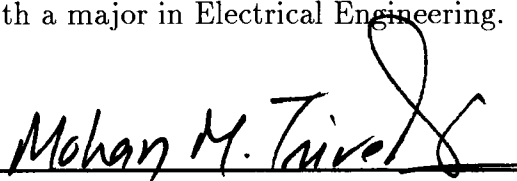


To the Graduate Council:

I am submitting herewith a thesis written by Kim C. Ng entitled "Intelligent Robotic Learning Systems: A Fuzzy-Neural Architecture and Real-Time Implementations." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.



Prof. Mohan M. Trivedi, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:



Associate Vice Chancellor
and Dean of The Graduate School

**INTELLIGENT ROBOTIC LEARNING
SYSTEMS: A FUZZY-NEURAL
ARCHITECTURE AND REAL-TIME
IMPLEMENTATIONS**

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Kim C. Ng

December 1995

Dedicated to my
loving parents

Vision

A vision without a task is but a dream.

A task without a vision is a drudgery;

A vision with a task is the hope of the world.

Acknowledgments

I especially would like to thank my respected major advisor, Dr. Mohan Trivedi, for his patient instruction and guidance throughout these years. His constant encouragement energizes me to aim higher and better. Not only does he care for my academic learning process, but he also provides warm concern for my everyday life. Further, I would like to thank my committee members, Dr. Igor Alexeff and Dr. Marshall Pace, for their input and patience in preparing this thesis. I would also like to thank every wonderful team member of the Computer Vision and Robotics Research Laboratory at the University of Tennessee, for their constructive criticism and timely help.

Abstract

The main focus of this thesis is on the development of intelligent robotic systems which are capable of learning a wide array of intelligent behaviors. The framework utilized in this development is that of nonclassical control architectures to realize real-time and primarily reactive system performance.

A Neural integrated Fuzzy conTroller (NiF-T), which integrates the fuzzy logic representation of human knowledge with the learning capability of neural networks, is developed for nonlinear dynamic control problems. The NiF-T architecture comprises three distinct parts: (1) Fuzzy logic Membership Functions (FMF), (2) Rule Neural Network (RNN), and (3) Output-Refinement Neural Network (ORNN). FMF are utilized to fuzzify input parameters. RNN interpolates the fuzzy rule set; after defuzzification, the output is used to train ORNN. The weights of the ORNN can be adjusted on-line to fine-tune the controller.

NiF-T can be applied for a wide range of sensor-driven robotics applications, which are characterized by high noise levels and nonlinear behavior, and where system models are unavailable or are unreliable. In this thesis, real-time implementations of Ball Balancing Beam (BBB), autonomous mobile robot navigation, and mobile robot convoying behavior utilizing the NiF-T are realized. The NiF-T successfully balances balls with one-third of the required 27 rules. With learning capability, these systems approach their goals more frequently, in general. The robot, SMAR-T, successfully and reliably hugs the wall, locks onto the hall center, and convoys a leader at any specified distance.

TABLE OF CONTENTS

CHAPTER	PAGE
1. Introduction: Approaches for Intelligent System Development	1
1.1 Evolution of Artificial Intelligence (AI)	1
1.2 Fuzzy Logic	2
1.3 Artificial Neural Networks	4
1.4 Statement of the Problem	7
1.5 Outline of the Thesis	7
2. Neural Integrated Fuzzy Controller (NiF-T) for Integrated, In-	
telligent Robotic Systems with Learning Abilities	9
2.1 Problem Statement and Specifications	9
2.2 Review of Other Approaches	12
2.3 Architecture	14
2.4 Training and Testing Procedures	14
2.5 Learning Algorithm	18
2.6 Design of the FMF: Fuzzy Membership Functions	19
2.7 Design of the RNN: Rule Neural Network	19
2.7.1 General Approach for Rule Derivation	20
2.7.2 Training Issues	21
2.8 Design of the ORNN: Output-Refinement Neural Network	22

CHAPTER	PAGE
2.9 Summary	24
3. Real-Time Control of Ball Balancing Beam (BBB)	26
3.1 Problem Statement and Specifications	26
3.2 Review of Other Approaches	28
3.3 The Control Scheme	29
3.3.1 Membership Functions Derivation	30
3.3.2 Rules Derivation	32
3.3.3 Objective Function	33
3.4 Experimental Testbed Development	33
3.4.1 Design of the Sensory System	37
3.4.2 Design of the Activator	39
3.5 Experimental Validation	40
3.6 Summary	45
4. Autonomous Navigation of a Mobile Robot Using Sonar Sensor	46
4.1 Problem Statement and Specifications	46
4.2 Review of Other Approaches	47
4.3 Experimental Testbed	50
4.4 Wall Following Behavior	51
4.4.1 The Control Scheme	51
4.4.2 Experimental Validation	54
4.5 Hall Centering Behavior	58

CHAPTER	PAGE
4.5.1 The Control Scheme	58
4.5.2 Experimental Validation	62
4.6 Summary	64
5. Controlling Multiple Mobile Robots Moving in Formation . .	65
5.1 Problem Statement and Specifications	65
5.2 Review of Other Approaches	66
5.3 Experimental Testbed	67
5.4 The Control Scheme	68
5.5 Experimental Validation	72
5.6 Summary	76
6. Summary and Conclusions	77
6.1 Summary	77
6.2 Unique Contributions	80
6.3 Future Direction	82
BIBLIOGRAPHY	83
APPENDICES	90
A. Fuzzy Logic Control Process	91
A.1 A BBB Example	92
B. Experimental Validation of the BBB FLC Behavior	96

CHAPTER	PAGE
C. Back Propagation Algorithm	100
C.1 Training Procedure	100
C.2 Mathematical Description	102
VITA	104

LIST OF TABLES

TABLE		PAGE
3.1	<i>These nine parent rules, one-third of the conventional FLC rule set, are exploited to generate the rest of the children rules. The first rule is the objective most stable condition, while the rest of them are intended to handle the most critical situations.</i>	34
3.2	<i>RNN is trained to remember these nine rules. Both inputs and outputs are fuzzy vectors. When input membership values fall between 0 and 1, RNN will interpolate the outputs after it learns.</i>	35
4.1	<i>The wall following parent rule set.</i>	53
4.2	<i>The nine hall centering parent rules.</i>	61
5.1	<i>The parent rules of robot convoy.</i>	71

LIST OF FIGURES

FIGURE	PAGE
2.1 <i>The schematic of the NiF-T. FMF are utilized to fuzzify input parameters. RNN interpolates the fuzzy rule set to provide logical reasoning. During off-line training, the RNN's outputs are defuzzified to train ORNN. The weights of ORNN can be adjusted to refine the controller throughout on-line operation.</i>	15
2.2 <i>The control surfaces of a fuzzy controller with missing or bad rules. (a) The objective first rule is missing. (b) There are many extremely incorrect rules. We plot the control output against other two parameters. In general, good control surfaces are monotonic without any obvious spikes or fold-around.</i>	23
3.1 <i>The schematic of the full hardware implementation of BBB. . . .</i>	27
3.2 <i>The control process block diagram of BBB</i>	30
3.3 <i>Input parameter membership functions. Observation of physical constraints helps to construct these membership functions. The symbol representations are listed as zero(z), positive(p), and negative(n).</i>	31

FIGURE	PAGE
3.4 Comparison of control surfaces derived using only Fuzzy Logic (a–c) and Fuzzy-Neural approach (d–f). The fixed parameter is set at negative extreme. All of these surfaces are monotonic without any obvious spikes and fold-around.	36
3.5 The parts list.	37
3.6 Noisy voltage signal.	38
3.7 The BBB experimental testbed.	41
3.8 A large ball was controlled on the beam continuously from (a) to (e).	41
3.9 The system performance. Results of (a)–(d) are directly output from the ORNN without further training. The balls were balanced off the center. After learning continued, the controller was able to shift the balancing back to around the center in (e) to (h).	42
3.10 Relationship of position and theta. The linear relationship indicates that changes of theta directly set the ball position. Understanding the parameter’s relationship will help further improve the controller.	43
3.11 Control output versus system parameters. The controller is trying to keep up with the changes in the ball’s speed and direction. The results validate the appropriateness of the rule set used.	44
4.1 SMAR-T: Small Mobile Autonomous Robotic Testbed.	51
4.2 The robot scanning regions and the input denotations for following wall.	52
4.3 The membership functions for wall following behavior.	53

FIGURE	PAGE
4.4 <i>SMAR-T followed the left wall in a hallway.</i>	55
4.5 <i>The experimental results of wall following. The robot speed is set at 50Hz in (a) and (b), at 100Hz in (c) and (d).</i>	56
4.6 <i>The experimental results of wall following. The robot speed is set at 200Hz in (a) and (b), at 300Hz in (c) and (d).</i>	57
4.7 <i>The hall centering input parameter denotations.</i>	59
4.8 <i>Hall centering membership functions.</i>	60
4.9 <i>SMAR-T moved to the hall center.</i>	62
4.10 <i>The experimental results of hall centering.</i>	63
5.1 <i>Experimental Laboratory Vehicle Integrated with Sensors (ELVIS) is a leader in the robot convoy.</i>	68
5.2 <i>The robot convoy schematic.</i>	70
5.3 <i>The membership functions of robot convoy.</i>	70
5.4 <i>SMAR-T convoyed ELVIS in a hallway.</i>	73
5.5 <i>The leader was stationary. The follower started from different ini- tial positions, and it was able to stop at the specified "safe" distance in a very short time.</i>	74
5.6 <i>The follower convoyed the moving leader. The leader kept changing its speed and direction. The follower was able to cope with the change and stop close to the specified safe distance when the leader stopped.</i>	75

FIGURE	PAGE
A.1 <i>The schematic of the fuzzy logic controller. Input parameters are fuzzified to pass through the fuzzy control rules logic unit. After reasoning, control output is defuzzified to send to process. New parameter values are obtained and sent through the same process again.</i>	91
A.2 <i>(a) Membership functions of the input parameters. (b) Output membership function. Observation of physical constraints is used to construct these membership functions. The symbol representations are listed as zero(z), positive(p), negative(n), small($\pm s$), medium($\pm m$), and large($\pm l$).</i>	92
A.3 <i>The procedure of defuzzification. (a) Fired label clipped at its corresponding level. (b) Find their trapezoidal areas and compute the horizontal coordinate of the union area centroid.</i>	95

B.1	<i>(a) A small ball was oscillating back and forth symmetrically around the center. (b) The results of the medium ball are close to the small ball's. (c) Some improvement is shown for the large ball. The system happened to oscillate less after a certain time, and then it oscillated again. It is due to the imperfection of the hardware. The ball with less mass is easy to accelerate toward the down-inclined side with a small initial force. (d) Both small and medium balls were rolling together around the center after a few hundred iterations. (e) Both medium and large balls became completely steady after around 160 iterations. The system was disturbed again thereafter many times. It was still able to go stable after awhile. (f) Shows that the balls in (e) really stopped.</i>	97
C.1	<i>A neural network for the Back Propagation Algorithm notations. .</i>	101

CHAPTER 1

Introduction: Approaches for Intelligent System Development

1.1 Evolution of Artificial Intelligence (AI)

Artificial intelligence is “the study of ideas that enable computers to be intelligent” [Win84]. Professor Trivedi pointed out that AI involves two types of tasks: “Type 1 tasks deal with understanding intelligent activities performed by humans and then attempting to emulate them in machines; type 2 tasks deal with developing computers that can perform activities that are believed to require intelligence” [TG86]. Since the mid-1950s, after the invention of the computer, interest in building human-like machines has been growing. Traditional AI—such as expert systems, game-playing systems, and natural language processing—was based almost entirely on symbol manipulation and first-order logic [Win84]; in fact, Zadeh [Kos91] believed that “AI would have made much progress toward its goals if it had not committed itself so exclusively to symbol manipulation and first-order logic.” Expert systems which were originally developed to solve static problems [AP93] drawing an inference in forward/backward chaining are not suitable for dynamic control [CC94, YZ94]. Additionally, a symbol cannot be

mathematically differentiated, so this prevents us from directly applying the tools of numerical mathematics and from directly implementing AI systems in large-scale integrated circuits although the symbolic framework enables us to represent structured knowledge as rules [Kos91]. Consequently, traditional AI has found its limitation on many real-world problems, including the category of uncertainty and imprecision.

Today, the main domain of AI includes fuzzy logic, neural network theory, probabilistic reasoning, belief networks, and the theory of chaotic systems. The emergence of fuzzy logic and neural networks has helped solve a wide variety of problems. Fuzzy logic concentrates on imprecision and approximate reasoning; neural networks are capable of learning and curve fitting. Interpolation plays a major role in both of them; they are complementary rather than competitive [KL93]. Since fuzzy logic and neural networks have their own strengths and weaknesses, fusion and combination of these two techniques are the rapidly growing interests of recent research.

1.2 Fuzzy Logic

When Professor Zadeh presented his first paper on fuzzy set in 1965, his primary intention was to apply the theory in those fields where conventional mathematical techniques lack effectiveness [KL93]. He addressed the fact that most real-world problems fall in the category of uncertainty and imprecision. Take, for

example, a teenager learning to drive a car: once he has acquired the skill, he can get into any car, regardless of its size and weight, and apply the same techniques. Balancing a ball on a beam is another example: we learn to tilt the beam in order to compensate for the ball's movement. Afterward, balancing with a different beam and ball is nothing more than applying similar learned knowledge from the past experience. Certainly, however no one learns these behaviors by thinking in mathematical equations with precise, measured numerical values, such as turning the car 5 degrees with 50.3 degrees per second at that 0.3-mile-radius corner. Thus, it is the imprecision of the goal that makes it possible for humans to solve everyday problems quickly at little or no cost.

Mamdani and Assilian in 1975 were the first to apply fuzzy set theory to a working control problem. Along with the contributions of Sugeno [Sug85], numerous products which use fuzzy logic control are now on the market. Fuzzy logic gives us an alternative way to solve control problems. Both expert systems and fuzzy systems can represent any continuous function to any desired level of accuracy [Kos91]. Fuzzy logic provides a means of linking symbolic processing and numerical computation. The fuzzy model uses a rule base of proposition of the form **IF A THEN B**. Unlike expert systems, fuzzy logic includes not only rules to direct the decision-making process but also membership functions to convert linguistic variables into the precise numerical values that the computer requires for control process. It has interactions among antecedent-consequence rules to influence the control action. There are two common paradigms to translate these

kinds of propositions by the theory of approximate reasoning. One approach, introduced by Mamdani and Assilian, has been the prevalent approach to the fuzzy logic control applications. It interprets a rule as a conjunction of the antecedent and consequence and uses disjunction to aggregate the output of the individual rules. Another approach is the TSK method, which simplifies fuzzy reasoning by consequents that are singletons rather than fuzzy sets [YZ94]. Refer to Appendix A for a process of fuzzy logic controller.

For any given design, it usually has fewer parameters to consider with the use of fuzzy logic than the conventional control approaches. The membership functions provide continuous representation. Since fuzzy logic controller utilizes if-then rules, no extra technique consideration is required for either a linear or a nonlinear system. Fuzzy logic emphasizes imprecision; it is therefore expected to have more tolerance of noisy signals. However, a designer has to be fairly familiar with a system to derive all the rules and membership functions. Rule derivation is time-consuming, especially when the number of input parameters increases. Particularly, its inability to learn makes the rule and membership function tuning process a drudgery.

1.3 Artificial Neural Networks

Artificial neural network is defined as “a data processing system consisting of a large number of simple highly-interconnected processing elements in an ar-

chitecture inspired by the structure of the cerebral cortex of the brain” [Uhr95]. A neural network usually has at least three layers: input, hidden, and output. Every layer has a certain number of nodes, which cross-connected to another layer’s nodes with variable weight vectors. Inside each neuron node, a nonlinear transfer function is used to keep the output within a reasonable range and to emulate nonlinear phenomena. There are two kinds of networks: auto-associative and hetero-associative. The auto-associative network’s input and output layers have the same number of nodes, while those of the hetero-associative network do not. These networks can be fully or partially connected between each layer in the feedforward or feedback manner. Artificial neural network performs only two functions: learning and recall. Supervised, reinforced, and unsupervised learning [Rei90, dar88, neu93] are three basic weight updated schemes. Among all the learning algorithms, the most popular one is the Back Propagation algorithm (see Appendix C) since it has a strong mathematical background. It may sometimes be trapped in the local minima, but most of the time it converges. Nevertheless, it may have relatively longer training time than other algorithms.

There are some special features of artificial neural networks:

- They contain a large number of very simple neurons, like processing elements.
- They learn by example. Learning time may be long; however, testing usually takes just one computer clock cycle.

- They constitute a distributed, associative memory.
 - Distributed: each processing element makes a decision based only on its own local situation. All of these local actions add up to a global solution.
 - Associative: patterns can be stored. Retrieving needs only a portion of it.
- They are highly parallel.
- They are fault tolerant.

Only neural network can learn any function. However, the network is like a black box which transforms inputs into outputs. How it encodes and recalls is a still unsolved issue. There is no natural inferential audit trail. Unlike expert system, the inferential paths of the network to reach a given output are not known. We do not know what the neural network encoded during training or what it will encode or forget in further training. A further problem is sample generation. Neural network requires numerical values as the input vectors; however, the original reliable data are difficult to obtain. Usually an expert feels most comfortable giving the original linguistic rules. As a result, the unstructured computational black box is left with the problem of not being able to incorporate the reliable human expert linguistic rules directly into the system.

1.4 Statement of the Problem

The main focus of this thesis is on the development of intelligent robotic systems which are capable of learning a wide array of intelligent behaviors. The framework utilized in this development is that of nonclassical control architectures to realize real-time and primarily reactive system performance.

As we have stated that fuzzy logic and neural network both have their own strengths and weaknesses, we would like to utilize the advantages of their strengths to overcome each of their weaknesses. The objective here is to develop a control scheme, which integrates the fuzzy logic representation of human knowledge with the learning capability of neural networks, for intelligent robotic systems. The realization of the control scheme is implemented on sensor-based robots which possess real-time response.

1.5 Outline of the Thesis

This thesis starts with the introduction of a new fuzzy-neural hybrid control architecture in Chapter 2. The training/testing procedures and related design issues are thoroughly discussed. Chapters 3, 4, and 5—the real-time implementations of the proposed control scheme—cover the methods of rule and membership function derivation, testbed developments, and experimental validations. Chapter 3 shows the design of a fairly difficult nonlinear dynamic control problem, Ball Balancing Beam. Single mobile robot autonomous navigation and multiple robot

convoying behaviors are realized in Chapters 4 and 5, respectively. Finally, in Chapter 6 summary, unique contribution and conclusions of this research project are discussed.

CHAPTER 2

Neural Integrated Fuzzy Controller (NiF-T) for Integrated, Intelligent Robotic Systems with Learning Abilities

2.1 Problem Statement and Specifications

Conventional methods for solving nonlinear control problems necessitate strong mathematical background to derive equations. However, many system behaviors may not be mathematically formulated. Additionally, experts may share their valuable experience only in terms of linguistic rules. Fuzzy logic controllers which have continuous functions mimic expert minds. Nevertheless, deriving and fine-tuning the entire rule set and membership functions are just too drudging. Neural controllers learn, yet discrete input representations may cause systems to be unstable. In addition, sufficient training patterns are usually difficult to obtain, and training time for the whole dynamic range is very long. Therefore, taking advantage of the best of fuzzy logic and neural networks—assimilating human expertise with continuous representation, fewer rules and mathematics, and learning capability—a control architecture blending neural networks and fuzzy logic is proposed for nonlinear control problems.

The main objective here is to develop robust and practical control approaches, using nonclassical means such as fuzzy logic and neural networks. This objective of neural integrated fuzzy controller design motivates us to develop a general architecture for integrated sensor-based robotic systems which have the following capabilities:

- Integrated sensing, control, and activator modules.
- Real-time performance requirement.
- Ability to successfully handle noisy sensory signals.
- Reactive controller design which effectively captures high-level, linguistically based human expertise in a set of fuzzy rules.
- Training the neural networks directly with fuzzy rules instead of numerical sample data. The linguistic expert rules are easier to obtain and more reliable.
- Learning capability to interpolate new sets of rules for robust and smooth operation of the system. New rules are derived of parent rules as specified by a human so that a minimum of rules are extracted from the expert.
- Learning capability to refine the performance of the integrated intelligent systems.
- General applicability of the architecture. That is, the same overall system

level architecture will be able to support realization of a wide range of robotic systems.

