strcmp(FIS.FLC.type,'mumdani')
90     matFIS(5,1,1,1) = 2;
91 end
92
93 %andMethod
94 if strcmp(FIS.FLC.andMethod,'min')
95     matFIS(6,1,1,1) = 1;
96 elseif strcmp(FIS.FLC.andMethod,'prod')
97     matFIS(6,1,1,1) = 2;
98 end
99
100 %orMethod
101 if strcmp(FIS.FLC.orMethod,'max')
102     matFIS(7,1,1,1) = 1;
103 elseif strcmp(FIS.FLC.orMethod,'probor')
104     matFIS(7,1,1,1) = 2;
105 end
106
107 %defuzzMethod ['wtaver' = 1]
108     matFIS(8,1,1,1) = 1;
```

```
109
110 %impMethod ['min' = 1]
111     matFIS(9,1,1,1) = 1;
112
113 %aggMethod ['max' = 1]
114     matFIS(10,1,1,1) = 1;
115
116 %typeRedMethod [char] - Matrix(10,1,1) - Convert char to number [sugeno =
    1,'EKM' = 2,'IASC' = 3,'EIASC' = 4,'EODS' = 5,'WM' = 6,'NT' = 7,'BMM'
    = 8]
117
118 switch FIS.FLC.typeRedMethod
119     case 'KM'
120     matFIS(11,1,1,1) = 1;
121     case 'EKM'
122     matFIS(11,1,1,1) = 2;
123     case 'IASC'
124     matFIS(11,1,1,1) = 3;
125     case 'EIASC'
126     matFIS(11,1,1,1) = 4;
127     case 'EODS'
128     matFIS(11,1,1,1) = 5;
129     case 'WM'
130     matFIS(11,1,1,1) = 6;
131     case 'NT'
132     matFIS(11,1,1,1) = 7;
133     case 'BMM'
134     matFIS(11,1,1,1) = 8;
135 end
136
137 for i = 1:length(FIS.scaleInput1Array)
138     matFIS(13,i,1,1) = FIS.scaleInput1Array(1,i);
139 end
140
141 for i = 1:length(FIS.scaleInput2Array)
142     matFIS(14,i,1,1) = FIS.scaleInput2Array(1,i);
143 end
144
145     matFIS(15,1,1,1) = FIS.scale1Index;
146     matFIS(16,1,1,1) = FIS.scale2Index;
147
```

```
148
149 %FLC.outputType ['icrisp' = 1]
150      matFIS(12,1,1,1) = 1;
151 %Set 1, Sheet 2
152 try
153 for i = 1:3
154     matFIS(1,i,2,1) = FIS.firedRule1(1,i);
155 end
156 for i = 1:3
157     matFIS(2,i,2,1) = FIS.firedRule2(1,i);
158 end
159 for i = 1:3
160     matFIS(3,i,2,1) = FIS.firedRule3(1,i);
161 end
162 for i = 1:3
163     matFIS(4,i,2,1) = FIS.firedRule4(1,i);
164 end
165 catch
166 end
167
168 %Set 1, Sheet 3     %ruleMemory [1x49 array] -
169 %Matrix([1x49],[2xlength],3,1)   Rule Memory Counter
170
171     %for i=1:49
172     %        matFIS(i,1:2*FIS.variableConsequentRange/FIS.
    variableConsequentResolution,3,1) = 1;%FIS.ruleMemory(1,i);
173     %end
174 %Set 1, Sheet 4
175
176             %matFIS(1,1,4,1) = FIS.variableConsequentArray;
177             %matFIS(1,2,4,1) = FIS.variableConsequentRange;
178             %matFIS(1,3,4,1) = FIS.variableConsequentResolution;
179
180 %Set 1, Sheets 5/6 [Input 1/2]
181 for inp = 1:2
182 for i=1:2
183     matFIS(1,i,5,1) = FIS.FLC.input(inp).range(i);
184 end
185
186 for i = 2:length(FIS.FLC.input(inp).mf)*2+1
187     if strcmp(FIS.FLC.input(inp).mf(i-1).type, 'trimf')
```

```
188         matFIS(i,1,inp+4,1) = 1;
189         for ii = 2:length(FIS.FLC.input(inp).mf(i-1).params)+1
190             matFIS(i,ii,inp+4,1) = FIS.FLC.input(inp).mf(i-1).params(ii-1)
    ;
191         end
192     elseif strcmp(FIS.FLC.input(inp).mf(i-1).type, 'zmf')
193         matFIS(i,1,inp+4,1) = 2;
194         for ii = 2:length(FIS.FLC.input(inp).mf(i-1).params)+1
195             matFIS(i,ii,inp+4,1) = FIS.FLC.input(inp).mf(i-1).params(ii-1)
    ;
196         end
197     elseif strcmp(FIS.FLC.input(inp).mf(i-1).type, 'smf')
198         matFIS(i,1,inp+4,1) = 3;
199         for ii = 2:length(FIS.FLC.input(inp).mf(i-1).params)+1
200             matFIS(i,ii,inp+4,1) = FIS.FLC.input(inp).mf(i-1).params(ii-1)
    ;
201         end
202     end
203 end
204 end
205
206 %Set 1, Sheet 7 [Output]
207 for i=1:2
208     matFIS(1,i,7,1) = FIS.FLC.output.range(i);
209 end
210 for i = 2:length(FIS.FLC.output.mf)+1
211         matFIS(i,1,7,1) = 1; %['constant' = 1]
212         for ii = 2:length(FIS.FLC.output.mf(i-1).params)+1
213             matFIS(i,ii,7,1) = FIS.FLC.output.mf(i-1).params(ii-1);
214         end
215 end
216
217 %Set 1, Sheet 8 [Rule Base]
218 for i = 1:length(FIS.FLC.rule)
219 matFIS(i,1,8,1) = FIS.FLC.rule(i).antecedent(1);
220 matFIS(i,2,8,1) = FIS.FLC.rule(i).antecedent(2);
221 matFIS(i,3,8,1) = FIS.FLC.rule(i).consequent;
222 matFIS(i,4,8,1) = FIS.FLC.rule(i).weight;
223 matFIS(i,5,8,1) = FIS.FLC.rule(i).connection;
224 end
225
```

```
%Create Rule Consequent Change Sign Tracking, This is not from the FIS
for i = 1:length(FIS.FLC.output.mf)
        for ii = 1:length(FIS.FLC.output.mf(i).params)
            matFIS(i,ii,9,1) = 0;
        end
end

%Create Rule Consequent Change Quantity Tracking, This is not from the FIS
for i = 1:length(FIS.FLC.output.mf)
        for ii = 1:length(FIS.FLC.output.mf(i).params)
            matFIS(i,ii,10,1) = 0;
        end
end
end
```

B.14 MAIN CONTROLLER

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main Controller Function
% mainController.m
% This function processes the current sample data to be sent to each of
% the controllers, and detects when a new command from the test sequence
% is sent to the reference model.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [refModel, pitchData, rollData, yawData] = mainController(
    refModel, pitchData, rollData, yawData , theta, thetaDot, phi, phiDot,
     psi, psiDot, Command, sampleTime, elevator, aileron, rudder, index,
    noise)
%% Set Sample Time
pitchData.samplePeriod = queuePush(pitchData.samplePeriod, sampleTime);
rollData.samplePeriod = queuePush(rollData.samplePeriod, sampleTime);
yawData.samplePeriod = queuePush(yawData.samplePeriod, sampleTime);


%% Store Histories

pitchData = storeHistories(pitchData, theta, thetaDot, Command(4));
rollData = storeHistories(rollData, phi, phiDot, Command(5));
yawData = storeHistories(yawData, psi, psiDot, Command(6));

```

```
%% Detect New Input Command
[pitchData] = newCommand(pitchData);
%[rollData] = newCommand(rollData);
%[yawData] = newCommand(yawData);

%% Calculate Angle(Dot) Error
[pitchData, rollData, yawData] = calculateError(refModel, pitchData,
   rollData, yawData, theta, thetaDot, phi, phiDot, psi, psiDot, index,
   noise);
end
```

B.15 QUEUE PUSH

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Queue Create
% queuePush.m
% This function moves the queue structure time window by one sample
% period.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ varStruct ] = queuePush( varStruct,data )
%Stores data within structure
            varStruct.data = [varStruct.data(2:end) data];
            varStruct.maximum = max(varStruct.data);
            varStruct.minimum = min(varStruct.data);
            varStruct.spread = varStruct.maximum - varStruct.minimum;
            varStruct.previous = varStruct.last;
            varStruct.last = data;
            varStruct.dc = varStruct.spread / varStruct.size;
            varStruct.average = mean(varStruct.data);
            varStruct.change = varStruct.last - varStruct.previous;
            varStruct.windowMean = mean(varStruct.data(end, end-min(length
   (varStruct.data)-1, 10)));
            varStruct.count = varStruct.count;
end
```

B.16 STORE HISTORIES

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Store Histories
% storeHistories.m
```

```
% This function stores variable histories in queue structure.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [data] = storeHistories(data, angle, angleDot, angleCommand)
data.angleHistory = queuePush(data.angleHistory, angle);

data.angleDotHistory = queuePush(data.angleDotHistory, angleDot);
data.angleCommandHistory = queuePush(data.angleCommandHistory,
    angleCommand);

data.angleDotDotHistory = queuePush(data.angleDotDotHistory, ...
    (data.angleDotHistory.previous - data.angleDotHistory.last) ...
    / data.samplePeriod.spread);



%Store Smoothed Data for Controller Steady State Performance
%data.smoothAngle = fastsmooth(data.angleHistory.data,1);
%data.smoothAngleDot = fastsmooth(data.angleDotHistory.data,1);

end
```

