University of Cincinnati

Date: 10/22/2015

I, Sophia Mitchell , hereby submit this original work as part of the requirements for the degree of Master of Science in Aerospace Engineering.

It is entitled:

A Cascading Fuzzy Logic Approach for Decision Making in Dynamic Applications

Student's name: **Sophia Mitchell**

This work and its defense approved by:

Committee chair: Kelly Cohen, Ph.D.

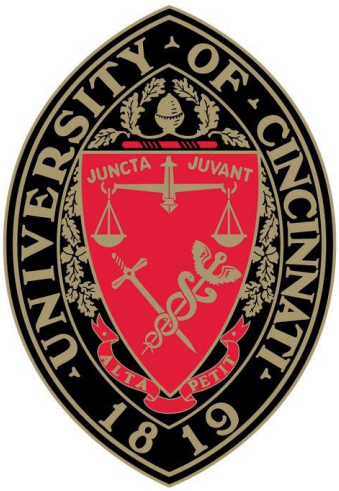
Committee member: Nicholas D. Ernest, Ph.D.

Committee member: Manish Kumar, Ph.D.

Committee member: Grant Schaffner, Ph.D.

19631

A Cascading Fuzzy Logic Approach for Decision Making in Dynamic Applications



Sophia M. Mitchell
B.S. University of Cincinnati

A thesis submitted to the University of Cincinnati
School of Aerospace Systems for the partial
fulfillment of the degree of

Master of Science in Aerospace Engineering

October 22, 2015

Committee Chair: Dr. Kelly Cohen

Abstract

There is growing interest in the effectiveness of emulating human decision making and learning in modern aerospace applications. The following thesis is an examination of several applications in which cascading type 1 and 2 fuzzy logic has been utilized in artificial intelligence and machine learning problems to demonstrate its capabilities. In Fuzzy Logic Inferencing for PONG (FLIP), the effectiveness of cascading type 1 logic is examined as an optimal controller for players in the game of PONG. Robotic collaboration is also developed as the PONG game was expanded into a multi-player option. Precision Route Optimization using Fuzzy Intelligence (PROFIT) examines the use of fuzzy logic as an optimizer in a cascaded algorithmic solution to a modified Traveling Salesman Problem (TSP). The TSP is modified in a way to better mimic a real-life scenario where footprints must be visited instead of simply points, which gives an interesting complexity to the problem. Collaborative Learning using Fuzzy Inferencing (CLIFF) is an extension of the PONG game introduced in FLIP, however a type-2 fuzzy logic toolbox is developed for potential use in development of a robotic coach that could optimize its players to beat an opponent in an application of layered fuzzy learning. Considering the successes associated with these research endeavors, it can be concluded that cascading type 1 and 2 fuzzy logic are both interesting tools that

can further the abilities of intelligent systems and machine learning algorithms.

Acknowledgments

I would like to thank my adviser, Dr. Kelly Cohen, for the enormous amount of patience, guidance and support he has given me on this research endeavor, and for overall being an awesome mentor. I want to thank Dr. Nicholas Ernest for his mentor-ship with genetic fuzzy systems, and for being a co-author on our JAIS paper. Thanks to Dr. Manish Kumar for his mentor-ship with PROFIT and introduction to the traveling salesman problem, and thanks to Brandon Cook for his contributions to the book chapter on FLIP and the modified version of the game. I would also like to thank my parents, boyfriend, family and friends for all of their support and encouragement.

The research in this thesis was funded through grants from several organizations. I would like to thank the University of Cincinnati Women in Science and Engineering summer REWU program, the University of Cincinnati Academic Year REU program, the Ohio Space Grant Consortium, and the National Science Foundation for the financial support of my research. I would also like to thank Dr. Urmila Ghia for being a fantastic director of the REU programs at UC, as well as being an inspiring mentor for female engineers at UC.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Motivation

Intelligent machines are becoming an increasingly normal part of everyday life. Applications can be found everywhere: in transportation with autonomous cars being developed by tech giants such as Google and Tesla, and in the autopilot and air traffic control that keep air traffic safe. Intelligent machines are being integrated into the medical industry in patient diagnosis and image recognition to allow more accurate treatment for patients both in and outside of the hospital. Intelligent systems can even be found in the home, such as in vacuum cleaners, cell phones and computers. Many other less visible applications of intelligent machines exist in big data and the business sector making artificial intelligence (AI) more important to our general society. A major challenge in improving the effectiveness of AI includes developing situational awareness that is capable of decision making in a real world environment that is uncertain, time critical, and dynamic. Many systems that excel in decision making in a controlled, simplified environment when presented with the sheer number of

variables that humans easily take into account become so complex they often take too long for effective real time use. Therefore the purpose of this research is to explore a more efficient approach to artificial intelligence that may allow these systems to work more effectively in the complexities of a real world environment through emulating human reasoning.

Along with being able to manage decision making in the real world, moving forward intelligent machines will need to be developed to work both interactively with humans (for example in space or on search and rescue missions) as well as completely autonomously (i.e. hazardous material cleanup at Fukushima-like disasters, deep space exploration and colonization, dangerous search and rescue, etc). Both situations present challenges, as systems working autonomously need to be capable of solving unforeseen problems (or at least managing them until a human can intervene) while collaborative systems will have to be capable of understanding things like task allocation, what its own capabilities are, and when and how it should ask for assistance as a project is being completed.

1.2 Research Objective

This thesis focuses on developing improvements on one method of artificial intelligence, fuzzy logic, for systems that don't interact with humans. This is a challenging objective, as the applications for this sort of system make it critical for the "'brains"' of the system to be robust (trustworthy), as well as have the capability to problem solve and learn how to work with its environment without the checks that human interaction provides. More specifically, the objective of this thesis is to explore applications of cascading fuzzy logic systems, and hopefully determine if this method can increase both the effectiveness and

efficiency of intelligent machines.

Three applications are presented in the following chapters that showcase varied uses of cascading fuzzy logic, as well as form an argument for the strengths and weaknesses of type 1 and type 2 fuzzy logic as a whole. The first application is Fuzzy Collaborative Robotic Pong (FLIP) which uses a complex system of cascading type-1 fuzzy logic as an optimal game controller. Precision Route Optimization using Fuzzy Inferencing (PROFIT) is the second application described in this thesis. It examines the use of cascading fuzzy logic with a genetic algorithm to create an optimal result in a modified traveling salesman problem. The final application of cascading fuzzy logic in this thesis is Collaborative Learning using Fuzzy Inferencing (CLIFF), which cascades type-1 and type-2 fuzzy logic to create an intelligent and dynamic robotic coach.

Chapter 2

Literature Review

Since its introduction in the 1960's and resurgence in the last 20 years, the benefits of fuzzy logic has been studied in a variety of applications.

2.1 Type-1 Fuzzy Logic

Type-1 fuzzy logic is known for producing results that are near-optimal even though it's a heuristic, computationally light method. Due to this characteristic, fuzzy logic becomes quite useful in applications that need to be scaled for large problems. An example of this can be found in work by Shi and Eberhart [38], where a fuzzy system is used to dynamically adapt the inertia weight in a particle swarm optimization (PSO) algorithm. Using fuzzy logic in this application was found to increase the performance of the PSO over more commonly used methods such as linear and evolutionary methods for determining weights. It was also found that the performance of a PSO algorithm that used dynamic fuzzy weighting did not degrade as much as with other methods when the problem was scaled, showing how the

computational lightness of fuzzy logic can be of use in large scale problems.

An example of the usefulness of fuzzy architecture can be found in work presented by Berenji [3]. In studying reinforcement learning Berenji presents a discussion on learning algorithm use across a wide range of applications, and shows that algorithms that can build upon knowledge from an experienced operator perform better and learn faster than networks that are trained from scratch. This finding is important because fuzzy systems are naturally created from expert knowledge of a system, so based on the findings of this study, creating a method to iteratively improve the performance of a fuzzy system will create a better controller.

2.2 Type-2 Fuzzy Logic

The use of type-2 fuzzy logic has been highlighted by several studies showing its effectiveness in mitigating controls errors produced by environmental and instrumentation noise.

A comparison of type-1 and type-2 fuzzy systems can be found in research conducted by Karnik and Mendel [18], in which a type-1 system is created and then a type-2 system is designed from it based on information about noise in the training data. While both systems were manually tuned (which opens the door to some loss in performance), there was substantial evidence towards the conclusion that the type-2 system could outperform their type-1 system in high-noise controls applications.

A later study of type-1 and type-2 controllers by Sepulveda, Castillo, et. al. [37] compared effectiveness of controllers that were automatically adjusted in high-noise application performance. Experimental results showed that the type-2 fuzzy logic controller outper-

formed the type-1 controller in terms of overshoot error and settling time. In a time-series prediction problem also conducted by this study, the type-2 system had a smaller prediction error than an equivalent adaptive neuro-fuzzy inference system (ANFIS) which uses type-1 logic. Since type-2 fuzzy logic has more parameters and design degrees of freedom it is often overlooked as type-1-based systems such as ANFIS generally perform well and are easier and faster to create. However, this study shows a "remarkable" difference between the performance of a well-made type-2 system and the ANFIS system in high-noise situations (which is extremely common in if not wholly representative of the real world), which gives reason to further research type-2 systems and potentially find an easier and more effective way of creation and implementation.

Expanding on the idea of type-2 systems being an effective controller for noisy time-series forecasting, Liang and Karnik [24] present research that uses survey-based type-2 fuzzy logic for connection admission control in networks. Fuzzy logic is useful in this application as it simplifies the combination of many inputs, such as voice, video and time data, creating a system that linguistically determines connection admission. Using survey based type-2 logic in this application makes sense as it allows the controls system to be more intelligently created based on the experience of many experts. Since the type-2 system can be used to forecast the future input rate of traffic, the system can dynamically allocate bandwidth as well as connection admission control in the system. Liang and Karnik demonstrate the usefulness of type-2 logic in this situation, as the features presented by their type-2 system could only be matched by a set of type-1 systems or a very complex type-1 system. What's more, based on research previously mentioned in this literature review, these type-1 systems might not even perform as well as a type-2 system.

We have seen so far that type-1 systems can be improved through combining it with a neural system to improve performance in noisy environments (such as ANFIS), and that a type-2 system can still out-perform this combination. What happens if a type-2 system is combined with a neural network to create a Type-2 Fuzzy Neural Network (T2FNN)? Wang, Cheng and Lee [39] explore this idea by creating a T2FNN algorithm consisting of an interval type-2 fuzzy system as the antecedent and a two-layer neural network as the consequent. A genetic algorithm (GA) based dynamic training method was also developed to optimize the T2FNN system, so the optimal learning and spread rates could be determined for this new type of algorithm. This research gave empirical evidence that the T2FNN performed better than an equivalent type-1 system (a T1FNN), and that it was possible to determine an optimal learning and training algorithm for the system. This second part is especially important, as having a method for dynamic optimal training means the T2FNN can be applied and adapted to many different types of problems.

Type-2 fuzzy logic has shown evidence of high performance in complex navigational controls problems. A study conducted by Martinez, Castillo and Aguilar [29] employs a type-2 fuzzy logic tracking controller mixed with a genetic algorithm (for optimization) that takes into account the kinematics and dynamics of the autonomous vehicle it is controlling in order to stabilize it as it traverses a known path. This method of using type-2 logic for stabilization allows for the controller to act optimally without having to do the many complex equations associated with the kinematics and dynamics of a (in this case unicycle) vehicle. While more computationally complex than a type-1 system, it is architecturally less complex, as well as less computationally complex than a more traditional controller that takes all of the relevant motion and dynamics equations into account.

Another application of type-2 logic in complex physical controls problems is presented in work by Hagras [14]. In his paper, Hagras discusses his novel reactive architecture for a hierarchical fuzzy system, which simplifies the design of the robotic controller and reduces the number of rules in the overall system by about 64%. This allows for real-time operation of the interval type-2 fuzzy logic controller (IT2FLC), meaning it can also mitigate real-time uncertainties as they arise. Due to the intended noisiness of the simulation (to emulate a real world environment) the IT2FLC architecture was able to outperform both a type-1 fuzzy system as well as a hierarchical type-1 system.

In later chapters of this thesis the creation and development of a type-2 system is examined and discussed. In creating this system it was quickly found that two main points of research were necessary: 1) an examination of methods for type-2 fuzzification and defuzzification and 2) finding a publication with a simple system that could be replication for use in verification of the type-2 programming algorithm that I created. The following 2 paragraphs address the research in these two areas.

Reading into the literature on type-2 fuzzy logic systems, it is clear to see that most (if not all) of these systems use interval type-2 fuzzy logic systems (IT2FLS), a special type of general type-2 fuzzy logic system (GT2FLS) where the third dimension of the footprint of uncertainty is uniform. Using this special case of type-2 logic has proved to be useful, as the type-reduction method for an IT2FLS is much less complex than methods for GT2FLS. [31]. Some research has uncovered methods that attempt to simplify type-reduction strategies for GT2FLS. An example of this research can be found in work produced by Liu [26]. In his work, Liu created a type-reduction strategy based on centroid type-reduction on each alpha plane of a GT2FLS. From this he was able to simulate that the result converged on

a defuzzified, crisp value much more quickly than the exhaustive computation approaches presented by Mendel in [31]. His research showed great promise in expanding the applicability of GT2FLS; however, in order to use it in this research, we would have to either get a hold of (or even re-create) Liu's type-reduction work - an entire research project in itself. Therefore, it was determined that time would be better spent for this project by using an IT2FLS as it combines the benefits of using type-2 fuzzy logic with type-reduction methods that have more evidence behind them (as well as open source code examples from Mendel's resources). Switching to a GT2FLS in the future may still be possible for this project, and comparing effectiveness may prove extremely interesting, however for the sake of time (and sanity) it was decided that an IT2FLS would be the best choice for our current application.

In researching applications of interval type-2 fuzzy logic systems (IT2FLS) one paper stood out not only as a good example of comparing the performance of types 1 and 2 fuzzy logic, but also in reproducibility. This was important because while it was decided not to re-invent the wheel in terms of creating a type-reduction method by using an interval instead of general system, we still would need to create our own type-2 fuzzy logic toolbox. Therefore, finding a research paper with a lot of detail in both simulation development and results would allow us to verify our own system against found results, thereby verifying our work. For this, a study by Wu and Tan [40] was chosen. This article gives a comprehensive analysis of the development and application of an interval type-2 fuzzy logic controller (IT2FLC) in controlling the depth of liquid in a drum. Since the dynamics equations for the liquid level system were given in the paper as well as an in-depth look at the way the type-1 and type-2 fuzzy systems were designed in their study, it was easy to replicate the work in this paper with our IT2FLS. While more will be discussed about using this problem as a benchmark

later in this thesis, it is important to note that the data in this paper suggests a type-2 system that performs better than a type-1 system in an optimal control application. This conclusion caused skepticism, as this application has very little noise which is the major weakness of a type-1 system compared to a type-2. In non-noise control applications a type-1 system has a lot of evidence of being a universal optimizer. Therefore along with using this paper as a benchmark problem, we were able to further explore the reasoning behind the conclusion of this paper. This will be discussed later in this thesis as well.

2.3 Cascading Fuzzy Logic

Several research studies have been found that take a different approach to the challenges of using fuzzy logic with highly complex applications. Instead of choosing one type of logic or method over another, cascading logic is used to simplify fuzzy systems, or gain some of the benefits of other types of systems such as neural networks.

All of the applications presented in this thesis use cascading fuzzy logic in some form, as we believe this type of logic is capable of significantly simplifying applications with large amount of situational awareness is necessary. Examples of this simplification exist in the literature, for example research by Lin, Lee and Pu[25] cascaded a fuzzy system with a neural network in a satellite sensor imaging application. Due to the complex nature of terrain images, using traditional methods of image processing becomes far too computationally heavy to identify features and provide classification within a reasonable amount of time and effort. Therefore, a cascaded system was created that condenses and classifies general measurements before several layers of a neural fuzzy network works to fine tune results.

While their system worked well in simplifying the system while preserving its effectiveness, an interesting side effect of cascading with a fuzzy neural network was also noted. In initial testing with a singular fuzzy neural network, the network parameters would often orient themselves to existing patterns within the training data. While this is not generally desired, it is an effect that is known to occur with neural networks. When comparing this to the trained cascading fuzzy neural network, it was found that the cascaded system not only generalizes well, but also it avoids this trend, allowing for a system that is less effected by the training data. This was a really interesting find, as it suggests that a cascading architecture can both improve efficiency and preserve (or possibly improve) the effectiveness of a system compared to a singular version.

Another example of using cascading logic to simplify a system with both fuzzy and crisp logic can be found in research presented by Heider and Drabe [15]. Their research cascades two genetic algorithms with a fuzzy system to create a method for more optimally creating fuzzy control systems. Heider and Drabe take a unique approach to tuning their fuzzy system by having multiple genetic algorithms layered within the system: one to optimize the membership function parameters and another to optimize the rule base. While this approach took longer than a traditional genetic fuzzy algorithm to converge with training data, it was found that in general the resulting systems were more effective than those produced by a non-cascading genetic fuzzy algorithm. While the idea of cascading a genetic algorithm with fuzzy logic is similar to what is presented in Chapter 5 of this thesis (PROFIT), this application differs in that the genetic algorithm alters the fuzzy system that controls the output rather than interacting with the output of the system itself. While it's possible adopting this multi-genetic algorithm architecture could produce better results, we were

also concerned with optimizing solution time in PROFIT. Therefore it would not have been desirable to use a method that slows down the solution convergence time.

In research by Berenji and Khedkar [4] a novel method of tuning a fuzzy logic controller based on reinforcements is explored within a learning environment. Berenji and Khedkar determined that using neural network representations of fuzzy control rules allowed for faster development and faster learning than a traditional neural network. This was a result of both getting the focused tuning the gradient descent of neural networks adds to a learning algorithm, as well as globally tuning fuzzy rules instead of locally tuning them as is usually done. Using this new architecture, Berenji and Khedkar claim that others may be able to model the knowledge of an experienced operator in a controls process using linguistic terms and later refine this model to more quickly create an optimal controller.

Several examples were found where fuzzy logic is cascaded with more fuzzy logic, much like the system architecture in FLIP and CLIFF described later in this thesis. One example is research conducted by Russo and Ramponi [35] that examines the benefits of using two cascaded FIRE (Fuzzy Inference Ruled by Elseaction) filters in smoothing noise in image processing applications. They determined that a cascading fuzzy system was quite beneficial for this application because it proved to be easier to design than a singular fuzzy system due to the increased number of rules necessary for stronger noise cancellation, while also being less computationally heavy. In several cases, the cascading FIRE system was found to produce better results than a singular FIRE system, which suggests that in the right circumstances and applications a cascading logic system should be preferred over a singular logic system.

In research by Mahapatra, Nanda and Panigrahy [27] a cascaded fuzzy inference system

(FIS) is applied to a system for predicting river water quality in India. A cascaded fuzzy logic system was used in this case due to its simplicity compared to a MIMO system which can also have more uncertainty. Mahapatra and his team created a water quality indexing system based on cascading fuzzy logic that was able to take into account the range of factors that is generally used in determining water quality while giving its result. Through comparison it was determined that the cascading fuzzy logic system gave better results than a traditional fuzzy system. The levels of cascading also allowed for the researchers to gain better insight on the effects of individual factors in final water quality output, which they hope will be an effective tool for environmental policy.

Autonomous vehicles have become a popular point of research recently, as it seems in the next half-century driving cars may become a thing of the past. Contributing to this trend, Mar and Lin [28] applied a cascading fuzzy logic system to the car-following collision prevention control system in an autonomous vehicle to determine if this may be a more effective and efficient method. The fuzzy system was cascaded into two layers: one layer for managing velocity and one for controlling acceleration. The proposed device worked very well in avoiding collisions, showing clear evidence of providing a "'reasonable drive"' meaning the spacing it created between itself and the next vehicle made sense, a "'comfortable drive"' meaning the vehicle didn't jolt around accelerating and decelerating rapidly, and a "'safe drive"' because it didn't run into the other vehicle. Overall this study was a good look at how even a small (two layer) cascading system can make an effective controller for complex applications where a lot of situational awareness is necessary.

2.4 Fuzzy Applications in Gaming

Since this thesis applies cascading fuzzy logic to a version of the Atari game Pong, it only made sense to find research of fuzzy logic applications in other games. Li, Musilek and Wyard-Scott [23] introduce the idea of using fuzzy logic to enhance interaction between agents within a gaming environment. They cite the ability for fuzzy logic to handle complex control problems while retaining details and having low computational cost as reasons for wanting to apply fuzzy logic in this manner. This research has a lot in common with the project outlined in Chapter 4 of this thesis, as fuzzy logic is being used in an attempt to simplify collaboration between autonomous agents. While cascading logic wasn't considered, the researchers did have success with applying their fuzzy system to agents within a game to create a more natural and inclusive environment for the players. Behaviors of these agents were reported to be more natural as fuzzy logic allowed them to take into account more aspects of what both the player and other autonomous agents within the game were doing. Some level of strategy arose within the agents which is very similar to some of the strategy that arose with FLIP.

Looking into the physical controls associated with gaming, Dadios and Maravillas [9] take on the challenge of applying a fuzzy logic controller to soccer playing robots. In their system, fuzzy logic was applied to control the motion of the players, generating the velocities that the motors in each player would use to play the game. They also studied using fuzzy logic for cooperative behavior between players, to determine if fuzzy logic would be an effective means for creating a cooperating team. Experimentally the fuzzy controller was difficult to tune correctly at first, but once tuned it was capable of obstacle avoidance, as well as determining

role selection between players based on real time game play. This was an interesting project to compare to this thesis, as there was success even without cascading logic. It is important to note, however, that the major drawback identified by Dadios and Maravillas was that the situational awareness could be improved, however this would greatly increase the complexity of their fuzzy system. Using cascading fuzzy logic could be a solution for their issue, however it is not mentioned in the paper.

2.5 Genetic Fuzzy Systems

It was found that genetic algorithms are used alongside fuzzy logic for two general purposes: learning applications and tuning of fuzzy logic systems. While research wasn't found that applied a genetic algorithm to a solution and then cascaded fuzzy logic as well as described in Chapter 5 of this thesis, reading the literature on the two major applications of genetic algorithms within fuzzy logic still gave an idea of where the ideas developed in this thesis stand relative to the current literature.

While a genetic algorithm could not be found tuning type-2 fuzzy logic in learning applications, it is always useful to take into account the current methods to see how work such as CLIFF (described in Chapter 6) relates to the current body of intelligent learning research. Research by Herrera, Lozano and Verdegay [17] presents a method of learning where a fuzzy system is almost completely constructed and tuned using a genetic process that is broken down into three steps. The first step is to create the fuzzy rules for the system using an iterative genetic rule learning approach, which was found to greatly optimize the process of creating fuzzy rules due to a reduction in the solution space. The second step includes using

expert and fuzzy rules to optimize the rule base and remove redundancy, and the third step optimizes the learning database through adjustments of the membership functions within the fuzzy system. This method was found to work quite well, however it would be interesting to see how it compares (both in output and runtime) to currently established learning algorithms.

Another application of genetic fuzzy logic to learning algorithms was found in research by Russo [36]. His research examines GEFREX, a hybrid approach to supervised learning based on a genetic-neuro algorithm. It was found that GEFREX produces very effective fuzzy systems from knowledge of input and output systems, and is quite efficient in making these fuzzy systems due to the method's compression of learning patterns. In a way this method is a cascading algorithm as it first creates the fuzzy system using a genetic algorithm, then applies this result to a neuro hill-climbing operator before inserting this into the genetic population. Once the genetic algorithm which is peppered with these neuro results converges the resulting fuzzy system was found to be very effective. In a way this is similar to the layered learning of CLIFF (Chapter 6) because it uses multiple methods to manipulate a fuzzy system, creating a form of system learning.

Research by Castillo, Gonzales and Perez [6] introduces an interesting idea of using simplicity criteria in genetic fuzzy learning algorithms to reward simplicity within their fuzzy algorithms. Their paper describes the creation of SLAVE (Structural Learning Algorithm in Vague Environment) which produced two types of rule sets based on the type of simplicity criteria: more descriptive or more discriminant. However the addition of any type of simplicity criteria was found to improve results for SLAVE making it a more effective system. It was interesting to note similarities and differences in this approach to the cascading ap-

proach for simplifying learning algorithms with fuzzy logic. In both cases the solution space is reduced, however the cascading method removes extreme solutions by not making them desirable to the next layer of logic whereas the simplicity criteria in SLAVE forces it to throw bad solutions out.

A common application of genetic algorithms to fuzzy systems is in tuning of rules and membership functions during the creation of the fuzzy system. While this is similar to having the system "'learn"' these applications differ from learning algorithms as they generally do not run once the controller has been created, so if something in the environment changes they do not automatically alter themselves. An example of this can be found in research by Pratihar, Deb and Ghosh [34] in which a fuzzy controller is optimized using a genetic algorithm to reduce travel time of a robot to its destination (motion planning) while simultaneously avoiding obstacles. The created system was found to be extremely effective, as in all test cases it found obstacle-free paths which were also faster routes to the destination than the fuzzy logic controller that was defined by humans. This conclusion was also found in a very similar study by Arslan and Kaya [2], which examined the effects of adjusting the shapes of their membership functions with genetic algorithms on the output of the fuzzy system. Again, the GA-altered fuzzy system was found to be a more optimal solution than a fuzzy system created by humans.

2.6 Fuzzy Logic in Traveling Salesman Problems

Due to its heuristic nature, fuzzy logic is a sensible method to use in solving traveling salesman problems. As the number of targets that must be visited grows, the fuzzy system

doesn't necessarily have to, allowing it to be more easily applied to large and complex traveling salesman problems. Feng and Liao [13] examine this application in their research, as they develop a hybrid evolutionary fuzzy learning algorithm that is optimized to work with large-scale traveling salesman problems (LSTSPs). Through combining adaptive fuzzy C-means, a min-max merging concept, simulated annealing and an efficient particle swarm optimization they were able to create an algorithm that achieves better results than other learning methods both in route quality and computing time. This is an interesting research study to compare to PROFIT, which is outlined in Chapter 5 of this paper, as both studies cascade various types of algorithms and fuzzy logic to gain advantages from each one. It was encouraging to find other research taking this approach and having great success with it, even if it was with different types of fuzzy logic and other algorithms.