All of these features are verified by the later implementation chapters.

The basic idea of the integrated fuzzy-neural hybrid control approach is to exploit the learning of neural networks to interpolate missing fuzzy if-then rules with RNN so that a minimal number of rules are to be generated. Furthermore, ORNN helps to refine the defuzzification output. FMF, in turn, provide continuous representation for input parameters. As a result, slight variations in input parameters' constraints, such as slight alteration in beam length or ball mass, will not necessitate the system redesign. If, for example, balancing a ball at another beam position is desired, the control goal may be easily modified by shifting the position membership function or/and changing a few rules as long as adequacy of beam space is available to decelerate the ball from high-speed movement. The ability to embody high-level reasoning into the control system insures the ORNN immediate success toward the control goal with less on-line training time as well.

The big picture of the whole control process is this: FMF are utilized to fuzzify input parameters; RNN interpolates the fuzzy rule set; after defuzzification, RNN's outputs are used to train a smaller size of neural network, ORNN; and the weights of ORNN can be adjusted on-line to fine-tune the controller.

2.2 Review of Other Approaches

Many researchers are trying to correlate fuzzy logic and neural networks. Neural networks are believed to be able to help the development of fuzzy logic. Likewise, fuzzy logic can be utilized for the fusion of multiple neural networks [CK95]. In designing fuzzy logic expert systems, tremendous effort is required to obtain the rules [Yag94]. Therefore, researchers have started to focus on developing basic fuzzy neurons to obtain the membership functions and rules. Pedrycz has introduced two basic aggregation neurons named AND and OR, neurons based on logic-oriented processing mechanisms of fuzzy sets in [PR93, HP94]. Four other types of fuzzy neurons have been proposed and successfully implemented in pattern recognition problem [KC94]. These neurons can be aggregated into a single computational structure to generate a collection of if-then statements describing input/output relationships. With the use of neural methods, researchers hope that membership functions and rules may eventually be automatically generated and tuned. The fuzzy-neural networks in [HFU92] identified the fuzzy model of a nonlinear system automatically. FLIP-net could detect rule deficiencies and tune membership functions [FMMM95]. Identifying/tuning the fuzzy inference rules and the membership functions simultaneously are still problems to overcome. To overcome the problems of large-scale fuzzy knowledge bases using neural methods may, however, "change the logical implications of AND and OR operations in fuzzy logic during the learning process." This harms the fuzzy logic's two major

advantages, readability and maintainability [FMMM95].

When constructing a controller, two kinds of information are available: numerical information from measuring instruments and linguistic information from human experts. The proposed NiF-T does not utilize numerical data such as Higgins's [HG94] and Ishibuchi's methods [IT93] to generate rules, because converting an existing controller into a fuzzy controller is not desired. We try to maintain the originality of fuzzy logic so that its major advantages can be preserved. For the membership functions, a similar method to Ishibuchi's is used. Fuzzy numbers are propagated through neural networks unlike Horikawa [HFU92], NeuFuz4 [Tha93], and Iskarous [IK95], constructing membership functions using neurons. Bouslama [BI93] has used a three-layer neural network to apply for fuzzy control problem. However, instead of training the controller directly with rules, his training vectors were discrete data of a reference fuzzy logic controller. The proposed architecture is also different from fuzzy neurons and fuzzy min-max neural networks. A minimal number of rules are also used to train this NiF-T. In addition, tuning membership functions and rules are not of interest. Therefore, there is no physical output membership layer connected with variable weights for defuzzification such as Higgins and Berenji [BK92] have. The hidden and output layers of RNN perform the fuzzy logical operations, such as min-max operations, and interpolates the rest of the untrained rules. Basically, after RNN has learned, it acts something like a reference controller. A smaller network, ORNN, copies the behavior from RNN and tunes the controller performance using objective function.

2.3 Architecture

This NiF-T shown in Figure 2.1 includes three main parts: Fuzzy Membership Functions (FMF), Rule Neural Network (RNN), and Output-Refinement Neural Network (ORNN). Piecewise triangular functions or any other functions can be used for FMF. The number of membership functions equals the number of input parameters, and the number of labels of each membership function determines the number of RNN input nodes. For instance, three input parameters, each having three labels, contribute a total of nine RNN input nodes. The hidden layer of RNN can have any number of nodes. Usually, the number is set at least the same as the number of training pairs. The number of RNN output nodes is decided by the number of different output weight labels. ORNN's inputs are linked together with the RNN's outputs. The ORNN output layer always has a single neuron, and again the ORNN hidden layer can have arbitrarily the same number or fewer nodes than the ORNN's input layer.

2.4 Training and Testing Procedures

FMF fuzzify input parameters before they are fed into RNN. Both RNN and ORNN are multilayer feedforward neural networks using Back Propagation algorithm (BP) [Kni90, Kos91, Gal93], and their learning algorithms are derived from objective functions [IT93, CXL92]. (Refer to Appendix C on Back Propagation Algorithm). RNN maps fuzzy input vectors to fuzzy output vectors. Its objective

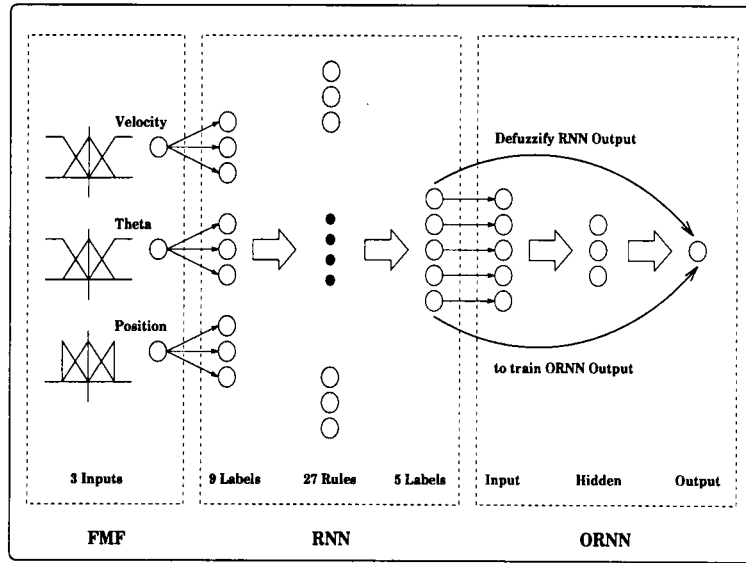


Figure 2.1: *The schematic of the NiF-T. FMF are utilized to fuzzify input parameters. RNN interpolates the fuzzy rule set to provide logical reasoning. During off-line training, the RNN's outputs are defuzzified to train ORNN. The weights of ORNN can be adjusted to refine the controller throughout on-line operation.*

function is defined by the fuzzy actual outputs and the corresponding fuzzy target outputs. After RNN is trained satisfactorily, training ORNN follows. There are two training phases for ORNN: the first is off-line, and the second is on-line. During the first phase, pairs of fuzzy input vectors, which are connected directly from the fuzzy output vectors of the RNN, and a nonfuzzy target output are the training patterns. The nonfuzzy target output is the defuzzification output of the RNN output vectors. ORNN's on-line training objective function is to minimize the controller errors based on the continuous feedback input parameters. During on-line operation, the weights of RNN are frozen. Only the ORNN can be selected

to be trained further.

The following is a brief procedure for using the controller:

Step 1. Determine the input parameters.

Step 2. Construct membership functions and derive fuzzy rules.

Step 3. Conduct preliminary training.

- (a) Train the RNN with the fuzzy rules (input/output pairs are specified).
- (b) Train the ORNN with the fuzzy rules through the RNN (the ORNN desired output patterns are the RNN defuzzification crisp output). Notice that any change in the input membership function will not require retraining the RNN and ORNN.

Step 4. Test the controller.

- (a) Without on-line training: pass the input parameters through FMF, RNN, and ORNN.
- (b) With on-line training: pass the input parameters through FMF, RNN, and ORNN; change weights of ORNN according to the objective function based on the feedback input parameters.

Step 5. If any rule is changed, repeat from Step 3. Or if any fuzzy output membership weight is changed, repeat from Step 3(b).

It is very important to note that for a control problem, RNN and ORNN must be symmetrically trained; that is, we train the networks from the first training pattern to the last one and then from the bottom up. If the trained weights from Step 4(b) are to be loaded onto Step 4(a), we must be sure that they are successfully trained at the instance the controller quits from Step 4(b); then the controller will have an improved version for that particular environment. Additionally, we are not supposed to train the controller in a long irregular environment using Step 4(b). In order to eliminate some unintentional disturbances (errors that are not due to the controller itself, such as a short, sudden environment change), a counter can be set to record the number of times the controller approaches its control goal. Increase the counter when the goal is reached; decrease the counter otherwise. The counter can start up with some number according to the confidence that we have in our developed rule set at the beginning of Step 4(b). The controller starts to learn again when the counter drops to zero. This will give some time for the controller to verify its present weights. In addition, this enables the ORNN to serve as an output refinement network and to prevent the network from overfitting [CR95] (keep the controller from overreacting to every action). Furthermore, the networks tend to reconstruct partial inputs to complete sample inputs. When the membership values are close to the trained rules, the networks will have a sudden jump to its consequence. Therefore, these networks may just need to be trained loosely.

2.5 Learning Algorithm

The BP uses an objective function (2.1) [CXL92] defined as the sum of the squared errors between the desired and actual state variables. The learning of neural networks involves minimizing this objective function.

$$J = \frac{c_1 \epsilon_1^2 + c_2 \epsilon_2^2}{2} \quad (2.1)$$

where c_1 and c_2 are constants; ϵ_1 and ϵ_2 are the errors between the desired and actual state variables.

By applying the method of greatest gradient descent, the BP continually adjusts each weighting factor and each threshold bias according to the following recursive formulae.

The errors:

$$\begin{aligned} \sigma_k &= \beta |Y_k| (1 - |Y_k|) \\ \sigma_j &= H_j (1 - H_j) \sum_{k=1}^c \sigma_k W_{jk} \end{aligned} \quad (2.2)$$

where σ_k and Y_k are the error and the output of the k -th neuron in the output layer respectively; $\beta = \mathbf{c}_1 \epsilon_1 + \mathbf{c}_2 \epsilon_2$, which is the derivative of the objective function; σ_j is the error and H_j is the output of the j -th neuron in the hidden layer.

The weights updating:

$$\begin{aligned} W_{jk}[t+1] &= \lambda W_{jk}[t] + \eta \sigma_k H_j \\ W_{ij}[t+1] &= \lambda W_{ij}[t] + \eta \sigma_j X_i \end{aligned} \quad (2.3)$$

where λ and η are commonly known as the momentum factor and the learning rate particularly; X_i is the i -th incoming signal; W_{ij} and W_{jk} are the synaptic

weight factors— W_{ij} connects neuron i in the input layer to neuron j in the hidden layer, and W_{jk} connects neuron j in the hidden layer to neuron k in the output layer.

2.6 Design of the FMF: Fuzzy Membership Functions

FMF are employed to fuzzify input variables, and the number of input variables determines the number of membership functions to be constructed. The membership values range from zero to one since the sigmoid function is used. If scale factor is applied to the input/output, the membership values can be in any range if that is necessary. Piecewise triangular functions are commonly in use; however, any kind of function is acceptable as well. Observation of physical constraints is usually a good way to construct these membership functions.

2.7 Design of the RNN: Rule Neural Network

Out of any entire rule set, we believe that there are some basic parent rules [HG94], which are used to generate the rest of the children rules. Neural networks have learning and interpolating abilities; therefore, in order to extract as few rules from the human experts as possible, RNN is utilized to learn or capture the most important parent rules and then interpolate or generate the children rules. In the following, the general approach for rule derivation and the way to select the parent rules are discussed.

2.7.1 General Approach for Rule Derivation

There are some guideline for rule derivation:

1. Know your system.
2. Focus on the *objective* of the controller.
3. Prioritize the control parameters.

Knowledge of the system is the key to generating rules. In developing the entire rule set, we must always keep the control objective in mind. To develop the entire rule set, the following are some general procedures:

Step 1. Initialize a stable system.

Step 2. Perturb parameter k .

Derive a rule to affect system behavior in order to counter the perturbation in k . There are some important consideration here:

- (i) When will this rule be fired? (knowing from the *past*)
- (ii) What should be the controller's reaction to the perturbation? (reacting *now*)
- (iii) What is the expected behavior after controller reacts? (predicting the *future*)

Step 3. Entire range of parameter k considered?

Yes, go to Step 4.

No, go to Step 2.

Note that, for example, if an input parameter has the membership labels of n, z, and p, Step 2 will loop through three times at most before passing to Step 4.

Step 4. Perturb another parameter from the stable condition and iterate Steps 2 and 3 till all possible perturbations are accounted for. Perturb multiple parameters thereafter.

With all of the above knowledge, a whole set of rules could be developed. Nevertheless, as we have stressed that a minimal number of rules are to be created manually, we have to investigate which rules are the parent rules. There is one inspiration from solving differential equations—*boundary conditions*. Probably some rules are the *boundary rules*, which can be exploited to generate the rest of the children rules. A stable system is a controller's goal, so it must be one of the boundary rules. We observe further how a human balances a ball or drives a car. If a person, for example, can balance a fast-rolling ball toward one end of the beam on a highly inclined theta, we assume that he should be able to handle any mild situation as well. Therefore, the boundary rules are for the moments when the system is most stable and most critical.

2.7.2 Training Issues

RNN maps fuzzy input vectors to fuzzy output vectors. Its objective function is defined by the fuzzy actual outputs and the corresponding fuzzy target outputs.

According to Equation (2.2), every output neuron has its β , which is the difference between the target and the actual output.

Before going on to train the ORNN or to build the hardware, we can simulate our controller and plot the control surfaces to test our model. We plot the control output against other two parameters. From the control surfaces, fuzzy rules can be investigated if they are approximately correct. Yet they do not provide adequate information to fine-tune the rules. Generally, good control surfaces are monotonic without any obvious spikes or fold-around.

In Figure 2.2(a), it is not noticeable that the objective rule is missing. This may imply that fuzzy logic has good tolerance to missing and incorrect rules. Overall performance is not easily affected so long as most of the rules are approximately correct; on the other hand, the control surfaces do not give us adequate information to tune the rules. The plot in Figure 2.2(b), which is fold-around, tells us that the controller will definitely not work, since there are too many incorrect or missing rules.

2.8 Design of the ORNN: Output-Refinement Neural Network

There are three main reasons for having ORNN: First, the knowledge that we have embedded in RNN will not be altered when learning continues. Second, a smaller network has faster weight updated rate. Third, a sole neuron at the output layer eases the credit assignment problem.

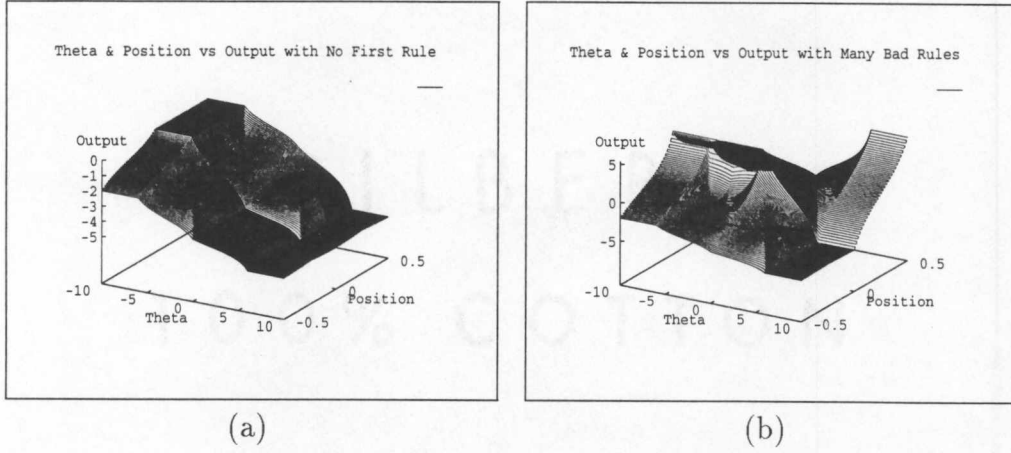


Figure 2.2: *The control surfaces of a fuzzy controller with missing or bad rules. (a) The objective first rule is missing. (b) There are many extremely incorrect rules. We plot the control output against other two parameters. In general, good control surfaces are monotonic without any obvious spikes or fold-around.*

ORNN is trained for two phases, off-line and on-line. During off-line training, ORNN's inputs are connected directly from the outputs of RNN. The defuzzified output of the RNN is ORNN's target output. For defuzzification, a singleton method [HG94], which uses weights w_i for each output fuzzy set and the output membership values V_i , is computationally efficient. However, the method described here is different from the Higgins and NeuFuz4 method [HG94, Tha93]. This NiF-T does not have a physical layer for output membership functions in the RNN, and the weights w_i are all fixed. The crisp output of RNN is calculated as follows:

$$O = \frac{\sum w_i V_i}{\sum V_i}$$

Giving an example of 0.5z and 1.0s at the RNN's output, where 0.5 and 1.0 are

the V_i while z and s have the weights of 0 and 2, respectively, the defuzzification results in a crisp output of 1.33.

After transferring the rule knowledge to ORNN, the controller is ready to test out. As we have mentioned in Section 2.7.2, control surfaces can be plotted here again to verify the knowledge transfer. All of them should resemble the RNN's. For on-line operation, ORNN can be selected for continual learning. While learning continues, ORNN uses an objective function to minimize the controller errors. Note that the constants, such as c_1 and c_2 , require a full dynamic of trial and error. Usually it is a good practice to start with some small numbers as the idea of fine-tuning.

2.9 Summary

This chapter has introduced a new neural integrated fuzzy controller. It possesses the beauty of both fuzzy logic and neural networks. All the design issues and implementing procedures have been discussed. In the next three chapters—real-time control of BBB, autonomous navigation of a mobile robot, and controlling multiple mobile robots moving in formation—utilizing the knowledge which has been developed here; are presented. We use the BBB problem to represent a class of nonlinear control problems involving noisy sensory signals and real-time feedback requirements. Mobile robot navigation and especially multiple mobile robots moving in formation are difficult to obtain their mathematical models. The NiF-T

that has nonclassical means is applied to solve these problems in the hope that it can accomplish the tasks more efficiently because fewer parameters are in the design consideration.

CHAPTER 3

Real-Time Control of Ball Balancing Beam (BBB)

3.1 Problem Statement and Specifications

Ball Balancing Beam (BBB) is a system that moves a metal ball along two wires on a rigid beam which is centrally hinged, as shown in Figure 3.1. To compensate for the ball movement, a motor rotates the rigid beam so that the ball will stay balanced and eventually centrally positioned on the beam. This system, which is nonlinear, has delayed feedback along with noisy signals, especially when the ball jumps. It needs some special techniques and complicated mathematical derivation in conventional control methods [Wel79]. Therefore, here we use the BBB problem to represent a class of nonlinear control problems involving noisy sensory signals and real-time feedback requirements. The NiF-T that has nonclassical means is applied to solve this BBB problem in the hope that it can accomplish the task more efficiently because fewer parameters are in the design consideration. The following are the BBB NiF-T's special features:

- No detailed mathematical derivation.
- Qualitative description of the nonlinear system behavior (model-free).

- Continuous representation and human knowledge incorporation.
- Learning capability.
- High noise tolerance.
- Ball size and mass not in the design consideration.
- Ease of scaling the beam to different lengths.

We would like to conduct experiments to verify as many of the items as we have listed. In order to show that NiF-T does have the characteristics mentioned above, we choose to perform hardware experiments on a simple mechanical system but a fairly difficult control problem, i.e., the *Ball Balancing Beam*.

