LEARNING FUZZY LOGIC FROM EXAMPLES

A Thesis Presented to

The Faculty of the College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Luis Alfonso Quiroga Aranibar

March 1994

ACKNOWLEDGMENTS

I would like to first thanks my advisor, Dr. Luis C. Rabelo, for his continuos commitment to quality and his tireless contributions for a better Industrial and Systems Engineering Department at Ohio University. His encouragement during my initial "fuzzy" days was fundamental in the development of this work.

I dedicate this work to my parents, because all that I am I owe it to them. Their infinite love and understanding for their children will forever be the guidelines in my life. I also dedicate this work to my sister and my brothers whose constant care and affection always give me the perseverance to try my best.

Special thanks to Dr. Donald D. Scheck for being the person who introduced me to the world of Fuzzy Logic. I appreciate very much his interest on my experiences in the manufacturing environment and my research.

I owe special gratitude to Dr. James J. Fales and Mr. Roger S. Vincent. They have been the most influential people in my years at Ohio University. Their willingness to share their knowledge and professional experience has granted me with the invaluable opportunity to perceive the demands of the professional world early on in my career. I am indebted for their trust in my assignments at the Center for Automatic Identification.

My friend and teacher, Dr. Evgueni Mourzine, deserves my recognition for his teachings in C++. His input and suggestions were always helpful and insightful. My deepest appreciation to Mrs. Linda Stroh, Mrs. Brenda Stover, and Mrs. Polly Sandenburgh. Their kind friendship and words of encouragement were essential for the completion of this work.

My endless gratitude to Monica, Connie, Tonga, Kevin, Franklin, Gaspar, and Makoto: they accompanied me during the writing of this thesis and they are the foundation in which it rests. You made the good times better and the tough times easier: the last few years worth living for. My love to all of you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I

Introduction

The study of the basic philosophies or ideologies of scientists is very difficult because they are rarely articulated. They largely consist of silent assumptions that are taken so completely for granted that they are never mentioned.....[But] anyone who attempts to question these "eternal truths" encounters formidable resistance.

- Ernst Mayr

In order to achieve excellent quality, manufacturing process planners make use of an array of methods and techniques to optimize or "control" problems such as job shop scheduling, material replenishment orders, sales order picking, etc. The optimality of these methods is difficult to define quantitatively. Quality "measures" for manufacturing planning fall into one of two groups: variables and attributes (Phadke 1989). A variable is a property that can be measured on a continuous scale, e.g., lateness is a variable measured in some unit of time. An attribute is a property that the process does or does not have. For example, the ability to adapt to sudden changes in demand periods would be an attribute. This ability to adapt to changes in demand, reflected in the total lateness, could determine the overall quality of the plan.

The initial effort in manufacturing planning is usually the responsibility of an industrial engineer. The typical approach involves the minimization of some function of the process properties. The engineer makes subjective decisions based on observation and experience. Good, quality planning, therefore, can be viewed as a set of specifications that the expert knows (or thinks) will satisfy many objectives. The expert will have to make tradeoffs among the properties so that the objectives are reached and the manufacturing process is operating at the "best" point.

A fundamental basis of manufacturing planning is that there exists a system model that defines a functional relationship between systems inputs and system outputs. The relationship between these inputs and outputs reflects the objectives of the manufacturing

effort which quality is typically measured along many dimensions: there are many objectives that must be optimized simultaneously (Lochner and Matar 1990). Very often, these different properties are incommensurate and competing. Thus, the concept of optimality must include accepting tradeoffs among the different quality characteristics.

Systems are traditionally represented with mathematical programming and analytical models that differ from the engineer's original commonsense reasoning in two aspects; first, they cannot deal with a degree of belief, and, second, they can not deal with inconsistent evidence (Dummermuth 1991 a, b, Cox 1992, Cox 1993). Most commercial expert systems attempt to use logical formalisms to represent expert human knowledge and reasoning. The major limitation of these systems is that they are not flexible when information required to answer a query is uncertain or incomplete. A human being, on the other hand, when faced with incomplete and uncertain information is able to sift through evidence and arrive at some sort of conclusion, uncertain perhaps, but sufficient to guide him or her through practical decisions (Mamdani and Assilian 1975, Zadeh 1985, Zadeh 1992). When faced with inconsistent information, classical logic fails invariably, all propositions are treated alike, neither is more true or false than the other.

In sharp contrast to most of the reasoning in physical sciences, commonsense reasoning is predominantly qualitative in nature (Zadeh 1985). In formal terms, the denotations of the concepts related to common sense reasoning are fuzzy sets, that is, classes with fuzzy boundaries in which the transition from membership to non membership is more gradual than abrupt (Zadeh 1965, Braae and Rutherford 1979, Zadeh 1985). This means that a fuzzy set can accommodate a "normal" distribution of items falling within a membership category, so that an item could be a member to a degree defined in the range of zero (non-member) to one (full member). This is the very essence of Fuzzy Logic.

Fuzzy Logic, FL, derived from fuzzy set theory, is a methodology that simulates thinking by incorporating the imprecision inherent in all physical systems. During the past

several years, FL and its application to control problems has emerged as one of the most prosperous areas for research in the application of fuzzy set theory. Fuzzy Logic Controllers (FLC) are already a promising alternative approach for controlling processes, especially those that are too complex for analysis by conventional techniques (Gupta and Tsukamoto 1980, Gottwald and Pedrycz 1985, Lee 1990). The rational behind FLCs is that an "expert" human operator can control a process without understanding the details of its underlying dynamics. The effective and real control strategies that the expert learns through experience can often be expressed as a set of condition-action, IF-THEN rules, that describe the process state and recommend actions using linguistic, fuzzy terms instead of classical, crisp, 0/1 rules.

According to many researchers (Lee 1990, Chen et al. 1992, Daugherity et al. 1992, Sugeno and Yasukawa 1993), FLCs offer three important benefits. First, developing a FLC can be more cost-effective than developing a model-based or other controller with the equivalent performance. Second, FLCs can cover a much wider range of operating conditions than traditional control (PID control, for example),. Third, FLCs are customizable, since it is easier to understand and modify their rules, which not only mimic human operator strategies, but also are expressed in linguistic terms used in natural language.

Based on this current literature, most publications fail to present a "user friendly" explanation for the process of developing a fuzzy controller. In general, the fuzzy machine is configured by a crisp-to-fuzzy transform/fuzzifier, the fuzzy inference, and the fuzzy-to-crisp transform/defuzzifier. In this general scheme, the major drawback of FL is the lack of theoretical justifications for the experimental observations. At present there is no systematic procedure for the design of a FLC. This has partly been due to the novelty of the fuzzy mathematics involved and to the operations required to produce a single valued deterministic control element from the linguistic rules (describing the input-output

map of the controller) and the fuzzy language (defining the meaning of the fuzzy sets) (McNeil and Freiberger 1992.) It is important to point out that this characteristic alone could be part of the reason why fuzzy logic has not been promoted much faster. The publications are too cryptic and the ideas presented are too "fuzzy" for the "fuzzy" novice.

This thesis implements a control strategy using a fuzzy methodology proposed by Li-Xin Wang and Jerry M. Mendel (1991). The methodology is first applied to a simple robot motion control study. Equations are developed and the criteria for the robot arm motion is simulated in a computer program written in C++.

Once the required data-examples are collected, the fuzzy methodology is presented with a "walk-through" example. Computations for membership functions, fuzzication, mapping of inputs and outputs, and defuzzification, are all explained to gain a clear understanding of the control algorithm and fuzzy logic in general.

The flexibility of implementing "day-to-day" knowledge in control analysis represents a promising alternative in the area of manufacturing planning. As a result, and taking into account the reported advantages of the FLC architecture, this work will further explore the applicability of fuzzy logic in manufacturing planning. The Wang-Mendel methodology will be applied to a simple production scheduling problem where a set of jobs required for processing are assigned to a single machine.

Research on job shop scheduling encompasses various approaches depending on the emphasis; therefore, it is not the aim of this work to compare the fuzzy controller against traditional techniques such as heuristic scheduling, mathematical modeling, or simulation studies. The objective is to introduce the concept of fuzzy logic in the field of manufacturing planning through a uncomplicated production floor scenario and ponder its applicability.

Chapter II

Literature Review

"Something like $100 billion worldwide has gone into AI and you can't point to a single AI product in the office, home, automobile, anywhere. With neural networks, it's approaching $500 million, and you can't point to any products and you probably won't be able for some time. Fuzzy techniques have gotten almost zero dollars from the government, and you can point to a lot of products, and will be pointing to a whole lot more."

- Bart Kosko

2.1 *Fuzzy Logic: Basic Concepts*

FL is based on Fuzzy Set Theory that was established by Lofti A. Zadeh in 1965 (Zadeh 1985, Cox 1992, Barron 1993). Currently, fuzzy sets are equipped with their own mathematical foundations, rooting from set theory basis and multi-valued logic (Braae and Rutherford 1979). Their achievements have already enriched the classic two valued calculus with a deep and novel perspective.

To understand the reasons for this extensive development, there are two main aspects worthy to be mentioned. First, the notion of fuzzy sets is important as a tool for modeling intermediate grades of belonging that occur in any concept, especially from an applications point of view. Second, a variety of tools incorporated in the framework of fuzzy sets, allow the planner to find suitable concepts to cope with reality.

In contrast to classical knowledge systems, FL is aimed at a formalization of modes of reasoning that are approximate rather than exact (Kosko 1992). FL is much closer in spirit to human thinking and natural language than the traditional logical systems. Basically, it provides an effective means of capturing the approximate, inexact nature of the world.

In this respect, what is used in most practical applications is a relatively restricted and yet important part of FL, centering on the use of IF-THEN rules. This part of FL

constitutes a fairly self contained collection of concepts and methods for handling varieties of knowledge that can be represented in the form of a system of IF-THEN rules in which the antecedents, consequences, or both, are fuzzy rather than crisp. Viewed in this perspective, the essential part of the FLC is a set of linguistic rules related to the dual concepts of fuzzy implication and the compositional rule of inference (Zadeh 1985, Meredith et al. 1991, Kosko 1992). In essence, then, the FLC provides an algorithm which can convert the linguistic control strategy, based on expert knowledge, into an automatic control strategy. What is of central importance, therefore, is that the fuzziness of the antecedents eliminates the need for a precise match with the input.

According to Zadeh (1985), Mamdani (1975), and others (Sugeno and Murakami 1985, Evans et al. 1989, Lee 1990, etc.), in a fuzzy rule-based system, each rule is fired to a degree that is a function of the match between its antecedents and the input (the term "fire" refers to the activation of a rule). The mechanism of imprecise matching provides a basic interpolation. Interpolation, in turn, serves to minimize the number of fuzzy IF-THEN rules needed to describe the input/output relationship. Recent applications of fuzzy control, (Pappis and Mamdani 1977, Yasunobu et al. 1983, Sugeno and Murakami 1985, Yagishita et al. 1985, Lee 1990, Brubaker 1992, Wiggins, 1992, etc), show that FLCs yield results that are superior to those obtained by conventional control algorithms. In particular, the methodology of FLCs appears very useful when processes are too complex for analysis by conventional quantitative techniques or when the available sources of information are interpreted qualitatively, inexactly, or uncertainly. As indicated by Gupta (1980), a FLC can be viewed as a step toward a rapprochement between conventional precise mathematical control and human-like decision making.

Fuzzy set theory deals with a subset A of the universe of discourse X, where the transition between full membership and no membership is gradual rather than abrupt. The fuzzy subset, A, is usually defined by a membership function, $\mu_A(x)$, mapping the degree

to which an element x belongs to a fuzzy subset A from domain X to the range [0,1]. Suppose that X is the set of all non-negative integers and A is a fuzzy subset of X labeled "approximately 10". Then the fuzzy subset can be represented by a membership function, $\mu_{10}(x)$. Figure 2.1 depicts a possible definition of the membership function.

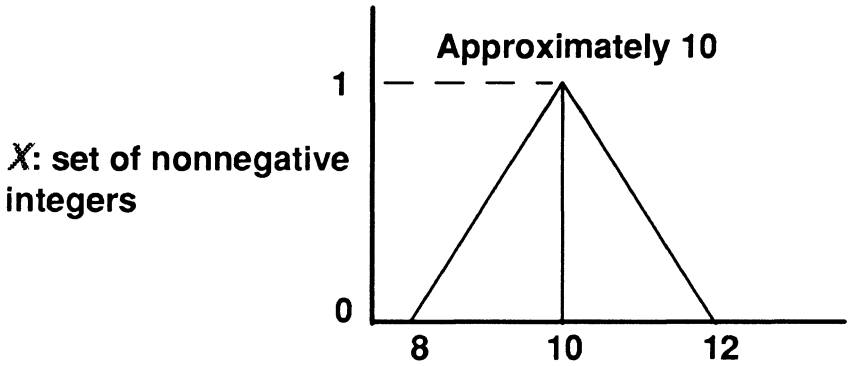

Figure 2.1. Membership function defining the concept "approximately 10" on a fuzzy system.

According to this fuzzy subset, the number 10 has a membership value of 1.0 (i.e., 10 is exactly 10), and the number 9, a membership value of 0.5 (i.e., 9 is roughly 10 to the degree of 0.5). Note that although illustrated as a triangle in the figure, a membership function in general can be linear, trapezoidal, convex curve, or in many other forms (Lee 1990, Dummermuth 1991). The proper definition of the fuzzy subset for a specific application is a major issue to be addressed in employing fuzzy set theory.

Although the membership function of a fuzzy set has some resemblance to a probability function when x is a countable set (or a probability density function when x is continuous), there is an essential difference which is clear when the underlying concept that each attempts to model is considered. Probability is concerned with undecidability in the outcome of clearly defined and randomly occurring events. Fuzzy logic is concerned

with the ambiguity inherent in the description of the event itself. Fuzziness is often expressed as ambiguity rather than imprecision or uncertainty, and remains a characteristic of perception as well as concept (McNeill and Freiberger 1993). Thus, referring to the previous example, 0.5 is the compatibility of 9 with the fuzzy concept "approximately 10"; 0.5 is not the probability that 9 is close to 10 half (or 50%) of the time.

A concept that plays a determinant role in the application of fuzzy logic is that of linguistic variables (Zadeh 1985). The motivation for this concept derives from the observation that in most common sense reasoning, humans employ words rather than numbers to describe the values of variables. For example, the lateness of a manufacturing process may be described as "very late", the quality as "average", the replenishment criteria as "not too accurate", etc., with each linguistic value representing a fuzzy set comparable to "approximately 10". In general, the values of a linguistic variable are generated from a primary term and its antonym (e.g., "excellent" and "poor" in the case of the linguistic variable "quality").