B.17 NEW COMMAND

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% New Command Detection
% newCommand.m
% This function detects when a new reference command is sent to the
% reference model.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [data] = newCommand(data)

if data.angleCommandHistory.previous ~= data.angleCommandHistory.last
    fprintf('New Command (Alpha-> %1.2f) Detected...\n', data.
    angleCommandHistory.last);

end
```

B.18 CALCULATE ERROR

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Error Calculation
% calculateError.m
% This function calculates the pitch, roll and yaw error between the
% desired attitude angles and the measured attitude angles. The estimates
% for rates and acceleration errors are also stored.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [pitchData, rollData, yawData] = calculateError(refModel,
    pitchData, rollData, yawData, theta, thetaDot, phi, phiDot, psi,
    psiDot, index, noise)
%% Store Reference Model Values
pitchData.angleRef = queuePush(pitchData.angleRef, refModel.sysRefThetaDTY
    (1,index));
rollData.angleRef = queuePush(rollData.angleRef, refModel.sysRefPhiDTY(1,
    index));
yawData.angleRef = queuePush(yawData.angleRef, refModel.sysRefPsiDTY(1,
    index));

%% Calculate Pitch Error
pitchData.angleError = queuePush(pitchData.angleError, refModel.
    sysRefThetaDTY(1, index) - theta); % (INPUT1)
pitchData.angleDotError = queuePush(pitchData.angleDotError,refModel.
    sysRefThetaDTY(2, index) - thetaDot);
pitchData.angleDotDotError = queuePush(pitchData.angleDotDotError,
    refModel.sysRefThetaDTY(3, index) - pitchData.angleDotDotHistory.last)
    ;

if pitchData.samplePeriod.change >= 1
    pitchData.angleErrorDot = queuePush(pitchData.angleErrorDot,0); %First
     sample
else
    %Adjust noise from error to error rate
    pitchData.angleErrorDot = queuePush(pitchData.angleErrorDot, (
    pitchData.angleError.last - pitchData.angleError.previous)/(pitchData.
    dt) - ( noise(3)*noise(1) - noise(3)*noise(2))/(2*pitchData.dt) + (
    noise(4)*noise(2) - noise(4)*noise(1))/(pitchData.dt));% (INPUT2)
    pitchData.angleDotErrorDot = queuePush(pitchData.angleDotErrorDot, (
    abs(pitchData.angleDotError.last) - abs(pitchData.angleDotError.
    previous))/pitchData.dt);
end
```

```
27
28 %% Calculate Roll Error
29 rollData.angleError = queuePush(rollData.angleError, refModel.sysRefPhiDTY
       (1, index) - phi);%   (INPUT1)
30 rollData.angleErrorDot = queuePush(rollData.angleErrorDot, (rollData.
       angleError.last - rollData.angleError.previous)/rollData.dt );
31 rollData.angleDotError = queuePush(rollData.angleDotError, refModel.
       sysRefPhiDTY(2, index) - phiDot); % (INPUT2)
32 rollData.angleDotDotError = queuePush(rollData.angleDotDotError, refModel.
       sysRefPhiDTY(3, index) - rollData.angleDotDotHistory.last);
33
34
35 %% Calculate Yaw Error
36 yawData.angleError = queuePush(yawData.angleError, refModel.sysRefPsiDTY
       (1, index) - psi);
37 yawData.angleDotError = queuePush(yawData.angleDotError, refModel.
       sysRefPsiDTY(2, index) - psiDot);
38 yawData.angleDotDotError = queuePush(yawData.angleDotDotError, refModel.
       sysRefPsiDTY(3, index) - yawData.angleDotDotHistory.last);
39 end
```

B.19 FUZZY PITCH CONTROLLER

```
1 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Fuzzy Logic Controller for Pitch
3 % fuzzyPitchController.m
4 % Fuzzy Logic Controller for aircraft pitch angle tracking
5 % This function evaluates the fuzzy logic inference system and outputs a
6 % control signal to be sent to the aircraft control surface actuators.
7 % Chris Scott 3/11/21
8 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9 function [ elevatorCmd, pitchData, Pitch_FISa, matPitch_FISa, Pitch_FISi,
       matPitch_FISi ] = fuzzyPitchController(pitchData, Pitch_FISa,
       matPitch_FISa, Pitch_FISi, matPitch_FISi, controlDirection)
10 %% Calculate Absolute + Incremental Controller Output
11     [FISoutputa, matPitch_FISa] = evalflct2faster(...
12         matPitch_FISa,...
13         pitchData.angleError.last, ...
14         pitchData.angleErrorDot.last,7, pitchData);
15
16     FISoutputa = controlDirection*FISoutputa;
```

```
17
18      [FISoutputi, matPitch_FISi] = evalflct2faster(...
19          matPitch_FISi,...
20          pitchData.angleError.last, ...
21          pitchData.angleErrorDot.last,7, pitchData);
22
23  %% Store Fired Fuzzy Logic Rules
24      try
25          pitchData.FR1a = queuePush(pitchData.FR1a,matPitch_FISa(1,1,2,1));
26          pitchData.FRS1a = queuePush(pitchData.FRS1a,matPitch_FISa(1,2,2,1)
    );
27
28          pitchData.FR1i = queuePush(pitchData.FR1i,matPitch_FISi(1,1,2,1));
29          pitchData.FRS1i = queuePush(pitchData.FRS1i,matPitch_FISi(1,2,2,1)
    );
30
31          pitchData.FRI = queuePush(pitchData.FRI,matPitch_FISa(1,3,2,1));%
    Index for all fired rules (current pitch angle)
32      catch
33
34      end
35      try
36          pitchData.FR2a = queuePush(pitchData.FR2a,matPitch_FISa(2,1,2,1));
37          pitchData.FRS2a = queuePush(pitchData.FRS2a,matPitch_FISa(2,2,2,1)
    );
38
39          pitchData.FR2i = queuePush(pitchData.FR2i,matPitch_FISi(2,1,2,1));
40          pitchData.FRS2i = queuePush(pitchData.FRS2i,matPitch_FISi(2,2,2,1)
    );
41      catch
42
43      end
44      try
45          pitchData.FR3a = queuePush(pitchData.FR2a,matPitch_FISa(3,1,2,1));
46          pitchData.FRS3a = queuePush(pitchData.FRS2a,matPitch_FISa(3,2,2,1)
    );
47
48          pitchData.FR3i = queuePush(pitchData.FR2i,matPitch_FISi(3,1,2,1));
49          pitchData.FRS3i = queuePush(pitchData.FRS2i,matPitch_FISi(3,2,2,1)
    );
50      catch
```

```
51
52      end
53      try
54          pitchData.FR4a = queuePush(pitchData.FR4a,matPitch_FISa(4,1,2,1));
55          pitchData.FRS4a = queuePush(pitchData.FRS4a,matPitch_FISa(4,2,2,1)
    );
56
57          pitchData.FR4i = queuePush(pitchData.FR4i,matPitch_FISi(4,1,2,1));
58          pitchData.FRS4i = queuePush(pitchData.FRS4i,matPitch_FISi(4,2,2,1)
    );
59      catch
60
61      end
62
63  %% Store Elevator Command Data
64      pitchData.ctrlSurfaceHistory = queuePush(pitchData.ctrlSurfaceHistory,
      ...
65          min(max(matPitch_FISi(4,1,1,1)*FISoutputi + pitchData.
    ctrlSurfaceHistory.last,...
66          -pitchData.ctrlSurfaceLimit),pitchData.ctrlSurfaceLimit));
67
68      pitchData.AbsOut = matPitch_FISa(4,1,1,1)*FISoutputa; %Store History
69      pitchData.IncOut = pitchData.ctrlSurfaceHistory.last; %Store History
70
71      elevatorCmd = pitchData.AbsOut + pitchData.ctrlSurfaceHistory.last;
72
73
74  %% Controller Output
75  %Limit for elevator command
76  %This prevents trim and FIS outputs to grow larger than actuator range
77  elevatorCmd = min( ...
78      max(elevatorCmd, -pitchData.ctrlSurfaceLimit), ...
79      pitchData.ctrlSurfaceLimit);
80
81  fprintf('Pitch:%1.2f \tPitchRate: %1.2f \tPitchError: %1.2f \
    tPitchErrorDot: %1.2f \tElevator:%1.4f (Abs:%1.4f + Inc:%1.4f) \n',...
82          pitchData.angleHistory.last, pitchData.angleDotHistory.last,
    pitchData.angleError.last,pitchData.angleErrorDot.last,...
83          elevatorCmd, pitchData.AbsOut, pitchData.IncOut);
84
85
```

```
end
```

B.20 EVALUATE FLC

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluate Fuzzy Inference System (Type-2) - Improved
% evalflct2faster.m
% Improved method for evaluating the rule base firing by a priori
% knowledge of a graded set of input membership functions. The structure
% of the input membership functions are currently equally spaces
% triangles, with 50% overlap. This means that at most two membership
% functions for each input can be fired at once, and they must be
% sequential. By knowing this, the evaluation can be truncated as soon
% as these two membership functions have been found. Additionally,
% instead of evaluating all membership functions for firing strength
% [0,1], and most of the membership functions returning zero, as only
% two membership functions fire in each set, a search is performed for
% which rules will have non-zero firing for BOTH inputs, and only these
% rules are evaluated. This resulted in a computation reduction of 98% in
% the case of 49 rules (7 MF for each input).
% The set of rules which are fired for each sample period are stored,
% to be used in future implementation of a self-organizing fuzzy system.
% Chris Scott - 5-15-2018
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [output, matFIS] = evalflct2faster(matFIS,input1,input2,TRType,
    Data)

%Toggle for Input Specific Rule Consequents
if matFIS(1,1,4,1) == 1
try
    state = Data.angleHistory.last;
    range = matFIS(1,2,4,1);
    resolution = matFIS(1,3,4,1);
    page = round((range + state)/resolution);
    if page < 1
        page = 1;
    end
catch
    page = 1;
    end
else
```