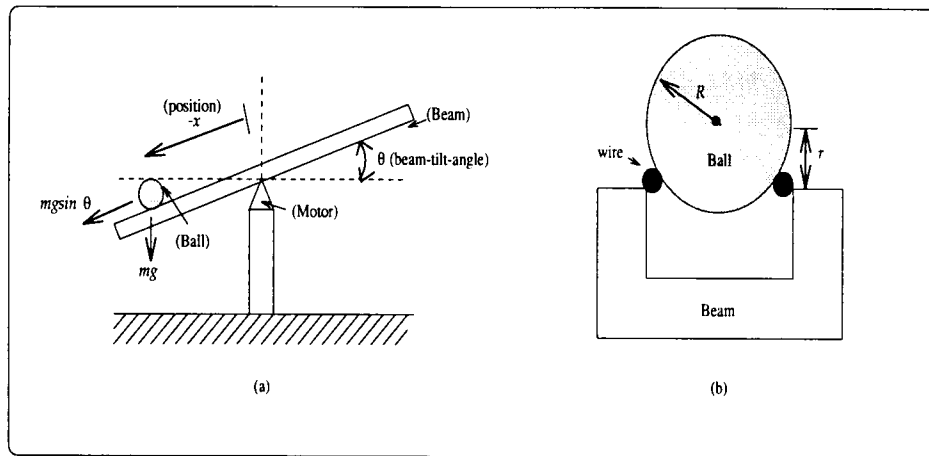


Figure 3.1: *The schematic of the full hardware implementation of BBB.*

3.2 Review of Other Approaches

A number of diverse methods have been applied to the BBB problem [Wel79, SH94, FB94, NT95]. Classical control theorists have claimed that they could bring a ball to rest on a beam in approximately four seconds [SH94]; they, however, did not balance the ball at a specific position. In addition, fairly advanced mathematical background and control theory understanding are required to derive the required controller [Yam93]. A reinforcement neural controller learned to balance a ball; nevertheless, the ball typically never settled down. This is acknowledged to be due to the use of “discrete resolution of the boxes representation,” and the ball was not balanced at a specific position either [FB94]. With this pure neural network alone, it is hard to incorporate expert knowledge into a control system, because neural networks are typically treated like a “black box” in which the function learned can be observed only through the input/output relationship [HG94]. Fuzzy logic which has continuous functions mimics a human mind to realize a control goal without intensive involvement in mathematical derivation [KI93, Bez93a]. The fuzzy logic controller (FLC) balanced balls around a specified position. The drawbacks are (1) generating all these fuzzy rules is just too tiresome; (2) it is too time-consuming to check many if-then conditions by usual microprocessors in the real-time implementations [BI93]; (3) FLC does not have the learning capability. Utilizing either classical methods or neural networks or fuzzy logic is probably sufficient to achieve a control goal, such as balancing a ball. Still and all, taking

advantage of the best of fuzzy logic and neural networks is believed to be more efficient in design and to have better results [AP93].

3.3 The Control Scheme

We have chosen three parameters—ball velocity, beam theta, and ball position—for inclusion in this design. For a rigid beam, theta-dot is insignificant. Besides, ball and beam accelerations are not considered, mainly due to their limited effect on the system but also in order to limit the total number of parameters (the more parameters we have, the more rules have to be generated). Each of these three parameters has the evaluation of negative(n), zero(z), and positive(p). Subsequent to mapping, there are a maximum of $3^3 = 27$ possible rules [Bez93b]. The BBB control process block diagram is illustrated in Figure 3.2. A ball position sensor and a motor controller feed these three data successively to the NiF-T. The NiF-T sends the command to the motor to balance a ball after logical reasoning. That produces a new beam angle and ball position to the sensory system. Thereafter, the cycle repeats.

The NiF-T contains three main parts: FMF, RNN, and ORNN, which we have already introduced (see Figure 2.1). Ball velocity, beam theta, and ball position are fuzzified through FMF. Each fuzzified parameter has labels of n, z, and p, with membership values from zero to one. Accordingly, three parameters yield a total of nine input labels for RNN. The hidden layer of RNN is set to 27 nodes

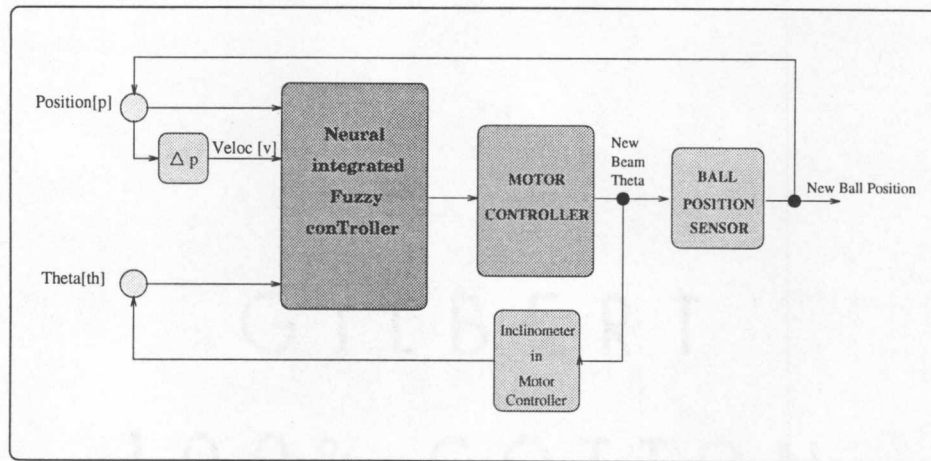


Figure 3.2: *The control process block diagram of BBB*

arbitrarily to represent 27 rules. Five labels—zero(z), small($\pm s$), and large($\pm l$)—are RNN outputs. We skip the medium($\pm m$) labels since the rules that are used for training patterns do not have these labels. ORNN's inputs are linked together with the RNN's outputs. There are three hidden nodes and a single neuron in the ORNN.

3.3.1 Membership Functions Derivation

With three necessary parameters—ball velocity, beam theta, and ball position—there are three membership functions to be constructed. All the membership functions in this design (shown in Figure 3.3) are fully overlapping piecewise triangular functions.

Ball position is considered zero at the beam center, very negative at the left end of the beam, and very positive at the right end. Velocity membership function

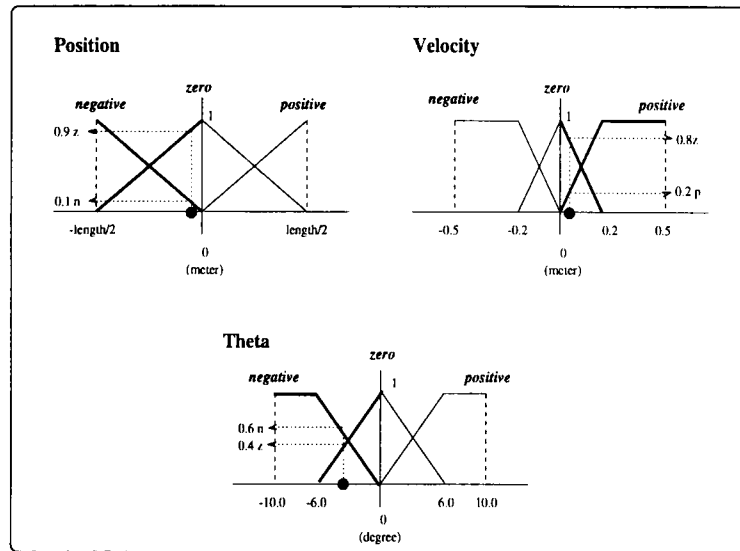


Figure 3.3: *Input parameter membership functions. Observation of physical constraints helps to construct these membership functions. The symbol representations are listed as $\text{zero}(z)$, $\text{positive}(p)$, and $\text{negative}(n)$.*

is not as instantly constructed as that of the position, because we do not know how fast is considered fast enough. Hence, a ball was experimentally rolled along the wires on the beam as fast as possible, and every point of the movement was recorded. From the data points, we determined that 20 centimeters away from the last position is considered fast enough. Theta membership can be set according to how much the initial inclination is still allowable to balance a ball. Tilting clockwise is in the negative direction.

3.3.2 Rules Derivation

We rolled a ball on the beam at various positions with different velocities and also balanced the ball by hand to see how the system responded. From our observations, the ball must be stopped and prevented from accelerating. In other words, our first priority is to slow the ball's speed. After that, we can send the ball to the center and balance it there when it is rolling slowly on a much-less-inclined beam. Due to the imperfection of the hardware, the beam need not be at zero theta so long as the ball is eventually stationary around the beam center; we expect the ball to roll in the direction that the beam is tilted. For example, the ball should start to roll to the left when the beam is tilting counterclockwise; otherwise, we must produce more effort to tilt the beam counterclockwise. Thus, the important sequence of the control parameters is velocity, theta, and finally position. Understanding the nature of the problem by watching the physical phenomenon, we are ready to convert our observations and knowledge into linguistic rules.

The objective of this system is to send a ball back to the center and keep it balanced there. The "delta-rule" is used to minimize each parameter's errors. Data are fed back to the system; if errors are huge, the system should respond faster and with greater effort. Before we achieve our goal, the ball speed must be slowed down. When we develop the entire rule set, this objective must always be kept in mind.

With three input parameters, a pure FLC may need a total of 27 rules to cover the entire rule base. Of these 27 rules, we believe some are basic parent rules, as

we mentioned in Section 2.7.2. Table 3.1 shows these boundary rules, which are some nine rules extracted from the FLC in [NT95]. They must be converted into binary format, which is shown in Table 3.2, before they are usable for training.

The results of the RNN rule interpolation are shown in Figure 3.4 (d)–(f). When they are compared with those of FLC’s control surfaces from (a)–(c), they appear to have similar shapes, except that the FLC’s are more structured [Kos91], while the NiF-T’s are smoother. The smoother control results obtained by this NiF-T may be closer to those that experts have in mind [BI93].

3.3.3 Objective Function

The two errors are ball position (ϵ_1) and velocity (ϵ_2) for Equation 2.1. Since ball velocity has the higher priority to be controlled, the maximum value of $\epsilon_2 * c_2$ should at least equal to $\epsilon_1 * c_1$. The weights of the RNN output fuzzy set are -6.0(-l), -2.0(-s), 0.0(z), 2.0(s), and 6.0(l).

3.4 Experimental Testbed Development

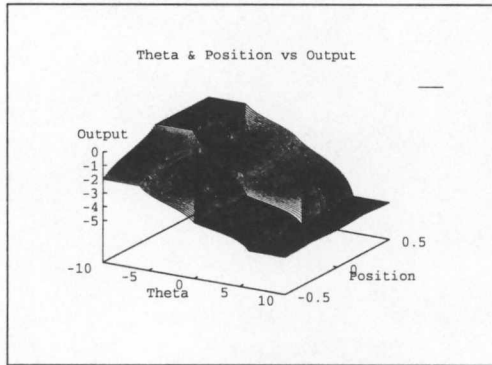
The hardware implementation and the control process of the BBB have been illustrated individually in Figures 3.1 and 3.2. This testbed consists of four major parts: beam, sensor, motor, and computer system. The sensory system continuously feeds data to the controller. A VME, which logs in from a workstation computer, processes these data, acting as the NiF-T. Then the motor serves as an activator to balance a ball. We have discussed the NiF-T block in the previous

Table 3.1: *These nine parent rules, one-third of the conventional FLC rule set, are exploited to generate the rest of the children rules. The first rule is the objective most stable condition, while the rest of them are intended to handle the most critical situations.*

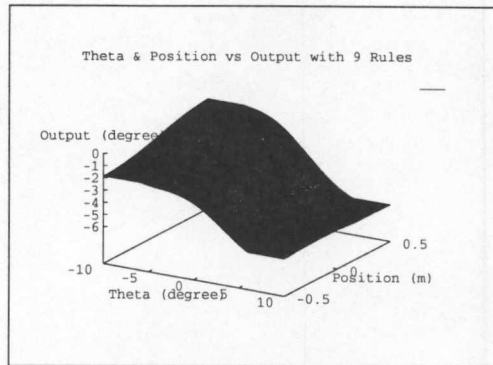
Parent Rules					
Rule	Inputs			Output	Cond.
	Velocity	Theta	Position		
1	z	z	z	z	Stable
2	p	p	p	s	Critical
3	p	p	n	-l	
4	p	n	p	z	
5	p	n	n	-l	
6	n	p	p	l	
7	n	p	n	z	
8	n	n	p	l	
9	n	n	n	-s	

Table 3.2: *RNN is trained to remember these nine rules. Both inputs and outputs are fuzzy vectors. When input membership values fall between 0 and 1, RNN will interpolate the outputs after it learns.*

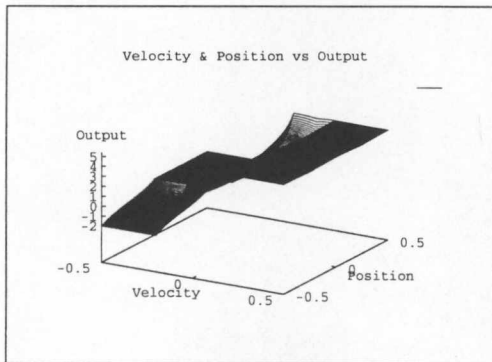
RNN Training Patterns														
Rule	Inputs									Target Outputs				
	Velocity			Theta			Position							
	<i>n</i>	<i>z</i>	<i>p</i>	<i>n</i>	<i>z</i>	<i>p</i>	<i>n</i>	<i>z</i>	<i>p</i>	<i>-l</i>	<i>-s</i>	<i>z</i>	<i>s</i>	<i>l</i>
1	0	1	0	0	1	0	0	1	0	0	0	1	0	0
2	0	0	1	0	0	1	0	0	1	0	0	0	1	0
3	0	0	1	0	0	1	1	0	0	1	0	0	0	0
4	0	0	1	1	0	0	0	0	1	0	0	1	0	0
5	0	0	1	1	0	0	1	0	0	1	0	0	0	0
6	1	0	0	0	0	1	0	0	1	0	0	0	0	1
7	1	0	0	0	0	1	1	0	0	0	0	1	0	0
8	1	0	0	1	0	0	0	0	1	0	0	0	0	1
9	1	0	0	1	0	0	1	0	0	0	1	0	0	0



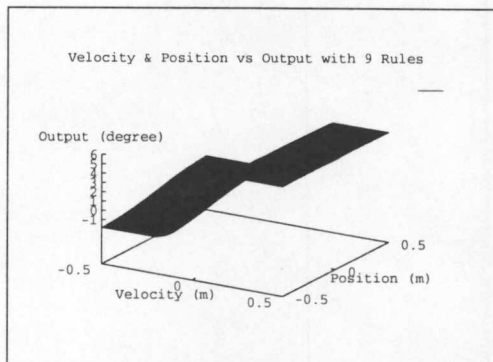
(a)



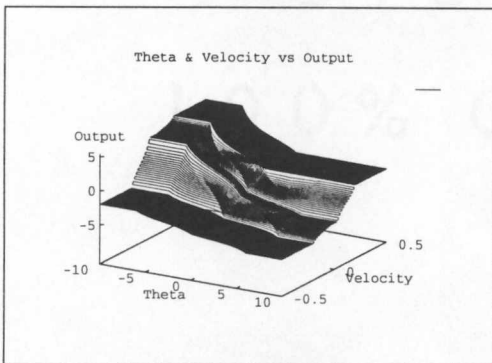
(d)



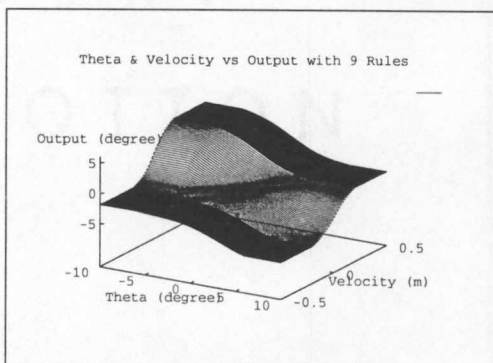
(b)



(e)



(c)



(f)

Figure 3.4: Comparison of control surfaces derived using only Fuzzy Logic (a-c) and Fuzzy-Neural approach (d-f). The fixed parameter is set at negative extreme. All of these surfaces are monotonic without any obvious spikes and fold-around.

sections. The sensory system and the motor activator designs are presented here.

The components that are required for the setup are listed in Figure 3.5.

Parts List	
Sensor	Beam
(1) +10v, 1 Amp Power Supply	(1) Aluminium Sheet
(2) Nichrome Wires	System
(3) A/D in VME	(1) VME
(4) Metal Ball	(2) Workstation
Motor	
(1) Pan-tilt-unit from manufacturer, which has built-in inclinometer	
(2) RS232 connection to VME	

Figure 3.5: *The parts list.*

3.4.1 Design of the Sensory System

As we have shown in Figure 3.1(b), the sensor is a contact of a simple metal ball and two strips of nichrome wires. We use the nichrome wires due to their high resistivity, which can give better voltage resolution (position reading depends on the voltage resolution). A power supply constantly feeds one amp of current to the nichrome wires when a metal ball rolls along on the wires. Varying the contacted wire length gives us different voltages according to Equation (3.1).

$$\begin{aligned}
 R &= \eta \frac{l}{A} \\
 V &= RI
 \end{aligned}
 \tag{3.1}$$

where R = resistance, V = voltage, I = current; η , l , and A are the resistivity, length, and area of the nichrome wires, respectively. The changing voltages are measured by an analog-to-digital (A/D) board on the VME. The voltage reading performance, which is very noisy, is shown in Figure 3.6.

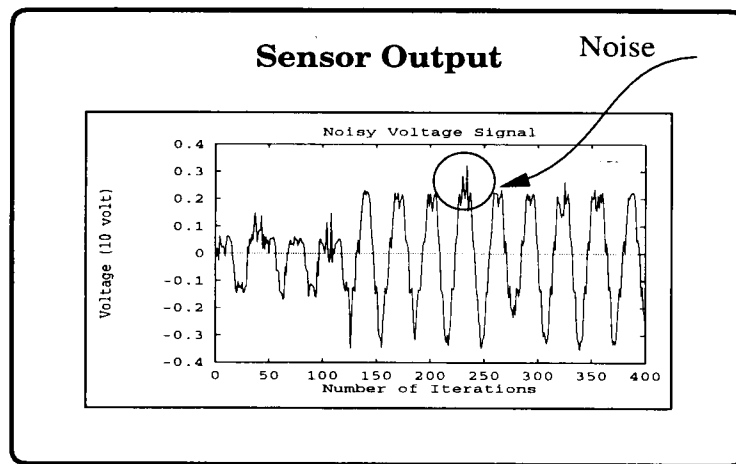


Figure 3.6: *Noisy voltage signal.*

There are many reasons why the signal is so noisy. The surface of the ball and the wires are not very even or smooth, which can cause the ball to bounce when it rolls on the straightened wires. As you look at Figure 3.6 again, the signal is particularly noisy when the ball reaches the center, or at the moment that it changes direction. The center of the two tightened wires has less stress, so it tends to spread out when the ball approaches that point. When the ball changes direction, it causes the system its most unstable moment. After all, these results are the best we can get with this setup, and the signal has already been filtered and averaged.

The VME system has been brought up by an interrupt routine to a frequency of 5KHz to acquire 250 data points from the A/D board for each iteration. These 250 data points contribute to one stationary ball position for each iteration. We choose to get 250 points because if the data points are not enough, we will have less position accuracy; or if we get too many points, the ball may not be at the one stationary position. All of these decisions have been tested out in the experiments. These 250 points are band-pass-filtered, and those passing points are averaged. Therefore, we get every position point at around 20Hz. From acquiring data to completing the control output, however, the system takes about 0.2 second. The system will run faster if we acquire fewer data points.

Once we have the ball position, we can calculate the ball velocity. The ball velocity is the difference of the new ball position from the old ball position.

3.4.2 Design of the Activator

This is a pan-tilt unit capable of panning and tilting with changing commands on the fly. The pan-tilt commands are sent out via the RS232 serial links to a VME-based real-time computing system (Motorola 68030) that is running VxWorks. The beam angle and its angular velocity can be set and acquired directly by these units. The ability to control motor speed can reduce jerking in the system, and it further enables the controller to run with fewer delays when associating with the feature of changing commands on the fly.

Each time we send an output to tilt the beam, we set the motor speed as

well so that the process ends at another new output. Additionally, the ability to change commands on the fly can reduce the effects of acquiring new data and control process delays. Without having to wait for the motor to finish its tilting command, we can get new sensory data and go through the whole control process. Unfortunately, our data still may not be the most up-to-date.