One of the most important characteristics of a linguistic variable is that the meaning of any of its linguistic values can be deduced from the meaning of the primary terms from which it is generated. For example, if lateness is stated to be "very late" and the linguistic value "late" is defined by a membership function which assigns a grade 0.8 to 5 days late, then the grade of membership of 5 days late in the fuzzy set "not very late" may be expressed as: 1.0 - (0.8)*(0.8) = 0.36, in which the effect of the modifier *very* is represented by the squaring operation, while the negation is accounted for by the subtraction of (0.8)*(0.8) from 1.0. Further description of modifiers can be found in Zadeh (1985), Sibigtroth (1991), and Kosko (1992).

Another important characteristic of linguistic variables is the universality of their structure. For example, the variables "late" and "accurate" have the same structure in the sense that their respective linguistic values have the same form and differ only in the

primary terms. This implies that the meaning of the linguistic values of any linguistic variable may be computed by a general procedure which applies to all linguistic variables (Kosko 1992). Figure 2.2 depicts the concept of linguistic variables, linguistic values, and modifiers.

The design of a fuzzy rule-based control system is largely a "trial and error" task. There has been extensive research in this area. B.H. Wang and G. Vachtsevanos (1991), presented a systematic methodology to the design of a learning fuzzy logic system. In this concept of indirect control approach, the selection of control parameters relied on the estimates of process parameters. The control law consisted of three components: on-line fuzzy identifier, the desired transition model, and the fuzzy controller. With only a crude initial description of the process, the controller could be modified on-line using the fuzzy process model identified during the system's real time operation. This method exhibited the trade-off between performance and computational complexity.

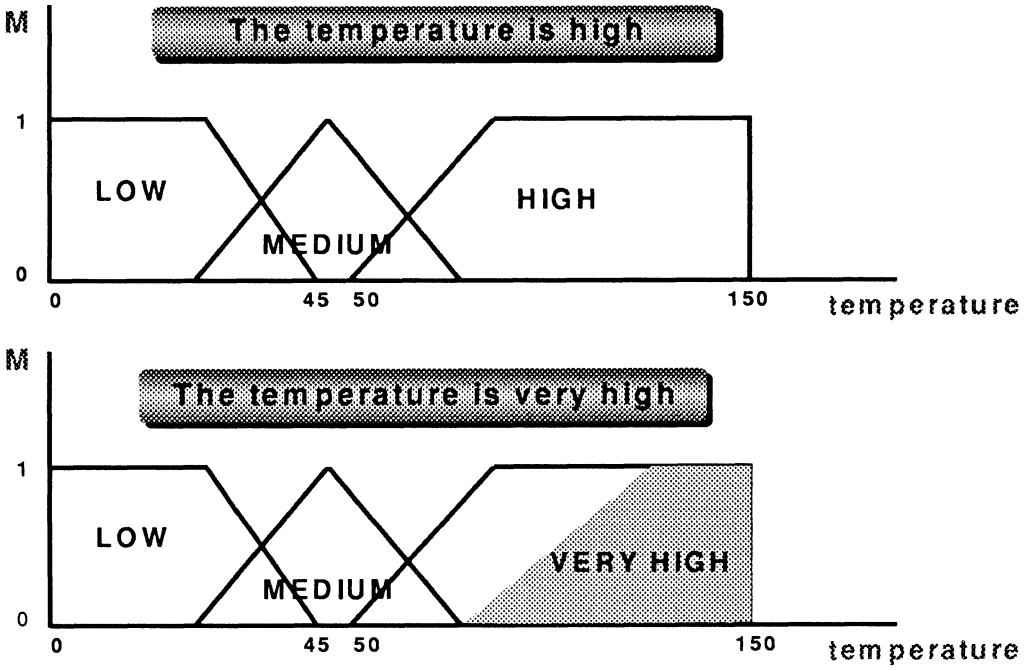

Figure 2.2. The first graph describes the linguistic variable "temperature" with linguistic fuzzy, values low , medium, and high. The gray area on the bottom graph depicts the effect of the modifier *very*.

R. M. Tong (1980) reported two attempts to construct fuzzy relational descriptions of experimental data. He reported that the great advantage of a fuzzy model is that it is relatively simple to construct and it is itself a simple structure. His main conclusions were two: first, the overall concept did not have a common framework and second, the concept needed considerably more investigation before its true worth could be evaluated.

H. Nomura, et.al. (1991) proposed a learning method of fuzzy inference in which the inference rules expressing the input-output relation of the data were obtained automatically from the data gathered by specialists. In this method, triangular membership functions were tuned through experience. The learning speed and the generalization capability of this method were higher than those of a conventional back propagation neural network. Furthermore, the system developer expressed the knowledge acquired from the input-output data in the form of fuzzy inference rules.

T. Takagi and M. Sugeno (1983), reported that most fuzzy controllers had two problems. First, there was a defect in the reasoning algorithms. Second, the way of acquiring rules introduced ambiguity, particularly when defining precise parameters for the rules. These researchers presented four methods to derive fuzzy control rules based on: (1) operators experience and/or control engineer's knowledge, (2) the fuzzy model of the process, (3) the operator's control actions, and (4) knowledge. The results of the study suggested that there was no one best method and that any of the methods was desirable to be combined with another. Moreover, Triantaphyllou (1990) showed that small changes in the membership values or even repetitions of identical alternatives can mean the difference between selecting one alternative over another in many decision-making problems. He recommended further research in the evaluation of membership function definitions.

The studies cited indicate that there is a lack of a formal procedure for the design of a FLC. Nevertheless, there are some basic steps that remain standard in the design of such system. The first step mandates the definition of the inputs affecting the final state or output. Once the acting inputs are determined, the fuzzy values of each input are declared and the degree of membership functions for each range are proposed. The second step, calls for the identification of the outputs and the fuzzy ranges for those outputs, including the degree of membership functions. Third, an inference engine, or black box, with a decision rule base is developed so as to map the inputs to the outputs. Finally a defuzzification procedure converts fuzzy control decisions into crisp non-fuzzy control orders.

Designing the fuzzy controller requires general knowledge of the process and also knowledge of the effect of the membership functions on the performance of the fuzzy controller. During the initial design, membership functions are most likely to be chosen in a trial and error basis. This means that membership functions will have to be retuned to achieve more desirable behavior. As a result, evaluation of appropriate membership functions is an important issue for engineering control problems (Brubaker 1992). In addition, in the process of constructing a scale that satisfies some conditions imposed on the overall performance of the system, an ambiguity related to justification is likely to be introduced in the system. For this reason, a membership function is typically determined heuristically; however, it requires a large number of trial and error evaluations.

Figure 2.3 shows the configuration of a typical FLC. The crisp-to-fuzzy transform is a mapping of an actual crisp value to a degree of membership via an input degree-of-membership function. Figure 2.4 describes the crisp to fuzzy transform. The resulting degree of membership then becomes an input to the next system block, the inference mechanism.

In the inference mechanism, inputs and truth values serve as conditions for the rules that make up the rule base. At regular intervals, the fuzzy controller samples inputs and applies them to the rule base, resulting in appropriate system outputs. Theoretically, the rule base should cover all possible combinations of input values, but such coverage is typically redundant and only those rules with a higher degree of membership are kept.

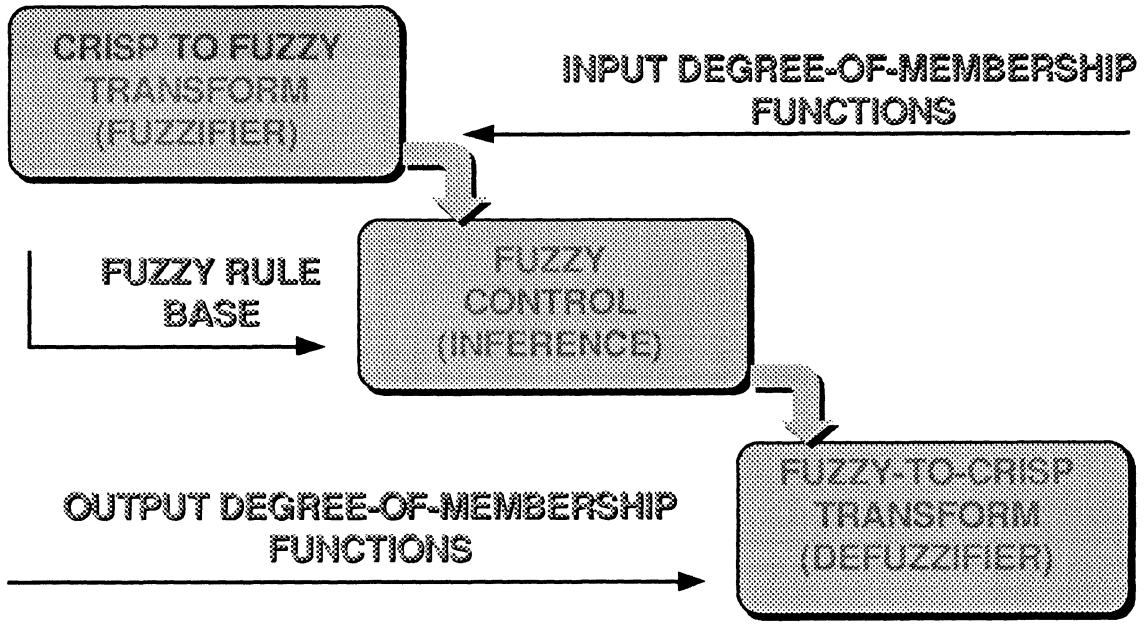

Figure 2.3. A FLC has three major components. First, a crisp-to-fuzzy transform changes crisp inputs into degrees of membership, or truth values. Second, an inference mechanism determines the action to take by applying the truth values as conditions to rules. Finally, a fuzzy-to crisp transform converts fuzzy actions and combines them to form a single, executable action.

In the fuzzy-to-crisp transform, there are mainly four ways to obtain a defuzzified output. The maximizer technique takes the maximum degree of membership value from the various triggered rules and performs the corresponding single action. If two actions have the same degree of membership, and that value happens to be the maximum, then one possibility to solve the problem is to use an average of the corresponding outputs; another is to select the action associated with a rule's position in the rule base.

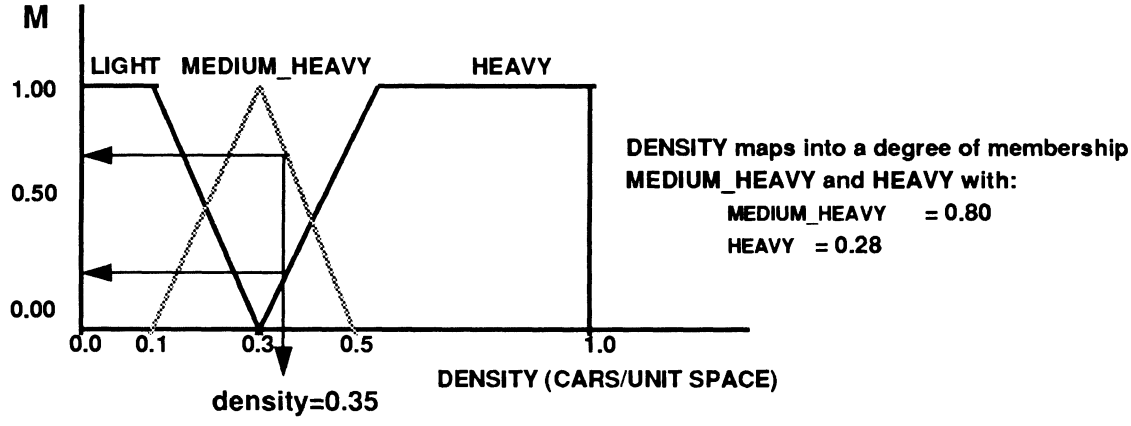

Figure 2.4. The crisp-to-fuzzy step takes a crisp input value and converts it into a fuzzy value or degree of truth. In this case, a car density of 0.35, or 35 cars per mile, corresponds to a 0.70 degree of membership in the "medium_heavy" fuzzy range; also, it corresponds to a 0.20 degree of membership in the "heavy" fuzzy range. In this case, "medium_heavy" is predominant over "heavy" due to its higher degree of membership.

The maximizer technique is the simplest combination/defuzzification approach, but it ignores potentially important actions. A second technique is referred as the weighted average method. This method averages the various actions after assigning weights based on the degree of membership values. Although conceptually strong and not computationally demanding, this approach suffers form ambiguity in the output section, as does the maximizer method.

As illustrated in Figure 2.5, the centroid method results in an output action that is associated with the center of mass of the active outputs. Although this method does away with the ambiguity introduced by the previous methods, it is computationally intensive and it suffers from an additional shortcoming (Brubaker 1992). For vertically symmetric degree of membership functions, the centroid always corresponds to a single output value, regardless of the value of the input degree membership. As a result, to achieve smooth operations over the entire output range, several (or at the very least two) rules must fire at each system iteration. Therefore, in order for multiple rules to fire at once, the input

degree of membership functions must overlap. Despite these shortcomings, the centroid method is currently the best technique for combination and defuzzification.

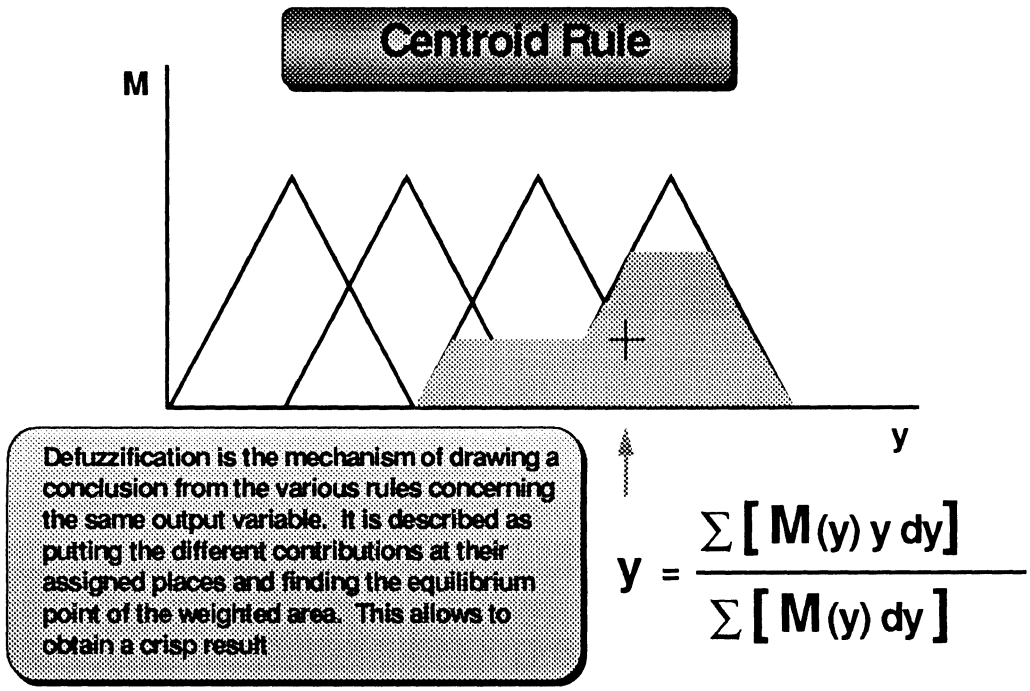

Figure 2.5. The centroid method. Notice that the crisp output value is the center of gravity of the area defined by all rules that fired together.

