Full design of fuzzy controllers using genetic algorithms

Abdollah Homaifar
Ed McCormick

NASA Center of Research Excellence / Controls and Guidance Group
North Carolina A&T State University, Dept. of Electrical Engineering
McNair Building, Greensboro, North Carolina 27411

ABSTRACT

This paper examines the applicability of genetic algorithms in the complete design of fuzzy logic controllers. While GA has been used before in the development of rule sets or high performance membership functions, the interdependence between these two components dictates that they should be designed together simultaneously. GA is fully capable of creating complete fuzzy controllers given the equations of motion of the system, eliminating the need for human input in the design loop. We show the application of this new method to the development of a cart controller.

1.INTRODUCTION

Genetic algorithms (GA) are powerful search procedures based on the mechanics of natural selection. They use operations found in natural genetics to guide them through the paths in the search space. They provide a means to search poorly understood, irregular spaces. Because of its robustness, GA has been successfully applied to a variety of function optimizations, self-adaptive control systems, and learning systems.

Fuzzy systems arose from the desire to describe complex systems with simple tools. In contrast to boolean systems where an item either has a membership of {1} or {0} in a set, fuzzy systems allow for degrees of membership over the range {0-1}. This imitates the linguistic approach to describing conditions (i.e. cold, very warm) used in everyday life.

Interest in fuzzy controllers has recently been gaining in popularity across a broad array of disciplines and with good reason. Fuzzy controllers allow for a simpler, more human approach to control design and do not demand the mathematical modelling knowledge of more conventional control design methods. As systems become more complex, the ability to describe them mathematically becomes more difficult. For this reason, fuzzy controllers provide reasonable, effective alternatives to classical or state-space controllers.

By using a linguistic approach, it is easy to see that fuzzy theory can be integrated into control theory using rules of the form *IF*{condition} *THEN*{action}. Using these rules, one can create a functional controller. The problem with this method comes from determining the appropriate rules and determining the shape of the membership functions.

This work sought to use genetic algorithms in the design and implementation of fuzzy logic controllers. Previously, generation of membership functions had been a task mainly done either iteratively, by trial-and-error, or by human expert. A task such as this was a natural candidate for GA since GA will attempt to create membership functions that will cause the controller to perform optimally. In much the same manner, GA could be used to generate the rules which use these membership functions. Work had been done using GA to do each of these tasks separately, but since the two are co-dependent, using a hand-designed rule set with GA designed membership functions or hand-designed membership functions with a GA designed rule set does not use GA to its full advantage. Thus, the use of GA to determine both simultaneously and determine an optimal or near-optimal controller was the main objective of this work.

The problem used to check the effectiveness of this method was centering and stopping a cart located on a one-dimensional track as described by Thrift[1]. Given an initial velocity and location on the track, the objective was to determine a controller which will bring the cart to zero velocity and zero location in minimum time. Different controllers were designed for this problem by dividing the input and output spaces into different partition sizes.

2. GENETIC ALGORITHMS AND FUZZY CONTROLLERS

2.1 Genetic Algorithms

Genetic Algorithms are general purpose optimization algorithms with a probabilistic component. They provide a means to search poorly understood, irregular spaces. John Holland originally developed GA and provided its theoretical

foundation in his book, Adaptation in Natural and Artificial Systems[2]. Holland developed GA to simulate some of the processes observed in natural evolution. Evolution is a process that operates on chromosomes (organic devices for encoding the structure of living beings) rather than on living beings. Natural selection links chromosomes with the performance of their decoded structure. The processes of natural selection cause those chromosomes that encode successful structures to reproduce more often than those that do not. Recombination processes create different chromosomes in children by combining material from the chromosomes of the two parents. Mutation may cause the chromosomes of children to be different from those of their parents.

GA appropriately incorporates these features of natural evolution in computer algorithms to solve difficult problems in the way that nature has done -- through evolution. GA requires the problem to be maximized (or minimized) to be stated in the form of a cost (objective) function. In GA, a set of variables for a given problem is encoded into a string (or other coding structure), analogous to a chromosome in nature. These strings are converted to a numerical value and then linearly mapped over the range allowed for the variable. This value is then used to evaluate the cost function, yielding a "fitness." GA selects parents from a pool of strings (population) according to the basic criteria of "survival of the fittest." It reproduces new strings by recombining parts of the selected parents in a random manner. Although GA is a stochastic method, it is not a simple random walk. It exploits historical information to guide the search with improved performance.