3.5 Experimental Validation

There are two sets of experiments altogether. One is with the direct ORNN output, the other with the ORNN continuously trained. Each set includes four experiments: single ball of small, medium, and large, and the multiple balls of small and medium. The purpose for using different balls is to justify how well the NiF-T responses to the change of ball mass. In these runs, the controller succeeded in balancing all of them. The BBB real hardware setup is shown in Figure 3.7. A sequence of actions were applied to a large ball when it was rolling from left to right on the beam (Figure 3.8). The beam at first was inclined right. Then it was tilted counterclockwise to counteract the ball movement in order to reduce the ball speed.

The performances of the NiF-T are shown in Figure 3.9. Results of (a)–(d) are directly output from the ORNN with its weights frozen. In (a), (b), and (d), the balls were balanced off the center, the reason being that we did not train the ORNN off-line well enough to receive the complete knowledge of RNN. Therefore,

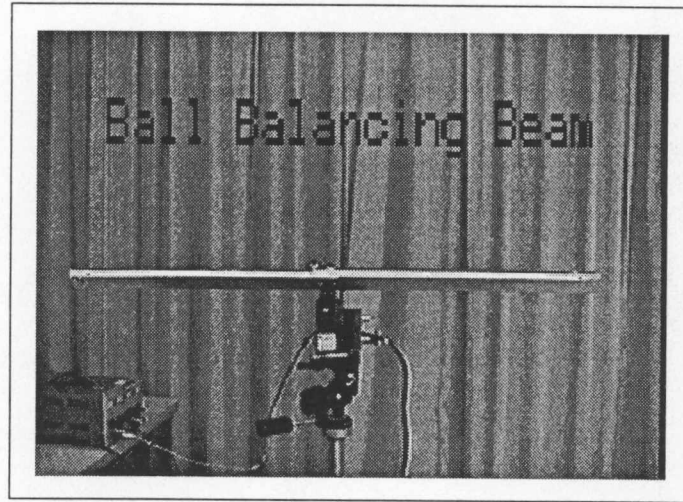


Figure 3.7: *The BBB experimental testbed.*

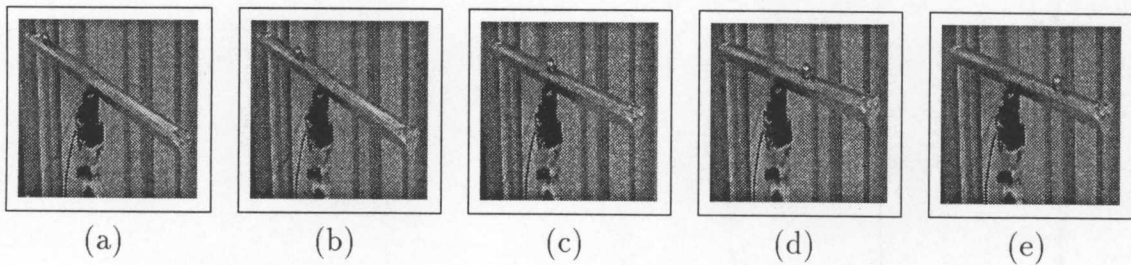


Figure 3.8: *A large ball was controlled on the beam continuously from (a) to (e).*

the remedy could be to train the ORNN further. Using more nodes in the ORNN's hidden layer should also help its learning memory. However, without going back to the off-line training, the controller approached its original goal by adjusting the ORNN's weights during on-line operation. The balls were balanced symmetrically around the center with small oscillation amplitudes in plots (e)–(h). At one time, they almost rested at the center. Yet a ± 0.01 degree can cause the balls to gain acceleration and gradually to speed off one end of the beam. When

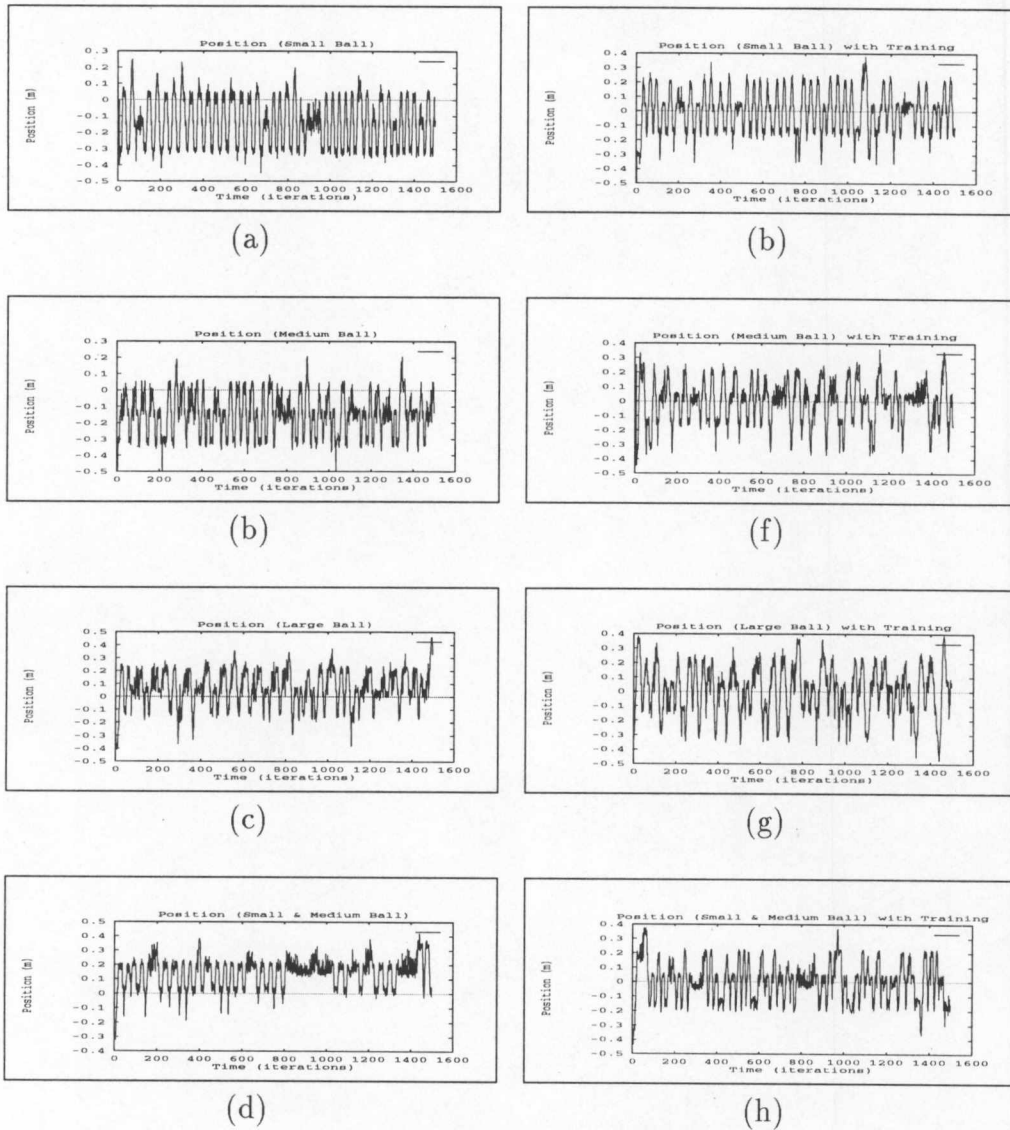


Figure 3.9: *The system performance. Results of (a)–(d) are directly output from the ORNN without further training. The balls were balanced off the center. After learning continued, the controller was able to shift the balancing back to around the center in (e) to (h).*

comparing to the BBB FLC performances in Appendix B, it is obvious that the NiF-T does not discriminate by ball size and mass.

After investigating the plot of position versus theta, Figure 3.10 shows that position and theta have linear relationship. There are some discontinuities along the position axis. This is due to the fact that the ball oscillated back and forth, and the system recorded the ball position mostly at approximately -0.1, 0, and 0.2 meters with different beam inclinations. As a result, with a system that runs faster than 5Hz or with a smaller motor output, the ball may oscillate with smaller amplitude or even be balanced at the beam center. Relationship of position and theta indicates that changes of theta directly set the ball position. Both position and theta data points fall mostly in the same sign quadrant for a functional controller. Theta is controlled to move with the ball in the same sign direction to reduce the ball acceleration. Understanding these parameters' relationship will help us further improve the performance.

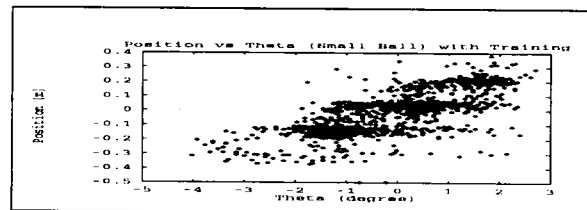


Figure 3.10: *Relationship of position and theta. The linear relationship indicates that changes of theta directly set the ball position. Understanding the parameter's relationship will help further improve the controller.*

The objective of the controller is to send the ball back to the center and balance

it there on a much-less-inclined beam by slowing down the ball first. The beam is tilted to achieve this goal. On the other hand, the ball velocity and position and the beam theta determine the tilting efforts and directions. From plot (a) of Figure 3.11, velocity is observed to be the most crucial parameter in this design. The controller is trying to keep up with the ball's speed and rolling direction in order to slow the ball down. As we have discussed in Section 3.3.2, theta need not be at zero as long as the ball is balanced around the center. Consequently, variation of theta in plot (b) does not give much influence to the controller output when the ball oscillates back and forth from -0.2 to 0.2 meter most of the time.

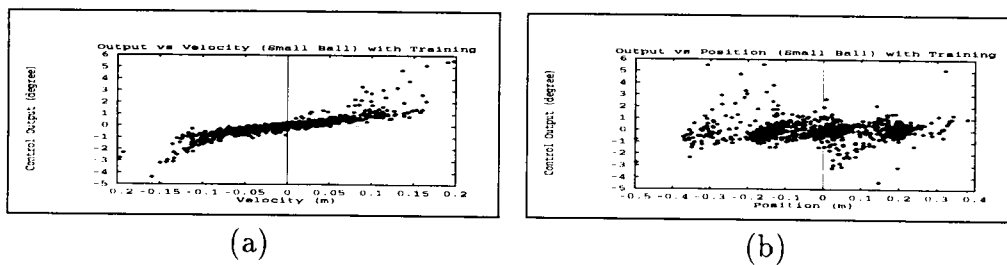


Figure 3.11: *Control output versus system parameters. The controller is trying to keep up with the changes in the ball's speed and direction. The results validate the appropriateness of the rule set used.*

Velocity is verified to be the top priority parameter to control. Furthermore, theta need not be zero when the ball is balanced around the center. The knowledge which is used to develop the controller's rules is again justified here. All of these results validate the appropriateness of the rule set used.

3.6 Summary

This NiF-T for BBB has been successfully designed and implemented using a real-time computing system to balance balls with one-third of the rules required in a conventional FLC rule set. With this NiF-T architecture, the controller capitalizes on the strengths of both fuzzy logic and neural networks. Fuzzy logic provides a continuous representation and human knowledge incorporation into the system. Membership functions make the controller less sensitive to slight variations in the physical parameters and control goal. With the learning capability of neural networks, fewer rules are to be created manually, and the controller is fine-tuned to its original goal. Fuzzy logic and neural networks are both model free; therefore, there is no mathematical equation to derive. The controller inherits their high fault tolerance feature and so can endure noisy signals.

CHAPTER 4

Autonomous Navigation of a Mobile Robot Using Sonar Sensor

4.1 Problem Statement and Specifications

Two behaviors, wall following and hall centering, are realized in this chapter. The robot is supposed to hug wall at any specified distance and to center itself in a hallway for the wall following and hall centering behaviors, respectively.

There are three basic questions for the problem of navigation: where am I?, where am I going?, and how should I get there? [LW92]. A rotary sonar sensor is used to acquire the localization of a robot. After processing the sensor information, the proposed NiF-T in Chapter 2 tells the robot where and how to go. The speed of the robot is set constant here. Only the orientation of the robot is changing continuously to reach the control goal. The following are the specifications to verify:

- No detailed mathematical derivation.
- Qualitative description of the nonlinear system behavior (model-free).
- Continuous representation and human knowledge incorporation.

- Learning capability.
- High noise tolerance.
- Little influence by changes in the robot speed.

4.2 Review of Other Approaches

In order for a robot to navigate autonomously without running into any obstacle and to reach its designated destination, there are three major problems to consider, as we have addressed in the previous section. The robot must recognize landmarks in order to know its localization, and it must recognize obstacles in order to navigate safely. A related but somewhat more complicated behavior is that of obstacle negotiating. In this not only the obstacle needs to be detected but the robot should climb over it rather than avoid it. Chen and Trivedi [CT93] described a controller for a sonar-based tracked mobile robot for obstacle negotiating. A goal-driven robot needs path-planning, and then a controller tells the robot how to get there. In the literatures, [Tri93, Tri94], the components required for the intelligent robotics systems were thoroughly discussed. Professor Trivedi [Tri93, Tri94] pointed out that the capabilities of an intelligent robot are provided by four system modules: (1) Sensing; (2) Perception/Planning and Control; (3) Motor; and (4) Workspace. In this chapter, we focus mainly on designing a controller for the intelligent robotic system using a sonar sensor with the motor activators.

Researches have attempted to use fuzzy logic and neural networks to solve these problems. MORIA [SHP95] had two rule blocks. One was used to recognize the perceptual environment, the other to drive the robot. Lee's AGV [LW94] avoided static and moving obstacles and navigates from a starting point toward the target using six fuzzy logic control modules. Although Fukuda has applied Fuzzy Template Matching [FIAY95] along with a neural network to detect the ceiling landmark, fuzzy logic basically has been acting more as a controller to tell the robot how it should react in a situation. On the other hand, neural networks have broader applications in robot navigation problems. That includes landmark recognition, path-planning, and controller. Kohonen network is more popular in topological environment maps building. ALICE robot [ZP94] recognized places where it had been before, using Kohonen's neural network clustering techniques. Tani [TF93] combined Kohonen and feedforward networks to navigate YAMABICO robot to a predetermined goal in arbitrary workspace without global information. Janet [JROC⁺95] was trying to globally self-localize a mobile robot utilizing Kohonen neural network as well. Using a topologically organized neural network of a Hopfield type, Glasins [GKG95] demonstrated that his simulated robot could move from any arbitrary start position to any static or moving target position by avoiding both static and moving obstacles. There are many other related works involved in robot autonomous navigation. Nagata [NSA90] had his cops and thieves robots controlled by a structured hierarchical neural network in the early 1990s. The success of ALVINN, MANIAC [JPT93], and NEURO-

NAV [MK92, MK93] also had an impact, especially their use of vision system to extract landmark recognition with neural networks. Meng had more involvement in working on the three mentioned issues. His navigation system included landmark detector, path planner, and rule-based supervisory controller.

In this chapter, we specifically select the indoor environment to perform two navigation behaviors using sonar sensor. No feature extraction is concerned due to the fact that the major purpose is to show that the proposed NiF-T is general enough for various feasibilities of control problems. The wall following behavior seems different from the hall centering behavior. Actually, both are required to lock onto their specified paths. From the control aspect, they are different mostly in the fact that wall following behavior needs only one side wall information.

For these two behaviors, some similar works have been done, such as [CXL92, LHGK92, YS94, Kos91, Hol94]. Cheng [CXL92] had a simple network which minimized both orientation error and road offset. He showed that it was easier to find an optimal parameter set for the neuromorphic controller than for the proportional controller. Lubin [LHGK92] and Yu [YS94] similarly were interested in locking their robots onto the center lane. Lubin's network required thousands of training samples to train for thousands of iterations. Yu's incremental learning suffered from long training time by trial and error, as well. Kosko's truck backer-upper systems [Kos91] are the related work, which minimize both orientation error and position offset so that the truck will align with the desired loading dock. As many as 35 rules were developed in his fuzzy system and more than 3,000

training samples were used to emulate the neural network (or with 35 training-sample vectors to train for 100,000 iterations). Holder's robot hugged wall [Hol94]; however, the road offset could be more than an inch although the robot speed was very slow. Since these kinds of navigation behaviors are easily understood, incremental learning may not be necessary. Therefore, with capabilities of learning with minimal number of training rules, the proposed NiF-T may again be proved to be efficient in design and better in results.

4.3 Experimental Testbed

The mobile robot shown in Figure 4.1, Small Mobile Autonomous Robotic - Testbed (SMAR-T), is utilized to accomplish the robot behaviors discussed in this chapter. SMAR-T is a robot which Holder built for his master's thesis [Hol94]. It has distributed multiprocessors which allow the high level control consisting of C/C++ routines to provide a multitude of functions. It is capable of intercommunicating with additional robots over wireless modems. Only one foot high and 18 inches in diameter at its base, SMAR-T weighs approximately 35 pounds. Its drive wheels are driven with a 2:1 gear ratio to reach motor torque of 250 oz.in. The maximum speed it can reach is one foot per second.

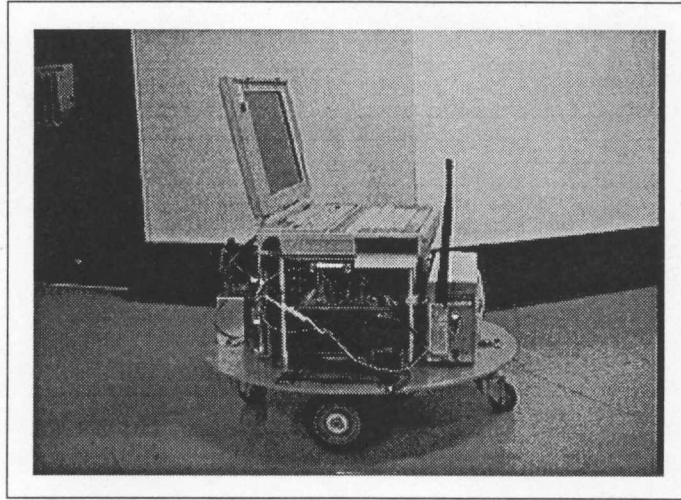


Figure 4.1: *SMAR-T: Small Mobile Autonomous Robotic Testbed.*

4.4 Wall Following Behavior

4.4.1 The Control Scheme

The robot can be designated to follow at any specified distance from the right or left wall. When following the right wall, the robot sonar sensor scans only the right plane region (from -15.0 to $+90.0$ degrees for the right plane and from $+15.0$ to -90.0 degrees for the left plane). Unfortunately, due to hardware constraints, sufficient sensor information cannot be acquired for the negative region of the right plane and for the positive region of the left plane. The robot scanning regions and the input parameter denotations are shown in Figure 4.2. We consider only two input parameters, road offset and orientation error, in this design. The robot speed can be one of the input parameters as well; for simplicity, it is set constant for the time being.

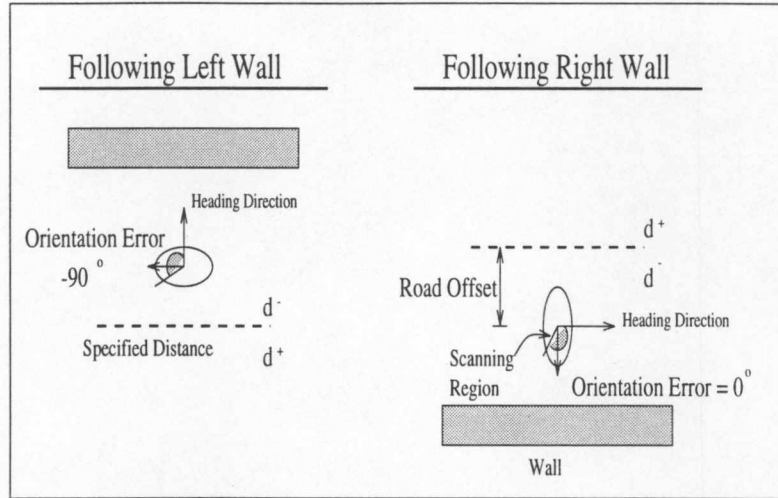


Figure 4.2: *The robot scanning regions and the input denotations for following wall.*

The membership functions are constructed as in Figure 4.3. When the robot is in position between the wall and the specified distance, the road offset is in the negative region (d^-); otherwise, it is in d^+ . The robot heading direction is supposed to align parallel with the wall. If the robot is heading toward the d^- region, the orientation error is a negative value; otherwise, it is positive. Since the sonar sensor can echo a distance as great as 30 feet, the road offset membership function is extended to 30 feet.