The remaining defuzzification method, the singleton technique, is a special case of the centroid rule. This technique represents each fuzzy output set as a single output value by using a weighted average to combine multiple actions. This approach requires much less computation than the centroid method, but it still requires overlapping input functions to avoid discontinuities in the output. Chapter III will explain in further detail both the fuzzification and defuzzification procedures.

In summary, management of uncertainty is a key issue in the design of a fuzzy controller. Much of the information available for use in knowledge-based systems is

incomplete, imprecise or not totally reliable. FL combined with linguistic variables provide a powerful mechanism for dealing with this problem.

2.2 *Fuzzy Logic Control: Current Applications*

Fuzzy logic and the concept of linguistic variables have found a number of applications in such diverse fields as industrial process control, medical diagnosis, assessment of credit worthiness, risk analysis, etc. (Schwartz and Klir 1992, IEEE Transactions on Fuzzy Systems Editorial 1993). The greater simplicity of implementing a control system with fuzzy logic can reduce complexity to a point where previously insolvable problems can be solved. Fuzzy systems typically result in a 10:1 rule reduction, requiring less software to implement the same decision-making capability (Kosko 1992).

An important application of the concept of a linguistic variable was pioneered by Mandani and Assilian in 1974, and described in their seminal paper on the fuzzy logic control of a steam engine (Mamdani and Assilian 1975). The apparatus had two parts: the boiler and the engine. Each sought a target. The boiler's was a specific pressure and it reached that goal by adjusting the heat. Raising heat increased the pressure while cutting the heat diminished it. The engine's target was the right piston speed, which is attained with a throttle valve. Opening up the valve boosted the speed and closing it down decreased it. Mandani and Assilian broke down the problem into fuzzy IF-THEN rules. Because the system used information of three kinds, they created three sliding scales. Two showed input, that is, information about the state of the boiler. The third was output, a command action.

Once the three continuous variables were in place, e.g., pressure error, change in pressure error, and heat change, Mandani and Assilian placed a series of seven triangles on each one. These overlapping fuzzy sets, as depicted in Figure 2.6, all had the same size

and shape, and were spaced evenly along each line. Moreover, the fuzzy sets were defined as NB (Negative Big), NM (Negative Medium), NS (Negative Small), CE (Center), PS (Positive Small), PM (Positive Medium), and PB (Positive Big).

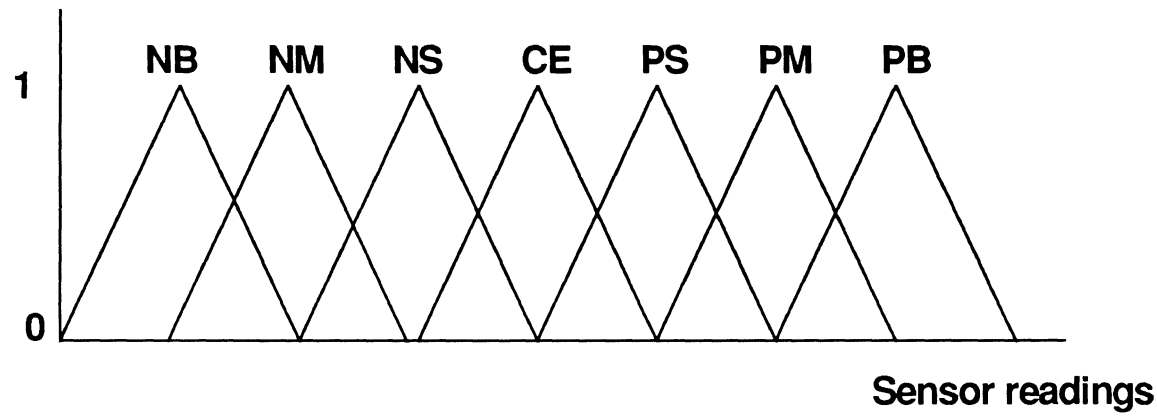

Figure 2.6. Overlapping fuzzy sets for the Mandani-Assilian steam engine. The fuzzy variables are pressure error, change in pressure error, and heat change.

Next, the pair had to build the network of rules using these sets. The new fuzzy device commenced in a state of knowledge. Mandani and Assilian provided the fuzzy controller with 24 rules. These rules described the workings of the engine in simple terms and also reflected the experience of skilled operators. The rules enabled the steam engine to respond to linguistic as well as numerical information. Moreover, the readings from the sensors fired several rules at once to different degrees. Each firing was like a recommendation that was assessed by the controller and merged with other recommendations in order to get a crisp command.

Figure 2.7 shows the fuzzy system mapping inputs to outputs for the steam engine. Most of the squares are blank. They are not necessary since, in practice, the system will not call on them. The controller then totaled up the recommendations and found the center of gravity of the geometric shape resulting from all fired rules. This value was crisp and it lead at once to a numerical command. Figure 2.5 illustrated this method.

	NB	NM	NS	CE	PS	PM	PB
NB				PB			
NM				PM			
NS				PS	NS		
CE	PB	PM	PS	CE	NS	NM	NB
PS			PS	NS			
PM				NM			
PB				NB			

Figure 2.7. Fuzzy mapping of inputs and outputs for the Mandani Assilian steam engine. Notice that it is not required to have all mappings active. The set of predominant mappings, rules, will suffice to control the system.

After Mandani and Assilian published their results in 1975, other applications appeared. Lemke and Kickert (1975) developed a fuzzy controller to regulate the temperature of a warm water plant. Larsen and Ostergaard (1976) began work on a fuzzy heat exchanger. In the same year, Rutherford (1976) applied fuzziness to a sinter-making process of a British Steel Corp plant. The fuzzy controllers outperformed human beings and slightly excelled the conventional control devices.

The first commercial application of a FLC came about with the F.L. Smidth & Company, a cement processing plant in Denmark (McNeill and Freiberger 1993). Conventional methods could regulate the blending of ingredients and other kiln process well enough but could not master the formation of clinker, the end product of the complex components of cement. A key factor was burning the material at the right temperature. If the furnace was too hot, the clinker was hard and the kiln wasted energy. If it was not hot enough, quality suffered. It was important to negotiate balance among the amount of fuel,

the amount of material fed into the kiln, and the kiln rotation speed. Based on a training text of 27 IF-THEN rules, the first trial operated for 6 days in June of 1978. Fuzzy control slightly improved on the human operator and also cut fuel consumption. In 1980, a fuzzy controller was permanently installed in the kiln. This resulted in the publication of the first commercial application of fuzziness. Furthermore, the researchers at F.L. Smidth & Company found that some calculations were necessary. They developed FCL, Fuzzy Control Programming language, which somewhat resembled FORTRAN. By 1982, a system based on this language was installed at a cement plant in Oregon. The controller has continuously been redefined and have since then sold hundreds all over the world. It is estimated that 10% of the world's cement kilns now use fuzzy logic, and that 50% could benefit greatly form it.

In more recent applications, NASA has reported encouraging results in a number of areas using FL (Cox 1993). One of the most advanced projects is a controller for space shuttle proximity operations; that is, maneuvering around or keeping position with respect to another object in space. Work has been progressing on a fuzzy-based translational controller that deals with the parameters of azimuth and angle and their respective rates of change, and the range and rate of distance change with respect to another object. The results of simulations have been encouraging, specially in terms of fuel efficiency. In holding position with respect to another space craft, the fuzzy controller required significantly less acceleration, that is, smaller increments of position change, than the human-controlled operation. In overall maneuvers, the fuzzy controller has shown 20% to 70% better fuel efficiency than the currently used digital auto pilot and the best simulation runs of human pilots.

Knowledge Based Systems (KBS), has added a fuzzy front end to the image database of a major European police department that significantly reduces the number of look-throughs witnesses have to review before finding a set of pictures they can seriously

work with to identify a subject (Cox 1993). In addition to implementing a front end that uses fuzzy sets to describe characteristics such as "old", "thin", and "tall", KBS built in what it calls "perspective shifting" and "semantic plies". Perspective shifting changes the shape of the fuzzy set representing "tall" if the witness is, for instance, a 16 year old girl or is Japanese. This quality lets the system search for "tall" from a "young" perspective or, for that matter, for any other characteristic as modified by the preconceptions of the searcher. Semantic plies perform a similar action, adjusting descriptions such as "tall for women" and "heavy for Samoans". According to KBS, the old manual system required an average of 16 look-throughs to find a set of pictures that a witness could actually work with, while the fuzzy system reduced the number of look-throughs to only two.

Applications of FL in water quality control, automatic train operation systems, elevator control, nuclear reactor control, automobile transmission control, fuzzy memory devices and fuzzy computers have pointed a way for an effective utilization of fuzzy control in the context of complex processes. Some of these applications are briefly presented below.

Sanyo Fisher USA Corp. turned to fuzzy logic for the design of a new video camcorder (McNeill and Freiberger 1992). The use of FL responded to the need of greater memory to implement all the functions and features required for the camcorder. Conventional control methods could only achieve the same goals at the expense of greater memory utilization. The fuzzy system reduced the amount of software needed to determine the optimum focus and lighting levels, allowing the functions to be implemented without external memory.

Rockwell International Corp. Science Center (Chiu 1991), has applied FL for the control of the engine idling problem. Because the engine's variables (ignition, timing, RPM, and airflow) can be represented by a relatively compact set of equations, the application of fuzzy logic improved little on conventional control methods. The time that

it took to implement the FLC, however, was an order of magnitude less than the time it took to develop the conventional system.

Canon Inc., in its auto focusing camera, uses a charge-coupled device (CCD) array to measure the clarity of an image in six regions of the screen, thus determining if the image is in focus (McNeill and Freiberger 1993). Because the lens moves during focusing, the camera also measures the rate of change of focus, and slows the movement of the lens to avoid overshooting the optimum focus. The Canon controller uses 12 inputs; six to measure the rate of change as the image comes into focus and six to measure instantaneous clarity for each region of the screen. There is one output, the position of the camera lens. Canon employs 13 rules in its auto-focusing camera, requiring only 1.1 kilobytes of memory to implement.

The power of a small rule base has been demonstrated at the Ford Motor Co., where a fuzzy application using just three rules backs a simulated tractor-trailer into a parking space without jackknifing (Cox 1992). Researchers at Ford first studied the operation of a neural network controller based on a similar program developed by Widrow, and from it they were able to infer the rules necessary to built a fuzzy rule base that performed the same function but took much less time to develop and seemed to simplify the learning process.

An example of the direct application of human input to the design of a fuzzy system was the development of the electronic continuously variable transmission (ECVT) (Self 1993). This transmission, used in the Subaru Justy automobile, employs a fuzzy control system to continuously adjust the position of a belt that changes the transmission pulley ratio and hence the vehicle's acceleration and performance. Subaru questioned test drivers about their response to different pulley ratios to define what belt position on the pulleys corresponded to maximum performance without uncomfortable acceleration. The data was then used to generate the functions that governed the shifting.

These examples demonstrate how the primary use of FL has been embedded in controllers. Currently, FL is entering the main stream with a wide range of desktop applications. Aptronix's Fuzzy Inference Development Environment supplies the designer with a tool to develop FL-based applications. It operates under Windows 386/486 machines and costs less than $2000. FIDE includes a fuzzy inference language, a fuzzy-system standard environment, a graphic editor, with which to draw graphs of membership functions, debugging tools, and a real time code generator. Fuzzy Ware makes several products including FuzzyCalc, Fuzzy Quote, Fuzzy Cell, Fuzzy Choice, and Fuzzy Cost (Barron 1993). These products are used as management decision support systems for custom, turnkey, and off-the-shelf applications, as well as software techniques to implement both fuzzy logic and fuzzy math. FuzziCalc can make decisions based on complicated combinations of hard (well understood) and soft (fuzzy) factors. It produces answers that are mathematically verifiable and easy for people new to the field to model and interpret. Fuzzy Ware claims that if the application is suitable for the use of FL, productivity may be improved as much as 30 fold.

Another low-cost tool, called the Fuzzy Systems Manifold Editor from Fuzzy Systems Engineering, defines and checks out the core of a fuzzy application in a Windows environment (Self 1993). The concept of a manifold refers to the Fuzzy Associative Memory (FAM), a n-dimensional combinatorial matrix that represents all the relationships between fuzzy input and output values. The use of the fuzzy manifold editor allows to graphically edit membership functions and examine the system behavior by simply setting input values and reading the resulting outputs. In addition, the fuzzy developer can select output functions, change or edit graphs that redefine the input functions and then access a manifold "walker". The manifold editor can output source-code including files in C, FORTRAN or BASIC that can be built into other programs.

As the use of FL becomes widespread, especially in consumer products, optimizing cost performance will become ever more important. Companies such as Togai, NeuraLogix and Omron have been offering parallel fuzzy processors for several years. Because of their parallel architecture, such processors offer very high speed for fuzzy inferencing, as well as bringing robustness to the hardware level of a fuzzy design. For example, fuzzification and defuzzification entail mapping an input value to a point on a function and vice versa. Linear interpolation takes longer than finding the point in a look up table, which takes more memory. Using complex curves as membership functions is still popular in some non-control applications, but it is almost completely out of favor in control applications because of the complex formulas and processing overhead involved.

2.3 *Approaches to Manufacturing Planning*

This section reviews different approaches to the manufacturing planning problem (Rabelo, 1990), with special emphasis to job-shop scheduling applications. The section is divided into two parts. The first part deals with traditional methodologies and strategies used to solve the manufacturing planning problem. The second part deals with Artificial Intelligence approaches.

