2018

Examination of Gain Scheduling and Fuzzy Controllers with Hybrid Reachability

Aaron W. Fifarek
Wright State University

Examination of Gain Scheduling and Fuzzy Controllers with Hybrid Reachability

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering

by

Aaron W. Fifarek
B.S.C.S., Wright State University, 2002

2018
Wright State University

Wright State University
GRADUATE SCHOOL

December 14, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Aaron W. Fifarek ENTITLED Examination of Gain Scheduling and Fuzzy Controllers with Hybrid Reachability BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Electrical Engineering.

Kuldip S. Rattan, Ph.D.
Thesis Director

Fred Garber, Ph.D.
Interim Chair, Department of
Electrical Engineering

Committee on
Final Examination

Kuldip S. Rattan, Ph.D.

Matthew Clark, M.S.Egr.

Marian Kazimierczuk, Ph.D.

Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Fifarek, Aaron W. M.S.E.E., Department of Electrical Engineering, Wright State University, 2018. *Examination of Gain Scheduling and Fuzzy Controllers with Hybrid Reachability*.

Modern aircraft with nonlinear flight envelopes predominately utilize gain scheduled controllers to provide stability of flight. Using gain scheduled control techniques, nonlinear envelopes can be linearized into collections of linear systems that operate under various system dynamics. Linear controllers approximate the nonlinear response over setpoints of operating conditions which allow traditional linear theory to be applied to maintain stability. Techniques to prove linear stability are well understood and realized in control systems, but when controllers are switched, interpolation methods must be used. Interpolation is necessary as gain scheduled systems do not have foundational switching paradigms as part of their realization and therefore can not naturally guarantee smooth (or stable) transitions. To ensure stability between linear controllers, empirical data must be obtained through test and simulation which adds significant time and fiscal cost to development.

This work examines if fuzzy controllers can provide similar response to that of gain scheduled controllers. By representing controllers as fuzzy representations, transitions between the designed linear setpoints can be smoothed by adding membership functions between defined linear controllers. However, fuzzy control lacks analytical tools to find the stability margins to test the stability of fuzzy systems.

In order to provide assurance of stability and performance concerns, fuzzy controllers are translated into hybrid automata representations. Hybrid Automata (HA) theory, which is gaining popularity to represent cyber-physical systems (CPS), is an extension of finite state machines (finite automata) which blends continuous dynamics with discrete switching conditions. The hybrid representation of the fuzzy system allows reachability tools and formal methods to examine stability and desired performance characteristics. This provides evidence that a fuzzy controller can produce, at a minimum, an equally effective controller.

The goal of this effort is to establish a process to use fuzzy controller design and reachability tools to provide evidence of control system key attributes. The work primarily focuses on using two reachability tools which capture flow-pipe construction in linear models. The first, SpaceEx, uses representations of continuous sets to compute an overapproximation of the reachable states. The second, HyLAA, provides a simulation-equivalent reachability representation. Through reachability evidence generated by these tools, the tested fuzzy systems show that they maintain stability over the entire range of normalized input signal.

Contents

List of Figures

List of Tables

Acknowledgment

I would like to take this opportunity to extend my thanks primarily to my wife, Ellen, for everything that comes with being married to a spouse going back for a graduate technical degree. She provided me comfort when disappointed, encouragement when frustrated, and motivation when I needed it most. It was only with her ability to handle her own professional challenges, be the rock of the family, and pick up all the items that I could not handle while in study that I will achieve this honor.

I would like to thank my boys who were too young to understand the commitment our family made to this process. They unknowingly sacrificed time with me to allow for this degree. It is my hope that they will one day realize that I did the best I could to balance the demands of professional, academic, and family life.

I would like to thank my thesis advisor, Dr. Kuldip Rattan, who I first met as an undergraduate student in 1998. Although it was many years before I would pursue a graduate degree, I remembered fondly his attitude toward his students as an inspiration to me. Through this process, he has steered me towards material that I am proud to present.

I would like to thank Mr. Matthew Clark, who initially convinced me that returning to achieve an engineering graduate degree in control theory was worthwhile. Mr. Clark's tutelage over the years has been critical to my professional growth. But beyond the professional world, his frank discussions about personal challenges allowed me to realize that others also experience balancing academic work with the demands of life are experienced by others.

I would like to thank Dr. Stanley Bak whose expertise in hybrid systems has been an inspiration to this work. Working with Dr. Bak to understand the current and future capabilities of the tools available as well as gaining an understanding of hybrid system reachablity enabled a significant aspect of this work.

Others I would like to thank include the following: Mr. Brian Hulbert, my direct supervisor who approved my commitment to achieving this degree. Dr. Derek Kingston, my

technical area lead whose guidance on my topic selection was influential. Ms. Erika Hoffman for working with me professionally and in the academic classes. Mr. Joseph Plantz and Mr. Matthew Rickey for all of the hours that we spent working through challenging course material and preparing for exams.

Finally, I would like to thank my parents who instilled upon me a will to succeed, bragged about me more than I deserved, and always tried to understand when I would talk about my study material.

Dedicated to

My wife, Ellen

Chapter 1

Introduction

1.1 Motivation

Integration of advanced control techniques within modern safety critical systems continue to encounter barriers that prevent widespread use. Although advanced analysis and experimentation has provided mathematical basis for capturing behavior of these systems, certification processes are based about system behavior that must be completely specified and verified prior to approval [9]. For instance, airworthiness standards in MIL-HDBK-516C [1] is highly linear-centric specifically calling out gain margin of 6dB and a phase margin of 45° for acceptable design parameters. For adaptive systems, this information is not known in a manner to allow for meeting this certification criteria [19]. Furthermore, MIL-HDBK-516C [1] also states that the controls laws must maintain "safe(ty) throughout the entire flight envelope" which is problematic due to non-deterministic characteristics of advanced controllers [19]. It is the contention of this author and experts in the Verification and Validation community, that for systems where nondeterministic attributes currently prevent certification, advanced capabilities must be considered in order to leverage cutting-edge control technology [18]. Systems that utilize adaptive control and/or any neural-net/machine learning techniques are found to exist primarily in academic or military

domain due to the relaxation of safety critical concerns in such applications and even then only in limited roles. Most often these control techniques are utilized on systems that do not directly impact the safety of human operators such as on the Joint Direct Attack Munition (JDAM) system [53]. In systems such as the F-16, whose airframe is inherently unstable (relaxed stability, statically unstable), control authority is largely governed on gain scheduling techniques that is highly reliant on experimentation and testing. Unfortunately, due to this reliance, empirical evidence has provided the foundation of trust for gain scheduling techniques. In order to provide mathematical and/or exhaustive evidence of stability guarantees, nonlinear responses have to be designed and tested with linear representations that are both time consuming and expensive. Advances in Lyapunov and Linear Matrix Inequalities (LMI) techniques have helped close this gap but largely requires significant human-in-the-loop effort to achieve. Work by [58] contends that even asymptotically stable systems can experience undesired responses due to wild start-up transients and therefore a supervisor should monitor the performance of the control system.

Modern implementations of advanced control techniques includes a wide array of options including implementation of neural nets, learning algorithms, and genetic fuzzy trees. Each produces attributes that have the potential to lessen the design time, improve the response, and improve the robustness of traditional control system development. Regardless of the advanced system type, they share a commonality of being classified as cyber-physical systems (CPS). Cyber-physical systems is the term for the blending of computer based algorithms that control or react to the physical environment. Although not a new concept, this classification has gained recent momentum in many disciplines due to the inherited benefits that computer integration allows: The integration of digital decision logic to allow advance control techniques.

With the advances in cyber-physical systems (CPS) becoming widely utilized, methods to represent them in a manner that is analyzable has become critical for safety and certification purposes. The challenge to represent such systems is centered around a sin-

gular challenge: the digital domain must discretize the continuous physical world in order to be effective. It is this challenge that has led computer science and engineering fields to refine automata theory to include hybrid automata (HA). Hybrid Automata allows for continuous dynamics to be encapsulated in discrete modes whose invariants dictate the mode validity. It is this union of the continuous and discrete systems that has paved the way for a new class of analysis tools that focus on Reachability.

Reachability analysis utilizes hybrid automata to aid reasoning on the current state of a system and ways for which it can evolve over time. There are a number of techniques that exist to aid this analysis and is an active research area. Most importantly, these tools provide users insight into the capabilities of advanced CPS and how their operating response can evolve over time as they interact with the physical domain.

1.2 Problem Description

In this thesis, the author intends to show that fuzzy controllers can create similar responses to the gain scheduled controllers for a linear plant model. This is important for two aspects: (1) previous work has demonstrated that fuzzy controllers can be directly converted to hybrid systems [15] and (2) fuzzy controllers provide flexible and embedded representation of controller switching conditions.

To show possible equivalency between these control techniques and demonstrate advantages of the alternate representation, the author will seek to answer the following research questions:

1. Gain scheduled systems rely on empirical data between designed controllers to ensure that unsafe conditions do not arise as switching occurs. Can the use of rule-based control paradigms (such as fuzzy controllers) reduce this reliance?

2. Can fuzzy control systems produce similar results as those of a gain scheduled control systems, and if so, is there a process that can be used to ensure equivalent response?

3. Can reachability analysis of hybrid automata that originated from a fuzzy representation provide insight to system performance characteristics?

1.3 Future Research

Should the goals of this thesis be obtained, future efforts can focus on refinement and expansion of the research topic in many different thrust areas where advancement would be beneficial to communities of interest. One such thrust is to convert current nonlinear systems using gain scheduling to be controlled using fuzzy systems. This would allow for direct conversion into hybrid automata and thus enable use of reachability analysis tools on current systems. It would also refine the process to characterize a class of nonlinear control systems. Another thrust could be an examination of current empirical data requirements for systems using gain scheduling. By using rule-based control, fuzzy systems may better represent the switching mechanics necessary for aircraft flight envelope coverage. A reduction on the volume of test points would save on cost while reachability could provide assurance of behavior.

Chapter 2

Background

2.1 Why Advanced Control?

Advancement in technology is a natural progression, but in order to bring value to society, a need must be identified. Such is the case in control theory where advanced techniques are being developed. Simply put, the exponential rise (and incorporation) of cyber-physical systems (CPS) illustrate that benefits exist and should be identified. A few motivating reasons that advanced control research continues today are seen below.

1. **Plant Decay**: Real world systems decay over time causing the designed controllers to *change* effectiveness over time.

2. **Catastrophic Damage**: Modern aircraft often can still operate (at a reduced effectiveness) with damage or failures that exceed any possible experimentation paradigm. Unfortunately, since these failures cannot be easily quantified, the development of explicit controllers are infeasible.

3. **Expansion of Capability**: Learning algorithms could refine control authority as a vehicle is used in the field to supplement idealized design.

2.2 Gain Scheduling

With roots in the flight control and aerospace community [26], gain scheduling has been utilized as a solution to complex control problems, especially in applications where the response is nonlinear. In general, to implement gain scheduling is to utilize techniques to change nonlinear problems into approximated sets of linear problems. Linear problems are well understood by the control community and therefore can be analyzed extensively. The fundamental flaw is that the physical world is rarely linear but is governed by physical properties that create nonlinear representations. Therefore as systems become more complex these approximations become less precise.

In order to best understand the history of gain scheduling controllers, the influential survey papers by Leith and Leithead [34] and by Rugh and Shamma [44] are often considered for coverage of the material. From these papers, it can be derived that the beginnings of gain scheduling as it is known today began in the 1960s but has some roots in the similar (but more simple) approaches in late World War II for V2 rocket control [26].

According to [26], there exists five primary categories of gain scheduling techniques: Classical, Linear Parameter Varying (LPV), Linear Fractional Transformation (LFT), Fuzzy, and modern techniques. In truth, these can be better grouped into two overarching categories of Classical and LPV gain scheduling with other techniques as refinements of each.

Classical gain scheduling

This technique is based on linearizations of a nonlinear system which is described as "Divide and Conquer" gain scheduling design [34]. Also known as "linearization gain scheduling," where the design of the controller requires that linearized family of controllers are paired with that of a linearized family of plants [44]. Using this method, the nonlinear plant is replaced by a set of linearized approximations, each of which can be represented by a linear controller. These designed linear controllers are built around equilibrium points

(also known as design points or set points) [44]. A major advantage of this technique is that it is capable of directly leveraging the benefits of traditional linear control design methods in both the time and frequency domains [26]. With direct linear control design techniques, evaluation of equilibrium point performance can be treated independently using methods that have been historically developed for linear systems. This includes development of PID (Proportional plus Integral plus Derivative) design. Challenges that arise from classical gain scheduling involves the behavior of the system away from the singular set points as the system transitions through the nonlinear system. Specifically, guarantees of the performance response of the system is only mathematically provable about the set points and their asymptotically stable areas of attraction as defined in Lyapunov theory [9]. In systems where the linearization is not appropriately dense, there can be transitional behavior that is not guaranteed. This is noted by [44] that the literature on the classical linearization gain scheduling assumes that controllers are able to be continuously interpolated. In the event that this is not a valid assumption, the closed-loop stability may not be guaranteed. Another challenge to the classical gain scheduling design methodology involves the development of the control for multivariable systems. To properly define such a system, the number of variables becomes a challenge to correctly capture, and even if properly captured, it takes a great deal of effort in design [45].

Fuzzy Gain Scheduling In the development of classical gain scheduling control, how to develop PID-based control is well understood about equilibrium point but lacks the ability to bring in human expertise [58]. To maintain behavior about equilibrium points, there has been a limitation of the slow variation requirement on gain scheduled systems. Fuzzy gain scheduling has sought to relax such restrictions while maintaining the classical gain schedule philosophy [34].

In the literature, typically Fuzzy Gain Scheduling is used to describe methods that utilize fuzzy rules to generate on-line tuning of controller gains [28]. In works such as [27],

existing gain scheduled control set points are introduced to provide data to automatically extract fuzzy rules. Such a technique leverages a Sugeno [50] fuzzy model translated to an ANFIS (Adaptive Network based Fuzzy Inference System) architecture in order to identify the necessary fuzzy rules [27].

Linear Parameter Varying (LPV) Gain Scheduling

According to Leith and Leithead [34], the earliest work on this technique originated from Shamma and Athans in 1988 [47, 45]. This methodology introduced a capability to directly synthesize the controller for a system which is in contrast to designing on a family of linear controllers developed by linear time-invariant techniques [34]. This work sought to capture two heuristic rules that emerged through ad hoc gain schedule development: (1) that the scheduling variable should capture the plant's nonlinearities, and (2) that the scheduling variable should vary slowly [46]. Therefore, LPV gain scheduling control is effectively a refinement of the classical gain scheduling paradigm in that it provides guidance on how to isolate and design to the varying parameters. Furthermore, it is seen in many approaches of LPV that norm based performance measures are utilized to structure frameworks, this is especially true in the use of L_2 norm [34]. Due to the ability of this technique to utilize *a priori* information about a system as well as to capture the exogenous variables as time-varying quantities, LPV has become widely utlized in the control community.

In LPV gain scheduling, most systems can be categorized into two distinct categories on how the slow-varying variables are identified and represented. The two categories are approaches that (1) utilize small-gains or (2) utilize Lyapunov-based techniques [34].

Small-gain Linear Fractional Transformation (LFT) Approaches This is a special case of LPV is the LFT methodology that uses small gain theory to define the parameter-dependent variables in the system [34, 44]. This refinement considered discrete-time systems in the work by Packard [38] and was extended to continuous-time systems by Ap-

karian and Gahinet [3]. In these LPV systems, the system is linear time-invariant with a feedback loop that encloses the time-varying parameter θ [34]. The generalized form of the plant is considered in eq. (2.1) with a feedback of $\beta = \theta\alpha$ where $\theta = blockdiag(\theta_1 I_{r_1} \cdots \theta_K I_{r_K})$ and θ_i represents the i$^{\text{th}}$ time-varying parameter and I_r represents an identity matrix of $r_i \times r_i$ [34].

$$\begin{bmatrix} \dot{x} \\ \alpha \\ e \\ y \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & B_{11} & B_{12} \\ A_{21} & A_{22} & B_{21} & B_{22} \\ C_{11} & C_{12} & D_{11} & D_{12} \\ C_{21} & C_{22} & D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} x \\ \beta \\ d \\ u \end{bmatrix} \tag{2.1}$$

An advantage of LFT approaches is that, with a special case of LFT parameter dependencies, solving linear matrix inequality (LMI) feasibility problems can construct a less complex controller than that of a general LPV case [44]. The disadvantages include: the representation assumption can be restrictive, parameter rate-of-variation bounds are not exploited, and robustness is achieved with gross overestimation of parameter trajectories [44].

Lyapunov-based Approaches In the Lyapunov based functions, design of LPV systems is governed on the ability to prove exponential stability of a system [34]. Lyapunov approaches are typically either in the class of quadratic functions or parameter-dependent functions [34]. In order to prove asymptotically stable systems, application of Barbalat's Theorem is applied to the nonlinear system to determine that change in the system will approach 0 as time approaches infinity. Without such stipulation, that system is not guaranteed to converge to a controllable equilibrium. An advantage of Lyapunov based approaches is that the application of Barbalat's provides strong evidence that the system is controllable. This basis is also applied as a key aspect to adaptive control. As a disadvantage, such calculations can be time consuming to identify the appropriate Lyapunov candidate equations to demonstrate stability. This disadvantage can further manifest itself

with an infinite numbers of constraints that meet the solvability conditions leading to the development of techniques to attempt to mitigate this challenge [34].

2.3 Fuzzy Control

In 1965, Lotfi Zadeh authored a paper on the necessity of fuzzy sets and logic to be included in the overall understanding of the uncertainties of the physical world [54]. The central tenant was the belief that objects in the physical world typically lack disjoint criteria of membership in representative sets. Therefore, attempting to constrain systems into restrictive collections (sets) was an inherently flawed approach to capture complex (and varied) systems. This premise established the concept that objects typically are found to have degrees of membership in interconnected sets [55]. Zadeh's concern was that development of controls for the imprecise "complex large-scale systems" have been affected by "mathematical intractability" [56]. With the evolution of advanced control techniques (i.e. adaptive, genetic algorithms, etc.), efforts to overcome the mathematical challenges are continuing in modern theory. The disadvantage of focusing on strict mathematical solutions to overcome imprecise complex system control is two-fold: (1) physical systems can be extremely complex to accurately characterize, and (2) the ability to leverage human-type refinement of response is often lost. The advantage of fuzzy control allows a relaxation of strict membership set theory to allow for "unsharp boundaries" found in the physical world [54].

2.3.1 What is Fuzzy Logic?

Fuzzy logic is a type of many-valued logic [8] in which set membership is determined to be between 0 and 1 effectively representing a membership percentage [37, 52]. This allows greater flexibility in contrast to traditional Boolean logic [39] in which the system

set membership is defined in independent bins. This is known as "crisp" membership where a value can belong only to true or false members [42]. This capability of multi-valued logic enabled mathematical representation of two phenomena that becomes more prevalent in systems as the complexity of real-world notions are modeled: uncertainty and vagueness. Uncertainty emerges due to the lack of knowledge about the occurrence of some event while vagueness raises due to the grouping together of objects with some ambiguous property [37].

There are two primary forms of fuzzy logic named for their foundational authors: (1) Mamdani and (2) Sugeno. The Mamdani methodology of fuzzy logic seeks to emulate the human expert through the application of a generated set of if-then rules [27, 36]. According to [27], the Mamdani technique has difficulties in formalization due to the design being application-specific. Therefore, it is highly reliant on the input from a human mental model. Even with this challenge, the ability to directly leverage the human expert logical decisions provides flexibility necessary in complex systems. The Sugeno methodology (also labeled as Takagi-Sugeno in some texts) uses the linear design methods such as linear quadratic and robust control techniques as an integral aspect of the fuzzy design method [27, 50]. Consequently, a lessened reliance of the human expert is achieved through alternative control design methodologies. Research to leverage the strengths of both techniques have been presented by [27] through the use of ANFIS (Adaptive Network based Fuzzy Inference) to systematically compute gradient vectors. Work by [41] introduced a new algorithm (KM-LOGIC) to lessen the computational time in order to avoid geometric explosion of calculating fuzzy rules for increasing number of inputs.

It is often best to define a fuzzy set in the mathematical representation. First, let U be the universe of discourse or the universal set which contains all possible elements of concern in a particular context or application. It is characterized by a membership function $\mu_A(x)$ that takes values in the interval $[0, 1]$ [43]. Therefore, since the set can take any value between 0 and 1, it is a continuous membership function.

A set A in U may be represented as ordered pairs with a generic element x as seen in eq. (2.2) [43].

$$A = \{X, \mu_A(x) | x \in U\} \tag{2.2}$$

Fuzzy sets are typically defined either by a list method or by a rule method. The rule method representation is more prevalent due to its natural association with the use of fuzzy sets to capture rules on a membership set. Furthermore, fuzzy sets can exist in many different shapes, chosen in order to best fit the evolution of the set away from the center (the mean value of the set). Some examples of set shapes include triangular, trapezoidal, etc. as seen in fig. 2.1. Due to computational complexity when dealing with higher order set representations, it is typical that triangular and trapezoidal functions are most often utilized.

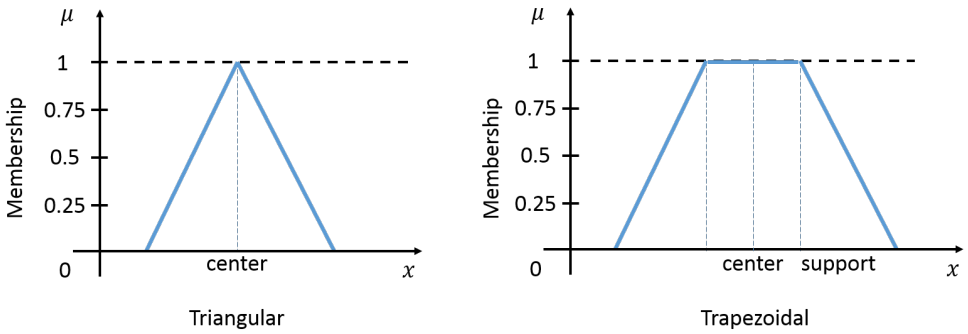

Figure 2.1: The triangular and trapezoidal fuzzy membership [43].

Fuzzy rules define the relationship between the domain of the antecedent U and the domain of the consequent W whether through implication or through functional approximation. To illustrate this relationship, observe an example given in the literature [57, 55] where the age of person is categorized into three bins: young, middle-aged, and old (2.2). In crisp set descriptions, a person would only be considered in one of the ranges at an time and that a universal understanding of the line between young and middle-aged would be necessary. These categories are not universally known, but gradually one transitions from

young to old throughout life.

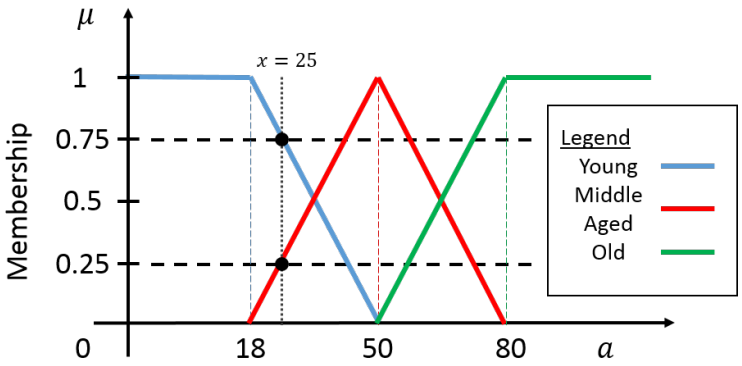

Figure 2.2: Representation of young, middle-aged, and old triangular membership functions.

2.3.2 Applications to Controls

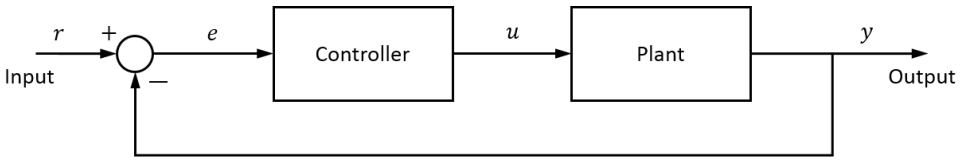

Figure 2.3: Traditional basic closed loop control system.

In control theory, a system (*plant*) is desired to operate in a specific way given specific input. This plant can represent systems that are not designed or accessible to an engineer, so to achieve specific behavior, a *controller* is introduced to change the input to the plant in a way for plant to provide desired output. To further enhance this paradigm, *feedback* is introduced that can provide the amount of *error* from the desired reference to that of the actual system output. Challenges emerge when designers move away from the static theoretical plants to those of the physical world with a number of non-trivial realizations: (1) the real-world is rarely captured via linear representation, (2) that a plant can change over its lifecycle, and (3) desired response must be able to handle flawed or imprecise input [57].

History demonstrated that gain scheduling attempted to handle a number of these challenges but it fundamentally encounters barriers by not having a mechanism to blend transitions between designed responses. Therefore, challenges such as switching between linear controls made it necessary to interpolate *between* differing design points. By using the fuzzy logic paradigm, control systems could introduce an explicit understanding of how to control transitions between membership sets while providing a mechanism to incorporate human design expertise into the control itself [57].

2.3.3 Building Fuzzy Controllers

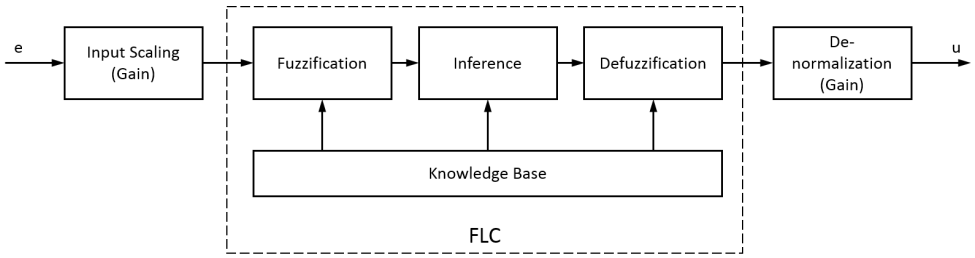

Figure 2.4: Fuzzy Logic Controller (FLC) Block Diagram.

In the basic closed loop control design, an input reference is compared with an output signal. A controller uses the difference in the signals to produce a signal that, when directed to a plant, will build a correlation between the output and reference signal. In order to implement a fuzzy control system, a Fuzzy Logic Controller (FLC) (fig. 2.4) must represent the controller in the closed-loop system (fig. 2.5). As a best practice, the FLC is surround by an input scaling block and a denormalization block. The input scaling block normalizes the input error signal (e) to a range of $[-1, 1]$ which closely matches the degree of membership in a fuzzy membership function. After the FLC provides a modification to error (e) by generating a control signal (u), the signal needs to be denormalized. The denormalization block returns the resulting control signal (u) to the appropriate range for the plant input. This best-practice is also demonstrated in figs. 2.4 and 2.5.

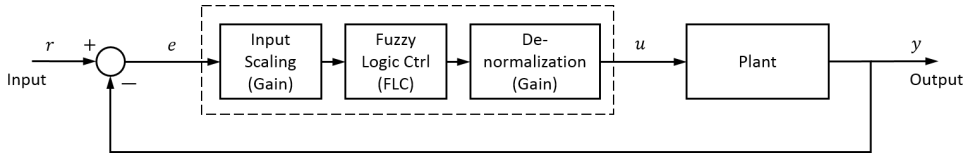

Figure 2.5: Implementation of a FLC into the basic closed-loop diagram.

In order to build a FLC, there are three steps that have to be followed (fig. 2.6):

1. **Fuzzification**: Identify the membership center points to generate an input fuzzy set.

2. **Specify a Rule Base and Inference Procedure**: Determine a set of rules and how they will be applied to the system. This rule base is used by the inference procedure to provide output fuzzy sets.

3. **Defuzzification**: Determine the procedural method to convert the output fuzzy sets to a crisp output.

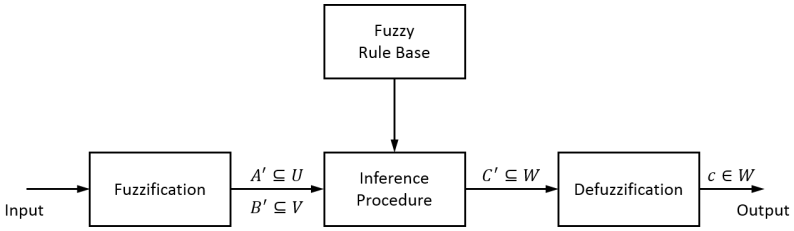

Figure 2.6: Linguistic variable and fuzzy inference.

In order to illustrate the steps defined, an example of a 1-input, 1-output fuzzy controller will be discussed.

2.3.4 Example Fuzzy System

In fuzzy systems, there is a correlation between the number of inputs with the number of input fuzzy sets as well as the number of outputs with the output fuzzy sets. In a single input, single output (SISO) fuzzy system, there is one input fuzzy set and one output fuzzy set which makes up the system knowledge base. Using these sets, as well as inference of

how they relate, allows the construction of an output fuzzy set which is defuzzified into a crisp value representation. One common representation of that SISO fuzzy system is that of a Proportional Fuzzy Logic Controller (PFLC) [43].

In this example, let us define three fuzzy membership functions to represent both the input (fig. 2.7) and output (fig. 2.8) fuzzy sets of a PFLC; Negative, Zero, and Positive.

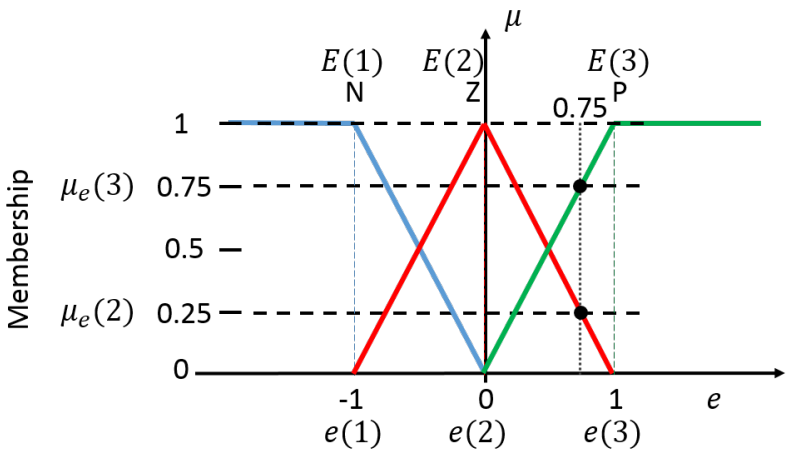

Figure 2.7: Input fuzzy set with three triangular membership functions.

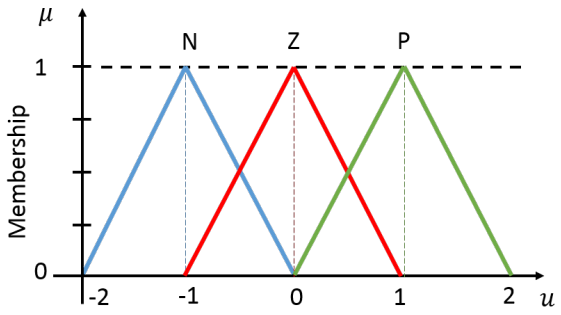

Figure 2.8: Output fuzzy set with three triangular membership functions.

Fuzzification

Fuzzification allows for smooth transitions between neighboring subspaces. It converts the crisp measured values into membership values in fuzzy membership functions. In fig. 2.7,

the input fuzzy sets are defined as N, Z, P (negative, zero, positive) membership functions with respect to the input variable e in a fuzzy partitioning arrangement normalized in $[-1, 1]$. The center membership function, zero (Z) is centered at zero, the negative (N) is centered at -1, and finally positive (P) is centered at 1. In a fuzzy partitioning system it is assumed that the width of an input membership values functions extends to the center point (also known as the peak value) of adjacent membership functions [42]. One advantage of using such a technique is that any input in the universe of discourse has membership in only two membership functions with a sum of membership values equal to one. To determine membership, let it be supposed that $e(j)$ is the center-point of the membership function $E(j)$ and $e(j+1)$ would represent the center point of the adjacent fuzzy set $E(j+1)$. Therefore, the membership values are then calculated using eqs. (2.3) and (2.4).

$$\mu_{E(j)} = \frac{e_1(j+1) - e_1}{e_1(j+1) - e_1(j)} \tag{2.3}$$

$$\mu_{E_(j+1)} = \frac{e_1 - e_1(j)}{e_1(j+1) - e_1(j)} \tag{2.4}$$

Analysis of fig. 2.7 for an input fuzzy set shows a case where $e = 0.75$. When this value of e is provided to the input fuzzy set, the amount of membership in each triangular fuzzy function can be determined. At the aforementioned value of e, the following membership values can be identified with respect to the negative (N), zero (Z), and postive (P) input membership functions:

- Value e has no membership in N ($\mu_{E(1)} = 0$).
- Value e has partial membership in Z ($\mu_{E(2)} = 0.25$).
- Value e has partial membership in P ($\mu_{E(3)} = 0.75$).

This membership can be represented in matrix form:

$$\mu_e = \underbrace{\begin{bmatrix} \mu_N & \mu_Z & \mu_P \end{bmatrix}}_{InputSet} = \begin{bmatrix} 0 & 0.25 & 0.75 \end{bmatrix} \tag{2.5}$$

Moving away from crisp gain values allows greater flexibility in control by introducing "continuous evolution" of gains. This more closely resembles what would be encountered in the physical world where data is often imprecise. With fuzzification, imprecision can be better tolerated since adjacent gains can partially contribute to the value necessary for the response. Furthermore, smooth transitions provide mathematical protection against issues with discrete switching. Fuzzification formalizes the continuous subspace transistions.

Linguistic Rules and Inference Design

Fuzzification allows for continuous transitions of input in input fuzzy sets, but the inference design enables generation of a rule base to describe the relationship between input and output fuzzy sets. Humans use relational rule sets often. Inference based on a knowledge base provides a mathematical way to enable control systems to relate input and output fuzzy sets [57].

Returning to the PFLC already introduced, the defined input and output fuzzy sets provide the rule base for a single input single output. It is important to note that each rule is considered a piece of information and the results are combined with the logical OR conjunction of the statements as seen below.

- If input e is N then the output u is N **-OR-**
- If input e is Z then the output u is Z **-OR-**
- If input e is P then the output u is P

This forms the general case of the rules as "$R(j)$: If input e is $E(j)$ then output u is $U(j)$."

These rules not only emulate how humans reason but has the flexibility to take on multiple inputs. The rules can be written in a table form as shown table 2.1.

Table 2.1: Input/Output fuzzy membership table.

	input $E(j)$		
	N	Z	P
ouput $U(j)$	N	Z	P

Fuzzy inference generates the fuzzy output set which combines the strength of each rule with the output membership function [42, 43]. In a fuzzy system, the input fuzzy set provides the antecedent of the rules with the output fuzzy set providing the consequent.

There are many fuzzy inference methods. Two of the most popular methods used for fuzzy inference include min-max and product-sum. In the min-max method (mathematically shown in eq. (2.6)) the union (by OR statements) of the resulting set of minimums fig. 2.9 generates the output fuzzy set by generating the maximum combined set area fig. 2.10.

$$U(x) = max[min(\mu_e(index_1), U(index_1)), min(\mu_e(index_1 + 1), U(index_1 + 1))] \tag{2.6}$$

This technique creates a shape in a generated output fuzzy set which is enclosed. The area of this enclosed region can be calculated (in the defuzzification step) to produce the crisp output value. A disadvantage of this inference technique emerges in the defuzzification step because of the computational cost of determining the area of the region.

In the product-sum method (eq. (2.7)), this inference technique scales the individual fuzzy sets according to their membership values (fig. 2.11). After the individual memberships are scaled, they are then summed together to form the output fuzzy set curve that will

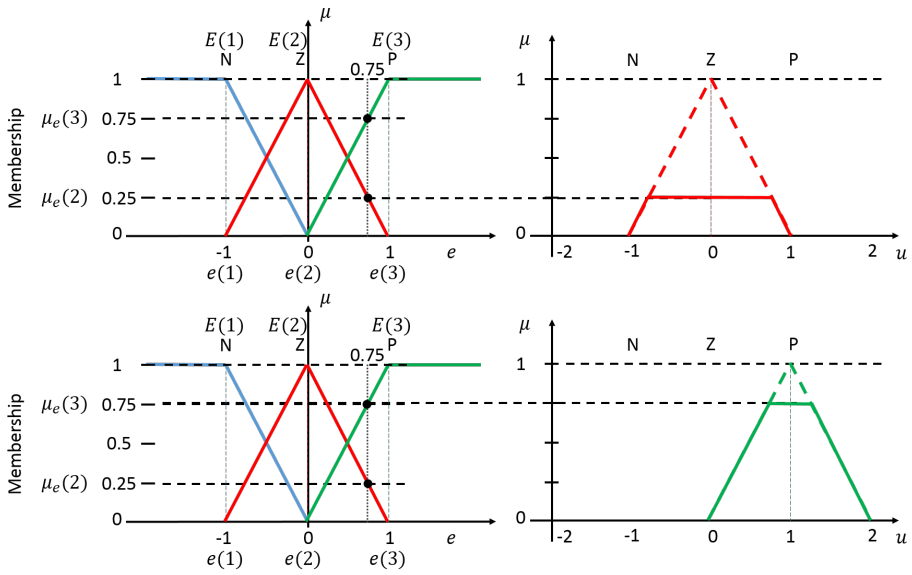

Figure 2.9: The minimum fuzzy membership of each the active sets.