The control goal here is to align the robot parallel with the wall at a specified distance. For instance, when the robot is heading farther away from the specified distance trail toward the wall, the controller is supposed to turn the robot with a large positive degree. A set of parent rules is listed in Table 4.1. The weights of the RNN output fuzzy set are $-60.0(-1)$, $0.0(z)$, and $60.0(1)$ degrees. We trained

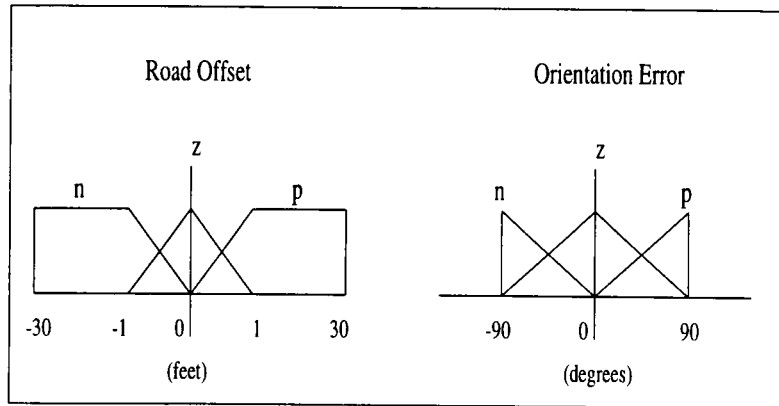


Figure 4.3: *The membership functions for wall following behavior.*

Table 4.1: *The wall following parent rule set.*

Parent Rules				
Rule	Inputs		Output	Cond.
	Road Offset	Orient. Error		
1	z	z	z	Stable
2	p	p	-1	Critical
3	p	n	z	
4	n	p	z	
5	n	n	1	

RNN and ORNN with the same five hidden layer nodes with the parameters of $\eta = 0.3$, $\lambda = 0.9$, and the desired RMS error = 0.05. The RNN converged at 966 iterations, while the ORNN converged at 90 iterations. This shows that the ORNN has a faster learning rate. We also include road offset and orientation error in the objective function; however, more emphasis has been placed on the road offset.

4.4.2 Experimental Validation

The experiments were carried out in a hallway with four doors on the left and three on the right. The doors were all closed and were recessed about four inches into the wall. Figure 4.4 shows the robot, SMAR-T, following the left wall in the hallway. SMAR-T was at first a few feet from the left wall, facing away. The robot turned toward the wall and hugged at the specified distance. There are four sets of experiments for four different robot speeds from 50Hz to 300Hz. Every set includes ORNN with or without on-line learning. All were able to follow the wall at the specified distance. The results are shown in Figures 4.5 and 4.6.

It is expected that the robot approaches its goal more steadily with slower robot speed (Figure 4.5) than with faster robot speed (Figure 4.6). When the robot runs too fast, the input data acquired by the rotary sonar sensor are quickly outdated. In addition, the effect of learning is not noticeable, for two primary reasons: (1) the developed rules are good enough; and (2) there are too few iterations (the hallway is not long enough) to effectively learn whether there is

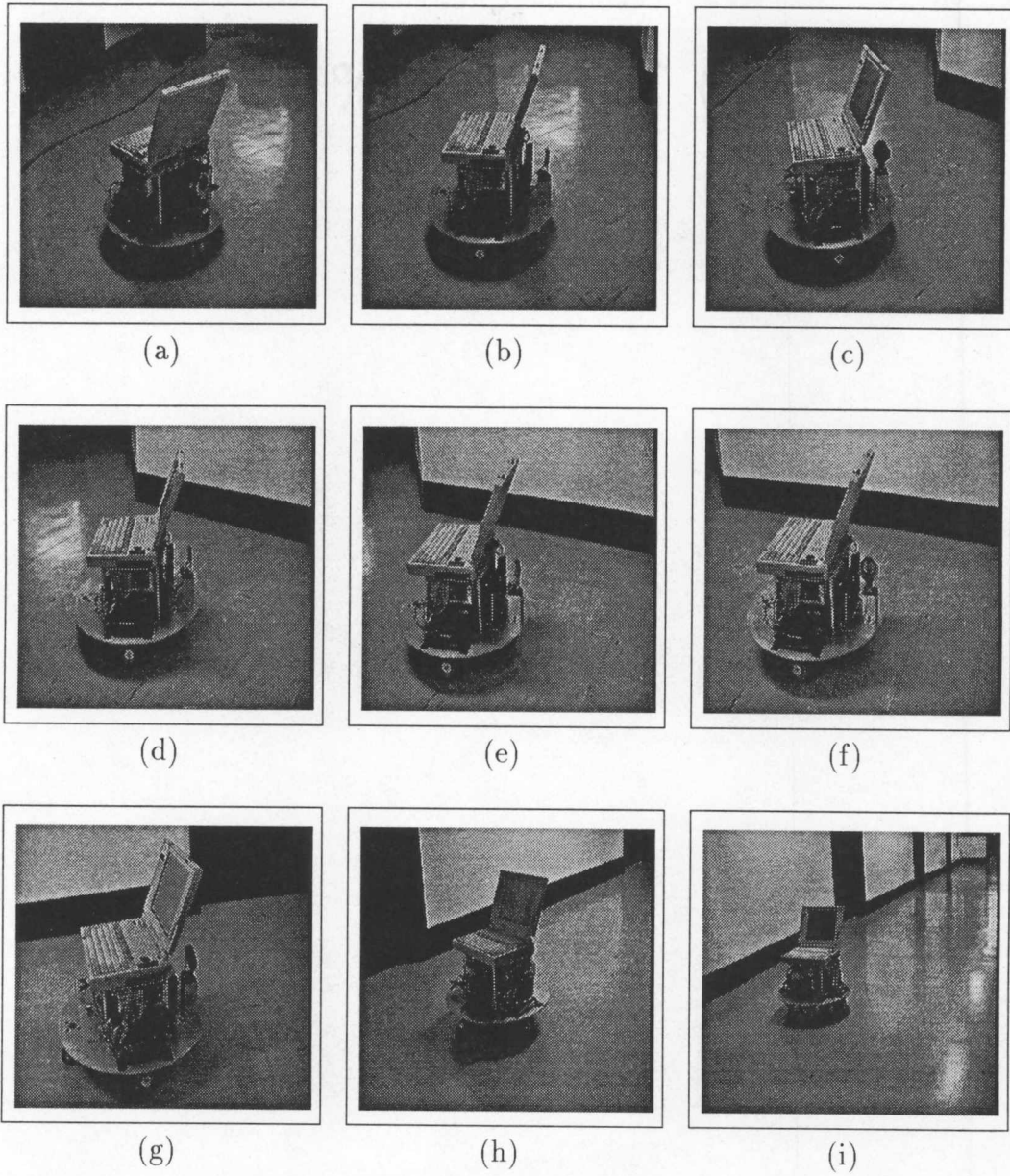
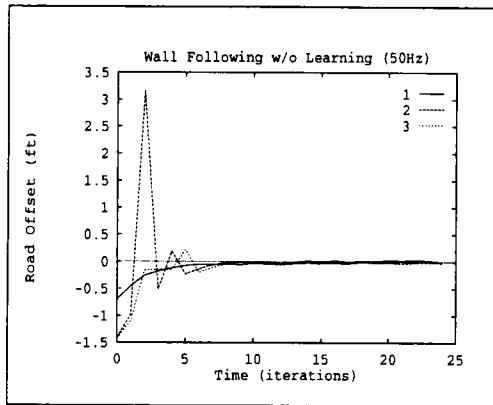
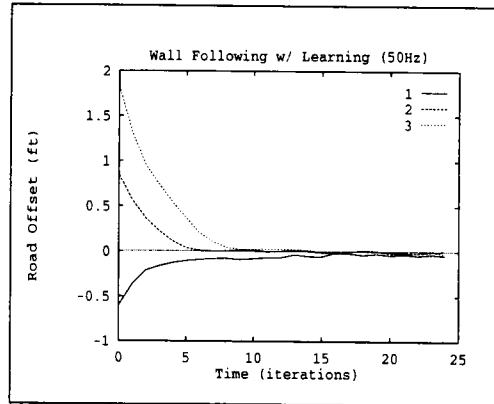


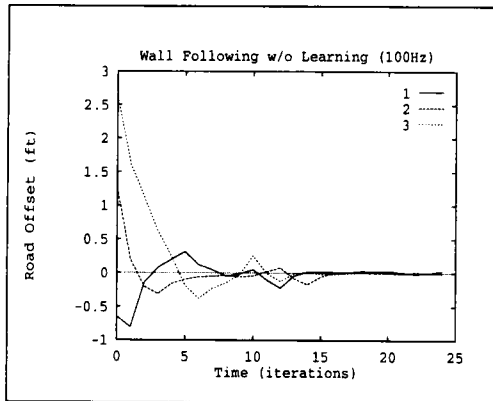
Figure 4.4: *SMAR-T followed the left wall in a hallway.*



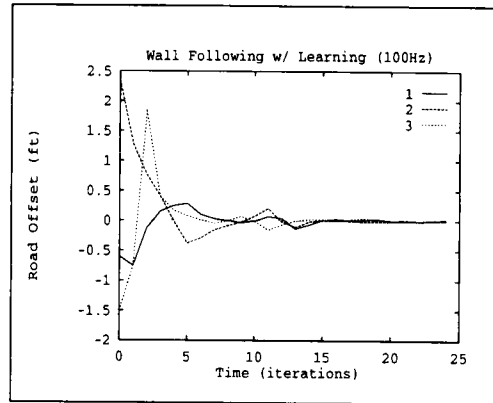
(a)



(b)

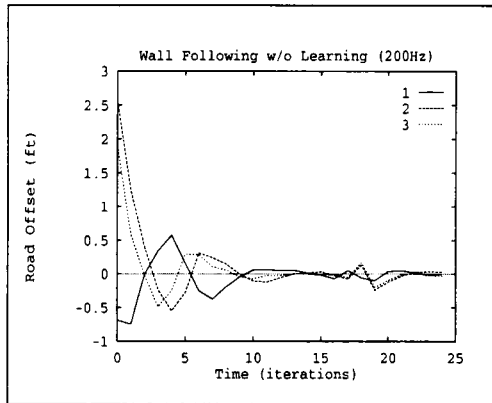


(c)

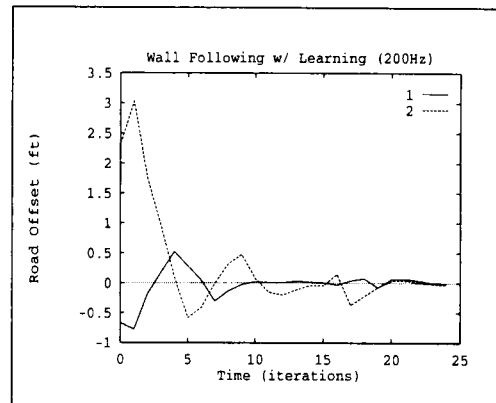


(d)

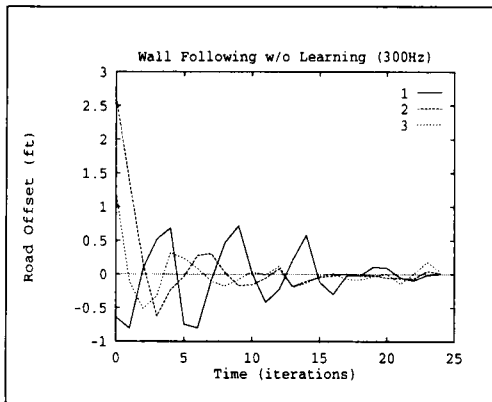
Figure 4.5: *The experimental results of wall following. The robot speed is set at 50Hz in (a) and (b), at 100Hz in (c) and (d).*



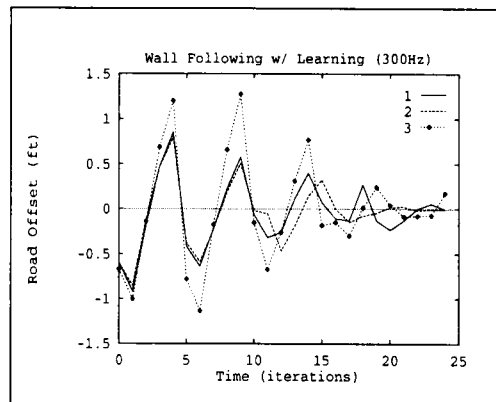
(a)



(b)



(c)



(d)

Figure 4.6: *The experimental results of wall following. The robot speed is set at 200Hz in (a) and (b), at 300Hz in (c) and (d).*

any error in the rules. However, the errors that cause the robot to run oscillatory result more from hardware imperfection and the unstructured environment than from the controller itself. The robot, which can scan only $+15.0$ degrees for the left wall (more negative orientation error resolution when hugging the left wall) and -15.0 degrees for the right wall, could produce bad results, such as the robot on path 2 going from -1.5 to 3.0 in Figure 4.5(a) and the path 3 going from -1.5 to around 2.0 in Figure 4.5(d). The robot sensor information may mislead the robot that the orientation error is still negative while it is already in the positive road offset zone when hugging the left wall. Therefore, the third rule in Table 4.1 is continuously misfired until a positive orientation error is sensed, and the second rule sends the robot back to the right track. The unstructured environment (door depth) could produce some sudden errors, too. The obvious errors at around 17 iteration in Figure 4.6(a) and (b) are due to the door depth. As we have mentioned before, when the ORNN learning takes place, the objective function's error constants may not be set too large. Otherwise, the controller may learn wildly (too fast), such as path 3 in Figure 4.6(d).

4.5 Hall Centering Behavior

4.5.1 The Control Scheme

As we have mentioned in Section 4.2, the hall centering problem is different from the wall following behavior mainly in the input data acquisition. The hall cen-

tering problem requires two side wall distances to determine the hall center point (see Figure 4.7). Road offset, orientation error, and robot speed are the three input parameters. The robot speed is included this time so that its performance can be compared to the wall following behavior, which does not consider the robot speed as an input factor.

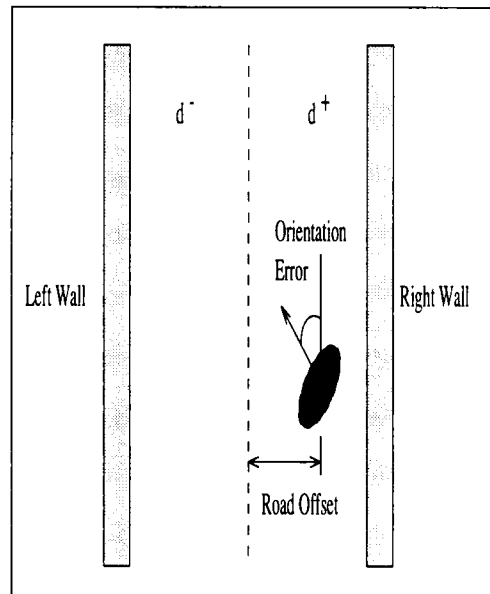


Figure 4.7: *The hall centering input parameter denotations.*

Figure 4.8 shows the membership functions. The right side of the hall center is considered the positive region. The robot heading direction should eventually align with both right and left walls. Similarly, if the robot is heading toward the d^- region, the orientation error is a negative value. Due to the hardware constraints of SMAR-T, only a $\pm 15^\circ$ of orientation error can be measured. SMAR-T speed can be set to a maximum of 300Hz, and the robot will still stay reliably functional. Thus, the unit of one in the speed membership function corresponds to 300Hz

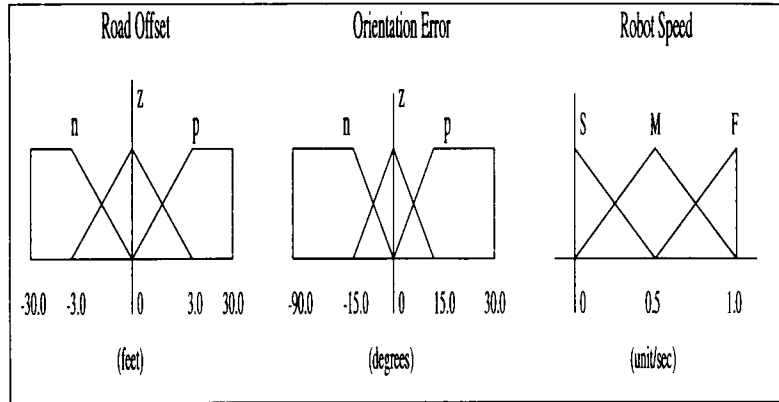


Figure 4.8: *Hall centering membership functions.*

(about 0.4 ft/sec).

The robot is supposed to lock onto the central lane. When the robot approaches the center slowly from the left side, the controller may not react as fast to turn the robot to its left as when the robot speed is fast. Nine parent rules are listed in Table 4.2. The weights of the RNN output fuzzy set are $\pm 30.0(\pm l)$, $\pm 20.0(\pm m)$, $\pm 5.0(\pm s)$, and $0.0(z)$ degrees. We trained RNN and ORNN with the same five hidden layer nodes with the parameters of $\eta = 0.3$, $\lambda = 0.9$, and the desired RMS error = 0.05. The RNN converged at 341 iterations, while the ORNN converged at 84 iterations. This again shows that the ORNN has a faster learning rate, since the network is typically smaller. Road offset and orientation error are the parameters included in the objective function. However, we placed more emphasis on correcting the road offset.

Table 4.2: *The nine hall centering parent rules.*

Parent Rules					
<i>Rule</i>	<i>Inputs</i>			<i>Output</i>	<i>Cond.</i>
	<i>Road Offset</i>	<i>Orient. Error</i>	<i>Speed</i>		
1	z	z	S	z	Stable
2	p	p	F	-l	Critical
3	p	p	S	-m	
4	p	n	F	s	
5	p	n	S	z	
6	n	p	F	-s	
7	n	p	S	z	
8	n	n	F	l	
9	n	n	S	m	

4.5.2 Experimental Validation

The experiments here are again conducted in the same environment as in Section 4.4.2. There are three sets of experiments with the robot speed of 50Hz, 100Hz, and 200Hz, respectively. Most of the runs approached around the hall center. In general, the controller shows better results with the help of the ORNN on-line learning capability. Figure 4.9 is SMAR-T moving toward the hall center.