2.3.1. Traditional Techniques

Mathematical Programming and Analytical Models. Mathematical programming and analytical models have been extensively applied in the area of manufacturing planning (Baker 1974, French 1982). Traditionally, these manufacturing situations have been formulated with the use of integer programming and mixed integer programming. There are two main limitations to these approaches. First, these models are appropriate to only a

hand full of problems due to the unique characteristics of the manufacturing environment. Second, the addition of multiple constraints increases dramatically the computational requirements of the formulation. Some approaches are reviewed below.

Integer Programming Models. Chang and Sullivan (1984) reported the development of a two-phase method for real-time scheduling of work in a flexible manufacturing system using an integer programming model. Although the main advantage of this approach is that it considers global information, optimality is determined by all feasible schedules, which in turn is computationally expensive. The integer programming formulations of Afentakis (1984) and Raman et al. (1984) included the relationship between the system setup phase and the scheduling phase. Afentakis (1984) maximized the production rate with the use of operation and layout graphs. Raman et al. (1984) formulated procedures for simultaneously scheduling jobs on machines and material handling devices in a static environment. This procedure uses a depth-first branch and bound algorithm which builds the solution forward in time.

Tang (1985), and Tang and Denardo (1986) developed job scheduling models for a single machine problem to minimize the number of tool switches and tool switching instants. The drawback of this analysis is that it is restricted to manufacturing process with a series of M machines where each job is processed from machine 1 to M exactly once. In addition, no work-in-process is allowed. Scheck et al. (1975), proposed a sequence programming algorithm were the inputs are the job routings and time standards for all jobs over all possible machines, the number of shifts per machine, and the efficiency of each shift on each machine. The core of the algorithm is an iterative process for setting up schedules on Gantt charts and evaluating these schedules with an objective function and a feedback element to produce an optimal or close to optimal schedule with limited computational resources. Results were consistently better than schedules worked by hand.

Relaxation of Constraints. In the Hutchinson 's et al. (1989) model, the relaxation of constraints methodology is compared to an scheduling algorithm that divides the problem into a loading/scheduling problem and to a dispatching rule. The optimal scheduling model uses a branch and bound scheme with relaxed precedence constraints. Although the scheduling model performed better, it recorded a 450% increase in computational resources over the decompose scheme.

Disjunctive Graphs. The use of disjunctive graphs is reported by Saleh (1988) and Odrey and Wilson (1989) who represented production operations, job precedence, and machine/operation relationships in the context of steady-state scheduling procedures. Short computational run times were reported a typical scheduling problem consisting of 6 to 10 machines and 10 to 20 jobs. The research showed that increasing the number of jobs or the number of machines did not have a considerable impact on the running times, while the combined effect of increasing jobs and machines, i.e. the number of operations, had more impact on the run times.

Dispatching Rules. Dispatching rules are procedures designed to provide good solutions to complex problems in real time. Hershauer and Ebert (1975) used simulation as a technique to "approximate a best sequencing that is simple in form and easily applied in the job shop environment". Gross (1987), with the use of control theory, and Bunnag and Smith (1985), through computer search, attempted to provide the practical advantage of a single dispatching rule to the required performance measures.

Classification. According to the performance criteria for which they are developed, dispatching rules are classified (Wu 1987) as follows:

i. Simple priority rules, which are based on information related to the jobs. Sub-classification is made based on information such as processing times, due dates, number of operations, etc.

ii. Combination of Simple Priority Rules, e.g., according to the different available information.

iii. Weight Priority Indexes, which are a combination of the above with different weights.

iv. Heuristic Scheduling Rules involving more complex considerations, such as anticipated machine loading, the effect of multiple routings, etc.

Simulation of Dispatching Rules. Dispatching rules have extensively been analyzed using simulation techniques to evaluate their performance under different scenarios. These studies can be divided in two groups: those involving (1) processing time criteria, and those involving (2) due date criteria.

In (1), Conway and Maxwell (1962) studied the shortest process time (SPT) rule and some of its variations. Although they found that some individual jobs could experience prohibitive long flow times, the STP performed well and the mean flow time for all jobs was substantially reduced. With respect to the mean value of the basic measures of waiting time and system status utilization, the STP was reported to be optimal for the set of all dispatching rules. Eilon and Choudury (1975) used the SPT rule, a modified SPT, and FIFO in their simulation studies. The SPT and its modified version produced lower variances and better results than FIFO for the mean flow time measured. Hershauer and Ebert (1975) applied six different standard sequencing rules and six combination-type rules to minimize the mean flow time. As opposed to the minimum mean flow time obtained through SPT, the combinatorial rules yielded poor results.

In (2), and through the use of job tardiness, Gere (1966) listed the effectiveness of due-date type rules. Lowest tardiness was optimal for static problems with minimum slack time, while slack per operation was found to be the most optimal for dynamic problems. SPT performed the lowest in both sets of tests. Rochette and Sadowski (1976) found that

earliest due date (EDD) produced the lowest tardiness and the best performance for 80% shop load.

Other Approaches. Bunnag and Smith (1985) presented a method for developing an effective rule specific to the particular job shop scheduling problem to be solved. The method involved a generalized objective function, a multifactor priority function, and a computer search to determine the best weights to use in the priority function. In order to adjust the control parameters, Gross (1987) utilized a closed loop feedback system which analyzed the trajectory followed by the flexible manufacturing system and the degree to which the assigned goals are achieved.

2.3.2 Artificial Intelligence Techniques

Artificial Intelligence (AI) techniques such as knowledge based expert systems (Fox and Smith 1984), artificial neural networks (Takefuji and Foo 1988) , and genetic algorithms (Walbridge 1989) have been applied to manufacturing scheduling systems.

This section, divided in two parts, describes AI applications in scheduling systems for manufacturing. In the first part, knowledge-based scheduling systems are initially presented. In the later part, current research in artificial neural networks and manufacturing scheduling is discussed.

Knowledge-based Expert Systems. Knowledge-based Expert Systems (KBESs) are one of the most relevant AI tools. KBESs have provided a suitable environment where ill structured dedicated knowledge with well structured generic knowledge form scheduling theory can be combined towards the development of a good schedule. Several of these systems are described below.

Intelligent Scheduling and Information Systems (ISIS). ISIS is a knowledge based expert system designed to provide intelligent support in the domain of job-shop scheduling (Fox and Smith 1984, Fox and McDermott 1986). This KBES is able to construct schedules by performing a hierarchical, constrained-directed search in the space of alternative schedules. In addition to incrementally scheduling orders for production, the ISIS architecture can be exploded in a reactive manner to handle unexpected events.

A Multi-Pass Expert Control System (MPECS). MPECS is an expert cell controller based on AI and simulation procedures (Wu 1987). Evaluation of the dispatching rules at short time intervals, combined with a continual alternation on different dispatching rules makes the system more responsive and adaptive to the environment. The major drawback of this system is its inability to learn from previous outcomes and therefore evolve toward a more adaptive system.

Knowledge-Base Scheduling System (KBSS). Kusiak (1989) presented the design of a Knowledge-based Scheduling System for an automated manufacturing environment. KBSS uses a data base, algorithms, declarative and procedural knowledge, and an inference engine. Two major drawbacks of KBSS are the inability to identify the best possible sequence of priority rules for a problem and automatic learning is not implemented in the problem solving architecture.

Manufacturing Decision Making (MADEMA). MADEMA is a decision making framework for manufacturing systems with utilizes some aspects of operations research approaches as well as some artificial intelligence techniques such as rule-based systems and heuristics (Chryssolouris et al. 1989). MADEMA assigns a work center's resources to production tasks following a number of steps that a human undertakes. This approach views a flexible manufacturing system as a job shop that consists of a number of work centers and as a result of this operational policy for decision making, the system is able to achieve the optimization of several aspects.

Automatic Reasoning Operations Research (AURORA). AURORA is a knowledge-based expert planning system which utilizes pattern matching to find problem patterns (Kusiak 1990). This system has been applied to solve Air Force mission-planning problems and an attempt has been made to apply this expert system to the cargo loading problem for the NASA space shuttle.

Artificial Neural Networks Applications. Artificial neural networks have been applied to Optimization Theory. Their applications have particularly been dedicated to NP-complete problems, and in specific to the Traveling Salesman Problems (Culioly et al. 1990). However, artificial neural networks for optimization problems have not produced consistently optimal/real time solution due to the limitations in hardware and the development of algorithms.

Stochastic Neural Networks for Solving Job-Shop Scheduling. Foo and Takefuji (1988) introduced an artificial neural network architecture based on a stochastic Hopfield (Lippmann et al. 1987) neural network model for solving job-shop scheduling.
Takefuji and Foo (1988) indicated that "this stochastic approach produces near optimal solutions, although it does not guarantee the finding of an optimum solution. Also, an optimum solution using simulated annealing is generally not found because the annealing procedure would take an infinite amount of time.

Integer Linear Programming Neural Networks. Foo and Takefuji (1988) developed an integer linear programming neural network (ILPNN) based on a modified Hopfield neural network to be used to solve job-shop scheduling problems. The constraints of the job-shop problem are formulated as a set of integer linear equations.. Optimal and near optimal solutions in real time are reported using the ILPNN approach. However, optimal solutions are not guaranteed.

Chapter III

The Wang and Mendel Methodology

We fuzzy logic theorists have brought control theory from seventh heaven to the ground
- Michio Sugeno

This chapter presents a detailed description of the methodology proposed by Li-Xi Wang and Jerry M. Mendel form the University of Southern California. In their 1992 seminal paper "Generating fuzzy rules by learning from examples", Wang and Mendel (1992) designed a five step procedure intended to generate fuzzy rules from numerical data. The method determines a mapping from the input space to the output space based on the combined fuzzy rule base using a defuzzification procedure. The mapping was proved to be capable of approximating any real continuous function to an arbitrary accuracy.

3.1 Fuzzy Associative Memory

In general, for any industrial control application, the information required for the initial analysis and design of the system is divided in two kinds: numerical information and linguistic information obtained from skilled human beings (Wang and Mendel 1992). Most of the current control schemes have replaced traditional approaches by integrating both control/signal processing methods and linguistic information from human experts. These systems are then calibrated until results start producing answers closer to the original goal. Simulations are ran to validate the system and prove its correct performance for the specific problem. This approach presents two weak points. First, the method works well for the specific problem; as a result, it is quite problem dependent in a sense

that a modification of the original problem may invalidate part or the entire formulation. Second, the approach uses different subjective criteria in order to calibrate the system; therefore, it is difficult to obtain a common theoretical framework for modeling variations of the original problem.

Wang and Mendel proposed a method for combining these two sources of information into a common framework called Fuzzy Associative Memory (FAM) bank (Wang and Mendel 1992). A technical comment is in order for sake of understanding the concept of FAMs. The fuzzy framework is numerical and multidimensional. The expert system framework is symbolic and one dimensional, with usually only bivalent expert "rules" or propositions allowed. Despite of this difference, both frameworks can encode structured knowledge in linguistic form. But, the fuzzy approach translates the structured knowledge into a flexible numerical framework and processes it in a manner that resembles neural-network processing (Kosko 1992.)

A fuzzy set defines a point in a cube. A fuzzy system defines a mapping between cubes, i.e., maps fuzzy sets to fuzzy sets (Kosko 1992). Therefore, the definition of a fuzzy system can be extended to allow arbitrary products of arbitrary mathematical spaces to serve as the domain or range spaces of the fuzzy sets. These continuous fuzzy systems behave as associative memories that map close inputs to close outputs. These systems are referred as FAMs. The simplest FAM encodes the FAM rule or association, which associates the n-dimensional fuzzy input with the n-dimensional output. These minimal FAMs essentially map one input to one output.

In the design of a fuzzy system, the first step refers to the collection of the available information. As described previously, the information has two forms: human experience and sampled input-output pairs recorded by the human expert. Human experience is presented as a set of IF-THEN rules explaining under what conditions what action should be taken. The sampled pairs give numerical information concerning the

inputs and the outputs. The types of information are different in nature but they both have a common characteristic; each is usually incomplete by itself. Although the system can be controlled successfully by a human operator, some of the information will be lost when transferring his or her experience onto an expert system structure. On the other hand, past numerical data is not enough to predict future control situations.

Combining these two types of information, Wang and Mendel formulated the key points for their approach. This consisted of the generation of fuzzy rules from numerical data pairs, the collection of the fuzzy rules into a common FAM, and , finally, designing a control system based on the FAM bank using the centroid defuzzifying method.

3.2 Generating Fuzzy Rules from Numerical Data

Assuming a given set of desired input-output data pairs:

$$(x_1^{(1)}, x_2^{(1)}; y^{(1)}), (x_1^{(2)}, x_2^{(2)}; y^{(2)}), \ldots\ldots \qquad (1)$$

where x_1 and x_2 are inputs, and y is the output. This simple two-input one-output case is chosen in order to emphasize and to clarify the basic ideas of the new approach. The task here is to generate a set of fuzzy rules from the desired input-output pairs of (1), and use these fuzzy rules to determine a mapping $f:(x_1, x_2) \rightarrow y$.

Step 1: Divide the Input and Output Spaces into Fuzzy Regions

Assuming that the domain intervals of x_1, x_2, and y are $[x_1^-, x_1^+], [x_2^-, x_2^+]$ and $[y^-, y^+]$, respectively, where "domain interval" of a variable means that most probably this variable will lie in this interval (the values of a variable are allowed to lie outside its

domain interval). The domain interval is divided into *2N+1* regions (N can be different for different variables, and the length of these regions can be equal or unequal), denoted by SN (Small N), ..., S1 (Small 1), CE (Center), B1 (Big 1), ..., BN (Big N). Then, each region is assigned a fuzzy membership function. Figure 3.1 shows an example where the domain interval of x_1 has been divided into five regions (N=2), the domain interval of x_2 has been divided into seven regions (N=3), and the domain interval of y has been divided into five regions (N=2).