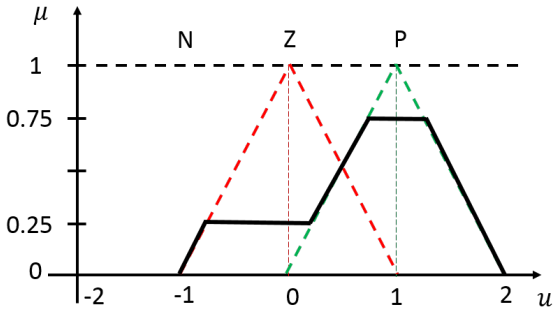

Figure 2.10: Min-Max inference with the aggregation to the fuzzy output set.

be used in the defuzzification process fig. 2.12.

$$U(x) = \mu_e(index_1) \times U(index_1) + \mu_e(index_1 + 1) \times U(index_1 + 1) \qquad (2.7)$$

An additional inference technique which is used in the PFLC example is that of a product-max method. Shown mathematically in eq. (2.8), this technique uses similar aspects of the previous two methods (min-max and product-sum). Again, the first step in the

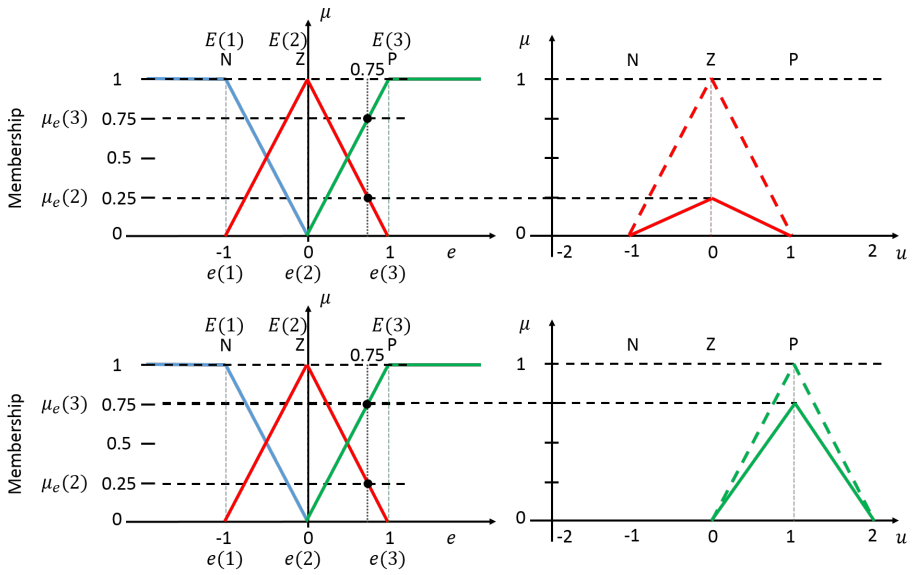

Figure 2.11: Scaling of the membership values of the output fuzzy set using the product-sum methodology.

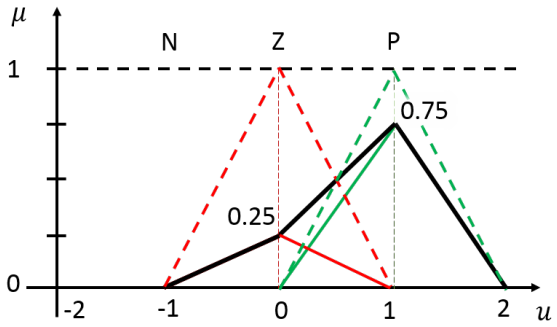

Figure 2.12: Product-Sum inference method with aggregation to a fuzzy output set.

inference is the product scaling of the individual fuzzy membership (recall fig. 2.11). The next step is to use the maximum of the aggregation of the active fuzzy sets fig. 2.13.

$$U(x) = max[\mu_e(index_1) \times U(index_1), \mu_e(index_1 + 1) \times U(index_1 + 1)] \qquad (2.8)$$

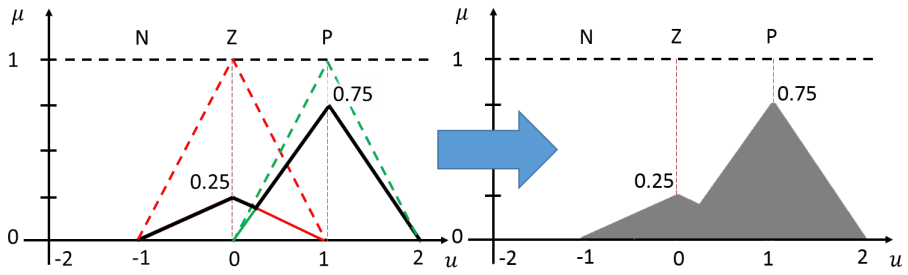

Figure 2.13: Product-Max inference method with shaded area under the enclosed.

Defuzzification

This step in the fuzzy control implementation involves the translation of the resulting output fuzzy set back into a crisp value for use of the system. To best illustrate in a proportional controller, the gain (K_p) is a single value. There are many ways to obtain the precise value from the union of adjoining membership sets. Leekwijck-Kerre states that the most applicable method to defuzzify depends highly on the underlying stucture of the universe for which a system is based [33]. With respect to control systems, two often used techniques are that of the center of gravity(area) (eq. (2.9)) and the weighted average (eq. (2.10)) [43].

In the center of gravity defuzzfication method, the aggregation of the area of adjacent output membership functions are combined in the output fuzzy set to determine the center of gravity. This as summarized in fig. 2.14 where the min-max, product-sum, and product-max inference techniques provide bounded areas. In order to calculate the center of gravity, the area under the memberships are combined and scaled to the degree of membership. This combined area produces a crisp output value at the center that is used for the system (eq. (2.9)).

$$\text{output} = \frac{\int u\, \mu_u(u)\, \mathrm{d}u}{\int \mu_u(u)\, \mathrm{d}u} \tag{2.9}$$

Although this technique, as in any of aforementioned inference techniques, will produce proper crisp output, the computational load is complicated by the need to integrate to find

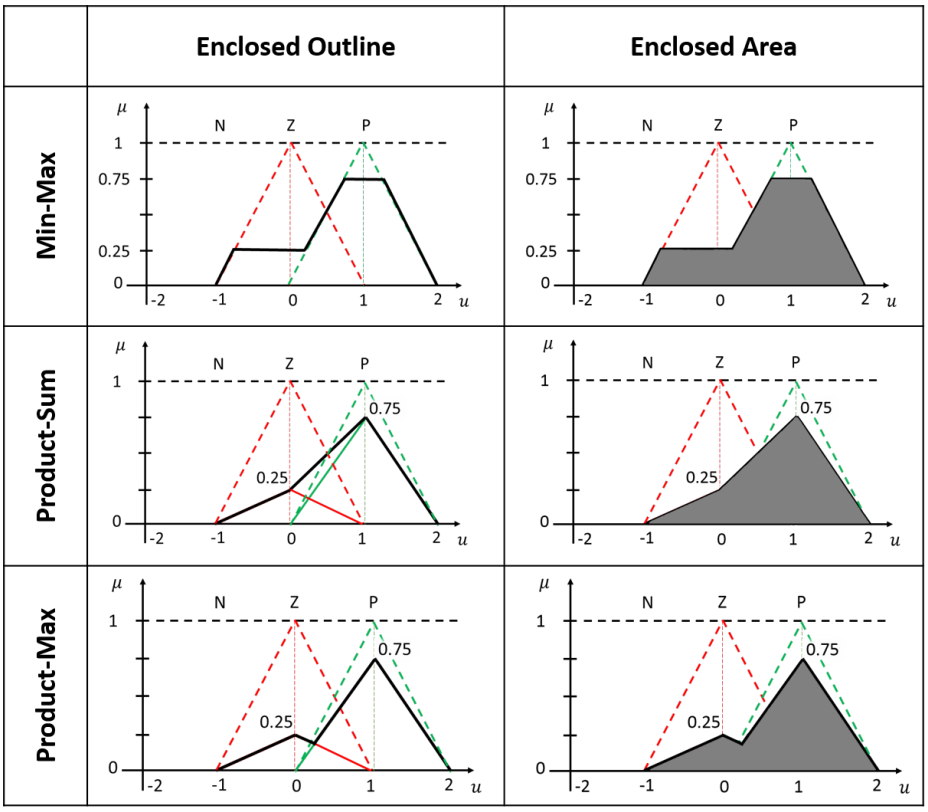

Figure 2.14: Enclosed area of an aggregation of fuzzy output sets as defined by min-max, product-sum, and product-max inference methods.

the area. It was this limitation that led to the second of the defuzzication techniques, Weighted Average.

The weighted average inference technique (eq. (2.10)) uses a version of the product-sum method to defuzzifiy the sets (see figs. 2.15 and 2.16).

$$\text{output} = \frac{\sum_{i=1}^{n} c_i^o \mu_e(i)}{\sum_{i=1}^{n} \mu_e(i)} \tag{2.10}$$

This improves efficiency by averaging the scaled centers of the fuzzy memberships while eliminating the denominator from calculation (it is always 1). Furthermore, it allows use of matrix operations. In matrix form, as seen in eq. (2.11), the value to $e = 0.75$ will produce a row vector of the input fuzzy set membership with a column vector of the rule vector (output fuzzy membership functions). The values in the row vector describes the degree of

membership which is only active in two adjacent input sets. Coupled with the adjacent sets of the output membership sets, the matrix multiplication can be simplified with a reduction to a $(1 \times 2)(2 \times 1)$ calculation through elimination of the N term that has no membership in this value of u (eq. (2.12)).

$$u = \underbrace{\begin{bmatrix} N & Z & P \end{bmatrix}}_{InputSet} \times \underbrace{\begin{bmatrix} N_{out} \\ Z_{out} \\ P_{out} \end{bmatrix}}_{OutputSet} = \begin{bmatrix} 0 & 0.25 & 0.75 \end{bmatrix} \times \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = 0.75 \tag{2.11}$$

Simplified representation:

$$u = \begin{bmatrix} 0.25 & 0.75 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0.75 \tag{2.12}$$

This simplification to only active membership sets becomes more critical as the number of input membership functions increases.

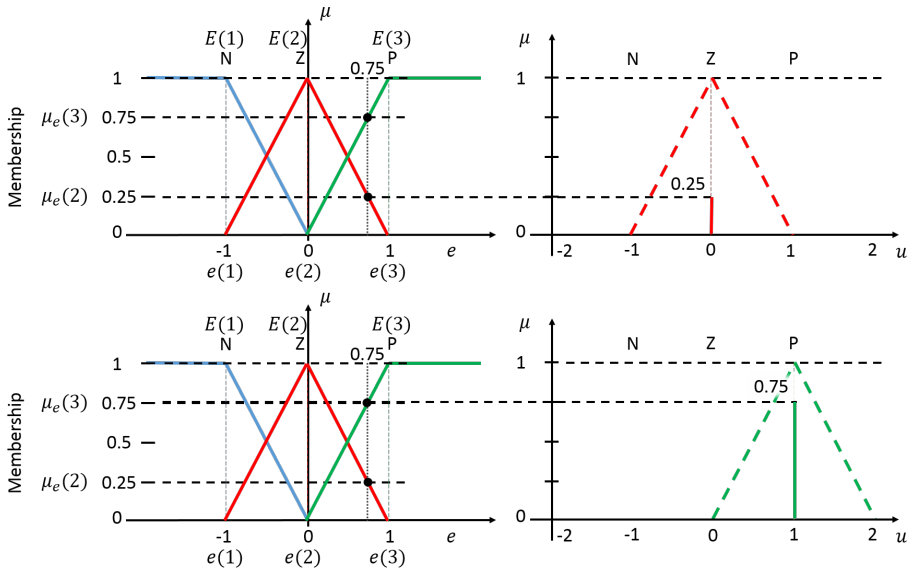

Figure 2.15: Scaling of the membership values using the weighted average inference methodology.

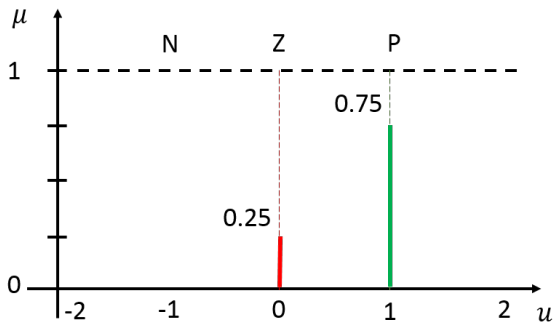

Figure 2.16: Combined scaled memberships creating the output fuzzy set in the weighted average methodology.

Control Surface Generation

To further describe the output of the fuzzy control system, the generation of a control surface provides insight on how the system evolves dependent on its membership variables. In the case of the three fuzzy set example, this is a relationship between the input e (horizontal axis) and that of the output u (vertical axis). For an equally spaced input fuzzy set (one-input, one-output), the control surface will appear as a single line over the domain of the output fuzzy set (fig. 2.17). If the there are additional input fuzzy sets (greater than three) demonstration of piecewise linear visualization can be realized throughout the surface. This will be demonstrated in greater detail in the gain schedule to fuzzy system example. With the increase in the number of input fuzzy sets that define a system, this surface changes from that of a line (1-input, 1-output) to that of a surface (2-input, 1-output).

2.3.5 Gain Scheduled to Fuzzy Conversion

When converting from a gain scheduled controller to that of a fuzzy control, it is important to maintain switch gains based on the "slow dynamics" of the system. This is critical because these dynamics establish the gain tables. For example, in aircraft control, altitude and mach number are two of the primary "slow dynamics" utilized to generate the flight envelope trim conditions. These trim conditions define the linear set points of the system

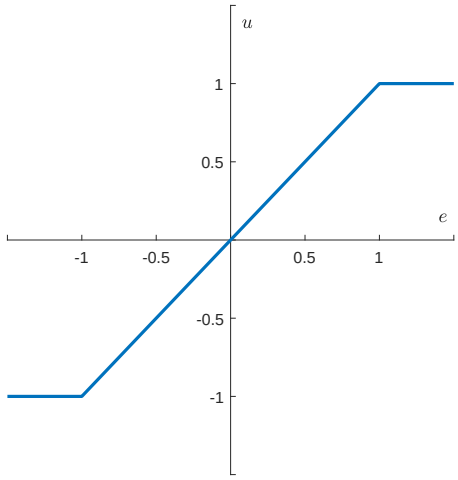

Figure 2.17: The generated control surface for the PFLC example fuzzy controller (one-input, one-output).

which are resilient to a level of noise when active. As previously established, transitions between linear set points are typically handled with linear interpolation to gradually change the gain over some range instead of utilizing dramatic jumps.

Converting from an established gain scheduled controller to an equivalent fuzzy controller requires that the interpolation between setpoints is maintained in the translation. In order to demonstrate the conversion the following linear plant in eq. (2.13) will be analyzed using only proportional control as a PFLC.

$$G(s) = \frac{10}{s^2 + 3.6s + 10} \tag{2.13}$$

In this theoretical gain scheduled controller, the two different setpoints are defined with gains of $K_{p1} = 3$ and $K_{p2} = 30$ with disjoint jumps between the K_p values at ± 0.2 (see fig. 2.18a).

Here, the gain scheduled controller is linearly interpolated by $e = \pm 0.05$ about the switching condition (see fig. 2.18b). Using this interpolation technique lessens the risk of instability during the transition by reducing the variability between the two gains. The

gains here need to be maintained when conversion to the fuzzy controller.

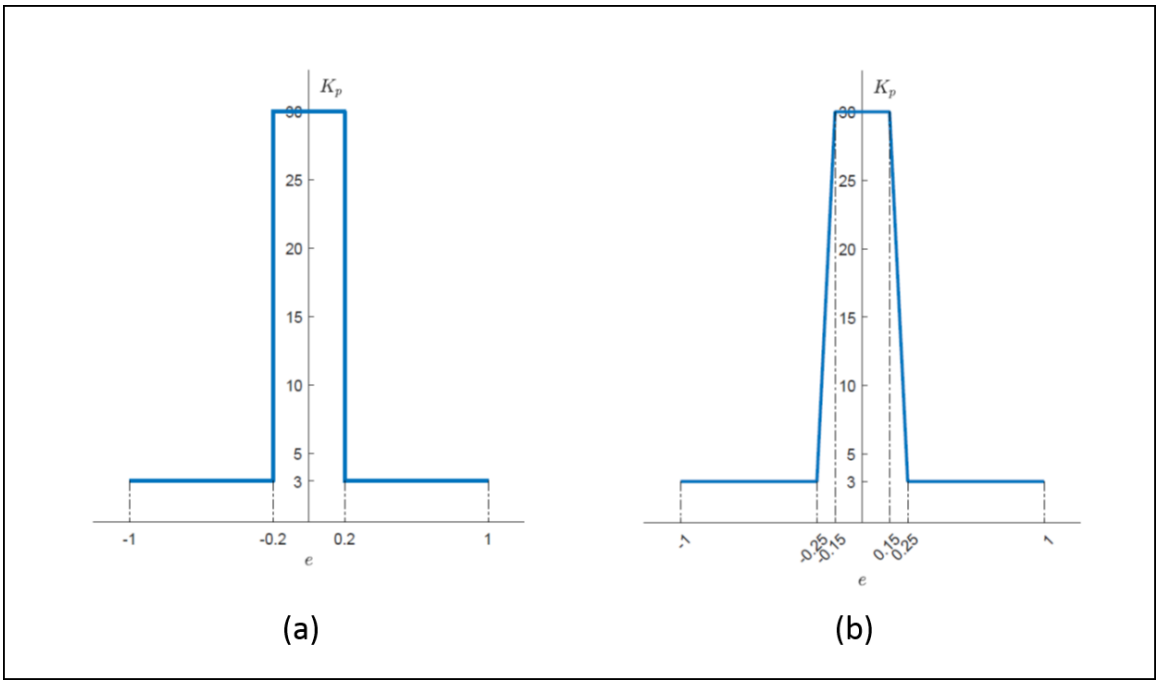

Figure 2.18: Gain scheduled controller with (a) discrete gains K_p values over a specific error (e) range and (b) linear interpolation of gain changes over the discrete jump error (e) values.

Now that the gain scheduled controller is defined with linear interpolation, an equivalent triangular fuzzy controller can be generated for this PFLC example. The horizontal areas of the graph show areas within the error (e) range which the gain values remains constant. The necessary gain value at the specific error values (as seen in fig. 2.18b) provides the basis of the knowledge base that is used to create the input and output membership fuzzy sets. To create the knowledge base, the input and output fuzzy membership functions are defined. The process to establish these sets in a proportional control deals with finding the correlation between the necessary gain for the error range. This will create a control surface that can be examined. Upon inspection of fig. 2.18b, identifying the correlation of the error value input to the output gain value is done by building a control surface. Since this is a proportional control system, there is only a continuous line between the error range.

In surface construction, the slope is defined by the gain that is present in that line segment. Therefore, by using the gain (K_p) as the associated slope of the line, a correlated output value can be determined for each piecewise section of the controller. One method to find the piecewise affine line segment is that of traditional slope-intercept form ($y = mx + b$) with a change terms with respect to the axes. These axes are based on the error (e) and the output (u) terms substituting for x and y. The constant term (b) is also relabeled as C. Implementing the changes in terminology changes the equations to $u = me + C$. Using the point-slope calculations the generated surface plot is shown in fig. 2.19. The relationship

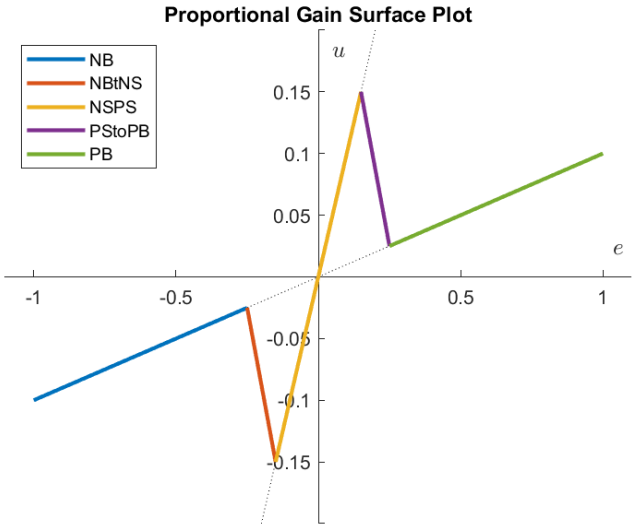

Figure 2.19: Gain surface plot for the 2 gain system.

between the control input error e and the control output u, input fuzzy set fig. 2.20 and output fuzzy rule base fig. 2.21 can be obtained.

The generation of these set point values establish the centers of the triangular fuzzy output set. Furthermore, graphing of these center values as a 2d surface shows the relationship between the error e from the closed loop systema and the output u from the controller.

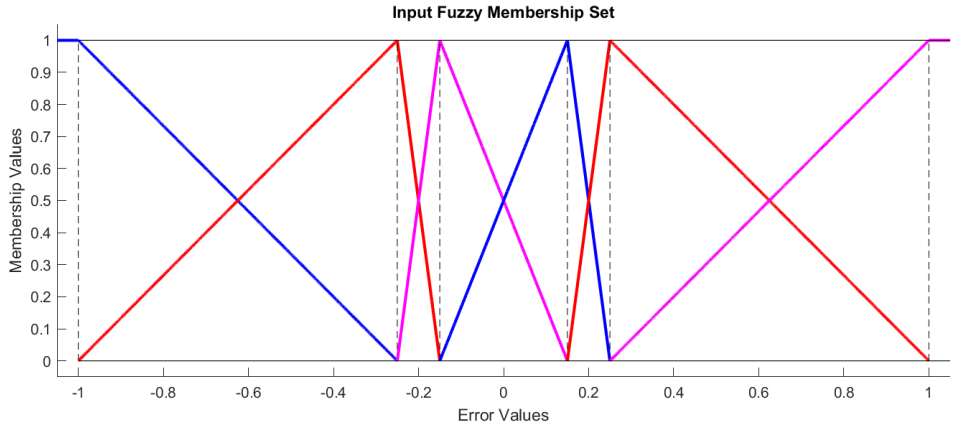

Figure 2.20: Input fuzzy membership set for the example PFLC with a gain scheduled switch at ± 0.2.

	input $E(j)$					
	NB	NBtoNS	NS	PS	PStoPB	PB
output $U(j)$	NM	NS	NB	PB	PS	PM

For example, when referring to section 2.3.5 with a value of $e = 0.12$ crisp input is presented to the fuzzification block of the knowledge base to realize membership in negative-small (NS) and positive-small (PS) for the input fuzzy membership set. Using the product-sum inference technique, it is shown that the output fuzzy membership aligns input NS to output NB and input PS to output PB.

Now that the inference is established, the defuzzification either by center of gravity or weighted average can be done. In order to ensure that the systems are functionally equivalent, Simulink was used to examine the response to a step input. Each technique was able to provide an identical response (see fig. 2.22).

2.3.6 Fuzzy to Piecewise Linear

In order to examine the behavior of cyber-physical systems, utilizing hybrid automata to embed continuous dynamics with discrete switching conditions has possibilities to conduct

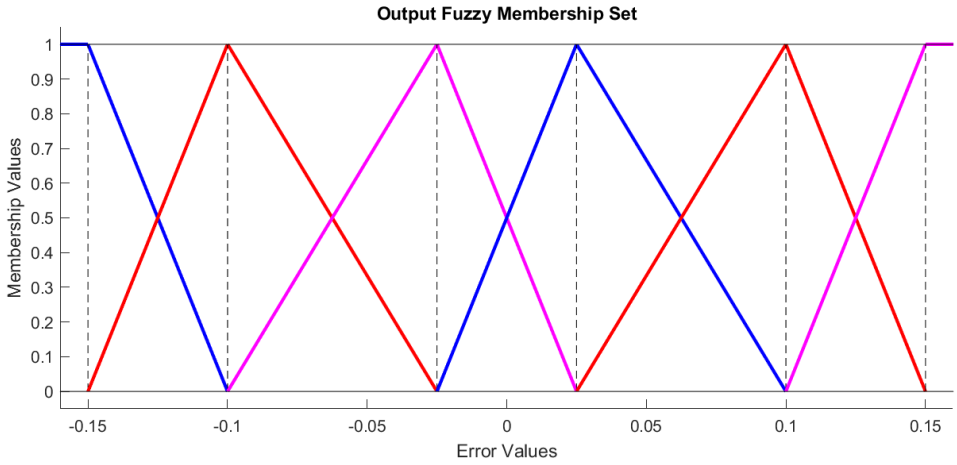

Figure 2.21: Output fuzzy membership set for the example PFLC with a gain scheduled switch at ± 0.2.

various analysis techniques such as system reachability. Hybrid systems are explained in greater detail in section 2.4. First, let us introduce a relationship between particular fuzzy set design and that of hybrid automata.

As a foundation to translate systems from fuzzy control to hybrid automata, previous authors have noted that a relationship between such systems exist [10, 35], even identifying that a fuzzy control system as a special class of hybrid dynamical systems [21]. Examination by [48] showed that piecewise linear hybrid systems may cover a hybrid system with over-approximation. Work by [12] showed that a fuzzy controller can be represented by piecewise affine (PWA) hybrid automata [15, 17].

In order to translate a fuzzy controller into a PWA hybrid system, each fuzzy inference rule would have to be represented by an equivalent affine controller [15].

- The affine controller can be represented within the closed loop system as a vector field of continuous dynamics.
- The finite set of discrete variables are represented by the regions between each fuzzy input inference vectors.
- The center points of the fuzzy input inference vectors represent guard conditions.

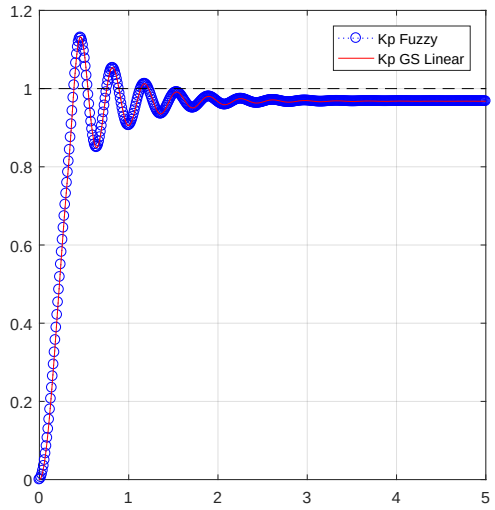

Figure 2.22: System response with a gain scheduled and fuzzy controller applied to identical plants with identical step inputs.

It has been shown that an equally spaced input fuzzy set is equivalent to a classical controller with constant gain, i.e., linear controller [11]. The center point of the input membership functions is placed at the transition points of the gain. The value of gain determines the center points of the output fuzzy set membership function. To switch smoothly from one gain to another, a membership function is added at the transition points. The duration of the membership function is kept small to quickly transition between gains. It has been shown that during transition, the output of the controller is a linear function of the input with an additional constant term.

With a representation of original gain scheduled controller as a fuzzy controller, accessibility to a hybrid representation can be achieved. Work previously conducted by Clark and Rattan [15] demonstrate that fuzzy sets can be directly converted to hybrid systems.

In the example, the input fuzzy sets (defined in fig. 2.20), determine the continuous dynamics that each hybrid state should hold valid. Therefore the invariants of states should not violate the conditions as defined in table 2.2.

Each mode represents a single location in the hybrid system. To determine the piece-

Table 2.2: System modes of the PFLC example system using piecewise linear calculations.

Mode	e range
1	$e < -0.25$
2	$-0.25 \leq e < -0.15$
3	$-0.15 \leq e \leq 0.15$
4	$0.15 < e \leq 0.25$
5	$e > 0.25$

wise linear representation of the hybrid system from the fuzzy controller, solve for $y(m)$ (eq. (2.14)) using $k(m)$ (eq. (2.15)) and $c(m)$ (eq. (2.16)) where m is the mode number. The mode number is determined by the a pair of input fuzzy membership functions. In the example, the fuzzy membership function at *mode 1* defines the input membership function the mode that spans NB (m) and NStoNB ($m+1$).

$$y(m) = k(m)x_1 + c(m) \tag{2.14}$$

$$k(m) = \frac{u(m+1) - u(m)}{x_1(m+1) - x_1(m)} \tag{2.15}$$

$$c(m) = \frac{x_1(m+1)u(m) - x_1(m)u(m+1)}{x_1(m+1) - x_1(m)} \tag{2.16}$$

2.4 Hybrid Systems

To capture the increasing interactions between the natural world (which is continuous) and the digital systems (which are discrete), use of hybrid systems have come to the forefront. The notation for such systems defined Hybrid Automata which characterizes the related behavior of such systems [24].

2.4.1 Finite State Machine

To understand Hybrid Automata, familiarity with Finite State Machine (FSM) notation is beneficial. A finite state machine captures an abstract representation of how specific changes applied to a machine state that evolves the overall behavior of a system. In this nomenclature, states represent the general condition of the machine with specific (finite) inputs that can transition to alternate states. Through such a representation, abstract behavior is represented by allowing common inputs to represent different effects on the system.

An example of a finite state machine can be seen in "Discrete Mathematics with Applications" which describes a turnstile [29]. In the example, the finite state machine is made of two finite states: *locked* and *unlocked*. A user wishing to move through the turnstile must insert a token which would allow the individual to pass. Initially, the turnstile is *locked* where, no matter how often the user attempts to *push the bar*, it will not move. If the user was to *insert a token*, the state of the machine would transition to an *unlocked* state. Should additional tokens be inserted, the state of the machine would still remain *unlocked*. Once a user does proceed through the turnstile (*pushes the bar*) the system transitions back to the *locked* state where it will remain until another token is inserted.

This example establishes that there are two finite states of the system: (1) locked(L) and (2) unlocked(UL). These states are directly influenced by two different inputs: (1) pushing the turnstile bar and (2) inserting a token. The FSM is represented in fig. 2.23.

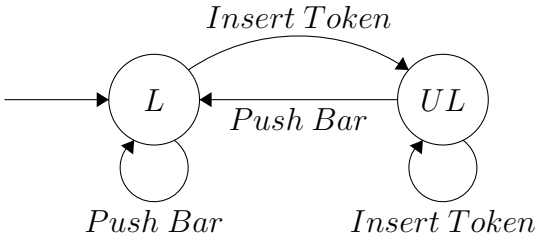

Figure 2.23: Simple finite state machine example of a turnstile [29].

Reflecting on fig. 2.23, note that there are no dynamics attributed to the states of the turnstile. At each state the system is either *locked* or *unlocked*. A limitation here is that

many systems exist where the evolution of the states will change over a time horizon. For such systems, the basis of a FSM design must be extended to include continuous dynamics.

2.4.2 Hybrid Automata

A hybrid automaton is a formal model which models both the continuous and discrete components of a system [24]. An extension of finite state machines, hybrid automaton adds continuous dynamics to the modes (states) of the system while maintaining the discrete transitions between each state. Continuous state dynamics allow for more complex behavior than that what is possible in a finite state machines with applications into cyber-physical systems. In cyber-physical systems, digital controllers (computers, etc.) govern discrete transitions between the interpreted states of the system. The states themselves are not bound by discrete dynamics but are able to evolve in a continuous fashion. These systems require a formal model to analyze the relationship between the continuous change of state with discrete decisions to alter the state. In the notation of hybrid automata, continuous evolutions of state are considered *flows* while the discrete state changes are considered *jumps* [24].

The main aspects of a hybrid automaton H are as follows [24]:

1. **Variables**: A finite set $X = \{x_1, ...x_n\}$ of real numbers where n is the *dimension* of H. The dervatives of variables are written $\dot{X} = \{\dot{x}_1, ...\dot{x}_n\}$. For primed (values at the conclusion of discrete steps) it is written $X' = \{x'_1, ...x'_n\}$

2. **Control Graph**: This is a finite directed multigraph (V, E) where the vertices in V are called *control modes* and the edges in E are called *control switches*.

3. **Initial, invariant, flow conditions**: These are the three predicates that define each control mode $(v \in V)$. These define the conditions in which a mode is valid in the continuous domain.

- *init(v)*: The *initial condition* is the entry state of the automaton where execution begins. It is a predicate whose free variables are from X
- *inv(v)*: The *invariant condition* is the acceptable range of variables that allow the state to be active. It is a predicate whose free variables are from X
- *flow(v)*: The *flow conditions* is the continuous dynamics that are active at each state. It is a predicate whose free variables are from $\dot{X} \bigcup X$

4. **Jump conditions**: Assigned to edges (control switches) $e \in E$ to represent the discrete transitions. Each edge jump condition ($jump(e)$) is a predicate of whose free variables are in $X \bigcup X'$

5. **Reset** A command to return the automaton to the initial state.

In hybrid automata, there are numerous introductory examples that aid in understanding the concepts and provide insight on how they are extended from finite state machines. One often used example is that of a heater thermostat which contains two modes (see fig. 2.24). In mode *Off* the heater is off while in mode *On* the heater is on. While the heater is *Off*, the temperature of the environment will be falling according to the flow condition $\dot{x} = -0.1x$. While the heater is *On*, the temperature of the environment will rise according the flow condition $\dot{x} = 5 - 01.x$. Initially, the heater is *Off* and the temperature of the environment is at 20 degrees. The jump (also known as the *guard*) condition states that when the temperature is less than 19 degrees the system can transition (or jump) to a different continuous dynamics (different mode). Observe the invariant of the *Off* mode. It is defined as the room temperature must be greater than or equal to 18 degrees $(x >= 18)$ meaning that once the dynamics reach 18 degrees, a transition (jump) must occur [24].

In hybrid automata, the relationship between mode invariants and jump conditions (guards) represents a source of nondeterminism. A guard only provides an opportunity for modes to transition, it is the mode invariants that force transitions. As long as the current mode invariant remains valid, the system does not need to transition from that mode. It is

only when the invariant is violated *must* a jump occur. This can be seen in the thermostat example (fig. 2.24). If the system is currently in the *Off* mode, the temperature (x) variable can demonstrate the nondeterminism. For instance, when ($x = 18.5$) the transition can be taken (since the guard allows a jump at $x < 19$) but it also can remain in the *Off* mode since the invariant of $x >= 18$ is not violated.

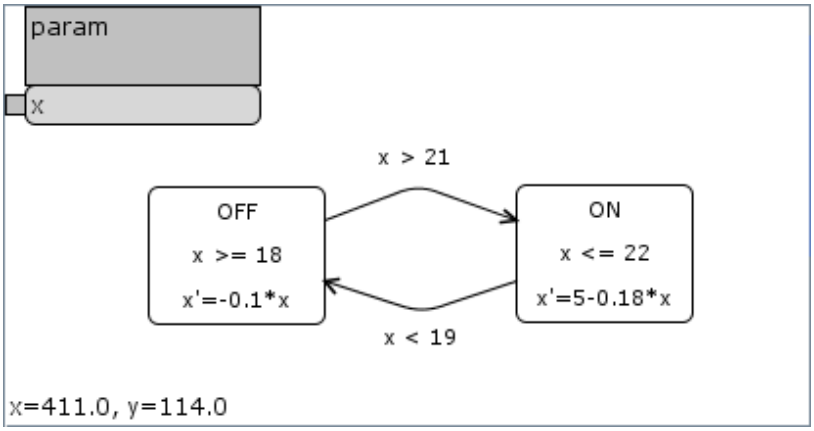

Figure 2.24: A simple heating thermostat hybrid automaton example [24].

2.4.3 Reachability

With hybrid automata as a formal representation of hybrid systems, analysis tools can be used to check system properties that are important to the safety critical domain. Reachability of hybrid automata is an analysis technique where a known initial set of system states is provided to the algorithm. The initial set of states is then propagated over a step (often known as a timestep) by the dynamics represented by Ordinary Differential Equations (ODEs) to construct a possible set of states in that step. This process is repeated allowing for representation of how the system can evolve over time. In systems with linear dynamics, this construction may take on polyhedral characteristics but this is not necessarily the case for systems with nonlinear dynamics. As these techniques propagate from a *known* initial set, each calculated reach set also includes an error term of that grows as the delta time from the known initial set grows. This error creates a conservative error bound

that may not be reached in actual system implementation. If interesting regions do not exist in the calculated region actual plus error set growth (known as a *flowpipe*), then reachability tools maintain those regions are unreachable (given correct dynamics representation). The combination of this set propagation can form a *flowpipe* that encapsulates the possible trajectories of the hybrid automaton. This is not the case for timed or rectangular dynamics that can be analyzed exactly.