A lot of traveling salesman problem solution methods focus on ant colony optimization (ACO) as more research has shown the advantages of emulating ant-like search techniques in TSPs. Collings and Kim [8] use a decentralized peer-to-peer approach in implementing an ACO method that also uses a fuzzy logic controller to control convergence and allow the system to better scale for use with a large number of targets. The results of this method demonstrate that using fuzzy logic with a distributed ACO produces better results both in time and resulting path as opposed to traditional ACO methods. This research is similar to the method developed in Chapter 5 of this thesis with PROFIT because it is taking a TSP solution and optimizing it with fuzzy logic, much in the way the Genetic Algorithm with Fuzzy Optimization in PROFIT works. Another example of using fuzzy optimization with an ACO to produce a better output can be found in work conducted by Khanna and Rajpal [20] in which their fuzzy ACO system is used to reconstruct curves from point clouds.

This was an interesting system to read about because these point clouds could result in a single or multiple TSPs depending on each different cloud. For example the cloud making a bold cross was shown as a multi-TSP solution while a cloud to make a simple "'c"' curve is singular. The system proved to be robust to noise, non-uniform distribution of points within the cloud and for missing data, which provides a decent argument that systems constructed from multiple methods can gain the benefits of each system while still keeping, or even improving, robustness.

Recently, genetic fuzzy algorithms have been cascaded and applied to TSPs in order to create more effective and efficient TSP solvers. Recently several papers have been published in this area giving tremendous insight into the capabilities of a cascaded genetic fuzzy algorithm, especially in multi-TSP problems.

Generally in multi-TSP applications, targets are divided between agents according to individual preference and competition between agents to get the most preferred targets. In recent work by Dr. Ernest, et. al., [12] an alternative approach is taken, where agents decide which targets to attack according to collective preferences through a new program called LETHA (Learning Enhanced Tactical Handling Algorithm). The main idea behind LETHA is that since each agent is equipped with different accessories, this will encourage decisions that bring the greatest benefit to the group rather than to each agent. In certain scenarios such as fighting off an airborne dogfight, this strategy was shown to lead to more cooperation within the group of agents, and overall a better outcome for the group as a whole. Given a worst-case scenario, the inclusion of LETHA made the system significantly more robust, allowing for an increased safety margin in mission planning as well as increased damage to the enemy force (in this scenario). In order to train the intelligent controllers using LETHA,

a simulation space called HADES (Hopological Autonomous Defend and Engage Simulation) was created, and described by Ernest et.al. [10]. HADES allows LETHA to be examined at a high level, as well as can train the agents using LETHA to apply to high-fidelity models. By creating challenging scenarios with many different situations and characteristics, it was possible to test LETHA's robustness as well as display the deep learning that occurs within the cascaded version of the system.

The cascading logic developed for use with LETHA is examined in another recent publication by Ernest, et. al. [11], where the cascading system using LETHA is compared to LETHA working on the problem alone. The cascading logic used is a Genetic Fuzzy Tree (GFT) which allows for more situational awareness to be taken into account while still being computationally reasonable. While it was determined that LETHA alone could potentially perform better, using LETHA with a GFT was considered to be a more efficient and scalable solution. When sent on many different training missions, the cascading structures of the GFT were able to be easily tuned for the new environment, showing the robustness of the system and resiliency to uncertainty and randomness it may encounter in real world scenarios. Overall, this application is a fantastic example of how cascading logic can allow for optimal systems to become more scalable, effective, and useful in real world applications with a high amount of situational awareness.

2.7 Contributions

The research for this thesis falls into three related projects, each of which develops along three lines that further our understanding of the use of fuzzy logic as a controller and an

intelligent learning tool in artificially intelligent systems.

FLIP introduces a development in cascading logic that was novel (as known by the author) at the time of its creation. The idea of using 11 cascading systems to overly simplify a very complex control and AI problem took something that was originally simple (just cascading two, maybe three systems) and showed how expanding on this idea could help solve even larger problems than anticipated.

Further expanding on the theme of cascading logic in this thesis is PROFIT, where a genetic algorithm was cascading with fuzzy logic. The created system in our research differs from the aforementioned genetic fuzzy systems in how the genetic algorithm and fuzzy logic system are connected. Having a layer that is fully a GA and then fully a fuzzy system is unique, as usually the two are mixed in a way to optimize the other (usually a genetic algorithm optimizing fuzzy rules, membership function sets or both).

The final major contribution of this thesis is an exploration of type-2 fuzzy logic, and its potential application in robotic learning. By performing a robust comparison of type-1 and type-2 systems this thesis adds to the conversation the idea that type-2 systems have use in certain applications, however cannot necessarily replace a type-1 system in all applications. In performing this comparison the researcher also had to create their own type-2 fuzzy logic toolbox in MATLAB, which, while not necessarily the focus of that project, could definitely be considered another contribution of this thesis.

Chapter 3

Methodology

3.1 Fuzzy Logic

Fuzzy logic allows for classification of variables in a program for more human-like reasoning. Normally in programming, one would use binary logic, where things are either zero or one, on or off, black or white. In real world decision making, using a binary system for decision making generally doesn't produce an optimal solution, as nothing in the universe is completely one thing OR another, but on a continuum. Fuzzy logic takes this continuum into account, therefore allowing a program to make decisions in between definite boundaries. So if the binary boundaries are zero and one, a fuzzy output could be anything between these two numbers. Another way to think about fuzziness is through considering the spectrum of visible light. While it's obvious that the color orange appears on the spectrum, there is no certain point where orange begins and ends. What's more, if 100 people were to point to their ideal orange on the spectrum, "'orange"' would become several places on the spectrum with varying degrees of belonging to red and yellow, and each one would be correct.

There are three main parts of a fuzzy logic system: fuzzification which takes a crisp input and makes it "'fuzzy"', linguistic reasoning which uses these fuzzy inputs to determine a fuzzy output, and defuzzification which turns the fuzzy output into a crisp output that a program can then use.

3.1.1 Fuzzification

All fuzzy systems begin with an observed "input" variable that is taken into the system. The input is assigned a degree of membership to one, or more, fuzzy sets known as the input membership functions. This process is called "fuzzification".

To get a better idea of what is going on during fuzzification it is important to have a good understanding of membership functions. A membership function ($\mu_{\tilde{A}}$) is a function that defines the amount of membership (x) an input or an output has to a fuzzy set ($\tilde{A}$). So the degree of membership of x in $\tilde{A}$ would be written as $\mu_{\tilde{A}}(x)$, and determined by finding the intersection of x and $\mu_{\tilde{A}}$ as shown in Figure 3.1. Fuzzification of an input variable then means to find the degree of membership each variable has to each input membership function in the fuzzy system.

In the case of multiple membership functions, a variable will have a degree of membership to each function. This relationship is visualized in Figure 3.2.

Membership functions can have many different shapes depending on user preference, but generally are triangular, trapezoidal or follow a Gaussian curve. Custom shapes are possible but almost never seen in use.

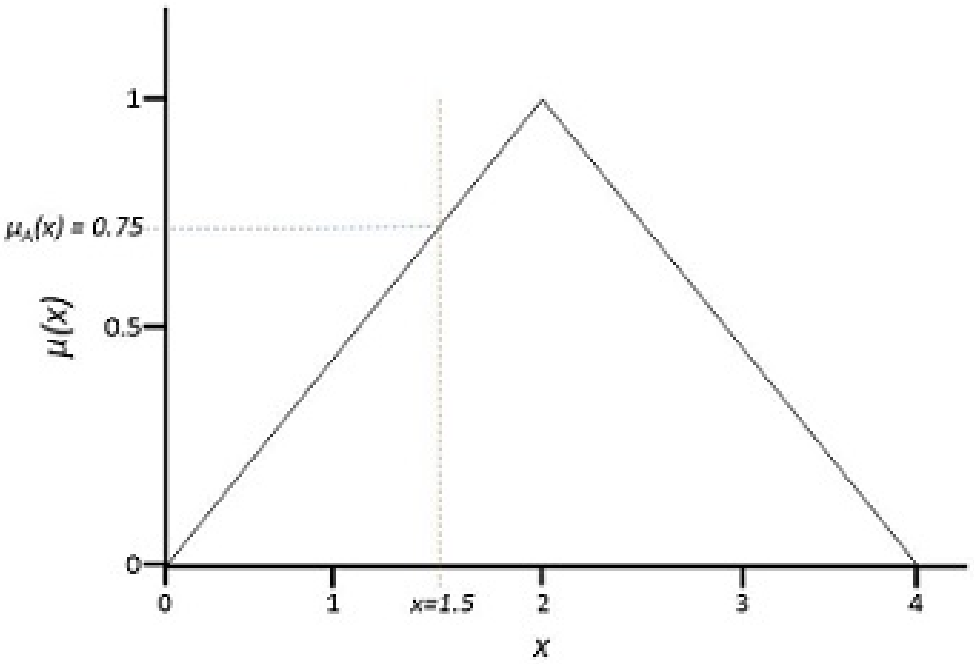


Figure 3.1: Example of a simple variable fuzzification with one membership function.

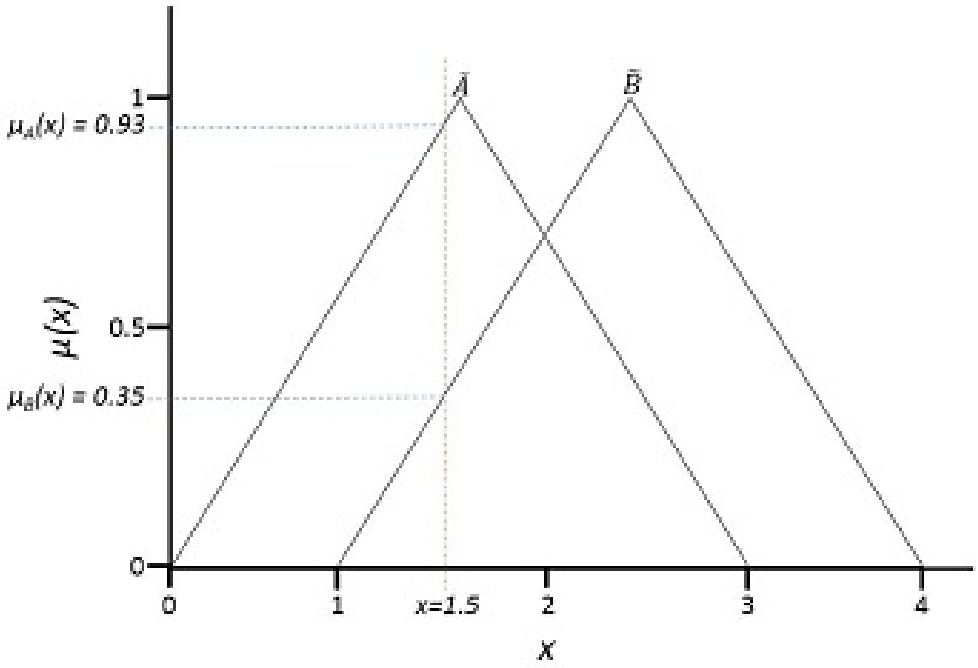


Figure 3.2: Example of a simple variable fuzzification with two membership functions.

3.1.2 Fuzzy Inferencing

The core of a fuzzy system is the linguistic mapping between input and output membership functions that allows the whole system to "'reason"'. This mapping is created through a set

of rules that connect input membership functions to output membership functions using logic operators, which are defined with respect to fuzzy systems in Table 3.1. A major feature of fuzzy logic is its ease of use due to the intuitiveness of rule creation using these operators. Creating rules that read "'IF this AND that THEN output"' makes linguistic sense, and allows the user to easily create and modify rules in a way that makes sense. While the creation of these rules occurs when the fuzzy system is set up, the rule base isn't used until after all input variables are fuzzified.

Table 3.1: Commonly used fuzzy rule operators

Operator	Logic Symbol	Equation
AND	$\cap$	$\mu_{\tilde{A}\cap\tilde{B}}(x) = min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$
OR	$\cup$	$\mu_{\tilde{A}\cup\tilde{B}}(x) = max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$
NOT	$\mu_{\bar{A}}(x)$	$1 - \mu_{\tilde{A}}(x)$

Fuzzified variables are run through the rule base, which creates fuzzy outputs. An example of a simple fuzzy system can be found in Figure 3.3 , where a fuzzy controller is being used to determine if a heating/cooling unit needs to heat or cool a room. In this example, the input for the system is the current room temperature which is linguistically mapped to the output that tells the heating and A/C unit what to do. An example of fuzzification and defuzzification can be found in Figure 3.3, where input membership functions as very cold (VC), cold (C), neutral (N), hot (H) and very hot (VH), and the output membership functions include fan (F), low fan (LF), nothing (N), low heat (LH) and heat (H). The rule set mapping the logic of this system can be found in Table .

All of the fuzzy output membership functions are then combined into what's called a total

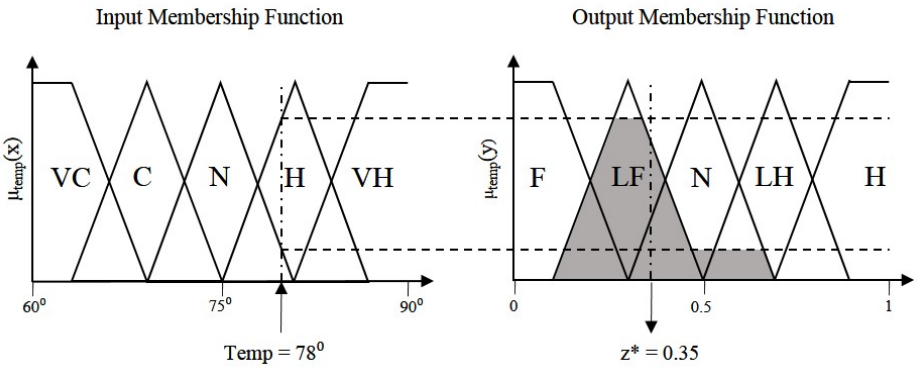


Figure 3.3: Fuzzy thermostat control system example

Table 3.2: Rule mapping for fuzzy thermostat control system example

IF (Very Cold) THEN (Heat)
IF (Cold) THEN (Low Heat)
IF (Neutral) THEN (Nothing)
IF (Hot) THEN (Low Fan)
IF (Very Hot) THEN (Fan)

output membership function ($\mu_{\tilde{A}'}(x)$), which is the fuzzy output of the linguistic reasoning part of a fuzzy inference system.

3.1.3 Defuzzification

Defuzzification is the process by which a fuzzy inference system (FIS) translates a fuzzy output into a crisp output. This is the final step in a FIS, as this crisp output can be used in a given controls application. There are many methods that exist for defuzzification, as can be seen in Table 3.3. The defuzzification method used in a FIS does effect the crisp output, so one must carefully consider the pros and cons of the method being used when creating a fuzzy system. Generally methods are chosen based on consideration of the shape of the input and output membership functions for that FIS as well as the computational complexity of

each defuzzification method.

Table 3.3: Common defuzzification methods[30]

Method	Arithmetic Equation
Centroid	$y_c(x) = \frac{\sum_{i=1}^{N} y_i \mu_B(y_i)}{\sum_{i=1}^{N} \mu_B(y_i)}$
Center of Sums	$y_a(x) = \frac{\sum_{l=1}^{M} c_{Bl} a_{Bl}}{\sum_{l=1}^{M} a_{Bl}}$
Center of Average	$y_h(x) = \frac{\sum_{l=1}^{M} \bar{y}^l \mu_{Bl}(\bar{y}^l)}{\sum_{l=1}^{M} \mu_{Bl}(\bar{y}^l)}$
Modified Center of Average	$y_{mh}(x) = \frac{\frac{\sum_{l=1}^{M} \bar{y}^l \mu_{Bl}(\bar{y}^l)}{\delta l^2}}{\frac{\sum_{l=1}^{M} \mu_{Bl}(\bar{y}^l)}{\delta l^2}}$
Center of Sets	$y_{cos}(x) = \frac{\sum_{l=1}^{M} c^l T_{i=1}^{p} \mu_{F_i^l}(x_i)}{\sum_{l=1}^{M} T_{i=1}^{p} \mu_{F_i^l}(x_i)}$

In this thesis, the centroid defuzzification method is utilized. This is the same defuzzification method that is depicted in Figure 3.3 for the heating & A/C example. While it is a bit more computationally heavy than other methods due to the calculation of the union of the included membership functions, it is more effected by the shape and size of all included membership functions. This sensitivity makes the centroid defuzzification give a more responsive output to changes in the fuzzy input making it a very popular defuzzification method.

3.1.4 Type-2 Logic

Type-2 fuzzy logic brings uncertainty into the membership functions of a fuzzy set, thereby creating the possibility of more noisy measurements to be quantified. Due to the extended level of "'fuzziness"', type-2 fuzzy logic allows for a certain level of vagueness that is common in human decision making yet impossible even with type-1 fuzzy logic.

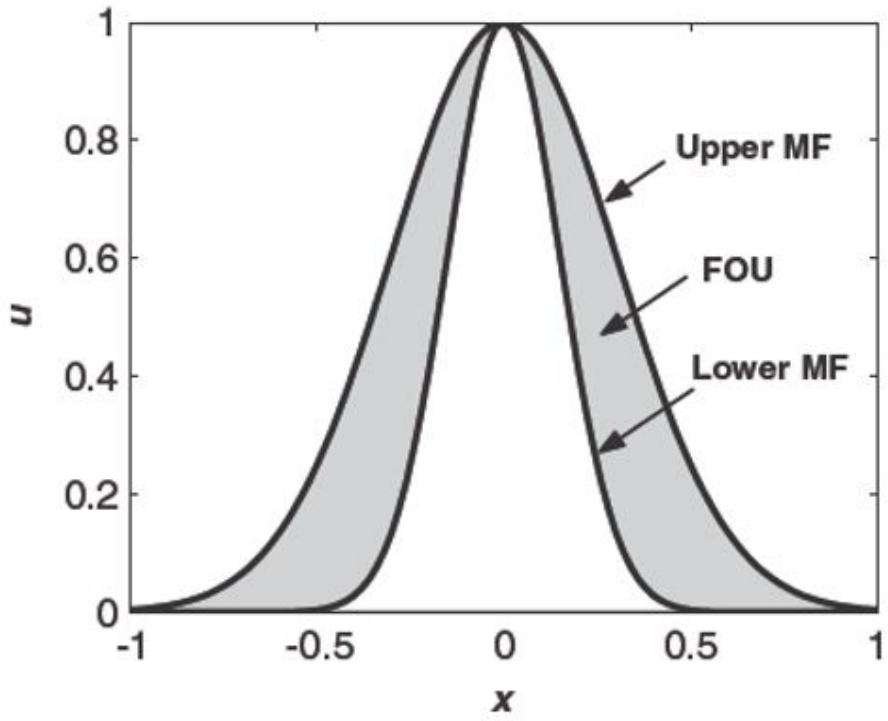


Figure 3.4: Parts of a Type-2 membership function

A type-2 fuzzy membership function has several parts as can be seen in Figure 3.4: an upper membership function, primary membership function, lower membership function and a FOU which stands for Footprint of Uncertainty. The primary membership function determines the central positioning and shape of the membership function, just like a type-1 membership function does in type-1 logic. The upper and lower membership functions are determined by two decisions made by the creator of the system. The first decision determines if the function will vary by its mean (wide top and narrow bottom, lower image, Figure 3.5) or by its standard deviation (wide bottom and narrow top, upper image, Figure 3.5). The second decision is how much this variance will be. A larger variance allows for better variable coverage and therefore a more smooth control, however too large of a variance will confuse the system as all of the rules will run together. It is therefore necessary for the researcher

to determine arbitrarily or through testing what this variance should be.

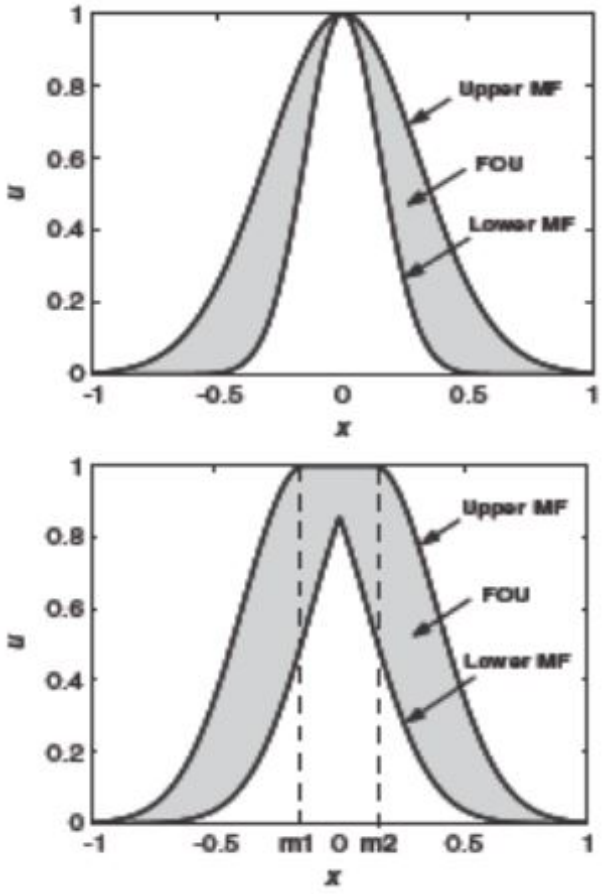


Figure 3.5: Examples of type-2 membership functions when variance is based on standard deviation vs mean

After the upper and lower membership functions have been determined, the space between them is known as a Footprint of Uncertainty, or FOU. An FOU is quite useful in a fuzzy inference system (FIS) as it is what makes the membership function act as more than just a thick area. If one were to think of a type-2 membership function in the third dimension, the FOU is an area that would stick out of this paper at the reader. A visualization of this can be found in Figure 3.4. What makes this so sophisticated is that while the FOU can all have the same value (meaning the area popping out at the reader is all at the same height; this is called an interval type-2 FIS), it can also have a varying amount of

value, meaning the 3-dimensional FOU can be shaped. It is possible for both the primary membership function and the shape of the FOU to be based on a Gaussian distribution. The ability for the system to do this is extremely useful, as now the closeness each value has to its primary membership function has meaning, which allows for extremely complex yet efficient and effective systems to be created. This functionality has not been explored much in research or in practice, however, due to the extreme complexity both in programming this system (there is no toolbox as there is with type-1 logic) as well as in the defuzzification of a non-interval FIS.

Aside from being inherently more fuzzy, a type-2 fuzzy inference system has a similar overall process to a type-1 system, as can be seen in Figure 3.6. It should be noted that in a type-2 system not only does the output need to be defuzzified, but it also must be type-reduced.

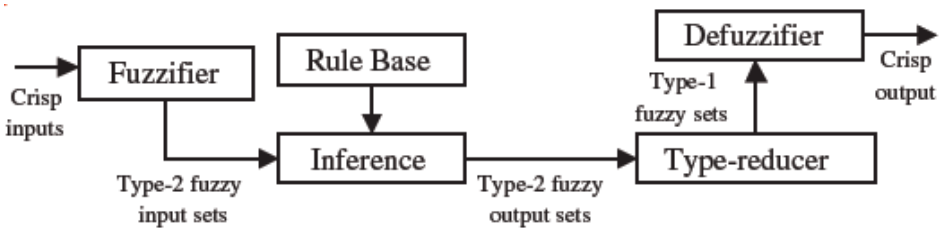


Figure 3.6: An overview of a type-2 fuzzy logic system

Type reduction of type-2 systems reduces the fuzzy output array of the type-2 logic system into a type-1 fuzzy set output that can then be defuzzified into a crisp output. The type-reduction method used in a T2FIS relies on the desired defuzzification method for the system, as each defuzzification method has a corresponding type-reduction method. Table 3.4 presents the associated type-reduction methods for the defuzzification methods given in Table 3.3.

Table 3.4: Common type reduction methods [30]. ** denotes values that require algorithms outlined in [30] which require a lot of explanation but are not used in this thesis. If the reader is interested, they may refer to chapters 10-12 in the aforementioned text.

Method	$y(x)$	$\mu(y)$	$Y(x)$
Centroid	$y_c(x) = \frac{\sum_{i=1}^{N} y_i \mu_B(y_i)}{\sum_{i=1}^{N} \mu_B(y_i)}$	$\mu_{\tilde{B}}(y) = \coprod_{l=1}^{M} \mu_{\tilde{B}^l}(y)$	$Y_c(x) = \int_{\theta_1 \in J_{y^1}} \cdots \int_{\theta_N \in J_{y^N}} [f_{y^1}(\theta_1) \bigstar \cdots \bigstar f_{y^N}(\theta_N)] / \frac{\sum_{i=1}^{N} y_i \theta_i}{\sum_{i=1}^{N} \theta_i}$
Center of Sums	$y_a(x) = \frac{\sum_{l=1}^{M} c_{B^l} a_{B^l}}{\sum_{l=1}^{M} a_{B^l}}$	$\mu_{\tilde{B}}(y) = \frac{\sum_{l=1}^{M} \mu_{B^l}(\bar{y}^l)}{\sum_{l=1}^{M} \mu_{B^l}(\bar{y}^l)}$	$Y_a(x) = \int_{\theta_1 \in J_{y^1}} \cdots \int_{\theta_N \in J_{y^N}} [f_{y^1}(\theta_1) \bigstar \cdots \bigstar f_{y^N}(\theta_N)] / \frac{\sum_{i=1}^{N} y_i \theta_i}{\sum_{i=1}^{N} \theta_i}$
Center of Average	$y_h(x) = \frac{\sum_{l=1}^{M} \bar{y}^l \mu_{B^l}(\bar{y}^l)}{\sum_{l=1}^{m} \mu_{B^l}(\bar{y}^l)}$	**	$Y_h(x) = \int_{\theta_1 \in J_{\bar{y}^1}} \cdots \int_{\theta_M \in J_{\bar{y}^M}} [f_{\bar{y}^1}(\theta_1) \bigstar \cdots \bigstar f_{\bar{y}^M}(\theta_M)] / \frac{\sum_{l=1}^{M} \bar{y}^l \theta_l}{\sum_{l=1}^{M} \theta_l}$
Modified Center of Average	$y_{mh}(x) = \frac{\sum_{l=1}^{m} \bar{y}^l \mu_{B^l}(\bar{y}^l)/\delta^{l^2}}{\sum_{l=1}^{M} \mu_B^l(\bar{y}^l)/\delta^{l^2}}$	**	$Y_{mh}(x) = \int_{\theta_1 \in J_{\bar{y}^1}} \cdots \int_{\theta_M \in J_{\bar{y}^M}} [f_{\bar{y}^1}(\theta_1) \bigstar \cdots \bigstar f_{\bar{y}^M}(\theta_M)] / \frac{\sum_{l=1}^{M} \bar{y}^l \theta_l / \delta^{l^2}}{\sum_{l=1}^{M} \theta_l / \delta^{l^2}}$
Center of Sets	$y_{cos}(x) = \frac{\sum_{l=1}^{M} c^l T_{i=1}^{p} \mu_{F_i^l}(x_i)}{\sum_{l=1}^{M} T_{i=1}^{p} \mu_{F_i^l}(x_i)}$	**	$Y_{cos}(x) = \int_{d_1 \in C_{\tilde{G}^1}} \cdots \int_{d_M \in C_{\tilde{G}^M}} \int_{e_1 \in E_1} \cdots \int_{e_M \in E_M} T_{l=1}^{M} \mu_{C_{\tilde{G}^l}}(d_l) \bigstar T_{l=1}^{M} \mu_{E_l}(e_l) / \frac{\sum_{l=1}^{M} d_l e_l}{\sum_{l=1}^{M} e_l}$

3.1.5 Cascading Logic

Cascading fuzzy logic is a method of layering fuzzy inference systems (FIS) so the output from one fuzzy system becomes the input for another fuzzy system. It is believed that the layers in a cascading fuzzy logic system emulate the way humans reason about complex problems through either consciously or sub-consciously breaking a problem down into smaller, more simple problems.