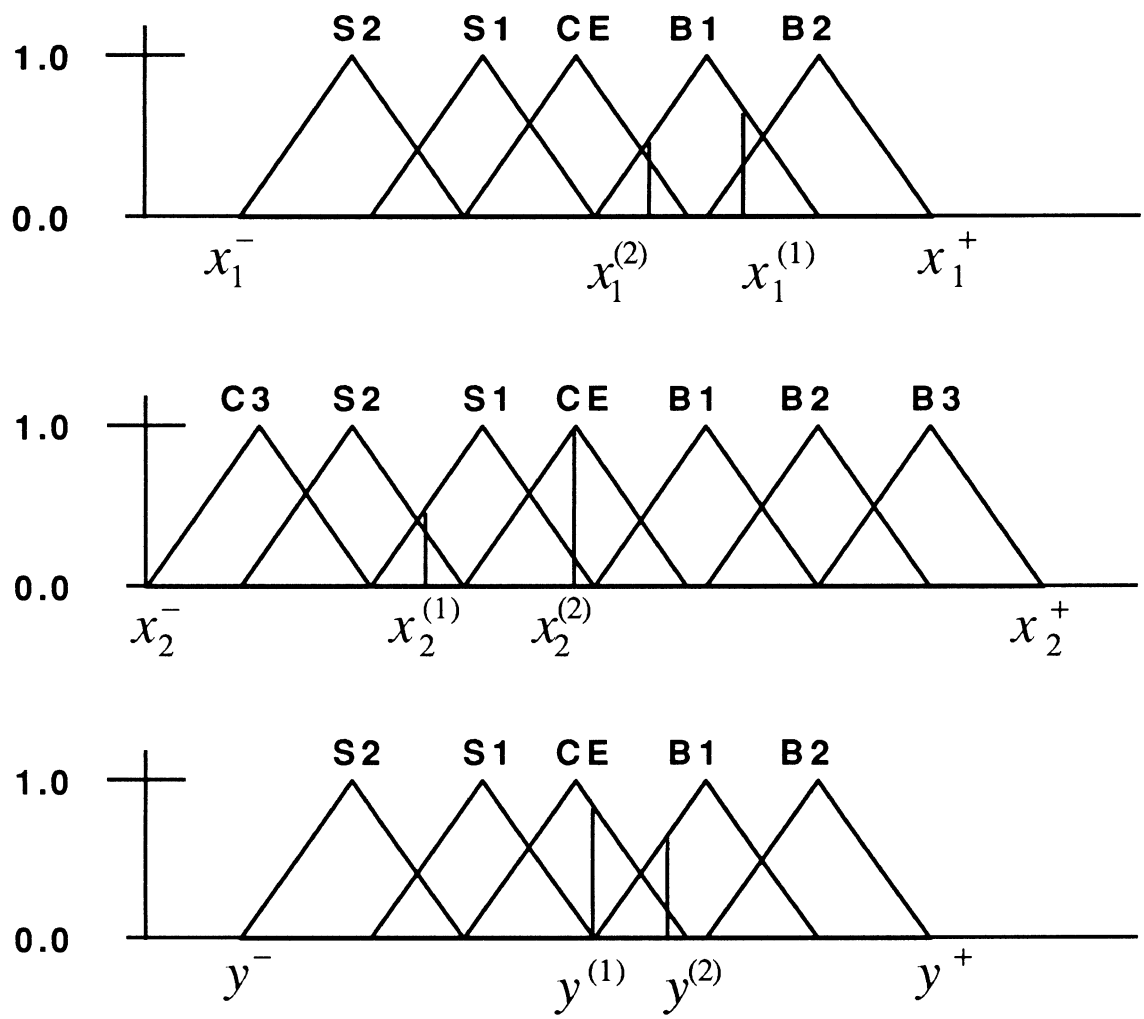

Figure 3.1 Divisions of the input and output spaces into fuzzy regions and the corresponding membership functions.

The shape of each membership function is triangular; one vertex lies at the center of the region and has membership value unit; the other vertices lie at the centers of the two neighboring regions, respectively, and have membership values equal to zero. Of course, other divisions of the domain regions and other shapes of membership functions are possible.

Step 2: Generate Fuzzy Rules from the Given Data Pairs

First, it is necessary to determine the degree of membership of the inputs and outputs in their corresponding regions. For example $x_1^{(1)}$ in Figure 3.1 has degree 0.8 in B1, degree 0.2 in B2, and zero degrees in all other regions. Similarly, $x_2^{(2)}$ in Figure 3.1 has degree 1 in CE, and zero degrees in all other regions.

Then the given inputs and outputs are assigned to the region with maximum degree. For example, $x_1^{(1)}$ in Figure 8 is considered to be B1, and $x_2^{(2)}$ is considered to be CE. Finally, a rule is obtained form one pair of desired input-output data, e.g.,

$(x_1^{(1)}, x_2^{(1)}; y^{(1)}) \Rightarrow [x_1^{(1)}(0.8 \text{ in B1, max}),\ x_2^{(1)}(0.7 \text{ in S1, max});\ y^{(1)}(0.9 \text{ in CE, max}) \Rightarrow$

Rule 1: IF x_1 is B1 and x_2 is S1, THEN y is CE.

$(x_1^{(2)}, x_2^{(2)}; y^{(2)}) \Rightarrow [x_1^{(2)}(0.6) \text{ in B1, max}),\ x_2^{(2)}(1 \text{ in CE, max});\ y^{(2)}(0.7 \text{ in B1, max})] \Rightarrow$

Rule 2: IF x_1 is B1 and x_2 is CE, THEN y is B1.

The rules generated in this way are "and" rules, i.e., rules in which the conditions of the IF part must be met simultaneously in order for the result of the THEN part to occur. For the problems considered in this work, i.e., generating fuzzy rules form numerical data, only "and" rules are required.

Step 3: Assign a Degree to Each Rule

Since there are usually lots of data pairs, and each data pair generates one rule, it is highly probable that there will be some conflicting rules, i.e., rules which have the same IF part but a different THEN part. One way to resolve this conflict is to assign a degree to each rule generated form the data pairs, and accept only the rule from a conflict group that has maximum degree. In this way, not only is the conflict problem solved, but also the number of rules is greatly reduced.

The following product strategy is used to assign a degree to each rule; *D(Rule)* is defined as:

$$D(Rule) = m_A(x_1)m_B(x_2)m_C(y) \qquad (2)$$

In this way, Rule 1 has degree:

$$D(Rule1) = 0.8 * 0.7 * 0.9 = 0.504; \qquad (3)$$

and Rule 2 has degree:

$$D(Rule2) = 0.6 * 1.0 * 0.7 = 0.420. \qquad (4)$$

In practice, there is some a priori information about the data pairs. For example, if an expert checks the given data pairs, the expert may suggest that some are very useful and crucial, but others are very unlikely and may be caused just by measurement errors. A degree can be assigned to each data pair representing belief of usefulness. In this sense, the data pairs constitute a fuzzy set, i.e., the fuzzy set is defined as the useful measurements; a data pair belongs to this set to a degree assigned by the human expert. The degree of Rule 1 can be redefined as:

$$D(Rule) = m_{B1}(x_1)m_{S1}(x_2)m_{CE}(y)m^{(1)} \qquad (5)$$

i.e., the degree of a rule is defined as the product of the degrees of its components and the degree of the data pair which generates this rule. By using this strategy, the environment is totally fuzzy in that everything has a degree of membership. This is important in

practical applications, because real numerical data have different reliabilities, e.g. some real data can be faulty ("wild data"). For good data, the expert assigns higher degrees, and for bad data the expert assigns lower degrees. In this way, human experience about the data is used in a common base as other information. If one emphasizes objectivity and does not want a human to judge the numerical data, the strategy still works by setting all the degrees of data pairs equal to unity.

Step 4: Create a Combined FAM Bank

The form of the FAM bank is shown in figure 3.2. The boxes are filled with fuzzy rules according to the following strategy:

> *A combined FAM Bank is assigned rules from either those generated from numerical data or linguistic rules (assume that a linguistic rule also has a degree which is assigned by the human expert and reflects the expert's belief in the importance of the rule); if there is more than one rule in one box of the FAM Bank, use the rule that has the maximum degree.*

In this way, both numerical and linguistic information are codified into a common framework - the combined FAM Bank. If a linguistic rule is an "and" rule, it fills only one box of the FAM Bank; but, if a linguistic rule is an "or" rule (i.e., a rule for which the THEN part follows if any condition of the IF part is satisfied), it fills all the boxes in the rows or columns corresponding to the regions of the IF part. For example, if the linguistic rule is: *"IF x_1 is S1 or x_2 is CE, THEN y is B2"* for the FAM Bank of Figure 3.2, then the seven boxes in the column of S1 and the five boxes in the row of CE would be filled with B2. The degrees of all the B2's in these boxes equal the degree of this "or" rule.

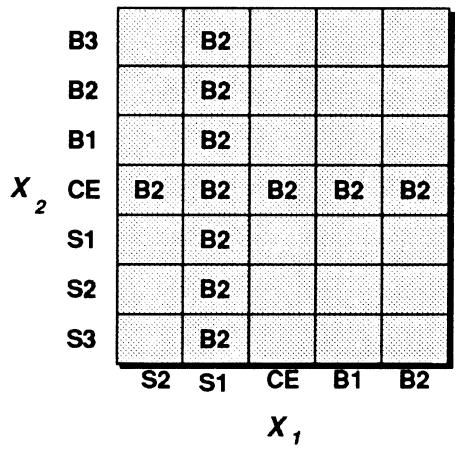

Figure 3.2. We fill seven boxes in the column of S1 and the five boxes in the row CE with B2. The degrees of all B2s in these boxes equal the degree of this "or" rule.

Step 5: Determine a Mapping based on the FAM Bank

The following defuzzification strategy is used to determine the output control y for the given inputs (x_1, x_2): first, all fuzzy rules in the FAM Bank are represented by membership function forms (Figure 3.1 shows an example for Rules 1 and 2); then, for given inputs (x_1, x_2), the antecedents of the i'*th* fuzzy rule are combined using product operations to determine the degree, m^i_{Oi}, of the output control corresponding to (x_1, x_2), i.e.,

$$m^i_{Oi} = m_{I^i_1}(x_1) m_{I^i_2}(x_2) \qquad (6)$$

where O^i denotes the output region of Rule i, and I^i_j denotes the input region of Rule i for the j^{th} component, e.g., Rule 1 gives:

$$m^1_{CE} = m_{B1}(x_1) m_{S1}(x_2);$$

finally, the centroid defuzzification formula determines the output control as follows:

$$y_j = \frac{\sum_{i=1}^{K} m^i_{O^i_j} \overline{y}^i_j}{\sum_{i=1}^{K} m^i_{O^i}} \qquad (7)$$

where $\overline{y^i_j}$ denotes the center of region O^i_j.

If this five step procedure is viewed as a block, then the inputs to the block are the "examples" (desired input-output data pairs) and expert rules (linguistic IF-THEN statements), and the output is a mapping from input space to output space. For control problems, the input space is the state of the plant to be controlled, and the output space is the control applied to the plant. The method essentially "learns" from the "examples" and expert rules to obtain a mapping which, hopefully, has the "generalization" property that when new inputs are presented the mapping continues to give desired or successful outputs. The method can be viewed as a very general *Model-Free Trainable Fuzzy System* for a wide range of control and signal processing problems where "Model-Free" means no mathematical model is required for the problem; "Trainable" means the system learns from "examples" and available expert rules; and, "Fuzzy" denotes the fuzziness introduced into the system by linguistic fuzzy rules, fuzziness of data, etc.

This procedure can be extended to the general multi-input multi-output cases. Steps 1 to 4 are independent of how many inputs and how many outputs there are. In step 5, m_{Oi}^{i} should be replaced with $m_{O_j^i}^{i}$ where j denotes the j^{th} component of the output vector (O_j^i is the region of the Rule i for the j^{th} output component; $m_{O_j^i}^{i}$ is the same for all j). Eq. 7 is then rewritten as:

$$y_j = \frac{\sum_{i=1}^{K} m_{O_j^i}^{i} \overline{y_j^i}}{\sum_{i=1}^{K} m_{O_j^i}^{i}} \qquad (8)$$

where $\overline{y_j^i}$ denotes the center of region O_j^i.

Chapter IV

Application of the Wang-Mendel Methodology in Robot Motion Control

No theory is good unless
one uses it to go beyond
- Andre Guide

4.1 *Robot Motion Case Study*

Robot control is an important branch in control science because of its intelligent aspects, non-linear characteristics, variable parameters, real time computing, and also mathematically non-resolvable inverse kinematics. In order for a robot to perform useful work, the arm must be jointed. It is also necessary to control the path which the end of the arm follows, as opposed to simply controlling the final positions of the joints. Kinematics of a robot arm deals with the movements of the arm with respect to a fixed coordinate system as a function of time. In conventional analysis, the basis of a robot is taken as a reference, and all the other movements are measured from the base reference point (Critchlow 1985).

This approach relies on the exact inverse kinematics model to convert user specified Cartesian trajectory commands to joint set points. These inverse kinematic equations are robot specific and may not be applied to other robot kinematics structures. Furthermore, they form a set of highly coupled non-linear equations and pose considerable computational burden on the host processor. It is not uncommon for the host controller processor to dedicate as much as 80% of its resources in solving the inverse kinematics equations alone.

Some of these aspects have been studied on the basis of Fuzzy Set Theory: sensor based manipulator control (Hirota et al. 1989), joint control, assemblage, grasping force

control, etc. (Lakov 1985, Murakami 1989, Koh et al. 1990, etc.). In addition, Raju and Zhou (1991), as well as Negundani (1992), reported promising results from utilizing FL to replace the computationally intensive inverse kinematic equations. Their research laid the foundation for a robot controller with a much simpler architecture.

Robots in commercial use today utilize a combination of rotary and sliding joints. These joints are the connections between the links of a robot arm or portions of the wrist assembly. Rotary joints allow two links to move only in an angular relation, while sliding joints permit only linear movement. In principle, other joint relationships are permissible, but in practice, only those two types of joints are being used. Robots, therefore, will be capable of performing only those tasks that they have been designed or trained to perform. The performance of such tasks depends on the precise description of the problem. Therefore, for a robot to work properly in a given work configuration, it is important to have a precise description of the work space configuration. However, the latter is not always available due to an incomplete knowledge of the space, or perhaps, due to noise and error in the sensing measurements.

If the location of all joints and links of a robot is known, it is possible to compute the location in space of the end of the arm. This is defined as the direct kinematics problem (Grover et al. 1986). The inverse kinematics problem determines the necessary positions of the joints and links in order to move the end of the robot arm to a desired position and orientation in space. The inverse kinematics problem is much more difficult to solve and there may be more than one solution. To compensate for this problem, a fuzzy logic approach appears to provide the basis for a robot controller with a less complicated architecture and less dependency on the training and the factors affecting its performance.

In order to develop a scheme for controlling the motion of a manipulator it is necessary to develop techniques for representing the position of the arm at points in time.

The robot manipulator will be defined using two basic elements, joints and links. Each joint represents one degree of freedom, and the links are assumed to be the rigid structures that connect the joints.