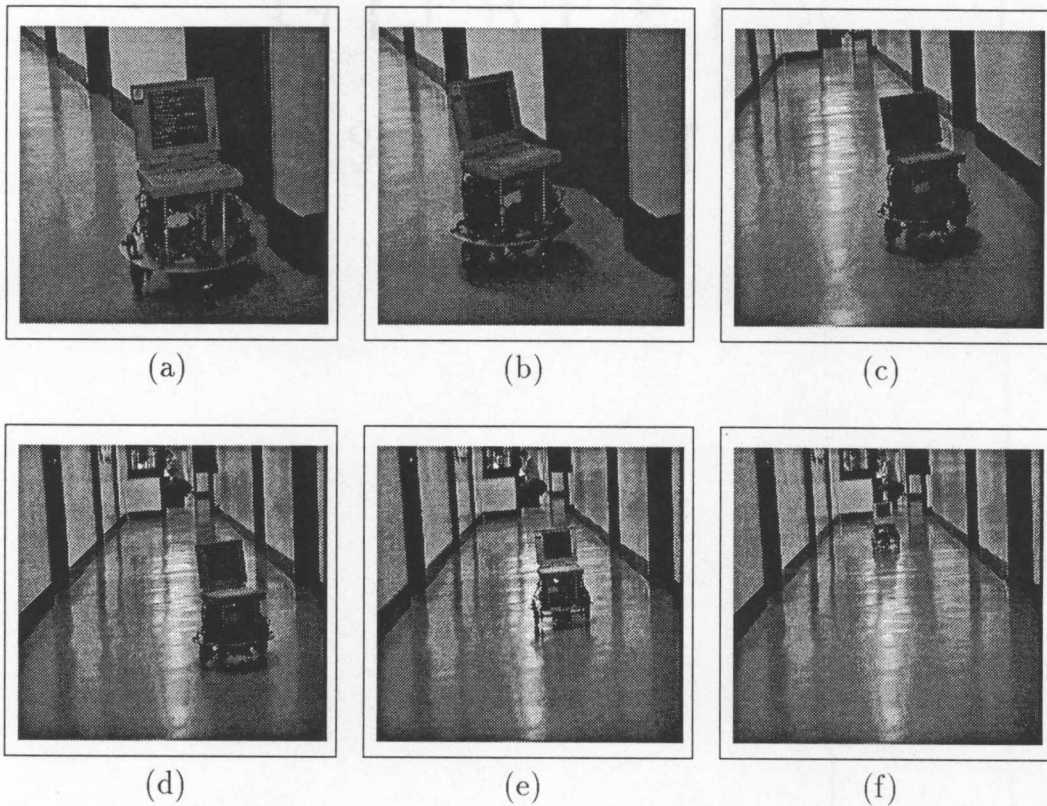
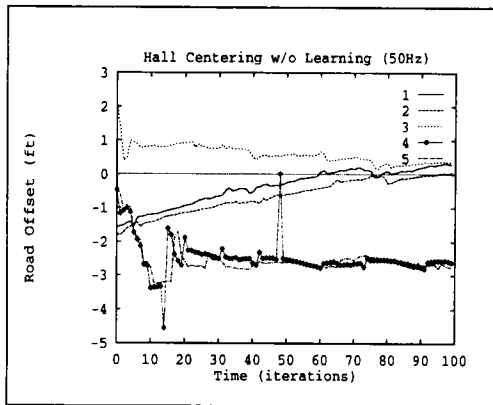
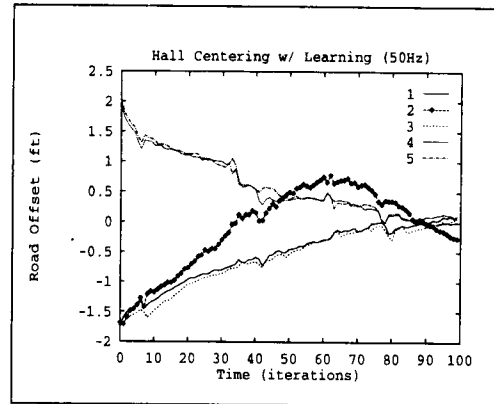


Figure 4.9: *SMAR-T moved to the hall center.*

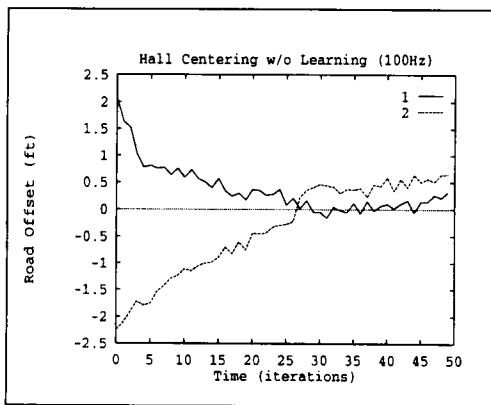
In Figure 4.10(a), the robot on paths 4 and 5 failed to return to the hall center. The principal reason could be that not enough weight is assigned to the output



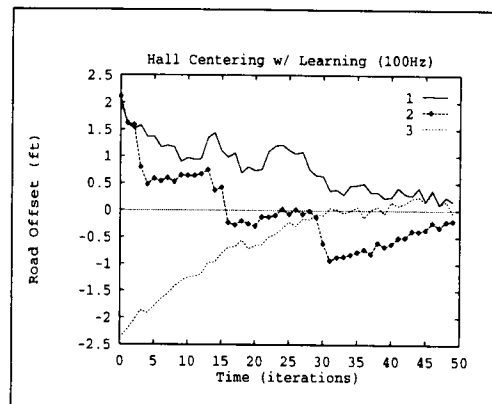
(a)



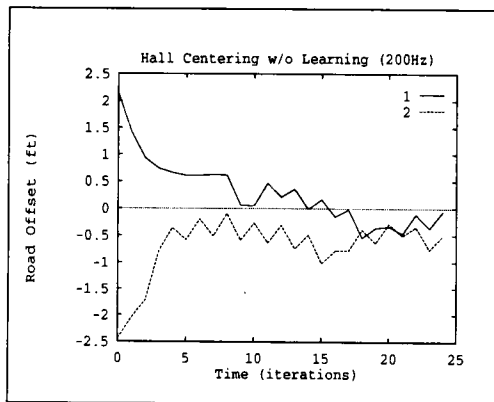
(b)



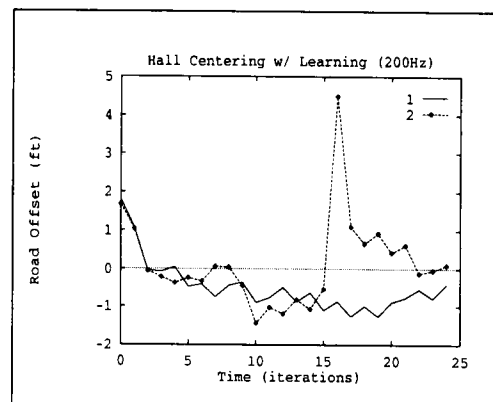
(c)



(d)



(e)



(f)

Figure 4.10: *The experimental results of hall centering.*

label $\pm m$ (± 20 degrees). Therefore, rule 9 in Table 4.2 is not able to turn the robot back to the center in that situation. However, with on-line learning, the robot almost locked onto the center path in all five runs (Figure 4.10[b]). Except for the fact that for the second path, the error constant of the objective function was set a little bit too high (-0.01), the path appears to have a higher overshoot. The same occurs for the learning part of Figure 4.10(d) path 2 and of (f) path 2. Some large number jumps in Figure 4.10(a) path 4 and (f) path 2 are caused by the sensor noises. Still, when the robot is operating at high speed with the limited sonar sensor scanning regions, the sensor uncertainty make the robot hard to lock onto the center (plots [e] and [f]).

4.6 Summary

The wall following and hall centering behaviors have been successfully implemented. Five rules were used to train the wall following NiF-T, while nine were used for the hall centering. Compared with previous works which require many rules, even thousands of training data and iterations, this can be considered an accomplishment. With learning capability, robot behaviors can be modified. If the parameters (especially the objective function constants) are properly set, learning will only enhance a controller's performance. When a behavior (wall following) is already under control, the learning does not have a negative effect on the results. However, bad results of the hall centering behavior were improved by the NiF-T.

CHAPTER 5

Controlling Multiple Mobile Robots Moving in Formation

5.1 Problem Statement and Specifications

For this convoying behavior, a leader is neither cross communicating with nor giving any explicit hint to the follower. The follower is supposed to use its sensing information to change its speed, moving direction, and orientation. The following behaviors are on the part of the follower:

1. Keep close to the specified distance away from the leader when moving.
2. Stop at the specified distance away from the leader when the leader stops.
3. Change speed smoothly and instantly, corresponding to the leader's speed change.
4. Move away with the behaviors described in 1, 2, and 3 if the leader is moving toward the follower (herding behavior).

Again, we would like to use this example to show that the proposed NiF-T has these special features:

- No detailed mathematical derivation.
- Qualitative description of the nonlinear system behavior (model-free).
- Continuous representation and human knowledge incorporation.
- Learning capability.
- High noise tolerance.

5.2 Review of Other Approaches

Today research interest on robotics has extended from single-robot systems to multiple-robot systems. Certainly, no matter how capable a single robot is, multiple-robot systems can economically and efficiently accomplish complex tasks that no single robot can accomplish. Instead of building a single powerful robot for each separate task, using several simple robots can be easier, cheaper, more flexible, and more fault tolerant. Cooperative behaviors can be carried out by a team of robots (usually mobile robots), either with different skills (referred to as heterogeneous robots) or with the same skills (referred to as homogeneous robots). The applications that have been worked on can be classified mainly into three types: cooperative transportation, cooperative sensing, and foraging [Par95]. In cooperative transportation, multiple mobile robots transport objects such as a box/furniture [RDJ95, MNS95] cooperatively. Cooperative sensing makes individual robots that are equipped with their own sensors recognize their own

environment and put their pieces of information together into a complete picture. Communication is a key design issue for multiple-robot systems. Different communication frameworks were discussed in [AHE91]. There are researchers proposing robot architecture such as [Nor93, Tri95], while others are putting their efforts toward solving geometric problems [KOAY95, LT95]. However, very few applications of cooperative robotics have been yet reported, and supporting theory is still in its formative stages [CFKM95].

Convoying behavior involves the use of multiple robots. Wang [Wan91] has simulated some navigation path-planning strategies for more than three mobile robots to move in formation. Sensing and communication for robot convoy were experimented in [DJMW95]. The work discussed in this chapter is focusing on the motion control using the proposed NiF-T. [MHT95] has implemented fuzzy control on this similar problem. As many as 36 rules were derived in that work, but no herding behavior can be performed, and the robot does not convoy at a specified distance. With the methods described here, fewer rules are required. In addition, without any communication and hint from the leader, the follower is convoying at the specified safe distance away from the leader.

5.3 Experimental Testbed

SMAR-T (Figure 4.1) and ELVIS (Figure 5.1) are the robots to be used to demonstrate convoying behavior. ELVIS is just an arbitrary wheeled mobile robot

to be utilized as a leader regardless of its other real-time processing capabilities. SMAR-T is the robot which has the NiF-T to coordinate its motion with ELVIS.

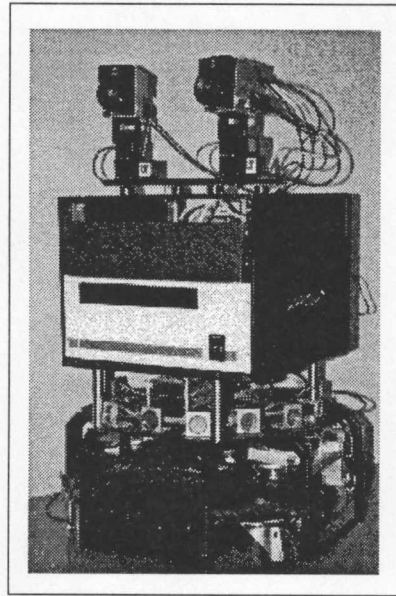


Figure 5.1: *Experimental Laboratory Vehicle Integrated with Sensors (ELVIS) is a leader in the robot convoy.*

5.4 The Control Scheme

While convoying, four basic parameters are constantly monitored. Without any communication from the leader, the follower has to measure its distance from the leader, know its own and the leader's speed, and detect the leader's orientation. Using sonar sensor, the distances can easily be obtained. Subtracting the two immediate distances gives us the relative speed of the two robots. Since the follower has the record of its own speed, adding that speed to the relative speed

results in the leader speed. The follower is always trying to align itself with the closest point to the leader. Although both the speed and the steering angle of the follower are being controlled continuously, we are currently more interested in designing the speed controller using the NiF-T.

Three parameters—following offset, follower speed, and leader speed—are the inputs of the controller. The follower is considered to be in the negative following offset region (d^-) when it is more than the specified distance from the leader (see Figure 5.2). When the robots go forward, the speed is positive; otherwise, the speed is negative. The constructed membership functions appear in Figure 5.3. SMAR-T speed can be set to a maximum of 300Hz and the robot will still stay reliably functional. Thus, the unit of one in the speed membership function corresponds to 300Hz (about 0.4 ft/sec).

The parent rule set developed in Table 5.1 is typically based on observation of daily driving behavior: the greater the distance between my car and the car ahead of me, and the faster that car is traveling, the harder I will tend to accelerate. I will also stop at a safe distance from the front car at a red light. If a car is backing toward me and passes the specified “safe” distance, that will intrude on my sense of security and make me backing up as well. Although backing up and the exact “safe” distance specification are not really necessary in our daily driving practice, they just show that the NiF-T can easily assimilate many interesting behaviors. The weights of the RNN output fuzzy set are -1.75(-l), 1.0(-m), 0.0(z), 1.0(m), and 1.5(l). We trained RNN and ORNN with the same five hidden layer nodes

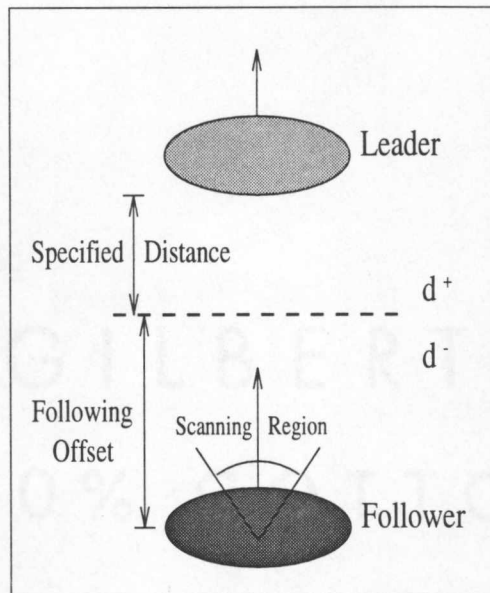


Figure 5.2: *The robot convoy schematic.*

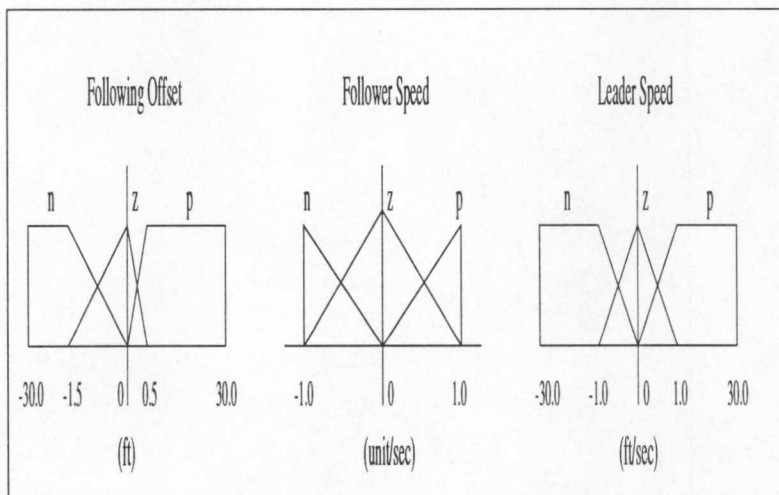


Figure 5.3: *The membership functions of robot convoy.*

Table 5.1: *The parent rules of robot convoy.*

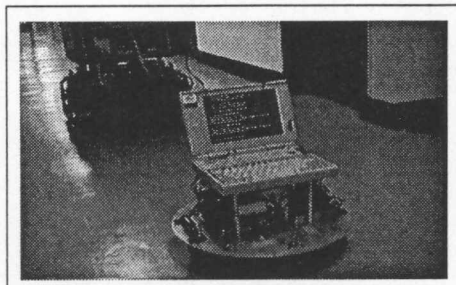
Parent Rules				
<i>Rule</i>	<i>Inputs</i>			<i>Cond.</i>
	<i>Following Offset</i>	<i>Follower Speed</i>	<i>Leader Speed</i>	
1	z	z	z	Stable
2	p	p	p	Critical
3	p	p	n	
4	p	n	p	
5	p	n	n	
6	n	p	p	
7	n	p	n	
8	n	n	p	
9	n	n	n	

with the parameters of $\eta = 0.3$, $\lambda = 0.9$, and the desired RMS error = 0.05. The RNN converged at 735 iterations, while the ORNN converged at 89 iterations. This again shows that the ORNN has faster learning rate. The following offset is only to minimize in the objective function.

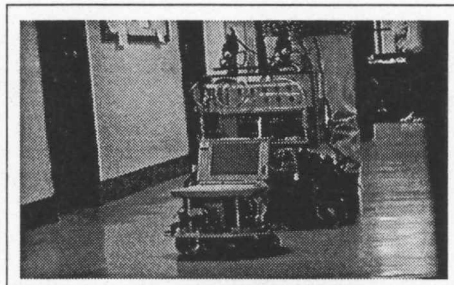
5.5 Experimental Validation

Figure 5.4 shows how SMAR-T convoyed ELVIS in a hallway. Two sets of experiments are discussed here. First, when the leader is stationary, the follower starts from any arbitrary initial position to reach the specified safe distance and stop. Secondly, as the leader keeps moving randomly, the follower tries to keep up with the change. Learning is unnecessary or undesirable in the second case because the leader's motion is irregular or unformatted. In both cases, the follower convoyed the leader closely and smoothly by varying its speed and changing its direction. When the leader stopped, the follower stopped very close to the specified safe distance. The follower backed up quickly when the leader moved backward toward it.

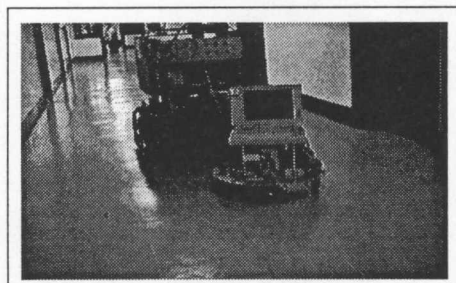
In Figure 5.5, the leader was stationary. The follower was initially placed in either the negative or positive following region. It stopped very close to the specified safe distance in a very short period of time, with or without learning. In plot (b) path 1, the following offset, which abruptly drops to -8 (ft), is the sensor noise.



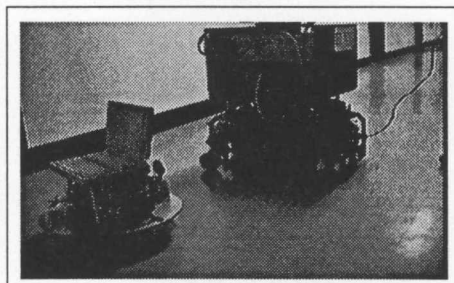
(a)



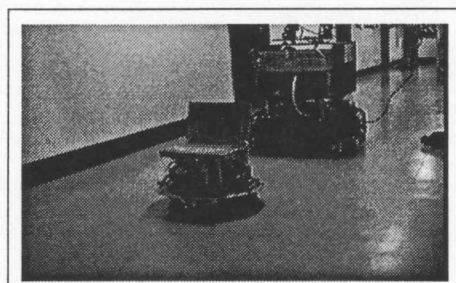
(b)



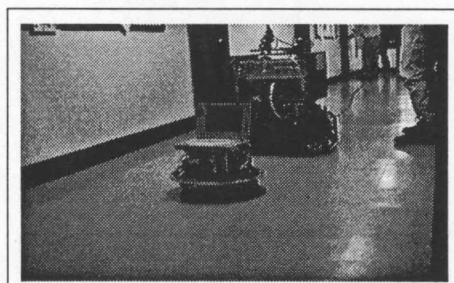
(c)



(d)



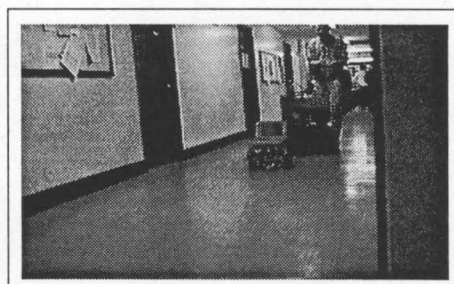
(e)



(f)



(g)



(h)

Figure 5.4: *SMAR-T* convoyed *ELVIS* in a hallway.