While the following chapters of this thesis give several examples of the effectiveness of cascading fuzzy logic in simplifying complex AI reasoning problems, a mathematical approach to the cascading logic solution was also explored. By showing mathematically that cascading fuzzy systems simplifies the logic, it is possible to make the argument that a cascading system is a more efficient and potentially effective method for solving complex problems with fuzzy logic.

As with any controls system for AI, a fuzzy inference system (FIS) can be greatly improved with more inputs and more membership functions added to the inputs, as both of these changes increase the situational awareness of the system. Improving the FIS in this way doesn't come without cost, however. Adding more inputs and membership functions to

a FIS means the user must identify more rules for the system, or that the genetic algorithm, in the case of genetic fuzzy systems, must tune more rules. This puts the system in a difficult position, as more information to make the fuzzy system more precise creates so many rules that the researcher is either left searching thousands of rules for the one causing an error, or waiting an extremely long amount of time for a genetic algorithm to tune the system. Adding more situational awareness makes the problem grow exponentially. Equation gives the number of rules produced (R) by a FIS with (N) membership functions in each input, and (M) inputs. It is assumed for simplicity that the number of membership functions in each input is equal. While this is not always true in the real world, the resulting trend is the same.

$$R = N^M \tag{3.1}$$

By cascading fuzzy logic systems, it is possible to reduce not only the number of rules in each FIS file, but significantly reduce the number of rules in the overall cascading system. Equations and have been derived (based on patterns found when considering the cascading system as a binary system [22]) to show the relationship between the number of layers in the cascading system (F), the total number of inputs in the system (M), the number of inputs each layer can have (L), the number of membership functions in each input (N), and the resulting number of rules the entire system will have (CR). This system of equations is very useful as it allows for educated planning of cascading logic systems.

$$F = \left\lceil \frac{(M-1)}{(L-1)} \right\rceil \tag{3.2}$$

$$CR = F * (N^M) \tag{3.3}$$

Using Equations 3.2, and 3.3, Table 3.5 gives the resulting number of rules in the entire system for four example FIS where two have 3 membership functions per input (one has 3 inputs and the other has 6) and two have 5 membership functions per input (again one has 3 inputs and the other has 6), which is very common in applied FIS. The number of rules in a singular FIS are compared to the rules in a cascading FIS where there is a set limit of 3 inputs per layer (CR_3), and a cascading FIS with a limit of 2 inputs per layer (CR_2). Finally, the average number of membership functions per input and inputs from the 2 player FLIP application (chapter) is presented so one can see how cascading logic in an application reduced the solution space for the FIS significantly.

N	M	R	CR_3	CR_2
3	3	27	27	18
3	6	729	63	45
5	3	125	125	50
5	6	15,625	275	125
5	9	1,953,125	500	200

Table 3.5: Comparison of singular and cascading fuzzy inferencing systems. Note the extreme decrease in complexity in the cascaded systems.

From this it can be seen that limiting each cascade to 3 inputs results in (on average) a 60% reduction of rules from a singular system, and limiting each cascade to 2 inputs results in (on average) an 80% reduction of rules from a singular system. Therefore this exercise suggests that a cascading logic architecture creates a much more efficient control system with a smaller solution space, which allows for the system to include more situational awareness,

increasing its effectiveness.

3.2 Genetic Algorithms

One of the applications of cascading fuzzy logic in this thesis includes cascading a fuzzy system with a genetic algorithm. A genetic algorithm is an algorithm that uses mixtures of a population in an optimization problem to create better solutions to the problem. A good analogy for this process is to think of how humans go about breeding animals. Before the algorithm runs, the 'best solutions' from either the starting values or the most recent iteration are chosen to create the next solution set, much in the same way a farmer chooses only the most desirable cows to start her breeding stock from. There are several methods that can be used to make this determination, each one having its own strengths. These methods include: Fitness proportionate selection, Boltzmann selection, tournament selection, rank selection, steady state selection, truncation selection and local selection. As with actual genetics, as the algorithm runs several things can happen including crossover and mutations [16].

Crossover is the act of strings of traits or "'genes"' being mixed to create a "'child"' with traits from both "'parents"' Several types of crossover exist.

- One point crossover is the most simple crossover technique. A point is randomly chosen, beyond which the data from both parents is swapped.

- Two point crossover is similar to one point crossover, however everything between the two randomly chosen points in the parent genes is exchanged.

- Cut and splice is a method where two different random points are chosen within the

parent genes. Data from the parents is swapped, however unlike in one point crossover, the resulting genes may have different lengths.

- Uniform and half uniform crossover are similar techniques which use a user-defined mixing ratio between the two parents to create the children. A mixing ratio of 0.5 means the children have about half of their genes from the first parent, and half from the second. Half uniform crossover is a bit more complicated, as the Hamming distance must be calculated which defines the number of bits that differs between the parents. This is then taken into account when determining what to crossover from each parent.

- Three parent crossover is exactly as it sounds. Three parent genes produce offspring together, with genes randomly chosen from each parent.

- Crossover methods for ordered chromosomes are various methods that are used when the genetic algorithm is solving a solution where a direct swap of the parents wouldn't make sense. This is the case for solving traveling salesman problems, where a direct swap could result in the algorithm skipping some cities while visiting others multiple times. Examples of crossover operators that preserve the solution order include: cycle crossover, partially matched crossover, order crossover operator, order-based crossover operator, position-based crossover operator, voting recombination crossover operator, alternating-position crossover operator, and sequential constructive crossover operator.

Mutations are what happens when the algorithm randomly places values in the solution space, to help the algorithm avoid settling in local minima. Mutations occur in a genetic algorithm based on a user-defined mutation probability, which should be set low to encourage convergence on a best solution. Setting the probability too high makes the system act more

like a random search function which is generally not the most efficient solution for genetic algorithm applications. There are several types of mutations that exist and can be used simultaneously within a genetic algorithm.

- A bit string mutation can occur with binary genetic algorithms when one of the bits that makes up a gene string is reversed.
- Flip bit mutation takes genes at random and reverses them completely.
- Boundary mutations replace the genes of an integer based genetic algorithm with either the lower or upper bound values for that gene.
- Non-Uniform mutation operators are useful for keeping the population of a genetic algorithm from stagnating in the early stages of its search for a solution. This is accomplished by having a dynamic mutation operator probability, which approaches 0 as each generation progresses.
- Uniform mutation replaces values of chosen genes with a different, random value that is determined by the user. This type of mutation is only possible with integer-based genes.
- Gaussian mutation chooses a random value from a Gaussian distribution and adds it to chosen genes. This can only be used with integer-based genes.

Once each set of possible solutions is defined for the current generation, they are tested against each other for the optimal solution to the problem at hand. Based on each sets' performance, a new set of 'parents' are then chosen to create the next generation. This cycle continues until the solutions to the algorithm reach some sort of convergence.

3.3 Traveling Salesman Problem

The fuzzy system cascaded with a genetic algorithm in this thesis is applied to a modified traveling salesman problem. The traveling salesman problem is a challenging optimization problem, with many variations that have been studied by mathematicians (and, more recently, theoretical computer scientists) for over a century. The main premise of the problem is: given a set of points in a sample space, what is the optimal route that touches each point in the sample space before returning to the starting position. Here, an optimal route is generally defined as either the fastest or shortest route between two targets.

Two main methods are currently accepted as means to come to a solution of a traveling salesman problem [1]. The first idea is to create a complex numerical algorithm that can find exact solutions to the problem. These work rather quickly and are effective with small sized problems (very few targets); however in real world applications something with a bit more versatility is necessary, as not all problems are small. A second method is using "suboptimal" or heuristic algorithms which produce effective solutions at small sample sizes, but begin to become less optimal as the number of targets increase. It should be noted that the number of targets in which a heuristic algorithm begins to become less optimal is much higher than that of a numerical algorithm. Genetic algorithms are often used to create suboptimal algorithms for a TSP, as they can be tuned to be quite effective and quick at very high numbers of targets.

It is also possible to take a suboptimal solution to a heuristic algorithm, break it down and optimize it. This is called dividing the TSP into "sub-problems". Due to the nature of the fuzzy system and further straight line correction (both of which will be explained later in

this paper), this research endeavor could be considered the creation of a heuristic algorithm that optimizes sub problems of another heuristic algorithm, thereby further optimizing the entire solution.

3.4 Cascading Fuzzy Logic vs. Artificial Neural Networks

An Artificial Neural Network (ANN) is usually presented as a system of interconnected neurons which exchange messages with each other, much like biological neurons do in animal brains. ANNs are used for approximating functions dependent on a large number of varied inputs, which makes them very useful in real world applications where adequate situational awareness depends on many inputs. The neuron connections within an ANN are mapped with numeric weights, and can be tuned based on the experience of the system, which allows it to learn from experience. Learning is based on the cost function chosen for the ANN system, and can be supervised, unsupervised or reinforced based on the task at hand. Generally speaking, the structure of ANNs consist of a set of input nodes, a set of hidden nodes that connect the inputs, and then a (usually smaller in number) set of output nodes created by the hidden nodes. This layering of neurons is analogous to the layers in a cascading fuzzy system, however there are some differences.

A simple analogy for understanding the difference would be to think of two racecar drivers; one with a cascaded fuzzy logic mind, and the other with an Artificial Neural Network mind. Both racecar drivers learn how to drive in a high-end, high performance car

like a Lamborghini. Through trials of driving this speedy car, both drivers fine tune their abilities and both become top racers. Now, let's place both drivers behind the seat of a 1995 Oldsmobile mini-van, and see how they do. This is where the main difference between the abilities of the cascaded fuzzy and the ANN drivers can be found. The driver with the ANN brain will have some difficulty because the network they have created based on the Lamborghini is so incredibly different than the mini-van, they have a hard time knowing how to get started. So while they may have optimized for racecar driving very quickly and became arguably the world's best racecar driver, the ANN is not that great at being transplanted into a new situation - even though both are cars. Meanwhile the cascaded fuzzy driver would have some difficulty at first, but would be much more ready to map the basic controls from the sports car to the basic controls of the mini-van, because they are both cars. Both have steering wheels, gas, break, etc. In this way, the fuzzy system would be able to learn its new environment, and become a fantastic old-school mini van driver.

The main point behind this analogy is that Artificial Neural Networks are optimal for many applications, but it is difficult to take a network that has been trained in one environment and expect it to do well in another. On the other hand, since cascading fuzzy systems are based more on heuristics than optimization of weights, they are much more resilient to big environmental changes. Finally, cascading fuzzy logic has been shown to reduce the solution space of a complex fuzzy system, which opens up the possibility of fuzzy logic use in applications where ANNs were more popular due to the complexity of the problem at hand.

Chapter 4

Fuzzy Collaborative Robotic Pong (FLIP) - Development and Application of Cascading Fuzzy Logic

There are a growing number of aerospace applications demonstrating the effectiveness of emulating human decision making using fuzzy logic. Main research challenges include situational awareness and decision making in an uncertain time critical spatio-temporal environment. In this effort, a MATLAB simulation environment of the classic arcade game PONG is utilized and a fuzzy logic system has been created that uses real-time reasoning and awareness to represent a single a human player. A second fuzzy system was then created that uses real-time collaboration as well as fuzzy reasoning and awareness capabilities to play two human players. This collaboration strategy is based on the researcher's experience and learning lessons from watching two recent professional tennis matches. After an iterative process of

tuning the system, both the singles and doubles games proved difficult for human players to beat. The results demonstrate the effectiveness of the fuzzy logic collaborative robotic system.

4.1 Problem Formulation

A MATLAB game depicting singles ice hockey was uncovered from an online source, with clearly written code that served as a good template from which the researcher could start creating the game simulation. It was decided that the simulation would be more appropriate if it depicted the 1980s Atari game PONG, so the vertical side walls of the hockey game were deleted making the goals equal to the height of the game, the center circle was taken out of the background, and colors were added to make it easier to see the paddles.

PONG was considered a good choice for the simulation as the game is a dynamic, uncertain, spatio-temporal environment. It is dynamic because real-time game play is a natural aspect of PONG, and spatio-temporal since success in the game was highly dependent on being in the right place at the right time. The game was to be set up as robots vs. humans, so the uncertainty was provided by the human players. Robots have no insight into the mind of its human opponent, but can only react through situational awareness. For the collaboration aspect of the project, PONG was a great simulation, as both autonomous players have a common goal: to score on the humans and win the game.

To gain the expertise on PONG necessary for the creation of winning fuzzy rule sets, a carrot-and-stick method was used by the researcher. This method included one human researcher playing the singles PONG game against another human while paying attention

to the subconscious methods the human researcher was using to score on the other human. These strategies were written down and tested for accuracy, and then used in the creation of the fuzzy rule sets. After gaining necessary knowledge of the PONG game and becoming familiar with the MATLAB code, it was time to begin creating the singles robot vs. human game.

One major constraint that was imposed on this simulation was the way in which the fuzzy paddles receive and process information. In a real life scenario, the robots would have various sensors that could quantify the ball speed and trajectory, the location of the opponents, and their own location with respect to the court. However, due to this simulation being completely computerized with no physical system, the fuzzy opponents are merely provided their input values with no time delay. In a real life scenario these values would have some type of time lag due to sensor processing speed and being able to record and compute measurable input values. This method of merely providing the FIS with its necessary inputs is a way to imitate how a human perceives the game. As a human user, one can see the various components to the game. Seeing as the FIS has no way to physically see what is occurring, these values were simply provided when they occurred during game play.

One logistic assumption that was made while creating this simulation was that the ball's trajectory and bounce characteristics would neglect spin effects that otherwise occur in real life. A second assumption was made that each time the ball strikes an object (i.e. a wall or paddle) the ball's speed will increase. This makes the game more challenging as the game progresses. This ball speed increase will continue until a set maximum speed is achieved. In addition to the ball speed increasing with each bounce, if the ball is to strike the upper and lower court boundaries it will reflect with slightly less of an angle, than the incoming

angle of incidence. This assumption was imposed to assure that the ball does not get stuck bouncing between the upper and lower boundaries in a pure vertical trajectory (i.e. not moving towards one of the goals).

4.2 FLIP Solution

4.2.1 One Player

To create the singles human vs. robot game, the first step was to make one of the players in the game an autonomous fuzzy robot. To do this, it was necessary to decide which inputs, rules, and outputs would be appropriate for creating the Fuzzy Inference System (FIS) files that would control the fuzzy player. It was also necessary to decide what strategy would be fitting for the program to use. An example of the singles PONG setup can be found below in Figure 4.1.

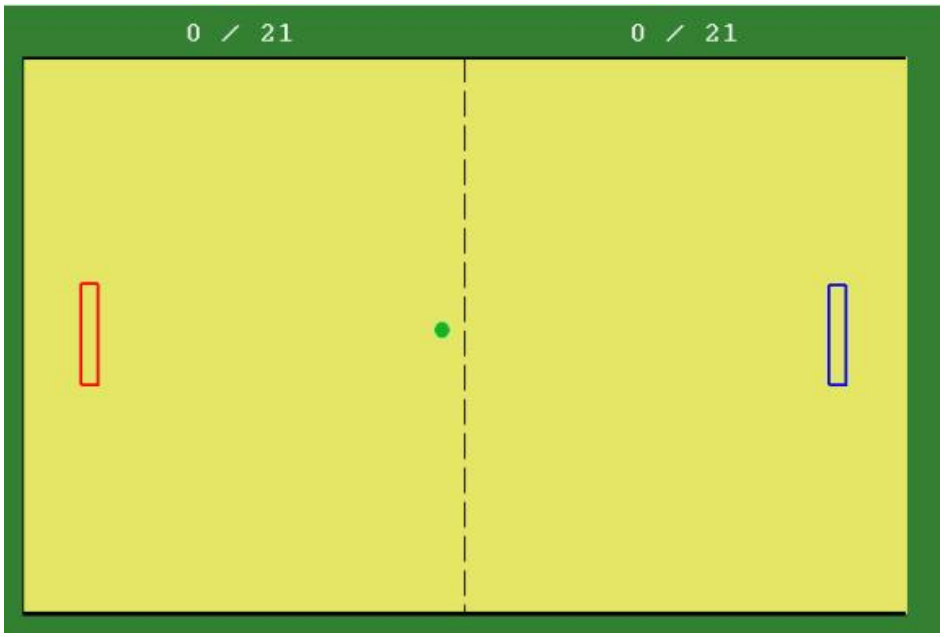


Figure 4.1: Singles PONG Configuration

While gaining knowledge from the human vs. human game, it was noted that the player could control where they want the ball to go by utilizing different parts of the paddle. The upper half sent the ball up, while the lower half sent the ball down. Using the middle would simply return the ball straight across the board, which was considered more defensive. Using the very tips of the paddle, one could make the ball bounce more in a very offensive move. While this strategy had little room for error, when it did work it was difficult for the human opponent to hit. The breakdown of the fuzzy paddle into these offensive and defensive areas can be seen in Figure 4.2. It was decided the first FIS file would be used for moving the fuzzy paddle up and down on the playing field, since paddle placement is what allows the player strategic control of the ball. To keep simplicity for the first phase of research, it was decided only the defensive (using the center of the paddle) strategy would be used since it allowed for the most error, and it was easy to find programming errors by seeing if the ball hit the center of the paddle.

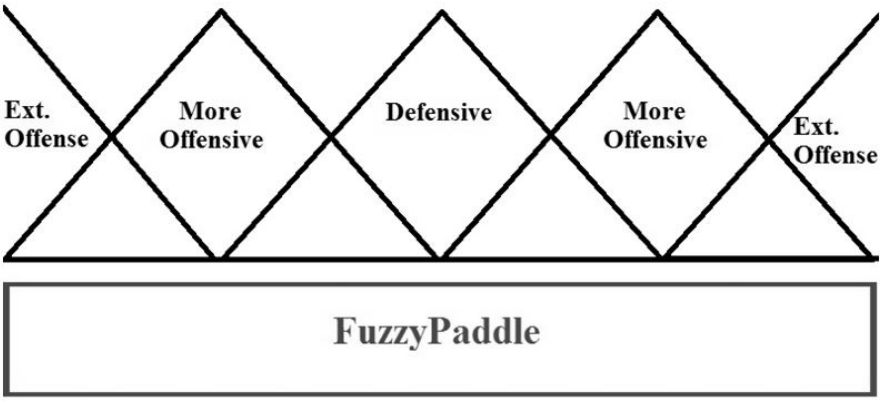


Figure 4.2: Breakdown of fuzzy paddle offensive and defensive areas.

A large aspect of the game of PONG is the user's ability to calculate where they believe the ball will intercept their paddle. Therefore it was necessary for the robot to be able to do

the same. A function was created within MATLAB that took into account the ball's current position, movement vector, and speed. Using these variables a temporary vector was then created by the robot that was a trajectory of the ball's movement. If the ball were to bounce off the top or bottom walls, it would use these same bounce calculations to make sure the trajectory was correct. The point of intersection was then calculated, which was the point where the ball was going to cross the autonomous paddle's axis of movement.

Since the FIS file was to control the paddle's movement, the input variable was the calculated difference between the point of intersection, and where the center of the paddle was at that point in time. For each iteration of the program the difference was re-calculated so the paddle would have a continuous input. Figure 4.3 shows that the FIS file created a continuous, smooth output.

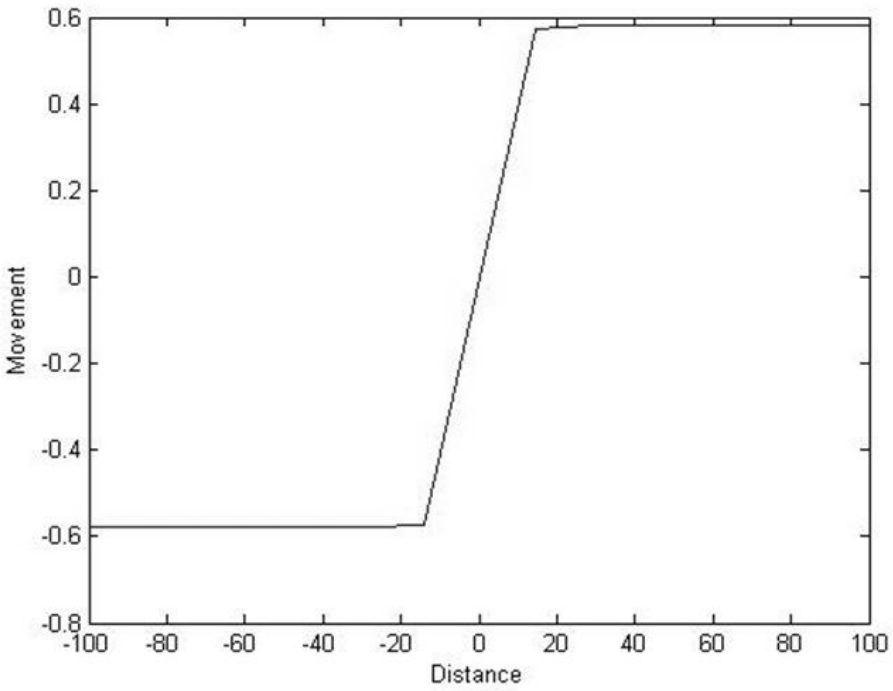


Figure 4.3: Paddle Movement FIS

The outputs were created next, as it would not make sense to create rules with inputs and no outputs. Since the FIS file was to move the paddle up or down, it was decided the output would alter a variable moving the paddle up or down. When a human plays PONG, they use two buttons on the keyboard, one to move the paddle up, and the other to move the paddle down. These keys were connected to a variable in the MATLAB code that either added one (makes the paddle move up) or negative one (makes the paddle move down) to the center of the paddle's current position. Therefore the output was to be any number between these one and negative one limits, so the FIS file could move the paddle up or down as needed.

Finally it was time to make the rules for movement. It was fairly simple to understand these rules, due to the linguistic properties of fuzzy logic. For example, if the input value was given membership to FarBelow, then the output would be a crisp number with membership to DownFast. All of the rules used the AND iterator for overlapping functions, as it allowed for smooth movement and transitions between membership functions as the paddle's center got closer to the point of intersection.

Once the autonomous fuzzy player was given the ability to decide what to do when the ball was going towards it, it was necessary to decide what should happen when the ball is moving away from it. To cover the most area, it was decided the paddle would reside in the center of the board while the ball moved toward the other player.

4.2.2 Two Player

Creation of the two player game began by making a copy of the first fuzzy player and its defensive strategy and calling the two a team. This created team was known as FLIP – Fuzzy colLaborative robotIc Pong. A second player was also added to the human side of the game. An example of the game's setup can be found in Figure 4.4. Much of the logic and placement for the doubles game was inspired by professional doubles tennis matches: The Australian Open 2010 (Bryan Brothers) and the team of Federer and Mirka.

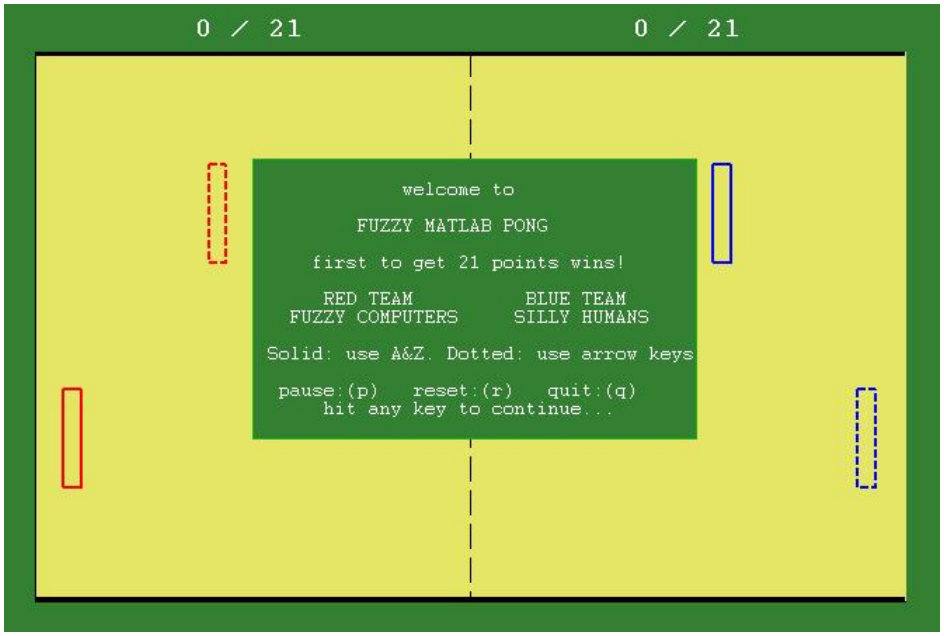


Figure 4.4: Doubles PONG configuration

While observing doubles tennis it was noted that the two players would use their own positioning to show their role in the game, as well as if the team feels if the game is offensive or defensive. Generally the back of the doubles court is considered defensive area, since it allows one more time to see the ball coming, and there is less of a chance of error. If the team is leading in a game they are more likely to both move to the back, making them both

defenders so the other team cannot score and catch up. Conversely, the front of the court (closer to the net) is considered and offense area, since it does not allow the opponent much time to observe the ball's movement before it gets to them. If the team is losing and wants to get a few quick scores in, they will both move to offensive positions closer to the net. In normal game play, having both players in the offensive or defensive positions is usually rare, because if one makes a mistake in offense there's no backup plan, and if both are in defense, there is little chance of scoring. Therefore the third way teams position themselves is with one player in the front playing offense, and one in the back playing defense. To cover as much of the court as possible, each player will stand about 1/3 of the court's width from each lateral side.

Another note that was made was how the players communicated. While studying these games it was also noted how the player in front would hold different hand signals behind their back that would correspond with certain plays or movements. These hand signals were an extremely efficient way of communicating and adapting to the game while playing, since pre-determined plays don't always go as planned, and yelling across the court would leave no surprises for the opponent.