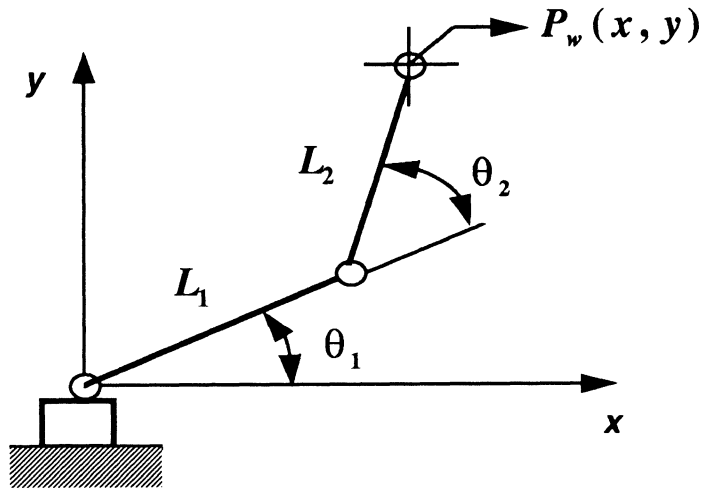

Figure 4.1. A two dimensional 2 degree-of-freedom manipulator. Notice that in this study, the links L_1 and L_2 are considered to have the same length.

Figure 4.1 illustrates the geometric form of the robot arm studied in this work. For the present discussion, the analysis will be limited to the two dimensional case. The position of the end of the arm may be represented in a number of ways. One way is to utilize the two joint angles θ_1 and θ_2. This is known as the representation in "joint" space and it may be defined as:

$$P_j = (\theta_1, \theta_2)$$

Another way to define the arm position is in the "world" space. This involves the use of a Cartesian coordinate system that is external to the robot. The origin of the Cartesian axis is often located in the robot's base. The end-of-arm position would be defined in world space as:

$$P_w = (x, y)$$

The position of the end of the arm in world space can be determined by defining a vector for link1 and another for link 2.

$$r_1 = [L_1 \cos\theta_1, L_2 \cos\theta_2] \quad (1)$$

$$r_2 = [L_2 \cos(\theta_1 + \theta_2), L_2 \sin(\theta_1 + \theta_2)] \quad (2)$$

Vector addition of (1) and (2) yields the coordinates x and y of the end of the arm in the workspace:

$$x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \quad (3)$$

$$y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \quad (4)$$

In many cases it is more important to be able to derive the joint angles given the end-of-arm position in world space. The typical situation is where the robot's controller must compute the joint angles required to move its end-of-arm to a point in space defined by the point's coordinates. For the two link manipulator, there are two possible configurations for reaching the point (x,y), as shown in figure 4.2.

In this example, it is assumed that θ_2 is positive as shown in figure 4.2. Using the trigonometric identities,

cos(A+B)=cosAcosB-sinAsinb

sin(A+B)=sinAcosB+sinBcosA

equations (3) and (4) can be re-written as:

$$x = L_1 \cos\theta_1 + L_2 \cos\theta_1 \cos\theta_2 - L_2 \sin\theta_1 \sin\theta_2$$

$$y = L_1 \sin\theta_1 + L_2 \sin\theta_1 \cos\theta_2 + L_2 \cos\theta_1 \sin\theta_2$$

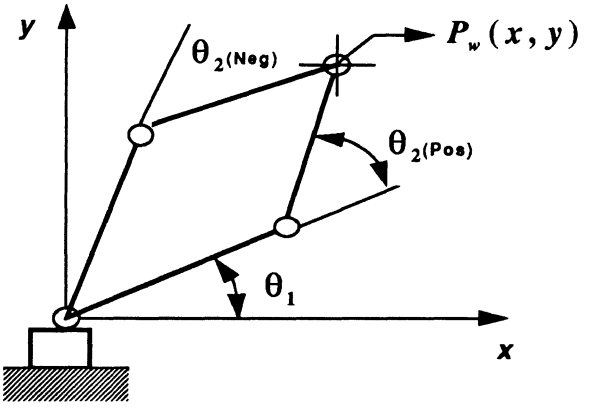

Figure 4.2. The arm at point (x.y), indicating two possible configurations to achieve the position. The robot arm can move to the final coordinate position following one of the two patterns shown in the picture. In this study only cases where θ_2 is positive are taken into account

squaring both sides and adding the two equations yields:

$$\cos\theta_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2} \tag{5}$$

Defining α and β and φ as in figure 4.3, it is possible to obtain:

$$\tan\alpha = \frac{L_2 \sin\theta_2}{L_2 \cos\theta_2 + L_1} \tag{6}$$

$$\tan\beta = \frac{y}{x} \tag{7}$$

assuming $L_1 = L_2$ and with the use of the *law of cosines* in figure 4.3:

$$\cos\varphi = \frac{2L^2 - x^2 - y^2}{2L^2} \text{ , and} \tag{8}$$

$$\theta_2 = 180 - \varphi \tag{9}$$

and finally with equations (7), (8), and (9),

$$\theta_1 = \beta - \alpha \tag{10}$$

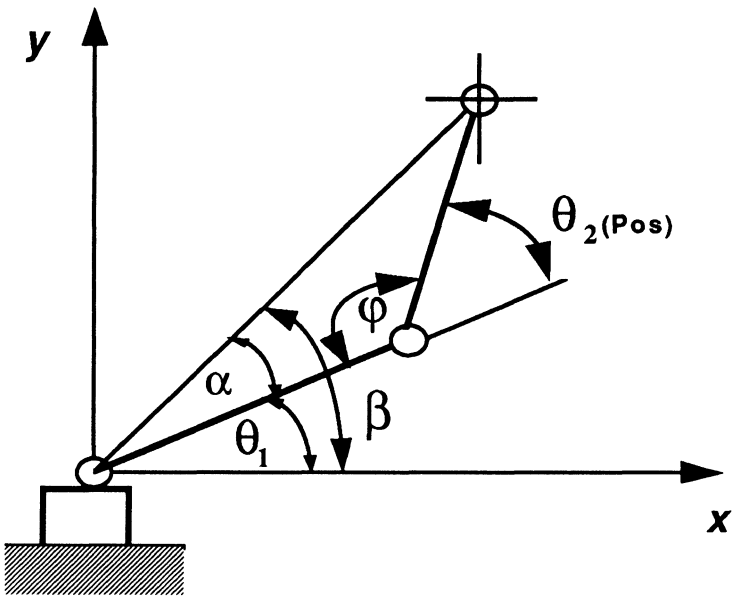

Figure 4.3. Solving for the joint angles.

Equations (9) and (10) provide the angle configuration required to determine the coordinate (x,y), which in turn defines the position of the robot arm in world space through equations (3) and (4).

Figure 4.4 illustrates the criteria used to add orientation to the motion of the robot arm. The coordinate (x,y) defines the center of a circle of radius r. In addition, a line of slope m (by line, it is meant the robot arm link following a specific linear motion of slope m), passes through the same coordinate (x,y). The line and the circle will intersect in two points, given a and b.

As it can be observed in figure 4.4, the robot arm can move to either point a or point b. A displacement towards a signifies an *increment* in both the x and y directions. Meanwhile, a displacement towards b means a *decrement* in both the x and y directions. In this case, only increments, i.e., overall positive change in x and y, will be considered. The amount of displacement is in direct relationship to the radius value. This characteristics allows for very fine increments that facilitate a motion "almost" linear.

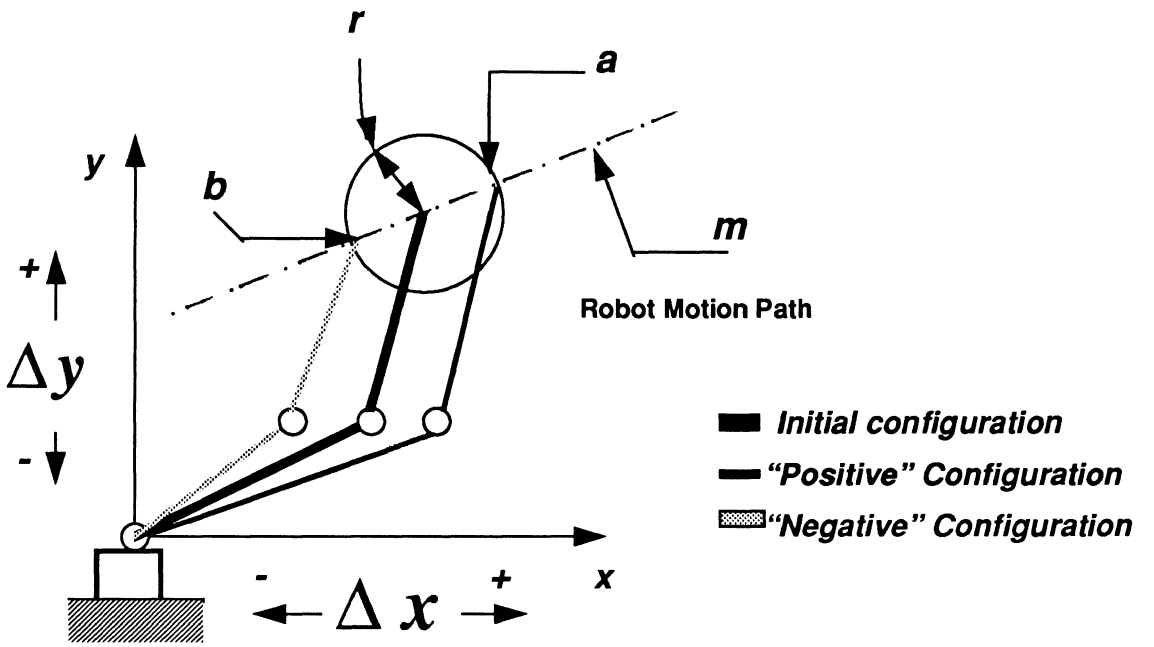

Figure 4.4. For a given radius *r*, and a defined slope-motion *m*, the robot arm can move to either location *a* or location *b*. This work will consider only positive displacements for both *x* and *y*. In order to keep the display clear, notice that there is no scale in the robot arm links.

Taken into account the above considerations, the problem is setup as follows:

1. The link length is equal to the unity of length. Both links are equal.
2. The displacement radius is 0.05 of the unity of length.
3. The maximum reach in the x direction is coordinate (2,0).
4. The maximum reach in the y direction is coordinate (0,2)
5. The work envelope is defined in the first quadrant as displayed in Figure 4.3.
6. The initial angle and coordinate configurations are randomly generated.
7. The path followed by the robot arm is defined through a positive angle measured from the horizontal and in the range 0 to 100 degrees ($m = \tan^{-1}[\frac{y}{x}]$, where x and y are randomly generated.)
8. With this data, the program then applies equations (5) through (10) to define first the final angle configuration, mandated by the value of the slope and the displacements in

x and y; and then, once the angle configuration is known, the coordinate configuration is computed.

9. Finally, the data is saved in a general data base file. This data base file becomes the training data for the fuzzy logic controller (Refer to Appendix B).
10. The total number of data sets generated is 1000 and the format of each record is:

slope

x_initial	y_initial	θ_1_initial	θ_2_initial
x_final	y_final	θ_1_final	θ_2_final

4.2 *Development of the Fuzzy Logic Controller*

As previously stated the first step in the development of any FLC is the definition of the inputs, the outputs, and their correspondent membership functions. This procedure becomes critical as it is here that the developer must clearly understand what are the relevant factors affecting the behavior of the problem under control and what type of response are those factors generating.

In this specific "sub" problem, control of the robot arm will be defined through its initial and final angle configurations. The robot arm will have two stages. In stage one, the robot arm is positioned at space location (x, y). This space location is randomly generated within the arm's work envelop as indicated in 3 and 4. The "joint" space value is determined by the θ_1 and θ_2 angle values.

In stage two, the robot arm has undergone displacement and it is located at (x_1, y_1). Displacement in the horizontal and vertical directions, as well as the angle-inclination (slope) of the path, are the criteria used to produce movement of the arm. For this specific control problem, the slope of the path is always positive; therefore, the displacement in the horizontal and vertical directions are always both positive or negative;

that is, if there is a positive increment from x to x_1, then is also a positive increment from y to y_1 (which is expected because of the always positive path slope.)

In this fashion, the inputs are the initial *x coordinate*, the initial *y coordinate*, and the *slope* of the robot arm trajectory. In the previous section, it was stated that the maximum reach of the robot arm corresponded to a totally flat arm on the horizontal, (2,0), or a totally erect arm on the vertical, (0,2). As a result, the range of possible values for the inputs x and y is 0 to 2.

The input *slope* is determined to always be positive and between 0 and 100 degrees; therefore, the fuzzy range of values for the *slope* covers all possible degrees between 0 and 100.

The beginning of this chapter indicated that a way to determine the final location of the robot arm is to utilize the two joint angles θ_1 and θ_2. This is known as the representation in "joint" space. The elected outputs for the FLC are the changes observed in the angles θ_1 and θ_2. Since the slope of the trajectory is known, and the radius of displacement is defined, the variation in angle configuration can be obtained.

Knowledge of the changes recorded on each angle will allow the controller to move to a new location by varying the total angle configuration by a number of degrees. In order to obtain these data, the system simply finds the difference between the initial and final angle configurations. The range for the difference between initial θ_1s and final θ_1s was found to be from -10 degrees to 0 degrees. The range found for θ_2s was from 0 degrees to 12.0 degrees.

Table 1 contains the upper and lower limit values for the inputs and outputs. In addition, Table 1 contains the number of fuzzy regions for each variable of interest. It should be noted that each domain interval was divided into 2N+1 regions, with N adopting different values for each different variable. For this case, the vector value of N is

{2,2,2,3,3} corresponding to the x coordinate, the y coordinate, the slope, $\Delta\theta_1$, and $\Delta\theta_2$ respectively.

Table 1. Fuzzy ranges for all inputs and outputs of the system.

		Lower Limit	Upper Limit	Fuzzy Regions
Inputs	*x* Coordinate	0.0	2.0	5
	y Coordinate	0.0	2.0	5
	Slope	0.0	100	5
Outputs	$\Delta\theta_1$	0.0	12.0	7
	$\Delta\theta_2$	-30.0	0.0	7

It is now valid to question how the N vector was determined. In previous chapters it has been indicated that the major drawback of fuzzy logic is its lack of mathematical foundations. Defining the number of fuzzy values for each variable is a clear example of the previous statement. In the present work, multiple values for N were tested. Results almost identical to the desired outputs were obtained for N values of at least 2 and no greater than 4 per variable. There is no specific justification as why this granularity was selected for each input and output (granularity refers to the number of fuzzy regions defined for each fuzzy variable). Trial and error suggested that the proposed combination was satisfactory for this specific control problem. Figure 4.5 shows the degree of membership functions for the present case study.