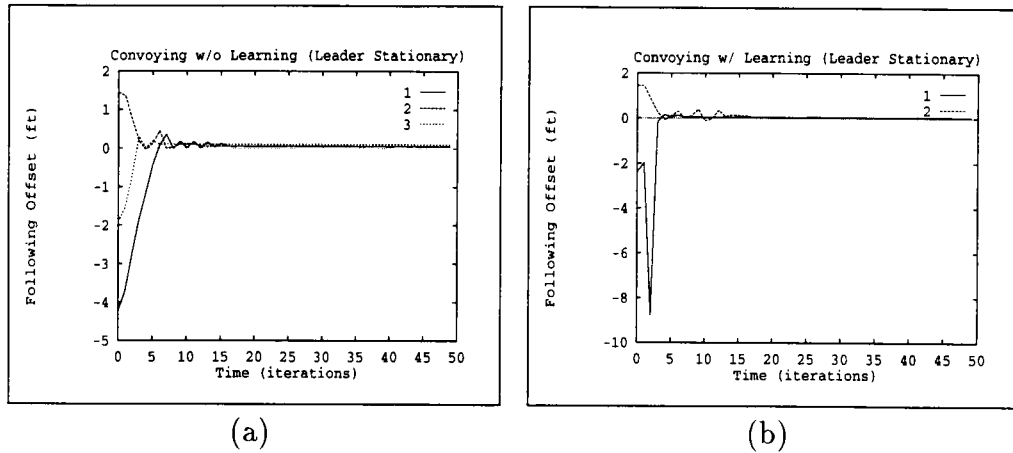
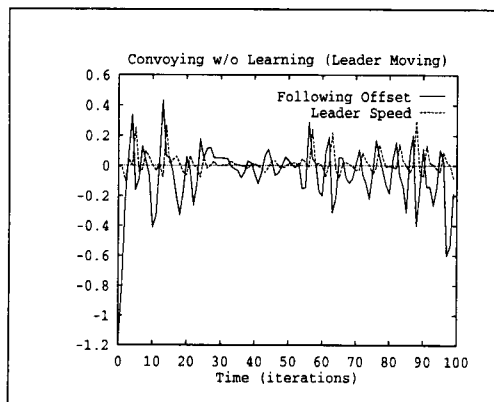
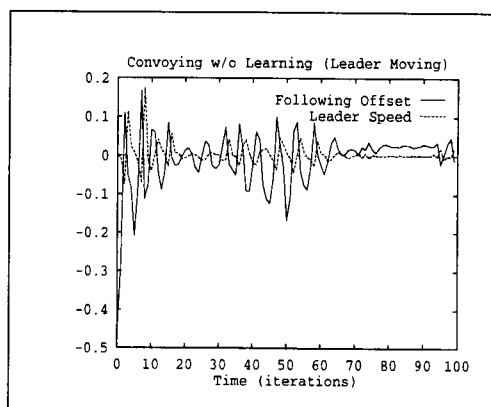


Figure 5.5: *The leader was stationary. The follower started from different initial positions, and it was able to stop at the specified “safe” distance in a very short time.*

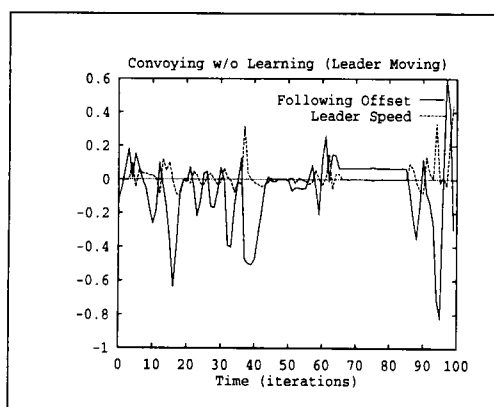
The follower, SMAR-T, cannot acquire its own speed directly from the hardware; therefore, the speed recording must be done in the software. However, SMAR-T does not response to any frequency that is set below 30.7 Hz. As a result, the calculated leader speed may not be exact. This is obvious in Figure 5.6(a) and (b). The leader actually was moving forward all the time; yet the plots show that the leader sometimes moved backward. The follower was still able to convoy closely with the leader although the sensors were noisy and uncertain. When the leader makes a sudden change in speed, the follower may lag or exceed the specified following safe distance for a while. Generally within 10 iterations, the follower catches up with the leader so long as the leader does not go faster than the maximum driving speed of SMAR-T (0.4 ft/sec in this case).



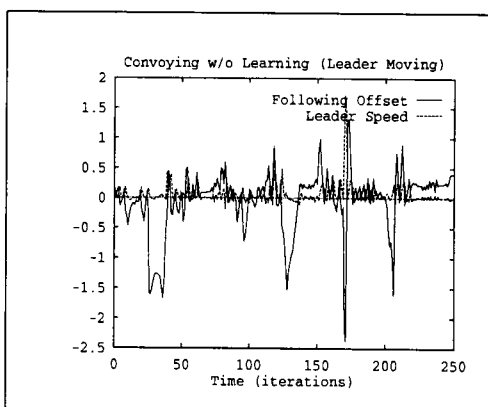
(a)



(b)



(c)



(d)

Figure 5.6: *The follower convoyed the moving leader. The leader kept changing its speed and direction. The follower was able to cope with the change and stop close to the specified safe distance when the leader stopped.*

5.6 Summary

In this chapter, the NiF-T has been applied to control the follower robot speed. With fewer rules developed than [MHT95]'s work, the convoying here shows more interesting behaviors. The follower stopped at the specified distance in a short length of time. It followed the leader closely with smooth and instant speed change. When the leader moved backward toward the follower, the follower instantly backed up as well.

CHAPTER 6

Summary and Conclusions

6.1 Summary

The main focus of this thesis is on the development of intelligent robotic systems which are capable of learning a wide array of intelligent behaviors. The framework utilized in this development is that of nonclassical control architectures to realize real-time and primarily reactive system performance.

Inspired by the advantages and disadvantages of both fuzzy logic and neural networks, a hybrid fuzzy-neural control architecture is proposed in this thesis. Fuzzy logic allows continuous input representation and direct human expert knowledge incorporation. However, its inability to learn makes the rule generation and membership function tuning a drudging task. Neural networks can learn, yet reliable sample data are difficult to obtain. Additionally, usually a large number of required training data would necessitate thousands of iterations to bring the network to convergence. After fusing these two technologies, the proposed NiF-T has the following important capabilities:

- Integrated sensing, control, and activator modules.
- Real-time performance requirement.

- Ability to successfully handle noisy sensory signals.
- Reactive controller design which effectively captures high-level, linguistically based human expertise in a set of fuzzy rules.
- Training the neural networks directly with fuzzy rules instead of numerical sample data. The linguistic expert rules are easier to obtain and more reliable.
- Learning capability to interpolate new set of rules for robust and smooth operation of the system. New rules are derived of parent rules as specified by a human so that a minimum of rules are extracted from the expert.
- Learning capability to refine the performance of the integrated intelligent systems.
- General applicability of the architecture. That is, the same overall system level architecture will be able to support realization of a wide range of robotic systems.

The NiF-T includes three major parts: FMF, RNN, and ORNN. FMF allow continuous input representation that also enables expert knowledge to be incorporated directly. With the interpolating nature of neural networks, RNN is exploited to interpolate missing rules so that a minimal number of rules need to be extracted from the human expert. A smaller network, ORNN, is utilized to refine the control action. Since the network is smaller, its learning rate is faster

than the RNN's. In short, the whole control process can be summarized as this: FMF are utilized to fuzzify input parameters; RNN interpolates the fuzzy rule set; after defuzzification, RNN's outputs are used to train a smaller size of neural network, ORNN; the weights of ORNN can be adjusted on-line to fine-tune the controller.

When constructing the membership functions, observation of the physical constraints always helps. In order not to develop the whole set of fuzzy rules, some parent rules are generated using the idea inspired from solving differential equations—*boundary conditions*. After the RNN is successfully trained, control surfaces are plotted to test the model. Generally, good control surfaces are monotonic without any obvious spikes or fold-around. For the ORNN on-line training, the objective function's constants cannot be set too large. They require a full dynamic of trial and error. In order to prevent the controller from overlearning, a counter needs to be set up to record the times the goal is reached. When the counter drops to zero, the controller starts to learn again.

Ball Balancing Beam, single mobile robot navigation, and multiple robot convoy have been successfully implemented in Chapters 3, 4, and 5, respectively. The NiF-T balanced different sizes and masses of balls with only nine rules. Despite the fact that ORNN was not trained symmetrically, the on-line learning corrected the control action back to its original goal. The balls almost rested around the beam center in many attempts. Nevertheless, the noisy signals and the hardware imperfection disturbed the system. Thus, the balls gained speed and rolled again.

Only five and nine rules were again used to navigate a mobile robot to perform wall following and hall centering behaviors. It took only a few hundred iterations to train these control networks off-line. With slower robot speed, the acquired sensor data were more reliable, and so the robot could lock onto the specified path in most cases. The on-line learning again navigated the robot toward the goal for the hall centering behavior when the output was not trained with enough weight in the case of slow robot speed (Figure 4.10[a]). Many interesting behaviors were carried out in the robot convoy experiments. With only nine rules, the follower convoyed the leader closely and smoothly by varying its speed and changing its direction. When the leader stopped, the follower stopped very close to the specified safe distance. The follower backed up quickly when the leader moved backward toward it (herding behavior).

6.2 Unique Contributions

The fuzzy-neural controller inherits the strengths of both fuzzy logic and neural networks, both of which are model-free. Therefore, there is not even a single equation derived to control all of the discussed behaviors. With the help of fuzzy logic, numerical sample data are no longer needed for the network training. Instead, the reliable expert rules are used to train the networks. In addition, membership functions provide continuous input representations. They make the controller less sensitive to slight variations in the physical parameters and control goal. Like-

wise, with the help of neural networks, fewer fuzzy rules are to be extracted from the human experts and the control action can be fine-tuned on-line. The NiF-T can also endure highly noisy signals. After all, fusion of these two technologies seems to produce better results in control. The following list highlights the unique contributions of this thesis:

- A new architecture NiF-T is proposed for developing intelligent robotic systems which combines strengths of fuzzy logic and neural networks.
- Fuzzy logic framework allows realization of powerful robotic controllers without requiring analytical solution of nonlinear models and it provides a very effective means for handling sensor noise and uncertainties.
- A neural network based approach for deriving the complete set of fuzzy rules utilizes only a small number of rules specified by the designer.
- Neural networks are trained with expert rules rather than numerical data.
- Learning rate improvement is accomplished using a small output-refinement network.
- Practical and robust approaches for realizing a wide range of diverse intelligent behaviors—BBB, wall following, hall centering, and coordinated group motion—are presented.
- The efficiency and generality of the integrated architecture are evaluated by extensive experimentation with real-world robotic systems.

6.3 Future Direction

The interconnections of RNN and ORNN are currently fully connected. In the future, we may want to connect them partially so that learning rate will be further improved and hardware implementation cost (onto a chip) will be reduced. In addition, the optimum constant of the objective function is difficult to obtain. Therefore, a better solution may be required to minimize the controller errors.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [AHE91] H. Asama, M.K. Habib, and I. Endo. Functional distribution among multiple mobile robots in an autonomous and decentralized robot system. In *Proceedings of the 1991 IEEE International Conference on Robots and Automation*, pages 1921–1926, Sacramento, California, April 1991.
- [AP93] P.J. Antsaklis and K.M. Passino, editors. *An Introduction To Intelligent and Autonomous Control*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.
- [Bez93a] J. Bezdek. Fuzzy models—what are they, and why? *IEEE Trans. on Fuzzy Systems*, 1(1), February 1993.
- [Bez93b] J.C. Bezdek. Fuzzy logic tutorial. Technical report, University of West Florida, Pensacola, 1993.
- [BI93] F. Bouslama and A. Ichikawa. Application of neural networks to fuzzy control. *Neural Networks*, 6:791–799, 1993.
- [BK92] H.R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3(5):724–740, September 1992.
- [CC94] C.L. Chen and W.C. Chen. Fuzzy controller design by using neural network techniques. *IEEE Trans. on Fuzzy Systems*, 2, August 1994.
- [CFKM95] Y.U. Cao, A.S. Fukunaga, A.B. Kahng, and F. Meng. Cooperative mobile robotics: Antecedents and directions. In *Proceedings of the 1995 IEEE/RSJ International Conference on Robots and Systems*, volume 1, pages 226–234, Pittsburgh, Pennsylvania, August 1995.
- [CK95] S.-B. Cho and J.H. Kim. Multiple network fusion using fuzzy logic. *IEEE Transactions on Neural Networks*, 6(2):497–501, March 1995.
- [CR95] M. Chiaberge and L.M. Reyneri. Cintia: A neuro-fuzzy real-time controller for low-power embedded systems. *IEEE Micro*, pages 40–47, June 1995.
- [CT93] C. Chen and M.M. Trivedi. Reactive locomotion control of a tracked mobile manipulator. In *Proceedings of the IEEE/RSJ Intelligent Robotic Systems (IROS'93) Conference*, pages 1349–1356, Yokohama, Japan, July 1993.

- [CXL92] R.M.H. Cheng, J.W. Xiao, and S. LeQuoc. Neuromorphic controller for AGV steering. In *Proceedings of IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.
- [dar88] *DARPA Neural Network Study*. AFCEA International Press, 1988.
- [DJMW95] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Experiments in sensing and communication for robot convoy. In *Proceedings of the 1995 IEEE/RSJ International Conference on Robots and Systems*, volume 2, pages 268–273, Pittsburgh, Pennsylvania, August 1995.
- [FB94] J.A. Franklin and H. Benbrahim. Ball balancer. *IEEE Control System*, pages 21–24, February 1994.
- [FIAY95] T. Fukuda, S. Ito, F. Arai, and Y. Yokoyama. Navigation system based on ceiling landmark recognition for autonomous mobile robot. In *Proceedings of the 1995 IEEE/RSJ International Conference on Robots and Systems*, volume 2, pages 150–155, Pittsburgh, Pennsylvania, August 1995.
- [FMMM95] M. Funabashi, A. Maeda, Y. Morooka, and K. Mori. Fuzzy and neural hybrid expert systems: Synergetic ai. *IEEE Expert*, 10(5):32–40, August 1995.
- [Gal93] S.I. Gallant. *Neural Network Learning and Expert Systems*. A Bradford Book, The MIT Press, Cambridge, Massachusetts, 1993.
- [GKG95] R. Glasins, A. Komoda, and S.C.A.M. Gielen. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1):125–133, 1995.
- [HFU92] S. Horikawa, T. Furuhashi, and Y. Uchikawa. On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Transactions on Neural Networks*, 3(5):801–806, September 1992.
- [HG94] C.M. Higgins and R.M. Goodman. Fuzzy rule-based networks for control. *IEEE Trans. on Fuzzy Systems*, 2, February 1994.
- [Hol94] M.B. Holder. Design and implementation of an integrated multi-processor mobile robot for experimental research in cooperative robotics. Master's thesis, The University of Tennessee, Knoxville, December 1994.
- [HP94] K. Hirota and W. Pedrycz. Or/and neuron in modeling fuzzy set connectives. *IEEE Trans. on Fuzzy Systems*, 2(2), May 1994.

- [IK95] M. Iskarous and K. Kawamura. Intelligent control using a neuro-fuzzy network. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 350–355, Pittsburgh, Pennsylvania, August 1995.
- [IT93] H. Ishibuchi and H. Tanaka. Neural networks that learn from fuzzy if-then rules. *IEEE Trans. on Fuzzy Systems*, 1, May 1993.
- [JPT93] T.M. Jochem, D.A. Poonerlean, and C.E. Thorpe. Maniac: A next generation neurally based autonomous road follower. In *International Conference on Intelligent Autonomous Systems*, Pittsburgh, Pennsylvania, 1993.
- [JROC⁺95] J.A. Janet, R.G.-Osuna, T.A. Chase, et al. Global self-localization for autonomous mobile robots using self-organization kohonen neural networks. In *Proceedings of the 1995 IEEE/RSJ International Conference on Robots and Systems*, volume 3, pages 504–509, Pittsburgh, Pennsylvania, August 1995.
- [KC94] H.K. Kwan and Y. Cai. A fuzzy neural network and its application to pattern recognition. *IEEE Trans. on Fuzzy Systems*, 2, August 1994.
- [KI93] B. Kosko and S. Isaku. Fuzzy logic. *Scientific American*, July 1993.
- [KL93] A. Kandel and G. Langholz, editors. *Fuzzy Control Systems*. CRC Press Inc., 1993.
- [Kni90] K. Knight. Connectionist ideas and algorithms. *Communications of the ACM*, 33(11):59–74, November 1990.
- [KOAY95] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. An algorithm of dividing a work area to multiple mobile robots. In *Proceedings of the 1995 IEEE/RSJ International Conference on Robots and Systems*, volume 2, pages 286–291, Pittsburgh, Pennsylvania, August 1995.
- [Kos91] B. Kosko. *Neural Networks and Fuzzy Systems: A Dynamical Approach to Machine Intelligence*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.
- [LHGK92] J.M. Lubin, E.C. Huber, S.A. Gilbert, and A.L. Kornhauser. *Analysis of a Neural Network Lateral Controller for an Autonomous Road Vehicle*, pages 23–44. Society of Automotive Engineers, Inc., Warrendale, Pennsylvania, August 1992.

- [LT95] C.-F. Lin and W.-H. Tsai. Optimal assignment of robot tasks with precedence for multi-robot coordination by disjunctive graphs and state-space search. *Journal of Robotic System*, 12(4):219–236, 1995.
- [LW92] J.J. Leonard and H.F.D. Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic Publisher, Norwell, Massachusetts, 1992.
- [LW94] P.-S. Lee and L.-L. Wang. Collision avoidance by fuzzy logic control for automated guided vehicle navigation. *Journal of Robotic Systems*, 11(8):743–760, 1994.
- [MHT95] S.B. Marapane, M.B. Holder, and M.M. Trivedi. Motion control of a mobile robot traveling in a convoy using fuzzy logic. In *Proceedings of SPIE in Applications of Fuzzy Logic Technology II*, volume 2493, pages 187–198, Orlando, Florida, April 1995.
- [MK92] M. Meng and A.C. Kak. Fast vision-guided mobile robot navigation using neural networks. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, pages 111–116, August 1992.
- [MK93] M. Meng and A.C. Kak. Neuro-nav: A neural network based architecture for vision-guided mobile robot navigation using non-metrical models of the environment. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 750–757, Atlanta, Georgia, May 1993.
- [MNS95] M.J. Mataric, M. Nilsson, and K.T. Simsarian. Cooperative multi-robot box-pushing. In *Proceedings of the 1995 IEEE/RSJ International Conference on Robots and Systems*, volume 3, pages 556–561, Pittsburgh, Pennsylvania, August 1995.
- [neu93] *Neural Computing: A Technology Handbook for Professional II/Plus and NeuralWorks Explorer*. Neural Ware, Inc., 1993.
- [Nor93] F.R. Noreils. Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The International Journal of Robotics Research*, 12(1):79–98, February 1993.
- [NSA90] S. Nagata, M. Sekiguchi, and K. Asakawa. Mobile robot control by a structured hierarchical neural network. *Control Systems*, 10(3):69–76, April 1990.
- [NT95] K.C. Ng and M.M. Trivedi. Fuzzy logic controller and real-time implementation of a ball balancing beam. In *Applications of Fuzzy Logic Technology II*, pages 261–272, Orlando, Florida, April 1995.

- [Par95] L.E. Parker. The effect of action recognition and robot awareness in cooperative robotic teams. In *Proceedings of the 1995 IEEE/RSJ International Conference on Robots and Systems*, volume 1, pages 212–219, Pittsburgh, Pennsylvania, August 1995.
- [PR93] W. Pedrycz and A.F. Rocha. Fuzzy-set based models of neurons and knowledge-based networks. *IEEE Trans. on Fuzzy Systems*, 1(4), November 1993.
- [RDJ95] D. Rus, B. Donald, and J. Jennings. Moving furniture with teams of autonomous robots. In *Proceedings of the 1995 IEEE/RSJ International Conference on Robots and Systems*, volume 1, pages 235–242, Pittsburgh, Pennsylvania, August 1995.
- [Rei90] M. Reinhardt. *Neural Networks—An Introduction*. Springer-Verlag, 1990.
- [SH94] B. Shahian and M. Hassul. *Control System Design Using Matlab*, chapter Ball-on-Beam Balancer, pages 465–476. Prentice Hall, Englewood Cliffs, New Jersey, 1994.
- [SHP95] H. Surmann, J. Huser, and L. Peters. A fuzzy system for indoor mobile robot navigation. In *FUZZ-IEEE 95*, Yokohama, Japan, 1995.
- [Sug85] M. Sugeno, editor. *Industrial Applications of Fuzzy Control*. North-Holland, 1985.
- [TF93] J. Tani and N. Fukumura. Learning goal-oriented navigation as attractor dynamics for a sensory motor system. In *Proceedings of 1993 International Joint Conference on NN*, pages 1747–1752, 1993.
- [TG86] M.M. Trivedi and J.F. Gilmore. Applications of artificial intelligence. *Optical Engineering: Applications of Artificial Intelligence*, 25(3):331–332, March 1986.
- [Tha93] S. Thaler. Fuzzy rule generation based on a neural network approach. *Electronic Engineering*, pages 43–50, July 1993.
- [Tri93] M.M. Trivedi. The abc of intelligent robotic systems. In *Proceedings of Automatique et Robotique, DRET*, Paris, France, January 1993.
- [Tri94] M.M. Trivedi. *Encyclopedia of Science and Technology*, volume Yearbook, chapter Intelligent Robotic Systems, pages 226–229. McGraw-Hill, New York, NY, 1994.
- [Tri95] M.M. Trivedi. Intelligent robots: Control and cooperation. In *Proceedings of SPIE in Applications of Fuzzy Logic Technology II*, volume 2493, Orlando, Florida, April 1995.