```
    page = 1;
end



% Scale Input1
errorInput = scaleInput(input1,-matFIS(2,1,1,1), matFIS(2,1,1,1), -1, 1);
% Scale Input2
errorDotInput = scaleInput(input2,-matFIS(3,1,1,1), matFIS(3,1,1,1), -1,
    1);

input = [errorInput errorDotInput];

%end
%%
rules = matFIS(1:end-1,1:2,8,1);
N = size(rules);
NofRule = N(1);
nInput = 2;
[inputN, ~] = size(input);
output = zeros(inputN,1);

%% Find Membership Functions vs Input Values to reduce computation time
F = zeros(NofRule,2);
C = zeros(NofRule,1);

x = input(1,:);
MFSet = [];
MFLength = 7;

count1 = 0;%Early break out if count goes over 1 (only 2 MF overlap at
    once), Input 1
count2 = 0;%Early break out if count goes over 1 (only 2 MF overlap at
    once), Input 2

%Check Input1 Rules which have nonzero values based on known rule
    structure

[min1, max1, dir1] = direction(input(1), MFLength);%Start at input 1 MF
    mid point and move in direction of input
```

```
[min2, max2, dir2] = direction(input(2), MFLength);%Start at input 2 MF mid point and move in direction of input


for index1 = min1:dir1:max1
    %Find which membership functions input 1 is within (Lower MF)
    MF1number = findMF(matFIS(2*index1+1, 1, 5, 1),...%type of MF
        matFIS(2*index1+1, 2, 5, 1),...%value 1 of MF
        matFIS(2*index1+1, 3 ,5, 1),...%value 2 of MF
        matFIS(2*index1+1, 4, 5, 1),... %value 3 of MF
        x(1),2*index1+1);%Output is the MF matched, otherwise 0

    if MF1number>0

    %Find which membership functions input 2 is within (Lower MF)
    %Input2 can only be within 2 MF at one time
        for index2 = min2:dir2:max2
            MF2number = findMF(matFIS(2*index2+1, 1, 6, 1),...%type of MF
                matFIS(2*index2+1, 2, 6, 1),...%value 1 of MF
                matFIS(2*index2+1, 3 ,6, 1),...%value 2 of MF
                matFIS(2*index2+1, 4, 6, 1),...%value 3 of MF
                x(2),2*index2+1);%Output is the MF matched, otherwise 0

                if MF2number>0
                MFSet = [MFSet; 2*index1+1 2*index2+1 (index1-1)*MFLength+index2 0];%Zero added here is for a placeholder for rule firing strength calculated later
                count2 = count2 + 1;
            end
            if count2 == 2
                count2=0;
                break;
            end
        end
        count1 = count1 + 1;
        if count1 == 2
            break;
        end
    end
end

```

```
%Calculate Upper and Lower Rule firings

for n=1:length(MFSet(:,1))

    f1U=1;
    f1L=1;

    for i=1:nInput
        UpperParams = [matFIS(MFSet(n,i), 2:5, 4+i, 1)];
        UpperMfType = matFIS(MFSet(n,i), 1, 4+i, 1);
        LowerParams = [matFIS(MFSet(n,i)-1, 2:5, 4+i, 1)];
        LowerMfType = matFIS(MFSet(n,i)-1, 1, 4+i, 1);

        MemberUpper = MFType(x, UpperParams, UpperMfType, i);
        MemberLower = MFType(x, LowerParams, LowerMfType, i);


     f1U=f1U*MemberUpper;
        f1L=f1L*MemberLower;

    end

    F(MFSet(n,3),:) = [f1L,f1U];

end


for n=1:length(MFSet(:,1))
        outMFPar = matFIS(MFSet(n,3)+1,page+1,7,1);

   if length(outMFPar) == 2
     C(MFSet(n,3),:) = outMFPar(1);
     C(MFSet(n,3),2) = outMFPar(2);
   else
    C(MFSet(n,3),:) = outMFPar;
    C(MFSet(n,3),2) = outMFPar;
   end
end

%% Store the current fired rules for future performance evaluation
try
```

```
151 matFIS(1,1:3,2,1) = [MFSet(1,3) F(MFSet(1,3),1) page];
152 catch
153 end
154 try
155 matFIS(2,1:3,2,1) =  [MFSet(2,3) F(MFSet(2,3),1) page];
156 catch
157
158 end
159 try
160 matFIS(3,1:3,2,1) =  [MFSet(3,3) F(MFSet(3,3),1) page];
161 catch
162 end
163 try
164 matFIS(4,1:3,2,1) =  [MFSet(4,3) F(MFSet(4,3),1) page];
165 catch
166 end
167
168 %% TYPE REDUCTION
169
170
171 TRMethod = matFIS(11,1,1,1); %Type Reduction Method identified in function
        call.
172 %[KM = 1,'EKM' = 2,'IASC' = 3,'EIASC' = 4,'EODS' = 5,'WM' = 6,'NT' = 7,'
       BMM' = 8]
173 switch TRMethod
174     case 1
175         %TRMethodfunc='t2f_TR_KM';
176         [yL,yR,L,R] = t2f_TR_KM(F,C);
177     case 2
178         %TRMethodfunc='t2f_TR_EKM';
179         [yL,yR,L,R] = t2f_TR_EKM(F,C);
180     case 3
181         %TRMethodfunc='t2f_TR_IASC';
182         [yL,yR,L,R] = t2f_TR_IASC(F,C);
183     case 4
184         %TRMethodfunc='t2f_TR_EIASC';
185         [yL,yR,L,R] = t2f_TR_EIASC(F,C);
186     case 5
187         %TRMethodfunc='t2f_TR_EODS';
188         [yL,yR,L,R] = t2f_TR_EODS(F,C);
189     case 6
```

```
190         %TRMethodfunc='t2f_TR_WM';
191         [yL,yR,L,R] = t2f_TR_WM(F,C);
192     case 7
193         %TRMethodfunc='t2f_TR_NT';
194         [yL,yR,L,R] = t2f_TR_NT(F,C);
195     case 8
196         %TRMethodfunc='t2f_TR_BMM';
197         alfa=str2num(TRMethod(5:regexp(TRMethod,',')-1));
198         beta=str2num(TRMethod(regexp(TRMethod,',')+1:end-1));
199         [yL,yR,L,R] = t2f_TR_BMM(F,C,alfa,beta);
200     case 'Custom'
201     otherwise
202         %TRMethodfunc='t2f_TR_KM';
203         [yL,yR,L,R] = t2f_TR_KM(F,C);
204
205 end
206
207 %% Calculate Output
208 output=(yL+yR)/2;
209
210 end
211
212 %% MEMBERSHIP FUNCTIONS%%
213
214 function output = MFType(x,params, type, i) %trimf = 1,zmf = 2, smf = 3]
215 switch type
216     case{2}
217         output = zmf (x(i),params(1:end));
218     case{1}
219         output = trimf (x(i),params(1:end));
220     case{3}
221         output = smf (x(i),params(1:end));
222 end
223 end
224
225 function output = findMF(type, value1, value2, value3, inputValue,index) %
    trimf = 1,zmf = 2, smf = 3]
226 output = 0;
227 switch type
228     case{2}
229         if inputValue <= value2
```

```
            output = index;
        end
    case{1}
        if inputValue >= value1 && inputValue <= value3
            output = index;
        end
    case{3}
        if inputValue >= value1
            output = index;
        end
end
end

function [min, max, dir] = direction(input, MFnumber)
if input<0
    dir = -1;
    min = ceil(MFnumber/2);
    max = 1;
else
    dir = 1;
    min = floor(MFnumber/2);
    max = MFnumber;
end
end
```

B.21 SCALE INPUT

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scale Input
% scaleInput.m
% This function scales the input value to the FIS, to be contained within
% the defined Universe of Discourse.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function output = scaleInput(input, gainInMin, gainInMax, gainOutMin,
    gainOutMax)
 output = (input-gainInMin)*(gainOutMax-gainOutMin)/(gainInMax-gainInMin)+
    gainOutMin;
end
```

B.22 NT TYPE REDUCTION METHOD

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fuzzy Logic Type-2 Type Reduction Nei-Tan Method
% t2f_TR_NT.m
% A. Taskin and T. Kumbasar, 2015
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [yLeft,yRight,L,R]=t2f_TR_NT(F,Y)

%% IASC Algorithm for Computing Y Left

% Sort Y matrix
lowerY = Y(:,1);
upperY = Y(:,2);
[~, ind] = sort(lowerY);
lowerY = lowerY(ind,:);
upperY = upperY(ind,:);
sortedF = F(ind,:);
lowerF = sortedF(:,1);
upperF = sortedF(:,2);

% Convert to crisp Y matrix

if isequal(lowerY,upperY)
    y=lowerY;
else
    y=(lowerY+upperY)/2;
end


% y=()/();

yLeft=(sum(y.*(upperF+lowerF)))/sum(upperF+lowerF);
yRight=yLeft;
L='no';
R='no';
```

B.23 FUZZY ROLL CONTROLLER

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fuzzy Logic Controller for Roll
% fuzzyRollController.m
% Fuzzy Logic Controller for aircraft roll angle tracking
```

```
% This function evaluates the fuzzy logic inference system and outputs a
% control signal to be sent to the aircraft control surface actuators.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ aileronCmd, rollData ] = fuzzyRollController( rollData,
    Roll_FIS, matRoll_FIS )
%Fuzzy Logic Controller for aircraft roll angle tracking

%% Calculate Controller Output
[FISoutput, matRoll_FIS] = evalflct2faster(...%New
    matRoll_FIS,...
    rollData.angleError.last, ...
    rollData.angleDotError.last,7, rollData);

%% Store Aileron Command Data
rollData.ctrlSurfaceHistory = queuePush(rollData.ctrlSurfaceHistory, ...
    Roll_FIS.outputGain*FISoutput);
rollData.ctrlSurfaceDotHistory = queuePush(rollData.ctrlSurfaceDotHistory,
    rollData.ctrlSurfaceHistory.change);

aileronCmd = rollData.ctrlSurfaceHistory.last;

%% Controller Output
%Limit for elevator command
%This prevents trim and FIS outputs to grow larger than actuator range
aileronCmd = min( ...
    max(aileronCmd, -rollData.ctrlSurfaceLimit), ...
    rollData.ctrlSurfaceLimit);

end
```

B.24 DATALOG

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Data Log Function
% DataLog.m
% This function stores the data for each sample of the simulation or test
% in a data structure to be used for plotting and analysis.
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [ DataLog ] = DataLog(DataLog, refModel, x, xhat, u, time,
    Relevator, Lelevator, aileron, rudder, pitchData, rollData, yawData,
    u_pitch, u_roll, Pitch_FISa, Pitch_FISi, Roll_FIS, DEl, DAil, index)
%Data Log for storing controller and model parameters for plotting.