The domain interval of the *x* coordinate is divided into five regions (N=2, min=0.0,max=2.0), the domain interval of the *y* coordinate is divided into five regions (N=2, min=0.0, max=2.0), and the domain interval of the *slope* is divided in five regions (N=2, min=0.0, max=100.0).

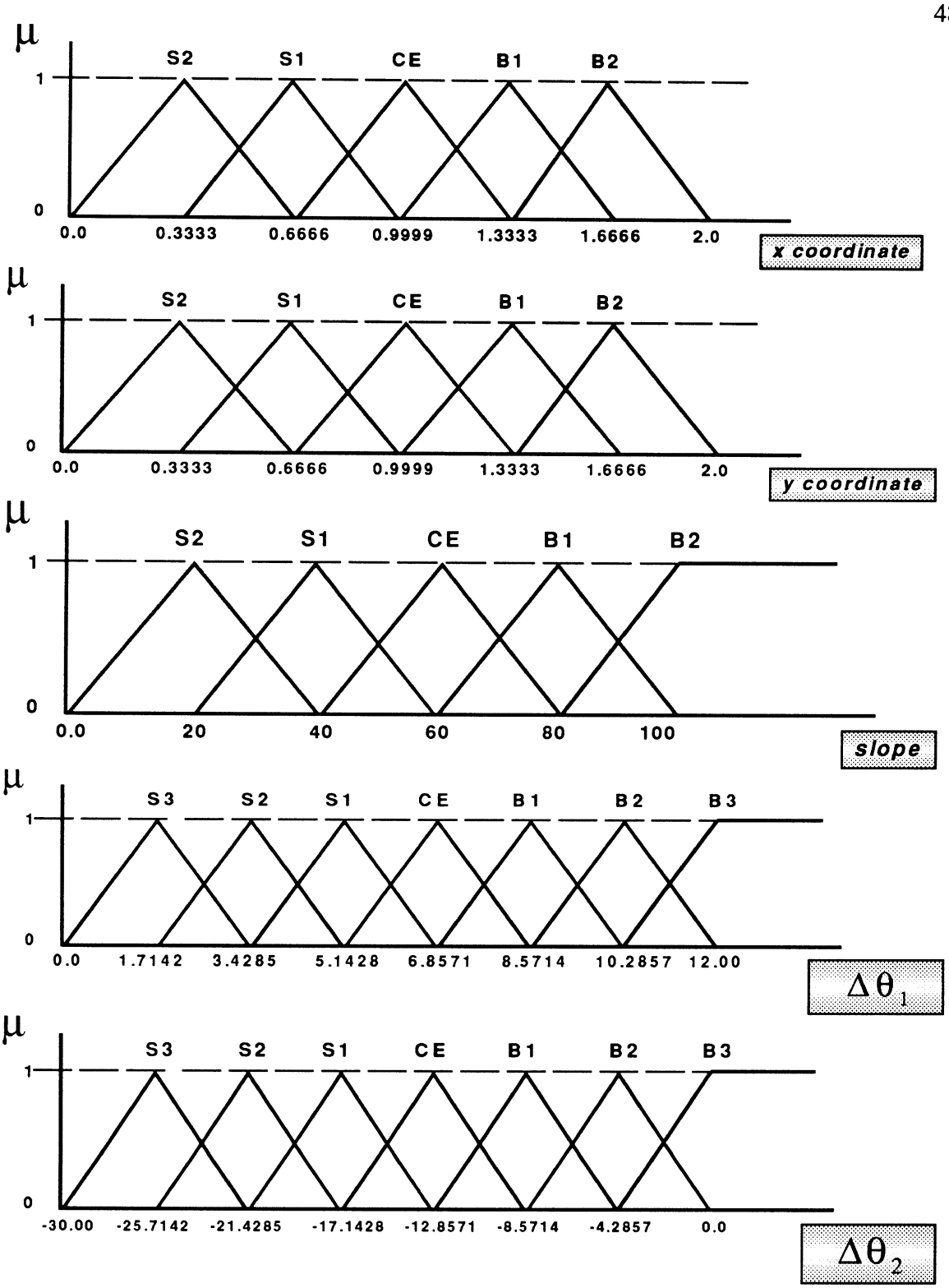

Figure 4.5. The degree of membership functions for the three systems inputs, *x* coordinate, *y* coordinate,

Also, the domain interval of $\Delta\theta_1$, is divided into seven regions (N=3, min=0.0, max=12.0), and the domain interval of $\Delta\theta_2$ is divided into seven regions (N=3, min=-30.0, max=0.0). The shape of each membership function is triangular; one vertex lies at the center of the region and has membership value unity; the other two vertices lie at the centers of the two neighboring regions, respectively, and have membership values equal to zero. Of course, other divisions of the domain territory and other shapes of membership functions are possible. Nonetheless, the present configuration was satisfactory for the controller.

4.3 *The Fuzzy Machine*

Once the degree of membership functions have been defined for each input and output, each fuzzy set is denoted by SN (Small N),..., S1 (Small 1), CE (Center), B1 (Big 1),...,BN (Big N). The degree of membership functions for the three inputs and the two outputs will transform the measured (crisp) values of Section 4.3 into degrees of membership in fuzzy sets. This is done as follows:

For each fuzzy set, a degree of membership is defined. In reference to the chart corresponding to the x coordinate in Figure 4.5, for example, it is observed that the linguistic variable S1 can be divided into two straight lines crossing at (0.3333,1.0), that is, the center value of the fuzzy set S1. Therefore, since the two points of each line are given, the equation of each one is computed. Figure 4.6 illustrates this process.

slope, and the two system outputs, $\Delta\theta_1$ and $\Delta\theta_2$, transform the measured (crisp) values into degrees of membership in fuzzy sets.

Point a = (0.0000,0.0000)
Point b = (0.3333,1.0000

$$\text{slope} = m_1 = \frac{1.0000}{0.3333} = 3.0000$$

$\therefore line_1 = y = 3x$, which corresponds to the first degree of membership function

Furthermore,

Point b = (0.3333,1.0000)
Point a = (0.6666,0.0000)

$$\text{slope} = m_2 = \frac{-1.0000}{0.3333} = -3.0000$$

$\therefore line_2 = y = -3x + 2$, which correspond to the second degree of membership function.

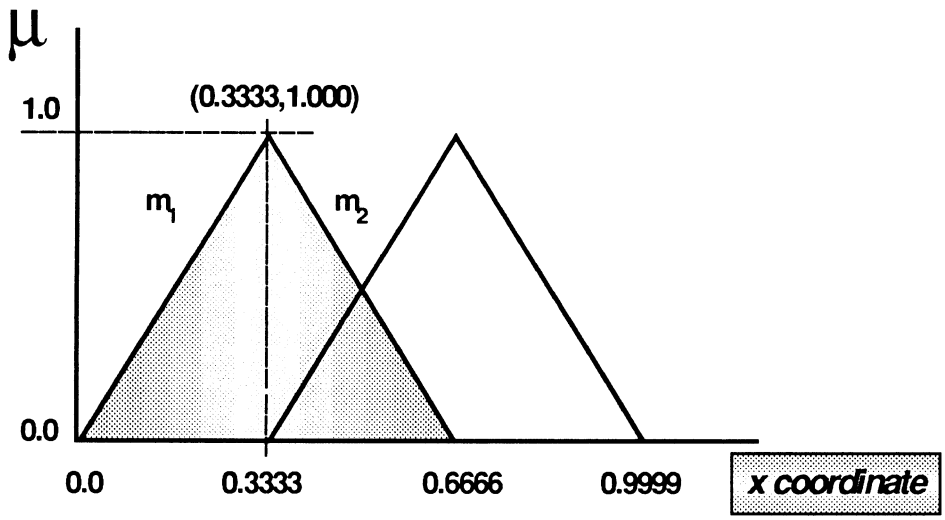

Figure 4.6. The Fuzzy set corresponding to the linguistic variable S1, can be divided into two regions: to the left of the vertical line, the degree of membership corresponds to the function m=3x. The right of the vertical line, the degree of membership corresponds to y=-3x+2. Similar analysis, is performed for the rest of the linguistic variables.

The above procedure is then applied to the rest of the linguistic variables for all inputs and outputs. After assigning each region a fuzzy membership function, the variables of the system look like *x* coordinate in Table 2. Appendix C presents similar tables for the other variables under consideration.

Once all the degree of membership functions have been declared for all the inputs and the outputs, the fuzzification procedure will transform the crisp data of section 4.3 into fuzzy values.

Table 2.

Each input and output is divided into 2N+1 fuzzy regions. The degree of membership function is found for each region, and a fuzzy value is assigned.

Interval	Membership Function	Fuzzy Value
$0.0000 \le x < 0.3333$	$y = 3x$	$S2$
$0.3333 \le x < 0.6666$	$y = -3x + 2$	$S2$
$0.3333 \le x < 0.6666$	$y = 3x - 0.999999$	$S1$
$0.6666 \le x < 0.9999$	$y = -3x + 3$	$S1$
$0.6666 \le x < 0.9999$	$y = 3x - 2$	CE
$0.9999 \le x < 1.3333$	$y = -3x + 3.99999$	CE
$0.9999 \le x < 1.3333$	$y = 3x - 3$	$B1$
$1.3333 \le x < 1.6666$	$y = -3x + 5$	$B1$
$1.3333 \le x < 1.6666$	$y = 3x - 3.99999$	$B2$
$1.6666 \le x < 2.0000$	$y = -3x + 6$	$B2$

The following example corresponds to the first data set of the data base file, as presented in Appendix D. For illustration purposes, this data set will be used throughout the remaining of this chapter. The original, crisp data set is:

24.000000				which is:	*slope*
1.649381	0.498457	-11.163102	59.066528		x_initial y_initial θ_1_initial θ_2_initial
1.651462	0.548413	-13.696254	61.023022		x_final y_final θ_1_final θ_2_final

In order to fuzzify this data, each member of the set is assigned a degree of membership on its corresponding fuzzy region. This is illustrated in figure 4.7. Because of the overlapping between fuzzy values, the crisp number is most likely to "hit" two degrees of membership functions (actually, depending on the granularity and the shape of the fuzzy regions, two or more functions can be hit). When this is the case, the maximum degree is kept. The other degree is ignored. Again, there is no mathematical proof for

this decision. The minimum value could have been chosen as well. Once the maximum degree is assigned to the crisp data, notice that the value has now obtained a degree in the range [0,1] and its linguistic "fuzzy" value corresponds to the fuzzy region of maximum degree. A numerical illustration for the value corresponding to the slope, would look like this:

slope = 24,

this *slope* value belongs to the interval $20 \leq slope < 40$. Therefore, the fuzzy regions S2 and S1 are candidates for fuzzification. The membership functions corresponding to those regions are:

$$20 \leq slope < 40 \qquad y = -0.05slope + 2 \qquad \text{S2}$$
$$20 \leq slope < 40 \qquad y = 0.05slope - 1 \qquad \text{S1}$$

substituting the value *slope* = 24, for the above equations, the following degrees of membership are obtained:

$20 \leq slope < 40$ slope=24 y=0.80
$20 \leq slope < 40$ slope=24 y=0.20

Based on the maximum degree of membership rule, the fuzzy value assigned to a slope value of 24 degrees is S2 with a degree of membership of 0.8.

Following similar steps for the *x*_initial, *y*_initial, θ_1_initial, θ_2_initial, *x*_final, *y*_final, θ_1_final, and θ_2_final values, a rule is obtained for the desired input output set, e.g.,

B2 S2 *S2*
S3 B3
0.948143 0.504629 *0.800000* 0.522328 0.543485 **(i)**

which would read:

IF *x_initial* is B2 and *y_initial* is S2 and *slope* is S2, **THEN** $\Delta\theta_1$ is S3 and $\Delta\theta_2$ is B3

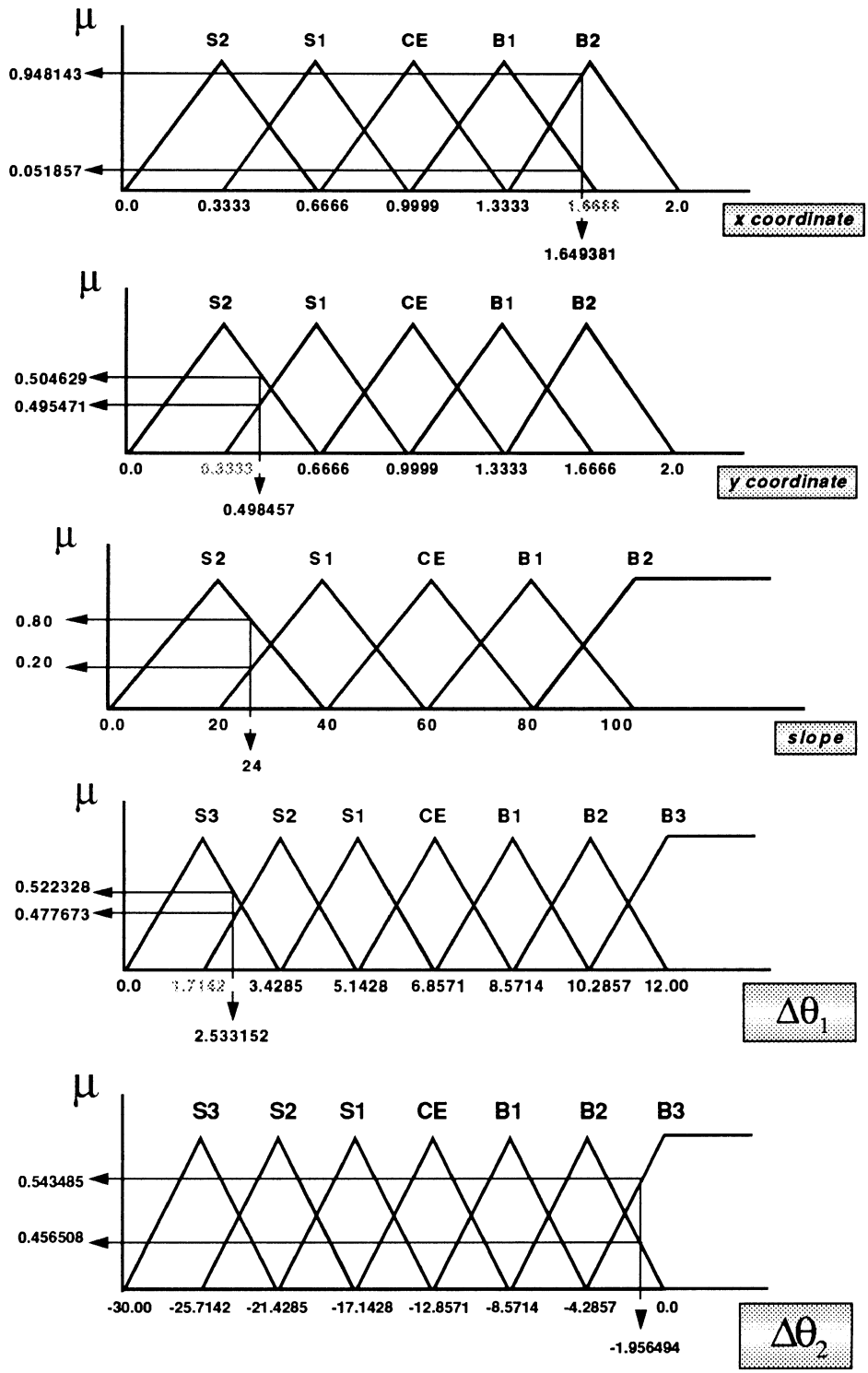

Figure 4.7. The degree of membership values are obtained by mapping the crisp input values with the corresponding degree of membership functions. The value that prevails is that corresponding to he highest degree of membership.