The repopulation of the next generation is done using three methods: reproduction, crossover, and mutation. Reproduction means simply that strings with high fitnesses should receive multiple copies in the next generation while the strings with low fitnesses receive fewer copies or even none at all. Crossover refers to taking a fit string, splitting it into two parts at a randomly generated crossover point and recombining it with another string which has also been split at the same crossover point. This procedure serves to promote changes in the best strings which will give them even higher fitnesses. Mutation is the random alteration of a bit in the string. This will assist in keeping diversity in the population.

In explaining the inner workings of GA, let us initially make a few definitions[3]. Since we are dealing with binary strings, a notation must be developed to denote similarity subsets (*schemata*). A schema is a similarity subset which contains strings that have similarities at some bit positions. We can expand this thinking even further with the introduction of a wild card character, *, in addition to the binary set {0,1}. For example, the set {0001,0101,0011} can be described by the *similarity template* 0**1. Using this notation, we can now define a schema's *order* and *defining length*. For a given schema, h, its order o(h) is defined as the number of fixed bit positions within that schema. The defining length of a schema, $\delta(h)$, is the distance between the outermost defining positions of a schema. As an example, the schema 01****0 has order 3 and defining length 5.

With these definitions we can now present the fundamental theorem of genetic algorithms, the *schema theorem*[4]. The schema theorem enables us to calculate a lower bound on the expected number of a particular schema, h, following reproduction, crossover, and mutation[2,3]. The theorem is stated as:

$$\lambda(h,t+1) \geq \lambda(h,t)\frac{f(h)}{f}\left[1 - p_c\frac{\delta(h)}{l-1} - p_m o(h)\right] \qquad (1)$$

where λ is the expected number of schemata, t is the generation index, l is the overall string length, f(h) is the average fitness of those strings representing the subset h, $\bar{f}$ is the average fitness of the entire population, p_c and p_m are, respectively, the crossover and mutation probabilities. Examining the schema theorem, we see that it states that a schema will grow when it is short, has low order, and has above average fitness.

Given a history of genetic algorithms, one might ask what advantages does it have over other methods. GA's primary advantage over other methods is its robustness. GA works through function evaluation, not through differentiation or other such means. Because of this trait, GA does not care what type of problem it is asked to maximize, only that it be properly coded. Thus GA is able to solve a wide range of problems: linear, nonlinear, discontinuous, discrete, etc.

2.2 Fuzzy Controllers

Fuzzy theory extends from the inability to describe some physical phenomena with the exact mathematical models dictated by more conventional, boolean models. Fuzziness describes event ambiguity. It measures the degree to which an event occurs, not whether it occurs[5]. The fact that fuzziness is lacking in precision has led to its dismissal by some researchers. Others, however, see fuzzy theory as a powerful tool in the exploration of complex problems because of its ability to determine outputs for a given set of inputs without using a mathematical model. As Jain noted[6], the basic motivation behind fuzzy set theory was the fact that the conventional methods had become so complex that researchers trying to apply

them had to make a choice between a complex system and a complex tool.

Fuzzy theory owes a great deal to human language. As explained by Leung[7], daily languages cannot be precisely characterized on either the syntactic or semantic level. When we speak of temperature in terms such as "hot" or "cold" instead of in physical units such as degrees Fahrenheit or Celsius, we can see language becomes a fuzzy variable whose spatial denotation is imprecise. In this sense, fuzzy theory becomes easily understood because it can be made to resemble a high level language instead of a mathematical language. As an example, consider the fuzzy variable TEMPERATURE. The fuzzy set describing TEMPERATURE can be categorized as five fuzzy-set values {very cold (VC), cold (C), medium (M), hot (H), very hot(VH)}. Figure 1 shows one possible set of the membership functions of the fuzzy-set values VC, C, M, H, and VH for the range of TEMPERATURE 0°-130° F. Note that every value of temperature has a membership in every fuzzy-value set although in most cases this membership is 0.