DataLog.theta(index) = x(5); %Pitch / Alpha
DataLog.thetaDot(index) = x(11); %Pitch Rate
DataLog.Relevator(index) = Relevator; %Measured Right Elevator Angle (
    Positive: Deflect Down)
DataLog.Lelevator(index) = Lelevator; %Measured Left Elevator Angle (
    Positive: Deflect Down)
DataLog.phi(index) = x(4); %Roll
DataLog.phiDot(index) = x(10); %Roll Rate
DataLog.aileron(index) = aileron; %Measured Aileron Angle (Positive: -
    Right/+Left)
DataLog.psi(index) = x(6); %Yaw
DataLog.psiDot(index) = x(12); %Yaw Rate

DataLog.thetahat(index) = xhat(5); %Pitch / Alpha (noisy sensor)
DataLog.thetaDothat(index) = xhat(11); %Pitch Rate (noisy sensor)
DataLog.phihat(index) = xhat(4); %Roll (noisy sensor)
DataLog.phiDothat(index) = xhat(10); %Roll Rate (noisy sensor)

DataLog.rudder(index) = rudder; %Measured Rudder Angle (Positive: Deflect
    Right)
DataLog.thetaRef(index) = refModel.sysRefThetaDTY(1,index); %Pitch
    Reference Model
DataLog.thetaDotRef(index) = refModel.sysRefThetaDTY(2,index); %Pitch Rate
     Reference Model
DataLog.thetaDotDotRef(index) = refModel.sysRefThetaDTY(3,index); %
    PitchDot Rate Reference Model
DataLog.phiRef(index) = refModel.sysRefPhiDTY(1,index); %Roll Reference
    Model
DataLog.phiDotRef(index) = refModel.sysRefPhiDTY(2,index); %Roll Rate
    Reference Model
DataLog.phiDotDotRef(index) = refModel.sysRefPhiDTY(3,index); %RollDot
    Rate Reference Model
DataLog.psiRef(index) = refModel.sysRefPsiDTY(1,index); %Yaw Reference
    Model
DataLog.psiDotRef(index) = refModel.sysRefPsiDTY(2,index); %Yaw Rate
    Reference Model
```

```
35 DataLog.psiDotDotRef(index) = refModel.sysRefPsiDTY(3,index); %YawDot Rate
       Reference Model
36
37 DataLog.thetaError(index) = pitchData.angleError.last;%Ptich FIS input1
38 DataLog.thetaErrorDot(index) = pitchData.angleErrorDot.last;%Pitch FIS
       input2
39 DataLog.thetaDotError(index) = pitchData.angleDotError.last;
40
41 DataLog.phiError(index) = rollData.angleError.last;%Roll FIS input1
42 DataLog.phiErrorDot(index) = rollData.angleErrorDot.last;%Roll FIS input2
43 DataLog.phiDotError(index) = rollData.angleDotError.last;
44
45 DataLog.psiError(index) = yawData.angleError.last;
46 DataLog.psiErrorDot(index) = yawData.angleErrorDot.last;
47 DataLog.psiDotError(index) = yawData.angleDotError.last;
48
49 DataLog.pitchCTRL(index) = u_pitch(1);%Elevator Command
50 DataLog.rollCTRL(index) = u_roll(1);%Aileron Command
51 DataLog.yawCTRL(index) = 0;%Add u_yaw when yaw controller is added
52 DataLog.time(index) = time;
53 DataLog.thetaCommand(index) = u(4); %Commanded Pitch Angle
54 DataLog.phiCommand(index) = u(5); %Commanded Roll Angle
55 DataLog.psiCommand(index) = u(6); %Commanded Yaw Angle
56 DataLog.thetaFISaInput1(index) = Pitch_FISa.input1Gain; %Absolute Pitch
       Error Input Scale
57 DataLog.thetaFISaInput2(index) = Pitch_FISa.input2Gain; %Absolute Pitch
       ErrorDot Input Scale
58 DataLog.thetaFISaOutput(index) = Pitch_FISa.outputGain; %Absolute Pitch
       FIS Output Scale
59 DataLog.thetaFISiInput1(index) = Pitch_FISi.input1Gain; %Incremental Pitch
       Error Input Scale
60 DataLog.thetaFISiInput2(index) = Pitch_FISi.input2Gain; %Incremental Pitch
       ErrorDot Input Scale
61 DataLog.thetaFISiOutput(index) = Pitch_FISi.outputGain; %Incremental Pitch
       FIS Output Scale
62 DataLog.phiFISInput1(index) = Roll_FIS.input1Gain; %Roll Error Input Scale
63 DataLog.phiFISInput2(index) = Roll_FIS.input2Gain; %Roll ErrorDot Scale
64 DataLog.phiFISOutput(index) = Roll_FIS.outputGain; %Roll FIS Output Scale
65 DataLog.psiFISInput1(index) = 0;%Add when yaw controller is added
66 DataLog.psiFISInput2(index) = 0;%Add when yaw controller is added
67 DataLog.psiFISOutput(index) = 0;%Add when yaw controller is added
```

```
DataLog.DEl(index) = DEl;%Pitch Controller Output
DataLog.DAil(index) = DAil;%Roll Controller Output
DataLog.DRud(index) = 0;%Add when yaw controller is added

%F16 State Log
DataLog.x1(index) = x(1);%north position
DataLog.x2(index) = x(2);%east position
DataLog.x3(index) = x(3);%altitude
DataLog.x4(index) = x(4);%roll
DataLog.x5(index) = x(5);%pitch
DataLog.x6(index) = x(6);%yaw
DataLog.x7(index) = x(7);%speed
DataLog.x8(index) = x(8);%angle of attack
DataLog.x9(index) = x(9);%side slip
DataLog.x10(index) = x(10);%roll rate
DataLog.x11(index) = x(11);%pitch rate
DataLog.x12(index) = x(12);%yaw rate


%Absolute + Incremental Output Data
DataLog.AbsOut(index) = pitchData.AbsOut;
DataLog.IncOut(index) = pitchData.IncOut;


end
```

B.25 MATRIX TO STRUCTURE

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Matrix to Structure
% mat2struc.m
% This function converts the
% Chris Scott 3/11/21
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ FIS ] = mat2struc( FIS, matFIS )
%This function updates the intial FIS structure with values from the
    modified matrix
%version of the data structure

%% Construct Main FIS Matrix
%Set 1, Sheet 1: Update FIS input/output scaling
```

```
    FIS.input1Gain = matFIS(1,1,1,1);
    FIS.input2Gain = matFIS(2,1,1,1);
    FIS.outputGain = matFIS(3,1,1,1);

%Set 1, Sheet 7 [Output]: Update output consequents
for i = 2:length(FIS.FLC.output.mf)+1
        for ii = 2:length(FIS.FLC.output.mf(i-1).params)+1
            FIS.FLC.output.mf(i-1).params(ii-1) = matFIS(i,ii,7,1);
        end
end


end
```