It is important to point out that hybrid systems reachability is an active research area with a variety of sub-topics. For instance, it can be claimed that safety verification and reachability analysis are not decidable [23]. Decidability in the logic domain is based upon the capability to effectively answer decision problems (yes/no questions of input values to an algorithm). This was extended by [25] to state that a system will always be able to visit all its modes and reach a terminated condition. But the undecidable aspect only refers to the exact reachable set computation in all cases (for any system model). Some models (algorithms) may allow for the reachable set to be exactly computed (exact union computations) while others opt to do an overapproximation (convex hull operations). Regardless of which method is utilized, reachability can provide insight into hybrid system stability as long as the dynamics are correctly captured.

The challenge of undecidablity is not unique solely to hybrid systems but to other formal methods tools such as model checking of software. This is mitigated in those domains by bounding loops; i.e. if a system is deemed safe for a certain depth then it is acceptably safe. Similarly, this technique has been applied to hybrid systems in the continuous input domain [51] and the discrete domain [13] [23].

In the work by Tomlin [51], the concept of forward and backward reachable sets were introduced as a methodology to verify hybrid systems. These concepts use set information of unsafe states for which the system cannot exist should safety considerations be maintained. With this knowledge and reachability analysis, flowpipe propagation in the forward and backward direction can provide insight of system safety (fig. 2.25). Summary of the

definitions of forward and backward reachable sets are as follows: [51]

- **Forward reachable sets**: Is where the initial set of states is propagated forward in order to show all trajectories that originate from the initial set. This is traditionally how the flowpipe paths of hybrid automaton from an initial set are defined.
- **Backwards reachable sets**: This is an *a priori* defined set that would be considered "unsafe" for the system. Using this defined set as the initial set and propagating backwards can determine a union of sets that would lead to the "unsafe" initial set (fig. 2.26).

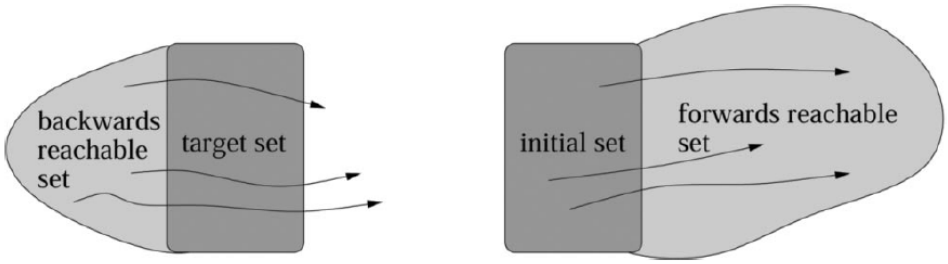

Figure 2.25: Representation of forward and backward reachable sets [51]

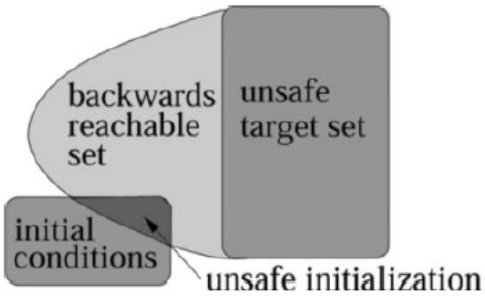

Figure 2.26: Using backward reachable sets to verify safety [51]

Other work on reachability included that of Althoff et.al [2] whose team examined computing reachable sets for uncertain time-varying linear systems then applying the technique to gain scheduled controllers. This methodology used set propagation which formed

zonotopes (a class of polyhedra) as the flowpipes generation (fig. 2.27). This was possible because their application to nonlinear and hybrid systems used overapproximated unions of convex hulls to encapsulated the reachable. The caveat here was the uncertain parameters applied to the system provided the flexibility to overapproximate the more complex sets. It is necessary to realize that overapproximation may present a more restrictive reachablity answer than other propagation techniques. Conversely, since nonlinear and hybrid system representations are encapsulated within the overapproximated convex hull, there can be a risk that the overapproximate not be sufficient.

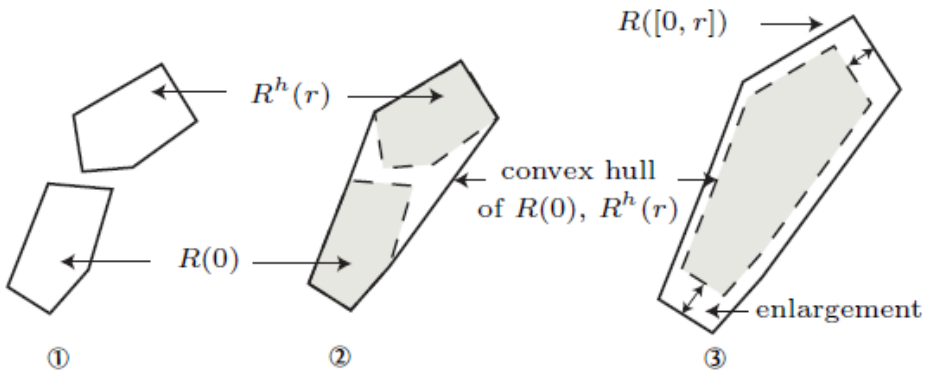

Figure 2.27: Development of a convex hull with overapproximation with two zonotopes [2]

2.4.4 Analysis Tools

In order for analysis to be conducted on hybrid automata, there has been a variety of tools that have been developed [22, 5, 14, 40, 20] to name a few. Many of these tools have made significant progress in the advancement of hybrid systems analysis allowing investigation into safety-critical systems. Here the author will briefly discuss two tools that will be significant for the examination in this thesis.

SpaceEx

SpaceEx is a tool platform developed out of Verimag in France as the next iteration of the work produced from PHAVer [30]. When initial development took place, there was a need for a platform for which new algorithms on hybrid automata could be examined for reachability (which consequently would lead to safety verification). Therefore, SpaceEx sought to fulfill this need for a platform while implementing polyhedra (often as zonotopes) based solutions for the generation of flowpipes for piecewise-linear systems [22, 30]. Analysis is limited to linear systems (nonlinear systems would require linearization) but advancements in the efficiency of propagation techniques using support functions and refinements of these algorithms proved to make SpaceEx well known in the hybrid systems community.

One benefit that SpaceEx brought to the overall hybrid community was the establishment of a XML format (SpaceEx XML Format) to define the hybrid automata accepted by its analysis engine. This, coupled with a configuration file format, defined the model and inputs necessary to execute system analysis. The development of the SpaceExMOE [31] used this format to enable practitioners in the community to generate visual representations of the hybrid automata through a Java-based program.

HyLAA

HyLAA (HYbrid Linear Automata Analyzer) is a tool that uses simulation-equivalent reachability to analyze hybrid systems over bounded time [5]. Previous work in hybrid systems used variations of the traditional reachability algorithms which propagated a set of a viable states forward within the evolution of a system. The algorithm generally alternates between calculation of reachable sets by discrete dynamics to that of reachable sets for continuous dynamics [32]. The traditional reachability algorithms, while powerful due to its ability to capture how a convex set evolves, can be extremely expensive, especially in systems with a high degree of state variables.

Simulation-equivalent reachability in HyLAA instead leverages the superposition prop-

erty of linear systems. By using superposition, evolution of reachable sets from an initial set maintains the same predicate throughout. In [6], this is illustrated with a 2-dimensional system which executes three simulations. These simulations define a center point and two external boundaries. Bak and Duggirala introduced the generalized star concept which uses the three simulations to determine the reachable set after a given time [6, 7]. This is demonstrated in their diagram in fig. 2.28.

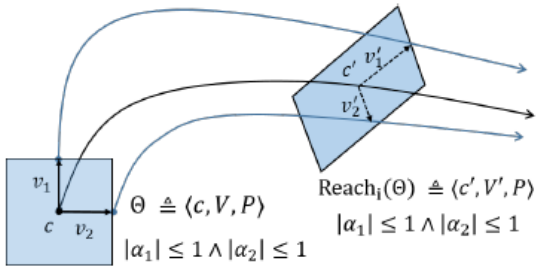

Figure 2.28: Example 2-dimensional reachable set evolution that shows the predicate of the initial and reachable sets are maintained [6].

2.4.5 Processing Tools

HYST

HYST (HYbrid Source Transformer) is a tool that was designed to aid in the overall analysis of hybrid systems by allowing the community to evaluate its collection tools with a variety of strengths. Unfortunately, even with previous attempts to create a uniform input syntax, the existing tools were still too varied to allow for single problems to be examined by more than the hybrid tool for which it was developed [4]. This limited the ability to objectively benchmark the tool landscape to most clearly idenitfy the strengths and weaknesses of different approaches. Therefore, HYST was developed to be able to (1) accept the different input formats of hybrid system tools, (2) translate and transform these input formats into a universal format, and (3) output equivalent system formats for a number of

hybrid system analysis engines [4]. The universal input format file type used by HYST is the SpaceEx XML format (as previously mentioned) as it became a generalized representation that could aid conversion from one format to another. HYST could translate and transform the format to work with many tools including dReach, Flow*, PySim, and HyCreate [4]. Another advantage is that HYST is architecturally designed to accept new translation capabilities as new innovations arise. It works with a scripted "printer" system that can be expanded and refined over time.

2.4.6 Hybrid Reachability Summary

The author chose to use SpaceEx and HyLAA as the reachability analysis tools for this thesis. SpaceEx provides a solid foundation of piecewise linear reachability analysis that many tools can trace their lineage. The ability of SpaceEx to be able to generate flowpipes of convex hulls with limited error growth allowed the author to make general observations of a number of control responses. The most significant to this work is a visualization of system stability. Should a system be unstable, the flowpipe show expands indefinitely without being bound to a steady state region. SpaceEx should be able to demonstrate this capability given the proper definition of the system.

Although SpaceEx does provide visualization of flowpipes that can aid in stability conclusions, it has been found to be sensitive to system complexity over extensive iterations. Many times, the fidelity of the engine is adjusted to compensate for scalability issues such as lessening the flowpipe error tolerance, lessing the maximum iterations, etc. HyLAA, on the other hand, is specifically designed to leverage the advances in simulation-based reachability to handle higher dimensionality over longer analysis periods. This work will leverage HyLAA to confirm that the basis of SpaceEx is maintained while propagating forward reachability flowpipes to a resolution that the author can feel confident claiming that system stability is maintained.

The significance of reachability analysis, particularly with HyLAA, allows examina-

tion of control systems that could not be as readily analyzed for demonstrable characteristics. An added benefit is that reachability can be used to better focus or reduce the numbers of required tests of a system. The flowpipes provide an evolution of all the reachable system states allowing tests to be directed into regions where the system is shown to enter rather than areas where the tools show are inaccessible. Observe fig. 2.29 reachability plot of a proportional controller. The gray areas are shown to be unreachable by the system, and therefore, testing resources can be moved to other parts of the system response.

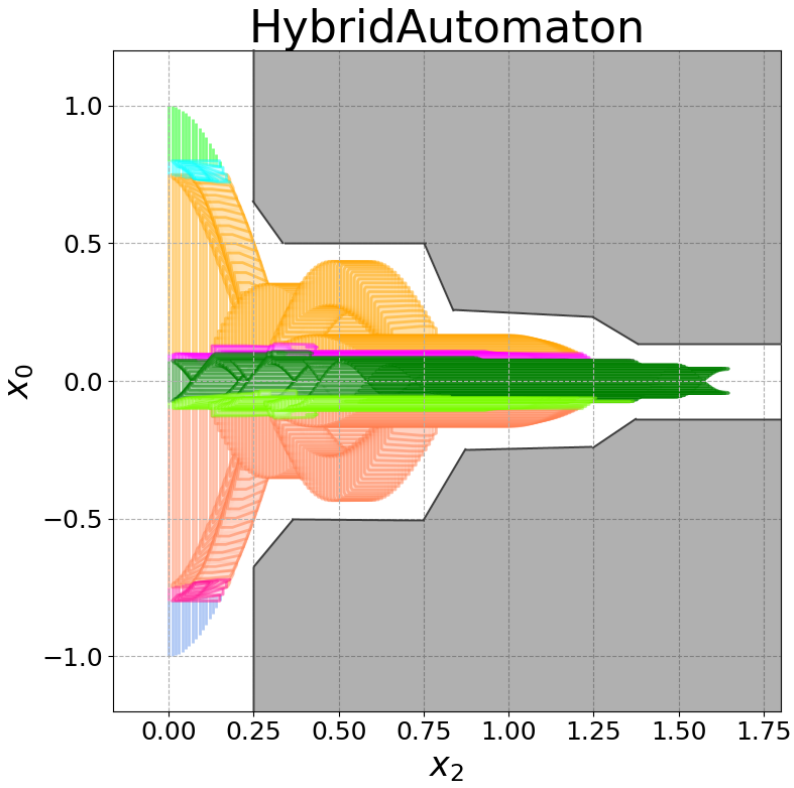

Figure 2.29: A proportional control reachability response with areas that are unreachable by the system in gray.

2.5 Summary

In this chapter, the author introduced concepts that are necessary for the stated thesis. First, an introduction of various gain scheduling techniques were presented based off of the researched work in the control community. The author then proceeded to present a foundation of understanding for fuzzy control systems. By introducing fuzzy control with a simple example, the author presented the reader a simplified version of techniques that would be later used in the examination. The chapter contained a discussion on hybrid automata (HA) which included background for the reader on finite state machines and a number of tools to analyze these constructs. Finally, the chapter closed with an introduction to reachability analysis of hybrid automata and the tools (used in this thesis) that are used to examine reachability.

Chapter 3

Gain Schedule / Fuzzy Controllers

As previously discussed, gain scheduling is a necessary control implementation for two primary reasons. The first is that modern systems under control often are best represented by nonlinear equations. Nonlinear equations are difficult to analyze for performance and stability characteristics. The second is that nonlinear system representations can be partitioned into linear approximations that are valid under certain dynamics. This is often the case in aircraft dynamics where speed (in the form of mach number) and altitude form the basis of gain scheduled tables. The advantage of representing a nonlinear system as a collection of linear systems is that each linear system can use traditional linear control techniques to determine the characteristics desired by design.

Unfortunately, the associated cost of this linear separation is that switching from one linear controller to another does not carry guarantees of performance and stability. Given the physical realization of such a system, this switching region also must not be completely abrupt to avoid instability. In a gain scheduled switching range, instantaneous switching could introduce mathematically infinite acceleration to the system. In practice, the heavy use of testing and simulation provides evidence that the conditions are suitably close to prevent instability. Techniques to ensure smooth transitions between controllers can include development of various mathematical splines that can also introduce nonlinearity. In

practice, it is often popular to linearly interpolate between the linear controllers testing at a tight range of input values. Should instability be detected, additional linear controllers are added to the controller and tuned to provide the desired response characteristics.

To mimic the nonlinear representation and to establish a process for alternate representations of the controllers with similar performance characteristics, this work will use a linear control system with multiple gains. The plant system to be tested is shown in eq. (3.1).

$$G(s) = \frac{75}{s^2 + 3.6s + 10} \tag{3.1}$$

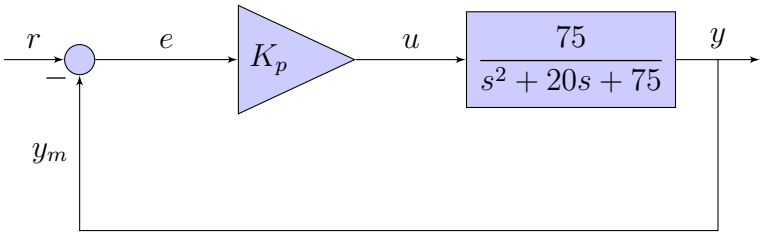

Figure 3.1: Block diagram of a traditional proportional feedback control.

In a traditional proportional control feedback system, the block diagram representation is shown in fig. 3.1 where the proportional gain (K_p) is static. To test the response to a proportional controller, the author chose three gains to examine the response to a step input. The following gains were chosen in order to provide a range of response attributes; $K_p = (1,\ 2,\ 12.5)$. Using these gains (with a scaling gain of 5) the responses were modeled as seen in fig. 3.2;

Upon examination of the response set in fig. 3.2, the following general characteristics can be observed. First that the higher gains produced lessened steady state error but caused extreme overshoot and oscillation in the response. The lower gains were slow and unable to achieve near desired steady-state error (observe $Kp_{low} = 1$). These attributes were expected as typical with proportional controllers. If the multiple gains are able to be utilized

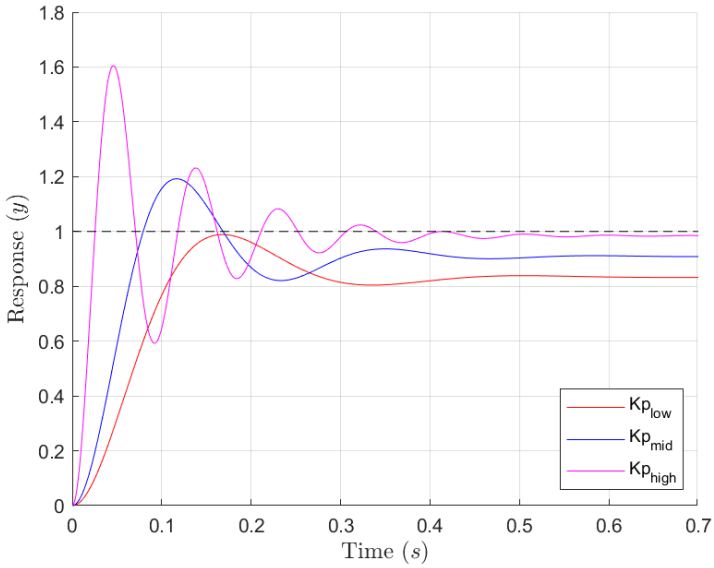

Figure 3.2: Various proportional gain controllers implemented with the example plant.

as a set, then the response could be improved. This will be done with a gain scheduled proportional controller.

3.1 Gain Scheduled Proportional Controller

Gain Scheduling is a control technique where complex plant models are sectioned off into multiple control zones and can be represented by a linear approximation. Such zones enable, with some variance of asymptotic stability, that the linear approximation will ensure a controlled desired response. In order to achieve such a technique, a methodology must be used to determine two aspects: (1) how to separate the nonlinear response curve and (2) the linear approximation of that section of the curve. Separation of a nonlinear curve or envelope is often done using slow dynamics. In aircraft, this is done with a correlation between altitude and speed which affect the wing lift dynamics (the control "plant"). As these dynamics change, so can the required gains to stabilize the system. The second aspect uses the discovered separation boundaries to approximate a linear response for each zone. This linear approximation is then subjected to the traditional control techniques (as

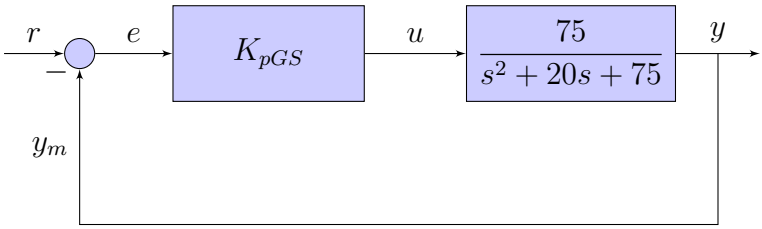

Figure 3.3: In a gain scheduled proportional system the K_{pGS} block gain changes based on the chosen dynamics.

previously discussed) to control that specific state of the overall system. It is important to note that control of these zones is purposely designed to have an acceptable asymptotic range for which the system will be resistant to noise and external influences. This is to be most accommodating to the approximation of the zone.

In gain scheduling, switching between linear approximations does not guarantee that stability is maintained. At the edge zones, the approximation contains the largest error ranges within the system. In aircraft design practice, empirical evidence of the zone boundaries are identified to categorize the resulting response. If the approximation is not close enough, risk of infinite acceleration vectors can occur. In theoretical gain scheduling, techniques such as spline development can mitigate such a concerns. But in traditional practice, an additional linear zone approximation is often preferred due to the breadth of tools for examining and tuning linear control systems.

In order to demonstrate gain scheduling in this work, eq. (3.1) will again be used but this time with proportional controllers with switching conditions built upon the loop error value. To demonstrate the gain scheduled system design, let us modify the previous block diagram with the gain being defined in a new block.

Recall in fig. 3.2, the static proportional gain values of $K_p = (1,\ 2,\ 12.5)$ with a static gain multiplier of five. These gain values were shown to provide insufficient characteristics throughout the entire range of the response. Let us observe the effect on the overall response should these gain values be applied to signal error (e) ranges. An error range in this gain scheduled controller is based on the feedback loop value of the system. When a normalized

step input is applied to a system that starts with no initial value, the distance from the target value is the greatest. Similarly, should the normalized response settle from value twice that of the target, it represents an equally large distance from the target. Therefore, the total error range of the system can reasoned as $e = (-1,\ 1)$. In traditional proportional control, the static gains are applied throughout this range. To demonstrate a gain scheduled proportional control (GS_PCtrl), gains are assigned to the continuous range of error in the system. Here the ranges for each gain value is shown in table 3.1.

Table 3.1: Proportional gains (K_p) for gain scheduled control dependent on the error value (e).

error range	K_p
$e < -0.50$	1
$-0.50 \leq e < -0.08$	2
$-0.08 \leq e \leq 0.08$	12.5
$0.08 < e \leq 0.5$	2
$e > 0.50$	1

The step response of the system with the gain scheduled controller is shown in fig. 3.4. Note the improvement of the overall response by using the error signal to switch between the different gain values. More complex systems may utilize different states to determining switching actions (such as in aircraft). Here, with respect to linear response, this feedback signal error provides an effective switching condition.

Unfortunately, the response simulated in this system with instantaneous switching is unrealistic in actual application. Abrupt changes can cause unsafe conditions to emerge. In a physical system, instantaneous changes of gains can provide mathematical instability by the creation of infinite acceleration. In this example, since this is a second-order linear system, stability should be maintained in switching, but other systems, this is not guaranteed. To ensure that this is represented, interpolation between the linear controllers should be included. Therefore, between each K_p value over an error range there is a linearly interpolated region where a smooth gain is achieved. This error (e) range uses the previously

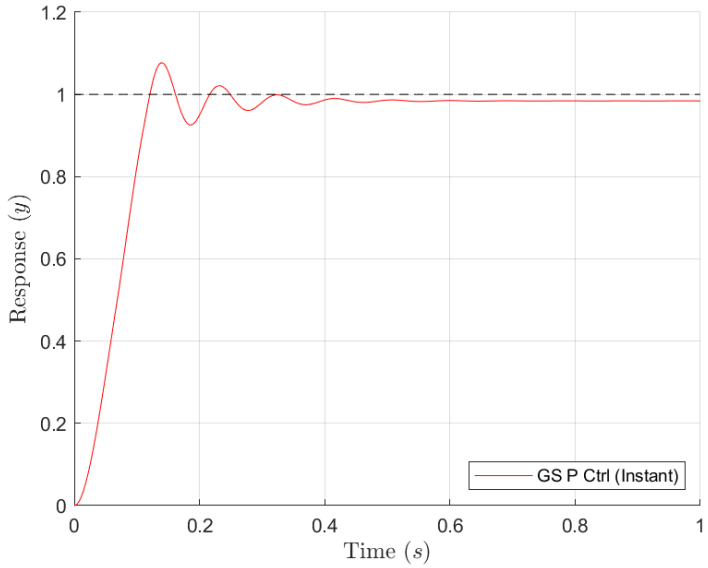

Figure 3.4: System step response with instantaneous switching between the gain values.

defined ranges (recall table 3.1) with a interpolation value of $e = 0.01$ as shown in fig. 3.5. To calculate the interpolated representations, a gain surface plot of the system can be generated. This is done by calculating the slope of the surface using a point-slope equation over the interpolated region connecting two static regions.

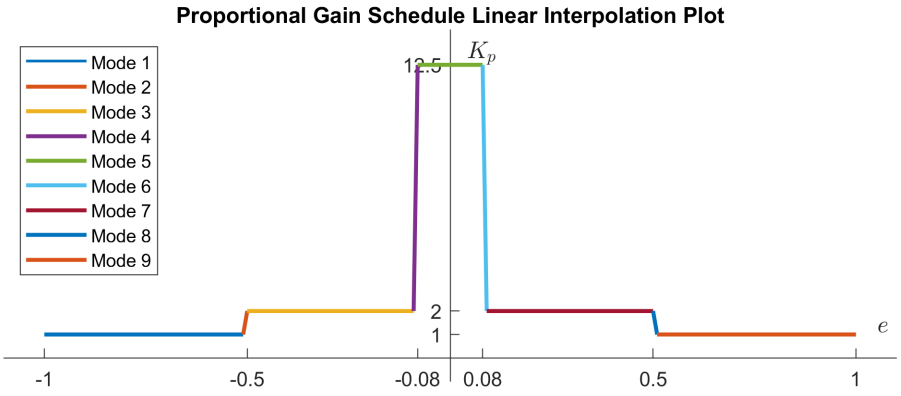

Figure 3.5: Gain scheduled proportional control with interpolated ranges.

To demonstrate the interpolation calculation consider the region where the error range is $[0.5,\ 0.51]$ and $K_p = [2,\ 1]$. First, the slope (m) is determined using the boundary points

(see eq. (3.2)).

$$m = \frac{K_{p2} - K_{p1}}{e_2 - e_1} = \frac{-1}{0.01} = -100 \tag{3.2}$$

Next, one of the boundary points is utilized to calculate the equation of the line as shown in eq. (3.3).

$$\begin{aligned} K_p - K_{p1} &= m(e - e_1) = m(e) - m(e_1) + K_{p1} = m(e) + C \\ C &= -m(e_1) + K_{p1} = -100(0.5) + 2 = 52 \\ K_p &= -100e + 52 \end{aligned} \tag{3.3}$$

Completing these calculations for all the entire possible error range of the system produces the control surface as shown in table 3.2. Now that the system is defined with interpolation between the gain scheduled linear controllers, the response can be compared to that of the previous instantaneous gain system (fig. 3.6). Note that the response between the interpolated and instantaneous switching is similar.

Table 3.2: Proportional gains with interpolated regions dependent on the error value (e).

error range	K_p
$e < -0.51$	u = e
$-0.51 \leq e < -0.5$	100e + 52
$-0.5 \leq e < -0.09$	2
$-0.09 \leq e < -0.08$	1050e + 96.5
$-0.08 \leq e \leq 0.08$	12.5e
$0.08 < e \leq 0.09$	-1050e + 96.5
$0.09 < e \leq 0.5$	2
$0.5 < e \leq 0.51$	-100e + 52
$e > 0.51$	1

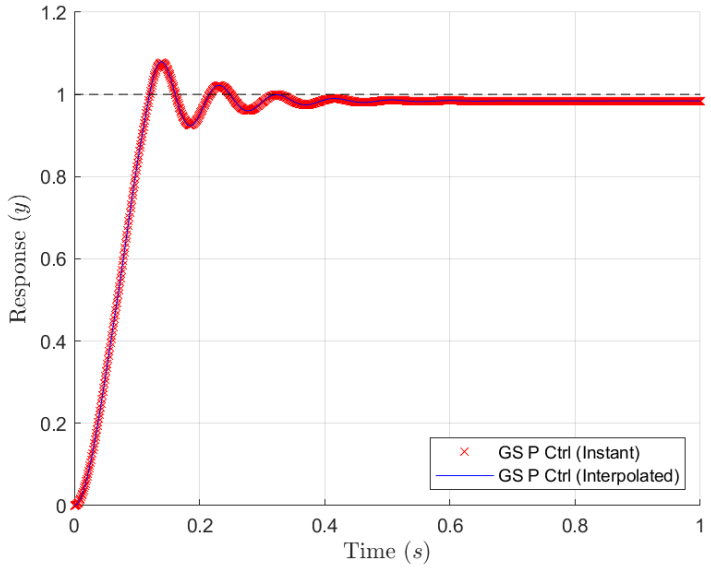

Figure 3.6: Comparison between instantaneous switching of gains vs a linear interpolated switching region over $e = 0.05$.

3.2 Fuzzy Logic Controller

The process to utilize a gain scheduled controller was previously shown to improve a response to input by utilizing only proportional manipulation. This section will show the process to build an equivalent fuzzy controller. The basic fuzzy control system will be represented with a block diagram as seen in fig. 3.7. The inclusion of a Proportional Fuzzy Logic Controller (PFLC) allows for the input signal (u) to the plant to be adjusted with respect to the incoming error (e) signal.

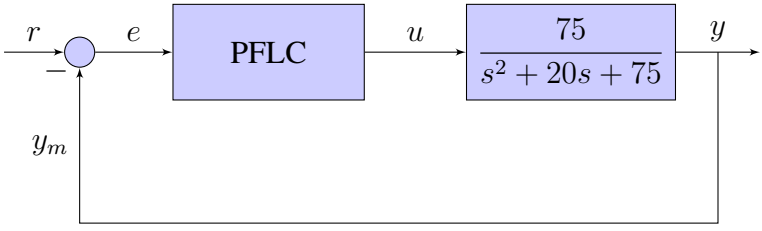

Figure 3.7: A representative system using a Proportional Fuzzy Logic Controller (PFLC) where the block gain changes based on the fuzzy rule set.

In order to make the conversion from a gain scheduled controller to a fuzzy based

system, the existing gains associated with the established error ranges are used to generate the input and output fuzzy sets. In order to capture the multiple gains in the system through fuzzy representation, the use of triangular fuzzification was used. Triangular fuzzy sets enforce that there exists only a singular value of the membership set where full membership occurs. Any set membership value that is not a singular point represents a contribution of the adjoining input ranges to that of the membership set.

Building the Input Fuzzy Membership Set

Using triangular fuzzy set representation, the first step is to build the input fuzzy set. To build the input fuzzy set, the gain scheduled error (e) ranges are used. Each area of static gain is represented by a singular membership set whose setpoint is the error value at a gain boundary. To illustrate this, observe the error range of $[0.51,\ 1]$ in the gain scheduled controller. The triangular input membership set is defined at $e = 0.51$ as the full membership value. Due to the rules of the input membership paradigm (triangular sets), each side of the triangle proceeds toward the adjoining setpoint via linear progression to zero membership. Therefore, the two sides of the error 0.51 membership setpoint connects to the error 0.5 setpoint (left side) and the error 1.0 setpoint (right side) both at zero membership at those points. Triangular membership continues infinitely when no other membership functions are present as seen at the error setpoint at 1.0. The left side connects to the zero membership at error of 0.51 while the right side has no additional setpoints and therefore continues at full membership. This can be seen in the input fuzzy membership set in table 3.3.

With the process defined to establish the input fuzzy membership set, the full input set can be illustrated. To coincide with the gain schedule error ranges the membership setpoints will exist at the following values as seen in table 3.3. Furthermore, the setpoints will be labeled for ease of reference using a semi-descriptive encoding: (P = positive, N = negative, S = Small, M = Medium, B = Big, Z = Zero).

Table 3.3: Setpoints of the input fuzzy set based on error (e).

	NB	NMB	NM	NMS	NS	Z	PS	PMS	PM	PMB	PB
e	-1	-.51	-.5	-.09	-.08	0	0.08	0.09	0.5	0.51	1

Building the Output Fuzzy Membership Set

The next step to creating the fuzzy representation of the gain scheduled controller is to generate the output fuzzy set. This set will associate the output signal (u) to the input fuzzy set (error) membership setpoints. There are two methods to approach construction of the output fuzzy set: (1) through piecewise linear calculations or (2) through point-slope calculations. This work will focus on the piecewise linear representation but also include in Appendix A the point-slope calculations. In a Single-Input Single-Output (SISO) fuzzy system, these techniques will produce equivalent results.

The first step is to normalize the gain values of the system. Although this is not required, it often aids the practitioner by removing the large magnitude representation. It allows for a constant gain valued to be added to the system design. Adding a static gain multiplier is often much more economical than trying to provide the full gain value. By normalizing on that gain, the decision block sees a maximum gain of 1 ($K_p = 1$) with the static gain added later. Therefore, in the calculations for the output fuzzy membership set, the provided gains ($K_{pGS} = [1,\ 2,\ 12.5]$) will normalize to $K_{pnGS} = [0.08,\ 0.16,\ 1]$.

To construct the output fuzzy set, first calculate the output signal value (u) at each of the gain schedule boundary conditions. A gain schedule boundary point here is any point where the linear representation of the gains slope changes. An example of this is at error point where ($e = -0.51$). Here the static gain value is going to encounter an interpolation region. Progressing from the negative range of the system, the point where error is -1 has a gain value of $K_p = 0.08$. The u term is determined as the product of the error and the gain. This is expected as by definition as $u = K_p e$. Hence, the first output membership function is $u = K_p e = 0.08(-1) = -0.08$.

Let us examine the next setpoint where error is -0.51. The gain value at that point

is still $K_p = 0.08$ but now the error term affects the output signal term. Here, by again resolving $u = K_p e$ the resulting output fuzzy set value is $u = K_p e = 0.08(-0.51) = -0.0408$.

Continuing through each of the boundary points, the following output signal values (u) is calculated as proceeding from the negative to the positive error range in the gain scheduled set as seen in table 3.4.

Table 3.4: Normalized output signal values (u) for the gain scheduled boundary conditions.

u	-0.08	-0.0408	-0.08	-0.0144	-0.08	0	0.08	0.0144	0.08	0.0408	0.08

The values shown in table 3.4 are unordered because their magnitudes are not aligned numerically. To make the output fuzzy membership set properly, these have to be reordered by value and then a similar labeling, as was present in the input fuzzy set, can be applied as seen in table 3.5. Because of the construction of this example, the output fuzzy set contains multiple instances of the same value. This is easily handled within a fuzzy control system as we can eliminate the extra cases in the output set.

Table 3.5: Normalized output signal values (u) for the gain scheduled boundary conditions.

	NB	NM	NS	Z	PS	PM	PB
u	-0.08	-0.0408	-0.0144	0	0.0144	0.0408	0.08

Now that the input and output fuzzy sets are defined, the correlation between the sets can establish the rules of the PFLC. Return to the unordered table (table 3.4). As previously mentioned these values are obtained by the setpoint boundary conditions and therefore provide a mapping of the rule set. Defining the rules for the system provides the following statements in table 3.6. Recall that the linguistic rules are phrased as: "If [the] *input value* is [in this] *input membership set* then [the] *output value* is [in this] *output membership set*."

Table 3.6: Linguistic fuzzy rules using the input and output fuzzy membership sets.

If e is NB then u is NB	or
If e is NMB then u is NM	or
If e is NM then u is NB	or
If e is NMS then u is NS	or
If e is NS then u is NB	or
If e is Z then u is Z	or
If e is PS then u is PB	or
If e is PMS then u is PS	or
If e is PM then u is PMB	or
If e is PMB then u is PMS	or
If e is PB then u is PM	

With the correlation defined, matrix operations can characterize the gain surface plot. We are able examine an number of error (e) points in the input fuzzy membership set, and the total output signal value (u) can be calculated using the simplified weighted technique.

The process to calculate the gain surface plot [15] is generalized in the following eq. (3.4) where j is the index region, $u(j)$ is the output signal of the index region, $K_{pe}(j)$ is the equivalent gain of the index region, and $cons(j)$ is the constant offset value.

$$u(j) = K_{pe}(j) \times x + cons(j) \tag{3.4}$$

The general form of the terms where $U(j)$ denotes the index of the output fuzzy membership function as seen in eqs. (3.5) and (3.6).

$$K_{pe}(j) = \frac{U(j+1) - U(j)}{x(j+1) - x(j)} \tag{3.5}$$

$$cons(j) = \frac{x(j+1) * U(j) - x(j) * U(j+1)}{x(j+1) - x(j)} \tag{3.6}$$

To solve these equations for the entire system (thus providing a complete gain surface) the following use of the weighted average system was refined in [17, 16] where eqs. (3.5)

and (3.6) are refined to eqs. (3.7) and (3.8).

$$K_{pe}(j) = \frac{1}{x(j+1) - x(j)} \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} u(j) \\ u(j+1) \end{bmatrix} \tag{3.7}$$

$$cons(j) = \frac{1}{x(j+1) - x(j)} \begin{bmatrix} x(j+1) & -x(j) \end{bmatrix} \begin{bmatrix} u(m) \\ u(m+1) \end{bmatrix} \tag{3.8}$$

The work to develop the mode equations and a Matlab script that calculated the piecewise linear can be found in Apendix A. The resulting values can be seen for the system under consideration in table 3.7.

Table 3.7: System modes of the PCtrl system using piecewise linear calculations.

Mode	error range	u
1	$e < -0.51$	u = e
2	$-0.15 \leq e < -0.5$	u = -49e - 25.5
3	$-0.5 \leq e < -0.09$	u = 2e
4	$-0.09 \leq e < -0.08$	u = -82e - 7.56
5	$-0.08 \leq e \leq 0.08$	u = 12.5e
6	$0.08 < e \leq 0.09$	u = -82e + 7.56
7	$0.09 < e \leq 0.5$	u = 2e
8	$0.5 < e \leq 0.51$	u = -49e + 25.5
9	$e > 0.51$	u = e

Using the defined equations for u, a proportional gain surface graph can be constructed. The column labeled *Mode* in table 3.7 provides the reader a linguistic representation of the modes. Graphing the PFLC surface plot based on the table produces fig. 3.8.