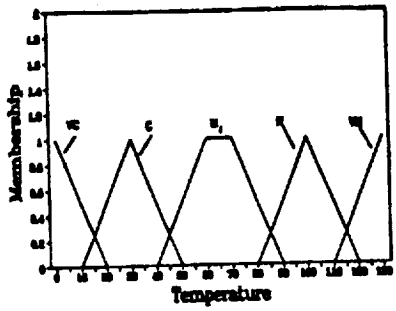


Figure 1 Fuzzy-Set Variables for the Fuzzy Variable TEMPERATURE

Also, some values of TEMPERATURE overlap into two fuzzy-value sets. For example a temperature of 47° has membership in both "cold" and "medium" although the membership in "medium" is larger than the membership in "cold." This example shows how membership functions play the role of discretizing the linguistic terminology to values a computer can use. Of course in most respects these membership functions are subjective in nature. What determines the ranges for these fuzzy-set values or the shape of these membership functions? In most cases, membership functions are designed by experts with a knowledge of the system being analyzed. However, human experts cannot be expected to provide optimal membership functions for a given system. Often, these functions are modified iteratively while trying to obtain optimality.

How are these membership functions used in fuzzy controllers? In its simplest form a fuzzy logic controller is simply a set of rules describing a set of actions to be taken for a given set of inputs. It is easiest to think of these rules as if-then statements of the form *IF*{set of inputs} *THEN*{outputs}. For the example above, a fuzzy controller can be used for a thermostat. One rule might be *IF*{very cold} *THEN*{turn furnace on for x minutes}. Another may be *IF*{hot} *THEN*{turn air conditioner on for y minutes}. Since "very cold" applies to a range of temperatures which also may belong to another fuzzy-set variable (i.e. "cold") which has rules of its own, the output which results from "defuzzification" of the application of these rules must take into account how much each rule applies before determining how much output must be applied. Usually a centroid method is used to account for the influence of each rule on the output.

2.3 Applicability of GA to Fuzzy Controllers

The application of genetic algorithms to fuzzy logic controllers holds a great deal of promise. Previous work has been done mainly in two areas: learning the fuzzy rules used in a controller and learning membership functions. These two areas are the most time consuming of fuzzy controller design and are for the most part done by trial-and-error. This methodology is lacking in two main respects: it may take too much time to get a satisfactory rule set or set of membership functions and there is little chance that these sets will be optimal. Genetic algorithms has the ability to improve both of these shortcomings. GA's robustness enables it to cover a complex search space in a relatively short period of time while ensuring an optimal or near-optimal solution. Because of this capability, GA is a natural match for fuzzy controllers.

Thrift's paper[1] examined the feasibility of using GA to find fuzzy rules. In this paper, fuzzy control synthesis was done in decision table form. The problem examined was centering a cart of mass m on a one dimensional track. The objective is to move the cart from a given initial position and velocity to zero position and velocity in minimum time. This is done through the application of a force F from the controller. For 100 runs with random starting points, the average number of time steps for the hand-designed fuzzy controller to bring the cart to zero position and velocity was 164. In

comparison, a GA designed controller using the same starting points had an average of 143 time steps. As Thrift noted, while the GA based fuzzy rules performed reasonably well, work could be done to further improve its performance, such as letting GA determine the endpoints of the membership functions.

Karr[8] examined using GA to find high-performance membership functions for a controller for the a pole-cart system. The task for the controller is as follows:

> A wheeled cart has a rigid pole hinged to its top. The cart is free to move right or left along a straight bounded track and the pole is free to move within the vertical plane parallel to the track. The cart is to be kept within the predefined limits of the track and the pole should be prevented from falling beyond a predefined vertical angle by applying a force of fixed magnitude to the left or right of the base of the cart.

The objective is to bring the cart to rest at the center of the track with the pole balanced, much the same as in Thrift's paper. Also, he examined the use of micro-GA, a small population GA developed by Krishnakumar[9], to determine an adaptive real-time controller for the same problem where system parameters may be time varying. In determining the membership functions, GA was used to determine the anchor points for each of the linguistic variables used. For the non-adaptive problem, the GA designed fuzzy logic controller consistently outperformed the hand-designed controller. For the adaptive controller, the performance was even better: the non-adaptive author designed controller always became unstable, while the non-adaptive GA controller and the micro-GA designed adaptive controller were always able to complete the task. The difference between the two GA designed controllers was in their convergence times; the micro-GA controller consistently balanced the system faster than the non-adaptive GA controller.