B.26 IT1 FIS PITCH - ABSOLUTE FIS

```
[System]
Name='FuzzyControllerNap(SelfOrg)'
Type='sugeno'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=49
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'
TypeRedMethod='NT'
outputType='icrisp'

[Input1]
Name='Error'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.7852 -0.6356 0.833]
MF1L='mf1L':'zmf',[-0.7852 -0.6356 0.833]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 0.833]
MF2L='mf2L':'trimf',[-1 -0.6667 -0.3333 0.833]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 0.833]
MF3L='mf3L':'trimf',[-0.6667 -0.3333 0 0.833]
```

```
MF4U='Z':'trimf',[-0.3333 0 0.3333 0.833]
MF4L='mf4L':'trimf',[-0.3333 0 0.3333 0.833]
MF5U='PS':'trimf',[0 0.3333 0.6667 0.833]
MF5L='mf5L':'trimf',[0 0.3333 0.6667 0.833]
MF6U='PM':'trimf',[0.3333 0.6667 1 0.833]
MF6L='mf6L':'trimf',[0.3333 0.6667 1 0.833]
MF7U='PB':'smf',[0.6356 0.7852 0.833]
MF7L='mf7L':'smf',[0.6356 0.7852 0.833]

[Input2]
Name='dError'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.7852 -0.6356 0.833]
MF1L='mf1L':'zmf',[-0.7852 -0.6356 0.833]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 0.833]
MF2L='mf2L':'trimf',[-1 -0.6667 -0.3333 0.833]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 0.833]
MF3L='mf3L':'trimf',[-0.6667 -0.3333 0 0.833]
MF4U='Z':'trimf',[-0.3333 0 0.3333 0.833]
MF4L='mf4L':'trimf',[-0.3333 0 0.3333 0.833]
MF5U='PS':'trimf',[0 0.3333 0.6667 0.833]
MF5L='mf5L':'trimf',[0 0.3333 0.6667 0.833]
MF6U='PM':'trimf',[0.3333 0.6667 1 0.833]
MF6L='mf6L':'trimf',[0.3333 0.6667 1 0.833]
MF7U='PB':'smf',[0.6356 0.7852 0.833]
MF7L='mf7L':'smf',[0.6356 0.7852 0.833]

[Output1]
Name='output1'
Range=[-1 1]
NumMFs=49
MF1='1':'constant',[-1 -1]
MF2='2':'constant',[-1 -1]
MF3='3':'constant',[-1 -1]
MF4='4':'constant',[-1 -1]
MF5='5':'constant',[-1 -1]
MF6='6':'constant',[-0.666666666666667 -0.666666666666667]
MF7='7':'constant',[-0.666666666666667 -0.666666666666667]
MF8='8':'constant',[-1 -1]
MF9='9':'constant',[-1 -1]
```

```
MF10='10':'constant',[-1 -1]
MF11='11':'constant',[-0.666666666666667 -0.666666666666667]
MF12='12':'constant',[-0.666666666666667 -0.666666666666667]
MF13='13':'constant',[-0.333333333333333 -0.333333333333333]
MF14='14':'constant',[-0.333333333333333 -0.333333333333333]
MF15='15':'constant',[-1 -1]
MF16='16':'constant',[-0.666666666666667 -0.666666666666667]
MF17='17':'constant',[-0.666666666666667 -0.666666666666667]
MF18='18':'constant',[-0.333333333333333 -0.333333333333333]
MF19='19':'constant',[0 0]
MF20='20':'constant',[0.333333333333333 0.333333333333333]
MF21='21':'constant',[0.333333333333333 0.333333333333333]
MF22='22':'constant',[-0.666666666666667 -0.666666666666667]
MF23='23':'constant',[-0.666666666666667 -0.666666666666667]
MF24='24':'constant',[-0.333333333333333 -0.333333333333333]
MF25='25':'constant',[0 0]
MF26='26':'constant',[0.333333333333333 0.333333333333333]
MF27='27':'constant',[0.666666666666667 0.666666666666667]
MF28='28':'constant',[0.666666666666667 0.666666666666667]
MF29='29':'constant',[-0.333333333333333 -0.333333333333333]
MF30='30':'constant',[-0.333333333333333 -0.333333333333333]
MF31='31':'constant',[0 0]
MF32='32':'constant',[0.333333333333333 0.333333333333333]
MF33='33':'constant',[0.666666666666667 0.666666666666667]
MF34='34':'constant',[0.666666666666667 0.666666666666667]
MF35='35':'constant',[1 1]
MF36='36':'constant',[0.333333333333333 0.333333333333333]
MF37='37':'constant',[0.333333333333333 0.333333333333333]
MF38='38':'constant',[0.666666666666667 0.666666666666667]
MF39='39':'constant',[0.666666666666667 0.666666666666667]
MF40='40':'constant',[1 1]
MF41='41':'constant',[1 1]
MF42='42':'constant',[1 1]
MF43='43':'constant',[0.666666666666667 0.666666666666667]
MF44='44':'constant',[0.666666666666667 0.666666666666667]
MF45='45':'constant',[1 1]
MF46='46':'constant',[1 1]
MF47='47':'constant',[1 1]
MF48='48':'constant',[1 1]
MF49='49':'constant',[1 1]

```

```
[Rules]
1 1, 1 (1) : 1
1 2, 2 (1) : 1
1 3, 3 (1) : 1
1 4, 4 (1) : 1
1 5, 5 (1) : 1
1 6, 6 (1) : 1
1 7, 7 (1) : 1
2 1, 8 (1) : 1
2 2, 9 (1) : 1
2 3, 10 (1) : 1
2 4, 11 (1) : 1
2 5, 12 (1) : 1
2 6, 13 (1) : 1
2 7, 14 (1) : 1
3 1, 15 (1) : 1
3 2, 16 (1) : 1
3 3, 17 (1) : 1
3 4, 18 (1) : 1
3 5, 19 (1) : 1
3 6, 20 (1) : 1
3 7, 21 (1) : 1
4 1, 22 (1) : 1
4 2, 23 (1) : 1
4 3, 24 (1) : 1
4 4, 25 (1) : 1
4 5, 26 (1) : 1
4 6, 27 (1) : 1
4 7, 28 (1) : 1
5 1, 29 (1) : 1
5 2, 30 (1) : 1
5 3, 31 (1) : 1
5 4, 32 (1) : 1
5 5, 33 (1) : 1
5 6, 34 (1) : 1
5 7, 35 (1) : 1
6 1, 36 (1) : 1
6 2, 37 (1) : 1
6 3, 38 (1) : 1
6 4, 39 (1) : 1
6 5, 40 (1) : 1
```

```
6 6, 41 (1) : 1
6 7, 42 (1) : 1
7 1, 43 (1) : 1
7 2, 44 (1) : 1
7 3, 45 (1) : 1
7 4, 46 (1) : 1
7 5, 47 (1) : 1
7 6, 48 (1) : 1
7 7, 49 (1) : 1
```

B.27 IT1 FIS PITCH - INCREMENTAL FIS

```
[System]
Name='FuzzyControllerInc(SelfOrg)v4'
Type='sugeno'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=49
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'
TypeRedMethod='NT'
outputType='icrisp'

[Input1]
Name='Error'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.7852 -0.6356 0.833]
MF1L='mf1L':'zmf',[-0.7852 -0.6356 0.833]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 0.833]
MF2L='mf2L':'trimf',[-1 -0.6667 -0.3333 0.833]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 0.833]
MF3L='mf3L':'trimf',[-0.6667 -0.3333 0 0.833]
MF4U='Z':'trimf',[-0.3333 0 0.3333 0.833]
MF4L='mf4L':'trimf',[-0.3333 0 0.3333 0.833]
MF5U='PS':'trimf',[0 0.3333 0.6667 0.833]
MF5L='mf5L':'trimf',[0 0.3333 0.6667 0.833]
```

```
MF6U='PM':'trimf',[0.3333 0.6667 1 0.833]
MF6L='mf6L':'trimf',[0.3333 0.6667 1 0.833]
MF7U='PB':'smf',[0.6356 0.7852 0.833]
MF7L='mf7L':'smf',[0.6356 0.7852 0.833]

[Input2]
Name='ThetaDot'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.7852 -0.6356 0.833]
MF1L='mf1L':'zmf',[-0.7852 -0.6356 0.833]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 0.833]
MF2L='mf2L':'trimf',[-0.875 -0.6667 -0.4583 0.833]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 0.833]
MF3L='mf3L':'trimf',[-0.5417 -0.3333 -0.125 0.833]
MF4U='Z':'trimf',[-0.3333 0 0.3333 0.833]
MF4L='mf4L':'trimf',[-0.2083 0 0.2083 0.833]
MF5U='PS':'trimf',[0 0.3333 0.6667 0.833]
MF5L='mf5L':'trimf',[0.125 0.3333 0.5417 0.833]
MF6U='PM':'trimf',[0.3333 0.6667 1 0.833]
MF6L='mf6L':'trimf',[0.4583 0.6667 0.875 0.833]
MF7U='PB':'smf',[0.6356 0.7852 0.833]
MF7L='mf7L':'smf',[0.6356 0.7852 0.833]

[Output1]
Name='output1'
Range=[-1 1]
NumMFs=49
MF1='1':'constant',[.1 .1]
MF2='2':'constant',[.1 .1]
MF3='3':'constant',[.1 .1]
MF4='4':'constant',[.1 .1]
MF5='5':'constant',[0.015 0.015]
MF6='6':'constant',[0 0]
MF7='7':'constant',[-0.015 -0.015]
MF8='8':'constant',[.1 .1]
MF9='9':'constant',[0.05 0.05]
MF10='10':'constant',[0.015 0.015]
MF11='11':'constant',[0.015 0.015]
MF12='12':'constant',[0 0]
MF13='13':'constant',[0 0]
```

```
MF14='14':'constant',[-0.015 -0.015]
MF15='15':'constant',[.1 .1]
MF16='16':'constant',[0.05 0.05]
MF17='17':'constant',[0.015 0.015]
MF18='18':'constant',[0 0]
MF19='19':'constant',[0 0]
MF20='20':'constant',[-0.015 -0.015]
MF21='21':'constant',[-0.05 -0.05]
MF22='22':'constant',[.1 .1]
MF23='23':'constant',[0.015 0.015]
MF24='24':'constant',[0 0]
MF25='25':'constant',[0 0]
MF26='26':'constant',[0 0]
MF27='27':'constant',[-0.015 -0.015]
MF28='28':'constant',[-.1 -.1]
MF29='29':'constant',[0.05 0.05]
MF30='30':'constant',[0.015 0.015]
MF31='31':'constant',[0 0]
MF32='32':'constant',[0 0]
MF33='33':'constant',[-0.015 -0.015]
MF34='34':'constant',[-0.05 -0.05]
MF35='35':'constant',[-.1 -.1]
MF36='36':'constant',[0.015 0.015]
MF37='37':'constant',[0 0]
MF38='38':'constant',[0 0]
MF39='39':'constant',[-0.015 -0.015]
MF40='40':'constant',[-0.015 -0.015]
MF41='41':'constant',[-0.05 -0.05]
MF42='42':'constant',[-.1 -.1]
MF43='43':'constant',[0.015 0.015]
MF44='44':'constant',[0 0]
MF45='45':'constant',[-0.015 -0.015]
MF46='46':'constant',[-.1 -.1]
MF47='47':'constant',[-.1 -.1]
MF48='48':'constant',[-.1 -.1]
MF49='49':'constant',[-.1 -.1]

[Rules]
1 1, 1 (1) : 1
1 2, 2 (1) : 1
1 3, 3 (1) : 1
```

```
1 4, 4 (1) : 1
1 5, 5 (1) : 1
1 6, 6 (1) : 1
1 7, 7 (1) : 1
2 1, 8 (1) : 1
2 2, 9 (1) : 1
2 3, 10 (1) : 1
2 4, 11 (1) : 1
2 5, 12 (1) : 1
2 6, 13 (1) : 1
2 7, 14 (1) : 1
3 1, 15 (1) : 1
3 2, 16 (1) : 1
3 3, 17 (1) : 1
3 4, 18 (1) : 1
3 5, 19 (1) : 1
3 6, 20 (1) : 1
3 7, 21 (1) : 1
4 1, 22 (1) : 1
4 2, 23 (1) : 1
4 3, 24 (1) : 1
4 4, 25 (1) : 1
4 5, 26 (1) : 1
4 6, 27 (1) : 1
4 7, 28 (1) : 1
5 1, 29 (1) : 1
5 2, 30 (1) : 1
5 3, 31 (1) : 1
5 4, 32 (1) : 1
5 5, 33 (1) : 1
5 6, 34 (1) : 1
5 7, 35 (1) : 1
6 1, 36 (1) : 1
6 2, 37 (1) : 1
6 3, 38 (1) : 1
6 4, 39 (1) : 1
6 5, 40 (1) : 1
6 6, 41 (1) : 1
6 7, 42 (1) : 1
7 1, 43 (1) : 1
7 2, 44 (1) : 1
```

```
7 3, 45 (1) : 1
7 4, 46 (1) : 1
7 5, 47 (1) : 1
7 6, 48 (1) : 1
7 7, 49 (1) : 1
```