In the two player simulation, it was decided the most reasonable set up for players was the third set up mentioned above, with one player in front, one in back, and starting 1/3 of the court's width from each side. To keep simplicity only one degree of freedom was allowed for the paddle's movements, which essentially meant the paddle in front was the offensive player, while the player in the back was considered the defensive player. FLIP 1 became the name of the player in the back, and FLIP 2 became the name of the player in front, and both players were initially programmed with the same simple 2-layer cascading FIS that was

used in the singles game. It quickly became clear, however, that more layers and more fuzzy systems would be necessary to create a working team.

The question arose of how to exercise similar communication between the two fuzzy players that was observed between the two professional tennis players. It was not considered to be strategic for both players to be competing for the ball, as the player in front would usually get it, half the court would always be left open and more difficult to defend, and the strategy associated with player placement would not be utilized effectively. Looking back at the professional tennis game, the player in front, as stated before, would give hand signals to their partner telling them who should go, and what to look out for. To mimic this behavior it was decided that the robot in front in the offensive position would make the final decision on who should go, and then each robot, knowing the plan, would individually decide what it needed to do as an offensive or defensive player. By dividing the team into an offensive team leader and a defensive player (essentially a goalie), the unique cascading system of each player became evident based on the player's needs. For example, it was noted that the defensive player really only touched the ball when the gameplay was considered to be defensive. Therefore to help with efficiency, the "'Game Type"' calculation was removed from the logic of the defensive player. It was also noted that the situational awareness could be more precise, however by adding this precision to both players, the front would often miss the ball simply because it was closer and had less time to compute its output. Therefore, the defensive player was given a highly detailed situational awareness (with 226 rules!) while the front player was programmed with a smaller (81 rules) situational awareness.

A step-by-step explanation of this communication is described in the figures and descriptions below.

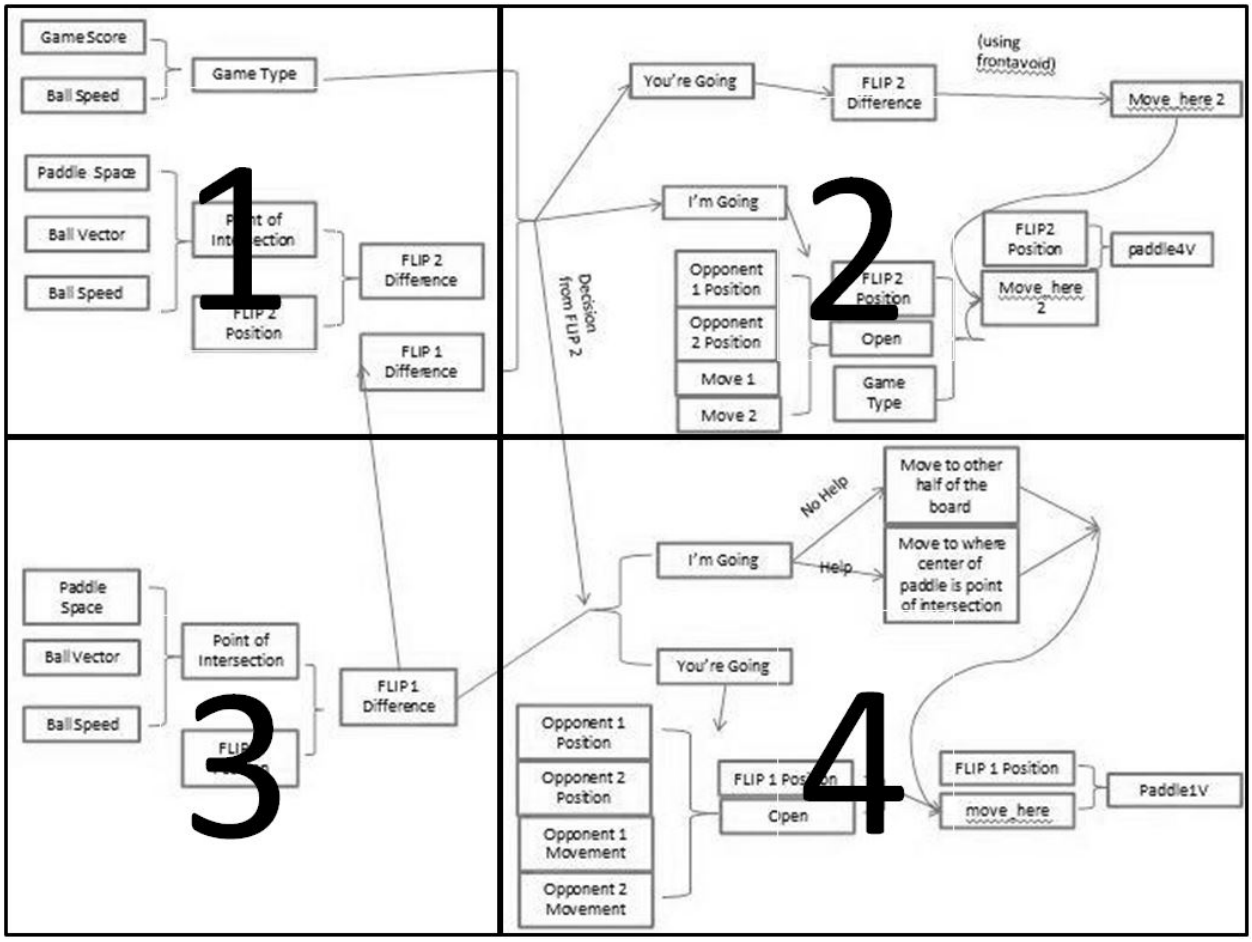


Figure 4.5: Collaborative reasoning used in FLIP

Figure 4.5 is an overview of the communication and thinking that goes on within the FLIP team. A more comprehensive version of this can be found in Figure 4.6, where the shaded terms are FIS files and the non-shaded terms are the crisp inputs into the cascading system. The instant one of the human player's paddles hits the ball, this cascading fuzzy and communication system within the FLIP team works to find who should go for the ball, where each player needs to be and what strategy they need to utilize. in this comprehensive figure, it is easy to see that the system must cascade through at least 10 fuzzy inference systems before coming out with a crisp output, which is the movement of the paddle. Figure 4.5 is divided into four areas for a more in-depth explanation of the "'fuzzy thinking"' that

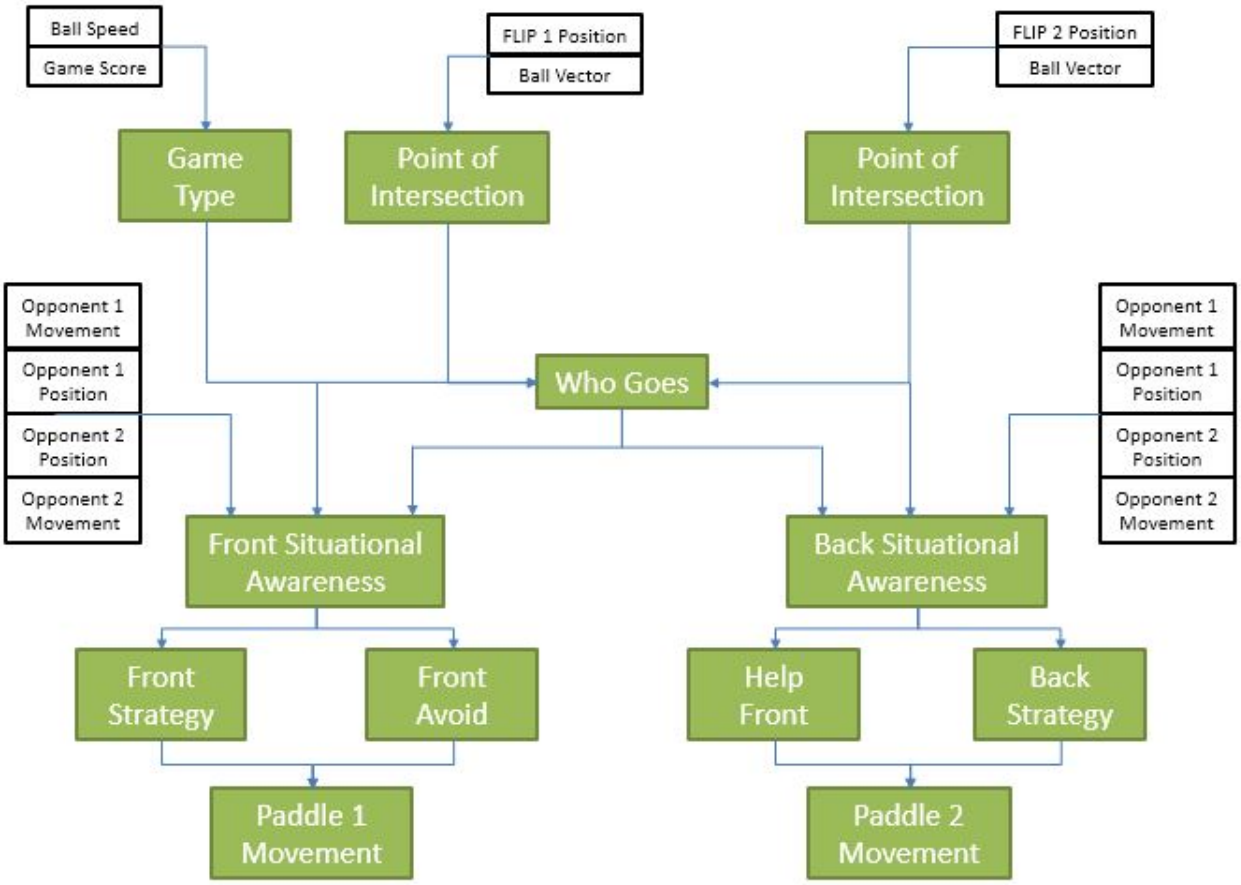


Figure 4.6: A more comprehensive look at the collaborative reasoning used in FLIP. Note that FIS are shaded, and the other outlined terms are crisp inputs.

occurs in each area.

Part 1 (Figure 4.7) shows the initial process that goes on within FLIP 2 who is in front, and the team leader. FLIP 2 calculates the distance between it and the point of intersection, while also waiting for FLIP 1 to send its own distance information. Meanwhile, FLIP 2 also uses a FIS file to calculate the game type (described more in depth in the Results section), classifying the game as a certain amount offensive and a certain amount defensive. With all of this information, FLIP 2 then decides which fuzzy player should go for the ball.

Part 2 (Figure 4.8) shows this decision being made, and what happens for each decision. If

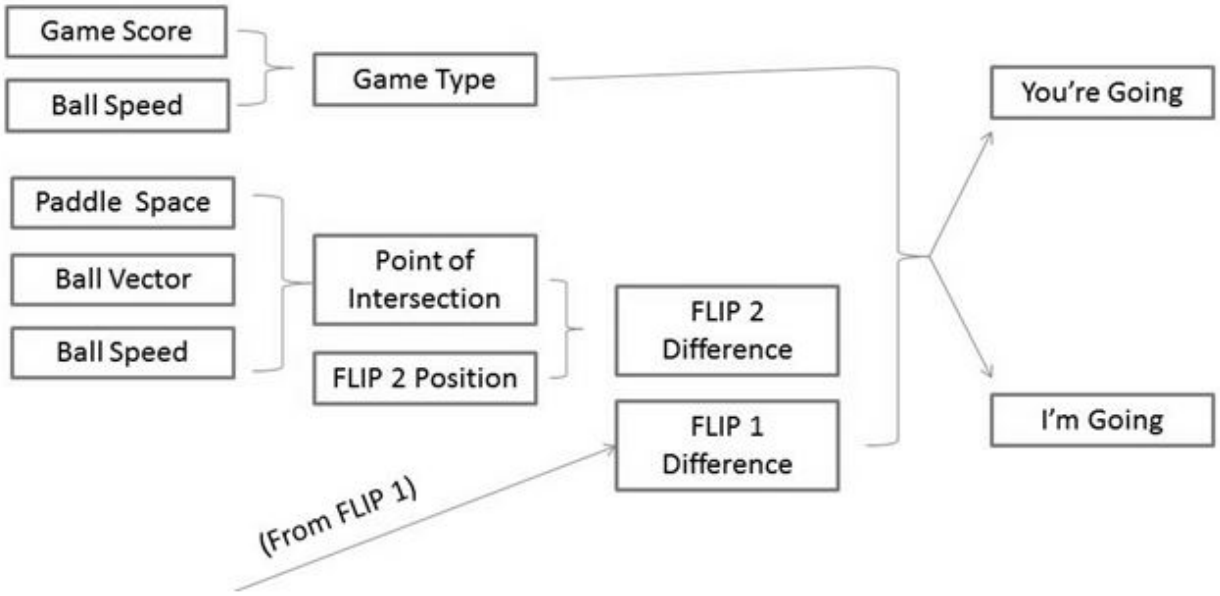


Figure 4.7: Collaborative reasoning part 1

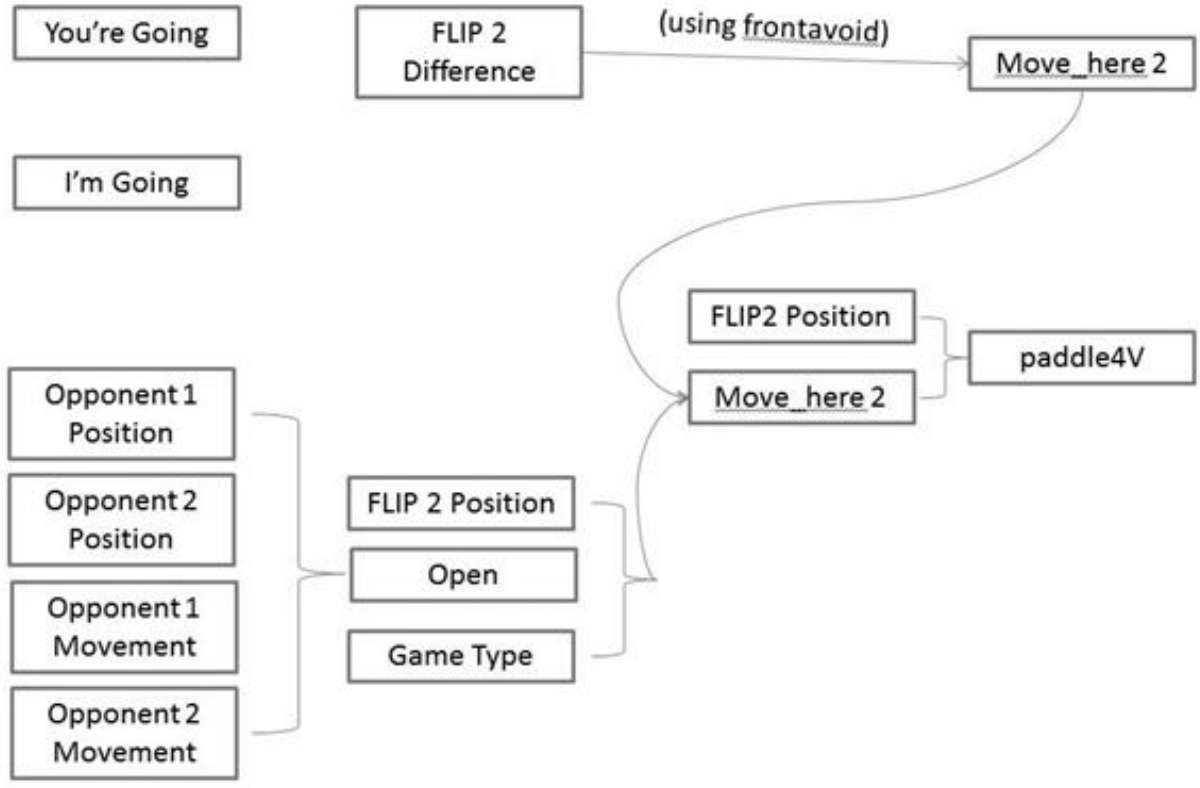


Figure 4.8: Collaborative reasoning part 2

FLIP 2 decides not to go for the ball, it utilizes a waiting strategy. The waiting strategy was created after noticing that, being the front paddle, FLIP 2 would accidentally move in front of the ball because it didn't know what it's partner was doing; just that it wasn't its turn. So, the waiting strategy for FLIP 2 became to move out of the way of the ball's trajectory, so if the player behind it hits the ball it will not accidentally hit the ball backwards and into its own goal. This action uses a simple FIS file with the inputs of the ball's trajectory, speed, and the position of the paddle. Its output has five membership functions that divide FLIP 2's playing area into very top, top, center, bottom, and very bottom. The rule sets are made in such a way that if the ball will be anywhere in the top area, FLIP 2 will move to the bottom, and vice-verse. If FLIP 2 decides it should hit the ball, it uses its fuzzy situational awareness to find the best place to hit the ball, and takes into account its own position and the game type when deciding what part of the paddle to hit the ball with. It must take its position into account, because if it is for example at the very bottom of the simulation and the best place to hit the ball is on the bottom of the opponent's side, hitting the ball down will cause it to hit the wall and bounce up. In this position the best thing to do would be to hit the ball straight. Finally FLIP 2 moves based on where it has decided it wants the ball to hit the paddle.

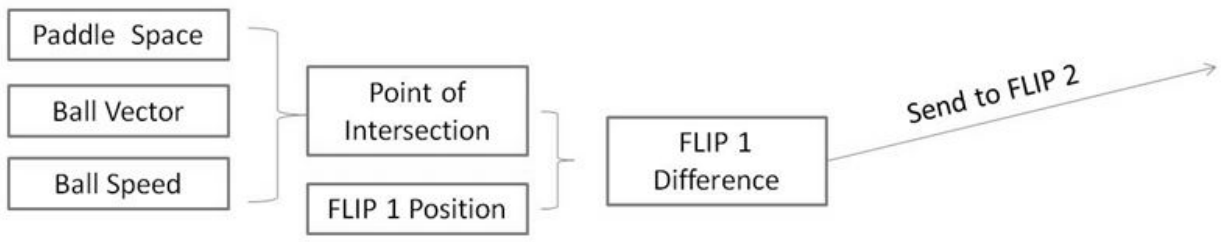


Figure 4.9: Collaborative reasoning part 3

Figure 4.9 depicts what FLIP 1 does before the decision has been made of who should go for the ball. All it has to do at this stage is find the ball's point of intersection, and calculate the difference between where it is and where it needs to be.

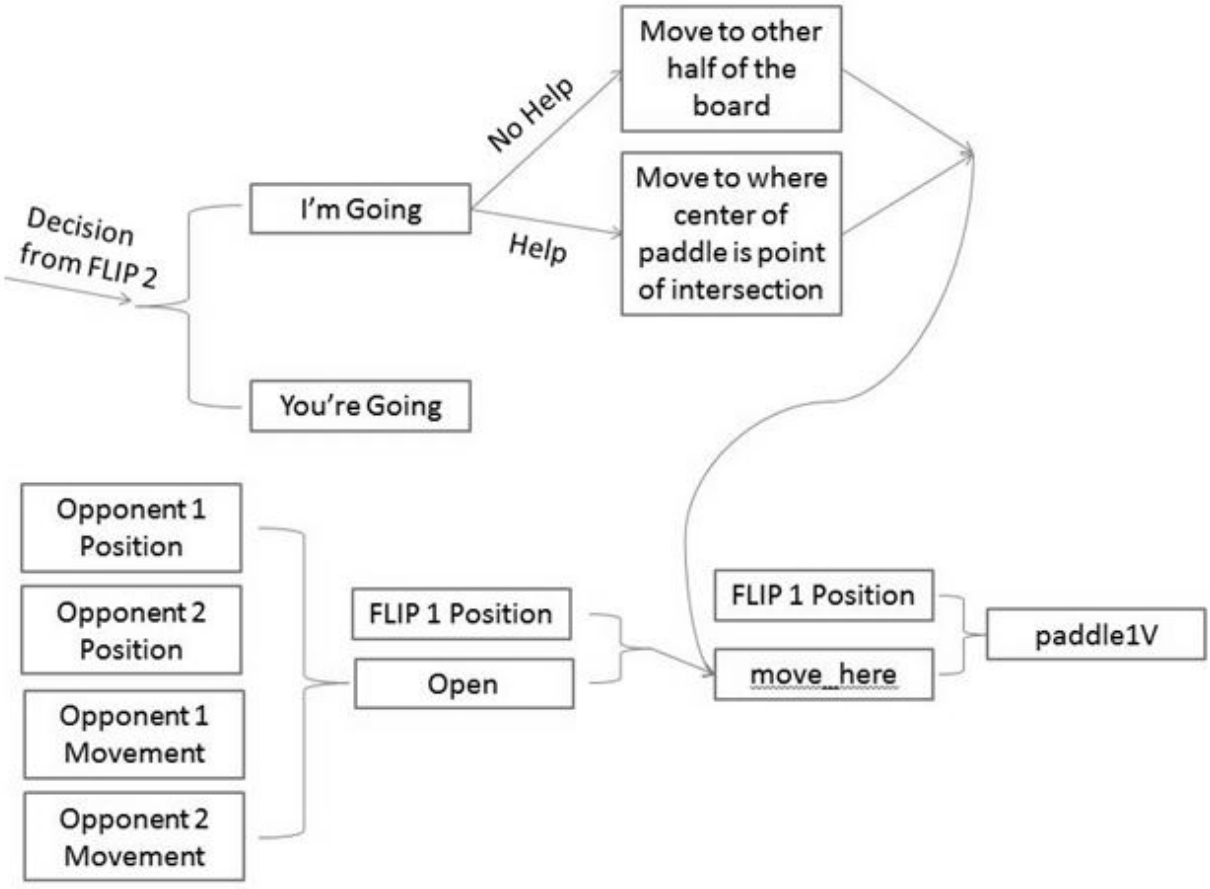


Figure 4.10: Collaborative reasoning part 4

Figure 4.10 shows the strategies FLIP 1 has for going. If FLIP 2 tells it to go, it uses situational awareness to only use defensive moves against the opponent since defensive moves have less error. FLIP 1 was developed to have several waiting strategies as well, as it was determined that constant communication between the team is beneficial to the game. If FLIP 2 tells FLIP 1 to wait, but that it needs help, then FLIP 1 will go to where the ball would otherwise hit the center of its paddle, so if FLIP 2 misses there is a backup plan. If

FLIP 2 tells it to wait but does not want help (usually because its move is more defensive), it is to cover the other side of the board. This waiting strategy for the defensive player is similar to the constant communication necessary to be successful in doubles tennis, and allowed the FLIP players to be a much more effective team.

4.3 Results

4.3.1 One Player

The one player game worked very well in Beta testing. The ball trajectory calculation worked well in calculating the point of intersection, while the FIS file created very smooth movements to get the paddle there. From Beta testing, however, a few major problems were found.

The first problem was when the ball was hit by the human paddle and then quickly hit a side wall the fuzzy paddle would incorrectly calculate the point of intersection. This didn't make much sense, as the bounce calculation in the trajectory function was identical to the bounce function used by the simulation to actually move the ball. Several solutions were considered, including rewriting the trajectory calculation and using a different strategy. After the new calculations still didn't work correctly, it was obvious something else was the problem. The cause for error was finally pinpointed when the ball's trajectory and slope were tracked during game play. When the ball bounced on a wall the trajectory and slope changed gradually, not instantaneously. This meant during the circumstance where the ball hit the human paddle and shortly thereafter the wall, the robot was observing incorrect data and

calculating the trajectory from that. A solution that was suggested was to create a feedback loop within the robot that would periodically re-calculate the ball's trajectory and point of intersection as it got closer to the paddle. If the real ball wasn't following a previously calculated trajectory, the robot was to correct its calculated point of intersection, and move accordingly. When tested this solution worked well, and the fuzzy paddle no longer moved to the wrong point of intersection.

With more Beta testing, it was found that the simple defensive strategy of having the ball hit the center of the paddle was not enough to defeat a human. It was time to develop more strategies for the fuzzy player to use. Normally when humans play tennis or PONG, they develop a strategy by first observing the other player. A good strategy would be to hit the ball where the other player is not, or even better, while the other player is moving in the opposite direction. So if the opponent is towards the top and moving up, a good strategy would be to hit the ball as low as possible. This way, not only does the opponent need time to get to the bottom, they must also change direction. In human players this means deciding to push another key, which takes up more time. If the ball is going fast enough, the human will not be able to get their paddle to the ball in time to return the hit. This allows the computer to score through playing off of a weakness. To utilize this strategy the robot needed the ability to track where the human opponent is and their direction of movement.

The single player fuzzy situational awareness was developed with two inputs, one output, and several rules. The two inputs were the position of the opponent, and their movement (found from the variable containing a 1, 0, or -1 depending on what button the human user was pushing) The membership functions for the position divided the game court into five parts: very top, top, center, bottom, and very bottom. Input membership functions for

the movement (known in the program as inertia) were: up, still and down. The output membership functions divided the court into the same five parts as the position input, and were used to determine the point the human player would have the most difficulty reaching.

It was also considered that the fuzzy player wouldn't need to be on the offense quite so much. But how was it to decide between using more offensive or defensive moves? For this, the game had to be classified by the fuzzy player as offensive or defensive. Another FIS file was used to do this since there was really no way to determine exactly when the game switched from being offensive to defensive. The FIS file had two inputs: the difference between the scores, and the ball's speed. The difference between scores had three membership functions for classification: ahead, tied, and behind. The ball's speed had one membership function called "fast" that classified the ball's speed. There was one output, whose two membership functions were called offensive and defensive. The rule logic was that if the fuzzy player is falling behind or tied it needed to use a more offensive strategy (use the tips of the paddle to hit the ball). If the fuzzy player was in the lead, it was to use a more defensive strategy (use the center of the paddle to hit the ball, not necessarily exactly with where the situational awareness said the other player couldn't go). Also the more the ball's speed held membership to "fast", the more defensive the fuzzy player would be.

With the addition of more FIS files, the game slowed down considerably which was worrisome since the plan was to have cascading FIS files in the two player game. The first solution that was considered was to simplify the FIS files, taking out some of the membership functions and rules. While this did speed up the game a little, the problem still persisted, and the strategies were no longer as precise. With the original FIS files back in the game, the simulation was tried on a different computer that had a higher processing speed. This

allowed for the realization that the problem lay in the computer's processing speed, and not the game itself. To allow for easier processing two things were then done within the MATLAB code. First, the function used to refresh the playing field was made independent of the game's iterations. This was done by creating a timer that would refresh the game every .001 seconds, which was enough to take away some of the choppiness of the game. The next change was to take the lines that loaded the FIS files and put them before the main loop for the simulation. In doing this, the necessary processing time was cut significantly as loading FIS files is not a small process. This change actually made the program progress too quickly for a human to react to the game, so a small pause was added to each iteration so, while smooth, the game worked at a playable speed.