Previous work done with optimizing fuzzy controllers has dealt with optimizing membership functions or rule sets. For example, Mamdani and Procyk[10] iteratively designed membership functions, Thrift[1] used GA to design rule sets, and Karr used GA to design membership functions[11]. These methodologies have a major limitation; how can an optimal design be obtained when one of the two main components is designed in a non-optimal method. Logically, to obtain an optimal rule set and set of membership functions, the two must be designed together so the links between them can be fully exploited. By using GA to design both simultaneously, the two elements of fuzzy controllers can be fully integrated to deliver a more finely tuned, high performance controller.

3. PROBLEM DESCRIPTION AND METHODOLOGY

3.1 Cart-Centering Problem

A common problem used in literature is the centering of a cart of mass m, on a one-dimensional track. The input variables for this problem are the cart's location on the track, x, and the cart's velocity, v. The objective was to find a controller which could provide a force F which would bring the cart to x=0 and v=0 from an arbitrary initial condition (x_0 and v_0) in minimum time. The equations of motion for the cart are:

$$x(t+\tau) = x(t) + \tau\, v(t)$$
$$v(t+\tau) = v(t) + \tau \frac{F(t)}{m} \qquad (2)$$

where τ is the time step. The values for the constants and the range of values of the variables are given in Table I.

Three controllers were developed for the cart-centering problem. They will be referred to by the number of fuzzy sets that partition the x-location, velocity, and output. For example, the controller which had the velocity divided into 5 fuzzy sets, the x-location divided into 5 fuzzy sets, and the output divided into 5 fuzzy sets was called the 555 controller. While GA was allowed to determine the location of the triangle bases for the input variables, the output fuzzy set locations were fixed. The different output fuzzy membership functions are shown in Figures 2-4.

Table I Constants and Ranges for Cart Problem

Variable or Constant	Value
m	20 kg
τ	.02 sec
x	-2 to +2 m
v	-2 to +2 m/s
F	-150 to +150 N

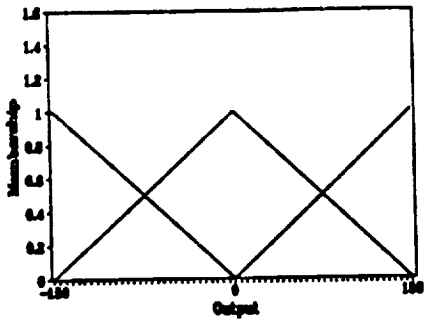


Figure 2 Output Divided Into 3 Fuzzy Sets

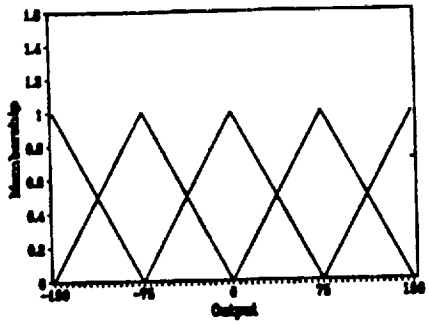


Figure 3 Output Divided Into 5 Fuzzy Sets

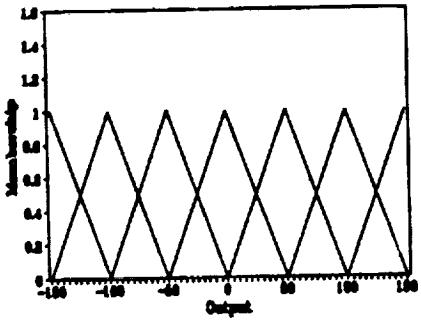


Figure 4 Output Divided Into 7 Fuzzy Sets

3.2 Software and Modifications

The basis for the software used in this paper is the Simple Genetic Algorithm (SGA) program developed by Goldberg[3]. The program was run on a 25 MHz 80386 computer with a 80387 math coprocessor. The SGA program allows the user to define the values for population size, maximum number of generations, probability of crossover, and probability of mutation. The values used for these parameters are given in Table II. Since run time became extremely long, population size was kept at a relatively small number, 100. This did lead to difficulties when larger string sizes were used and will be discussed in the following chapter.

3.3 Modification of String Structure

The Simple Genetic Algorithm uses binary strings to encode the parameters which are to be optimized. While this method could also be used in the determination of the fuzzy controller design, a more representative method was chosen.