B.28 IT2 FIS PITCH - ABSOLUTE FIS

```
[System]
Name='FuzzyControllerNap(SelfOrg)'
Type='sugeno'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=49
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'
TypeRedMethod='NT'
outputType='icrisp'

[Input1]
Name='Error'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.9426 -0.763 1]
MF1L='mf1L':'zmf',[-0.9656 -0.8578 0.6667]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 1]
MF2L='mf2L':'trimf',[-0.875 -0.6667 -0.4583 0.6667]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 1]
MF3L='mf3L':'trimf',[-0.5417 -0.3333 -0.125 0.6667]
MF4U='Z':'trimf',[-0.3333 0 0.3333 1]
MF4L='mf4L':'trimf',[-0.2083 0 0.2083 0.6667]
MF5U='PS':'trimf',[0 0.3333 0.6667 1]
MF5L='mf5L':'trimf',[0.125 0.3333 0.5417 0.6667]
MF6U='PM':'trimf',[0.3333 0.6667 1 1]
MF6L='mf6L':'trimf',[0.4583 0.6667 0.875 0.6667]
MF7U='PB':'smf',[0.763 0.9426 1]
MF7L='mf7L':'smf',[0.8578 0.9656 0.6667]
```

```
34
35 [Input2]
36 Name='dError'
37 Range=[-1 1]
38 NumMFs=7
39 MF1U='NB':'zmf',[-0.9426 -0.763 1]
40 MF1L='mf1L':'zmf',[-0.9656 -0.8578 0.6667]
41 MF2U='NM':'trimf',[-1 -0.6667 -0.3333 1]
42 MF2L='mf2L':'trimf',[-0.875 -0.6667 -0.4583 0.6667]
43 MF3U='NS':'trimf',[-0.6667 -0.3333 0 1]
44 MF3L='mf3L':'trimf',[-0.5417 -0.3333 -0.125 0.6667]
45 MF4U='Z':'trimf',[-0.3333 0 0.3333 1]
46 MF4L='mf4L':'trimf',[-0.2083 0 0.2083 0.6667]
47 MF5U='PS':'trimf',[0 0.3333 0.6667 1]
48 MF5L='mf5L':'trimf',[0.125 0.3333 0.5417 0.6667]
49 MF6U='PM':'trimf',[0.3333 0.6667 1 1]
50 MF6L='mf6L':'trimf',[0.4583 0.6667 0.875 0.6667]
51 MF7U='PB':'smf',[0.763 0.9426 1]
52 MF7L='mf7L':'smf',[0.8578 0.9656 0.6667]
53
54 [Output1]
55 Name='output1'
56 Range=[-1 1]
57 NumMFs=49
58 MF1='1':'constant',[-1 -1]
59 MF2='2':'constant',[-1 -1]
60 MF3='3':'constant',[-1 -1]
61 MF4='4':'constant',[-1 -1]
62 MF5='5':'constant',[-1 -1]
63 MF6='6':'constant',[-0.666666666666667 -0.666666666666667]
64 MF7='7':'constant',[-0.666666666666667 -0.666666666666667]
65 MF8='8':'constant',[-1 -1]
66 MF9='9':'constant',[-1 -1]
67 MF10='10':'constant',[-1 -1]
68 MF11='11':'constant',[-0.666666666666667 -0.666666666666667]
69 MF12='12':'constant',[-0.666666666666667 -0.666666666666667]
70 MF13='13':'constant',[-0.333333333333333 -0.333333333333333]
71 MF14='14':'constant',[-0.333333333333333 -0.333333333333333]
72 MF15='15':'constant',[-1 -1]
73 MF16='16':'constant',[-0.666666666666667 -0.666666666666667]
74 MF17='17':'constant',[-0.666666666666667 -0.666666666666667]
```

```
MF18='18':'constant',[-0.333333333333333 -0.333333333333333]
MF19='19':'constant',[0 0]
MF20='20':'constant',[0.333333333333333 0.333333333333333]
MF21='21':'constant',[0.333333333333333 0.333333333333333]
MF22='22':'constant',[-0.666666666666667 -0.666666666666667]
MF23='23':'constant',[-0.666666666666667 -0.666666666666667]
MF24='24':'constant',[-0.333333333333333 -0.333333333333333]
MF25='25':'constant',[0 0]
MF26='26':'constant',[0.333333333333333 0.333333333333333]
MF27='27':'constant',[0.666666666666667 0.666666666666667]
MF28='28':'constant',[0.666666666666667 0.666666666666667]
MF29='29':'constant',[-0.333333333333333 -0.333333333333333]
MF30='30':'constant',[-0.333333333333333 -0.333333333333333]
MF31='31':'constant',[0 0]
MF32='32':'constant',[0.333333333333333 0.333333333333333]
MF33='33':'constant',[0.666666666666667 0.666666666666667]
MF34='34':'constant',[0.666666666666667 0.666666666666667]
MF35='35':'constant',[1 1]
MF36='36':'constant',[0.333333333333333 0.333333333333333]
MF37='37':'constant',[0.333333333333333 0.333333333333333]
MF38='38':'constant',[0.666666666666667 0.666666666666667]
MF39='39':'constant',[0.666666666666667 0.666666666666667]
MF40='40':'constant',[1 1]
MF41='41':'constant',[1 1]
MF42='42':'constant',[1 1]
MF43='43':'constant',[0.666666666666667 0.666666666666667]
MF44='44':'constant',[0.666666666666667 0.666666666666667]
MF45='45':'constant',[1 1]
MF46='46':'constant',[1 1]
MF47='47':'constant',[1 1]
MF48='48':'constant',[1 1]
MF49='49':'constant',[1 1]

[Rules]
1 1, 1 (1) : 1
1 2, 2 (1) : 1
1 3, 3 (1) : 1
1 4, 4 (1) : 1
1 5, 5 (1) : 1
1 6, 6 (1) : 1
1 7, 7 (1) : 1
```

```
2 1, 8 (1) : 1
2 2, 9 (1) : 1
2 3, 10 (1) : 1
2 4, 11 (1) : 1
2 5, 12 (1) : 1
2 6, 13 (1) : 1
2 7, 14 (1) : 1
3 1, 15 (1) : 1
3 2, 16 (1) : 1
3 3, 17 (1) : 1
3 4, 18 (1) : 1
3 5, 19 (1) : 1
3 6, 20 (1) : 1
3 7, 21 (1) : 1
4 1, 22 (1) : 1
4 2, 23 (1) : 1
4 3, 24 (1) : 1
4 4, 25 (1) : 1
4 5, 26 (1) : 1
4 6, 27 (1) : 1
4 7, 28 (1) : 1
5 1, 29 (1) : 1
5 2, 30 (1) : 1
5 3, 31 (1) : 1
5 4, 32 (1) : 1
5 5, 33 (1) : 1
5 6, 34 (1) : 1
5 7, 35 (1) : 1
6 1, 36 (1) : 1
6 2, 37 (1) : 1
6 3, 38 (1) : 1
6 4, 39 (1) : 1
6 5, 40 (1) : 1
6 6, 41 (1) : 1
6 7, 42 (1) : 1
7 1, 43 (1) : 1
7 2, 44 (1) : 1
7 3, 45 (1) : 1
7 4, 46 (1) : 1
7 5, 47 (1) : 1
7 6, 48 (1) : 1
```

```
7 7, 49 (1) : 1
```