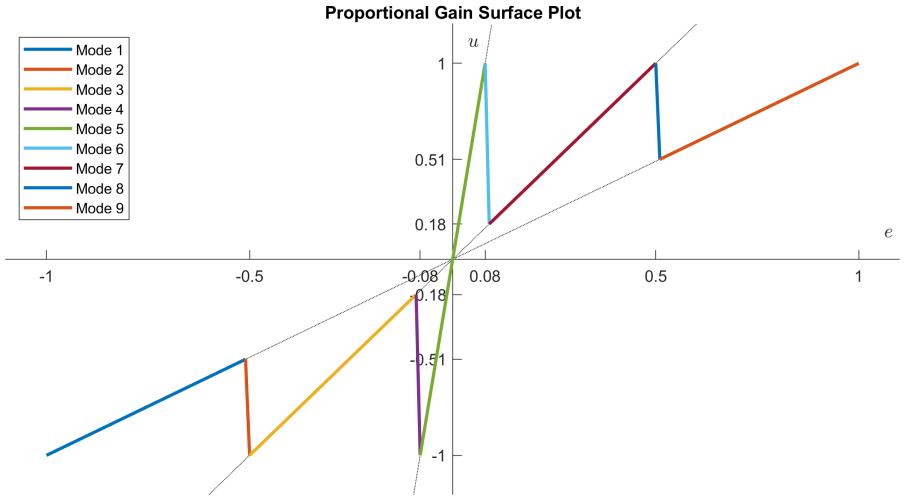

Figure 3.8: Gain surface of PFLC.

Matlab Fuzzy Representation

The first step to implement such this system is to build the Simulink model as shown in fig. 3.9. This example,uses the same represented plant that was previously described in (eq. (3.1)) and uses the same three proportional gains ($K_p = (1,\ 2,\ 12.5)$). In fig. 3.9, there are two separate systems that are excited by a step input. The top system, labeled *GS_PCtrl*, is the representation of the gain scheduled controller. Since it has been established that the gain schedule controller is using linear interpolation, in Simulink the author chose to represent the characterization of the controller using a Lookup Table (LUT) block as seen in fig. 3.9. In Simulink, the LUT block will automatically perform interpolation between any changes in the values over a range.

The bottom system, labeled *PFLC* in fig. 3.9, is the implemented fuzzy controller. The internal structure of the PFLC is a function block discussed in Appendix A that produces the PFLC response. There are two inputs that proceed into the PFLC function block, the first is the error (e) term and the second is that of the derivative of the error term ($\dot{e}$). In the case of this system, since it is proportional control only, the error input line is used. The

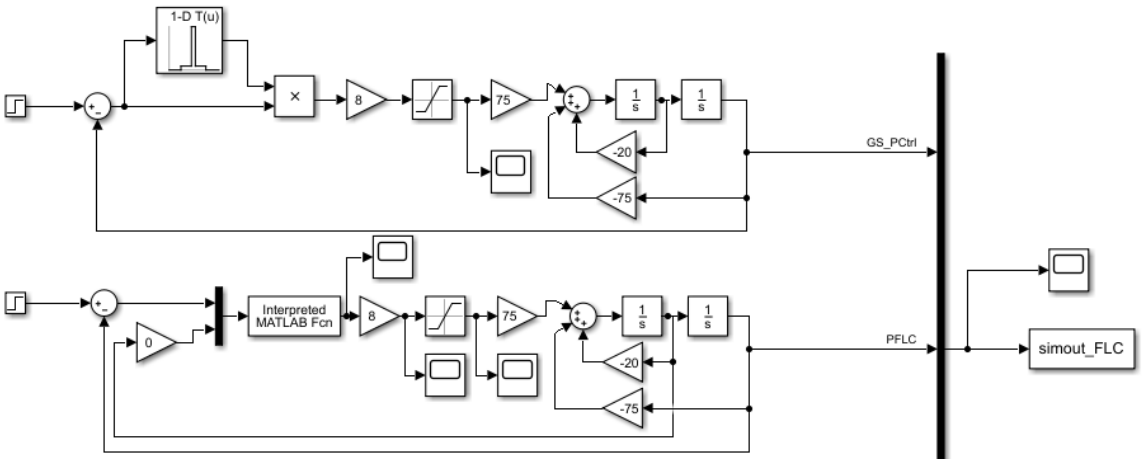

Figure 3.9: Simulink that plots the gain scheduled proportional control and the PFLC responses.

function is designed to leverage additional inputs of derivative (K_d) if desired, but it was not chosen for this example. Internal to the function, the vectors "centers" and "rule_vector" define input and output sets that define the fuzzy controller.

As seen in section 2.3 in chapter 2, the process to design a fuzzy controller requires the use of input and output fuzzy sets to define the system. In order to find the centers and the rule vector, the calculation was done with the weighted average technique with the results shown in table 3.8.

Table 3.8: PFLC definition of a gain scheduled three-gain controller.

centers	-1	-.51	-.5	-.09	-.08	0	.08	.09	.5	.51	1
rule vector	-1	-0.51	-1	-0.18	-1	0	1	0.18	1	.51	1

By showing equivalence in responses between the gain scheduled and fuzzy controllers, the author decided to move ahead with the transition to hybrid automaton representation for reachability analysis of the PFLC.

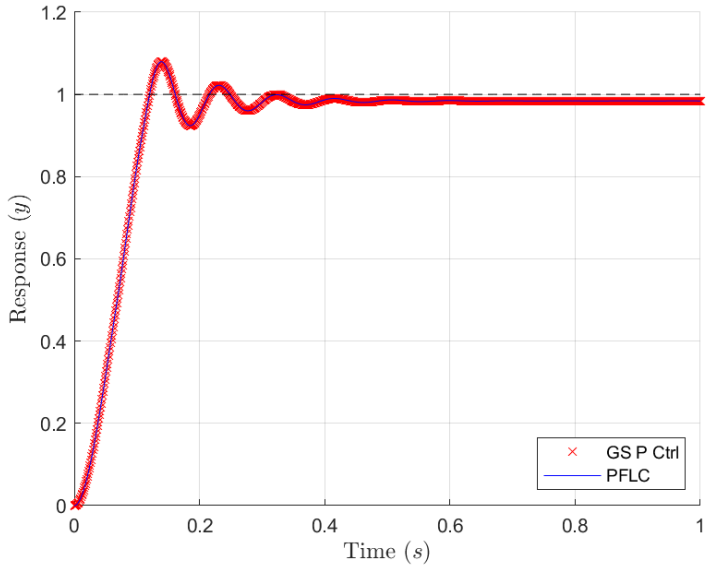

Figure 3.10: This is a comparison of the gain schedule and fuzzy proportional controllers.

3.3 Response Improvement - Selective Damping

As seen in fig. 3.10, there is desire to improve the overshoot and settling time (remove oscillations) in the system. This is due to the large gain in the final phase of the designed controller where the system is near the target settling value. To eliminate the steady-state error, a high gain is introduced. The most direct way in traditional control is to introduce an ability to further dampen the system and the same holds true with in the fuzzy control representation.

To accomplish the additional damping of the system, a selective fuzzy PDFLC (PselD) will be constructed to introduce limited system damping when the response nears the steady-state. The primary benefit of this technique is that it ensures that the rise time of the system is not increased.

In order to construct the PDFLC controller, the first step is to develop a new rule base that is two-dimensional. Recall the rule base defined in table 3.8 which defines a proportional fuzzy logic controller (PFLC). Adding the second dimension to the rule set represents the change of error $\dot{e}$ term. In this example, a three membership fuzzy second

input (change of error) is selected with center values at $[-1,\ 0,\ 1]$. This is a deliberate design decision to select a constant derivative input during the modes when the output is near the steady-state region. A gain of 0.15 was selected for the change of error input ($\dot{e}$), since the amount of derivative use will be limited to the near steady-state which will be described later.

In order to develop the 2-input/1-output rule base, the switching gain conditions need to be maintained. Therefore, the constant gain regions in the controller are characterized in the piecewise linear representation. For instance, observe the PFLC rule base (table 3.8), the output during the steady-state region $u = 12.5e$ is selected.

To maintain the PFLC capability but to add derivative refinement to modes near the steady-state of the response, the methodolgy in [17, 16] is desired. The first step in this process is the decision to only add derivative control to selected number of modes when building a hybrid automaton. The output of the fuzzy control system for a two-input system is defined in eq. (3.9).

$$u = K_{pn}e + K_{dn}\dot{e} + K_{nn}e\dot{e} + C \tag{3.9}$$

To add the derivative input during the steady-state region, the rule-base given in table 3.9 is selected.

Table 3.9: Rule-base for the PselD controller.

$\dot{e}$ \ e	-1	-.51	-.5	-.09	-.08	0	.08	.09	.5	.51	1
-1	-1	-0.51	-1	-0.18	-2	-1	0	0.18	1	.51	1
0	-1	-0.51	-1	-0.18	-1	0	1	0.18	1	.51	1
1	-1	-0.51	-1	-0.18	0	1	2	0.18	1	.51	1

In table 3.9, the steady-state region of the system is designed to be symmetrical about the positive and negative response and therefore will not need a zero input membership set. The zero case here is maintained only for these explanation but will not be directly represented in the hybrid automaton. To show that a constant derivative gain is added to

the PselD controller the following modes are computed. In these calculations, the input x_1 defines the error input fuzzy set while the input x_2 defines the $\dot{e}$ input fuzzy set. For the steady-state mode error index values are $x_1(j) = -0.08$ and $x_1(j+1) = 0.08$ while the $\dot{e}$ index values are $x_2(k) = -1$ and $x_2(k+1) = 0$. To calculate the common denominator of these equations refer to eq. (3.10).

$$Den = [x_2(k+1) - x_2(k)] * [x_1(j+1) - x_1(j)] = 0.16 \tag{3.10}$$

The values for K_{pn}, K_{dn}, K_{nn}, and C are determined as transformations.

$$K_{pn} = \frac{1}{Den} \begin{bmatrix} x_2(k+1) & -x_2(k) \end{bmatrix} \begin{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} u(L_1) \\ u(L_3) \end{bmatrix} \\ \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} u(L_2) \\ u(L_4) \end{bmatrix} \end{bmatrix} \tag{3.11}$$

Solving for the mode (let us label it as *Mode 5*), one will find the calculation to result in $K_{p5} = 12.5$ as shown in eq. (3.12).

$$\begin{aligned} K_{p5} &= \frac{1}{Den} \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} -2 \\ 0 \end{bmatrix} \\ \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} \end{bmatrix} \\ &= \frac{1}{0.16} \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \frac{2}{0.16} = 12.5 \end{aligned} \tag{3.12}$$

The K_{dn} term is given by eq. (3.13).

$$K_{dn} = \frac{1}{Den}\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} \begin{bmatrix} x_1(k+1) & -x_1(k) \end{bmatrix} & \begin{bmatrix} u(L_1) \\ u(L_3) \end{bmatrix} \\ \begin{bmatrix} x_1(k+1) & -x_1(k) \end{bmatrix} & \begin{bmatrix} u(L_2) \\ u(L_4) \end{bmatrix} \end{bmatrix} \tag{3.13}$$

Solving for K_{dn} with the appropriate substitutions for *Mode 5* produces the following result as see in eq. (3.14). It is important to note that the resulting $K_{d5} = 1$ is leveraged in the Simulink design by allowing the $\dot{e}$ input to be affected by a scaled multiplier (see fig. 3.11). This allowed tuning outside of maintaining the proportional input membership set.

$$\begin{aligned} K_{d5} &= \frac{1}{Den}\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} \begin{bmatrix} 0.08 & -(-0.08) \end{bmatrix} & \begin{bmatrix} -2 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0.08 & -(-0.08) \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} \end{bmatrix} \\ &= \frac{1}{0.16} * (0.16 + 0) = 1 \end{aligned} \tag{3.14}$$

Referring back to eq. (3.9), the next term to be defined is the K_{nn} term. This term represents a nonlinear element that may be introduced into the system. To solve the nonlinear

aspect in the two-input/one-output system matrix, calculations are used in eq. (3.15).

$$K_{nn} = \frac{1}{Den}\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} u(L_1) \\ u(L_3) \end{bmatrix} \\ \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} u(L_2) \\ u(L_4) \end{bmatrix} \end{bmatrix} \tag{3.15}$$

As expected in this system, the resulting nonlinear term effectively falls out of the system as shown in eq. (3.16).

$$\begin{aligned} K_{n5} &= \frac{1}{Den}\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} -2 \\ 0 \end{bmatrix} \\ \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} -1 \\ 1 \end{bmatrix} \end{bmatrix} \\ &= \frac{1}{0.16} * (-2+2) = 0 \end{aligned} \tag{3.16}$$

The final term in eq. (3.9) is that of the constant term (C)in the equation. Solved also by matrix calculation it is shown in eq. (3.17).

$$C_n = \frac{1}{Den}\begin{bmatrix} x_2(k+1) & -x_2(k) \end{bmatrix}\begin{bmatrix} \begin{bmatrix} x_1(k+1) & -x_1(k) \end{bmatrix} & \begin{bmatrix} u(L_1) \\ u(L_3) \end{bmatrix} \\ \begin{bmatrix} x_1(k+1) & -x_1(k) \end{bmatrix} & \begin{bmatrix} u(L_2) \\ u(L_4) \end{bmatrix} \end{bmatrix} \tag{3.17}$$

Substituting the values for *Mode 5* the value for C_5 is found in eq. (3.18).

$$C_n = \frac{1}{0.16}\begin{bmatrix}0 & -(-1)\end{bmatrix}\begin{bmatrix}\begin{bmatrix}0.08 & -(-0.08)\end{bmatrix}\begin{bmatrix}-2\\0\end{bmatrix}\\ \begin{bmatrix}0.08 & -(-0.08)\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix}\end{bmatrix}$$
$$= \frac{1}{0.16} * (0) = 0 \tag{3.18}$$

The aforementioned calculations allow for the definition of u_5 defined in eq. (3.9) to be completed for *Mode 5*. Furthermore, using the same calculation steps, there are eighteen modes full u values can be determined. Note that nine modes are represented in a Hybrid Automaton which is possible due to the symmetry built into this example system. Should symmetry not exist, then this example system would have to be represented as eighteen modes and necessitate switching on both the error (e) and change of error ($\dot{e}$) terms. This symmetry also allowed for the derivative term (K_d) to be selectively applied. Therefore the fully designed mode table is defined in table 3.10. With the mode table fully developed and the rule vector defined in table 3.9, a Simulink representation of the PDFLC is generated.

Table 3.10: Mode table by piecewise linear matrix calculations for the PDFLC.

Mode	C_n	K_{dn}	K_{pn}	Nonlinear
1	0	0	1.0	0
2	-25.5	0	-49.0	0
3	0	1.2195	2.0	2.439
4	-7.56	1.0	-82.0	0
5	0	1.0	12.5	0
6	7.56	1.0	-82.0	0
7	0	1.2195	2.0	-2.439
8	25.5	0	-49.0	0
9	0	0	1.0	0

3.3.1 Simulink Representation

In order to represent a selective addition of the derivative control aspect to the system, the feedback loop will include both the e and $\dot{e}$ inputs to the PDFLC function. Within the PDFLC function block, the centers of each input membership set is defined along with a rule vector that associates those memberships with the output membership set. The combination of this information produces the output signal as determined in table 3.10.

Observe that the difference in this Simulink diagram between the PFLC and the PDFLC is that the $\dot{e}$ feedback loop is multiplied by a 0 gain. This eliminates the derivative term in the PFLC. Since the designed K_d in the selective terms (Modes 4,5,6) are all $K_d = 1$ and the K_d term is not used in Modes 3 and 7, then the elimination of the $\dot{e}$ feedback produces the same response as the PFLC.

Note that in the PDFLC (PselDFLC) representation, the $\dot{e}$ gain multipier is 0.15 which is a designed value, tuned to improve the results. It is able to be further adjusted should it be deemed necessary.

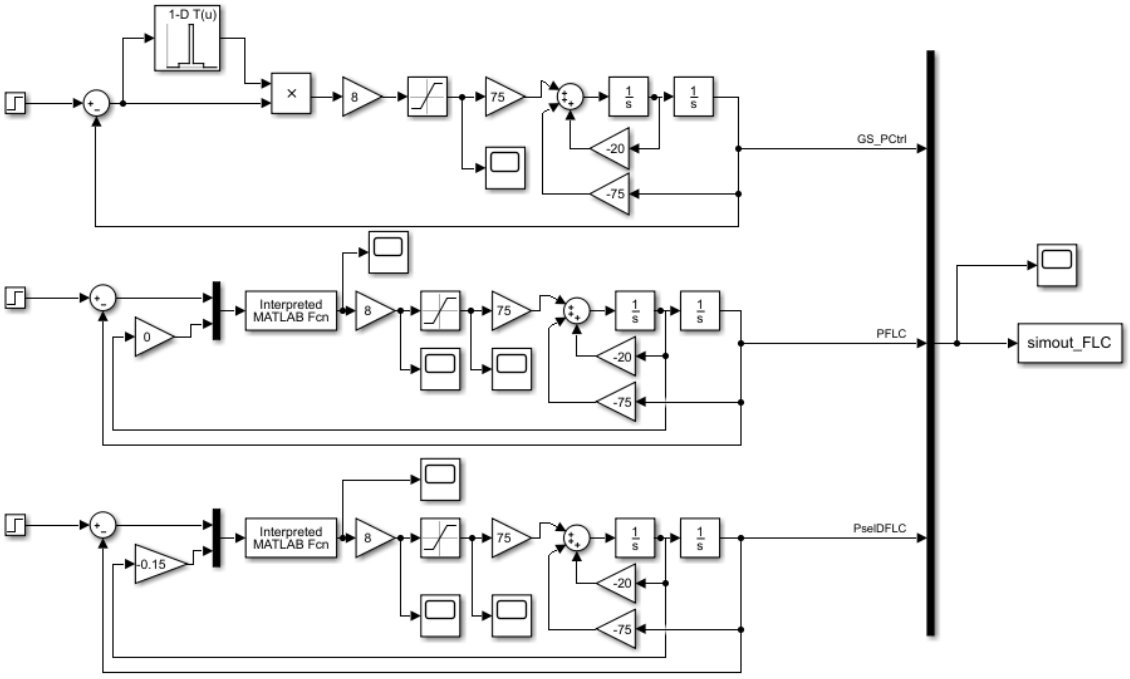

Figure 3.11: Representation of the GS PCtrl, the PFLC, and the PDFLC.

With a representation in Simulink now defined, response examination can be done on the system as shown in fig. 3.12. First, notice that the plots for the gain scheduled proportional controller and the PFLC are not both visible in the response. This is due to the equivalence seen in the behavior that was shown earlier in this chapter. Therefore, the responses are layered on top of each other, the last drawn only being visible. The next item to notice in the response is that of the PselDFLC line (PDFLC with selective derivative control). The derivative term allowed for improvement of the response by nearly removing the oscillations of the system and reducing the experienced overshoot. Furthermore, it maintains the same steady-state error that is seen in the gain scheduled and PFLC. These

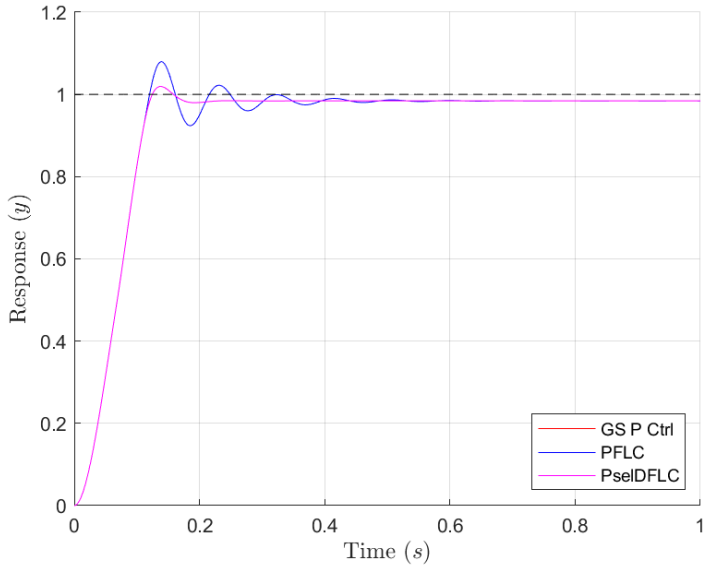

Figure 3.12: Response comparison of the GS PCtrl, the PFLC, and the PDFLC.

improvements show the value of selectively adding the derivative term into this PDFLC.

3.4 Summary

In this chapter the author introduced a gain scheduled (GS) controller to control a second-order plant system. This example demonstrated that a system response can be better tuned with a combination of three separate proportional gains. How to use each of the gains was determined by the feedback error to a step function. Upon demonstration of the improved response with the gain scheduled controller, the author then implemented a process to convert the gain scheduled controller into a proportional fuzzy logic control (PFLC). This process demonstrated that the responses can be made equivalent and therefore the advantages of fuzzy control can be made available to many GS applications.

The author then expanded upon the example system by adding selective damping to both the GS representations of the controller and that of the fuzzy controller. This fuzzy controller became a version of a proportional-plus-derivative fuzzy logic control that the author termed as PselD or PselDFLC. These terms were used because the derivative term

was selectively applied to remove effect on the rise-time response of the system. This chapter concluded with evidence that the GS control with selective derivative and the PselD control system are also equivalent.

Chapter 4

Hybrid System

With a fuzzy controller added to the system, a conversion to hybrid automaton is now directly possible. Previously established by Clark and Rattan [17], the conversion of fuzzy systems to hybrid automata provides a method to leverage the growing array of hybrid analysis tools in the community. Specific to this work, reachability of controlled systems can be examined.

4.1 ODE Dynamics Representation

Before conducting reachability analysis on a fuzzy control system, a direct representation of the gain scheduled controller is implemented. This representation, converted into an ordinary differential equation (ODE) would utilize a lookup table (LUT) to represent the linear changes in gains. In order to ensure that the ODEs are properly calculated and to ensure that the tooling that is later used functions properly, the system is tested using Matlab's ODE45 representation.

Recall that system transfer function is defined as $G(s) = 75/(s^2 + 20s + 75)$. In order to convert the system to an ODE, the standard proportional controller is defined in fig. 3.1. We can isolate the plant representation ($G(s)$) as an output/input relationship in

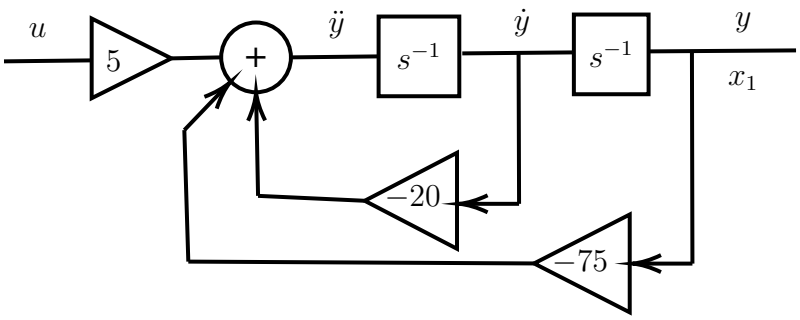

Figure 4.1: The plant represented by component blocks and feedback.

the frequency domain (Y/U) shown in eq. (4.1). When converted from a transfer function to that of an ODE, the equation is set in the terms of $\ddot{y}$ eq. (4.2). The transfer function of the plant is given by eq. (4.1).

$$\frac{Y}{U} = \frac{75}{s^2 + 20s + 75} \tag{4.1}$$

$$Y(s^2) + 20 * Y(s) + 75 * Y(0) = 75 * U(s)$$

$$\ddot{y} + 20\dot{y} + 75y = 75u$$

$$\ddot{y} = -20\dot{y} - 75y + 75u \tag{4.2}$$

Using the representation of eq. (4.2), the plant can be realized in an alternate form that will provide more direct access to the position and previous velocity of the system as seen in fig. 4.1. Finally, the terms are converted to an ODE for the reachability tools where x_1 is the position, x_2 is the previous velocity of the system, and u is the input of the system as shown in eq. (4.3).

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -20x_2 - 75x_1 + 75u \end{aligned} \tag{4.3}$$

Simulink block diagrams that were used previously model the response characteristics. In the reachability analysis tools instead of a graphical representation, the direct math-

ematical ODE representation is used. Therefore, using eq. (4.3), the system is functionally expressed. It is important to note that the error (e) term in the system was determined with the following characteristics: the target value is a normalized step function and the system will not exceed twice the target. Specifically, the operating range of $x_1 = [0, 2]$ with an initial starting point of the system would be zero. Using the ODE45 technique, the calculation of the error value is $e = 1 - x1$. This is the case as the error value is the distance from the normalized target of the step function represented by the x_1 position variable. Plotting the system showed similarities to that of the proportional gain scheduled interpolation in Simulink and PFLC responses. This was confirmed when all three responses (Fuzzy P Ctrl, PFLC, GS ODE45)were displayed on a single plot fig. 4.2.

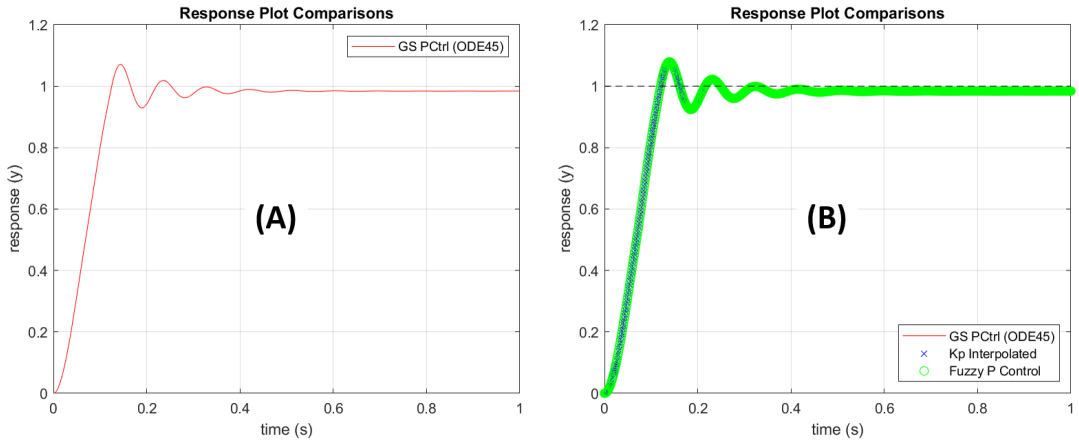

Figure 4.2: (A) Plot of the ODE representation of the PCtrl example system using ODE45. (B) Plot of the ODE45, the GS interpolated control, and the PFLC layered data sets.

4.2 PFLC to Hybrid Automaton

In order to construct the hybrid automaton from a fuzzy system, let us refer back to the ODE representation to provide the continuous dynamics of the system as seen in eq. (4.3). In the system block diagram fig. 3.7, the controller aligns a fuzzy rule set to make an association

between the incoming error (e) signal and that of the desired output signal (u). This is the same association made in the gain surface plot in fig. 3.8 and the subsequent output (u) equations found in table 3.7. In standard proportional control, this controller output signal is represented as $u = K_p e + C$ where K_p is the proportional gain, e is the incoming error signal, and C is a constant adjustment term.

With all the necessary information available, the construction of the hybrid automaton is undertaken by use of the continuous dynamics (eq. (4.3)) and use of table 3.7. The significance of table 3.7 is as follows.

Mode This column is used to label each the hybrid modes (locations). Although not necessary to process the behavior of the HA, labeling the modes is important for tracking purposes. Each of these modes are accessible by discrete jumps.

error range There are two pieces of information critical here to the automaton. The first is the generation of the mode invariants. Recall in automata theory, the invariants of a mode is the conditions for which a mode is valid. Should the mode invariant conditions become invalid (such as e is no longer in an acceptable range) the system must transition out of the mode. If the mode can not transition, then it is not properly formed. Also, recall that mode invariants in HA are the only way that mode must be existed. The second is the generation of the jumps which uses the same information as the invariants. Jumps only provide an opportunity to leave or enter a mode, it does not force mode changes. This allows for the inclusion of nondeterministic elements if desired. In the HA constructed in this work, the invariants and jumps are in full agreement as nondeterministic elements were not desired.

Input (u) The input (u) column in table 3.7 includes the static gain of 12.5, therefore, in order to properly represent the block output signal, a gain of 12.5 would need to be included. In addition, the PFLC gain value is then scaled with a multiple of five to further refine the control signal. For example, using the normalized u column the $\dot{x_2}$ dynamics

would be as follows for mode *Mode 1* in eq. (4.4).

$$\dot{x_2} = -20x_2 - 75x_1 + 75 * 12.5 * 5 * 1e \tag{4.4}$$

Even with the information provided by table 3.7, there still exists the need for definition of the state variables of the hybrid automaton. The choice of the number of state variables is significant as the number of these variables each add a dimension of calculations to the analysis tools that will be examined later in the text. For the HA representation of the PFLC, there are three state variables that must be defined: $\dot{x_1}$, $\dot{x_2}$, and $\dot{t}$. The variables $\dot{x_1}$ and $\dot{x_2}$ represent the position and velocity of the system and originates directly from eq. (4.3). The third variable, $\dot{t} = 1$ in the system denotes time often thought of in seconds. Technically, the $\dot{t}$ term is used to provide response tracking and is not necessary should a user have an intrinsic view of the graphed relation of $\dot{x_1}$ and $\dot{x_2}$. In practice, the graphed result of the response over time (x_1 over t) is ideal for the system examination in this work and worth the increased dimensional burden. It should be observed that eq. (4.4) is not completely correct for the HA representation. This is because we did not include a state variable for e which is not an oversight. Recall that the error term can be realized as $e = 1 - x_1$ and therefore is not necessary as a unique state variable saving a dimensional complexity. Furthermore, later use of the analysis tools will illustrate that the formulation for e is not optimal when looking at the full input range. This will be discussed in detail in chapter 5.

4.2.1 Hybrid Automaton Development

In order to build the hybrid automata (HA) representation, the generation of a SpaceEx XML input file format is used. This is done graphically through the SpaceEx Model Editor [30] which was developed as hybrid automata authoring tool in Java. The images in this work shows the HA represented in that tool. Although this tool can be used to build

the automaton, it is not necessary as the tool encodes a SpaceEx XML file format which can be directly edited. In the HA community, particularly with the Hybrid Reachability community, the SpaceEx XML file format has become a standard representation for many different tools. Therefore, building the system in this format provides the flexibility in analysis tools. In particular, using Hyst [4], a HA made in the SpaceEx format can be translated into alternate formats. This allows for direct comparison of tools, ensuring equivalent representations of the same model and its configuration. It is the use of Hyst that allows the comparison of the analysis results in chapter 5 and also the testing the simulation response in python.

Before showing the entire hybrid automaton, let us first examine the *Loc1* and *Loc2* modes in fig. 4.3, their invariants, continuous dynamics, state variables, and jumps.

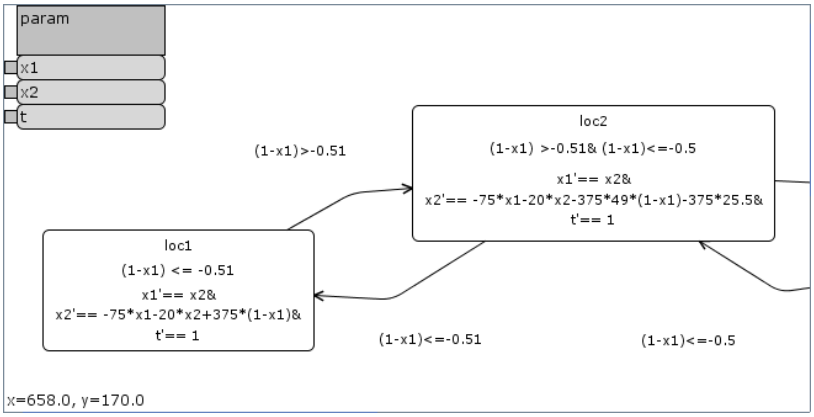

Figure 4.3: Partial representation of the PFLC hybrid automaton modes Loc1 and Loc2.

It can be seen in fig. 4.3 that invariant of mode *Loc1* is $e < -0.51$. By leaving the invariant as $e < -0.51$, should a situation arise where the e is less than -1.0 will be handled by the same application of gain. The continuous dynamics are a conjunction ($\&$) of all the state variables. The most significant of the state variables being $\dot{x_2}$ (which is denoted as *x2'*). For the $\dot{x_2}$ dynamics recall that the *Loc1* block output must be included for

u(table 3.7) making the dynamics in eq. (4.5).

$$\dot{x_2} = -20 * x_2 - 75 * x_1 + 75 * 5 * 1 * (1 - x1) \tag{4.5}$$

In the event that the invariant is no longer valid there is only one direction that the system can progress, on the jump condition (guard) that $e >= -0.51$. This is the arrow that goes from mode *Loc1* to mode *Loc2*. The mode *Loc2* is constructed in a similar fashion as *Loc1*. To return to mode *Loc1* from mode *Loc2*, the error (e) must evolve to violate the *Loc2* invariant and take the jump condition (guard) that returns to (points to) *Loc1* of $e < -0.51$. With this process defined, the entire automaton can be constructed and can be seen in fig. 4.4

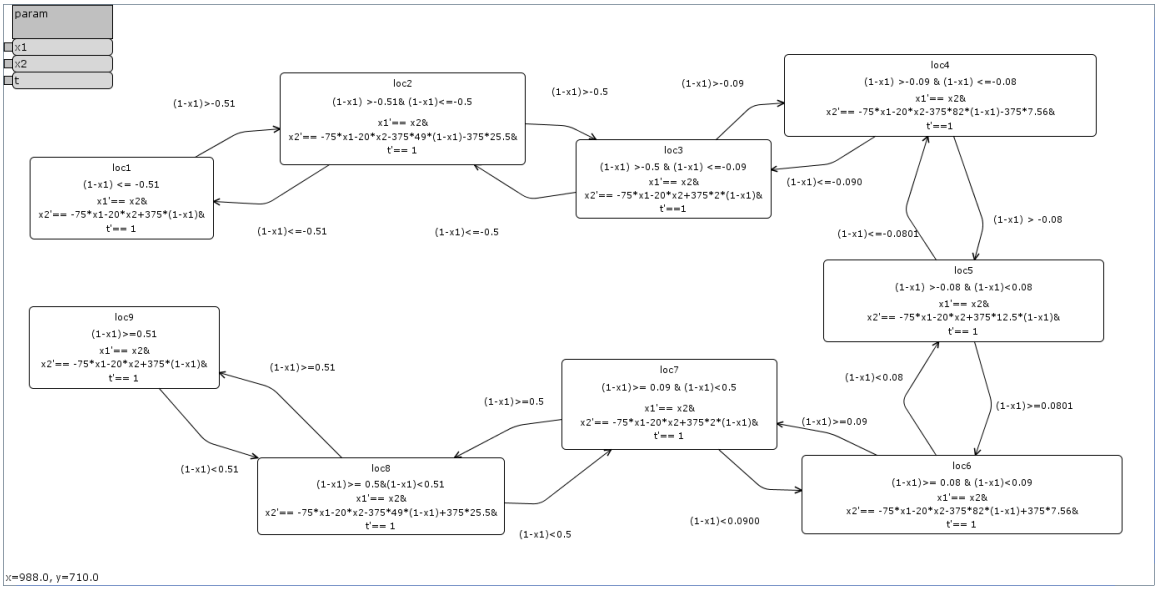

Figure 4.4: The hybrid automaton representation of the PFLC.

```
system = Example1
initially = "x1 == 0.0 & x2 == 0 & t == 0"

# Image FPCtrl_xxx_v1.png settings
set-aggregation = chull
scenario = stc
directions = box
sampling-time = 0.001
time-horizon = 3.5
iter-max = 20
output-variables = "t, x1"
output-format = GEN
rel-err = 1.0E-6
abs-err = 1.0E-5
flowpipe-tolerance = 0.01
```

Figure 4.5: SpaceEx configuration file (FPCtrl.cfg) to provide initial settings for a PFLC.

4.2.2 Configuration File

In SpaceEx, the configuration file (labeled as a .cfg), defines the initial values of the state variables, set aggregation techniques, the flowpipe growth preference, propagation limitations, etc. necessary to enable the system to produce acceptable results. In fig. 4.5 the complete configuration file for the three-gain system is defined. Descriptions of each line is explained in the following listing. It is worth noting that there are other configuration file entries that are acceptable and have an influence on the generated results. This subset is geared toward the three-gain system in this example. Additional result formulations may discuss these settings later in this work.

- system - The name of the automaton within the SpaceEx model file that this configuration applies.
- initially - This defined the initial values of the state variables. This field can include ranges of values if the initial value is to be chosen by a sudo-random selection.
- scenario - This is the type of algorithms to propagate the system states.

  supp - support functions

stc - refined support functions algorithm with better error control

sim - a single simulation of the dynamics

phaver - basic phaver tool algorithm

- directions - This is the number of discrete dimensions to grow the containing set. The higher the order number, the more processing intensive the calculations.

  box - Four directions of growth from a bounded center point

  oct - Eight directions of growth from a bounded center point

- set-aggregation - Defines the method on how to build the resulting set after each iteration.

  chull - Ensures that a convex hull is generated from the propagation of the set. If the hull is not convex (concave) and over-approximation is executed to ensure the convex set.