With the above changes, the fuzzy robot quickly became much too hard for a human to play, which was both good and bad. It was great because that stage of the project could be considered successful however it was bad because one needed to score on the robot to know how to further improve it. Therefore a mirror duplication of the fuzzy robotic player was created, and the robots were to play each other. To show the robots were identical in ability, they were set to play 10 arbitrary games. The results from these games can be found in Table 4.1. A difference in scores of 7.78% was found, and with this small of a sample size it can be inferred that with a sample of 1,000 or more games this difference would become insignificant. The pause function was removed to allow for faster game play, since humans no longer needed time to react to the robot's actions, as there no longer was a human player.

With the robots playing each other a few minor errors could be found, but nothing major. Whenever something was found, the correction was duplicated on the other side to keep the players identical.

Table 4.1: Singles robot vs. robot results

Game Number	Red Score	Blue Score	Winner
1	21	15	R
2	13	21	B
3	16	21	B
4	19	21	B
5	19	21	B
6	21	18	R
7	15	21	B
8	12	21	B
9	21	18	R
10	21	16	R
Total	178	193	–

4.3.2 Two Player

The two player pong game was very similar to the single game as it also used the FIS file for classifying the game type. It also had a FIS file for situational awareness; however this file was a bit more complicated due to the awareness tracking two players instead of just one. In addition to those files, the doubles game had several more FIS files due to each player having different strategic options. An example of the strategic FIS file for FLIP1 in the back can be seen in Figure 4.11. The figure shows the smoothness of the FIS output.

Aside from collaboration issues, the original problems found in the singles game were also found in the doubles game. These issues were solved easily with the same solutions that worked above. Collaboration issues were usually caused by simple errors within the MATLAB code, and could easily be solved once the real problem was pinpointed.

Like the singles game, the doubles PONG game soon became too difficult for two human players to beat. Eventually it was necessary to create a doubles robot team vs. robots team game to continue improving the fuzzy robotic teams. As with the singles game, a mirror duplication of the doubles fuzzy team was created, the pause function was removed and they

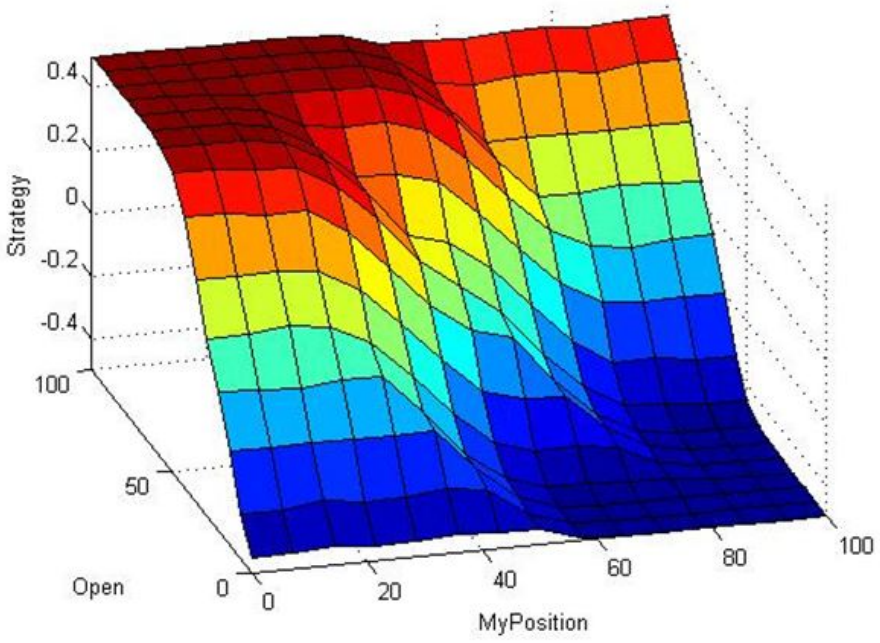


Figure 4.11: Back strategy FIS output

were set to play each other. Table 4.2 below shows the scores from 10 games to show the teams were equal in ability. The calculated difference in score was 2.53% showing that even with only 10 games one can see these abilities are equal.

Table 4.2: Doubles robot team vs. robot team results

Game Number	Red Team	Blue Team	Winner
1	21	20	R
2	21	20	R
3	17	21	B
4	16	21	B
5	21	19	R
6	20	21	B
7	21	18	R
8	21	16	R
9	20	21	B
10	15	21	B
Total	193	198	–

As with the singles game, no major errors were found, and continuous improvements were made based upon the fuzzy team vs. fuzzy team games.

4.4 Conclusions

Creating an architecture that effectively uses cascading fuzzy logic and demonstrates collaboration between two fuzzy systems would be a useful development to fuzzy robotics. Fuzzy robots with the ability to collaborate autonomously and effectively in an uncertain environment would be immensely useful in a variety of real world applications, especially in situations where humans would not be as effective due to stress or limited resources (air, water, food, etc).

From these results it can be concluded that cascading fuzzy logic is an effective means of imitating human reasoning and collaboration. The singles game shows how fuzzy reasoning and fuzzy situational awareness can be utilized in emulating a human player, while the doubles game is an example of effective use of cascading fuzzy logic to facilitate complex collaboration between two fuzzy systems.

Chapter 5

Precision Route Optimization using Fuzzy Intelligence - Cascading Algorithms and Fuzzy Logic in a Modified Traveling Salesman Problem

A MATLAB simulation of a surveillance environment was created that placed targets in random areas on a map, with each target having a circular area imposed around it. In the simulation, a fuzzy robot was to find the shortest path around the environment, where it touched each target area at least once (meaning the areas can be passed through) before returning to its starting position. Through fuzzy optimization of a path produced through a genetic algorithm, this task was completed and it was shown that a shorter path could be found through the fuzzy optimization.

A very well-known problem called the traveling salesman problem was built on in this study to determine if a fuzzy optimization tool could be developed, and if so how it could improve results. The given situation had an added level of uncertainty, which is common in real-world applications. Developments in this area could greatly improve the abilities and efficiency of intelligent reconnaissance, scientific, and transportation systems.

5.1 Problem Formulation

The problem statement was that of a traveling salesman problem, however slightly altered. In real world applications it is very common for vehicles to have a "'line of sight"', where they don't have to be directly over a target to see it. At the same time, if a target is emitting a signal that the vehicle is supposed to pick up for uses such as data transfer, the signal has some sort of footprint. Think, for example, of a UAV gathering data from several rovers that have landed on Mars, each with their own radio transmitters that can reach a mile or two into the atmosphere. The UAV in this situation would have a transmitter that can reach satellites in orbit where the data is then transmitted back to earth. To save energy, it would need to fly the shortest path between the radio signals of each rover, while still getting within range to aid in the transmission of the rover's data to the Earth.

This idea of a footprint and line of sight is to be simulated by having an area around each target that the vehicle must visit at least once, instead of having to visit the targets themselves. Each area is a circle around the target with the target in the center, and each circle has the same radius. These circles were contained within a large sample space, and overlapping of the areas was possible. The vehicle was set to start at the origin in the lower

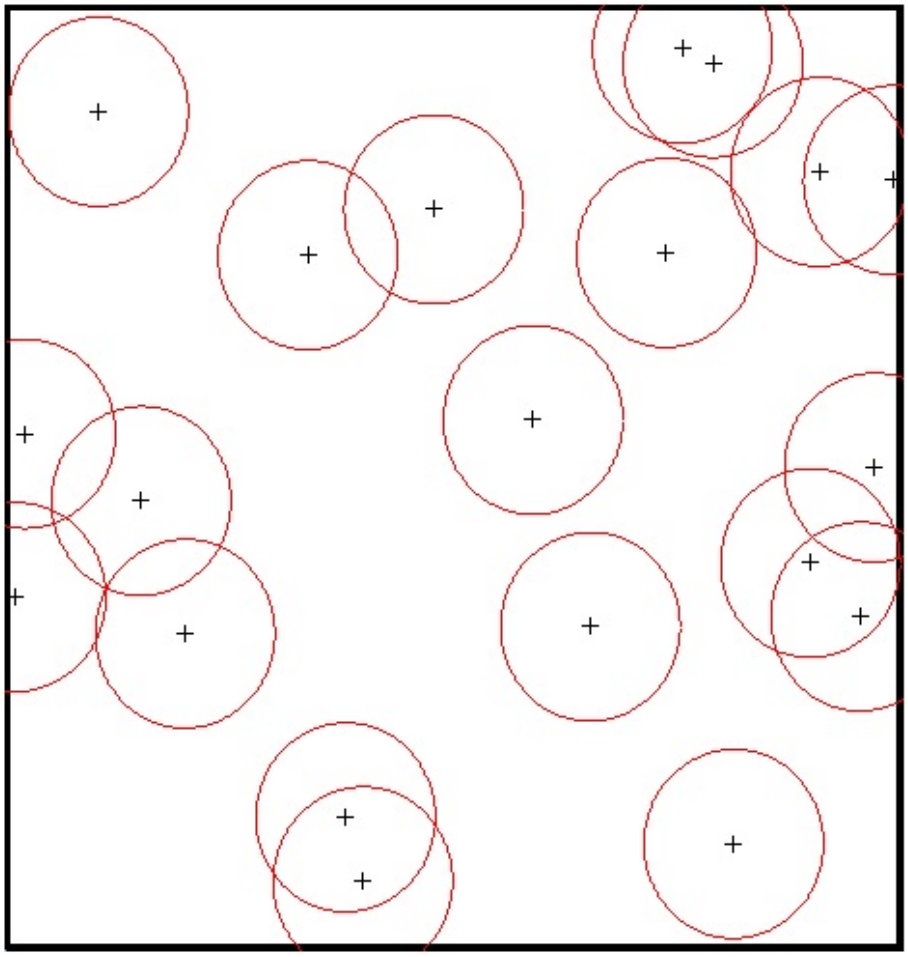


Figure 5.1: An example of the simulation space, with signal areas around each target.

left-hand corner, and it had to return there after visiting each target's footprint. An example of the simulation space can be found in Fig. 5.1.

5.2 Development of the Genetic Algorithm Fuzzy Optimization (GAFO) Solution

For this application, it was decided that MATLAB would be the best programming language to use, due to its abilities with fuzzy logic and genetic algorithms. The idea was to take the

solution from a well-established genetic algorithm, and optimize it for the signal footprints using fuzzy logic. A fuzzy optimization made sense, since each signal footprint can be thought of as a fuzzy target that must be visited. The chosen algorithm for this research was one developed by Joseph Kirk [21], which is a MATLAB application of the Dijkstra shortest path routing algorithm. This algorithm was determined to be appropriate for several reasons. The GA had been optimized for use on TSPs with less than 100 targets, and the maximum number of targets in this study would be 60. This number was chosen as it is enough below the maximum of the GA solver to be robust, yet large enough for some complexity to exist in the solution. Using a maximum of 60 targets would also simplify analysis of the Genetic Algorithm with Fuzzy Optimization (GAFO) system, as the Dijkstra algorithm program always converges to an optimal path at this small number. With a much larger number of targets, the path may not always converge causing variance within the GA paths produced by each iteration of the GAFO. Also Dijkstra's algorithm focuses on distance optimization in uniform-cost search applications, and it was decided that this study would focus on a distance-optimized path (as opposed to a time-optimized path, for example).

The simulation was developed with several assumptions. A simulation of 600 x 600 pixels in MATLAB would be used, where the space would be re-divided into 100 x 100 units to allow for a large sample space with map coordinates that are easy to understand. Identical target areas (circles with radius $r = 10$ units, making each target area about 3% of the total simulation area) would be randomly placed in the simulation space. The target area size was determined as large enough to highlight the strengths of a heuristic path planning system that can optimize a solution to areas rather than points, while small enough to allow for a mixture of overlapping and single target areas (as one would find in a real world application).

While the target areas could extend beyond the simulation space, their centers had to all be contained within the simulation's borders. Target areas could overlap, but would not be dynamic (no change in shape, size or position) during the course of the simulation.

The objective in this simulation would be to create a distance-optimized path that must either contact or travel through all target areas at least once. The solution would be a single agent solution, meaning only one path between all target areas would be made. The plan for solving this objective is as follows: create a set of random targets to be visited, run the genetic algorithm with respect to the target centers to get the optimal path, then run the fuzzy optimization to determine the best path between signal footprints. It was hypothesized that the GA solution could be optimized using fuzzy logic, since the signal areas would allow for distances to be minimized if visited in the correct manner.

Since a fuzzy logic system emulates human reasoning and problem solving, an important part of developing a fuzzy system is to examine the methods (and linguistic reasoning) humans use to solve the same, or a similar, problem. In this way, it can be realized what input variables are necessary for the needed level of situational awareness, as well as understand what rules will be necessary. 25 plots similar to Figure 5.1 were used for human trials, which included a set of targets, their signal area, and the GA solution to the map. Each set of five plots had five more targets than the last set (so a range from five to 25 targets), to determine if variables and rules created by patterns found in the human solutions and reasoning reported by those solving the maps were robust.

During the human trial it was quickly realized that the situational awareness necessary was a bit more sophisticated than originally thought. This was due to the fact that the optimal position to travel to or through each footprint area depended on the position of

the area it was to travel to next. Therefore, the new question became how to depict this information through variables, and then use it effectively within the fuzzy system. Since it was noted that the directions between the area about to be visited and the one after that held a lot of influence, it was decided to use available information to determine these directions (theta and theta prime). The x and y distance between each target's center could be easily calculated, and through taking note of the geometry presented in Fig 5.2, Eq. 5.1 and 5.2 were then used to find theta and theta prime. These two variables could then be used as inputs in the fuzzy system. As depicted in Eq. 5.1 and 5.2, theta and theta prime are calculated using the distances between the center of the target areas being taken into account, however the question still arose if the distance between the centers of these target areas would be a necessary input into the fuzzy logic system. To answer this question, it was considered how the next step in the path changed when target areas were closer or further apart. theta and theta prime were held constant in this case, as this would simulate a change in target distance that the fuzzy system would be blind to with inputs of only theta and theta prime. Human testing determined that as long as the targets aren't overlapping, a change in distance between targets where theta and theta prime remain constant does not change the next step in the path. In the case where the targets are overlapping, the path is affected. This finding would have to be taken into account when creating the Genetic Algorithm with Fuzzy Optimization (GAFO) system.

$$\theta = arctan(\frac{\Delta y}{\Delta x}) \tag{5.1}$$

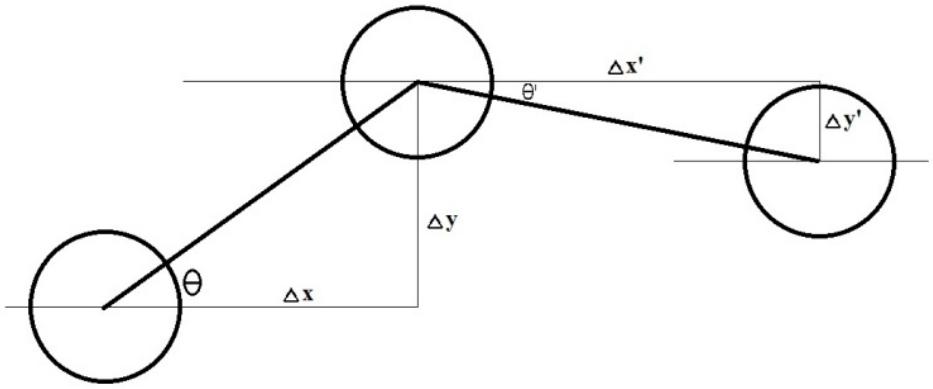


Figure 5.2: Geometry necessary to find inputs for the GAFO fuzzy system

$$\theta' = arctan(\frac{\Delta y'}{\Delta x'}) \tag{5.2}$$

In equations 5.1 and 5.2, θ is the angle between the current position and the center of the next target, Δx is the x distance between the current position and the next target and Δy is the distance in the y direction between the current position and the next target. θ' is the angle between the center of the next target and the center of the target after that, $\Delta x'$ is the x distance between the center of the next target and the center of the target after that, and $\Delta y'$ is the distance in the y direction between the center of the next target and the center of the target after that.

θ and θ' are measured relative to the path of the genetic algorithm in the Genetic Algorithm with Fuzzy Optimization (GAFO) system inputs and outputs, allowing the same heuristics to apply to each point on the path where a decision must be made, regardless of orientation. The output is then converted to global angle coordinates to allow for more simplified plotting in MATLAB to allow for more simplified path plotting.

The core of the Genetic Algorithm with Fuzzy Optimization (GAFO) system was a fuzzy

system responsible for deciding where the path should intersect with each signal footprint area (circle). With the two fuzzy inputs and an understanding of rules necessary from working through the maps, the Fuzzy Inferencing System (FIS) called "'nav"' was created. A graphic summary of this FIS file can be found in Figure 5.3. Table 5.1 gives a breakdown of the rule mapping used in this fuzzy inferencing system.

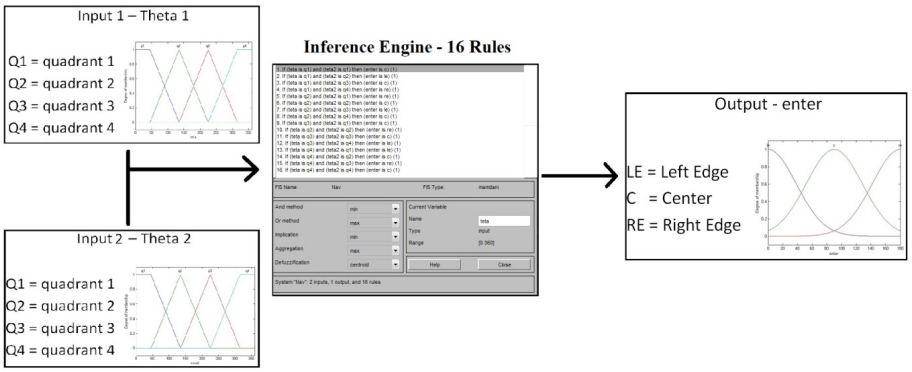


Figure 5.3: Graphic summary of the FIS used in GAFO fuzzy optimization layer

Table 5.1: Rule Mapping of the FIS used in GAFO fuzzy optimization layer

AND	θ				
	–	Q1	Q2	Q3	Q4
	Q1	C	RE	C	LE
θ'	Q2	LE	C	RE	C
	Q3	C	LE	C	RE
	Q4	RE	C	LE	C

θ and θ' are first classified as being in quadrant 1, 2, 3 or 4, and then sent into the inference engine where the rules determine the output, which is where the path should intersect with the next circle. The output values can be any angle between 0 and 180, and is classified with

The defuzzified output value is used in the path, and the GAFO moves on to the next turning point. This is an iterative procedure, which goes through each point given by the

GA (the shortest path between the target centers) until it has returned back to the origin where it began. An example of an output produced by the basic GAFO can be found in Figure 5.5.

It was determined that further optimization of the fuzzy path could be useful, since the basic Genetic Algorithm with Fuzzy Optimization (GAFO) system was unable to account for overlapping target areas, and it was determined during human trials that this special case would need to be addressed. When target areas overlap, it is possible to have fewer turning points in the path than there are areas, as the path can visit two or more areas with just one point.

The optimization strategy was based on the idea that if the path is entering its next target while still within its previous target area, then it could omit the previous point on the path that specifically went to that previous target area. Using a system of giving priority to each of the major points along the path (points that ensure each target area is met) and then searching for unnecessary nearby points, the fuzzy path could be further optimized (shortened) by omitting redundant parts of the path. This optimization was expected to create a more direct path between points, which allows it to better fit the objective of a more distance optimized path. The further optimized fuzzy path can be found in Figure 5.6.

5.3 Results

In the following sections an example scenario is solved using the GAFO method, followed by a statistical analysis of this novel method. The example solutions are for the scenario of randomly placed targets found in Figure 5.1.

5.3.1 Genetic Algorithm with Fuzzy Optimization (GAFO) Scenario Results

To give an idea of the layers within the GAFO method, results have been broken down into three major steps so the reader can better understand how the final solution is produced. Figure 5.4 shows the result after the first layer of the GAFO has run, which is the basic genetic algorithm. This initial solution simply gives a path between the centers of each target area.

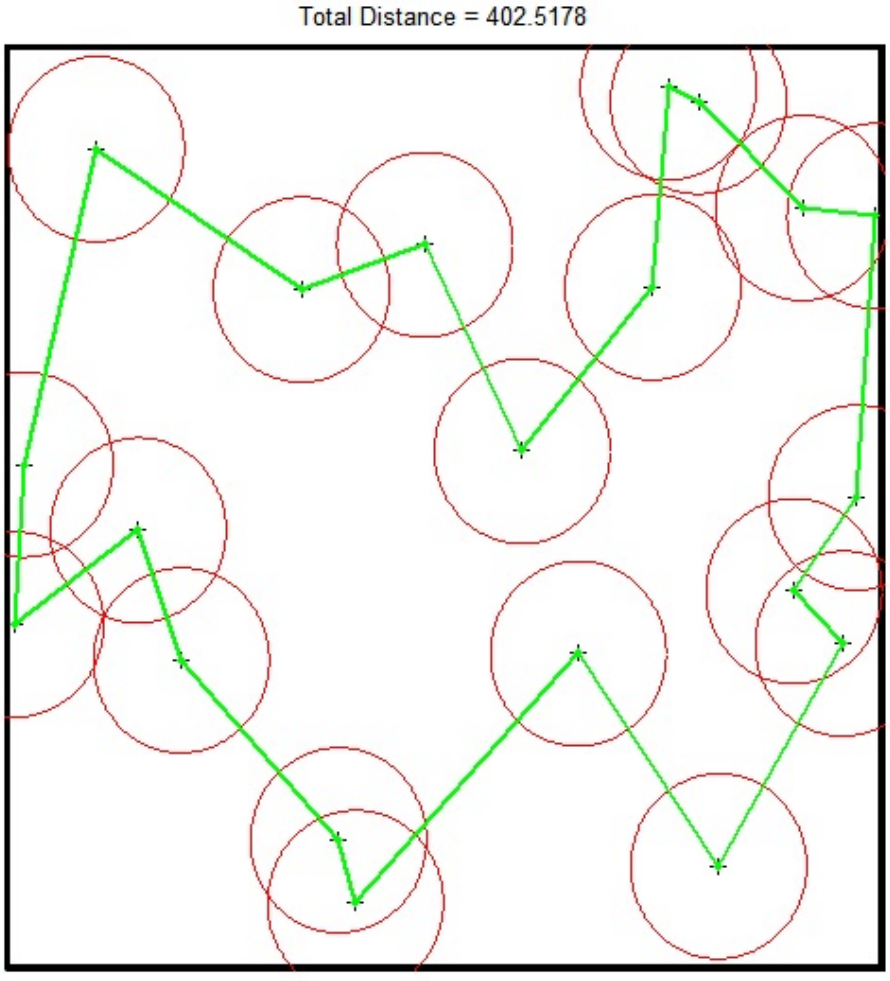


Figure 5.4: GAFO initial results (genetic algorithm)

The next layer in the GAFO system is the fuzzy optimization layer, which uses the GAFO

fuzzy system to determine a more optimal path between target areas based on the initial genetic solution. This intermediate solution of the GAFO system can be found in Figure 5.5.

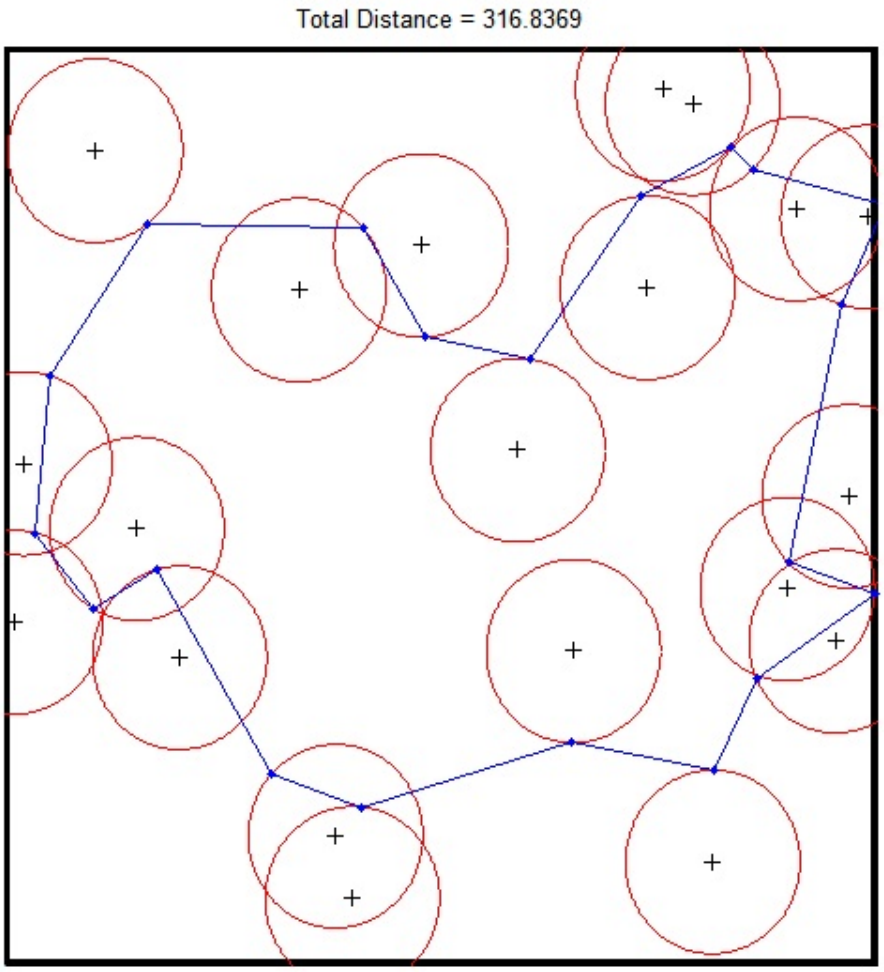


Figure 5.5: Intermediate GAFO results, with just the fuzzy optimization

The need for a second layer of optimization can easily be seen in Figure 5.5, as the intermediate solution has difficulty creating an efficient path when there are overlapping targets. Therefore, the second layer of optimization is a function that determines which points within the intermediate solution are unnecessary, and removes them creating a more efficient solution. The final solution for this example with all layers of optimization included

can be found in Figure 5.6.