B.29 IT2 FIS PITCH - INCREMENTAL FIS

```
[System]
Name='FuzzyControllerInc(SelfOrg)v4'
Type='sugeno'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=49
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'
TypeRedMethod='NT'
outputType='icrisp'

[Input1]
Name='Error'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.9426 -0.763 1]
MF1L='mf1L':'zmf',[-0.9656 -0.8578 0.6667]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 1]
MF2L='mf2L':'trimf',[-0.875 -0.6667 -0.4583 0.6667]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 1]
MF3L='mf3L':'trimf',[-0.5417 -0.3333 -0.125 0.6667]
MF4U='Z':'trimf',[-0.3333 0 0.3333 1]
MF4L='mf4L':'trimf',[-0.2083 0 0.2083 0.6667]
MF5U='PS':'trimf',[0 0.3333 0.6667 1]
MF5L='mf5L':'trimf',[0.125 0.3333 0.5417 0.6667]
MF6U='PM':'trimf',[0.3333 0.6667 1 1]
MF6L='mf6L':'trimf',[0.4583 0.6667 0.875 0.6667]
MF7U='PB':'smf',[0.763 0.9426 1]
MF7L='mf7L':'smf',[0.8578 0.9656 0.6667]

[Input2]
Name='ThetaDot'
Range=[-1 1]
```

```
NumMFs=7
MF1U='NB':'zmf',[-0.9426 -0.763 1]
MF1L='mf1L':'zmf',[-0.9656 -0.8578 0.6667]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 1]
MF2L='mf2L':'trimf',[-0.875 -0.6667 -0.4583 0.6667]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 1]
MF3L='mf3L':'trimf',[-0.5417 -0.3333 -0.125 0.6667]
MF4U='Z':'trimf',[-0.3333 0 0.3333 1]
MF4L='mf4L':'trimf',[-0.2083 0 0.2083 0.6667]
MF5U='PS':'trimf',[0 0.3333 0.6667 1]
MF5L='mf5L':'trimf',[0.125 0.3333 0.5417 0.6667]
MF6U='PM':'trimf',[0.3333 0.6667 1 1]
MF6L='mf6L':'trimf',[0.4583 0.6667 0.875 0.6667]
MF7U='PB':'smf',[0.763 0.9426 1]
MF7L='mf7L':'smf',[0.8578 0.9656 0.6667]

[Output1]
Name='output1'
Range=[-1 1]
NumMFs=49
MF1='1':'constant',[.1 .1]
MF2='2':'constant',[.1 .1]
MF3='3':'constant',[.1 .1]
MF4='4':'constant',[.1 .1]
MF5='5':'constant',[0.015 0.015]
MF6='6':'constant',[0 0]
MF7='7':'constant',[-0.015 -0.015]
MF8='8':'constant',[.1 .1]
MF9='9':'constant',[0.05 0.05]
MF10='10':'constant',[0.015 0.015]
MF11='11':'constant',[0.015 0.015]
MF12='12':'constant',[0 0]
MF13='13':'constant',[0 0]
MF14='14':'constant',[-0.015 -0.015]
MF15='15':'constant',[.1 .1]
MF16='16':'constant',[0.05 0.05]
MF17='17':'constant',[0.015 0.015]
MF18='18':'constant',[0 0]
MF19='19':'constant',[0 0]
MF20='20':'constant',[-0.015 -0.015]
MF21='21':'constant',[-0.05 -0.05]
```

```
MF22='22':'constant',[.1 .1]
MF23='23':'constant',[0.015 0.015]
MF24='24':'constant',[0 0]
MF25='25':'constant',[0 0]
MF26='26':'constant',[0 0]
MF27='27':'constant',[-0.015 -0.015]
MF28='28':'constant',[-.1 -.1]
MF29='29':'constant',[0.05 0.05]
MF30='30':'constant',[0.015 0.015]
MF31='31':'constant',[0 0]
MF32='32':'constant',[0 0]
MF33='33':'constant',[-0.015 -0.015]
MF34='34':'constant',[-0.05 -0.05]
MF35='35':'constant',[-.1 -.1]
MF36='36':'constant',[0.015 0.015]
MF37='37':'constant',[0 0]
MF38='38':'constant',[0 0]
MF39='39':'constant',[-0.015 -0.015]
MF40='40':'constant',[-0.015 -0.015]
MF41='41':'constant',[-0.05 -0.05]
MF42='42':'constant',[-.1 -.1]
MF43='43':'constant',[0.015 0.015]
MF44='44':'constant',[0 0]
MF45='45':'constant',[-0.015 -0.015]
MF46='46':'constant',[-.1 -.1]
MF47='47':'constant',[-.1 -.1]
MF48='48':'constant',[-.1 -.1]
MF49='49':'constant',[-.1 -.1]

[Rules]
1 1, 1 (1) : 1
1 2, 2 (1) : 1
1 3, 3 (1) : 1
1 4, 4 (1) : 1
1 5, 5 (1) : 1
1 6, 6 (1) : 1
1 7, 7 (1) : 1
2 1, 8 (1) : 1
2 2, 9 (1) : 1
2 3, 10 (1) : 1
2 4, 11 (1) : 1
```

```
2 5, 12 (1) : 1
2 6, 13 (1) : 1
2 7, 14 (1) : 1
3 1, 15 (1) : 1
3 2, 16 (1) : 1
3 3, 17 (1) : 1
3 4, 18 (1) : 1
3 5, 19 (1) : 1
3 6, 20 (1) : 1
3 7, 21 (1) : 1
4 1, 22 (1) : 1
4 2, 23 (1) : 1
4 3, 24 (1) : 1
4 4, 25 (1) : 1
4 5, 26 (1) : 1
4 6, 27 (1) : 1
4 7, 28 (1) : 1
5 1, 29 (1) : 1
5 2, 30 (1) : 1
5 3, 31 (1) : 1
5 4, 32 (1) : 1
5 5, 33 (1) : 1
5 6, 34 (1) : 1
5 7, 35 (1) : 1
6 1, 36 (1) : 1
6 2, 37 (1) : 1
6 3, 38 (1) : 1
6 4, 39 (1) : 1
6 5, 40 (1) : 1
6 6, 41 (1) : 1
6 7, 42 (1) : 1
7 1, 43 (1) : 1
7 2, 44 (1) : 1
7 3, 45 (1) : 1
7 4, 46 (1) : 1
7 5, 47 (1) : 1
7 6, 48 (1) : 1
7 7, 49 (1) : 1
```

B.30 IT1 FIS ROLL - ABSOLUTE FIS

```
[System]
Name='FuzzyControllerNap'
Type='sugeno'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=50
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'
TypeRedMethod='NT'
outputType='icrisp'

[Input1]
Name='Error'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.9426 -0.763 1]
MF1L='mf1L':'zmf',[-0.9426 -0.763 1]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 1]
MF2L='mf2L':'trimf',[-1 -0.6667 -0.3333 1]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 1]
MF3L='mf3L':'trimf',[-0.6667 -0.3333 0 1]
MF4U='Z':'trimf',[-0.3333 0 0.3333 1]
MF4L='mf4L':'trimf',[-0.3333 0 0.3333 1]
MF5U='PS':'trimf',[0 0.3333 0.6667 1]
MF5L='mf5L':'trimf',[0 0.3333 0.6667 1]
MF6U='PM':'trimf',[0.3333 0.6667 1 1]
MF6L='mf6L':'trimf',[0.3333 0.6667 1 1]
MF7U='PB':'smf',[0.763 0.9426 1]
MF7L='mf7L':'smf',[0.763 0.9426 1]

[Input2]
Name='dError'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.9426 -0.763 1]
MF1L='mf1L':'zmf',[-0.9426 -0.763 1]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 1]
```

```
MF2L='mf2L':'trimf',[-1 -0.6667 -0.3333 1]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 1]
MF3L='mf3L':'trimf',[-0.6667 -0.3333 0 1]
MF4U='Z':'trimf',[-0.3333 0 0.3333 1]
MF4L='mf4L':'trimf',[-0.3333 0 0.3333 1]
MF5U='PS':'trimf',[0 0.3333 0.6667 1]
MF5L='mf5L':'trimf',[0 0.3333 0.6667 1]
MF6U='PM':'trimf',[0.3333 0.6667 1 1]
MF6L='mf6L':'trimf',[0.3333 0.6667 1 1]
MF7U='PB':'smf',[0.763 0.9426 1]
MF7L='mf7L':'smf',[0.763 0.9426 1]

[Output1]
Name='output1'
Range=[-1 1]
NumMFs=49
MF1='1':'constant',[-1 -1]
MF2='2':'constant',[-1 -1]
MF3='3':'constant',[-1 -1]
MF4='4':'constant',[-1 -1]
MF5='5':'constant',[-1 -1]
MF6='6':'constant',[-0.666666666666667 -0.666666666666667]
MF7='7':'constant',[-0.666666666666667 -0.666666666666667]
MF8='8':'constant',[-1 -1]
MF9='9':'constant',[-1 -1]
MF10='10':'constant',[-1 -1]
MF11='11':'constant',[-0.666666666666667 -0.666666666666667]
MF12='12':'constant',[-0.666666666666667 -0.666666666666667]
MF13='13':'constant',[-0.333333333333333 -0.333333333333333]
MF14='14':'constant',[-0.333333333333333 -0.333333333333333]
MF15='15':'constant',[-1 -1]
MF16='16':'constant',[-0.666666666666667 -0.666666666666667]
MF17='17':'constant',[-0.666666666666667 -0.666666666666667]
MF18='18':'constant',[-0.333333333333333 -0.333333333333333]
MF19='19':'constant',[0 0]
MF20='20':'constant',[0.333333333333333 0.333333333333333]
MF21='21':'constant',[0.333333333333333 0.333333333333333]
MF22='22':'constant',[-0.666666666666667 -0.666666666666667]
MF23='23':'constant',[-0.666666666666667 -0.666666666666667]
MF24='24':'constant',[-0.333333333333333 -0.333333333333333]
MF25='25':'constant',[0 0]
```

```
MF26='26':'constant',[0.333333333333333 0.333333333333333]
MF27='27':'constant',[0.666666666666667 0.666666666666667]
MF28='28':'constant',[0.666666666666667 0.666666666666667]
MF29='29':'constant',[-0.333333333333333 -0.333333333333333]
MF30='30':'constant',[-0.333333333333333 -0.333333333333333]
MF31='31':'constant',[0 0]
MF32='32':'constant',[0.333333333333333 0.333333333333333]
MF33='33':'constant',[0.666666666666667 0.666666666666667]
MF34='34':'constant',[0.666666666666667 0.666666666666667]
MF35='35':'constant',[1 1]
MF36='36':'constant',[0.333333333333333 0.333333333333333]
MF37='37':'constant',[0.333333333333333 0.333333333333333]
MF38='38':'constant',[0.666666666666667 0.666666666666667]
MF39='39':'constant',[0.666666666666667 0.666666666666667]
MF40='40':'constant',[1 1]
MF41='41':'constant',[1 1]
MF42='42':'constant',[1 1]
MF43='43':'constant',[0.666666666666667 0.666666666666667]
MF44='44':'constant',[0.666666666666667 0.666666666666667]
MF45='45':'constant',[1 1]
MF46='46':'constant',[1 1]
MF47='47':'constant',[1 1]
MF48='48':'constant',[1 1]
MF49='49':'constant',[1 1]

[Rules]
1 1, 1 (1) : 1
1 2, 2 (1) : 1
1 3, 3 (1) : 1
1 4, 4 (1) : 1
1 5, 5 (1) : 1
1 6, 6 (1) : 1
1 7, 7 (1) : 1
2 1, 8 (1) : 1
2 2, 9 (1) : 1
2 3, 10 (1) : 1
2 4, 11 (1) : 1
2 5, 12 (1) : 1
2 6, 13 (1) : 1
2 7, 14 (1) : 1
3 1, 15 (1) : 1
```

```
3 2, 16 (1) : 1
3 3, 17 (1) : 1
3 4, 18 (1) : 1
3 5, 19 (1) : 1
3 6, 20 (1) : 1
3 7, 21 (1) : 1
4 1, 22 (1) : 1
4 2, 23 (1) : 1
4 3, 24 (1) : 1
4 4, 25 (1) : 1
4 5, 26 (1) : 1
4 6, 27 (1) : 1
4 7, 28 (1) : 1
5 1, 29 (1) : 1
5 2, 30 (1) : 1
5 3, 31 (1) : 1
5 4, 32 (1) : 1
5 5, 33 (1) : 1
5 6, 34 (1) : 1
5 7, 35 (1) : 1
6 1, 36 (1) : 1
6 2, 37 (1) : 1
6 3, 38 (1) : 1
6 4, 39 (1) : 1
6 5, 40 (1) : 1
6 6, 41 (1) : 1
6 7, 42 (1) : 1
7 1, 43 (1) : 1
7 2, 44 (1) : 1
7 3, 45 (1) : 1
7 4, 46 (1) : 1
7 5, 47 (1) : 1
7 6, 48 (1) : 1
7 7, 49 (1) : 1
```