- sampling-time - This is the discrete time step of the system to sample the evolution of the state variables.

- time-horizon - The maximum time for which to sample the system. This is one of two settings (the other being "iter-max") that can govern the attempted execution length of the analysis. The analysis is concluded whichever of these two settings occurs first.

- iter-max - This is that maximum number of sampling iterations for the system. This is the other setting that governs the execution length of the analysis. The analysis is concluded when this setting is reached or the "time-horizon" is reached.

- output-variables - These are the state variables to output for the system.

- output-format - The output format of the analysis.

  gen - Generates a graph of the output variables

- rel-err - An error tuning field.
- abs-err - Another error tuning field.
- flowpipe-tolerance - This is a tolerance level used to define how the exact the calculation of the flowpipe construction is to be completed. For instance, a flowpipe tolerance of 0.001 will calculate the tolerance within this defined range. Increasing the precision of this tolerance can have significant implications on the runtime capability of the models.

4.2.3 Python Plot Response

To verify that the PFLC hybrid automaton is properly configured prior to analysis and to ensure that the response behavior is correct, a simulation visualization is encouraged. Using Hyst [4], the SpaceEx XML file and the SpaceEx configuration files are translated into a python script. This allows PySim to be run on the HA which should create a similar response plot as we have previously seen in Matlab. The raw files can be seen in Appendix A under section A.7.

Running PySim generated the response as seen in fig. 4.6. When compared with the output generated from Matlab in (fig. 3.10) the response curves show agreement. This is an expected result as the same gain surface plot that was implemented in the PFLC was used to generated the hybrid automaton.

4.3 PselDFLC to HA: Inclusion of Selective Damping

As previously shown, the response can be further improved with the inclusion of the derivative term in the response. This is shown through the Simulink response where improvement of the overall system oscillations and overshoot are both better managed. Adding a deriva-

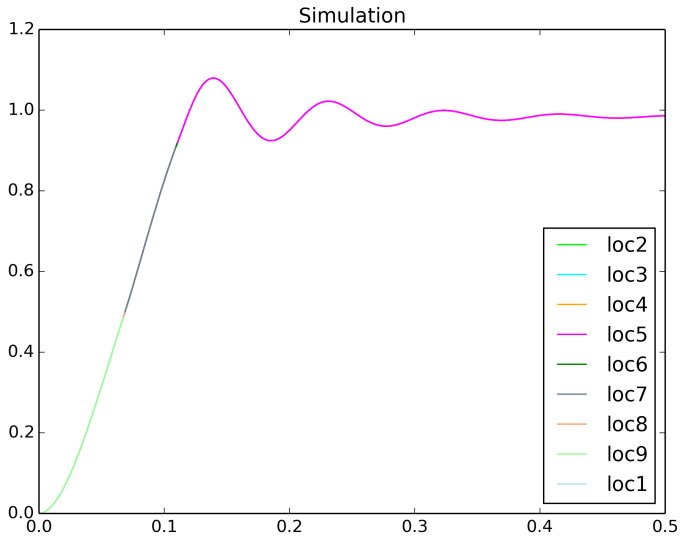

Figure 4.6: PySim response plot generation with initial conditions of $(x_1, x_2, t) = (0, 0, 0)$ for the PFLC.

tive term to the entire hybrid automaton can add significant additional modes and complexity to the system representation. This is because each transition must now be done on both the e and $\dot{e}$ terms. For instance, the product of membership functions minus one of each input is the largest number of modes that needs to be created in the system. In this example, the error fuzzy set has eleven membership functions making the error (e) mode dimension equal to ten. In the change of error ($\dot{e}$) dimension, the total of three membership functions provides two modes. Therefore, the total number of modes possible in the system is $mode_{num} = 10 * 2 = 20$. In the error (e) dimension, due to the symmetry of the gain about 0, the zero mode can be removed reducing the set of modes to nine in that dimension and therefore 18 total.

Before building a HA with 18 modes, let us consider an alternate approach to still only switch on the e dimension, thus using only nine modes. This can be done in a hybrid automaton since the added derivative terms are constant and are restricted to specific modes in the system. Since each mode's continuous dynamics are independently defined, the responsibility to ensuring that the dynamics are correct fall to that of the designer. Therefore,

should the continuous dynamics be changed to only include K_d in specific modes that is possible and is desired in this example system. Note is table 3.10 the $K_d = 1$ terms exist in *Modes 4,5,6*. Such a design choice allows a constant K_d input multiplier on the $\dot{e}$ feedback loop gain of 0.15. Therefore, adding the K_d term can improve the system response by changing the dynamics of $\dot{x_2}$ in those modes. This is done by changing the controller output to $u = K_p e + K_d \dot{e} + C$ as seen in eq. (4.6). Also note that the constant multiplier (0.15) is introduced to the K_d value.

$$\dot{x_2} = -20x_2 - 75x_1 + 75 * 5 * (K_p e + 0.15 * K_d \dot{e} + C) \tag{4.6}$$

With the derivative defined for these modes, it is now incorporated into the HA as seen in fig. 4.7.

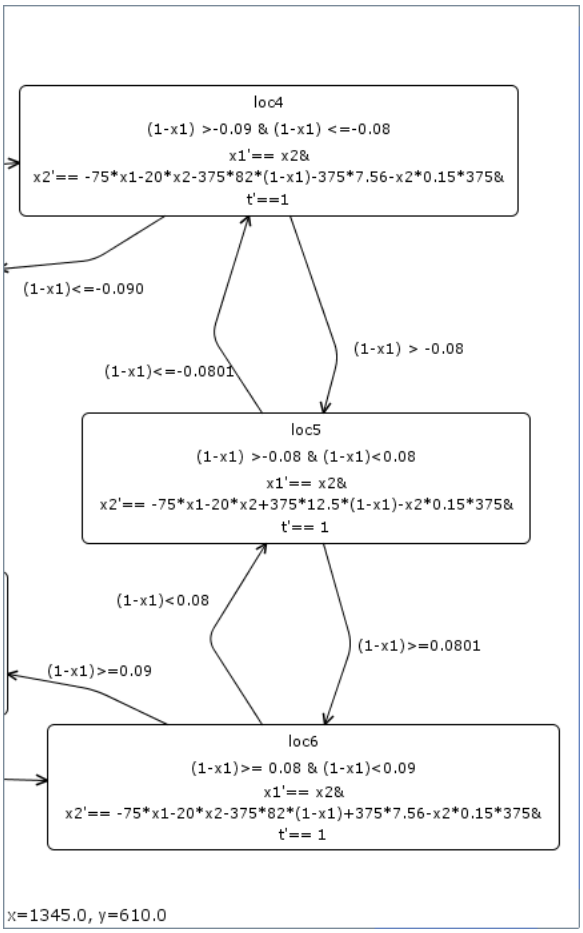

Figure 4.7: Modes 4, 5, and 6 of the PselDFLC HA that adds the derivative damping.

4.3.1 Python Response

To ensure that the representation as a hybrid automaton is comparable, again PySim is utilized to simulate the response. Here, the system is not a zero-input system and therefore uses initial $x1 = 0$ input to replicate the step response characteristics. The resulting response, as seen in fig. 4.8, maps well in comparison to the PselD representation built in Matlab (fig. 3.12). The only difference in the response is that of the extents of the time horizon on the horizontal axis. Therefore, the representation of the hybrid automaton of the PselD controller is determined to be correct for reachability analysis.

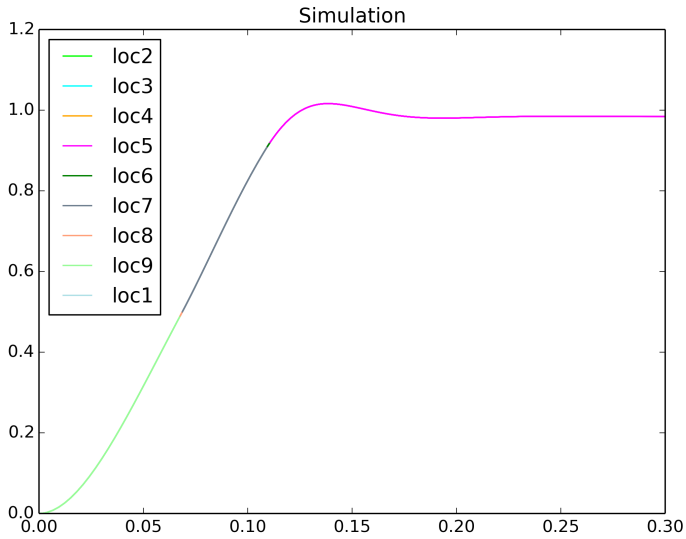

Figure 4.8: PySim response plot generation with initial conditions of $(x_1, x_2, t) = (0, 0, 0)$ for the PselD (PDFLC).

4.4 Summary

In this chapter the author used the previously defined example (both the PFLC and PselD-FLC) to translate the control systems to hybrid automata (HA) representations. As a key part of the thesis, this process was described in detail so the reader may be able to recreate the process in the future. With the defined fuzzy representations, the first step to convert to hybrid automata representations was redefining the plant continuous dynamics as ordinary differential equations. This formulation of the dynamics is necessary for tool use later in the thesis and therefore the author used response modeling to ensure that various formulations agreed. The author then constructed HA representations using previously defined linear piecewise affine calculations from the work of Rattan and Clark [17]. After generating the HA representations, the author then leveraged PySim to model the response of the new representation. Through inspection, agreement of the responses (GS, PFLC, HA) is demonstrated. As before, the author also executed the process with the selective derivative controller systems.

Chapter 5

Reachability Analysis

It has now been established that the original linear gain scheduled controller first described in chapter 3 was able to be converted to a PFLC using linear piecewise calculations. These same linear piecewise calculations were used to change the PFLC representation into a hybrid automaton by methods described in [17, 16]. Demonstrating that response characteristics were maintained through each translation provided evidence that system is able to be equivalently represented.

As described in chapter 2, the ability to capture the dynamics of cyber-physical systems (CPS) in automata theory provides capabilities to examine dynamics beyond the traditional methods. This work uses reachability analysis to analyze the example system. Recall that reachability analysis provides capability to examine the evolution of a system from a defined set of states. This defined set of states, known as the initial states, provides a realization of a system within a state space (not to be confused with state-space in the controls community). A state-space is the dimensional domain that state variables can evolve often with each independent state-space variable occupying a single dimension of the space.

The significance of this analysis is realized when applied to system evolution over time. The initial states are realizations of the overall system state at some instance. Reachability tools allow for forward or backward evolution from that initial realization providing a

visualization of how a the initial states can evolve. This evolution of sets that move forward or backward in a time-horizon from the initial set can be stitched together to form flowpipes. A significant benefit of reachability analysis is that it evolves the system bounded only by computational resources; the base algorithms do not artificially limit the propagation of the set of states. Current research into reachability tools have made advancements at improving the computational limitations of this analysis.

Two analysis methods include forward or backward reachability [51]. Forward reachability starts from an initial set of states, guaranteed to be *safe* and progresses the evolution forward in time. The goal is to be able to claim that any evolution (flowpipe) that originated from that set of states will not enter and state-space regions deemed to be *unsafe*. Backward reachability reverses the process to defining a set of states that encompasses an *unsafe* region. This region can be characterized and progressed back in time to find all regions that could lead to the *unsafe* state.

This work focuses on the use of forward reachable sets in order to confirm various system characteristics, specifically statements about stability. This is significant for the controls community as it provides for an analysis technique that is not bounded by many of the linear system limitations. Therefore, evidence of the applicability to the example linear problem will provide confidence in expansion to more complex nonlinear systems. To accomplish analysis of stability, this work will discuss results from two specific hybrid reachability tools: SpaceEx [32] and Hylaa [5].

5.1 Reachability Results for PFLC

With the control output signal (u) defined for the system, and the invariants and jump conditions also defined, a hybrid automaton was constructed as seen in fig. 4.4. In order to ease reachability calculation, the error (e) term was to be calculated against energy being taken out of the system. Therefore, the initial input range of $[-1, 1]$ covers the $\pm 100\%$

overshoot/undershoot assumption. As a simplification, $e = 0 - x_1 = -x_1$ for the zero input as seen in fig. 5.1.

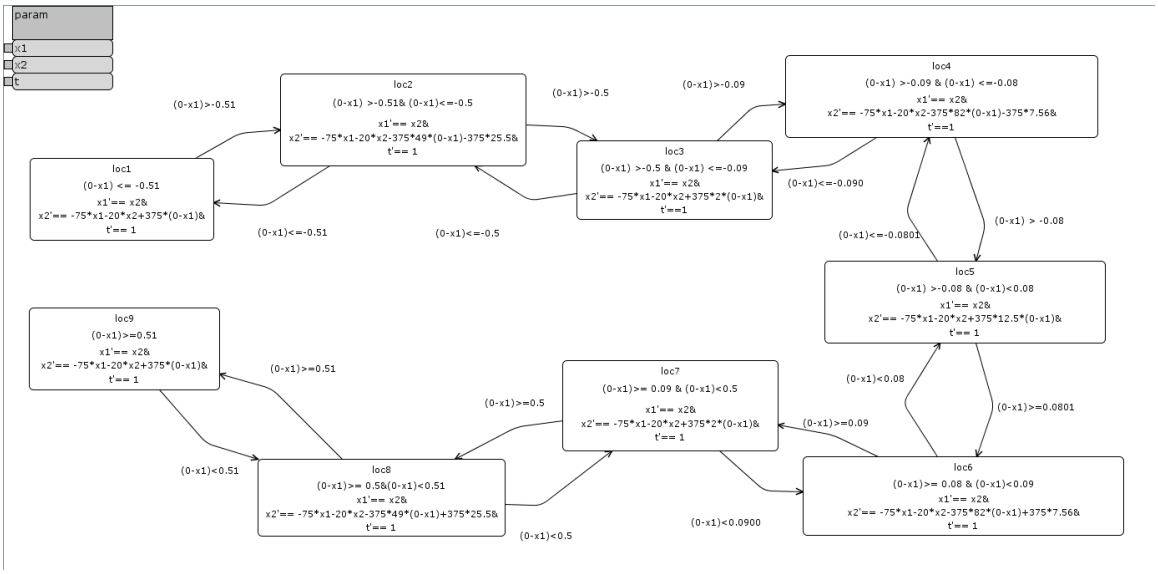

Figure 5.1: The hybrid automaton representation of the PFLC controlled system with zero input.

Using the zero input system lets the user analyze a number of traits to determine the performance characteristics. Most importantly, the overall system reachability can be observed.

5.1.1 SpaceEx Zero Input PFLC

In order to visualize the system reachability, the author chose to build up the number of iterations for flowpipe propagation. The first resulting plot was set to nine iterations, as seen in fig. 5.2, to demonstrate flows from initial values that originate within the specific modes. It shows that each flow evolves to the boundary condition without possibility to exhibit uncontrolled aspects. This is seen by all polytope growth ending in a flowpipe to the boundary extent.

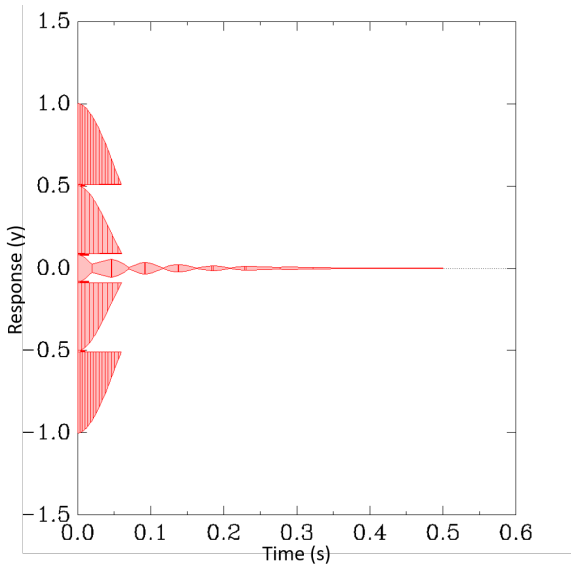

Figure 5.2: SpaceEx flowpipe representation of a zero input PFLC controlled system for 9 iterations.

The next case increased the target iterations to 18. This resulted in evidence that the system begins to build the pipes from the previous mode boundaries that were seen. Unfortunately, fig. 5.3 shows that doubling the iterations was insufficient to fully evolves reachability representation. It is shown that the iterations only propagated the modes that started from negative initial inputs.

After observing the previous system results, the author continued to increase the max iterations of the system by nine each time resulting in continued refinement of the flowpipes. This is seen in fig. 5.4 where at 27 iterations the flowpipe has generated expanded initial started values for adjacent modes. At 36 iterations, this expanded initial set is further propagated throughout the system. Notice that after 36 iterations, there still appears to be flowpipe elements that continue to expand. This is evident by the -0.51 to -0.5 mode (*Loc2*) where the propagation has not achieved symmetry with that of the 0.51 t0 0.5 mode.

The final reachability graph generated by SpaceEx included 41 iterations on the system and can be seen in fig. 5.5. Although it appears to have full symmetry as intended by the designed system, the flowpipe construction does not appear complete. In Mode 3 and 7

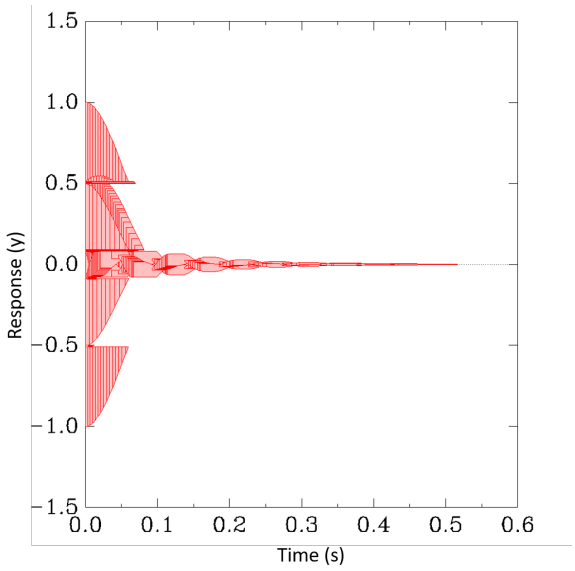

Figure 5.3: SpaceEx flowpipe representation of a PFLC controlled system for 18 iterations.

($[\pm 0.51, \pm 0.5)$], there appears to be adjacent mode initial values that is not propagated forward. Unfortunately, limitations on how the system was being generated prevented further iteration examination. This could be a limitation on the capability of the solver technique, the fidelity of the system with respect to system resources, or even an inability of the solver to handle zeno behavior. Zeno behavior occurs when flowpipes continually propagated without making progress on refining the set. Similar to infinite loop behavior in programming, the solver can not determine how to resolve the pipe evolution.

Even though the system was not able to be fully resolved with SpaceEx, there are a number of positive aspects to consider. First, the reachability between the modes appear to be more complete as iterations increased. Observe the difference between figs. 5.2 and 5.3, the abrupt terminations at the mode boundaries evident in nine iterations are starting to be reasoned about in 18 iterations. This demonstrated that the number of iterations needed to be increased overall to enable more complete analysis. When increasing the iterations to 41 in fig. 5.5, many of the abrupt mode boundaries are propagated, and therefore, it can be deduced that the reachability is more complete.

Second, notice the complete steady-state range in fig. 5.5. Within this steady-state

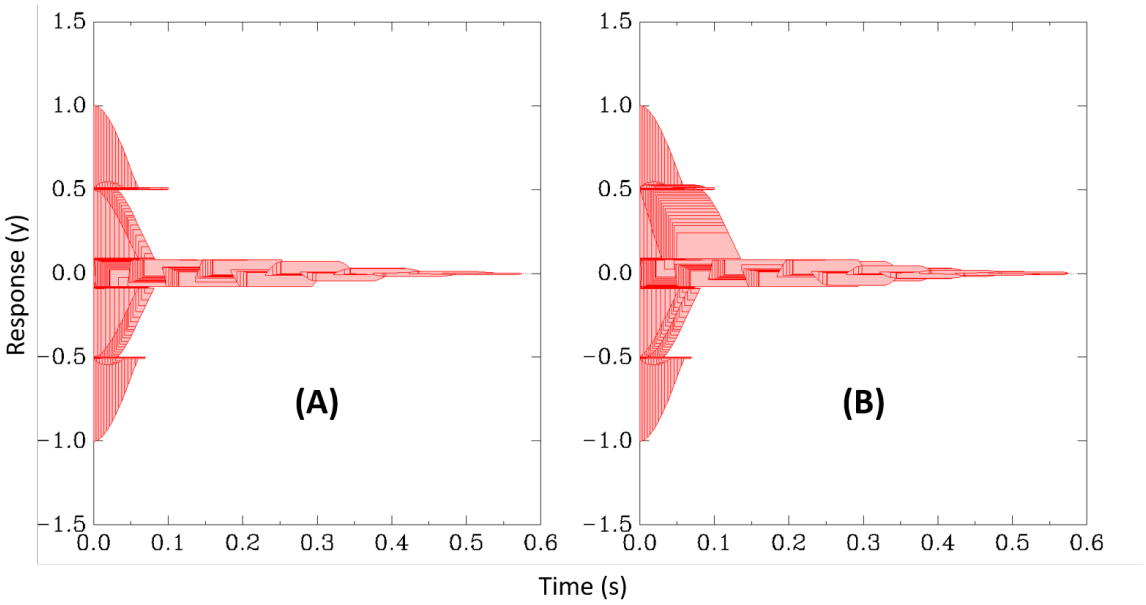

Figure 5.4: SpaceEx flowpipe representation of a PFLC controlled system for (A) 27 iterations and (B) 36 iterations.

area, there appears to be a number of boxes that propagate forward growing in size then stops. Then another box layer becomes defined. This is expected, especially in the PFLC, due to an inability to achieve zero steady state error with only the proportional element. What is being shown is that the oscillation may exist for a longer period of time that causes the flowpipe to be generated, a new flowpipe boundary condition being exposed, and then a new propagation from that exposed condition forward in time. This may be reason to believe that the time horizon is significant as in the last flowpipe appears to start right before that boundary and is propagated forward. Depending on computing resources and tool limitation max-iterations and time horizon terms could be refined to attempt to gain further insight through SpaceEx.

5.1.2 Reachability using HyLAA

With the successful representation of the system in SpaceEx, the choice was to examine the system with Hylaa so to see if the results were found to be similar. The author used Hyst [4]

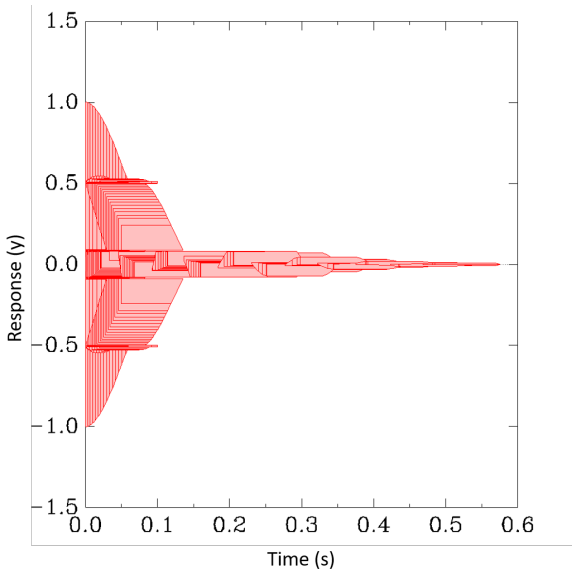

Figure 5.5: SpaceEx flowpipe representation of a zero input PFLC for 41 iterations.

to convert the SpaceEx representation of the system into a Hylaa compatible model. Since Hylaa leverages traditional control dynamics representation, the first step in this conversion is to change the continuous dynamics into $\dot{x} = Ax + Bu$. Since the system was already represented as an ODE, this conversion was well handled by Hyst. Hyst did require use of the Python Simplify package in order to properly represent the system dynamics. This is likely due to the representation of the error (e) in the system. The author, after simplification, saw no negative impacts on the expected dynamics.

Note that Hylaa leverages simulation-equivalent reachability which can handle higher orders of magnitude of the system. This was evident in the reachability generation as the execution time was reduced although exact numbers were not maintained. More importantly, fig. 5.6 shows a complex and well developed representation of the propagation of the flows while in each mode.

Notice in fig. 5.6 that Modes 3 and 7 experience a number of oscillations before settling prior to 0.25 seconds. Given that these zones do not have any derivative control and that the gains are rather small prior to the near target, this flow is not unexpected. Furthermore, recall previously that there is some nonlinear aspect to these zones that is not

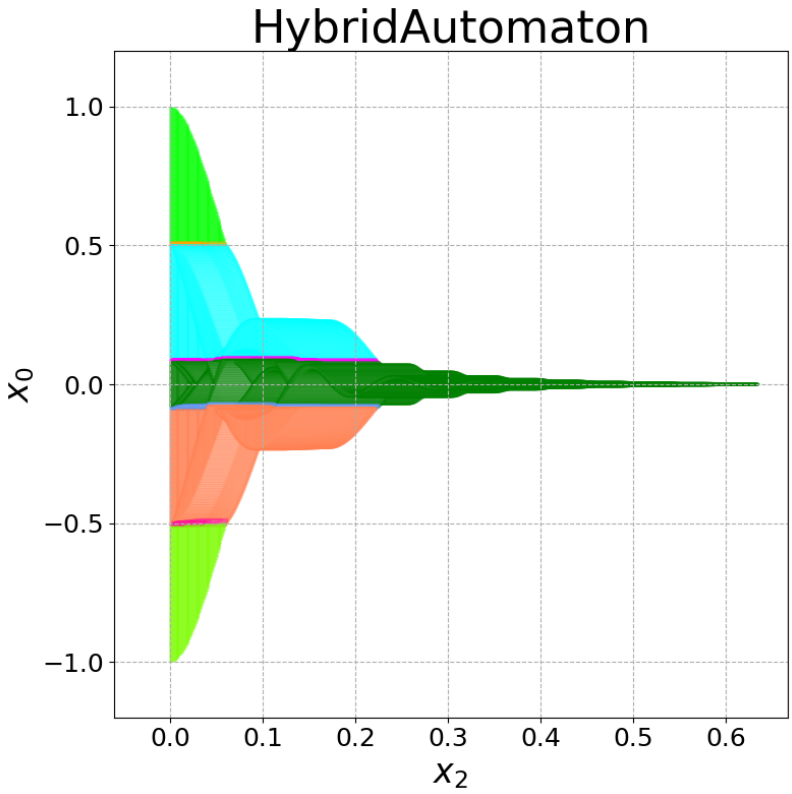

Figure 5.6: Hylaa flowpipe representation of the PFLC controlled system.

included in these mode dynamics which could also create this resulting propagation. This provides evidence that the system first moves away from the steady-state only to recover in these modes as the flowpipe evolves. One can observe that this effect is desired to be removed from the system. It can also be reasoned that this effect points to insufficient blending of the adjacent memberships in this case.

The resulting Hylaa plot proved not only beneficial to evaluation of the characteristics of the system but also was straight-forward to interpret. The coloring of the overall flow allowed for the author to visually decipher how the modes contributed to the entire reachability. The system also identified modes (such as 3 and 7) that may be primed for further refinement to tighten the response.

5.2 Proportional Plus Derivative Reachability Results

In order to improve the response of the example system, a derivative term was added to improve the damping of the system near steady-state. This designed improvement in the PDFLC is converted to a hybrid automaton as described in the previous chapter resulting in fig. 5.7.

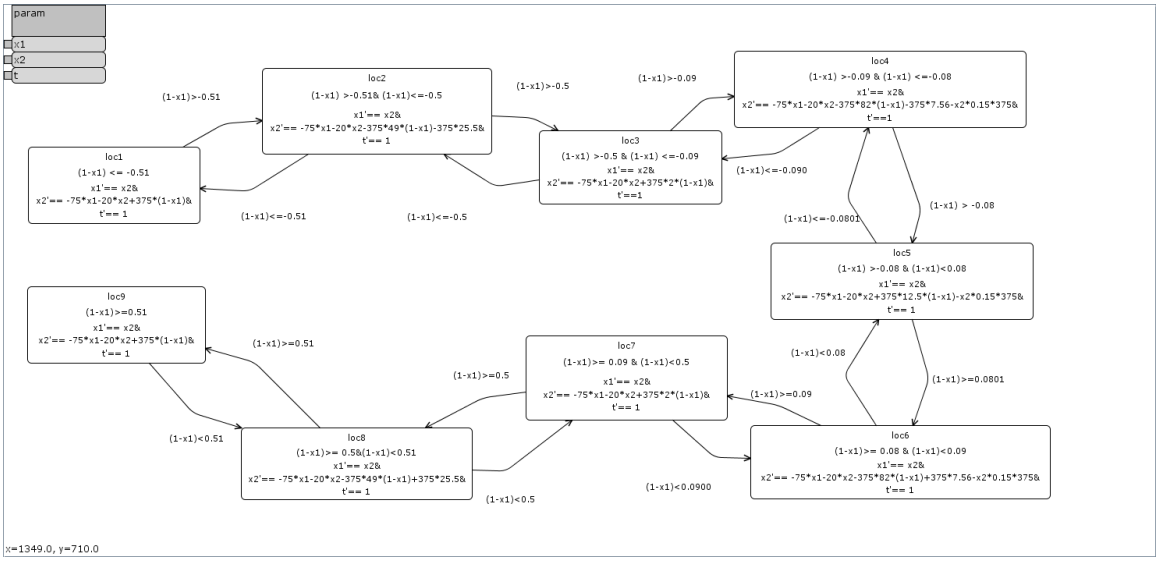

Figure 5.7: Hybrid automaton representing the system controller for the PselDFLC.

Recall that the difference between the PFLC and the PDFLC is that the K_d term was added to the continuous dynamics of *modes 4,5,6*. The other system modes are still only using proportional control. As previously discussed, this provided the benefit of only utilizing the derivative term when it would be most beneficial to the system. In this system, the derivative term is only added near steady-state.

5.2.1 SpaceEx Reachability Analysis

Using the hybrid automaton defined in fig. 5.7, the initial reachability work was done using SpaceEx [30]. In order to verify if the reachability of all of the the hybrid automaton modes,

the input range to the system was set as $-1.0 <= e <= 1.0$. This range, defined as the assumed lower/upper bound of the system in the zero input representation, illustrated the extent of the range.

Using a similar technique as conducted in the PFLC HA analysis, the author first started with nine max iterations of the PselD controlled system as shown in fig. 5.8.

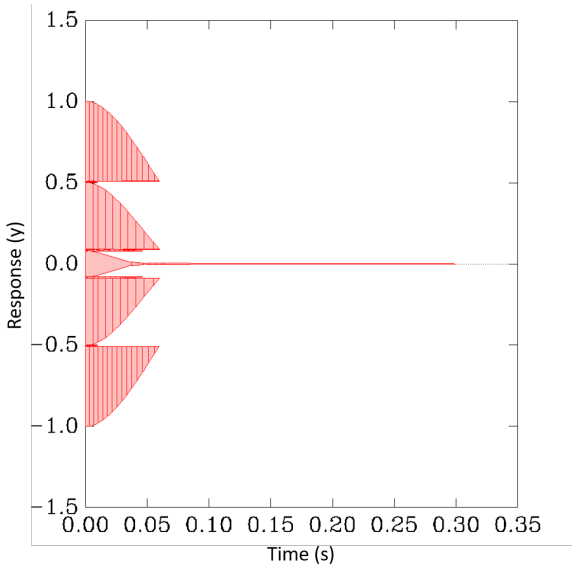

Figure 5.8: Selective PDFLC controlled system HA over 9 iterations using SpaceEx

It can be seen in fig. 5.8 that the initial set of modes are propagated from the initial set of values. One difference from that of the PFLC is evident in the central modes (4,5,6) which exist around the steady-state about the 0.0 value. In the PFLC controlled system, there were a number of set boxes that demonstrated that the flowpipe was being calculated. In the PDFLC, this area reduces quickly to the steady-state behavior. This shows that there is less uncertainty in the evolution of the response.

In a similar process as that of the PFLC, the author increased the number of iterations by nine until the system flowpipe was fully defined or until the solving engine could no longer execute. This process illustrated of the reachability flowpipe through the mode transitions. The maximum iteration results for the sets of 18, 27, 36, 45, 54, and 63 are

shown in fig. 5.9.

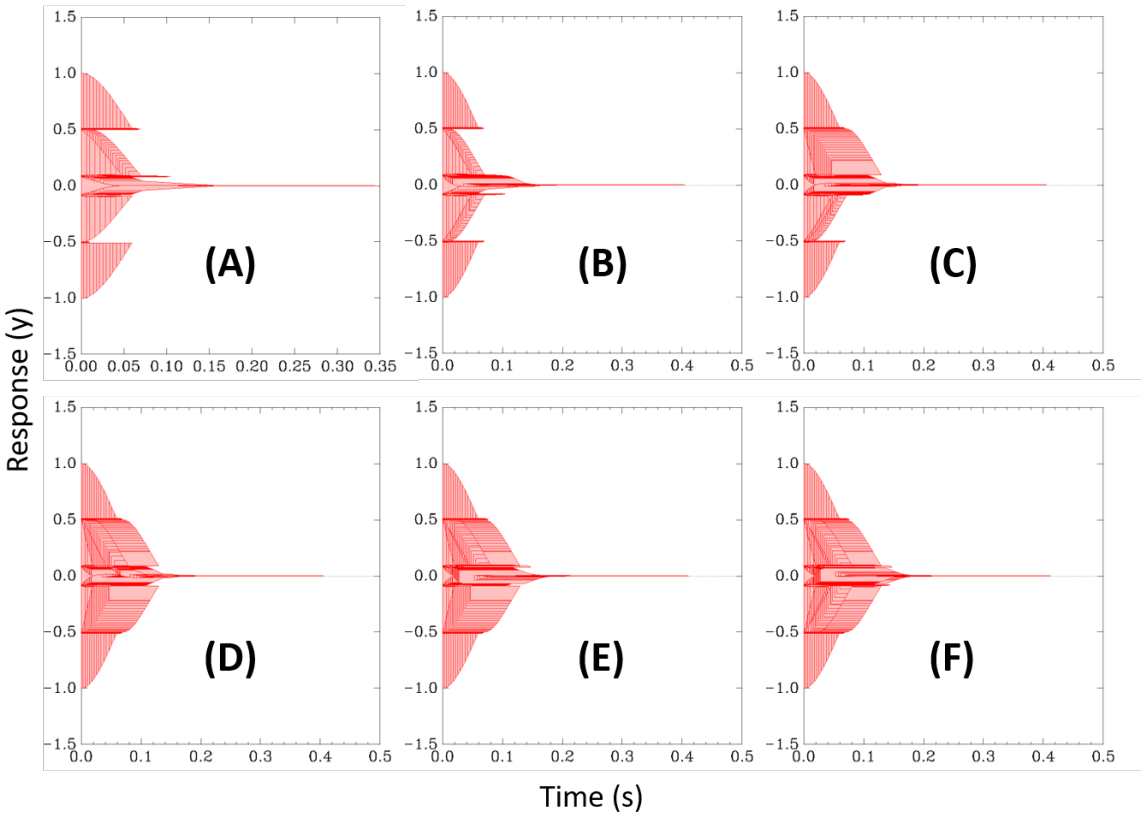

Figure 5.9: The resulting PDFLC controlled system flowpipe from SpaceEx for various maximum iterations: (A) 18, (B) 27, (C) 36, (D) 45, (E) 54, (F) 63.

Again the evolution of the flowpipes can be seen in the images first starting at 18 iterations (A) and continuing on to a maximum of 63 iterations (F). One may notice that the maximum number of iterations here exceeded that which was possible with the PFLC controller. This is not unexpected for one of the ways that reachability algorithms have difficulties is in the scaling the error bounds of a flowpipe. As a flowpipe evolves, the amount of error in the system grows from the initial set. In the PDFLC controlled system, the derivative term reduced the overshoot and oscillation present and thus reduced the amount of accumulated error that must be reasoned about. This allowed the solver to more efficiently progress and therefore system resource limitations were reduced allowing for

deeper execution.

Also notice in fig. 5.9 how many of the adjacent modes provide initial conditions to that of the next node toward the steady-state. In many of the modes, these appear nearly fully realized with no expanded flows that are not resolved forward with the exception of those near the steady-state. Since 63 iterations did not overload the solver, the author proceeded until computation failure at 78 iterations as seen in fig. 5.10.