First, the number of alleles was determined from the size of the rule set plus the number of fuzzy sets used to partition the

Table II Parameters Used in SGA Program

Parameter	Value
Population Size	100
Max. No. of Generations	100
P_c	0.7
P_m	0.03

spaces of the input variables. For the cart centering problem, the shapes of the triangles which formed the output space were fixed, while the input variables, x-location and velocity, were each partitioned using five triangles: negative medium (NM), negative small (NS), zero (ZE), positive small (PS), and positive medium (PM). The rule set, then, contained twenty-five (5 x 5) rules to account for every possible combination of input fuzzy sets. The rules are of the form, *IF*(x is {NM, NS, ZE, PS, or PM}) and (v is {NM, NS, ZE, PS, or PM}) *THEN* {*output*}, where *output* is one of the fuzzy sets used to partition the output space. The two input spaces use a total of ten triangles, so the string to represent a given rule set and membership function combination would have thirty-five alleles (25 + 10). Note that the term alleles is used instead of bits, because the value of each allele contains either the output fuzzy set to be used (for the first twenty-five alleles where NM=1, NS=2, etc.) or the value which will be converted to the length of the base of the triangles which make up the input spaces (the last ten alleles). The calculation of the triangle bases from the allele values (1-5) were done as follows:

1. Subtract 1 from the allele value (making the range now 0-4).
2. Subtract this value from 1 (which is the distance between the peaks of each triangle).
3. Double this value and divide by 10, giving the base length for each particular triangle. This value can be anywhere from 1.2m to 2.0m.

Thus we are able to incorporate the two main ingredients of a fuzzy controller, the rule set and the membership functions, into a single string which GA will seek to optimize. This is shown in the following example.

```
String: 14321524321245143122113454525234124
String: |1432152432124514312211345|  45252    |  34124    |
        |         rule set         | x-location | velocity  |
        |                          | locations  | locations |
```

velocity

	x-location: NM	NS	ZE	PS	PM
NM	1	4	3	2	1
NS	5	2	4	3	2
ZE	1	2	4	5	1
PS	4	3	1	2	2
PM	1	1	3	4	5

Rule Set

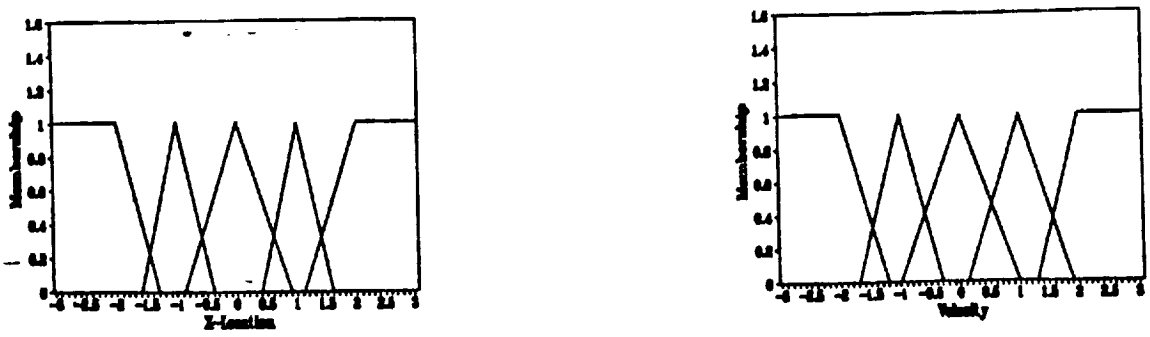


Figure 5 Example of String-Fuzzy Controller Conversion

4. SIMULATION RESULTS

4.1 Initial Conditions

To find a satisfactory controller, the controller must be able to operate over the entire range of the input spaces. For GA to properly design fuzzy controllers, this fact must be integrated into the function evaluation. This was done by using multiple initial conditions in the evaluation of each member of the population. If a single initial condition were used, for example $x_0 = 0.7$m and $v_0 = -0.5$m/s, then GA would find a controller which would work well around that particular point but may fail elsewhere. This makes the choice of initial conditions an important consideration. The points must be chosen to sufficiently cover the input spaces, but at the same time, the more initial conditions used, the more time the program takes to run. These initial conditions are listed in Table III.

4.2 Fitness Function

The fitness function proved to be the most challenging aspect of applying GA to fuzzy controller design. As stated earlier, the process finally was divided into two stages, an evolution stage and a refinement stage. In the evolution stage, GA was used to find satisfactory controllers, while in the refinement stage, GA used the previously developed controllers and attempted to minimize the amount of time needed to bring both x-location and velocity to zero.