- [Uhr95] R.E. Uhrig. *Neural Networks and Fuzzy Expert Systems*. In preparation, University of Tennessee, Knoxville, 1995.
- [Wan91] P.K.C. Wang. Navigation strategies for multiple autonomous mobile robots moving in formation. *Journal of Robotic System*, 8(2):177–195, 1991.
- [Wel79] P.E. Wellstead. *Introduction to Physical System Modeling*, chapter The Ball and Beam Problem, pages 252–259. Academic Press, London; New York, 1979.
- [Win84] P.H. Winston. *Artificial Intelligence*. Addison-Wesley Publishing Company, 2 edition, 1984.
- [Yag94] R.R. Yager. Modeling and formulating fuzzy knowledge bases using neural networks. *Neural Networks*, 7(8):1273–1283, 1994.
- [Yam93] T. Yamakawa. A fuzzy inference engine in nonlinear analog mode and its application to a fuzzy logic control. *IEEE Transactions on Neural Networks*, 4(3):490–522, May 1993.
- [YS94] G. Yu and I.K. Sethi. Road following using incremental learning. Technical report, Computer Science Dept., Wayne State University, Detroit, Michigan, 1994.
- [YZ94] R.R. Yager and L.A. Zadeh, editors. *Fuzzy Sets, Neural Networks, and Soft Computing*. Van-Nostrand Reinhold, New York, 1994.
- [ZP94] U.R. Zimmer and E.V. Puttkamer. Realtime learning on an autonomous mobile robot with neural networks. In *Euromicro 94-Realtime Workshop*, pages 15–17, Vaesteraas, Sweden, June 1994.

APPENDICES

APPENDIX A

Fuzzy Logic Control Process

The fuzzy controller design cycle is first to construct membership functions to fuzzify input parameters, then to pass the fuzzified variables through linguistic control rules to reason approximately. An output membership function will be used to defuzzify control output. The final outcome will be sent to process. This design process is illustrated in Figure A.1 [Bez93b].

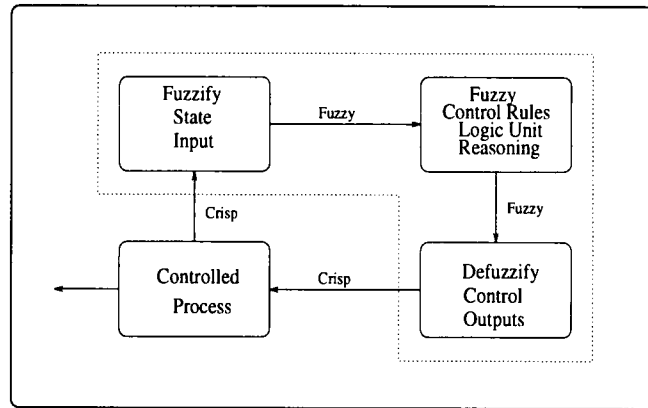


Figure A.1: *The schematic of the fuzzy logic controller. Input parameters are fuzzified to pass through the fuzzy control rules logic unit. After reasoning, control output is defuzzified to send to process. New parameter values are obtained and sent through the same process again.*

We use the *minimum conjunction* for the fuzzifying process, and then we choose

the *maximum* for each fired output label. Eventually, we find the *union area centroid* to defuzzify the numbers [Bez93b]. The following is a simple illustration of the process.

A.1 A BBB Example

After fuzzifying the state inputs, we have the fuzzy set values from the membership functions (see Figure A.2):

Theta (0.4 z & 0.6 n)

Theta-dot (0.3 z & 0.7 p)

Position (0.9 z & 0.1 n)

Velocity (0.8 z & 0.2 p)

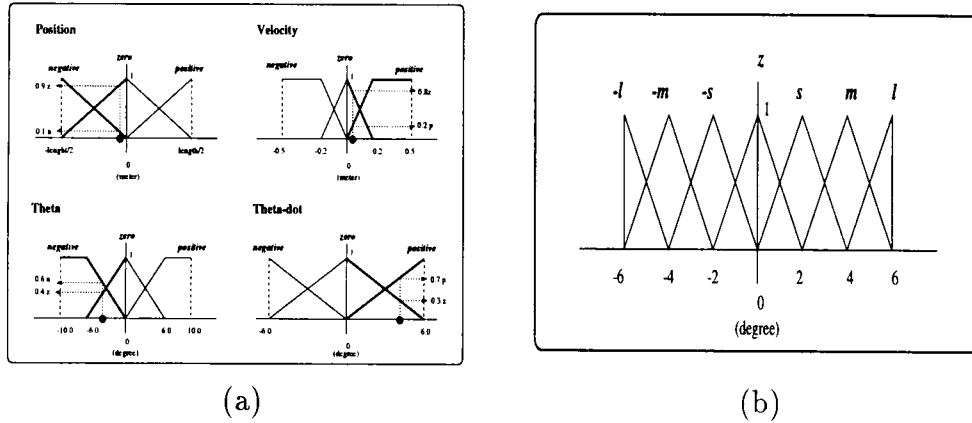


Figure A.2: (a) *Membership functions of the input parameters.* (b) *Output membership function.* Observation of physical constraints is used to construct these membership functions. The symbol representations are listed as $zero(z)$, $positive(p)$, $negative(n)$, $small(\pm s)$, $medium(\pm m)$, and $large(\pm l)$.

There are two fuzzy set values for each input parameter (there may be one to three, depending on where the membership function intercepts); however, the sum of the values for each parameter is always one. These values, which cross connect with one another, fire a total of 16 rules if there is no missing rule. All of the fired rules are listed below, and the reading sequence is theta, theta-dot, position, velocity, and output.

Minimum Conjunction on the Fired Rules

Rule 1. If (0.4 z) and (0.3 z) and (0.9 z) and (0.8 z) then **(0.3 z)**

Rule 2. If (0.4 z) and (0.3 z) and (0.9 z) and (0.2 p) then **(0.2 m)**

Rule 3. If (0.4 z) and (0.3 z) and (0.1 n) and (0.8 z) then **(0.1 -s)**

Rule 4. If (0.4 z) and (0.7 p) and (0.9 z) and (0.8 z) then **(0.4 z)**

Rule 5. If (0.6 n) and (0.3 z) and (0.9 z) and (0.8 z) then **(0.3 m)**

Rule 6. If (0.4 z) and (0.3 z) and (0.1 n) and (0.2 p) then **(0.1 s)**

Rule 7. If (0.6 n) and (0.7 p) and (0.9 z) and (0.8 z) then **(0.6 m)**

Rule 8. If (0.6 n) and (0.3 z) and (0.1 n) and (0.8 z) then **(0.1 s)**

Rule 9. If (0.6 n) and (0.3 z) and (0.9 z) and (0.2 p) then **(0.2 l)**

Rule10. If (0.4 z) and (0.7 p) and (0.1 n) and (0.8 z) then **(0.1 -s)**

Rule11. If (0.4 z) and (0.7 p) and (0.9 z) and (0.2 p) then **(0.2 m)**

Rule12. If (0.4 z) and (0.7 p) and (0.1 n) and (0.2 p) then **(0.1 s)**

Rule13. If (0.6 n) and (0.7 p) and (0.1 n) and (0.8 z) then **(0.1 s)**

Rule14. If (0.6 n) and (0.7 p) and (0.9 z) and (0.2 p) then **(0.2 l)**

Rule15. If (0.6 n) and (0.3 z) and (0.1 n) and (0.2 p) then **(0.1 l)**

Rule16. If (0.6 n) and (0.7 p) and (0.1 n) and (0.2 p) then (0.1 l)

For each similar fired output label, we find the maximum one and union it with the others. Selecting the maximum for each label yields the union (level, label)

(0.1 -s), (0.4 z), (0.1 s), (0.6 m), and (0.2 l)

out of the 16 fired rules after minimum conjunction.

Every fired label is clipped at its corresponding level. We find the trapezoidal area for each dominant rule and take the union of clipped areas. Finally, computing the horizontal coordinate of the union area centroid gives us the crisp controller output. The diagram in Figure A.3 below shows the procedure of defuzzification.

For this example, a ball at first is located a little left of center with slow speed to the right. The beam has been tilted counterclockwise (tilting clockwise is considered negative), but it is still in right declining position, which will accelerate the ball to the right side. So the crisp controller output of approximately (without calculation) three degrees, which tilts the beam more counterclockwise, should be able to stop the ball from accelerating to the right.

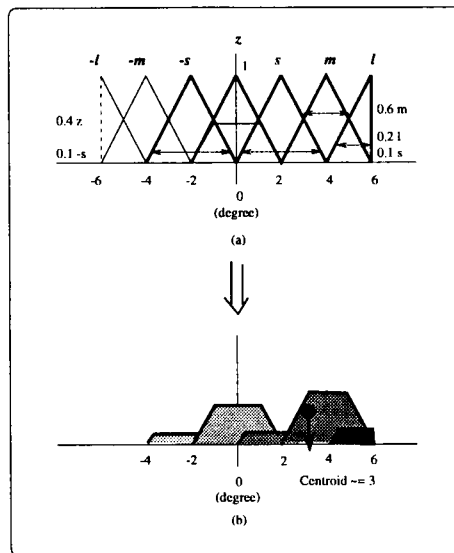


Figure A.3: The procedure of defuzzification. (a) Fired label clipped at its corresponding level. (b) Find their trapezoidal areas and compute the horizontal coordinate of the union area centroid.

APPENDIX B

Experimental Validation of the BBB FLC Behavior

There are five experiments altogether: (1) small ball alone, (2) medium ball alone, (3) large ball alone, (4) small and medium balls, (5) medium and large balls, all discussed in this section. We use different balls to justify how well the fuzzy controller can respond to the change of ball mass. In all of these runs, the fuzzy controller succeeded in balancing all of the ball. When ball (1) or (2) was used, the ball oscillated back and forth symmetrically around the center of the beam; some improvement is shown for (3). The balls oscillated less around the center after a few hundred iterations in the case of (4). The best results were obtained when both medium and large balls were rolling on the beam: the balls balanced around the center in less than 35 seconds.

The performances of the fuzzy logic controller are shown in Figure B.1. Different sizes of balls—small, medium, or large—roll on the beam individually or with other ball. The balls were oscillating back and forth almost symmetrically around the center of the beam in Figure B.1(a) and (b). There is no indication that the system will ever come to rest. A larger ball in Figure B.1(c) shows some improvement. The system happens to have less oscillation during a certain time. In Figure B.1(d), the balls did slow down at certain moments. Yet they will not

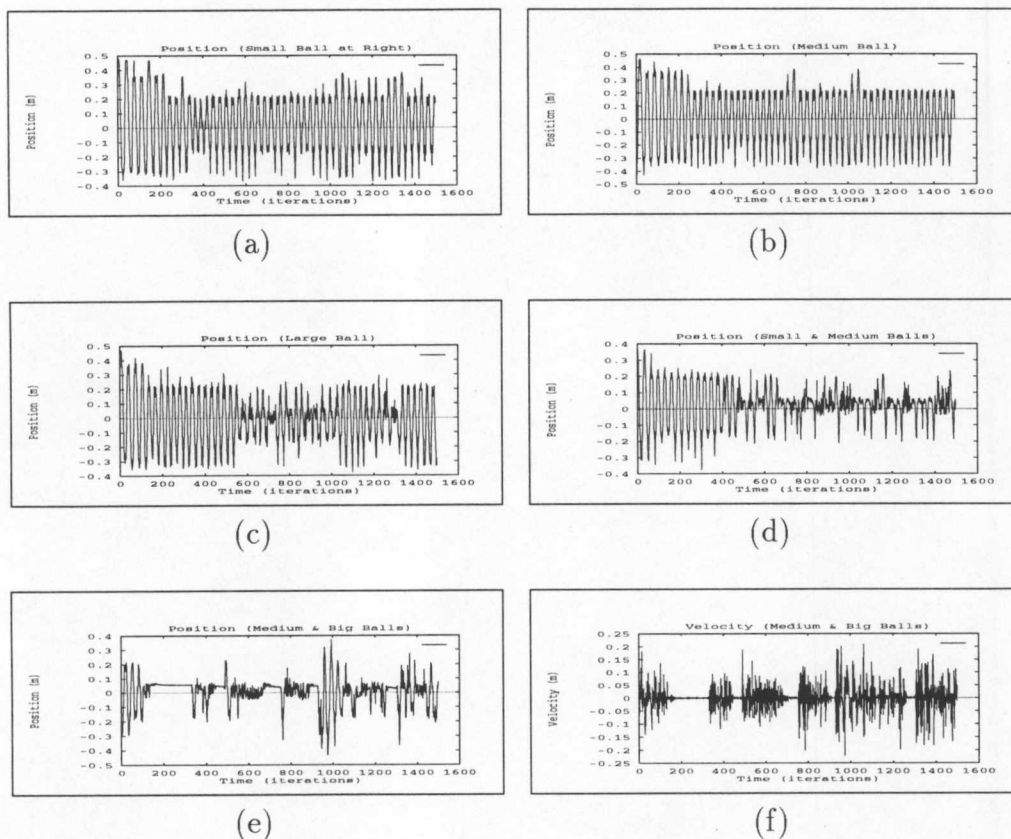


Figure B.1: (a) A small ball was oscillating back and forth symmetrically around the center. (b) The results of the medium ball are close to the small ball's. (c) Some improvement is shown for the large ball. The system happened to oscillate less after a certain time, and then it oscillated again. It is due to the imperfection of the hardware. The ball with less mass is easy to accelerate toward the down-inclined side with a small initial force. (d) Both small and medium balls were rolling together around the center after a few hundred iterations. (e) Both medium and large balls became completely steady after around 160 iterations. The system was disturbed again thereafter many times. It was still able to go stable after awhile. (f) Shows that the balls in (e) really stopped.

completely rest at the center due to the imperfection of the hardware construction. A theta of ± 0.01 degree can cause the light balls to gain acceleration gradually to speed off one end of the beam. Nevertheless, when two balls roll together on the beam, the system becomes more and more stable. It is especially obvious that the balls completely stopped near the beam center after about 160 iterations, which took about 32 seconds, in Figure B.1(e) and (f). The system was disturbed again thereafter many times, and it was able to go steady after awhile.

From the controller's performances, the system tends to perform better when the ball mass becomes heavier or when more balls are rolling together. There are three explanations for these results: first, better contact; second, less acceleration; third, the dynamic range of the activator force. Better contact to the sensory wires when the ball is heavier helps the system have more reliable position feedback. When two balls are rolling together, if one ball is jumping, the other is still in touch. This further secures the sensory system. The acceleration is inversely proportional to the ball mass. With the same motor torque, heavier balls accelerate more slowly. In addition, there is more friction when two balls are rolling together. One ball counteracts the other's motion when one changes direction from rolling together in the same direction. The results in Figure B.1(e), giving the best performances, make sense under these explanations, and the hardware imperfection is overcome by the large ball mass and the friction between two balls. In other words, this particular dynamic range of the activator force is suitable for the case in (e). The performances in the case of (a) to (d) can be improved by reducing

the weights of the output membership function. Therefore, the ball mass does affect the performances of this fuzzy logic controller.

APPENDIX C

Back Propagation Algorithm

Multilayered neural network can compute almost anything. The most common widely used neural network algorithm today is Back Propagation. Back Propagation algorithm has a strong mathematics foundation. Although it does not have convergence theorem to associate with, it rarely slips into the local minima; in other word, the network converges most of the time.

This network in Figure C.1 has one input layer, hidden layer, and one output layer. The input layer and every hidden layer has one thresholding unit or bias. There are weights connected from every node in one layer to every node in another layer; or they are fully connected. In every node, there is an activation function, that is, usually a sigmoid function.

C.1 Training Procedure

The following is a brief procedure of the algorithm implementation:

Step 1. Initialize the weights in the network. Each weight has to be set randomly between -0.1 and 0.1.

Step 2. Initialize the activations of the biases. These values are always 1.

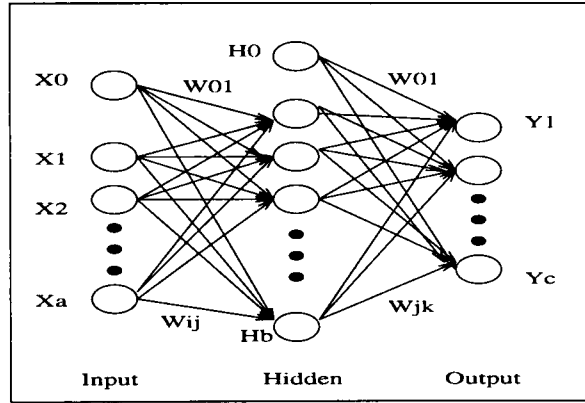


Figure C.1: A neural network for the Back Propagation Algorithm notations.

Step 3. Choose an input/output pair.

Step 4. Propagate the activations from the units in the input layer to the units in the output layer, using the sigmoid function.

Step 5. Compute the errors in the output and hidden layers. The errors are based on the network actual output and the target output.

Step 6. Adjust the weights in a way that minimizes these errors if the desired rms error is not reached. Rms errors are computed as in the Equation 3.1.

Step 7. Go to Step 3 and repeat for each pair of input/output vectors in the training set until the error for the entire system is acceptably low.

Note that the binary target outputs are usually given as 0.1 and 0.9 instead, because infinite weights would be required for the actual outputs of the network to reach 0.0 and 1.0. The rms error is used as a training stop scheme.

$$rmsError = \sqrt{\frac{\sum_{i=1}^n (t_i - Y_i)^2}{n}} \quad (3.1)$$

where t is the target output, Y is the network actual output, and n is the number of output layer nodes.

C.2 Mathematical Description

The following training steps correspond to the numbers in Section C.1, and the notations are described in Figure C.1.

Step 1. W_{ij} and $W_{jk} = \text{random}(-0.1, +0.1)$

Step 2. $X_0 = 1, H_0 = 1$

Step 3. Choose an input/output pair.

Step 4. Calculate the network outputs:

$$(i) H_j = \frac{1}{1 + e^{-1 \cdot \sum_{i=0}^a W_{ij} X_i}}, \text{ for all } j = 1, \dots, b$$

$$(ii) Y_k = \frac{1}{1 + e^{-1 \cdot \sum_{j=0}^b W_{jk} H_j}}, \text{ for all } k = 1, \dots, c$$

Step 5. Calculate the network errors:

$$(i) \sigma_k = Y_k(1 - Y_k)(T_k - Y_k), \text{ for all } k = 1, \dots, c$$

where Y_k is the actual output, T_k is the desired output.

$$(ii) \sigma_j = H_j(1 - H_j) \sum_{k=1}^c \sigma_k W_{jk}, \text{ for all } j = 1, \dots, B$$

Step 6. Adjust the weights:

(i) Without momentum function:

$$(a) \quad \delta W_{jk} = \eta \sigma_k H_j$$

$$(b) \quad \delta W_{ij} = \eta \sigma_j X_i$$

(ii) With momentum function:

$$(a) \quad \delta W_{jk}(t+1) = \eta \sigma_k H_j + \lambda \delta W_{jk}(t)$$

$$(b) \quad \delta W_{ij}(t+1) = \eta \sigma_j X_i + \lambda \delta W_{ij}(t)$$

Step 7. Repeat Steps 3-6.

VITA

Kim Chai Ng (Joelfer) was born in Johor Bahru, Malaysia, on February 26, 1971. He received the Bachelor of Science degree in Electrical Engineering with the highest honor, *Top Graduating Senior in the College of Engineering*, from the University of Tennessee, Knoxville, in December 1993 and received the Master of Science degree in Electrical Engineering from the same university in December 1995. He has a wide variety of interests. His current research interest is specified in fuzzy-neural control.