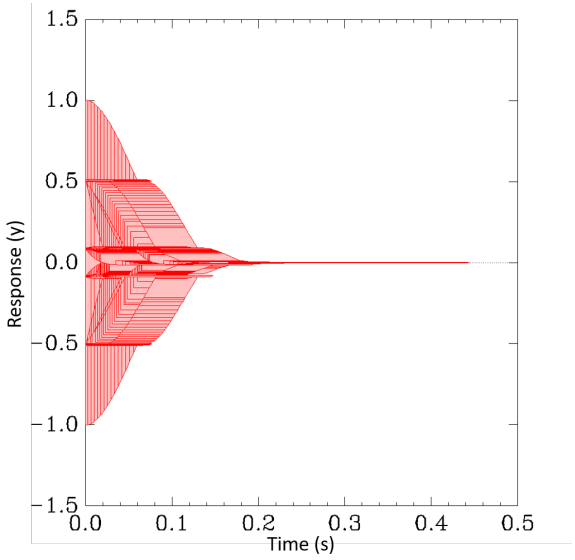

Figure 5.10: elective PDFLC controlled system HA over 78 iterations using SpaceEx.

Achieving 78 maximum iterations in the flowpipe construction of the PDFLC system is a significant achievement for the SpaceEx engine. This level of development shows nearly all mode value propagation and provides a robust reachability realization. Nearly all modes are fully evolved by 0.2 seconds into the generated flowpipe and has little evolved error after that time horizon. This result provides strong evidence that the additional derivative term in the steady-state modes is significant to the calculations. As before in the PFLC, the system does again exceed computational limits by the 79th iteration. Regardless, this result show the power of reachability analysis by providing evidence that this system stability is maintained. To further examine the system and to see if these results are verified in another reachability engine, the author then utilized HyLAA.

5.2.2 HyLAA Reachability Analysis

Leveraging the HYST tool [4], the SpaceEx hybrid automaton input file was converted into a HYLAA representation in Appendix D. HyLAA leverages traditional state space form ($\dot{x} = Ax + Bu$) to define the continuous dynamics of the modes. In the included HyLAA script, the resulting simulation fig. 5.11 appears to produce a similar result from that seen in SpaceEx after 78 iterations (see fig. 5.10). This shows that these tools, although calculating the flowpipe in different ways, are both able to achieve comparable results. One difference involved the processing improvement in HyLAA. The resulting graph was generated where termination conditions were met and flowpipe propogation was complete. In SpaceEx, the flowpipe generation at the higher numbers of iterations was not able to fully explore the grown boundary conditions of each propagation.

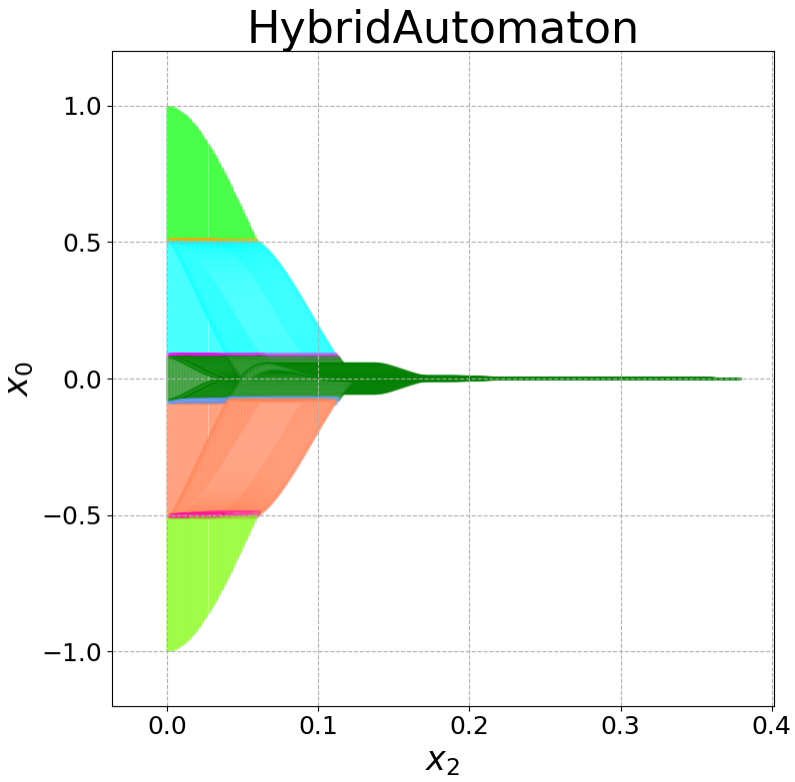

Figure 5.11: Final HyLAA plot of the reachability of the system as controlled by the hybrid automaton defined in fig. 5.7.

In HyLAA, the simulation reachability algorithm [5] conducts mulitple rounds of propagation in order to ensure that the system's full reachability envelope is constructed. The first iteration of this algorithm (fig. 5.12A) is shown where only the initial values that start within the mode invariants are shown. Observing the results in comparison to those generated by SpaceEx (fig. 5.8), there are strong similarities in the results. This provides agreement between both tools at the early stage of the reachability examination. Construction of the reachability envelope is continued from what is seen in fig. 5.12A. In fig. 5.12B, the first neighbor adjoining modes build onto the possible initial sets of the neighboring mode. For example, in moving from modes loc9 ($e \geq 0.51$) to loc8 ($0.50 \leq e < 0.51$) there is a significant range of values in time where loc8 would also propagate. This stage incorporates the range and ensures it is covered in a recursive manner: this process continues until the terminated mode is fully evaluated.

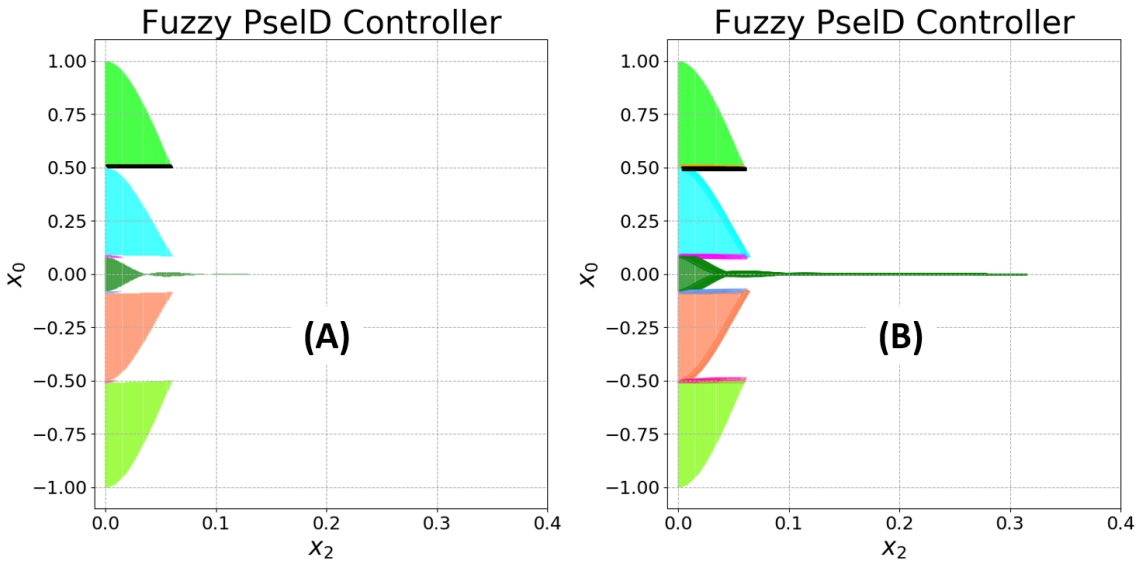

Figure 5.12: (A) Reachability of PselDFLC from all initial values of a mode. (B) Reachability of PselDFLC as adjoining mode propagation is taken into account.

The next pass in the algorithm expands the reachability growth to the next neighboring mode as seen in fig. 5.13A,B,C. This is captured to show that the simulation reachability

calculation is itself propagating a bounding shape in its calculations. In SpaceEx, a box (square) propagation shape was used while in Hylaa a hexagonal shape appears to be leveraged. Additional surfaces on the shape propagation can add complexity in the evaluation. At the end of the iteration, fig. 5.13D shows how the system has evolved with propagating the two neighbored modes initial values through the system. The final pass completes the

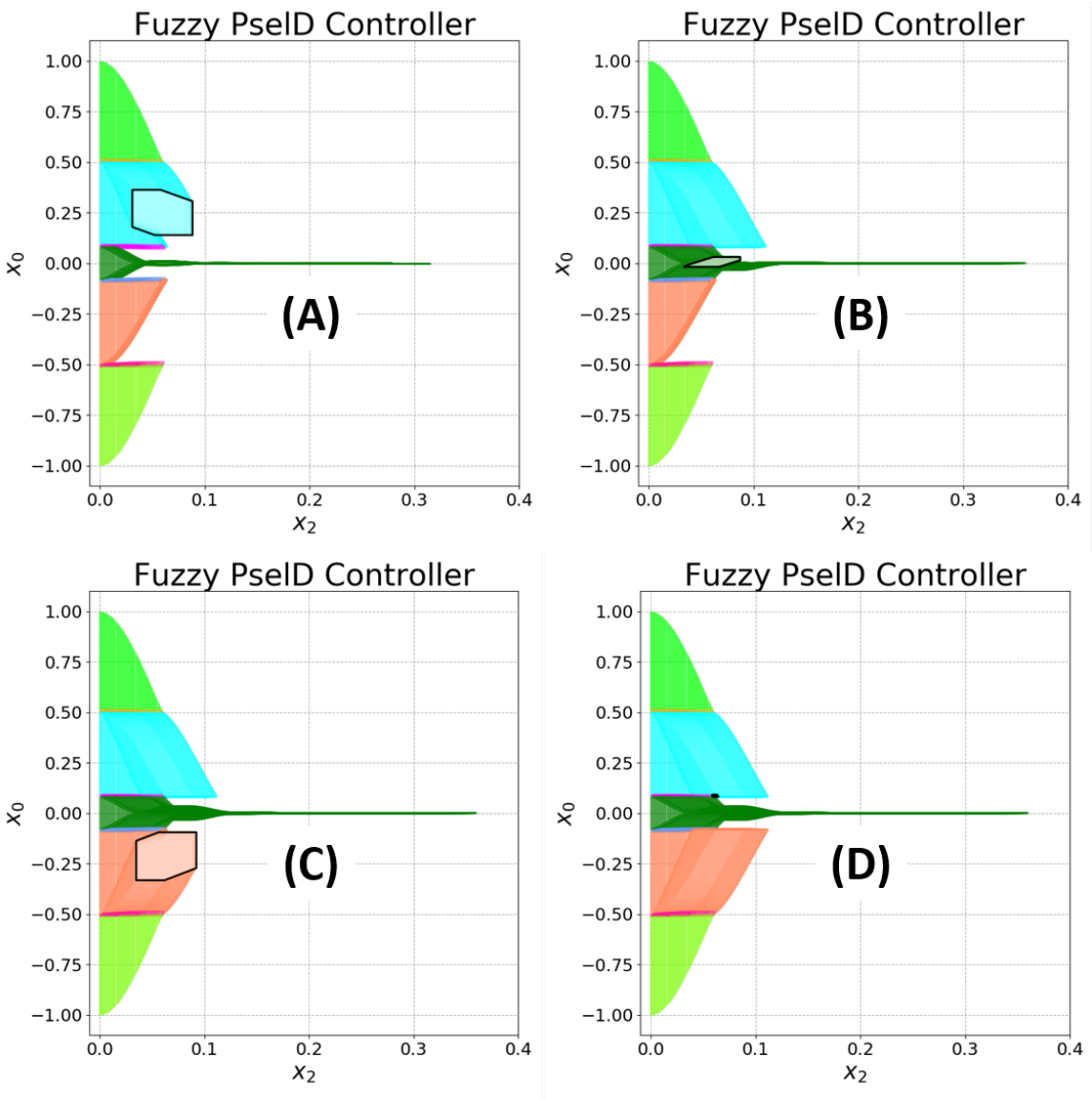

Figure 5.13: (A,B,C) Example of the shape dynamics being calculated in the system and (D) Completion of system reachability as the initial seeds of two adjoining modes are taken into account.

reachability area to combine all the flow from the initial seeded input range. The reachability result from HyLAA was previously shown in fig. 5.11.

5.2.3 Reachability Analysis Comparison

It can be seen from the SpaceEx and HyLAA reachability diagrams that the flowpipe propagation demonstrates the system is stable and will not experience unstable response. This conclusion is illustrated through the flowpipe evolution over time. As it progresses, the flowpipe became more settled at the steady-state region of the response. Even in the PFLC, a bounded region evolved naturally from the response representing an extent of the settling oscillations. Should the system have been made unstable, the flowpipes generated would grow in an unbounded fashion to the extent of the time horizon. Critical to the flowpipe validity, the accuracy of system dynamics ensured that undecidable aspects of the solution can be mitigated and hence the results can be trusted. It is only inaccuracies in the dynamics that can introduce flaws in the calculation of the reachable sets. To further strengthen the stability claim, additional techniques can be leveraged, such as, work to identify unstable zones, describe zones mathematically, and run simulations to ensure that these "keep out" zones can not be entered by the system evolution.

It is also worth noting that the benefits of simulation reachability with respect to the completeness of the system flow. In SpaceEx, the complexity of propagating past the initial mode stimuli was a significant challenge where adjoining mode propagation could not be taken into account. HyLAA navigated past this issue and produces a more complete representation of the effect of the entire hybrid automaton on the system.

5.3 Summary

In this chapter, the author used the hybrid automata representation constructed in the previous chapter to do reachability analysis of the systems. The author first used SpaceEx to analyze the PFLC system and described the resulting generated flowpipes of the system. Then the same PFLC system is analyzed with HyLAA which uses simulation based reachability. This tool demonstrated improved functionality to determine reachability bringing the community closer to using such a tool and analysis on actual systems.

As before, the author also included analysis on the PselD system. The inclusion of the derivative term proved to improved performance of both tools (SpaceEx and HyLAA) in the analysis capabilities. This was especially evident in SpaceEx where the number of iterations that were analyzed increased significantly over the PFLC. A likely reason for this improvement is that the derivative term improved the continuous dynamics calculations which cause less error to be propagated in the system.

Chapter 6

Conclusion and Future Research

This thesis demonstrated a process to convert represented gain scheduled controller into comparable fuzzy controller for a simple system. Gain scheduled controllers have traditionally necessitated empirical data to ensure that disjoint control representations converge to a stable transition range. Using process conversions, the gain scheduled controllers were implemented as triangular fuzzy system controllers. This allowed for smooth transitions between designed controller setpoints, thus eliminating the disjoint representation that is typically represented in gain scheduled control. Smooth control transitions also provide for a continuous system representation that allow for fuzzy control systems to be represented as hybrid automata. Many modern systems are being represented by hybrid automata which allow for continuous and discrete dynamics captured in a unifying construct. Due to this advancement, reachability tools, such as SpaceEx [30], HyLAA [5] and others, have been advancing the boundaries of the capabilities to represent and analyze hybrid system. The process in this thesis leveraged this hybrid automata reachability research to provide evidence of fuzzy system stability and performance not previously accessible.

To continue to explore the feasibility of this process, translations of known (and more complex) gain scheduled control systems should be undertaken. One example of a complex gain scheduled control system is that which governs the flight dynamics of the Lockheed

Martin F-16 fighter aircraft. A version of the flight controller, published in the public domain [49], includes a number of sub-components that would be applicable to this process. Evidence to show these more complex nonlinear control realizations can be implemented using this process and would be valuable to the overall control community.

Bibliography

[1] Mil-hdbk-516c. Technical report, United States Department of Defense, 2014.

[2] Matthias Althoff, Colas Le Guernic, and Bruce H Krogh. Reachable set computation for uncertain time-varying linear systems. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 93–102. ACM, 2011.

[3] Pierre Apkarian and Pascal Gahinet. A convex characterization of gain-scheduled h/sub/spl infin//controllers. *IEEE Transactions on Automatic Control*, 40(5):853–864, 1995.

[4] Stanley Bak, Sergiy Bogomolov, and Taylor T Johnson. Hyst: a source transformation and translation tool for hybrid automaton models. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 128–133. ACM, 2015.

[5] Stanley Bak and Parasara Sridhar Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 173–178. ACM, 2017.

[6] Stanley Bak and Parasara Sridhar Duggirala. Rigorous simulation-based analysis of linear hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 555–572. Springer, 2017.

[7] Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.

[8] J. Y. Beziau. What is many-valued logic? In *Proceedings 1997 27th International Symposium on Multiple- Valued Logic*, pages 117–121, May 1997.

[9] Siddhartha Bhattacharyya, Darren Cofer, D Musliner, Joseph Mueller, and Eric Engstrom. Certification considerations for adaptive systems. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 270–279. IEEE, 2015.

[10] Michael S Branicky. Topology of hybrid systems. In *Decision and Control, 1993., Proceedings of the 32nd IEEE Conference on*, pages 2309–2314. IEEE, 1993.

[11] Thomas Brehm and Kuldip Rattan. A classical controller: a special case of the fuzzy logic controller. In *Fuzzy Logic and Intelligent Systems*, pages 125–155. Springer, 1995.

[12] Thomas Brehm and Kuldip S Rattan. Proportional fuzzy logic controller: classical proportional-plus-derivative like response. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 3, pages 2029–2033. IEEE, 1995.

[13] Franck Cassez, Thomas A Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 134–148. Springer, 2002.

[14] Xin Chen, Sriram Sankaranarayanan, and Erika brahm. Flow*: A verification tool for cyber-physical systems.

[15] Mathhew A Clark and Kuldip S Rattan. Piecewise affine hybrid automata representation of a multistage fuzzy pid controller. In *2014 AAAI Spring Symposium Series*, 2014.

[16] Matthew Clark, Kuldip Rattan, Nicholas Ernest, Tim Arnett, and Kelly Cohen. Verification and validation of a genetic fuzzy tree for ucav control. 2017.

[17] Matthew A Clark and Kuldip S Rattan. Hybrid representation of rule-based systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 287–288. ACM, 2015.

[18] Department of Defense (DoD) Autonomy Community of Interest (COI). Department of defense research & engineering autonomy community of interest (coi) test and evaluation, verification and validation (tevv) working group technology investment strategy. Technical report, Office of the Assistant Secretary of Defense, 2015.

[19] Matthew Dillsaver, Matthew Clark, and Xiaodong Zhang. *Military Airworthiness Certification of Autonomous Air Vehicles with Adaptive Controllers*. AIAA Information Systems-AIAA Infotech @ Aerospace. American Institute of Aeronautics and Astronautics, 01/05; 2017 2017; 2017. M1: 0; doi:10.2514/6.2017-0564; M3: doi:10.2514/6.2017-0564.

[20] Georgios Fainekos, Sriram Sankaranarayanan, and Bardh Hoxha. Keymaera: A hybrid theorem prover for hybrid systems.

[21] Rafael Fierro, Frank L Lewis, and Kai Liu. Hybrid control system design using a fuzzy logic interface. *Circuits, systems, and signal processing*, 17(3):401–419, 1998.

[22] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer, 2011.

[23] Herv Guguen, Marie-Anne Lefebvre, Janan Zaytoon, and Othman Nasri. Safety verification and reachability analysis for hybrid systems. *Annual Reviews in Control*, 33(1):25 – 36, 2009.

[24] Thomas A Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.

[25] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 373–382. ACM, 1995.

[26] Adrain Ilka. *Gain-Scheduled Controller Design*. PhD thesis, 2015.

[27] J. S. R. Jang and N. Gulley. Gain scheduling based fuzzy controller design. In *Fuzzy Information Processing Society Biannual Conference, 1994. Industrial Fuzzy Control and Intelligent Systems Conference, and the NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic,*, pages 101–105, Dec 1994.

[28] C. S. Kang, C. H. Hyun, Y. T. Kim, J. Baek, and M. Park. A design of equivalent pid structure control using fuzzy gain scheduling. In *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 354–356, Oct 2013.

[29] Thomas Koshy. *Discrete mathematics with applications*. Elsevier, 2004.

[30] Verimag Labs. About spaceex, 2010.

[31] Verimag Labs. Spaceex downloads, 2010.

[32] Colas Le Guernic and Antoine Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010.

[33] Werner Van Leekwijck and Etienne E. Kerre. Defuzzification: criteria and classification. *Fuzzy Sets and Systems*, 108(2):159 – 178, 1999.

[34] Douglas J Leith and William E Leithead. Survey of gain-scheduling analysis and design. *International journal of control*, 73(11):1001–1025, 2000.

[35] John Lygeros. A formal approach to fuzzy modeling. *IEEE Transactions on Fuzzy Systems*, 5(3):317–327, 1997.

[36] Ebrahim H Mamdani and Sedrak Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*, 7(1):1–13, 1975.

[37] Vilém Novák, Irina Perfilieva, and Jiri Mockor. *Mathematical principles of fuzzy logic*, volume 517. Springer Science & Business Media, 2012.

[38] Andy Packard. Gain scheduling via linear fractional transformations. *Systems & Control Letters*, 22(2):79 – 92, 1994.

[39] Alan P. Parkes. *Boolean Logic and Propositional Logic*, pages 275–290. Springer London, London, 2002.

[40] Andr Platzer. Keymaera: A hybrid theorem prover for hybrid systems.

[41] Kuldip Rattan and Matthew Clark. Km-logic: A computationally efficient, real-time mimo fuzzy inference system. Technical report, Air Force Research Laboratory, 2015.

[42] Kuldip Rattan and Matthew Clark. Introduction to fuzzy control. 2017.

[43] Kuldip S Rattan. Ee619 course notes. University Lecture, 1997.

[44] Wilson J Rugh and Jeff S Shamma. Research on gain scheduling. *Automatica*, 36(10):1401–1425, 2000.

[45] J. S. Shamma and M. Athans. Analysis of gain scheduled control for nonlinear plants. *IEEE Transactions on Automatic Control*, 35(8):898–907, Aug 1990.

[46] J. S. Shamma and M. Athans. Gain scheduling: potential hazards and possible remedies. *IEEE Control Systems*, 12(3):101–107, June 1992.

[47] Jeff S Shamma. *Analysis and design of gain scheduled control systems*. PhD thesis, Massachusetts Institute of Technology, 1988.

[48] Cory Firmin Snyder. Provable run time safety assurance for a non-linear system, 2013. ID: wright1369443661.

[49] Brian L Stevens, Frank L Lewis, and Eric N Johnson. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[50] Michio Sugeno and GT Kang. Structure identification of fuzzy model. *Fuzzy sets and systems*, 28(1):15–33, 1988.

[51] C. J. Tomlin, I. Mitchell, A. M. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, July 2003.

[52] Wikipedia. Fuzzy logic — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Fuzzy%20logic&oldid=767030743, 2017. [Online; accessed 25-March-2017].

[53] Kevin A. Wise and Eugene Lavretsky. Robust adaptive control for the joint direct attack munition. 2011.

[54] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

[55] L.A Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3 – 28, 1978.

[56] Lofti A Zadeh. A rationale for fuzzy control. *Journal of Dynamic Systems, Measurement, and Control*, 94(1):3–4, 1972.

[57] Lotfi A. Zadeh. Is there a need for fuzzy logic? *Information Sciences*, 178(13):2751 – 2779, 2008.

[58] Z. Y. Zhao, M. Tomizuka, and S. Isaka. Fuzzy gain scheduling of pid controllers. In *[Proceedings 1992] The First IEEE Conference on Control Applications*, pages 698–703 vol.2, Sep 1992.

Appendix A

Calculations and Source Code

A.1 Piecewise Linear Fuzzy Representation

Recall the input fuzzy membership set table 3.3 and output fuzzy membership set table 3.5 defined in chapter 3. Using the established rule set that associates the membership sets (table 3.6), the modes are defined.

$$K_{pe}(j) = \frac{1}{x(j+1) - x(j)}[-1 \quad 1] \begin{bmatrix} u(j) \\ u(j+1) \end{bmatrix} \tag{A.1}$$

$$cons(j) = \frac{1}{x(j+1) - x(j)} \begin{bmatrix} x(j+1) & -x(j) \end{bmatrix} \begin{bmatrix} u(m) \\ u(m+1) \end{bmatrix} \tag{A.2}$$

A.1.1 Mode 1

$$j = 1 \tag{A.3}$$

$$DEN(j) = x(j+1) - x(j) = -0.8 - (-1) = 0.2 \tag{A.4}$$

$$K_{pe}(1) = \frac{1}{DEN(j)} \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -0.024 \\ -0.0192 \end{bmatrix} \tag{A.5}$$

$$= \frac{1}{0.2}(0.024 + (-0.0192)) = \frac{1}{0.2}(0.0048) \tag{A.6}$$

$$= 0.024 \tag{A.7}$$

$$cons(1) = \frac{1}{DEN(j)} \begin{bmatrix} -0.8 & -(-1) \end{bmatrix} \begin{bmatrix} -0.024 \\ -0.0192 \end{bmatrix} \tag{A.8}$$

$$= \frac{1}{0.2}(0.0192 + (-0.0192)) \qquad = 0 \tag{A.9}$$

$$\therefore \tag{A.10}$$

$$u(1) = K_{pe}(1)e + cons(1) \tag{A.11}$$

$$u(1) = 0.024e \tag{A.12}$$

A.1.2 Mode 2

$$j = 2 \tag{A.13}$$

$$DEN(j) = x(j+1) - x(j) = -0.75 - (-0.8) = 0.05 \tag{A.14}$$

$$K_{pe}(2) = \frac{1}{0.05}\begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} -0.024 \\ -0.0192 \end{bmatrix} \tag{A.15}$$

$$= -0.696 \tag{A.16}$$

$$cons(2) = \frac{1}{0.05}\begin{bmatrix} -0.75 & -(-0.8) \end{bmatrix}\begin{bmatrix} -0.0192 \\ -0.054 \end{bmatrix} \tag{A.17}$$

$$= -0.576 \tag{A.18}$$

$$\therefore \tag{A.19}$$

$$u(2) = K_{pe}(2)e + cons(2) \tag{A.20}$$

$$u(2) = -0.696e - 0.576 \tag{A.21}$$

A.1.3 Mode j

From the previous calculations, the pattern to calculate the u(j) modes is seen. Therefore, the author constructed a Matlab script to calculate the entire surface using the piecewise linear method.

```
1 %% Calculate the Piecewise Linear Representations of the Example ...
     PCtrl
2 % Date: 20181215
3 % Calculating by using
4 % - The defined Input Fuzzy Membership Set (ifms)
5 % - The rule-ordered (not magnitude ordered) Output Fuzzy ...
     Membership Set
```

```
% (ofms)
% - Removed the zero since it passes through the origin
% - KpeCSet is the effective gain (Kpe) and the Constant (C) for ...
    all modes
% --> Similar to slope-intercept form
% --> Each row index denotes a mode

ifms = [-1 -0.8 -0.75 -0.2 -0.15 0.15 0.2 0.75 0.8 1];
ofms = [-0.024 -0.0192 -0.054 -0.0144 -0.15 0.15 0.0144 0.054 ...
    0.0192 0.024];
% KpeCSet = [];
KpeCset = zeros((length(ifms)-1), 2);


for j=1:(length(ifms)-1)
    den = ifms(j+1) - ifms(j);
    KpeCset(j,1) = (1/den)*[-1 1]*[ofms(j) ofms(j+1)]';
    KpeCset(j,2) = (1/den)*[ifms(j+1) -ifms(j)]*[ofms(j) ofms(j+1)]';
%     Kpe = (1/den)*[-1 1]*[ofms(j) ofms(j+1)]';
%     cons = (1/den)*[ifms(j+1) -ifms(j)]*[ofms(j) ofms(j+1)]';
%     KpeCSet = [KpeCSet; [Kpe cons]];
end

KpeCset
KpeCset_50 = 50 * KpeCset
```

A.2 Proportional Fuzzy Control Using Simulink

```
function [ y ] = PFLC(x, centers, rulevector)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

        out = [];
```

```
        out = fuzzify(x, centers);
        y = out*rulevector';
end
```

```
%-------------------------------------------------------------------------
% m-file to determine kpe(j) and const(j) for a PFLC
%-------------------------------------------------------------------------
%
%-------------------------------------------------------------------------
% input: center points of input fuzzy set (centers)
% For example for 7 input fuzzy sets
%-------------------------------------------------------------------------
%centers = [-1 0 1];
centers = [-1 -.8 -.75 -.7 -.65 -.3 -.25 -.2 -.15 0 .15 .2 .25 .3 ...
    .65 .7 .75 .8 1];
%%-------------------------------------------------------------------------
% input: rule_vector containing the center points of the output ...
    fuzzy sets
%-------------------------------------------------------------------------
%rule_vector = [-1  0  1];
rule_vector = [-.12 -.096 -.09 -.252 -.234 -.108 -.09 -.384 -.288 ...
     0 ...
              .288 .384 .09 .108 .234 .252 .09 .096 .12];
% rule_vector = [-1 -.8 -.75  -1.05  -.975  -.45 -.375  -.6 -.45 ...
    0 ...
%                .45 .6 .375 .45 .975 1.05 0.75 .8 1];
e = centers;
U = rule_vector;
```

```
%Fuzzify.m %
%Sameep Singh %
% EE 619 Lab 2%

```

```
function [y]=fuzzify(x,centers)
%returns length of vectox 'centers')
d = length(centers);
%'ZEROS'  Zeros all the arrays
y=zeros(1,d);
%test for extremes of Universe of discourse and for values within ...
    universe of discourse'
if x ≤ centers(1)
   y(1) = 1;
elseif x ≥ centers(d)
   y(d)=1;
else
   for i=1:d
      if (x≥centers(i)) & (x ≤ centers(i+1));
         y(i)=(centers(i+1)-x)/(centers(i+1)-centers(i));
         y(i+1)=(x-centers(i))/(centers(i+1)-centers(i));
      end;
   end;
end;
```

```
%% Plot for PFLC Surface and Gain Schedule graph
% Plot for the running thesis example
% Plot thesis Example with the plant: 75/(s^2 + 20s + 75)

% Kp = [1, 2, 12.5, 2, 1]

centers = [-1 -0.51 -0.5 -0.09 -0.08 0.08 0.09 0.5 0.51 1.0];
gains = [1 2 12.5];
kps = gains;


%% PFLC Surface Plot
% Run the initial section to execute

```

```
mbVect = [];

% Draw the three plot lines
fullRng = -1:0.01:1;
kpLow = kps(1)*fullRng;
kpMid = kps(2)*fullRng;
kpHgh = kps(3)*fullRng;
figure(1)
plot(fullRng,kpLow,':k',fullRng,kpMid,':k',fullRng,kpHgh,':k','HandleVisibility','
hold on

% Surface line for NB
xNB = -1:0.01:-0.51;
yNB = kps(1)*xNB;
mbVect = [mbVect; [kps(1) 0]];

% Surface line for NMtoNB
xNMtNB = -0.51:0.001:-0.5;
xPt1 = -0.51;
xPt2 = -0.5;
eqNMtNB = polyfit([xPt1,xPt2],[(kps(1)*xPt1),(kps(2)*xPt2)],1);
yNMtNB = eqNMtNB(1)*xNMtNB + eqNMtNB(2);
mbVect = [mbVect; eqNMtNB];

% Surface line for NM
xNM = -0.5:0.001:-0.09;
yNM = kps(2)*xNM;
mbVect = [mbVect; [kps(2) 0]];

% Surface line for NM to NS
xNStNM = -0.09:0.001:-0.08;
xPt1 = -0.09;
xPt2 = -0.08;
eqNStNM = polyfit([xPt1,xPt2],[(kps(2)*xPt1),(kps(3)*xPt2)],1);
yNStNM = eqNStNM(1)*xNStNM + eqNStNM(2);
mbVect = [mbVect; eqNStNM];
```

```
51
52 % Surface line for NSPS
53 xNSPS = -0.08:0.001:0.08;
54 yNSPS = kps(3)*xNSPS;
55 mbVect = [mbVect; [kps(3) 0]];
56
57 % Surface line for PS to PM
58 xPStPM = 0.08:0.001:0.09;
59 xPt1 = 0.08;
60 xPt2 = 0.09;
61 eqPStPM = polyfit([xPt1,xPt2],[(kps(3)*xPt1),(kps(2)*xPt2)],1);
62 yPStPM = eqPStPM(1)*xPStPM + eqPStPM(2);
63 mbVect = [mbVect; eqPStPM];
64
65 % Surface line for PM
66 xPM = 0.09:0.001:0.5;
67 yPM = kps(2)*xPM;
68 mbVect = [mbVect; [kps(2) 0]];
69
70 % Surface line for PMtoPB
71 xPMtPB = 0.5:0.001:0.51;
72 xPt1 = 0.5;
73 xPt2 = 0.51;
74 eqPMtPB = polyfit([xPt1,xPt2],[(kps(2)*xPt1),(kps(1)*xPt2)],1);
75 yPMtPB = eqPMtPB(1)*xPMtPB + eqPMtPB(2);
76 mbVect = [mbVect; eqPMtPB];
77
78 % Surface line for PB
79 xPB = 0.51:0.01:1;
80 yPB = kps(1)*xPB;
81 mbVect = [mbVect; [kps(1) 0]];
82
83 % x = [xL x1 xR];
84 % y = [yL y1 yR];
85
86 % figure(2)
```

```
plot(xNB,yNB,xNMtNB,yNMtNB,xNM,yNM,xNStNM,yNStNM,xNSPS,yNSPS,...
     xPStPM,yPStPM,xPM,yPM,xPMtPB,yPMtPB,xPB,yPB,'LineWidth',2);

%
% plot(x,y,'linewidth',2);
ax = gca;
ax.Box = 'off';
ax.XAxisLocation = 'origin';
xlabel('$e$','Interpreter','latex','FontSize',12);
% xticks(centers);
xticks([-1 -0.5 -0.08 0.08 0.5 1]);
ax.XLim = [-1.1 1.1];
ax.YAxisLocation = 'origin';
ylabel('$u$','Interpreter','latex','FontSize',12);
% ylabel('$K_{p}$','Interpreter','latex','FontSize',12);
% ax.YLim = [-0.1 0.1];
yticks([-1 -0.51 -0.18 0.18 0.51 1]);
ax.YLim = [-1.2 1.2];

title('Proportional Gain Surface Plot')
% legend('NB','NMtNB','NM','NStNM','NSPS','PStPM','PM','PMtPB','PB')
legend({'Mode 1','Mode 2','Mode 3','Mode 4','Mode 5','Mode ...
    6','Mode 7','Mode 8','Mode 9'},'Location','northwest')
hold off

% % Output the normalized slope/intercept vector
diary('./PCtrl_mbVect.txt')
diary ON
%
gains
kps
mbVect
%
% % Re-multiple by 12.5
% % mbVect * 12.5
diary OFF
```

```
% Plot dashes for sets
% -- TODO --

fig = gcf;
fig.PaperUnits = 'inches';
fig.PaperPosition = [0 0 10 5];
print('./PCtrl_GainSurfacePlot','-dpng','-r300')  % Save figure ...
    as png

set(gcf, 'PaperPosition', [0 0 16 10]); %Position plot at left ...
    hand corner with width 5 and height 5.
set(gcf, 'PaperSize', [16 10]); %Set the paper to have width 5 ...
    and height 5.
saveas(gcf, './PFLC_Ctrl_Surface', 'pdf') %Save figure as pdf

%% Plot for Proportional Gain Schedule interpolated values.
mbVectGS = [];

% Surface line for NB
xNB = -1:0.01:-0.51;
yNB = kps(1)*ones(size(xNB));
mbVectGS = [mbVectGS; [0 kps(1)]];

% Surface line for NMtoNB
xNMtNB = -0.51:0.001:-0.5;
xPt1 = -0.51;
xPt2 = -0.5;
eqNMtNB = polyfit([xPt1,xPt2],[kps(1),kps(2)],1);
yNMtNB = eqNMtNB(1)*xNMtNB + eqNMtNB(2);
mbVectGS = [mbVectGS; eqNMtNB];

% Surface line for NM
xNM = -0.5:0.001:-0.09;
yNM = kps(2)*ones(size(xNM));
mbVectGS = [mbVectGS; [0 kps(2)]];