Table III Initial Conditions

Controller	Initial Conditions (x_0, v_0)
333	(-2,-2) (-2,2) (0,0) (2,-2) (2,2)
5557	(-2,-2) (-2,2) (-1,-1) (-1,1) (0,0) (1,-1) (1,1) (2,-2) (2,2)
777	(-2,-2) (-2,0) (-2,2) (-4/3,-4/3) (-4/3,4/3) (-2/3,-2/3) (-2/3,2/3) (0,0) (2/3,-2/3) (2/3,2/3) (4/3,-4/3) (4/3,4/3) (2,-2) (2,0) (2,2)

For the first stage, which lasted through generation 30, the fitness function rewarded a member of the population according to how well it came to the tolerance value, ± 0.5 for both x-location and velocity. If the controller succeeded in bringing x and v within the tolerance, it was given a fitness relative to the time it took. If the controller "timed out," it was either slightly punished with a negative fitness or slightly rewarded depending on x-location and velocity. If the controller diverged, the fitness was given a larger negative value. The first fitness function is shown below.

```
if (|x| < 0.5) and (|velocity| < 0.5) then
    fitness = 8 * 175 / time
else if (time = 175) then
    if (|x| < 1.0) and (|velocity| < 1.0) then
        fitness = 3.5 / sqrt( x² + velocity² )
    else
        fitness = -1
else
    fitness = -7
```

The total fitness was then given by the sum of the fitnesses determined for each initial condition. Through the use of this reinforcement/reward scheme, GA was able to develop controllers which could solve all the initial conditions.

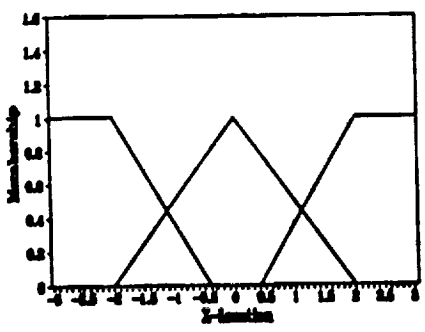
The second stage, from generation 31 to generation 100, was based almost completely on time. If the controller reached the tolerance values it was rewarded according to how short a time it took. If the controller "timed out," it was punished according to how much it missed the tolerance values, and if the controller diverged, it was given a very large negative fitness which would probably ensure its failure to continue on to the next generation. This fitness function was given as:

```
if (|x| < 0.5) and (|velocity| < 0.5) then
    fitness = 3 * (175 - time)
else if (time = 175) then
    fitness = -42 * sqrt( x² + velocity² )
else
    fitness = -300
```

4.3 333 Controller

The 333 controller was the simplest controller to design. It consisted of only 9 rules, and the number of triangle base locations to be determined was 6, yielding a total string length of 15. The controller determined by GA is shown in Figure 6.

velocity \ x	N	ZE	P
N	3	3	1
ZE	3	2	1
P	3	1	1



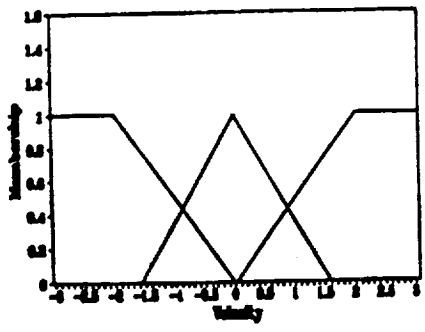
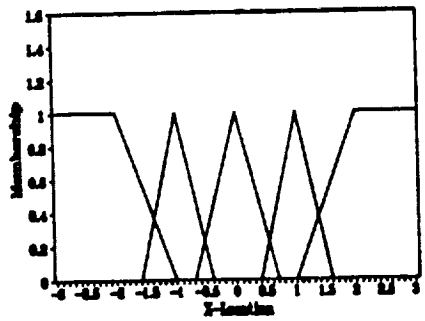


Figure 6 333 Controller

4.4 555 Controller

The 555 controller used 25 rules and needed 10 alleles to determine the location of the bases of the fuzzy sets covering the input spaces. With a total string length of 35, the first indication that better performance could be obtained with larger population sizes became apparent. Considering that each allele could have a value between 1 and 5, this meant that if a binary string had been used, 3 bits would be necessary to represent the same information. This would yield a string length of 105. A population of 100 cannot begin with enough diversity to ensure that the search space will be sufficiently covered to enable GA to find the optimal solution. Even with a relatively small population size, the run time took between two and one-half to three hours. However, the performance of the controller did indicate that GA was finding a near-optimal controller. Figure 7 shows the resultant best controller determined by GA.