B.31 IT2 FIS ROLL - ABSOLUTE FIS

```
[System]
Name='FuzzyControllerNap'
Type='sugeno'
Version=2.0
```

```
NumInputs=2
NumOutputs=1
NumRules=50
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'
TypeRedMethod='NT'
outputType='icrisp'

[Input1]
Name='Error'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.9426 -0.763 1]
MF1L='mf1L':'zmf',[-0.9656 -0.8578 0.6667]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 1]
MF2L='mf2L':'trimf',[-0.875 -0.6667 -0.4583 0.6667]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 1]
MF3L='mf3L':'trimf',[-0.5417 -0.3333 -0.125 0.6667]
MF4U='Z':'trimf',[-0.3333 0 0.3333 1]
MF4L='mf4L':'trimf',[-0.2083 0 0.2083 0.6667]
MF5U='PS':'trimf',[0 0.3333 0.6667 1]
MF5L='mf5L':'trimf',[0.125 0.3333 0.5417 0.6667]
MF6U='PM':'trimf',[0.3333 0.6667 1 1]
MF6L='mf6L':'trimf',[0.4583 0.6667 0.875 0.6667]
MF7U='PB':'smf',[0.763 0.9426 1]
MF7L='mf7L':'smf',[0.8578 0.9656 0.6667]

[Input2]
Name='dError'
Range=[-1 1]
NumMFs=7
MF1U='NB':'zmf',[-0.9426 -0.763 1]
MF1L='mf1L':'zmf',[-0.9656 -0.8578 0.6667]
MF2U='NM':'trimf',[-1 -0.6667 -0.3333 1]
MF2L='mf2L':'trimf',[-0.875 -0.6667 -0.4583 0.6667]
MF3U='NS':'trimf',[-0.6667 -0.3333 0 1]
MF3L='mf3L':'trimf',[-0.5417 -0.3333 -0.125 0.6667]
MF4U='Z':'trimf',[-0.3333 0 0.3333 1]
```

```
MF4L='mf4L':'trimf',[-0.2083 0 0.2083 0.6667]
MF5U='PS':'trimf',[0 0.3333 0.6667 1]
MF5L='mf5L':'trimf',[0.125 0.3333 0.5417 0.6667]
MF6U='PM':'trimf',[0.3333 0.6667 1 1]
MF6L='mf6L':'trimf',[0.4583 0.6667 0.875 0.6667]
MF7U='PB':'smf',[0.763 0.9426 1]
MF7L='mf7L':'smf',[0.8578 0.9656 0.6667]

[Output1]
Name='output1'
Range=[-1 1]
NumMFs=49
MF1='1':'constant',[-1 -1]
MF2='2':'constant',[-1 -1]
MF3='3':'constant',[-1 -1]
MF4='4':'constant',[-1 -1]
MF5='5':'constant',[-1 -1]
MF6='6':'constant',[-0.666666666666667 -0.666666666666667]
MF7='7':'constant',[-0.666666666666667 -0.666666666666667]
MF8='8':'constant',[-1 -1]
MF9='9':'constant',[-1 -1]
MF10='10':'constant',[-1 -1]
MF11='11':'constant',[-0.666666666666667 -0.666666666666667]
MF12='12':'constant',[-0.666666666666667 -0.666666666666667]
MF13='13':'constant',[-0.333333333333333 -0.333333333333333]
MF14='14':'constant',[-0.333333333333333 -0.333333333333333]
MF15='15':'constant',[-1 -1]
MF16='16':'constant',[-0.666666666666667 -0.666666666666667]
MF17='17':'constant',[-0.666666666666667 -0.666666666666667]
MF18='18':'constant',[-0.333333333333333 -0.333333333333333]
MF19='19':'constant',[0 0]
MF20='20':'constant',[0.333333333333333 0.333333333333333]
MF21='21':'constant',[0.333333333333333 0.333333333333333]
MF22='22':'constant',[-0.666666666666667 -0.666666666666667]
MF23='23':'constant',[-0.666666666666667 -0.666666666666667]
MF24='24':'constant',[-0.333333333333333 -0.333333333333333]
MF25='25':'constant',[0 0]
MF26='26':'constant',[0.333333333333333 0.333333333333333]
MF27='27':'constant',[0.666666666666667 0.666666666666667]
MF28='28':'constant',[0.666666666666667 0.666666666666667]
MF29='29':'constant',[-0.333333333333333 -0.333333333333333]
```

```
MF30='30':'constant',[-0.333333333333333 -0.333333333333333]
MF31='31':'constant',[0 0]
MF32='32':'constant',[0.333333333333333 0.333333333333333]
MF33='33':'constant',[0.666666666666667 0.666666666666667]
MF34='34':'constant',[0.666666666666667 0.666666666666667]
MF35='35':'constant',[1 1]
MF36='36':'constant',[0.333333333333333 0.333333333333333]
MF37='37':'constant',[0.333333333333333 0.333333333333333]
MF38='38':'constant',[0.666666666666667 0.666666666666667]
MF39='39':'constant',[0.666666666666667 0.666666666666667]
MF40='40':'constant',[1 1]
MF41='41':'constant',[1 1]
MF42='42':'constant',[1 1]
MF43='43':'constant',[0.666666666666667 0.666666666666667]
MF44='44':'constant',[0.666666666666667 0.666666666666667]
MF45='45':'constant',[1 1]
MF46='46':'constant',[1 1]
MF47='47':'constant',[1 1]
MF48='48':'constant',[1 1]
MF49='49':'constant',[1 1]

[Rules]
1 1, 1 (1) : 1
1 2, 2 (1) : 1
1 3, 3 (1) : 1
1 4, 4 (1) : 1
1 5, 5 (1) : 1
1 6, 6 (1) : 1
1 7, 7 (1) : 1
2 1, 8 (1) : 1
2 2, 9 (1) : 1
2 3, 10 (1) : 1
2 4, 11 (1) : 1
2 5, 12 (1) : 1
2 6, 13 (1) : 1
2 7, 14 (1) : 1
3 1, 15 (1) : 1
3 2, 16 (1) : 1
3 3, 17 (1) : 1
3 4, 18 (1) : 1
3 5, 19 (1) : 1
```

```
3 6, 20 (1) : 1
3 7, 21 (1) : 1
4 1, 22 (1) : 1
4 2, 23 (1) : 1
4 3, 24 (1) : 1
4 4, 25 (1) : 1
4 5, 26 (1) : 1
4 6, 27 (1) : 1
4 7, 28 (1) : 1
5 1, 29 (1) : 1
5 2, 30 (1) : 1
5 3, 31 (1) : 1
5 4, 32 (1) : 1
5 5, 33 (1) : 1
5 6, 34 (1) : 1
5 7, 35 (1) : 1
6 1, 36 (1) : 1
6 2, 37 (1) : 1
6 3, 38 (1) : 1
6 4, 39 (1) : 1
6 5, 40 (1) : 1
6 6, 41 (1) : 1
6 7, 42 (1) : 1
7 1, 43 (1) : 1
7 2, 44 (1) : 1
7 3, 45 (1) : 1
7 4, 46 (1) : 1
7 5, 47 (1) : 1
7 6, 48 (1) : 1
7 7, 49 (1) : 1
```

VITA

Christopher M. Scott
Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA 23529

EDUCATION

Bachelor of Science in Aerospace Engineering, May 2008
Virginia Polytechnic Institute and State University

PROFESSIONAL EXPERIENCE

Newport News Shipbuilding, June 2008 - December 2019
Senior Engineer Responsible for cable management and data analytics on Ford Class aircraft carriers, including shipboard data collection and processing to provide actionable data sets.

Newport News Shipbuilding, December 2019 - Present
Senior Electrical Engineer Responsible for shipboard aviation and afloat network systems integration on Ford Class aircraft carriers, including fiber optic network topology and UPS power distribution design.

PUBLICATIONS

On the Development of a Fuzzy Logic Model-less Aircraft Controller, AIAA Atmospheric Flight Mechanics Conference, AIAA SciTech Forum, 2020

Typeset using LaTeX.