```

```
% Surface line for NM to NS
xNStNM = -0.09:0.001:-0.08;
xPt1 = -0.09;
xPt2 = -0.08;
eqNStNM = polyfit([xPt1,xPt2],[kps(2),kps(3)],1);
yNStNM = eqNStNM(1)*xNStNM + eqNStNM(2);
mbVectGS = [mbVectGS; eqNStNM];

% Surface line for NSPS
xNSPS = -0.08:0.001:0.08;
yNSPS = kps(3)*ones(size(xNSPS));
mbVectGS = [mbVectGS; [0 kps(3)]];

% Surface line for PS to PM
xPStPM = 0.08:0.001:0.09;
xPt1 = 0.08;
xPt2 = 0.09;
eqPStPM = polyfit([xPt1,xPt2],[kps(3),kps(2)],1);
yPStPM = eqPStPM(1)*xPStPM + eqPStPM(2);
mbVectGS = [mbVectGS; eqPStPM];

% Surface line for PM
xPM = 0.09:0.001:0.5;
yPM = kps(2)*ones(size(xPM));
mbVectGS = [mbVectGS; [0 kps(2)]];

% Surface line for PMtoPB
xPMtPB = 0.5:0.001:0.51;
xPt1 = 0.5;
xPt2 = 0.51;
eqPMtPB = polyfit([xPt1,xPt2],[kps(2),kps(1)],1);
yPMtPB = eqPMtPB(1)*xPMtPB + eqPMtPB(2);
mbVectGS = [mbVectGS; eqPMtPB];

% Surface line for PB
xPB = 0.51:0.01:1;
```

```
yPB = kps(1)*ones(size(xPB));
mbVectGS = [mbVectGS; [0 kps(1)]];

% x = [xL x1 xR];
% y = [yL y1 yR];

figure(2)
plot(xNB,yNB,xNMtNB,yNMtNB,xNM,yNM,xNStNM,yNStNM,xNSPS,yNSPS,...
     xPStPM,yPStPM,xPM,yPM,xPMtPB,yPMtPB,xPB,yPB,'LineWidth',2);

%
% plot(x,y,'linewidth',2);
ax = gca;
ax.Box = 'off';
ax.XAxisLocation = 'origin';
xlabel('$e$','Interpreter','latex','FontSize',12);
% xticks(centers);
xticks([-1 -0.5 -0.08 0.08 0.5 1]);
ax.XLim = [-1.1 1.1];
ax.YAxisLocation = 'origin';
ylabel('$K_p$','Interpreter','latex','FontSize',12);

% ax.YLim = [-0.1 0.1];
yticks([1 2 12.5]);
ax.YLim = [-1 14.0];

title('Proportional Gain Schedule Linear Interpolation Plot')
% legend('NB','NMtNB','NM','NStNM','NSPS','PStPM','PM','PMtPB','PB')
legend({'Mode 1','Mode 2','Mode 3','Mode 4','Mode 5','Mode ...
    6','Mode 7','Mode 8','Mode 9'},'Location','northwest')
hold off

% % Output the normalized slope/intercept vector
diary('./PCtrl_mbVectGS.txt')
diary ON
%
```

```
226 gains
227 kps
228 mbVectGS
229 %
230 % % Re-multiple by 12.5
231 % % mbVect * 12.5
232 diary OFF
233 % Plot dashes for sets
234 % -- TODO --
235
236
237 %print -r500
238 fig = gcf;
239 fig.PaperUnits = 'inches';
240 fig.PaperPosition = [0 0 8 3];
241 print('./PCtrl_GainScheduleInterpPlot','-dpng','-r300')  % Save ...
      figure as png
242
243 set(gcf, 'PaperPosition', [0 0 16 10]); %Position plot at left ...
      hand corner with width 5 and height 5.
244 set(gcf, 'PaperSize', [16 10]); %Set the paper to have width 5 ...
      and height 5.
245 saveas(gcf, './PCtrl_GainScheduleInterp', 'pdf') %Save figure as pdf
```

A.3 Proportional Fuzzy Control Using ODE45

```
1 %% Solve the GS_PCtrl Example with ODE45
2 % Date: 20190102
3 % This is to prove the system dynamics are properly maintained
4
5
6 initCond = [0 0];
7 tspan = [0,1];
```

```
8
9  rng = [-1 -0.51 -0.5 -0.09 -0.08 0.08 0.09 0.5 0.51 1];
10 kpGs = [1 1 2 2 12.5 12.5 2 2 1 1];
11
12 mbVect = [1 0; -49 -25.5; 2 0; -82 -7.56; 12.5 0; -82 7.56; 2 0; ...
       -49 25.5; 1 0];
13
14 % Set the initial condition on x
15 x = 0;
16
17
18 [t,x] = ode45(@PCtrlGS_v2,tspan,initCond,x,rng,kpGs);
19 figure(5)
20 plot(t,x(:,1),'r-')
21 grid
22
23 title('Response Plot Comparisons')
24 xlabel('time (s)')
25 ylabel('response (y)')
26 legend('GS PCtrl (ODE45)')
27
28 print('./GS_PCtrl_ODE_Plot','-dpng')
29
30 %% Get comparison plot
31 % Run the simulink (GSP_PFLC_PselDFLC_Ctrl_Compare.slx), will ...
       populate
32 % simout_FLC
33 % Plot data set 1 (GS Interpolated) and set 2 (Fuzzy PFLC)
34
35 hold on
36 plot(simout_FLC.Time,simout_FLC.Data(:,1),'bx',simout_FLC.Time,simout_FLC.Data(:,2
37 legend('GS PCtrl (ODE45)','Kp Interpolated','Fuzzy P ...
       Control','location','southeast')
38
39 print('./GS_PCtrl_ODE_Interp_PFLC_Plot','-dpng')
```

```
function xdot = PCtrlGS_v2(t,x,rng,kpGs)
%PCtrlGS ODE45 calculation function
%   This function utilizes ODE45 to determine the response of a ...
    control
%   system based on a scheduled gain. The schedule gain is set ...
    through the
%   parameter u and where the error is associated with the -1 to ...
    1 range.

    e = 1 - x(1); % target at 1, change if not the target
    static_gain = 5;

%     u = e * interp1(rng,kpGs,e)
%     u = u*static_gain;

    row = 1; % Temp starting value

    mbVect = [1 0; -49 -25.5; 2 0; -82 -7.56; 12.5 0; -82 7.56; 2 ...
        0; -49 25.5; 1 0];

    for lcv = 2:length(rng)-1   % will explicitly test extremes
        if e < rng(2)   % low end
            row = 1;
            break;
        elseif e > rng(length(rng)-1)   % high end
            row = length(rng)-1;
            break;
        elseif e ≥ rng(length(rng)/2) && e ≤ ...
            rng((length(rng)/2)+1) % middle
            row = (length(rng)/2);
            break;
        elseif e < 0 && e ≥ rng(lcv) && e < rng(lcv+1) % negative ...
            side
            row = lcv;
            break;
```

```
30          elseif e ≥ 0 && e > rng(lcv) && e ≤ rng(lcv+1) % positive ...
                side
31              row = lcv;
32              break;
33          else
34              continue;
35          end
36      end
37
38      u = mbVect(row,1)*e + mbVect(row,2);
39
40      xdot = zeros(size(x));
41      xdot(1) = x(2);
42      xdot(2) = -20*x(2) - 75*x(1) + static_gain*75*u;
43
44  end
```

A.4 Proportional Single Mode Hybrid Automaton

```
1   function xdot = PCtrlGSRefine(t,x,rng,kpGs)
2   %PCtrlGS ODE45 calculation function
3   %   This function utilizes ODE45 to determine the response of a ...
        control
4   %   system based on a scheduled gain. The schedule gain is set ...
        through the
5   %   parameter u and where the error is associated with the -1 to ...
        1 range.
6
7       e = 1 - x(1); % target at 1, change if not the target
8
9       u = interp1(rng,kpGs,e);
10
11      xdot = zeros(size(x));
```

```
    xdot(1) = x(2);
    xdot(2) = -3.6*x(2) - 10*x(1) +10*u;

end
```

A.5 KMLOGIC Material

```
 function g = KMLOGIC(n,input,Test1,R,MFM)
%%
%  Find the two adjacent membership fvalues [y1 y2]
% and the index value (j) of the first active membership function
  [y1, y2, j]=Rec_Fuzzify(input(n),1,length(Test1{n}),Test1{n});
%%
% Find the grouping of rules needed for the next input of n inputs
  sindex1 = ((j-1)*MFM)+1;
  sindex2 = (sindex1-1)+MFM;
  eindex = sindex2+MFM;
  R1 = R(sindex1:sindex2);
  R2 = R(sindex2+1:eindex);

%%
  if n > 1
    MFM2=MFM/(length(Test1{n-1}));
    g=[y1 ...
        y2]*[KMLOGIC(n-1,input,Test1,R1,MFM2);KMLOGIC(n-1,input,Test1,R2,MFM2)];
  else
    g = [y1 y2]*[R1;R2];
  end
```

```
function out = pdflcnew(Input1,Input2)
% centers=[-1.0 -0.51 -0.50 -0.09 -0.08 0.0 0.08 0.09 0.50 0.51 1.0;
```

```
3  %          -1.0 -0.75 -0.50 -.375 -0.25 0.0 0.25 .375 0.50 0.75 1.0];
4
5  centers1 = [-1.0 -0.51 -0.50 -0.09 -0.08 0.0 0.08 0.09 0.50 0.51 ...
       1.0];
6  centers2 = [-1 0 1];
7
8  Test1{1} = centers1;
9  Test1{2} = centers2;
10
11 % R=[-1.00 -0.51 -1.00 -0.18 -2.00  -1.00  0.00  0.18 1.00 0.51 ...
       1.00 ...
12 %    -1.00 -0.51 -1.00 -0.18 -1.75  -0.75  0.25  0.18 1.00 0.51 ...
       1.00 ...
13 %    -1.00 -0.51 -1.00 -0.18 -1.50  -0.50  0.50  0.18 1.00 0.51 ...
       1.00 ...
14 %    -1.00 -0.51 -1.00 -0.18 -1.375 -.375  .625  0.18 1.00 0.51 ...
       1.00 ...
15 %    -1.00 -0.51 -1.00 -0.18 -1.25  -0.25  0.75  0.18 1.00 0.51 ...
       1.00 ...
16 %    -1.00 -0.51 -1.00 -0.18 -1.00   0.00  1.00  0.18 1.00 0.51 ...
       1.00 ...
17 %    -1.00 -0.51 -1.00 -0.18 -0.75   0.25  1.25  0.18 1.00 0.51 ...
       1.00 ...
18 %    -1.00 -0.51 -1.00 -0.18 -0.625  .375 1.375  0.18 1.00 0.51 ...
       1.00 ...
19 %    -1.00 -0.51 -1.00 -0.18 -0.50   0.50  1.50  0.18 1.00 0.51 ...
       1.00 ...
20 %    -1.00 -0.51 -1.00 -0.18 -0.25   0.75  1.75  0.18 1.00 0.51 ...
       1.00 ...
21 %    -1.00 -0.51 -1.00 -0.18  0.00   1.00  2.00  0.18 1.00 0.51 ...
       1.00];
22 %%
23 R = [-1.00 -0.51 -1.00 -0.18 -2.00  -1.00  0.00  0.18 1.00 0.51 ...
       1.00 ...
24       -1.00 -0.51 -1.00 -0.18 -1.00   0.00  1.00  0.18 1.00 0.51 ...
             1.00 ...
```

```
25      -1.00 -0.51 -1.00 -0.18  0.00   1.00  2.00  0.18 1.00 0.51 ...
            1.00];
26  %%
27  % Find the total number of fuzzy sets in all inputs except the last
28  n = 2;
29  MFM=1;
30  for i = 1:n-1
31  %    MFM=MFM*length(centers(i,:));
32      MFM=MFM*length(Test1{i});
33  end
34  %%
35  input = [Input1, Input2];
36  out = KMLOGIC(n,input,Test1,R,MFM);
```

```
1   %Fuzzify.m %
2   %
3   % ...
        *--------------------------------------------------------------------- ...
        *
4   % Copyright Kuldip S Rattan and Matthew Clark
5   % ...
        *--------------------------------------------------------------------- ...
        *
6   % ...
        *--------------------------------------------------------------------- ...
        *
7   % This m-file computes the membership values of the fuzzy sets ...
        (defined
8   % over a normalized universe of discourse between -1 and 1) for a ...
        given
9   % input . centers is a vector containing the center points or ...
        peak points
10  % of the triagular fuzzy sets whose membership values are being ...
        determined
```

```
% for the given input x,  Fuzzy partitioing is assumed.  y is the ...
    output
% vector. It is also assumed that the membership values for ...
    inputs greater
% than and less than -1 is 1 is [0 1] or [1 0], respectively..
% ...
    *------------------------------------------------------------------ ...
    *
%
function [y1, y2, j]=Rec_Fuzzify(x,i,d,centers)
if (i==1) && (x < centers(1));
   y1 = 1;
   y2 = 0;
   j = 1;
elseif (i==d) && (x > centers(d));
   y1 = 0;
   y2 = 1;
   j = i-1;
elseif (x≥centers(i)) && (x ≤ centers(i+1));
   y1=(centers(i+1)-x)/(centers(i+1)-centers(i));
   y2=(x-centers(i))/(centers(i+1)-centers(i));
   j=i;
else
   [y1, y2, j]=Rec_Fuzzify(x,i+1,d,centers);
end;
```

A.6 PFLC Python Response Modeling

```
'''
Created by Hyst v1.4
Hybrid Automaton in PySim
Converted from file: /home/verivital/Desktop/fuzzyP_v2/prop_p.xml
```

```
5  Command Line arguments: -tool pysim "" -output ...
       /home/verivital/Desktop/fuzzyP_v2/python/prop_p_pysim.py ...
       -input /home/verivital/Desktop/fuzzyP_v2/prop_p.xml ...
       /home/verivital/Desktop/fuzzyP_v2/prop_p.cfg
6  '''
7
8  import hybridpy.pysim.simulate as sim
9  from hybridpy.pysim.simulate import init_list_to_q_list, ...
       PySimSettings
10 from hybridpy.pysim.hybrid_automaton import HybridAutomaton, ...
       HyperRectangle
11
12 def define_ha():
13     '''make the hybrid automaton and return it'''
14
15     ha = HybridAutomaton()
16     ha.variables = ["x1", "x2", "t"]
17
18
19     loc2 = ha.new_mode('loc2')
20     loc2.inv = lambda state: 1.0 - state[0] > -0.51 and 1.0 - ...
           state[0] ≤ -0.5
21     loc2.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] - 18375.0 * (1.0 - state[0]) - 9562.5, 1.0]
22     loc2.der_interval_list = [[0, 0], [0, 0], [0, 0]]
23
24     loc3 = ha.new_mode('loc3')
25     loc3.inv = lambda state: 1.0 - state[0] > -0.5 and 1.0 - ...
           state[0] ≤ -0.09
26     loc3.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 750.0 * (1.0 - state[0]), 1.0]
27     loc3.der_interval_list = [[0, 0], [0, 0], [0, 0]]
28
29     loc4 = ha.new_mode('loc4')
30     loc4.inv = lambda state: 1.0 - state[0] > -0.09 and 1.0 - ...
           state[0] ≤ -0.08
```

```
31      loc4.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] - 30750.0 * (1.0 - state[0]) - 2835.0, 1.0]
32      loc4.der_interval_list = [[0, 0], [0, 0], [0, 0]]
33
34      loc1 = ha.new_mode('loc1')
35      loc1.inv = lambda state: 1.0 - state[0] ≤ -0.51
36      loc1.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 375.0 * (1.0 - state[0]), 1.0]
37      loc1.der_interval_list = [[0, 0], [0, 0], [0, 0]]
38
39      loc5 = ha.new_mode('loc5')
40      loc5.inv = lambda state: 1.0 - state[0] > -0.08 and 1.0 - ...
           state[0] < 0.08
41      loc5.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 4687.5 * (1.0 - state[0]), 1.0]
42      loc5.der_interval_list = [[0, 0], [0, 0], [0, 0]]
43
44      loc6 = ha.new_mode('loc6')
45      loc6.inv = lambda state: 1.0 - state[0] ≥ 0.08 and 1.0 - ...
           state[0] < 0.09
46      loc6.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] - 30750.0 * (1.0 - state[0]) + 2835.0, 1.0]
47      loc6.der_interval_list = [[0, 0], [0, 0], [0, 0]]
48
49      loc7 = ha.new_mode('loc7')
50      loc7.inv = lambda state: 1.0 - state[0] ≥ 0.09 and 1.0 - ...
           state[0] < 0.5
51      loc7.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 750.0 * (1.0 - state[0]), 1.0]
52      loc7.der_interval_list = [[0, 0], [0, 0], [0, 0]]
53
54      loc8 = ha.new_mode('loc8')
55      loc8.inv = lambda state: 1.0 - state[0] ≥ 0.5 and 1.0 - ...
           state[0] < 0.51
56      loc8.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] - 18375.0 * (1.0 - state[0]) + 9562.5, 1.0]
```

```
57      loc8.der_interval_list = [[0, 0], [0, 0], [0, 0]]
58
59      loc9 = ha.new_mode('loc9')
60      loc9.inv = lambda state: 1.0 - state[0] ≥ 0.51
61      loc9.der = lambda _, state: [state[1], -75.0 * state[0] - ...
            20.0 * state[1] + 375.0 * (1.0 - state[0]), 1.0]
62      loc9.der_interval_list = [[0, 0], [0, 0], [0, 0]]
63
64      t = ha.new_transition(loc2, loc1)
65      t.guard = lambda state: 1.0 - state[0] ≤ -0.51
66      t.reset = lambda state: [None, None, None]
67
68      t = ha.new_transition(loc2, loc3)
69      t.guard = lambda state: 1.0 - state[0] > -0.5
70      t.reset = lambda state: [None, None, None]
71
72      t = ha.new_transition(loc3, loc4)
73      t.guard = lambda state: 1.0 - state[0] > -0.09
74      t.reset = lambda state: [None, None, None]
75
76      t = ha.new_transition(loc3, loc2)
77      t.guard = lambda state: 1.0 - state[0] ≤ -0.5
78      t.reset = lambda state: [None, None, None]
79
80      t = ha.new_transition(loc4, loc5)
81      t.guard = lambda state: 1.0 - state[0] > -0.08
82      t.reset = lambda state: [None, None, None]
83
84      t = ha.new_transition(loc4, loc3)
85      t.guard = lambda state: 1.0 - state[0] ≤ -0.09
86      t.reset = lambda state: [None, None, None]
87
88      t = ha.new_transition(loc1, loc2)
89      t.guard = lambda state: 1.0 - state[0] > -0.51
90      t.reset = lambda state: [None, None, None]
91
```

```
92      t = ha.new_transition(loc5, loc6)
93      t.guard = lambda state: 1.0 - state[0] ≥ 0.0801
94      t.reset = lambda state: [None, None, None]
95
96      t = ha.new_transition(loc5, loc4)
97      t.guard = lambda state: 1.0 - state[0] ≤ -0.0801
98      t.reset = lambda state: [None, None, None]
99
100     t = ha.new_transition(loc6, loc7)
101     t.guard = lambda state: 1.0 - state[0] ≥ 0.09
102     t.reset = lambda state: [None, None, None]
103
104     t = ha.new_transition(loc6, loc5)
105     t.guard = lambda state: 1.0 - state[0] < 0.08
106     t.reset = lambda state: [None, None, None]
107
108     t = ha.new_transition(loc7, loc8)
109     t.guard = lambda state: 1.0 - state[0] ≥ 0.5
110     t.reset = lambda state: [None, None, None]
111
112     t = ha.new_transition(loc7, loc6)
113     t.guard = lambda state: 1.0 - state[0] < 0.09
114     t.reset = lambda state: [None, None, None]
115
116     t = ha.new_transition(loc8, loc9)
117     t.guard = lambda state: 1.0 - state[0] ≥ 0.51
118     t.reset = lambda state: [None, None, None]
119
120     t = ha.new_transition(loc8, loc7)
121     t.guard = lambda state: 1.0 - state[0] < 0.5
122     t.reset = lambda state: [None, None, None]
123
124     t = ha.new_transition(loc9, loc8)
125     t.guard = lambda state: 1.0 - state[0] < 0.51
126     t.reset = lambda state: [None, None, None]
127
```

```
128     return ha
129
130 def define_init_states(ha):
131     '''returns a list of (mode, HyperRectangle)'''
132     # Variable ordering: [x1, x2, t]
133     rv = []
134
135     rv.append((ha.modes['loc2'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
136     rv.append((ha.modes['loc3'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
137     rv.append((ha.modes['loc4'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
138     rv.append((ha.modes['loc1'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
139     rv.append((ha.modes['loc5'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
140     rv.append((ha.modes['loc6'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
141     rv.append((ha.modes['loc7'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
142     rv.append((ha.modes['loc8'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
143     rv.append((ha.modes['loc9'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
144     return rv
145
146
147 def define_settings():
148     '''defines the automaton / plot settings'''
149     s = PySimSettings()
150     s.max_time = 0.5
151     s.step = 0.001
152     s.dim_x = 2
153     s.dim_y = 0
154
```

```
155     return s
156
157 def simulate(init_states, settings):
158     '''simulate the automaton from each initial rect'''
159
160     q_list = init_list_to_q_list(init_states, center=True, ...
            star=True, corners=False, rand=0)
161     result = sim.simulate_multi(q_list, settings.max_time)
162
163     return result
164
165 def plot(result, init_states, image_path, settings):
166     '''plot a simulation result to a file'''
167
168     draw_events = len(result) == 1
169     shouldShow = False
170     sim.plot_sim_result_multi(result, settings.dim_x, ...
            settings.dim_y, image_path, draw_events, legend=True, ...
            title='Simulation', show=shouldShow, init_states=init_states)
171
172 if __name__ == '__main__':
173     ha = define_ha()
174     settings = define_settings()
175     init_states = define_init_states(ha)
176     plot(simulate(init_states, settings), init_states, ...
            'plot.png', settings)
```

A.7 PFLC Zero Input Hybrid Representations

```
1 system = state_diagram
2
3 initially = "-1.0 ≤ x1 ≤ 1.0 & x2 == 0 & t == 0"
4
```

```
set-aggregation = chull
scenario = stc
directions = box
sampling-time = 0.01
time-horizon = 0.5
#iter-max = 15
#iter-max = 30
output-variables = "t, x1"
output-format = GEN
rel-err = 1.0E-6
abs-err = 1.0E-5
flowpipe-tolerance = 0.01

# New attempt 20190103 - prop_p_zi_spx_im09.png
# - If don't want reactivate the above fields
#time-horizon = 0.5
#iter-max = 9
#flowpipe-tolerance = 0.01

# New attempt 20190103 - prop_p_zi_spx_im18.png
# - If don't want reactivate the above fields
#time-horizon = 0.5
#iter-max = 18
#flowpipe-tolerance = 0.01

# New attempt 20190103 - prop_p_zi_spx_im27.png
# - If don't want reactivate the above fields
#time-horizon = 0.5
#iter-max = 27
#flowpipe-tolerance = 0.01

# New attempt 20190103 - prop_p_zi_spx_im36.png
# - If don't want reactivate the above fields
#iter-max = 36

# New attempt 20190103 - prop_p_zi_spx_im45.png
```

```
# - If don't want reactivate the above fields
#iter-max = 45
# - Too large - Errored on the 42 iteration

# New attempt 20190103 - prop_p_zi_spx_im41.png
# - If don't want reactivate the above fields
iter-max = 41
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<sspaceex ...
    xmlns="http://www-verimag.imag.fr/xml-namespaces/sspaceex" ...
    version="0.2" math="SpaceEx">
  <component id="state_diagram">
    <param name="x1" type="real" local="false" d1="1" d2="1" ...
       dynamics="any" />
    <param name="x2" type="real" local="false" d1="1" d2="1" ...
       dynamics="any" />
    <param name="t" type="real" local="false" d1="1" d2="1" ...
       dynamics="any" />
    <location id="2" name="loc2" x="478.0" y="136.0" ...
       width="294.0" height="110.0">
      <note>75</note>
      <invariant>(0-x1) &gt;-0.51&amp; (0-x1)&lt;=-0.5</invariant>
      <flow>x1'== x2&amp;
x2'== -75*x1-20*x2-375*49*(0-x1)-375*25.5&amp;
t'== 1</flow>
    </location>
    <location id="3" name="loc3" x="803.0" y="208.0" ...
       width="234.0" height="94.0">
      <note>75</note>
      <invariant>(0-x1) &gt;-0.5 &amp; (0-x1) &lt;=-0.09</invariant>
      <flow>x1'== x2&amp;
x2'== -75*x1-20*x2+375*2*(0-x1)&amp;
t'==1</flow>
```

```
20      </location>
21      <location id="4" name="loc4" x="1123.0" y="108.0" ...
            width="374.0" height="98.0">
22        <note>75</note>
23        <invariant>(0-x1) &gt;-0.09 &amp; (0-x1) &lt;=-0.08</invariant>
24        <flow>x1'== x2&amp;
25  x2'== -75*x1-20*x2-375*82*(0-x1)-375*7.56&amp;
26  t'==1</flow>
27      </location>
28      <location id="6" name="loc1" x="141.0" y="231.0" ...
            width="220.0" height="98.0">
29        <note>5555</note>
30        <invariant>(0-x1) &lt;= -0.51</invariant>
31        <flow>x1'== x2&amp;
32  x2'== -75*x1-20*x2+375*(0-x1)&amp;
33  t'== 1</flow>
34      </location>
35      <location id="1" name="loc5" x="1151.0" y="354.0" ...
            width="336.0" height="98.0">
36        <note>75</note>
37        <invariant>(0-x1) &gt;-0.08 &amp; (0-x1)&lt;0.08</invariant>
38        <flow>x1'== x2&amp;
39  x2'== -75*x1-20*x2+375*12.5*(0-x1)&amp;
40  t'== 1</flow>
41      </location>
42      <location id="5" name="loc6" x="1149.0" y="586.0" ...
            width="384.0" height="94.0">
43        <note>75</note>
44        <invariant>(0-x1)&gt;= 0.08 &amp; (0-x1)&lt;0.09</invariant>
45        <flow>x1'== x2&amp;
46  x2'== -75*x1-20*x2-375*82*(0-x1)+375*7.56&amp;
47  t'== 1</flow>
48      </location>
49      <location id="7" name="loc7" x="798.0" y="478.0" ...
            width="258.0" height="108.0">
50        <note>75</note>
```

```
51        <invariant>(0-x1)&gt;= 0.09 &amp; (0-x1)&lt;0.5</invariant>
52        <flow>x1'== x2&amp;
53  x2'== -75*x1-20*x2+375*2*(0-x1)&amp;
54  t'== 1</flow>
55      </location>
56      <location id="8" name="loc8" x="452.0" y="588.0" ...
            width="296.0" height="92.0">
57        <note>75</note>
58        <invariant>(0-x1)&gt;= 0.5&amp;(0-x1)&lt;0.51</invariant>
59        <flow>x1'== x2&amp;
60  x2'== -75*x1-20*x2-375*49*(0-x1)+375*25.5&amp;
61  t'== 1</flow>
62      </location>
63      <location id="9" name="loc9" x="144.0" y="410.0" ...
            width="228.0" height="98.0">
64        <note>5555</note>
65        <invariant>(0-x1)&gt;=0.51</invariant>
66        <flow>x1'== x2&amp;
67  x2'== -75*x1-20*x2+375*(0-x1)&amp;
68  t'== 1</flow>
69      </location>
70      <transition source="6" target="2">
71        <guard>(0-x1)&gt;-0.51</guard>
72        <labelposition x="-98.0" y="-69.0" width="116.0" ...
              height="64.0" />
73        <middlepoint x="284.0" y="152.0" />
74      </transition>
75      <transition source="2" target="6">
76        <guard>(0-x1)&lt;=-0.51</guard>
77        <labelposition x="-19.0" y="-2.0" width="108.0" ...
              height="62.0" />
78        <middlepoint x="315.0" y="238.5" />
79      </transition>
80      <transition source="2" target="3">
81        <guard>(0-x1)&gt;-0.5</guard>
```

```
82        <labelposition x="-45.0" y="-81.0" width="124.0" ...
              height="78.0" />
83        <middlepoint x="678.0" y="144.5" />
84      </transition>
85      <transition source="3" target="4">
86        <guard>(0-x1)&amp;gt;-0.09</guard>
87        <labelposition x="-88.0" y="-55.0" width="86.0" ...
              height="50.0" />
88        <middlepoint x="898.5" y="120.0" />
89      </transition>
90      <transition source="4" target="1">
91        <guard>(0-x1) &amp;gt; -0.08</guard>
92        <labelposition x="6.0" y="-36.0" width="100.0" ...
              height="54.0" />
93        <middlepoint x="1176.0" y="262.0" />
94      </transition>
95      <transition source="1" target="5">
96        <guard>(0-x1)&amp;gt;=0.0801</guard>
97        <labelposition x="2.0" y="-35.0" width="104.0" ...
              height="64.0" />
98        <middlepoint x="1186.0" y="480.0" />
99      </transition>
100     <transition source="5" target="7">
101       <guard>(0-x1)&amp;gt;=0.09</guard>
102       <labelposition x="-41.0" y="-43.0" width="98.0" ...
              height="50.0" />
103       <middlepoint x="992.0" y="513.0" />
104     </transition>
105     <transition source="7" target="8">
106       <guard>(0-x1)&amp;gt;=0.5</guard>
107       <labelposition x="-41.0" y="-62.0" width="80.0" ...
              height="60.0" />
108       <middlepoint x="614.5" y="504.0" />
109     </transition>
110     <transition source="8" target="9">
111       <guard>(0-x1)&amp;gt;=0.51</guard>
```

```
112       <labelposition x="-11.0" y="-53.0" width="104.0" ...
              height="50.0" />
113       <middlepoint x="321.0" y="458.0" />
114     </transition>
115     <transition source="3" target="2">
116       <guard>(0-x1)&lt;=-0.5</guard>
117       <labelposition x="-119.0" y="2.0" width="96.0" ...
              height="70.0" />
118       <middlepoint x="630.0" y="232.5" />
119     </transition>
120     <transition source="4" target="3">
121       <guard>(0-x1)&lt;=-0.090</guard>
122       <labelposition x="-55.0" y="-8.0" width="102.0" ...
              height="50.0" />
123       <middlepoint x="992.0" y="190.5" />
124     </transition>
125     <transition source="1" target="4">
126       <guard>(0-x1)&lt;=-0.0801</guard>
127       <labelposition x="-89.0" y="-5.0" width="122.0" ...
              height="54.0" />
128       <middlepoint x="1081.0" y="248.0" />
129     </transition>
130     <transition source="5" target="1">
131       <guard>(0-x1)&lt;0.08</guard>
132       <labelposition x="-71.0" y="-59.0" width="82.0" ...
              height="50.0" />
133       <middlepoint x="1077.5" y="475.5" />
134     </transition>
135     <transition source="7" target="5">
136       <guard>(0-x1)&lt;0.0900</guard>
137       <labelposition x="-93.0" y="9.0" width="102.0" ...
              height="50.0" />
138       <middlepoint x="910.5" y="580.0" />
139     </transition>
140     <transition source="8" target="7">
141       <guard>(0-x1)&lt;0.5</guard>
```

```
      <labelposition x="-37.0" y="5.0" width="96.0" height="52.0" />
      <middlepoint x="642.5" y="598.5" />
    </transition>
    <transition source="9" target="8">
      <guard>(0-x1)&lt;0.51</guard>
      <labelposition x="-51.0" y="3.0" width="118.0" ...
          height="50.0" />
      <middlepoint x="243.0" y="524.0" />
    </transition>
  </component>
</sspaceex>
```

```
'''
Created by Hyst v1.5
Hybrid Automaton in Hylaa2
Converted from file: ...
    /home/awf/Thesis/MastersThesis/linux/WorkingDesktop/fuzzyP_v2/prop_p_zi.xml
Command Line arguments: -tool hylaa2 "" -passes simplify ...
    -python_simplify -output ...
    /home/awf/Thesis/MastersThesis/linux/WorkingDesktop/fuzzyP_v2/prop_p_zi.py ...
    -input ...
    /home/awf/Thesis/MastersThesis/linux/WorkingDesktop/fuzzyP_v2/prop_p_zi.xml ..
    /home/awf/Thesis/MastersThesis/linux/WorkingDesktop/fuzzyP_v2/prop_p_zi.cfg
'''

from hylaa.hybrid_automaton import HybridAutomaton
from hylaa.settings import HylaaSettings, PlotSettings
from hylaa.core import Core
from hylaa.stateset import StateSet
from hylaa import lputil

def define_ha():
    '''make the hybrid automaton and return it'''

```

```
    ha = HybridAutomaton()

    # dynamics variable order: [x1, x2, t, affine]

    loc2 = ha.new_mode('loc2')
    a_matrix = [ \
        [0, 1, 0, 0], \
        [18300, -20, 0, -9562.5], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc2.set_dynamics(a_matrix)
    # -x1 > -0.51 & -x1 ≤ -0.5
    loc2.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [0.51, ...
       -0.5, ])

    loc3 = ha.new_mode('loc3')
    a_matrix = [ \
        [0, 1, 0, 0], \
        [-825, -20, 0, 0], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc3.set_dynamics(a_matrix)
    # -x1 > -0.5 & -x1 ≤ -0.09
    loc3.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [0.5, ...
       -0.09, ])

    loc4 = ha.new_mode('loc4')
    a_matrix = [ \
        [0, 1, 0, 0], \
        [30675, -20, 0, -2835], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc4.set_dynamics(a_matrix)
```

```
51      # -x1 > -0.09 & -x1 ≤ -0.08
52      loc4.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [0.09, ...
            -0.08, ])
53
54      loc1 = ha.new_mode('loc1')
55      a_matrix = [ \
56          [0, 1, 0, 0], \
57          [-450, -20, 0, 0], \
58          [0, 0, 0, 1], \
59          [0, 0, 0, 0], \
60          ]
61      loc1.set_dynamics(a_matrix)
62      # -x1 ≤ -0.51
63      loc1.set_invariant([[-1, 0, 0, 0], ], [-0.51, ])
64
65      loc5 = ha.new_mode('loc5')
66      a_matrix = [ \
67          [0, 1, 0, 0], \
68          [-4762.5, -20, 0, 0], \
69          [0, 0, 0, 1], \
70          [0, 0, 0, 0], \
71          ]
72      loc5.set_dynamics(a_matrix)
73      # -x1 > -0.08 & -x1 < 0.08
74      loc5.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [0.08, ...
            0.08, ])
75
76      loc6 = ha.new_mode('loc6')
77      a_matrix = [ \
78          [0, 1, 0, 0], \
79          [30675, -20, 0, 2835], \
80          [0, 0, 0, 1], \
81          [0, 0, 0, 0], \
82          ]
83      loc6.set_dynamics(a_matrix)
84      # -x1 ≥ 0.08 & -x1 < 0.09
```

```
85      loc6.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], ...
            [-0.08, 0.09, ])
86
87      loc7 = ha.new_mode('loc7')
88      a_matrix = [ \
89          [0, 1, 0, 0], \
90          [-825, -20, 0, 0], \
91          [0, 0, 0, 1], \
92          [0, 0, 0, 0], \
93          ]
94      loc7.set_dynamics(a_matrix)
95      # -x1 ≥ 0.09 & -x1 < 0.5
96      loc7.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], ...
            [-0.09, 0.5, ])
97
98      loc8 = ha.new_mode('loc8')
99      a_matrix = [ \
100         [0, 1, 0, 0], \
101         [18300, -20, 0, 9562.5], \
102         [0, 0, 0, 1], \
103         [0, 0, 0, 0], \
104         ]
105     loc8.set_dynamics(a_matrix)
106     # -x1 ≥ 0.5 & -x1 < 0.51
107     loc8.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [-0.5, ...
            0.51, ])
108
109     loc9 = ha.new_mode('loc9')
110     a_matrix = [ \
111         [0, 1, 0, 0], \
112         [-450, -20, 0, 0], \
113         [0, 0, 0, 1], \
114         [0, 0, 0, 0], \
115         ]
116     loc9.set_dynamics(a_matrix)
117     # -x1 ≥ 0.51
```

```
118     loc9.set_invariant([[1, -0, -0, -0], ], [-0.51, ])
119
120     trans = ha.new_transition(loc2, loc1)
121     # -x1 ≤ -0.51
122     trans.set_guard([[-1, 0, 0, 0], ], [-0.51, ])
123
124     trans = ha.new_transition(loc2, loc3)
125     # -x1 > -0.5
126     trans.set_guard([[1, -0, -0, -0], ], [0.5, ])
127
128     trans = ha.new_transition(loc3, loc4)
129     # -x1 > -0.09
130     trans.set_guard([[1, -0, -0, -0], ], [0.09, ])
131
132     trans = ha.new_transition(loc3, loc2)
133     # -x1 ≤ -0.5
134     trans.set_guard([[-1, 0, 0, 0], ], [-0.5, ])
135
136     trans = ha.new_transition(loc4, loc5)
137     # -x1 > -0.08
138     trans.set_guard([[1, -0, -0, -0], ], [0.08, ])
139
140     trans = ha.new_transition(loc4, loc3)
141     # -x1 ≤ -0.09
142     trans.set_guard([[-1, 0, 0, 0], ], [-0.09, ])
143
144     trans = ha.new_transition(loc1, loc2)
145     # -x1 > -0.51
146     trans.set_guard([[1, -0, -0, -0], ], [0.51, ])
147
148     trans = ha.new_transition(loc5, loc6)
149     # -x1 ≥ 0.0801
150     trans.set_guard([[1, -0, -0, -0], ], [-0.0801, ])
151
152     trans = ha.new_transition(loc5, loc4)
153     # -x1 ≤ -0.0801
```

```
154     trans.set_guard([[-1, 0, 0, 0], ], [-0.0801, ])
155
156     trans = ha.new_transition(loc6, loc7)
157     # -x1 ≥ 0.09
158     trans.set_guard([[1, -0, -0, -0], ], [-0.09, ])
159
160     trans = ha.new_transition(loc6, loc5)
161     # -x1 < 0.08
162     trans.set_guard([[-1, 0, 0, 0], ], [0.08, ])
163
164     trans = ha.new_transition(loc7, loc8)
165     # -x1 ≥ 0.5
166     trans.set_guard([[1, -0, -0, -0], ], [-0.5, ])
167
168     trans = ha.new_transition(loc7, loc6)
169     # -x1 < 0.09
170     trans.set_guard([[-1, 0, 0, 0], ], [0.09, ])
171
172     trans = ha.new_transition(loc8, loc9)
173     # -x1 ≥ 0.51
174     trans.set_guard([[1, -0, -0, -0], ], [-0.51, ])
175
176     trans = ha.new_transition(loc8, loc7)
177     # -x1 < 0.5
178     trans.set_guard([[-1, 0, 0, 0], ], [0.5, ])
179
180     trans = ha.new_transition(loc9, loc8)
181     # -x1 < 0.51
182     trans.set_guard([[-1, 0, 0, 0], ], [0.51, ])
183
184     return ha
185
186 def define_init_states(ha):
187     '''returns a list of StateSet objects'''
188     # Variable ordering: [x1, x2, t, affine]
189     rv = []
```

```
190
191     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
192     mode = ha.modes['loc2']
193     mat = [[-1, 0, 0, 0], \
194          [1, 0, 0, 0], \
195          [0, 1, 0, 0], \
196          [-0, -1, -0, -0], \
197          [0, 0, 1, 0], \
198          [-0, -0, -1, -0], \
199          [0, 0, 0, 1], \
200          [-0, -0, -0, -1], ]
201     rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
202     rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
            mode))
203
204     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
205     mode = ha.modes['loc3']
206     mat = [[-1, 0, 0, 0], \
207          [1, 0, 0, 0], \
208          [0, 1, 0, 0], \
209          [-0, -1, -0, -0], \
210          [0, 0, 1, 0], \
211          [-0, -0, -1, -0], \
212          [0, 0, 0, 1], \
213          [-0, -0, -0, -1], ]
214     rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
215     rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
            mode))
216
217     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
218     mode = ha.modes['loc4']
219     mat = [[-1, 0, 0, 0], \
220          [1, 0, 0, 0], \
221          [0, 1, 0, 0], \
222          [-0, -1, -0, -0], \
223          [0, 0, 1, 0], \
```

```
224          [-0, -0, -1, -0], \
225          [0, 0, 0, 1], \
226          [-0, -0, -0, -1], ]
227      rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
228      rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
             mode))
229
230      # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
231      mode = ha.modes['loc1']
232      mat = [[-1, 0, 0, 0], \
233          [1, 0, 0, 0], \
234          [0, 1, 0, 0], \
235          [-0, -1, -0, -0], \
236          [0, 0, 1, 0], \
237          [-0, -0, -1, -0], \
238          [0, 0, 0, 1], \
239          [-0, -0, -0, -1], ]
240      rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
241      rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
             mode))
242
243      # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
244      mode = ha.modes['loc5']
245      mat = [[-1, 0, 0, 0], \
246          [1, 0, 0, 0], \
247          [0, 1, 0, 0], \
248          [-0, -1, -0, -0], \
249          [0, 0, 1, 0], \
250          [-0, -0, -1, -0], \
251          [0, 0, 0, 1], \
252          [-0, -0, -0, -1], ]
253      rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
254      rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
             mode))
255
256      # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
```

```
257     mode = ha.modes['loc6']
258     mat = [[-1, 0, 0, 0], \
259         [1, 0, 0, 0], \
260         [0, 1, 0, 0], \
261         [-0, -1, -0, -0], \
262         [0, 0, 1, 0], \
263         [-0, -0, -1, -0], \
264         [0, 0, 0, 1], \
265         [-0, -0, -0, -1], ]
266     rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
267     rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
            mode))
268
269     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
270     mode = ha.modes['loc7']
271     mat = [[-1, 0, 0, 0], \
272         [1, 0, 0, 0], \
273         [0, 1, 0, 0], \
274         [-0, -1, -0, -0], \
275         [0, 0, 1, 0], \
276         [-0, -0, -1, -0], \
277         [0, 0, 0, 1], \
278         [-0, -0, -0, -1], ]
279     rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
280     rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
            mode))
281
282     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
283     mode = ha.modes['loc8']
284     mat = [[-1, 0, 0, 0], \
285         [1, 0, 0, 0], \
286         [0, 1, 0, 0], \
287         [-0, -1, -0, -0], \
288         [0, 0, 1, 0], \
289         [-0, -0, -1, -0], \
290         [0, 0, 0, 1], \
```

```
291          [-0, -0, -0, -1], ]
292      rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
293      rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
             mode))
294
295      # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
296      mode = ha.modes['loc9']
297      mat = [[-1, 0, 0, 0], \
298          [1, 0, 0, 0], \
299          [0, 1, 0, 0], \
300          [-0, -1, -0, -0], \
301          [0, 0, 1, 0], \
302          [-0, -0, -1, -0], \
303          [0, 0, 0, 1], \
304          [-0, -0, -0, -1], ]
305      rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
306      rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
             mode))
307
308      return rv
309
310
311  def define_settings():
312      '''get the hylaa settings object
313      see hylaa/settings.py for a complete list of reachability ...
             settings'''
314
315      # step_size = 0.01, max_time = 0.5
316      settings = HylaaSettings(0.001, 0.5)
317      settings.plot.plot_mode = PlotSettings.PLOT_IMAGE
318      settings.plot.xdim_dir = 2
319      settings.plot.ydim_dir = 0
320
321      return settings
322
323  def run_hylaa():
```

```
    'runs hylaa, returning a HylaaResult object'
    ha = define_ha()
    init = define_init_states(ha)
    settings = define_settings()

    result = Core(ha, settings).run(init)

    return result

if __name__ == '__main__':
    run_hylaa()
```