velocity \ x	NM	NS	ZE	PS	PM
NM	5	4	4	4	2
NS	5	5	5	1	1
ZE	5	5	3	1	1
PS	5	5	1	1	1
PM	4	2	2	2	1



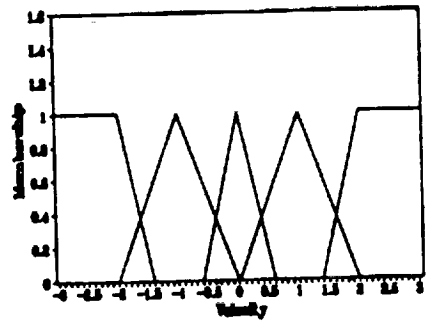


Figure 7 555 Controller

4.5 777 Controller

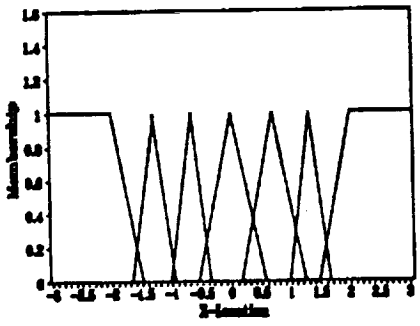
The final controller designed was the 777 controller. This controller took the most computer time to design because of the long string length (63 alleles) and the large number of initial conditions (17). Run times took between seven and eight hours. The best performing controller designed by GA is shown in Figure 8.

4.6 Comparison of Cart Controllers

Once GA had been used to determine the different controllers, a study was made to determine their performance and stability. Because there is no accepted method to determine stability for fuzzy controllers as there is for classical and state-space controllers, a "brute force" method was adopted. To examine the controllers, the input space of each variable was divided into 40 points. Then each point was examined, one by one, to determine if the controller diverged anywhere in the input space. For 40 points in the x-location space and 40 points in the velocity space, this yields a total of 1600 points. While the ability of a controller to satisfy all these points does not necessarily guarantee its stability (since it only takes one point to make a controller unstable), this did ensure some measure of confidence in the procedure. While examining each point, it was a simple task to also count the number of time steps used to bring all these points within the tolerance values, and these numbers are given in Table IV. Also shown in Table IV is an additional 555 controller designed with the

membership functions fixed. GA was used to design only a rule set, with the membership function being done by hand. This controller, shown in Figure 9, was created for comparison purposes to illustrate the importance of membership function selection. As Table IV shows all the controllers were able to successfully bring the system within the tolerance values.

velocity \ x	NL	NM	NS	ZE	PS	PM	PL
NL	7	5	3	7	7	1	2
NM	7	6	6	7	3	2	1
NS	7	6	7	7	1	1	1
ZE	7	5	7	5	1	2	2
PS	6	6	6	1	1	4	1
PM	6	6	6	1	1	6	1
PL	6	5	1	1	5	2	1



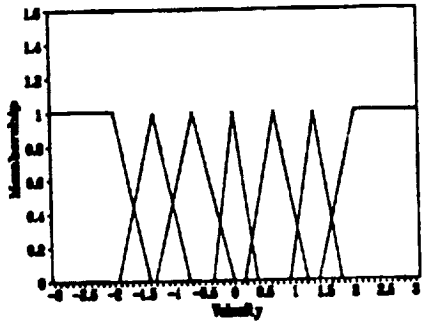


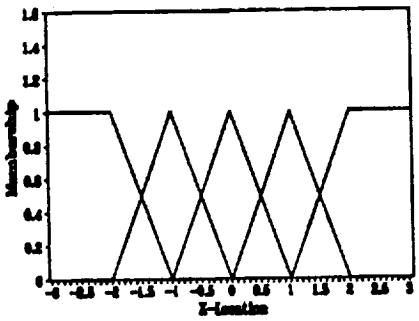
Figure 8 777 Controller

Table IV Comparison of Cart Controllers

	333	555	777	555 fixed
No. of Initial Conditions	1600	1600	1600	—
Time Steps Required	82380	68208	78958	90156
Avg. No. of Time Steps	51.49	42.63	49.35	56.35
% Difference w/555 Controller	20.8%	—	14.8%	32.2%
No. of Failures	0	0	0	0