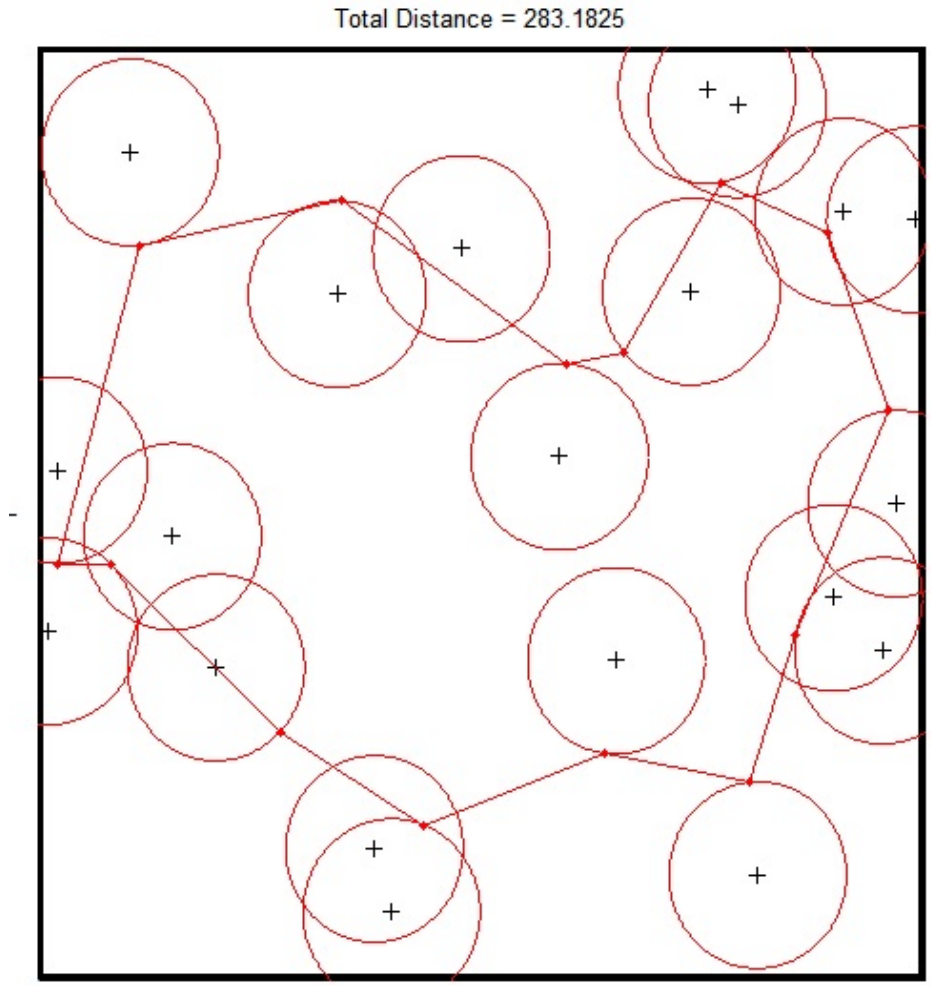


Figure 5.6: Final GAFO result, with full optimization.

It should be noted that in this example the difference between the initial GA result using the center of each target area (402.5178) and the final optimized result (283.1825) is 119.3353 units. This equates to decreasing the path length by almost a third (29.6%).

5.3.2 GAFO Statistical Analysis

In this section the GAFO method is compared to a traditional genetic algorithm. This was done to determine if the GAFO created results that were significantly more optimal than

traditional methods. In Figure 5.7 and 5.8, the difference in path length between the GAFO solution and a traditional GA solution is shown for randomly created scenarios with the number of targets increasing from 10 to 60. The GAFO is shown to produce solutions that are at least 8% shorter than those produced by a simple GA between center points, with the GAFO paths being 8 – 19% shorter in most statistically significant cases.

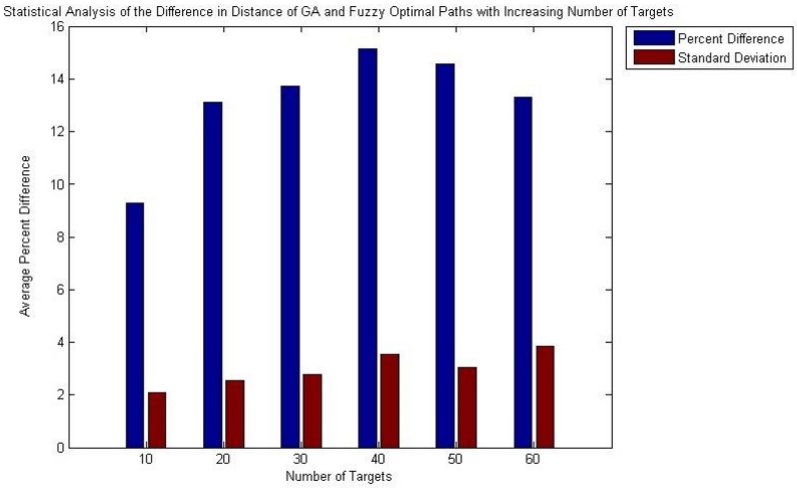


Figure 5.7: Difference in distance of GA and GAFO paths with increasing number of targets.

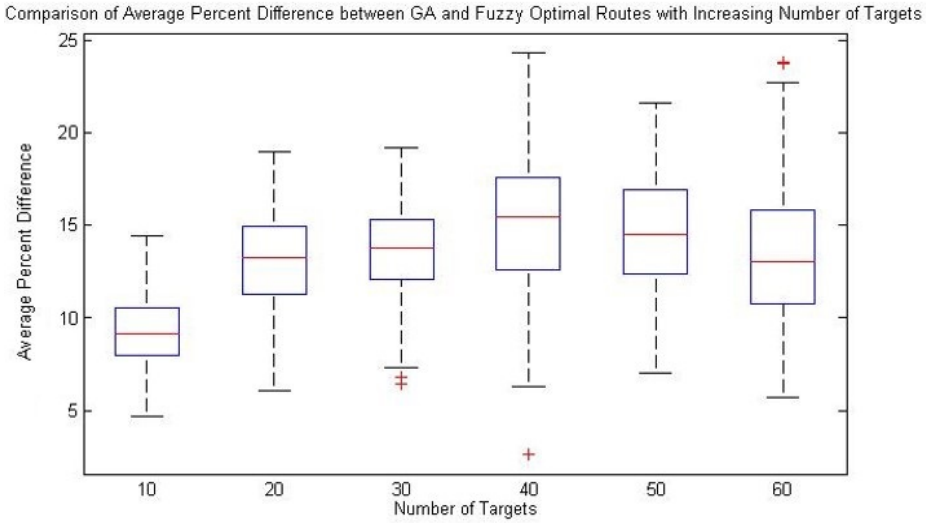


Figure 5.8: Boxplot of GA vs GAFO data for average percent difference in path distance.

5.4 Discussion and Conclusions

One can see that the total difference in path length obviously decreased between the genetic and Genetic Algorithm with Fuzzy Optimization (GAFO) solutions. This trend occurred every time the researcher observed the program run, which was very promising as the GAFO was designed to be an optimizing function. Visually, the results in Figure 5.6 promising, as the GAFO solution seemed to be more smoothed out and direct than the initial genetic solution.

The statistical analysis in general showed the data having average percent differences above 8%, with all quadrants of data being above the 5% mark. It was noted that an interesting trend occurred, where the average difference increased with increasing number of targets, however after a certain point (40) this average decreased with increasing targets. It was thought that the reasoning for this trend has to do with overlapping signal areas. The program did not account well for overlapping signal areas (circles), and therefore would sometimes take extra distance to hit an area twice; once during the overlap, and again by itself.

An attempt was made to correct this error by creating an additional layer of optimization that would be applied to the path. This layer started at the origin, and saw how far it could get using straight lines as paths while still touching the required signal areas. The endpoints for this optimization were the points created on the signal areas by the fuzzy optimization. Once the longest straight segment is found that is within its required signal areas is found, that endpoint becomes the new starting point and another straight line is created. This is done recursively to the entire path, and gets rid of issues with hitting overlapping signal

areas more than once. Results from this straight line optimization are shown in the final GAFO output in Figure 5.6

The data suggests that this research endeavor was successful in the creation of a heuristic (fuzzy) algorithm that improved the genetic algorithm solution by a significant percent in a somewhat realistic path planning simulation. As can be seen in Figure 5.8, all quadrants of each box plot were above or equal to a 5% difference, meaning the fuzzy optimization system was statistically successful in decreasing the path distance by at least 5% in most cases.

The use of fuzzy logic to create an optimal solution between uncertain targets was also a significant research step, as it is not only useful but necessary in many important real world applications.

Chapter 6

Collaborative Learning using Fuzzy Inferencing (CLIFF) - An Exploration of Type-2 Fuzzy Logic

There are a growing number of aerospace applications demonstrating the effectiveness of emulating human decision making with fuzzy logic. This overall project includes an investigation of the benefits of type-1 and type-2 fuzzy logic, as well as creation of the framework for a fuzzy system that can adapt to a situation using one of the two aforementioned fuzzy logic types in the form of a fuzzy robotic coach. This coach would the ability to better its team in a previously created fuzzy MATLAB game of PONG based upon its ability to adapt and learn from its opponent. In this effort, through extensive studying and understanding of type-2 fuzzy theory, a type-2 fuzzy logic toolbox was developed in the MATLAB environment. This system was compared to a benchmark problem, which raised the question of

type-2 logic effectiveness, however, as a created type-1 system showed better results than any results in the paper. Therefore due to type-2 systems being more computationally heavy, and difficult to program, it was concluded that cascading type-1 logic may be more suitable than type-2 logic in certain control applications. A framework for implementing the type-2 fuzzy logic system was also created, outlining how type-2 fuzzy logic could be cascaded to create an intelligent coach.

Overall, the developments of intelligence and learning within this project should prove to be an interesting stepping stone towards highly capable robots with the ability to learn from their environment; something that would be extremely helpful in disaster recovery, space exploration and military applications.

6.1 Problem Formulation

6.1.1 Development of a Type-2 Fuzzy Logic Toolbox

Type-2 fuzzy logic is unique in that its membership functions not only come in different shapes but also in different sizes. These shapes and sizes depend on if the mean or standard deviation is being varied, and by how much. Therefore before performing any actual coding it was necessary to learn about the different types of type-2 fuzzy logic that exists, and from that understanding decide on what type of type-2 membership function would be used in the simulation.

First, it was decided that a Gaussian primary membership function would make the most sense to use, as its gradual nature would allow for very smooth membership function outputs;

something that is important for an effective and efficient decision making system. Research conducted by Baklouti, Robert and Adel [7] suggested that a type-2 system that varies in standard deviation is more effective in controlling a system than one which varies by mean, so it was decided to use a Gaussian function that varies along its standard deviation as can be seen in Eqn. 6.1 below.

$$\mu_{\tilde{A}}(x) = exp[-\frac{1}{2}(\frac{x-m}{\sigma}^2]\sigma \in [\sigma_1, \sigma_2] \tag{6.1}$$

Through studying a benchmark problem provided by Dongrui and Woei [41] it was seen that using a singleton type-2 system would be best for comparison with a type-1 system as the outputs are more similar. It was decided that the type-2 system would be an interval system, which means that along the third dimension the type-2 membership functions would be constant which allows for a more simplistic function.

Once the type of type-2 logic to use was decided upon, it was then necessary to create a toolbox in MATLAB for the type of type-2 logic that would be used. Several type-2 fuzzy logic MATLAB files that have previously been created by Dr. Mendel are publicly available online on his website [19]. While the files didn't constitute a complete fuzzy toolbox, they were very important for understanding how to code a type-2 fuzzy system, and several of these files were used in creating this toolbox. Several MATLAB functions had to be created including the functions that determine the shape and effects of each type-2 membership function, as well as the function to communicate between the system being controlled and the type-2 system.

The final MATLAB toolbox code works as follows: an input is taken into the fuzzy system

where it is then fuzzified by the type-2 fuzzy membership functions. These membership functions are more complex than type-1 functions since a crisp input point results in a fuzzified set of numbers. Therefore a special MATLAB function was necessary to determine (1) overlapping membership function results and (2) what rules would be fired as both of these steps are extremely different and much more complex than a similar situation in a type-1 system. The firing of the rules creates an array which is then type reduced; a step that is unique to type-2 fuzzy logic. Type reduction takes the array from the rules and turns it back into a set of numbers that can then be defuzzified to produce a crisp output from the system.

Creating the entire system for just this one type of type-2 fuzzy logic was extremely time consuming, but very rewarding as it provided the researcher with a much better understanding of the inter-workings of type-2 fuzzy logic. Developing a full toolbox to be able to compute more shapes and types of type-2 systems would be a very useful future endeavor, however extends far beyond the time constraints of this research project.

The created software was then tested to see if it created useful Gaussian singleton interval type-2 fuzzy membership functions. The system was used to re-create both input membership functions found in the benchmark problem which can be found in Figures 6.1, 6.2, 6.3, and 6.4.

The creation of the membership functions was considered a success, and when tested manually with inputs the system produced results from these membership functions that made sense for controlling the water level in a drum.

The next step was to use the created simulation for the benchmark problem (described in part C below) to alter the input and output membership functions and create a system with

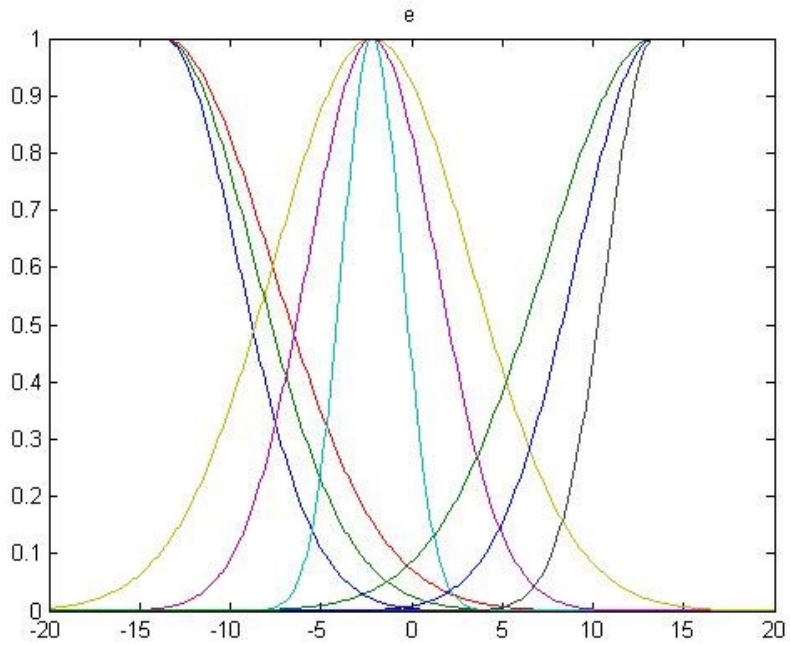


Figure 6.1: Produced type-2 membership functions for input variable e

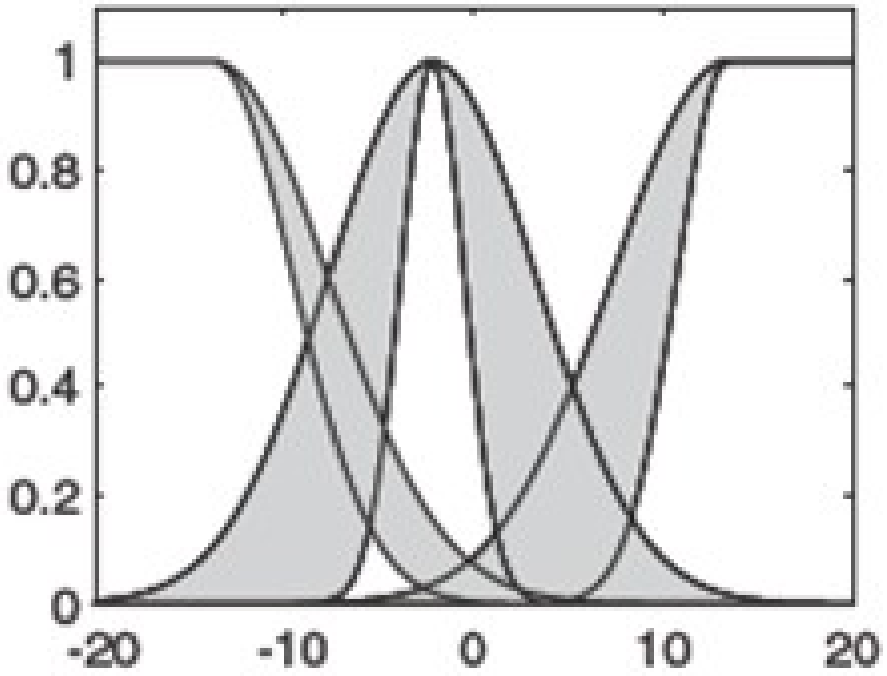


Figure 6.2: Type-2 membership functions for input variable e as presented in the benchmark problem [41]

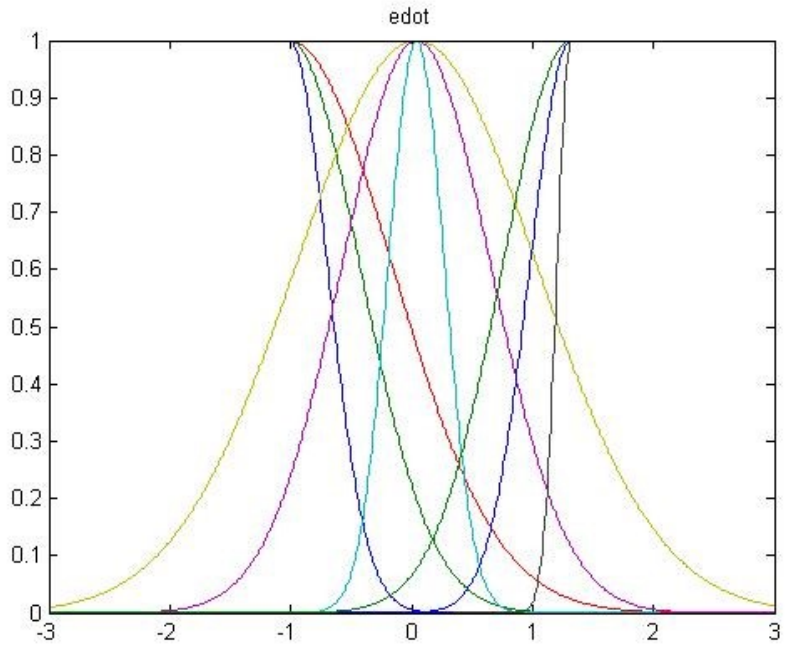


Figure 6.3: Produced type-2 membership functions for input variable $\dot{e}$

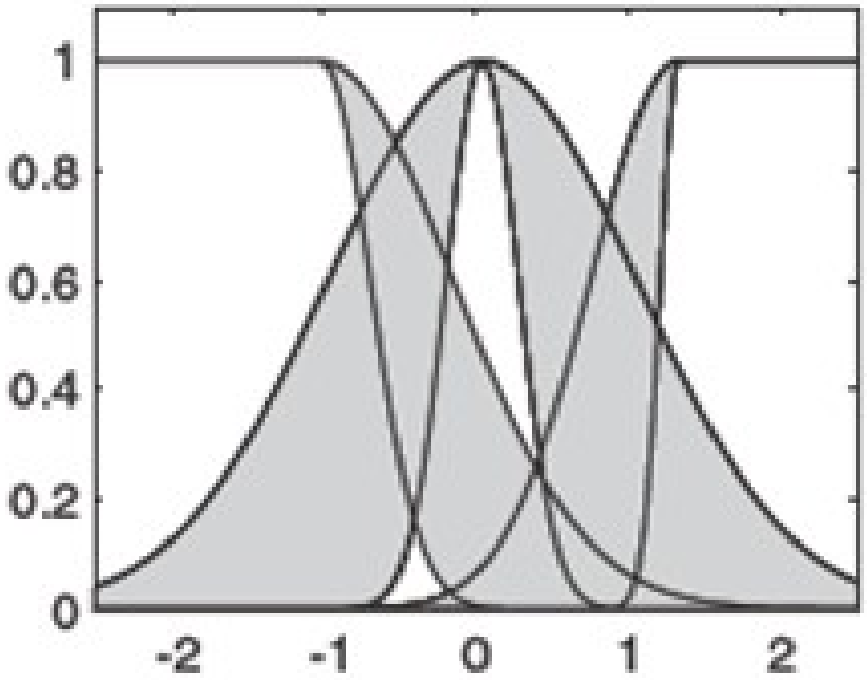


Figure 6.4: Type-2 membership functions for input variable $\dot{e}$ from the benchmark problem [41]

little to no oscillation and little to no error in the final output. The final input membership functions for inputs e and $\dot{e}$ can be found in Figures 6.5 and 6.6. The outputs for this system were crisp type-1 triangles, centered over the following positions: 0 (cm3/s), 23 (cm3/s), 44.69 (cm3/s), 60 (cm3/s) and 74 (cm3/s). It was determined that these were the best positions for the most responsive output that neither oscillated nor overshot the goal water level height. The inputs and outputs were connected with a set of 9 simple rules.

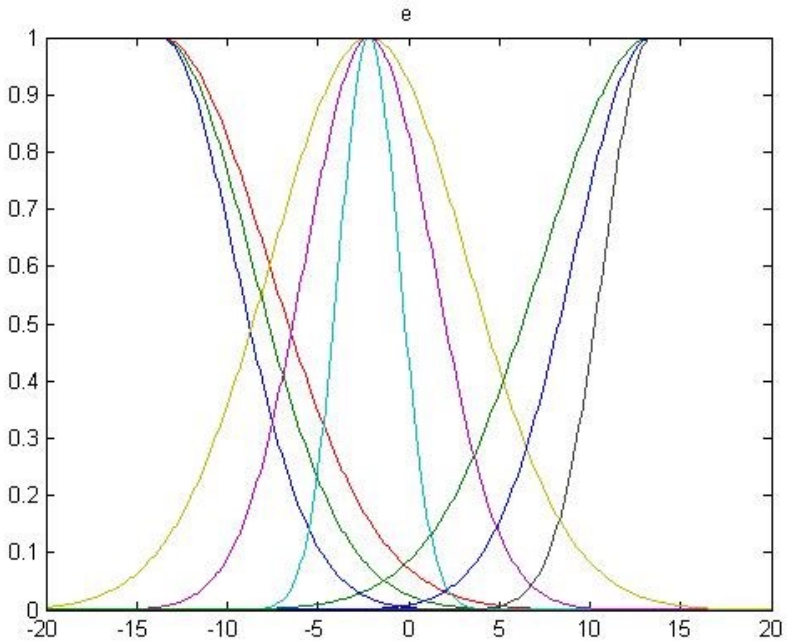


Figure 6.5: Modified Type-2 membership functions for input e.

6.1.2 Creation of a Type-1 System for Comparison

In working with the benchmark problem, it was questioned if there was potential for improvement on the type-1 results presented in the benchmark problem. This would be a crucial thing to note as it only makes sense to compare the best type-2 system possible with the best type-1 system possible.

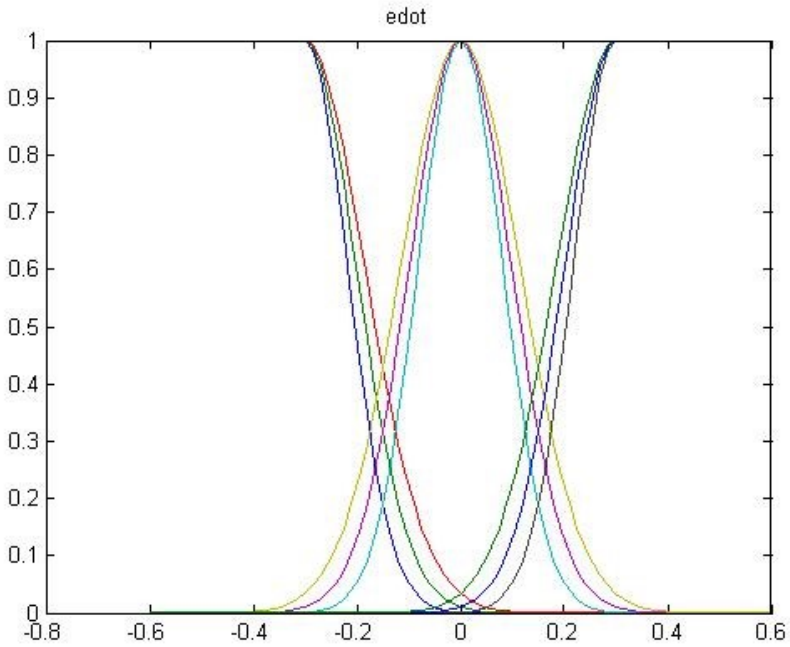


Figure 6.6: Modified Type-2 membership functions for input $\dot{e}$

Using previous knowledge of type-1 fuzzy system behavior, the following system was created with the same input and output variables (but different membership functions, obviously) to allow for a fair comparison. This also allowed the systems to be interchangeable within the simulation which allowed for more efficient programming.

The input membership functions can be found in Figures6.7 and 6.8. Several things can be noted from the input membership functions that may increase the performance of the system. First, the membership functions for the error, e, are rounded which allows for a smooth reaction of the controller. The membership functions become smaller around the desired point where the error equals zero, as this makes the system more sensitive to a change in error. However it can be noted that the membership functions are not as narrow as they could be. This width was intentional due to the dynamics of the scenario in the benchmark problem. During testing of the system it was noted that having membership functions too

wide would allow the output to settle with a slight amount of error, which membership functions that were too narrow would create an output that would oscillate more frequently. Therefore, this width was determined to give an output with little to no error upon settling and little to no oscillation.

The input membership functions for $\dot{e}$ show a similar situation, however, it can be noted that the membership functions for this input are narrower than the membership functions for e. This allowed the system to be much more sensitive to a change in error without adding any error or oscillation to the system. This was extremely useful as it allowed the system, upon reaching the goal height, to be very sensitive to any fluctuation in water level, and therefore hold the output steady.

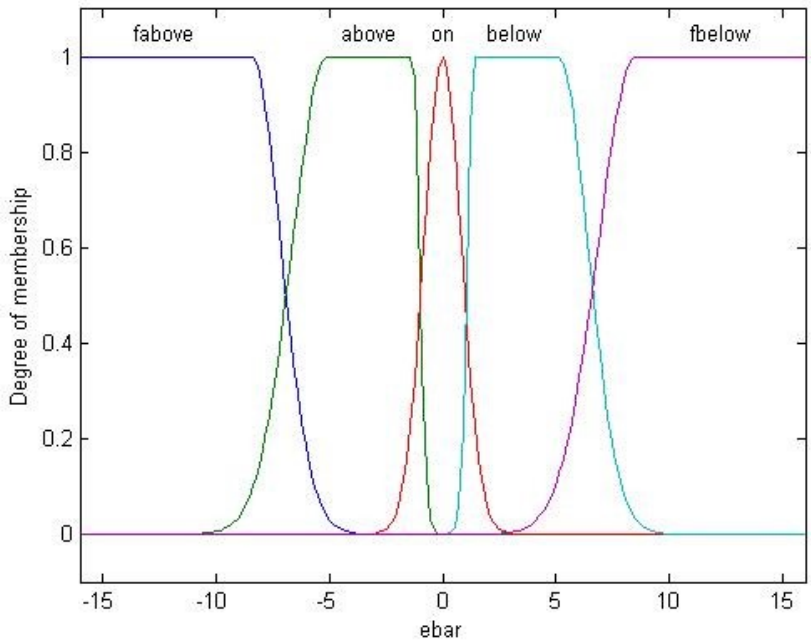


Figure 6.7: Input membership function for the created Type-1 fuzzy logic system.