A.8 PselDFLC Python Response Modeling

```
'''
Created by Hyst v1.4
Hybrid Automaton in PySim
Converted from file: ...
    /home/verivital/Desktop/fuzzyPselD_v2/prop_pd.xml
Command Line arguments: -tool pysim "" -output ...
    /home/verivital/Desktop/fuzzyPselD_v2/prop_pd_pysim.py -input ...
    /home/verivital/Desktop/fuzzyPselD_v2/prop_pd.xml ...
    /home/verivital/Desktop/fuzzyPselD_v2/prop_pd.cfg
'''

import hybridpy.pysim.simulate as sim
from hybridpy.pysim.simulate import init_list_to_q_list, ...
    PySimSettings
from hybridpy.pysim.hybrid_automaton import HybridAutomaton, ...
    HyperRectangle

def define_ha():
    '''make the hybrid automaton and return it'''
```

```
14
15      ha = HybridAutomaton()
16      ha.variables = ["x1", "x2", "t"]
17
18
19      loc2 = ha.new_mode('loc2')
20      loc2.inv = lambda state: 1.0 - state[0] > -0.51 and 1.0 - ...
           state[0] ≤ -0.5
21      loc2.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] - 18375.0 * (1.0 - state[0]) - 9562.5, 1.0]
22      loc2.der_interval_list = [[0, 0], [0, 0], [0, 0]]
23
24      loc3 = ha.new_mode('loc3')
25      loc3.inv = lambda state: 1.0 - state[0] > -0.5 and 1.0 - ...
           state[0] ≤ -0.09
26      loc3.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 750.0 * (1.0 - state[0]), 1.0]
27      loc3.der_interval_list = [[0, 0], [0, 0], [0, 0]]
28
29      loc4 = ha.new_mode('loc4')
30      loc4.inv = lambda state: 1.0 - state[0] > -0.09 and 1.0 - ...
           state[0] ≤ -0.08
31      loc4.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] - 30750.0 * (1.0 - state[0]) - 2835.0 - ...
           state[1] * 0.15 * 375.0, 1.0]
32      loc4.der_interval_list = [[0, 0], [0, 0], [0, 0]]
33
34      loc1 = ha.new_mode('loc1')
35      loc1.inv = lambda state: 1.0 - state[0] ≤ -0.51
36      loc1.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 375.0 * (1.0 - state[0]), 1.0]
37      loc1.der_interval_list = [[0, 0], [0, 0], [0, 0]]
38
39      loc5 = ha.new_mode('loc5')
40      loc5.inv = lambda state: 1.0 - state[0] > -0.08 and 1.0 - ...
           state[0] < 0.08
```

```
41      loc5.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 4687.5 * (1.0 - state[0]) - state[1] * ...
           0.15 * 375.0, 1.0]
42      loc5.der_interval_list = [[0, 0], [0, 0], [0, 0]]
43
44      loc6 = ha.new_mode('loc6')
45      loc6.inv = lambda state: 1.0 - state[0] ≥ 0.08 and 1.0 - ...
           state[0] < 0.09
46      loc6.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] - 30750.0 * (1.0 - state[0]) + 2835.0 - ...
           state[1] * 0.15 * 375.0, 1.0]
47      loc6.der_interval_list = [[0, 0], [0, 0], [0, 0]]
48
49      loc7 = ha.new_mode('loc7')
50      loc7.inv = lambda state: 1.0 - state[0] ≥ 0.09 and 1.0 - ...
           state[0] < 0.5
51      loc7.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 750.0 * (1.0 - state[0]), 1.0]
52      loc7.der_interval_list = [[0, 0], [0, 0], [0, 0]]
53
54      loc8 = ha.new_mode('loc8')
55      loc8.inv = lambda state: 1.0 - state[0] ≥ 0.5 and 1.0 - ...
           state[0] < 0.51
56      loc8.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] - 18375.0 * (1.0 - state[0]) + 9562.5, 1.0]
57      loc8.der_interval_list = [[0, 0], [0, 0], [0, 0]]
58
59      loc9 = ha.new_mode('loc9')
60      loc9.inv = lambda state: 1.0 - state[0] ≥ 0.51
61      loc9.der = lambda _, state: [state[1], -75.0 * state[0] - ...
           20.0 * state[1] + 375.0 * (1.0 - state[0]), 1.0]
62      loc9.der_interval_list = [[0, 0], [0, 0], [0, 0]]
63
64      t = ha.new_transition(loc2, loc1)
65      t.guard = lambda state: 1.0 - state[0] ≤ -0.51
66      t.reset = lambda state: [None, None, None]
```

```
67
68      t = ha.new_transition(loc2, loc3)
69      t.guard = lambda state: 1.0 - state[0] > -0.5
70      t.reset = lambda state: [None, None, None]
71
72      t = ha.new_transition(loc3, loc4)
73      t.guard = lambda state: 1.0 - state[0] > -0.09
74      t.reset = lambda state: [None, None, None]
75
76      t = ha.new_transition(loc3, loc2)
77      t.guard = lambda state: 1.0 - state[0] ≤ -0.5
78      t.reset = lambda state: [None, None, None]
79
80      t = ha.new_transition(loc4, loc5)
81      t.guard = lambda state: 1.0 - state[0] > -0.08
82      t.reset = lambda state: [None, None, None]
83
84      t = ha.new_transition(loc4, loc3)
85      t.guard = lambda state: 1.0 - state[0] ≤ -0.09
86      t.reset = lambda state: [None, None, None]
87
88      t = ha.new_transition(loc1, loc2)
89      t.guard = lambda state: 1.0 - state[0] > -0.51
90      t.reset = lambda state: [None, None, None]
91
92      t = ha.new_transition(loc5, loc6)
93      t.guard = lambda state: 1.0 - state[0] ≥ 0.0801
94      t.reset = lambda state: [None, None, None]
95
96      t = ha.new_transition(loc5, loc4)
97      t.guard = lambda state: 1.0 - state[0] ≤ -0.0801
98      t.reset = lambda state: [None, None, None]
99
100     t = ha.new_transition(loc6, loc7)
101     t.guard = lambda state: 1.0 - state[0] ≥ 0.09
102     t.reset = lambda state: [None, None, None]
```

```
103
104     t = ha.new_transition(loc6, loc5)
105     t.guard = lambda state: 1.0 - state[0] < 0.08
106     t.reset = lambda state: [None, None, None]
107
108     t = ha.new_transition(loc7, loc8)
109     t.guard = lambda state: 1.0 - state[0] ≥ 0.5
110     t.reset = lambda state: [None, None, None]
111
112     t = ha.new_transition(loc7, loc6)
113     t.guard = lambda state: 1.0 - state[0] < 0.09
114     t.reset = lambda state: [None, None, None]
115
116     t = ha.new_transition(loc8, loc9)
117     t.guard = lambda state: 1.0 - state[0] ≥ 0.51
118     t.reset = lambda state: [None, None, None]
119
120     t = ha.new_transition(loc8, loc7)
121     t.guard = lambda state: 1.0 - state[0] < 0.5
122     t.reset = lambda state: [None, None, None]
123
124     t = ha.new_transition(loc9, loc8)
125     t.guard = lambda state: 1.0 - state[0] < 0.51
126     t.reset = lambda state: [None, None, None]
127
128     return ha
129
130 def define_init_states(ha):
131     '''returns a list of (mode, HyperRectangle)'''
132     # Variable ordering: [x1, x2, t]
133     rv = []
134
135     rv.append((ha.modes['loc2'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
136     rv.append((ha.modes['loc3'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
```

```
137     rv.append((ha.modes['loc4'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
138     rv.append((ha.modes['loc1'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
139     rv.append((ha.modes['loc5'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
140     rv.append((ha.modes['loc6'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
141     rv.append((ha.modes['loc7'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
142     rv.append((ha.modes['loc8'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
143     rv.append((ha.modes['loc9'],HyperRectangle([(0, 0), (0, 0), ...
            (0, 0)])))
144     return rv
145
146
147 def define_settings():
148     '''defines the automaton / plot settings'''
149     s = PySimSettings()
150     s.max_time = 0.3
151     s.step = 0.01
152     s.dim_x = 2
153     s.dim_y = 0
154
155     return s
156
157 def simulate(init_states, settings):
158     '''simulate the automaton from each initial rect'''
159
160     q_list = init_list_to_q_list(init_states, center=True, ...
            star=True, corners=False, rand=0)
161     result = sim.simulate_multi(q_list, settings.max_time)
162
163     return result
164
```

```
165 def plot(result, init_states, image_path, settings):
166     '''plot a simulation result to a file'''
167 
168     draw_events = len(result) == 1
169     shouldShow = False
170     sim.plot_sim_result_multi(result, settings.dim_x, ...
            settings.dim_y, image_path, draw_events, legend=True, ...
            title='Simulation', show=shouldShow, init_states=init_states)
171 
172 if __name__ == '__main__':
173     ha = define_ha()
174     settings = define_settings()
175     init_states = define_init_states(ha)
176     plot(simulate(init_states, settings), init_states, ...
            'plot.png', settings)
```

A.9 PselDFLC Zero Input Hybrid Representations

```
1  system = state_diagram
2 
3  initially = "-1.0 ≤ x1 ≤ 1.0 & x2 == 0 & t == 0"
4 
5  set-aggregation = chull
6  scenario = stc
7  directions = box
8  sampling-time = 0.01
9  time-horizon = 0.3
10 #iter-max = 15
11 #iter-max = 30
12 output-variables = "t, x1"
13 output-format = GEN
14 rel-err = 1.0E-6
15 abs-err = 1.0E-5
```

```
flowpipe-tolerance = 0.01

# New attempt 20190103 - prop_pd_zi_spx_im09.png
# - If don't want reactivate the above fields
#iter-max = 9

# New attempt 20190103 - prop_pd_zi_spx_im18.png
# - If don't want reactivate the above fields
#iter-max = 18

# New attempt 20190103 - prop_pd_zi_spx_im27.png
# - If don't want reactivate the above fields
#iter-max = 27

# New attempt 20190103 - prop_pd_zi_spx_im36.png
# - If don't want reactivate the above fields
#iter-max = 36

# New attempt 20190103 - prop_pd_zi_spx_im45.png
# - If don't want reactivate the above fields
#iter-max = 45

# New attempt 20190103 - prop_pd_zi_spx_im54.png
# - If don't want reactivate the above fields
#iter-max = 54

# New attempt 20190103 - prop_pd_zi_spx_im63.png
# - If don't want reactivate the above fields
#iter-max = 63

# New attempt 20190103 - prop_pd_zi_spx_im90.png
# - If don't want reactivate the above fields
#iter-max = 90
# - Error on iteration 79

# New attempt 20190103 - prop_pd_zi_spx_im78.png
```

```
# - If don't want reactivate the above fields
iter-max = 78
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<sspaceex ...
    xmlns="http://www-verimag.imag.fr/xml-namespaces/sspaceex" ...
    version="0.2" math="SpaceEx">
  <component id="state_diagram">
    <param name="x1" type="real" local="false" d1="1" d2="1" ...
        dynamics="any" />
    <param name="x2" type="real" local="false" d1="1" d2="1" ...
        dynamics="any" />
    <param name="t" type="real" local="false" d1="1" d2="1" ...
        dynamics="any" />
    <location id="2" name="loc2" x="478.0" y="136.0" ...
        width="294.0" height="110.0">
      <note>75</note>
      <invariant>(0-x1) &gt;-0.51&amp; (0-x1)&lt;=-0.5</invariant>
      <flow>x1'== x2&amp;
x2'== -75*x1-20*x2-375*49*(0-x1)-375*25.5&amp;
t'== 1</flow>
    </location>
    <location id="3" name="loc3" x="803.0" y="208.0" ...
        width="234.0" height="94.0">
      <note>75</note>
      <invariant>(0-x1) &gt;-0.5 &amp; (0-x1) &lt;=-0.09</invariant>
      <flow>x1'== x2&amp;
x2'== -75*x1-20*x2+375*2*(0-x1)&amp;
t'==1</flow>
    </location>
    <location id="4" name="loc4" x="1123.0" y="108.0" ...
        width="374.0" height="98.0">
      <note>75</note>
      <invariant>(0-x1) &gt;-0.09 &amp; (0-x1) &lt;=-0.08</invariant>
```

```
24        <flow>x1'== x2&amp;;
25  x2'== -75*x1-20*x2-375*82*(0-x1)-375*7.56-x2*0.15*375&amp;;
26  t'==1</flow>
27      </location>
28      <location id="6" name="loc1" x="141.0" y="231.0" ...
            width="220.0" height="98.0">
29        <note>5555</note>
30        <invariant>(0-x1) &lt;= -0.51</invariant>
31        <flow>x1'== x2&amp;;
32  x2'== -75*x1-20*x2+375*(0-x1)&amp;;
33  t'== 1</flow>
34      </location>
35      <location id="1" name="loc5" x="1151.0" y="354.0" ...
            width="336.0" height="98.0">
36        <note>75</note>
37        <invariant>(0-x1) &gt;-0.08 &amp; (0-x1)&lt;0.08</invariant>
38        <flow>x1'== x2&amp;;
39  x2'== -75*x1-20*x2+375*12.5*(0-x1)-x2*0.15*375&amp;;
40  t'== 1</flow>
41      </location>
42      <location id="5" name="loc6" x="1149.0" y="586.0" ...
            width="384.0" height="94.0">
43        <note>75</note>
44        <invariant>(0-x1)&gt;= 0.08 &amp; (0-x1)&lt;0.09</invariant>
45        <flow>x1'== x2&amp;;
46  x2'== -75*x1-20*x2-375*82*(0-x1)+375*7.56-x2*0.15*375&amp;;
47  t'== 1</flow>
48      </location>
49      <location id="7" name="loc7" x="798.0" y="478.0" ...
            width="258.0" height="108.0">
50        <note>75</note>
51        <invariant>(0-x1)&gt;= 0.09 &amp; (0-x1)&lt;0.5</invariant>
52        <flow>x1'== x2&amp;;
53  x2'== -75*x1-20*x2+375*2*(0-x1)&amp;;
54  t'== 1</flow>
55      </location>
```

```
56      <location id="8" name="loc8" x="452.0" y="588.0" ...
           width="296.0" height="92.0">
57        <note>75</note>
58        <invariant>(0-x1)&amp;gt;= 0.5&amp;amp;(0-x1)&amp;lt;0.51</invariant>
59        <flow>x1'== x2&amp;amp;
60  x2'== -75*x1-20*x2-375*49*(0-x1)+375*25.5&amp;amp;
61  t'== 1</flow>
62      </location>
63      <location id="9" name="loc9" x="144.0" y="410.0" ...
           width="228.0" height="98.0">
64        <note>5555</note>
65        <invariant>(0-x1)&amp;gt;=0.51</invariant>
66        <flow>x1'== x2&amp;amp;
67  x2'== -75*x1-20*x2+375*(0-x1)&amp;amp;
68  t'== 1</flow>
69      </location>
70      <transition source="6" target="2">
71        <guard>(0-x1)&amp;gt;-0.51</guard>
72        <labelposition x="-98.0" y="-69.0" width="116.0" ...
             height="64.0" />
73        <middlepoint x="284.0" y="152.0" />
74      </transition>
75      <transition source="2" target="6">
76        <guard>(0-x1)&amp;lt;=-0.51</guard>
77        <labelposition x="-19.0" y="-2.0" width="108.0" ...
             height="62.0" />
78        <middlepoint x="315.0" y="238.5" />
79      </transition>
80      <transition source="2" target="3">
81        <guard>(0-x1)&amp;gt;-0.5</guard>
82        <labelposition x="-45.0" y="-81.0" width="124.0" ...
             height="78.0" />
83        <middlepoint x="678.0" y="144.5" />
84      </transition>
85      <transition source="3" target="4">
86        <guard>(0-x1)&amp;gt;-0.09</guard>
```

```
87        <labelposition x="-88.0" y="-55.0" width="86.0" ...
              height="50.0" />
88        <middlepoint x="898.5" y="120.0" />
89      </transition>
90      <transition source="4" target="1">
91        <guard>(0-x1) &gt; -0.08</guard>
92        <labelposition x="6.0" y="-36.0" width="100.0" ...
              height="54.0" />
93        <middlepoint x="1176.0" y="262.0" />
94      </transition>
95      <transition source="1" target="5">
96        <guard>(0-x1)&gt;=0.0801</guard>
97        <labelposition x="2.0" y="-35.0" width="104.0" ...
              height="64.0" />
98        <middlepoint x="1186.0" y="480.0" />
99      </transition>
100     <transition source="5" target="7">
101       <guard>(0-x1)&gt;=0.09</guard>
102       <labelposition x="-41.0" y="-43.0" width="98.0" ...
              height="50.0" />
103       <middlepoint x="992.0" y="513.0" />
104     </transition>
105     <transition source="7" target="8">
106       <guard>(0-x1)&gt;=0.5</guard>
107       <labelposition x="-41.0" y="-62.0" width="80.0" ...
              height="60.0" />
108       <middlepoint x="614.5" y="504.0" />
109     </transition>
110     <transition source="8" target="9">
111       <guard>(0-x1)&gt;=0.51</guard>
112       <labelposition x="-11.0" y="-53.0" width="104.0" ...
              height="50.0" />
113       <middlepoint x="321.0" y="458.0" />
114     </transition>
115     <transition source="3" target="2">
116       <guard>(0-x1)&lt;=-0.5</guard>
```

```
117       <labelposition x="-119.0" y="2.0" width="96.0" ...
              height="70.0" />
118       <middlepoint x="630.0" y="232.5" />
119     </transition>
120     <transition source="4" target="3">
121       <guard>(0-x1)&lt;=-0.090</guard>
122       <labelposition x="-55.0" y="-8.0" width="102.0" ...
              height="50.0" />
123       <middlepoint x="992.0" y="190.5" />
124     </transition>
125     <transition source="1" target="4">
126       <guard>(0-x1)&lt;=-0.0801</guard>
127       <labelposition x="-89.0" y="-5.0" width="122.0" ...
              height="54.0" />
128       <middlepoint x="1081.0" y="248.0" />
129     </transition>
130     <transition source="5" target="1">
131       <guard>(0-x1)&lt;0.08</guard>
132       <labelposition x="-71.0" y="-59.0" width="82.0" ...
              height="50.0" />
133       <middlepoint x="1077.5" y="475.5" />
134     </transition>
135     <transition source="7" target="5">
136       <guard>(0-x1)&lt;0.0900</guard>
137       <labelposition x="-93.0" y="9.0" width="102.0" ...
              height="50.0" />
138       <middlepoint x="910.5" y="580.0" />
139     </transition>
140     <transition source="8" target="7">
141       <guard>(0-x1)&lt;0.5</guard>
142       <labelposition x="-37.0" y="5.0" width="96.0" height="52.0" />
143       <middlepoint x="642.5" y="598.5" />
144     </transition>
145     <transition source="9" target="8">
146       <guard>(0-x1)&lt;0.51</guard>
```

```
147         <labelposition x="-51.0" y="3.0" width="118.0" ...
                height="50.0" />
148         <middlepoint x="243.0" y="524.0" />
149       </transition>
150     </component>
151 </sspaceex>
```

```
1  '''
2  Created by Hyst v1.5
3  Hybrid Automaton in Hylaa2
4  Converted from file: ...
       /home/awf/Thesis/MastersThesis/linux/WorkingDesktop/fuzzyPselD_v2/prop_pd_zi.x
5  Command Line arguments: -tool hylaa2 "" -passes simplify ...
       -python_simplify -output ...
       /home/awf/Thesis/MastersThesis/linux/WorkingDesktop/fuzzyPselD_v2/prop_pd_zi_h
       -input ...
       /home/awf/Thesis/MastersThesis/linux/WorkingDesktop/fuzzyPselD_v2/prop_pd_zi.x
       /home/awf/Thesis/MastersThesis/linux/WorkingDesktop/fuzzyPselD_v2/prop_pd_zi.c
6  '''
7
8  from hylaa.hybrid_automaton import HybridAutomaton
9  from hylaa.settings import HylaaSettings, PlotSettings
10 from hylaa.core import Core
11 from hylaa.stateset import StateSet
12 from hylaa import lputil
13
14 def define_ha():
15     '''make the hybrid automaton and return it'''
16
17     ha = HybridAutomaton()
18
19     # dynamics variable order: [x1, x2, t, affine]
20
21     loc2 = ha.new_mode('loc2')
```

```
    a_matrix = [ \
        [0, 1, 0, 0], \
        [18300, -20, 0, -9562.5], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc2.set_dynamics(a_matrix)
    # -x1 > -0.51 & -x1 ≤ -0.5
    loc2.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [0.51, ...
        -0.5, ])

    loc3 = ha.new_mode('loc3')
    a_matrix = [ \
        [0, 1, 0, 0], \
        [-825, -20, 0, 0], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc3.set_dynamics(a_matrix)
    # -x1 > -0.5 & -x1 ≤ -0.09
    loc3.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [0.5, ...
        -0.09, ])

    loc4 = ha.new_mode('loc4')
    a_matrix = [ \
        [0, 1, 0, 0], \
        [30675, -76.25, 0, -2835], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc4.set_dynamics(a_matrix)
    # -x1 > -0.09 & -x1 ≤ -0.08
    loc4.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [0.09, ...
        -0.08, ])

    loc1 = ha.new_mode('loc1')
```

```
    a_matrix = [ \
        [0, 1, 0, 0], \
        [-450, -20, 0, 0], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc1.set_dynamics(a_matrix)
    # -x1 ≤ -0.51
    loc1.set_invariant([[-1, 0, 0, 0], ], [-0.51, ])

    loc5 = ha.new_mode('loc5')
    a_matrix = [ \
        [0, 1, 0, 0], \
        [-4762.5, -76.25, 0, 0], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc5.set_dynamics(a_matrix)
    # -x1 > -0.08 & -x1 < 0.08
    loc5.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [0.08, ...
        0.08, ])

    loc6 = ha.new_mode('loc6')
    a_matrix = [ \
        [0, 1, 0, 0], \
        [30675, -76.25, 0, 2835], \
        [0, 0, 0, 1], \
        [0, 0, 0, 0], \
        ]
    loc6.set_dynamics(a_matrix)
    # -x1 ≥ 0.08 & -x1 < 0.09
    loc6.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], ...
        [-0.08, 0.09, ])

    loc7 = ha.new_mode('loc7')
    a_matrix = [ \
```

```
89          [0, 1, 0, 0], \
90          [-825, -20, 0, 0], \
91          [0, 0, 0, 1], \
92          [0, 0, 0, 0], \
93          ]
94      loc7.set_dynamics(a_matrix)
95      # -x1 ≥ 0.09 & -x1 < 0.5
96      loc7.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], ...
            [-0.09, 0.5, ])
97
98      loc8 = ha.new_mode('loc8')
99      a_matrix = [ \
100         [0, 1, 0, 0], \
101         [18300, -20, 0, 9562.5], \
102         [0, 0, 0, 1], \
103         [0, 0, 0, 0], \
104         ]
105     loc8.set_dynamics(a_matrix)
106     # -x1 ≥ 0.5 & -x1 < 0.51
107     loc8.set_invariant([[1, -0, -0, -0], [-1, 0, 0, 0], ], [-0.5, ...
            0.51, ])
108
109     loc9 = ha.new_mode('loc9')
110     a_matrix = [ \
111         [0, 1, 0, 0], \
112         [-450, -20, 0, 0], \
113         [0, 0, 0, 1], \
114         [0, 0, 0, 0], \
115         ]
116     loc9.set_dynamics(a_matrix)
117     # -x1 ≥ 0.51
118     loc9.set_invariant([[1, -0, -0, -0], ], [-0.51, ])
119
120     trans = ha.new_transition(loc2, loc1)
121     # -x1 ≤ -0.51
122     trans.set_guard([[-1, 0, 0, 0], ], [-0.51, ])
```

```
123
124     trans = ha.new_transition(loc2, loc3)
125     # -x1 > -0.5
126     trans.set_guard([[1, -0, -0, -0], ], [0.5, ])
127
128     trans = ha.new_transition(loc3, loc4)
129     # -x1 > -0.09
130     trans.set_guard([[1, -0, -0, -0], ], [0.09, ])
131
132     trans = ha.new_transition(loc3, loc2)
133     # -x1 ≤ -0.5
134     trans.set_guard([[-1, 0, 0, 0], ], [-0.5, ])
135
136     trans = ha.new_transition(loc4, loc5)
137     # -x1 > -0.08
138     trans.set_guard([[1, -0, -0, -0], ], [0.08, ])
139
140     trans = ha.new_transition(loc4, loc3)
141     # -x1 ≤ -0.09
142     trans.set_guard([[-1, 0, 0, 0], ], [-0.09, ])
143
144     trans = ha.new_transition(loc1, loc2)
145     # -x1 > -0.51
146     trans.set_guard([[1, -0, -0, -0], ], [0.51, ])
147
148     trans = ha.new_transition(loc5, loc6)
149     # -x1 ≥ 0.0801
150     trans.set_guard([[1, -0, -0, -0], ], [-0.0801, ])
151
152     trans = ha.new_transition(loc5, loc4)
153     # -x1 ≤ -0.0801
154     trans.set_guard([[-1, 0, 0, 0], ], [-0.0801, ])
155
156     trans = ha.new_transition(loc6, loc7)
157     # -x1 ≥ 0.09
158     trans.set_guard([[1, -0, -0, -0], ], [-0.09, ])
```

```
159
160     trans = ha.new_transition(loc6, loc5)
161     # -x1 < 0.08
162     trans.set_guard([[-1, 0, 0, 0], ], [0.08, ])
163
164     trans = ha.new_transition(loc7, loc8)
165     # -x1 ≥ 0.5
166     trans.set_guard([[1, -0, -0, -0], ], [-0.5, ])
167
168     trans = ha.new_transition(loc7, loc6)
169     # -x1 < 0.09
170     trans.set_guard([[-1, 0, 0, 0], ], [0.09, ])
171
172     trans = ha.new_transition(loc8, loc9)
173     # -x1 ≥ 0.51
174     trans.set_guard([[1, -0, -0, -0], ], [-0.51, ])
175
176     trans = ha.new_transition(loc8, loc7)
177     # -x1 < 0.5
178     trans.set_guard([[-1, 0, 0, 0], ], [0.5, ])
179
180     trans = ha.new_transition(loc9, loc8)
181     # -x1 < 0.51
182     trans.set_guard([[-1, 0, 0, 0], ], [0.51, ])
183
184     return ha
185
186 def define_init_states(ha):
187     '''returns a list of StateSet objects'''
188     # Variable ordering: [x1, x2, t, affine]
189     rv = []
190
191     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
192     mode = ha.modes['loc2']
193     mat = [[-1, 0, 0, 0], \
194         [1, 0, 0, 0], \
```

```
195          [0, 1, 0, 0], \
196          [-0, -1, -0, -0], \
197          [0, 0, 1, 0], \
198          [-0, -0, -1, -0], \
199          [0, 0, 0, 1], \
200          [-0, -0, -0, -1], ]
201      rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
202      rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
             mode))
203
204      # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
205      mode = ha.modes['loc3']
206      mat = [[-1, 0, 0, 0], \
207          [1, 0, 0, 0], \
208          [0, 1, 0, 0], \
209          [-0, -1, -0, -0], \
210          [0, 0, 1, 0], \
211          [-0, -0, -1, -0], \
212          [0, 0, 0, 1], \
213          [-0, -0, -0, -1], ]
214      rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
215      rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
             mode))
216
217      # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
218      mode = ha.modes['loc4']
219      mat = [[-1, 0, 0, 0], \
220          [1, 0, 0, 0], \
221          [0, 1, 0, 0], \
222          [-0, -1, -0, -0], \
223          [0, 0, 1, 0], \
224          [-0, -0, -1, -0], \
225          [0, 0, 0, 1], \
226          [-0, -0, -0, -1], ]
227      rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
```

```
228     rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
            mode))
229
230     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
231     mode = ha.modes['loc1']
232     mat = [[-1, 0, 0, 0], \
233          [1, 0, 0, 0], \
234          [0, 1, 0, 0], \
235          [-0, -1, -0, -0], \
236          [0, 0, 1, 0], \
237          [-0, -0, -1, -0], \
238          [0, 0, 0, 1], \
239          [-0, -0, -0, -1], ]
240     rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
241     rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
            mode))
242
243     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
244     mode = ha.modes['loc5']
245     mat = [[-1, 0, 0, 0], \
246          [1, 0, 0, 0], \
247          [0, 1, 0, 0], \
248          [-0, -1, -0, -0], \
249          [0, 0, 1, 0], \
250          [-0, -0, -1, -0], \
251          [0, 0, 0, 1], \
252          [-0, -0, -0, -1], ]
253     rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
254     rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
            mode))
255
256     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
257     mode = ha.modes['loc6']
258     mat = [[-1, 0, 0, 0], \
259          [1, 0, 0, 0], \
260          [0, 1, 0, 0], \
```

```
261             [-0, -1, -0, -0], \
262             [0, 0, 1, 0], \
263             [-0, -0, -1, -0], \
264             [0, 0, 0, 1], \
265             [-0, -0, -0, -1], ]
266         rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
267         rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
                mode))
268 
269         # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
270         mode = ha.modes['loc7']
271         mat = [[-1, 0, 0, 0], \
272             [1, 0, 0, 0], \
273             [0, 1, 0, 0], \
274             [-0, -1, -0, -0], \
275             [0, 0, 1, 0], \
276             [-0, -0, -1, -0], \
277             [0, 0, 0, 1], \
278             [-0, -0, -0, -1], ]
279         rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
280         rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
                mode))
281 
282         # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
283         mode = ha.modes['loc8']
284         mat = [[-1, 0, 0, 0], \
285             [1, 0, 0, 0], \
286             [0, 1, 0, 0], \
287             [-0, -1, -0, -0], \
288             [0, 0, 1, 0], \
289             [-0, -0, -1, -0], \
290             [0, 0, 0, 1], \
291             [-0, -0, -0, -1], ]
292         rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
293         rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
                mode))
```

```
294
295     # -1.0 ≤ x1 & x1 ≤ 1.0 & x2 = 0.0 & t = 0.0 & affine = 1.0
296     mode = ha.modes['loc9']
297     mat = [[-1, 0, 0, 0], \
298         [1, 0, 0, 0], \
299         [0, 1, 0, 0], \
300         [-0, -1, -0, -0], \
301         [0, 0, 1, 0], \
302         [-0, -0, -1, -0], \
303         [0, 0, 0, 1], \
304         [-0, -0, -0, -1], ]
305     rhs = [1, 1, 0, -0, 0, -0, 1, -1, ]
306     rv.append(StateSet(lputil.from_constraints(mat, rhs, mode), ...
            mode))
307
308     return rv
309
310
311 def define_settings():
312     '''get the hylaa settings object
313     see hylaa/settings.py for a complete list of reachability ...
            settings'''
314
315     # step_size = 0.01, max_time = 0.3
316     settings = HylaaSettings(0.001, 0.3)
317     settings.plot.plot_mode = PlotSettings.PLOT_INTERACTIVE
318     settings.plot.xdim_dir = 2
319     settings.plot.ydim_dir = 0
320     settings.plot.label.title = "Fuzzy PselD Controller"
321     settings.plot.label.axes_limits = (-0.01, 0.4, -1.1, 1.1)
322     settings.stdout = HylaaSettings.STDOUT_VERBOSE
323
324     return settings
325
326 def run_hylaa():
327     'runs hylaa, returning a HylaaResult object'
```

```
328     ha = define_ha()
329     init = define_init_states(ha)
330     settings = define_settings()
331 
332     result = Core(ha, settings).run(init)
333 
334     return result
335 
336 if __name__ == '__main__':
337     run_hylaa()
```