Table IV shows that the best performance, on average, came from the 555 controller. The 333 controller, while being the simplest, did not have the flexibility to produce fast response times. On the other hand the 777 controller had too much flexibility and became bogged down in the number of rule evaluations required for each force calculation. Finally, note that while the 555 controller with the fixed membership function was able to bring the cart to equilibrium for all points, its performance was clearly inferior, needing almost 1/3 longer than the GA designed rule set and membership function combination, indicating the importance of proper membership function design.

velocity \ x	NM	NS	ZE	PS	PM
NM	5	5	5	3	2
NS	5	5	5	1	1
ZE	5	5	3	1	1
PS	5	5	1	1	1
PM	5	1	4	2	1



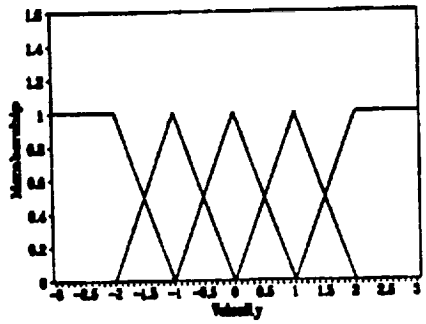


Figure 9 555 Controller w/ Fixed Membership Functions

5. CONCLUSION

This paper clearly shows the potential for using genetic algorithms to solve optimization problems. The ability of fuzzy logic controllers to provide control where more conventional methods become too complex has also been shown by researchers. This work has shown these two, fairly new, methods can be used to together to form controllers without the previously needed human expert. This methodology allows the complete design of both major components of fuzzy controllers, the rule sets and membership functions, leading to high performing controllers which are completely computer designed. We have shown three different controllers for the cart problem, each of which was able to bring the cart to equilibrium over the entire ranges of the input spaces. While these results are encouraging, more work must be done on refining the process. First, more powerful and faster computers will allow the use of larger population sizes, and, therefore, greater diversity. Work will be done to examine the development of a robust controller, where the parameters in the equations of motion are nc longer fixed to a specific value, but can instead be a range of values. Also, the need to ensure the performance of the controller when faults occur in the rule set should be investigated to see if this would alter the configuration of the rule set. Finally, controllers for other problems should be developed to show the effectiveness of this method.

6. ACKNOWLEDGMENTS

This work is supported by grants from Honeywell Inc. under grant number 48057 and the NASA Center for Research Excellence at N.C. A&T State University under grant number NAGW-2924. The authors wish to thank them for their

financial support which made this work possible.

7. REFERENCES

1. Thrift, P., "Fuzzy Logic Synthesis with Genetic Algorithms," Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991, pp.509-513.
2. Holland, J.H., Adaptation in Natural and Artificial Systems, The University of Michigan, Ann Arbor, MI, 1975.
3. Goldberg, D.E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, MA, 1989.
4. Holland, J.H., "Schemata and intrinsically Parallel Adaptation," Proceedings of the NSF Workshop on Learning System Theory and Application, Gainsville, FL, University of Florida Press, 1975, pp.43-46.
5. Kosko, B., Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence, Prentice Hall, Englewood Cliffs, NJ, 1992.
6. Jain, R., "Fuzzyism and Real World Problems," Fuzzy Sets: Theory and Applications to Policy Analysis and Information Systems, Wang, P.P, and Chang, S.K. (Eds.), Plenum Press, New York, NY, 1980.
7. Leung, Y., Spatial Analysis and Planning Under Imprecision, Elsevier Science Publishers B.V., New York, NY, 1988.
8. Karr, C.L., "Design of an Adaptive Fuzzy Logic Controller Using a Genetic Algorithm," Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991, pp.450-457.
9. Krishnakumar, K., "Microgenetic Algorithms for Stationary and Nonstationary Function Optimization," SPIE Proceedings on Intelligent Control and Adaptive Systems, Vol. 1196, pp.289-296, November, 1989.
10. Procyk, T.J., and Mamdani, E.H., "A Linguistic Self-Organizing Process Controller," Automatica, Vol. 15, No.1, pp.15-30, 1979.
11. Karr, C., "Genetic Algorithms for Fuzzy Controllers,", AI Expert, February 1991, PP.26-33.