The output membership functions can be found in Figure 6.9. One can see these functions are also rounded, and become larger the further away they are from "on", the point where

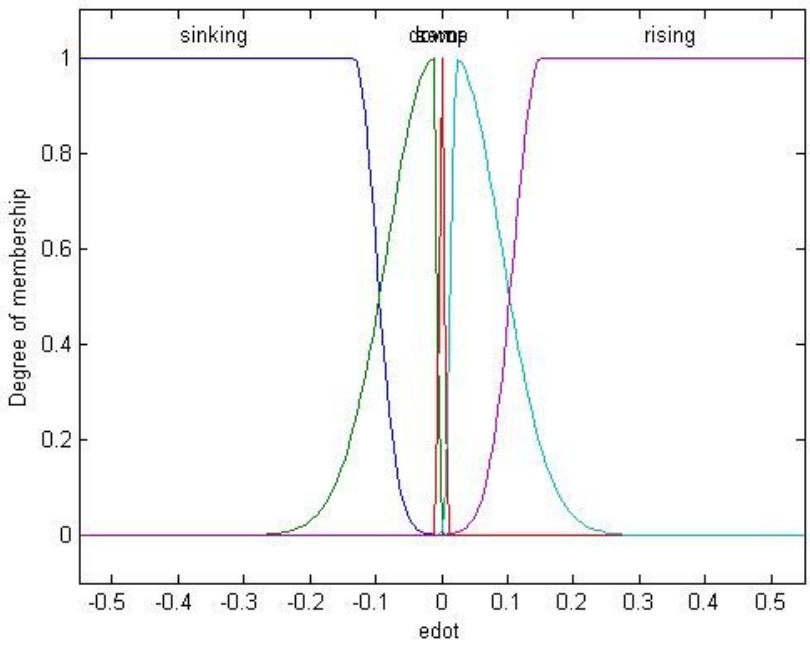


Figure 6.8: Input membership function for the created Type-1 fuzzy logic system.

the fluid flow in is equivalent to the fluid flow out (so the system should stabilize). This allows for the system to be more sensitive as it comes closer to its goal. As with the input membership functions for e, these membership functions were also created to be somewhat wide to relieve the system of oscillation. It was found that making the "up" membership function more narrow kept the system from oscillating in the event of an overshoot, as the filling of the drum slowed down more upon reaching the goal. The rule structure of the type-1 fuzzy system was quite simple, with 25 rules.

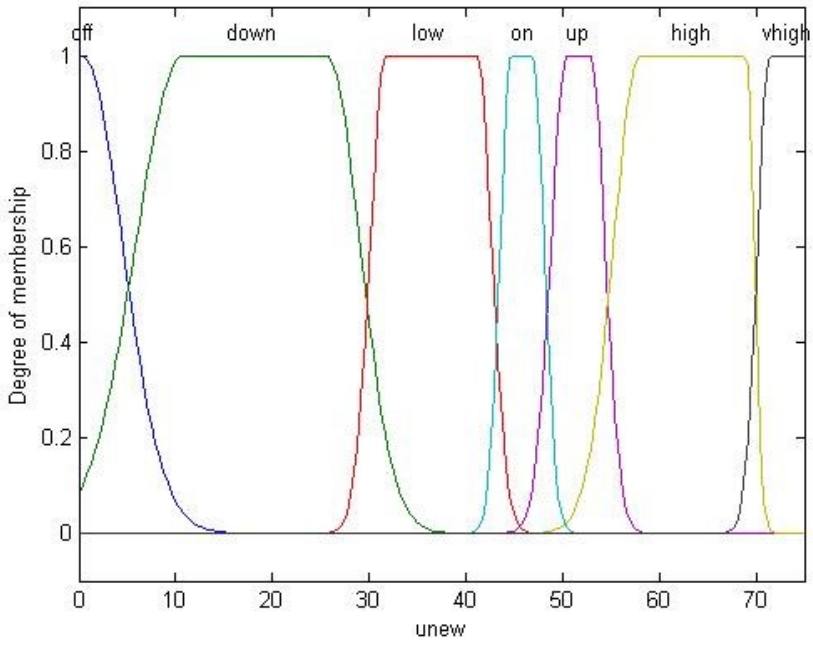


Figure 6.9: Output membership function for the created Type-1 fuzzy logic system.

6.1.3 Creation of a Simulation to Emulate the Benchmark Problem

In order to fully verify that the created type-2 system and toolbox were in true working order, it was necessary to re-create a simulation environment for testing that had results for comparison. For this, the problem presented by Dongrei and Woei [41] was used, as it had a straightforward problem scenario, included the dynamic equations that control the dynamic system and had interesting results for comparison.

To start, the dynamic scenario had to be re-created in a MATLAB environment. The presented scenario was that the type-2 system was to control the flow rate of water being pumped into a set of drums which are divided by a wall. There are holes in the bottom of both drums as well as a space in the wall for water to flow into the second drum, as can be

seen in Figure 6.10. In order to simulate this dynamic system, some equation manipulation was necessary in order to use the equations given in the text with the MATLAB ode45 solver. The ode45 solver uses a Runge-Kutta method in order to bring a set of differential equations to their solution iteratively. The following differential equations (Eqns. 6.2 and 6.3) were given to describe the system, where H_1 is the height of the water in drum one, and H_2 is the height of the water in drum two.

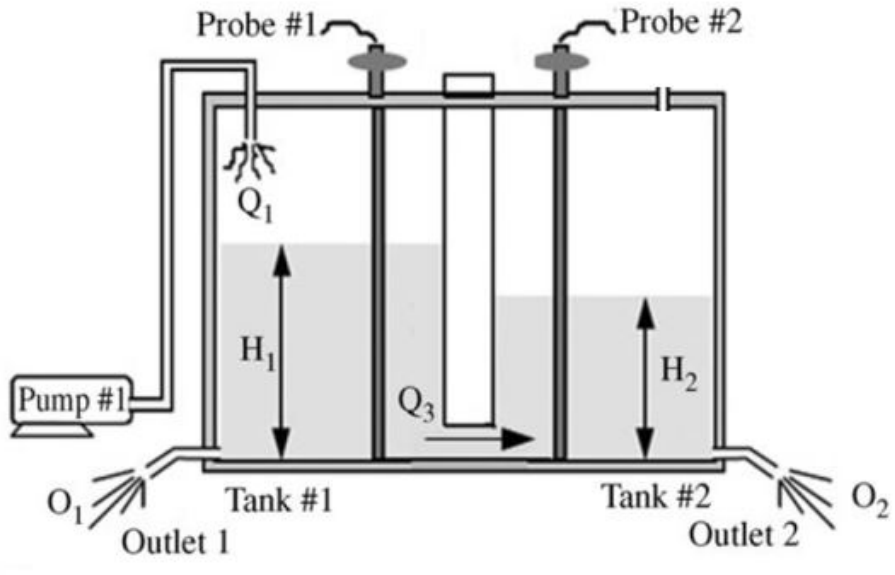


Figure 6.10: Schematic diagram of the dynamic system being controlled for the benchmark problem. Note: probes do not restrict or effect water flow.

$$A_1 \frac{dH_1}{dt} = Q_1 - \alpha_1 \sqrt{H_1} + \alpha_3 \sqrt{H_1 - H_2} \tag{6.2}$$

$$A_2 \frac{dH_2}{dt} = -\alpha_2 \sqrt{H_2} + \alpha_3 \sqrt{H_1 - H_2} \tag{6.3}$$

A was the area of each drum, while α was a constant for the amount of water lost through holes in the bottom of the drums. These constants could be defined as follows:

$A_1 = A_2 = 36.52cm^2, \alpha_1 = \alpha_2 = 5.6186cm^2$ and $\alpha_3 = 10cm^2$. Q was the output from the fuzzy systems, which was an instantaneous mass flow rate of water into the system (Q_1 in Figure 6.10).

Each of these equations was then used to derive the following differential equations which define the change in water height for both drum 1 and drum 2.

$$\frac{dH_1}{dt} = fracQ_1 - \alpha_1\sqrt{H_1} + \alpha_3\sqrt{H_1 - H_2}A_1 \tag{6.4}$$

$$\frac{dH_2}{dt} = \frac{-\alpha_2\sqrt{H_2} + \alpha_3\sqrt{H_1 - H_2}}{A_2} \tag{6.5}$$

The MATLAB model could then use Eqns. 6.4 and 6.5 in the ODE45 solver to simulate the solution for the benchmark problem.

6.2 Results

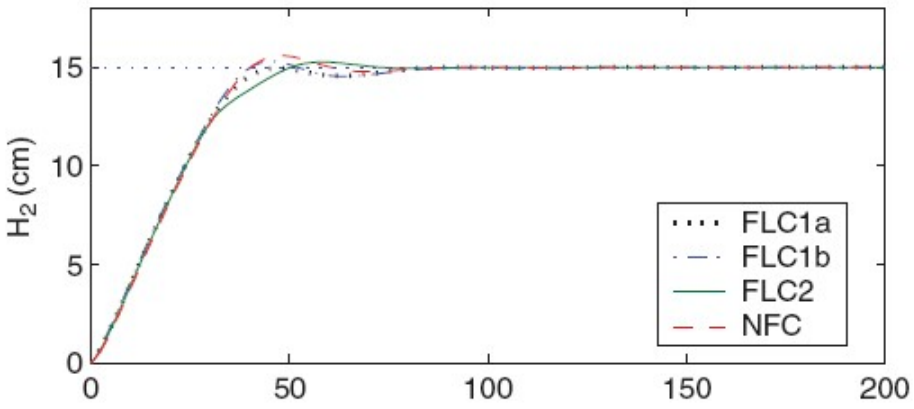


Figure 6.11: Results obtained by the benchmark problem [41]

Figure 6.11 gives the results from the benchmark paper, so that the reader may more

easily compare the performance of the type-1 and type-2 systems created in this chapter to the methods compared in the paper. FLC1a is a Type-1 fuzzy system with 3 membership functions in each input and output. FLC1b is a Type-1 fuzzy system with 5 membership functions in each input and output. FLC2 is a Type-2 fuzzy system, and NFC is a neural-fuzzy controller.

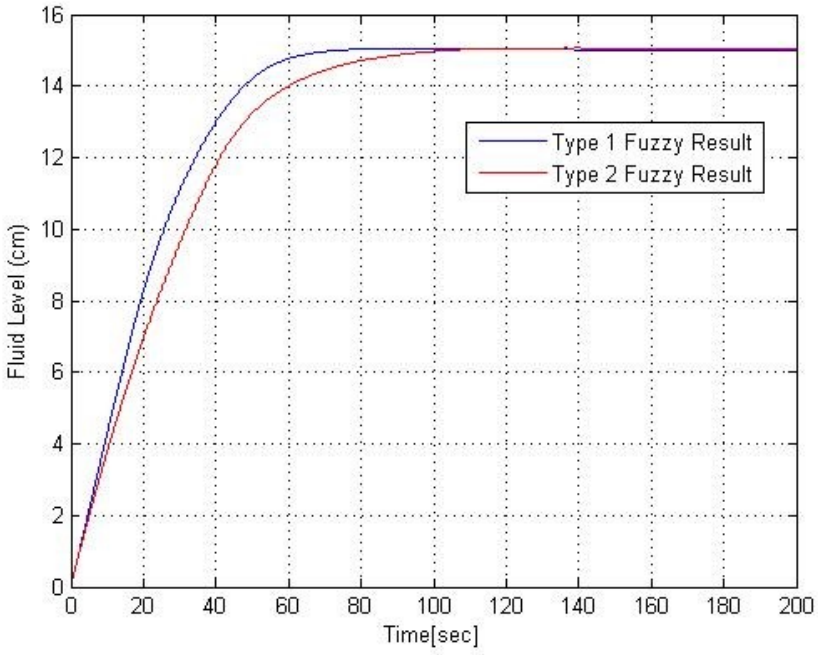


Figure 6.12: Results obtained by this study

One can see in Figures 6.11 and 6.12 that all of the systems fill the drum to the target height of $15cm$; however, the results for CLIFF do not overshoot the limit. One can also see that not only does the Type-1 result created in this project outperform the Type-2 result created in this project, but it also outperforms all of the fuzzy and neural network solutions presented in the benchmark problem, as it settles within 1% of the goal height (15 cm) in 69.1 seconds and does not overshoot the goal. The type-2 system created in this project settles within a time of 90 seconds and also does not overshoot its goal. Meanwhile, the

Statistic	Type 1 FIS	Type 2 FIS
Average Convergence Time (sec)	5.247783	10.35766
Standard Deviation	0.650419	1.111025
Variance	0.423045	1.234376
Percent Difference	50.67	–

Table 6.1: Convergence time performance comparison for type-1 and type-2 systems

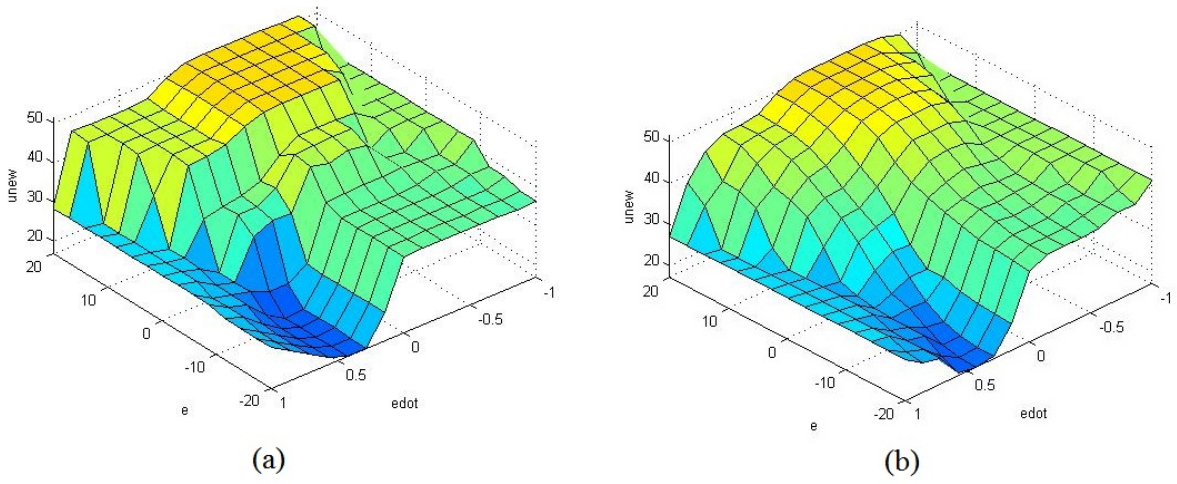


Figure 6.13: Type-2 fuzzy system output surfaces for upper (a) and lower (a) membership functions

fuzzy systems created in the benchmark problem not only overshoot the goal before settling, but settle in 80 seconds. This result is extremely important, as it shows that while a Type-2 fuzzy system can give comparable results to a Type-1, in the end the Type-1 result is still better in this application.

Table 6.1 gives a comparison of the performance of the type-1 and type-2 systems with respect to convergence time. This data was collected by recording the convergence time of both systems solving the benchmark problem 100 times each. The average difference is the difference between the two average times of convergence for each method.

Figures 6.13 and 6.14 give an interesting look at the output surfaces produced by the type-2 system, and how they compare to the output surface of the created type-1 system

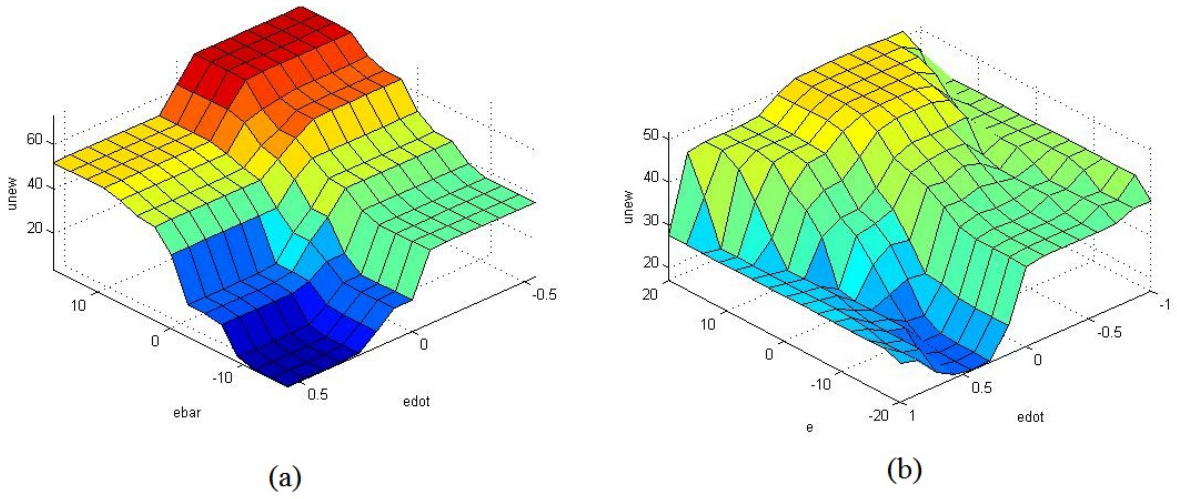


Figure 6.14: Comparison of output surface for type-1 (a) and type-reduced type-2 (b) systems.

for this benchmark problem. The type-2 system has multiple surfaces, one for the lower membership function boundary (Figure 6.13a), one for the upper (Figure 6.13b), and one for the type-reduced membership functions (Figure 6.14b). When a type-2 system is type-reduced, the membership functions become lines that aren't necessarily straight but occur within the boundaries of the upper and lower membership functions. Different type-reduction methods will produce different results, however given the same parameters, the type-reduced surface will not change. Therefore when looking at these surfaces, one can see how the type-reduced surface given in figure 6.14b exists within the boundaries of the surfaces produced by the upper and lower membership functions in figure 6.13.

6.3 Discussion and Conclusions

The data suggests several things. First the type-2 MATLAB fuzzy logic toolbox that has been created during this research gives results which make sense when verified with a comparable

benchmark problem as found in Reference 2. This development will be extremely important in the future for working with Type-2 systems and determining when the tool may be useful to a research endeavor over a Type-1 system.

The data also shows that the benchmark problem did test their Type-2 fuzzy system against decent Type-1 systems, but the fact that a better Type-1 system could be made shows that Type-2 logic may not be superior to Type-1 logic in all cases. This was a very interesting note because the whole argument for using type-2 fuzzy logic is that it is suggested by some to always perform better than a type-1 system. This does not necessarily mean that a type-2 system is useless, however. Applications where there is a lot of noise or uncertainty may still show a Type-2 system as being more useful, however it is uncertain how it would perform in comparison to a layered Type-1 system which is much less computationally heavy. Something interesting that was noted during development of the type-2 system was that it took much less tuning with the membership functions to get a result that did not oscillate or overshoot the goal. This could be due to the membership functions being footprints of uncertainty, so the user did not have to get them in the exact correct positions as they did with the Type-1 membership functions. This feature could be of use in future applications, however between the added computational effort involved with using type-2 and being outperformed by a Type-1 system, it has not yet shown superiority over very precise type 1 systems.

Table 6.1 gives a comparison of the type-1 and type-2 systems in terms of time to convergence, giving an idea of the overall performance of the two systems. It should be noted that the type-1 system, on average, converges twice as quickly as the type-2 system. This is an important piece of data, because in this application the type-1 system gives a better result twice as quickly which suggests that a type-1 system is a better choice for an optimization

application like the benchmark problem.

In solving this benchmark problem with the new toolbox, the data provides a unique look at the inner-workings of a type-2 fuzzy inference system. Figure 6.14 allows for a comparison to be made between the output surfaces of type-1 and type-2 systems. It is important to note that while in this example the surface output of the type-2 system is quite similar to the type-1 system, however some differences do exist. One major difference is the fact that the extremities of the outputs of the type-1 system are both higher and lower than the extremities present in the output of the type-2 system. This, as well as the difference in shape in the two surfaces, is thought to be due to the selection of type-reduction method in the type-2 system, differing numbers of membership functions and differences in input membership function positioning. The type-2 system only has 3 input membership functions for each input, whereas the type-1 system has 5 membership functions per input. This difference has a significant effect on the number of rules within the system, as the type-1 system has 25 rules while the type-2 system only has 9. In a more complex or cascading system and in an application more suited for a type-2 system (where type-1 and type-2 perform much more similarly) it is possible that this reduction of the rule base could allow the type-2 system to become faster than the type-1 system while still giving similar results. This would be a very interesting topic for future work.

In conclusion the creation of a type-2 fuzzy logic toolbox for a Gaussian singleton interval type-2 fuzzy inference system was considered a success, and the created Type-1 and Type-2 systems from this project were compared with the results of a benchmark problem. While the benchmark research paper considered the Type-2 fuzzy logic to be the greatest performer, it was found that a more precisely created Type-1 system could outperform it, both in result

and convergence time. Future work will include moving into phase two of CLIFF as described in section 6.4, as well as working with the Type-2 fuzzy toolbox on the side to determine where it may be useful.

6.4 Expansion of Type-2 Toolbox for use in Pong

Part of the overall objective of CLIFF is to use type-2 fuzzy logic to create a team of robots that learn from their environment to work effectively and collaborate within an uncertain spatio-temporal environment. This objective was to be met through creation of the framework for a robotic coach that uses cascaded type-2 fuzzy logic to alter the logic of the players, allowing them to learn from and beat their opponent. From the in-depth understanding of type two systems that learning how to make the aforementioned toolbox provided, it was then possible to create a cascaded system architecture for the fuzzy coach, and determine how the type-1 and type-2 systems would interact between the coach and the players. This system architecture can be found in Figure 6.15, and is described as follows.

During game play, an event occurs in which one of the players scores a point. When humans play a game such as PONG, our immediate reaction is to first closely observe the current state of the game. Therefore, the first step after a score in Figure 6.15 is to take in as much situational awareness as possible depending on if the computer scored (to answer the question "why did I score?") or if the opponent scored (to answer the question "why did they score?"). Events called "long volleys", in which the ball is hit many times before a score occurs or is hit many times in the same region of the court, are also recorded during game play, and considered part of the situational awareness.

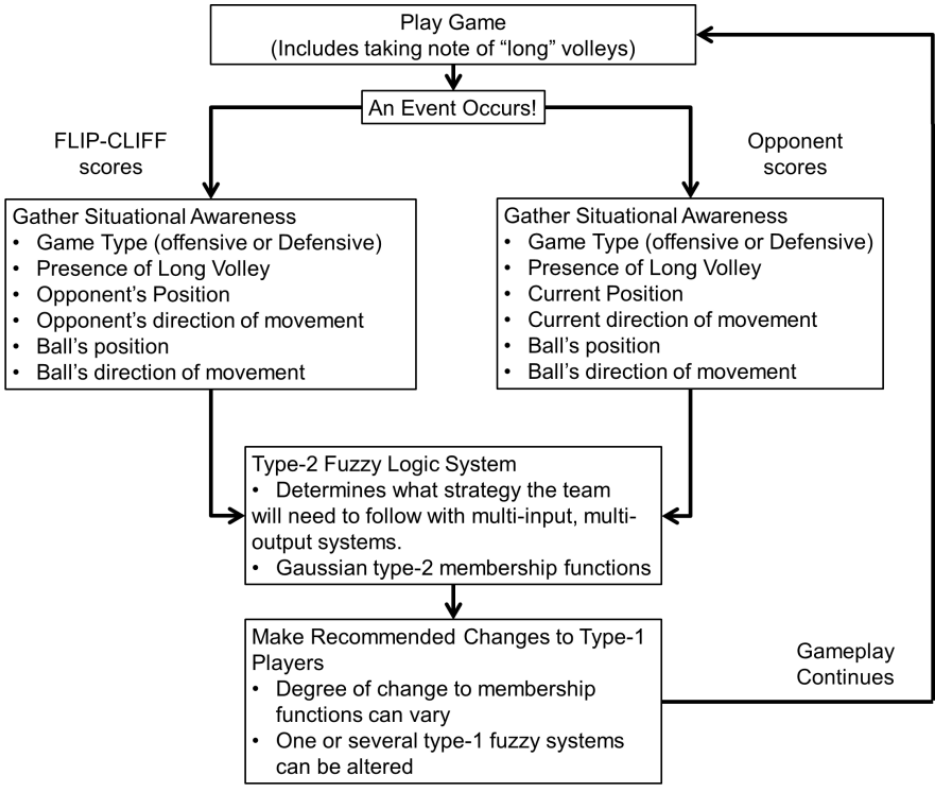


Figure 6.15: Framework for a type-2 fuzzy PONG coach

The next step in the algorithm is to use a type-2 fuzzy system within the coach to determine what strategy needs to be either changed or reinforced based on the previously recorded situational awareness. These changes are made within the fuzzy PONG player, and the game play continues until the next event occurs. Due to the computational speediness of fuzzy logic (once it has been initialized) this whole cycle would be extremely fast, allowing for the iteration of learning to occur smoothly during real time game play.

It is hypothesized that this cascaded type-2 fuzzy logic system that has been outlined would competently alter a type-1 team of fuzzy robots to improve and win against an opponent in the game of PONG. Creating the framework in Fig. 6.15 is an extremely useful takeaway from this research, as designing a robot or team of robots that can iteratively learn how to perform optimally in their environment will be a crucial development in the world

of artificial intelligence. No literature could be found that cascades type-1 and type-2 fuzzy systems, so this development would be extremely useful in exploring the abilities of these types of fuzzy logic. It is possible that the ability of the type-2 system to solve high degree of freedom problems effectively (and with noisy inputs) could be complimented quite well with the near-optimality of type-1 fuzzy inference system.

Having a robot team with the ability to perform in a variety of environments is not only efficient (you can give one robot many tasks instead of having to make a new one for each task), but also a useful development as one could then send these robots into unknown or uncertain environments where they could learn and become an optimal controller for that environment. Thinking of applications such as search and rescue, disaster cleanup, space exploration or military applications, the application of using type-2 logic in a cascading framework that can learn is considered to be a step in the right direction.

Chapter 7

Conclusions

In all of the above examples, cascading type 1 and/or 2 fuzzy logic is used to create better outcomes in an artificial intelligence applications. Cascading logic is shown both mathematically and through examples to reduce the solution space of complex problems, making them more solvable. From creating a system that can work effectively as a team in a spatio-temporal environment, to showing how the addition of a fuzzy logic optimizer improves on traditional solutions for complex traveling salesman problems, to allowing a sensible framework to be made for a system that teaches another robotic team to react to their surroundings, it can be concluded that cascading fuzzy logic is an effective and beneficial addition to artificially intelligent systems.

7.1 Previous Publications

Fuzzy Logic Inferencing in PONG [32] is the first chapter of the book Logic Programming: Theory, Practices and Challenges which was published in 2014. This chapter is based on the

same FLIP project that became Chapter 3 of this thesis. An addition to the chapter was a study on adding rotation to FLIP, and how that effected the established cascading fuzzy system.

Comparison of Fuzzy Optimization and Genetic Methods in Solving a Modified Traveling Salesman Problem [33] is a manuscript currently in the process of being published in the Journal of Aerospace Information Systems (JAIS). In this paper the authors compare their methods of solving a modified traveling salesman problem to determine which method is more effective (makes better results) and which (if there is a difference) is quicker to run. One of the two methods being compared is the Genetic Algorithm with Fuzzy Optimization (GAFO) method that is described in detail in Chapter 5 of this thesis. The results of this paper were quite interesting, as comparing two methods that both use fuzzy logic and genetic algorithms (but in slightly different ways) produced two very different systems.

It is also worth noting that the cascading architecture and methods focused on in this thesis has had some level of influence on other research in the author's group. The GAFO TSP solution method was referenced in research by Boone, Sathyan and Cohen [5] as they were investigating enhanced approaches of solving modified traveling salesman problems. The cascading architecture outlined in FLIP has been referenced and the cascading idea built upon by several others in the MOST-Aerospace labs at the University of Cincinnati.

7.2 Future Work

Ideas from this thesis could be expanded in several ways. It would be very interesting to find other applications of cascading logic as a whole, weather that includes just type-1 fuzzy

logic of a mixture of types 1 and 2 fuzzy logic along with other methods such as neural networks and genetic algorithms. Since cascading logic has been shown to greatly reduce the complexity of the solution space for problems, it would be most meaningful to find highly complex applications. These applications could be in deep learning, task allocation and collaboration, managing priorities for complex systems, or even things that are less directly related to engineering such as determining medical diagnoses or making financial market decisions. Since large cascading logic systems, and large cascading fuzzy logic systems in particular, are just beginning to be studied there are many applications that can be explored.

Another expansion of this thesis could look at type-2 fuzzy logic and its strengths. It would be extremely useful to determine which control problems would better respond to Type-2 fuzzy logic instead of Type-1. For example in the case of path planning, due to the nature of Type-2 systems, positive results might allow for quick optimization of a path between irregularly shaped, fuzzy target areas. Cascaded type-2 logic could also perform better than cascaded type-1 logic in deep learning applications due to the ambiguity of its membership functions. An exciting study might be to compare cascaded type-2 logic to neural networks in learning applications.

While continuing work on CLIFF, it would be useful to create a performance metric that measures the abilities of each robot team. An example of this metric could be analyzing the elapsed time before each point scored, the speed of the ball each time a point is scored, or how many times each player (or team) hits the ball before a point occurs. In this way the player(s) could then be evaluated more objectively, which would allow for learning within the system to be measured (has my player gotten better over time?) as well as give the type-2 coach a metric to use while deciding how to alter the player or team's fuzzy systems.

This research could also be expanded into multi-TSP problems, which means there are multiple vehicles that need to visit multiple targets. The targets in this situation are (through some means) divided between vehicles, and then each vehicle does its own TSP to travel to its assigned targets. It is easy to see how a cascaded system could be applied in this scenario. Any signs of multi-TSP research using Type-2 fuzzy logic could not be found, suggesting this idea could be an interesting development in fuzzy multi-TSP research.

PROFIT is an interesting concept in that it layers fuzzy logic with a genetic algorithm in a novel way. Instead of integrating the two methods as is usually done, PROFIT very clearly has a GA layer and a FIS layer. Therefore, it would be an interesting study to replace the genetic component in PROFIT with a different, more effective GA. Compared to current methods, it may be worth it to see how having a layer of fuzzy optimization can improve upon current Genetic Fuzzy Algorithm Traveling Salesman Problem (GFATSP) solution methods.

Bibliography

[1] D. L. Applegate, R. M. Bixby, V. Chvatal, and W. J. Cook. *The Travelling Salesman Problem*. Princeton University Press, 2007.

[2] Ahmet Arslan and Mehmet Kaya. Determination of fuzzy logic membership functions using genetic algorithms. *Fuzzy Sets and Systems*, 118(2):297 – 306, 2001. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/S0165-0114(99)00065-2. URL http://www.sciencedirect.com/science/article/pii/S0165011499000652.

[3] Hamid R. Berenji. A reinforcement learning—based architecture for fuzzy logic control. *International Journal of Approximate Reasoning*, 6(2):267 – 292, 1992. ISSN 0888-613X. doi: http://dx.doi.org/10.1016/0888-613X(92)90020-Z. URL http://www.sciencedirect.com/science/article/pii/0888613X9290020Z.

[4] H.R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *Neural Networks, IEEE Transactions on*, 3(5):724–740, Sep 1992. ISSN 1045-9227. doi: 10.1109/72.159061.

[5] Nathan Boone, Anoop Sathyan, and Kelly Cohen. Enhanced approaches to solving hte

multiple travelling salesman problem. In *Conference Proceedings of 2015 AIAA Infotech at Aerospace Conference*, 2015.

[6] Luis Castillo, Antonio Gonzalez, and Raul Perez. Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm. *Fuzzy Sets and Systems*, 120(2):309 – 321, 2001. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/S0165-0114(99)00095-0. URL `http://www.sciencedirect.com/science/article/pii/S0165011499000950`.

[7] Oscar Castillo. *Type-2 Fuzzy Logic in Intelligent Control Applications*. Springer, 2012.

[8] J. Collings and Eunjin Kim. A distributed and decentralized approach for ant colony optimization with fuzzy parameter adaptation in traveling salesman problem. In *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, pages 1–9, Dec 2014. doi: 10.1109/SIS.2014.7011805.

[9] E.P. Dadios and Jr. Maravillas, O.A. Fuzzy logic controller for micro-robot soccer game. In *Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE*, volume 3, pages 2154–2159 vol.3, 2001. doi: 10.1109/IECON.2001.975627.

[10] N. Ernest, K. Cohen, C. Schumacher, and D. Casbeer. Learning of intelligent controllers for autonomous unmanned combat aerial vehicles by genetic cascading fuzzy methods. *SAE Technical Papers*, 2014. doi: 10.4271/2014-01-2174.

[11] N. Ernest, K. Cohen, E. Kivelevitch, C. Schumacher, and D. Casbeer. Genetic fuzzy trees and their application towards autonomous training and control of a squadron

of unmanned combat aerial vehicles. *Unmanned Systems*, 3(3):185–204, 2015. doi: 10.1142/S2301385015500120.

[12] N. Ernest, E. Garcia, D. Casbeer, K. Cohen, and C. Schumacher. Multi-agent cooperative decision making using genetic cascading fuzzy systems. *Proceedings of the 2015 AIAA Infotech@Aerospace Conference*, 2015. doi: 10.2514/6.2015-0888.

[13] Hsuan-Ming Feng and Kuo-Lung Liao. Hybrid evolutionary fuzzy learning scheme in the applications of traveling salesman problems. *Information Sciences*, 270(0):204 – 225, 2014. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2014.02.098. URL `http://www.sciencedirect.com/science/article/pii/S0020025514002047`.

[14] H.A. Hagras. A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots. *Fuzzy Systems, IEEE Transactions on*, 12(4):524–539, Aug 2004. ISSN 1063-6706. doi: 10.1109/TFUZZ.2004.832538.

[15] Henning Heider and Thorsten Drabe. A cascaded genetic algorithm for improving fuzzy-system design. *International Journal of Approximate Reasoning*, 17(4):351 – 368, 1997. ISSN 0888-613X. doi: http://dx.doi.org/10.1016/S0888-613X(97)00003-0. URL `http://www.sciencedirect.com/science/article/pii/S0888613X97000030`. Genetic Fuzzy Systems for Control and Robotics.

[16] Joerg Heitkoetter and David Beasley. The hitch-hiker's guide to evolutionary computation. webpage, 2001.

[17] F. Herrera, M. Lozano, and J.L. Verdegay. A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and Systems*, 100(1–3):143 – 158,

1998. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/S0165-0114(97)00043-2. URL http://www.sciencedirect.com/science/article/pii/S0165011497000432.

[18] Nilesh N. Karnik and Jerry M. Mendel. Applications of type-2 fuzzy logic systems to forecasting of time-series. *Information Sciences*, 120(104):89 – 111, 1999. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/S0020-0255(99)00067-5. URL http://www.sciencedirect.com/science/article/pii/S0020025599000675.

[19] Nilesh N. Karnik, Qilian Liang, Feilong Liu, Dongrui Wu, Jhiin Joo, and Jerry M. Mendel. Type-2 fuzzy logic software. software, 2012.

[20] Kavita Khanna and Navin Rajpal. Reconstruction of curves from point clouds using fuzzy logic and ant colony optimization. *Neurocomputing*, 161(0):72 – 80, 2015. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2014.11.071. URL http://www.sciencedirect.com/science/article/pii/S0925231215002076.

[21] Joseph Kirk. Dijkstra's shortest path algorithm. Computer Software, 2006.

[22] Donald Knuth. *The art of Computer Programming*. Addison-Wesley, 1997. ISBN 0-201-89683-4.

[23] Yifan Li, P. Musilek, and L. Wyard-Scott. Fuzzy logic in agent-based game design. In *Fuzzy Information, 2004. Processing NAFIPS '04. IEEE Annual Meeting of the*, volume 2, pages 734–739 Vol.2, June 2004. doi: 10.1109/NAFIPS.2004.1337393.

[24] Qilian Liang, N.N. Karnik, and J.M. Mendel. Connection admission control in atm networks using survey-based type-2 fuzzy logic systems. *Systems, Man, and Cybernetics,*

Part C: Applications and Reviews, IEEE Transactions on, 30(3):329–339, Aug 2000. ISSN 1094-6977. doi: 10.1109/5326.885114.

[25] Chin-Teng Lin, Yin-Cheung Lee, and Her-Chang Pu. Satellite sensor image classification using cascaded architecture of neural fuzzy network. *Geoscience and Remote Sensing, IEEE Transactions on*, 38(2):1033–1043, Mar 2000. ISSN 0196-2892. doi: 10.1109/36.841983.

[26] Feilong Liu. An efficient centroid type-reduction strategy for general type-2 fuzzy logic system. *Information Sciences*, 178(9):2224 – 2236, 2008. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2007.11.014. URL `http://www.sciencedirect.com/science/article/pii/S0020025507005385`.

[27] S.S. Mahapatra, Santosh Kumar Nanda, and B.K. Panigrahy. A cascaded fuzzy inference system for indian river water quality prediction. *Advances in Engineering Software*, 42(10):787 – 796, 2011. ISSN 0965-9978. doi: http://dx.doi.org/10.1016/j.advengsoft.2011.05.018. URL `http://www.sciencedirect.com/science/article/pii/S0965997811001256`.

[28] Jeich Mar and Hung-Ta Lin. A car-following collision prevention control device based on the cascaded fuzzy inference system. *Fuzzy Sets and Systems*, 150(3):457 – 473, 2005. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/j.fss.2004.09.004. URL `http://www.sciencedirect.com/science/article/pii/S016501140400404X`.

[29] Ricardo Martinez, Oscar Castillo, and Luis T. Aguilar. Optimization of interval type-2 fuzzy logic controllers for a perturbed autonomous wheeled mo-

bile robot using genetic algorithms. *Information Sciences*, 179(13):2158 – 2174, 2009. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2008.12.028. URL http://www.sciencedirect.com/science/article/pii/S0020025509000073. Special Section on High Order Fuzzy Sets.

[30] Jerry M. Mendel. *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*. Prentice Hall, 2001.

[31] Jerry M. Mendel, Hani Hagras, and Robert I. John. Standard background matrial about interval type-2 fuzzy logic systems that can be used by all authors. *(Self published online)*, 2001.

[32] Sophia Mitchell, Brandon Cook, and Dr. Kelly Cohen. *Logic Programming: Theory, Practices and Challenges*, chapter Fuzzy Logic Inferencing for Pong (FLIP). NOVA Science Publishers, 2014.

[33] Sophia Mitchell, Dr. Nicholas Ernest, and Dr. Kelly Cohen. Comparison of fuzzy optimization and genetic methods in solving a modified traveling salesman problem. *Journal of Aerospace Information Systems*, 2015.

[34] Dilip Kumar Pratihar, Kalyanmoy Deb, and Amitabha Ghosh. A genetic-fuzzy approach for mobile robot navigation among moving obstacles. *International Journal of Approximate Reasoning*, 20(2):145 – 172, 1999. ISSN 0888-613X. doi: http://dx.doi.org/10.1016/S0888-613X(98)10026-9. URL http://www.sciencedirect.com/science/article/pii/S0888613X98100269.

[35] F. Russo and G. Ramponi. A noise smoother using cascaded fire filters. In *Fuzzy*

Systems, 1995. International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium., Proceedings of 1995 IEEE Int, volume 1, pages 351–358 vol.1, Mar 1995. doi: 10.1109/FUZZY.1995.409703.

[36] M. Russo. Genetic fuzzy learning. *Evolutionary Computation, IEEE Transactions on*, 4(3):259–273, Sep 2000. ISSN 1089-778X. doi: 10.1109/4235.873236.

[37] Roberto Sepulveda, Oscar Castillo, Patricia Melin, Antonio Rodriguez-Diaz, and Oscar Montiel. Experimental study of intelligent controllers under uncertainty using type-1 and type-2 fuzzy logic. *Information Sciences*, 177(10):2023 – 2048, 2007. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2006.10.004. URL `http://www.sciencedirect.com/science/article/pii/S002002550600332X`. Including Special Issue on Hybrid Intelligent Systems.

[38] Yuhui Shi and R.C. Eberhart. Fuzzy adaptive particle swarm optimization. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 101–106 vol. 1, 2001. doi: 10.1109/CEC.2001.934377.

[39] Chi-Hsu Wang, Chun-Sheng Cheng, and Tsu-Tian Lee. Dynamical optimal training for interval type-2 fuzzy neural network (t2fnn). *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(3):1462–1477, June 2004. ISSN 1083-4419. doi: 10.1109/TSMCB.2004.825927.

[40] D. Wu and W.W. Tan. A type-2 fuzzy logic controller for the liquid-level process. In

Fuzzy Systems, 2004. Proceedings. 2004 IEEE International Conference on, volume 2, pages 953–958 vol.2, July 2004. doi: 10.1109/FUZZY.2004.1375536.

[41] Dongrui Wu and Woei Wan Tan. Genetic learning and performance evaluation of interval type-2 fuzzy logic controllers. *Engineering Applications of Artificial Intelligence*, 19(8):829 – 841, 2006. ISSN 0952-1976. doi: http://dx.doi.org/10.1016/j.engappai.2005.12.011. URL http://www.sciencedirect.com/science/article/pii/S0952197606000388.

Appendix

FLIP FIS File Outlines

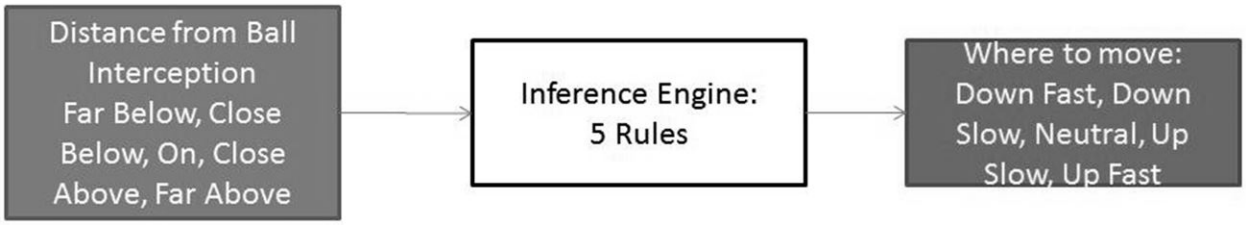


Figure 1: Paddle movement FIS

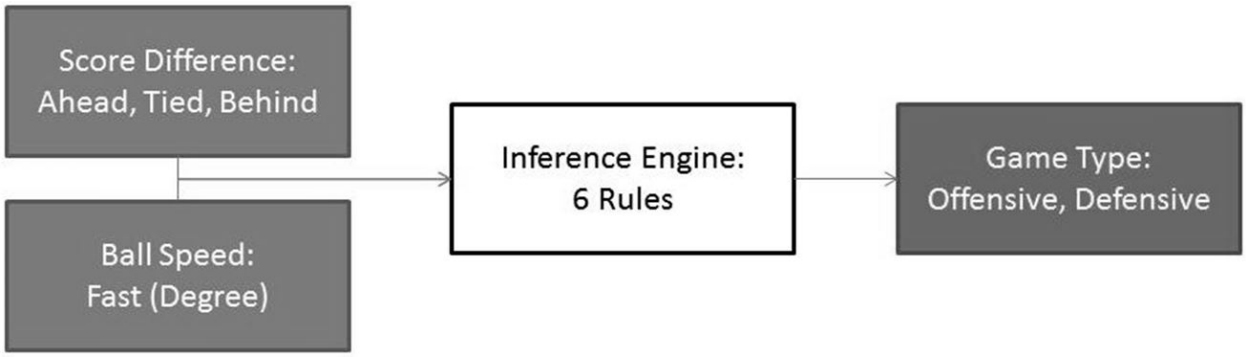


Figure 2: Game classification FIS

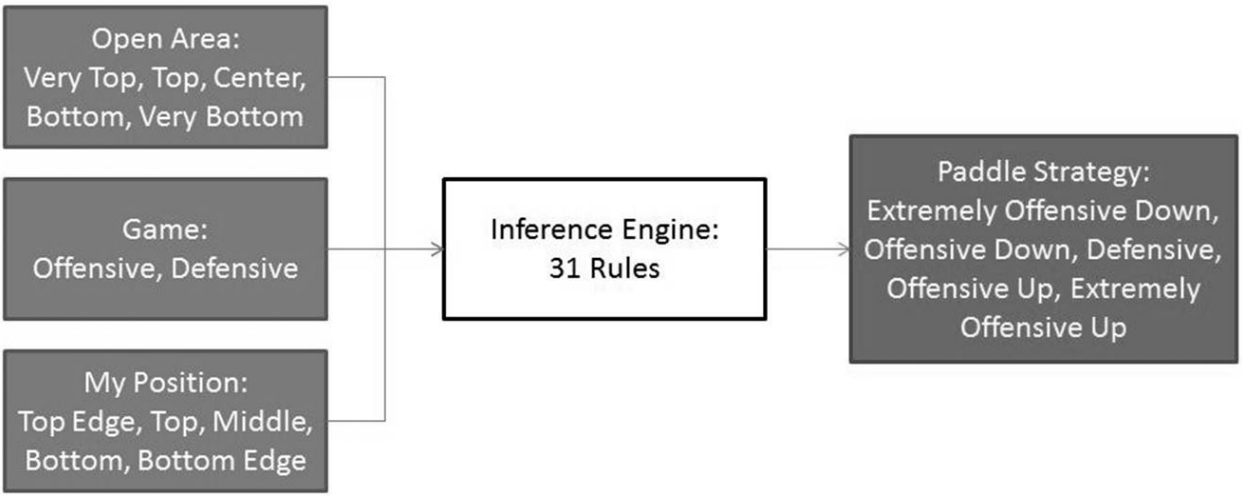


Figure 3: Front or singles paddle strategy FIS

Figure 4: Back paddle strategy

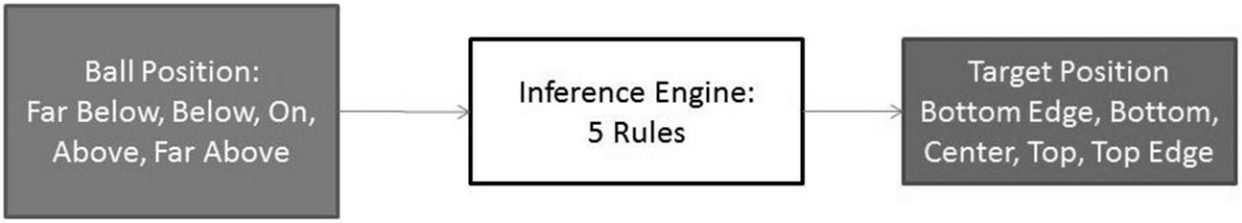


Figure 5: Front paddle waiting strategy

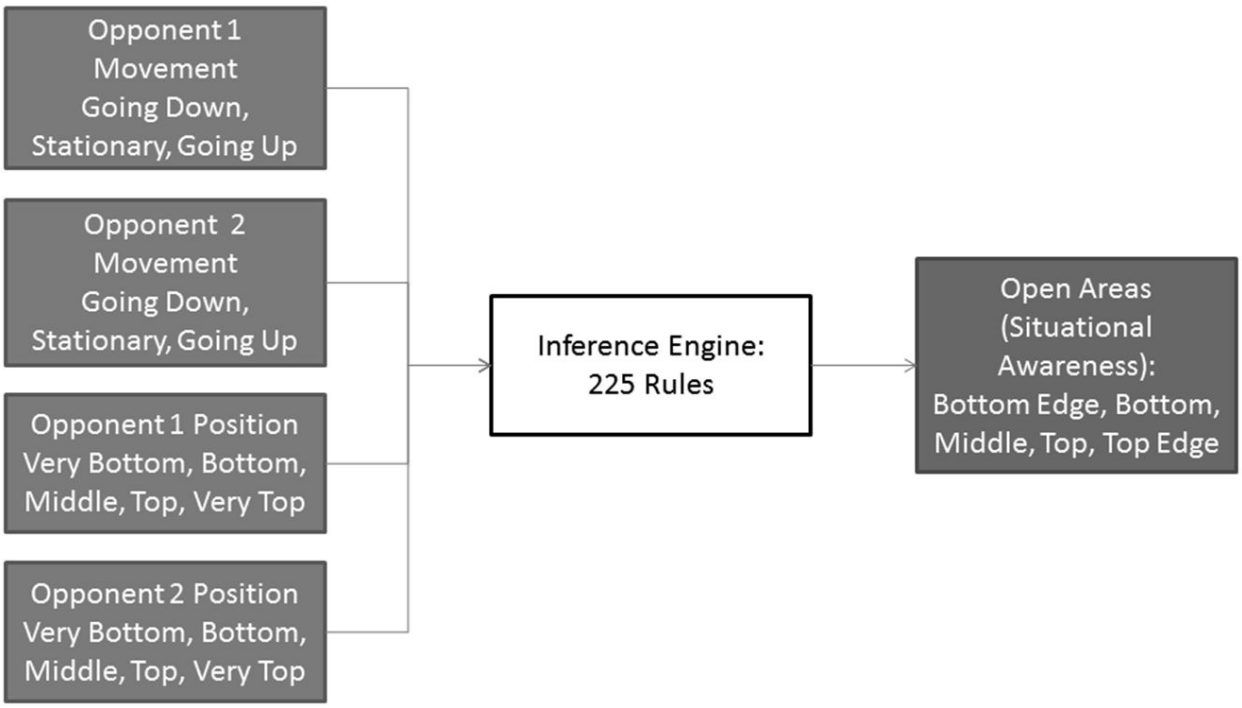


Figure 6: Back paddle situational awareness FIS

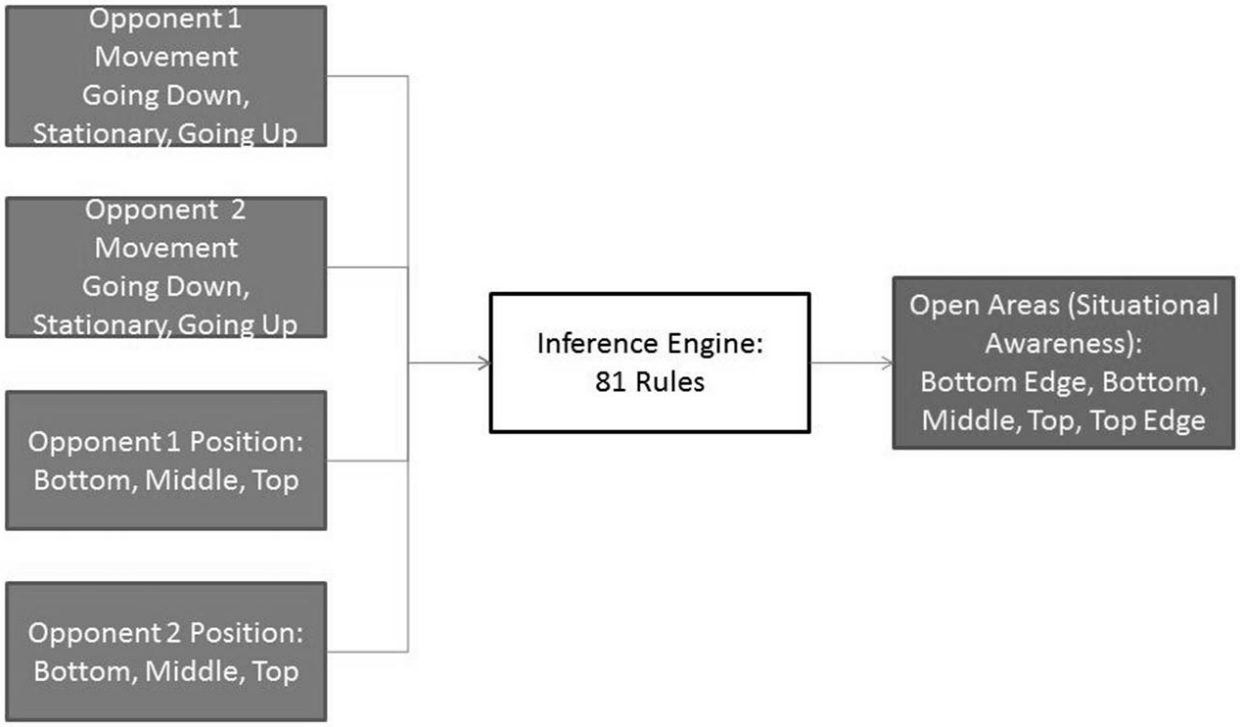


Figure 7: Font paddle situational awareness FIS