Taking a closer look at (i), it can be observed that the format of the rule file presented in Appendix D has a slight modification from the original data base file. The way this file should be read is:

x_initial y_initial slope
$\Delta\theta_1$ $\Delta\theta_2$
degrees of membership

In addition, $\Delta\theta_1$ and $\Delta\theta_2$ correspond to the differences between the initial values for θ_1 and θ_2, and the final values for θ_1 and θ_2. These differences are first computed. Then, the domain interval is defined, and finally the fuzzy regions are determined.

Reviewing the rules obtained from the crisp data (for each data set, one rule), it becomes apparent that there exists rules with the same input values, but different output values. This situation exits each time that the input values have the same fuzzy value, that is, data from different sets happen to meet the maximum degree of membership rule for the

same linguistic variable. Although it holds true for the specific rule, it does not assure that a different rule may have the same input values but with a higher degree of membership. In order to take care of this situation, the Wang-Mendel methodology assigns a degree to each conflicting rule and it accepts only the rule that has the maximum degree. This not only takes care of the conflict, but it also reduces dramatically the number of rules for the final controller. For example, the rule developed above, has a conflict with another rule of the controller:

```
B2 S2 S2
S3 B3
0.948143 0.504629 0.800000 0.522328 0.543485

B2 S2 S2
S3 B3
0.282837 0.600000 0.850000 0.662187 0.617012
```

The Degree of the first rule is:

Degree(Rule1) = (0.948143)(0.504629)(0.800000)(0.522328)(0.543485) = 0.108659
Degree(Rule1) = (0.282837)(0.600000)(0.850000)(0.662187)(0.617012) = 0.058936

Since the degree of Rule1 is bigger that the one corresponding to Rule2, Rule2 is eliminated and only Rule1 remains in the rule base until a new conflict is found.

Moreover, and what is more important, this step removes those rules that introduce ambiguities. In other words, if the IF part of the rules happens to be the same, although the THEN part is not, the methodology still assigns a corresponding degree to each rule. Those rules that have a higher degree, that is, that contain the other rules, are kept in the rule base. The contained rules are then eliminated. The data sets below illustrate this characteristic.

B1 S1 B1
S1 B2
0.697568 0.545775 0.900000 0.639108 0.623849

B1 S1 B1
CE B1
0.961674 0.692493 0.750000 0.620386 0.566062

Degree(Rule1) = (0.697568)(0.545775)(0.900000)(0.639108)(0.623849) = 0.137108
Degree(Rule2) = (0.961674)(0.692493)(0.750000)(0.620386)(0.566062) = 0.175400

Since the degree of Rule2 is bigger that the one corresponding to Rule1, Rule1 is eliminated and only Rule2 remains in the rule base until a new conflict is found.

Appendix E presents a listing of the final 111 control rules for this case study. This rule base represents 10% of the original set of rules created during the fuzzification step. Like in most control problems, there is the existence of redundant data that is not required for the correct performance of the system. In this specific case, the method used to eliminate redundant rules afforded proper generalization of those rules remaining for control of future data.

Once the conflicting rules have been eliminated according to the degree value of the rules, the next step in the development of the fuzzy machine involves the arrangement of the obtained rules in the form of a Fuzzy Associative Memory, FAM. The FAM Bank is a mapping of the inputs and outputs into one common framework and its architecture is shown in Figure 4.8.

In this case study, each control input has five fuzzy numbers or fuzzy regions. This would define a total of 125 possible control outputs over a three-dimensional matrix with 5 cells per side. Since there are two outputs, the system has two three-dimensional matrices; therefore, the 111 rules will be distributed in a total of 10 bank matrices as depicted in Figure 4.9. Each matrix entry can equal one of seven $\Delta\theta_i$ fuzzy set values or equal no fuzzy set at all (the fuzzy range for the output variables is divided in seven fuzzy areas).

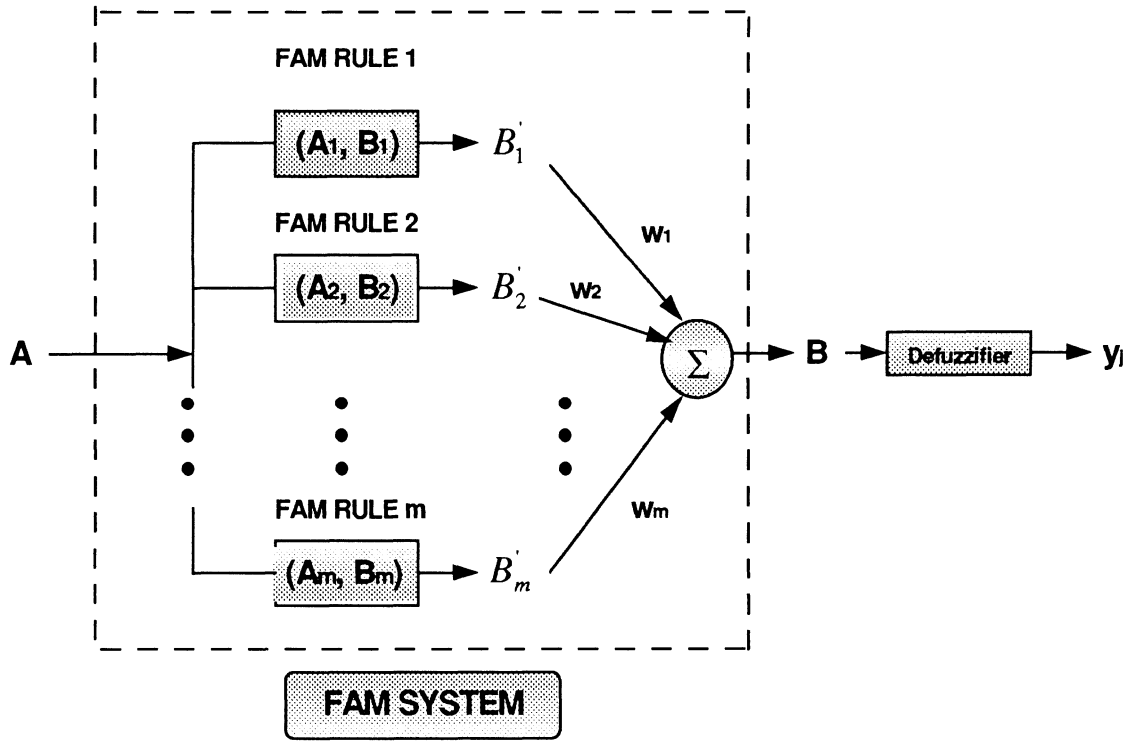

Figure 4.8. The FAM system maps inputs form the fuzzy set A into the fuzzy set B. The user can then defuzzify the output fuzzy set B to yield a crisp, exact output data.

Since a FAM rule is a mapping function, there is exactly one output $\Delta\theta_i$ value for every set of input data. So, the 111 entries per FAM output represent a subset of the total 125 (5^3) possible three antecedent FAM rules. In practice some of the entries are blank. The inference procedure, then, activates in parallel the antecedents of all affected FAM rules. The pulse nature of the inputs picks off single fit values from the quantizing fuzzy-set variables generating the fuzzy output. Through the deffuzification technique, this value is then converted into a crisp result. This procedure is explained in detail in the next section.

4.4 *Testing the Fuzzy Controller*

Section 4.4 explained the development of the fuzzy controller based on 1000 data set samples obtained in section 4.3. This was done by applying the first four steps of the Wang-Mendel algorithm. This section explains the definition of a mapping based on the combined fuzzy rule base or FAM.

In order to test the accuracy of the controller, another set of 1000 data points was generated as in section 4.3. The output control $\Delta\theta_1$ and $\Delta\theta_2$ was obtained by combining the antecedents of the *i*th fuzzy rule using the product operation to determine the degree of the output control corresponding to the specific input triplet. A data set of the testing file is selected to walk through the deffuzification strategy below.

80.000000

0.430020 0.688921 -4.916889 129.366104

0.430645 0.738917 -8.014722 132.085236

Form this data, the input values slope, x initial, and y initial, will be fuzzified in order to determine their fuzzy values. This is shown in figure 23. Mathematically, this is:

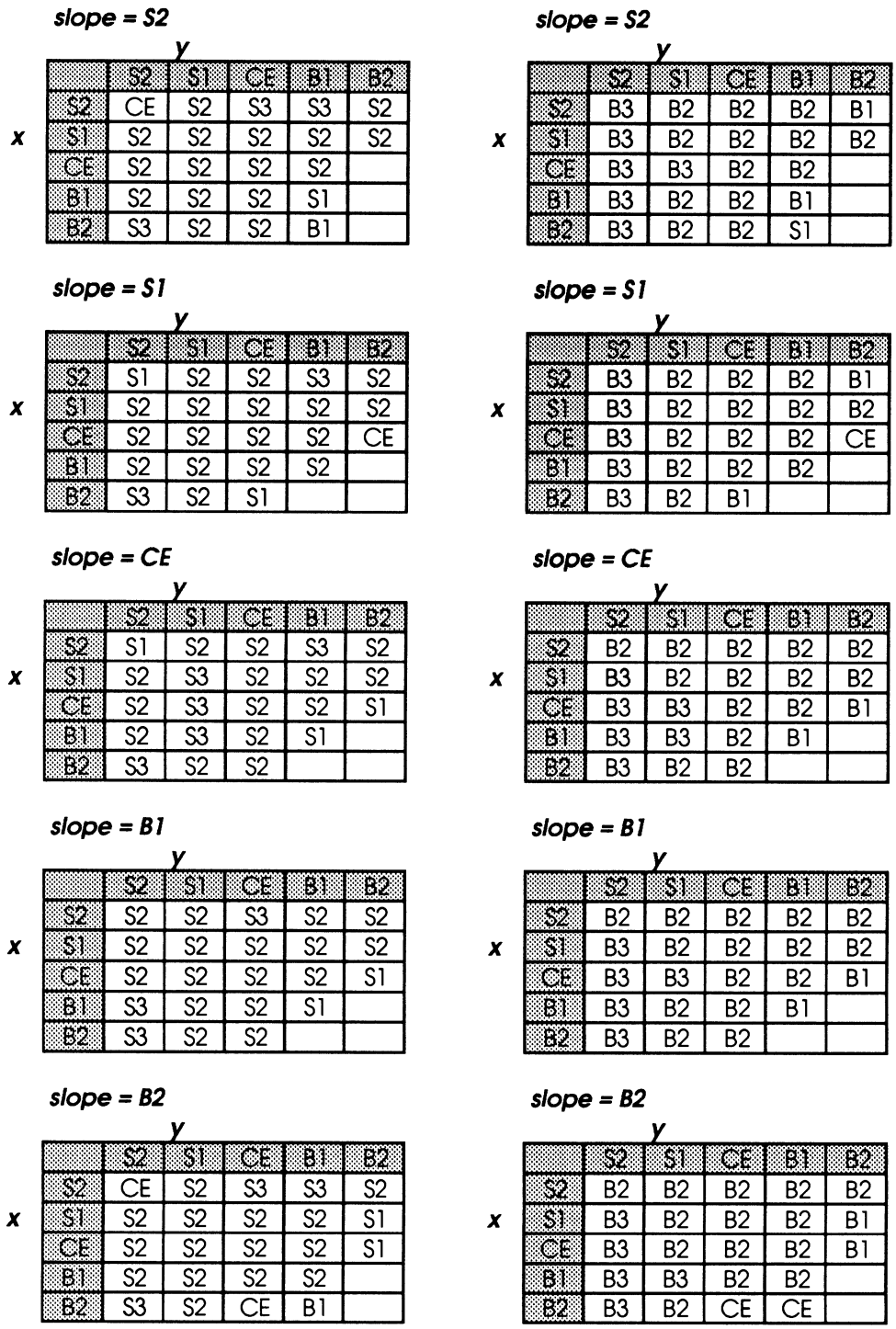

Figure 4.9. The FAM Bank for the robot motion case study. There are three inputs and two outputs in the system. For all purposes, the FAM would look like a cube of rules for each output. The sides of the cube are x-initial, y_initial, and the slope.

x initial = *0.4300290*:

$$0.3333 \le x < 0.6666 \quad \therefore \quad y = -3x + 2.00 = 0.709940 \quad S2$$
$$y = 3x - 0.999 = 0.290160 \quad S1$$

y initial = *0.688921:*

$$0.6666 \le y < 0.9999 \quad \therefore \quad y = -3x + 3.00 = 0.933237 \quad S1$$
$$y = 3x - 2.000 = 0.066763 \quad CE$$

slope = *80.000:*

$$80.00 \le slope < 100 \quad \therefore \quad y = -0.05x + 5 = 1.0 \quad B1$$
$$y = 0.05x - 4 = 0.0 \quad B2$$

Given that the FAM gets activated in parallel, the total possible number of rules that will be fired at the same time is 8, or 2^3, corresponding to three inputs and two fuzzy values hit at the same time. The antecedents of the 8 rules are:

x initial	y initial	slope
S2	S1	B1
S2	CE	B1
S2	CE	B2
S1	CE	B2
S1	S1	B1
S1	S1	B2
S2	S1	B2
S1	CE	B1

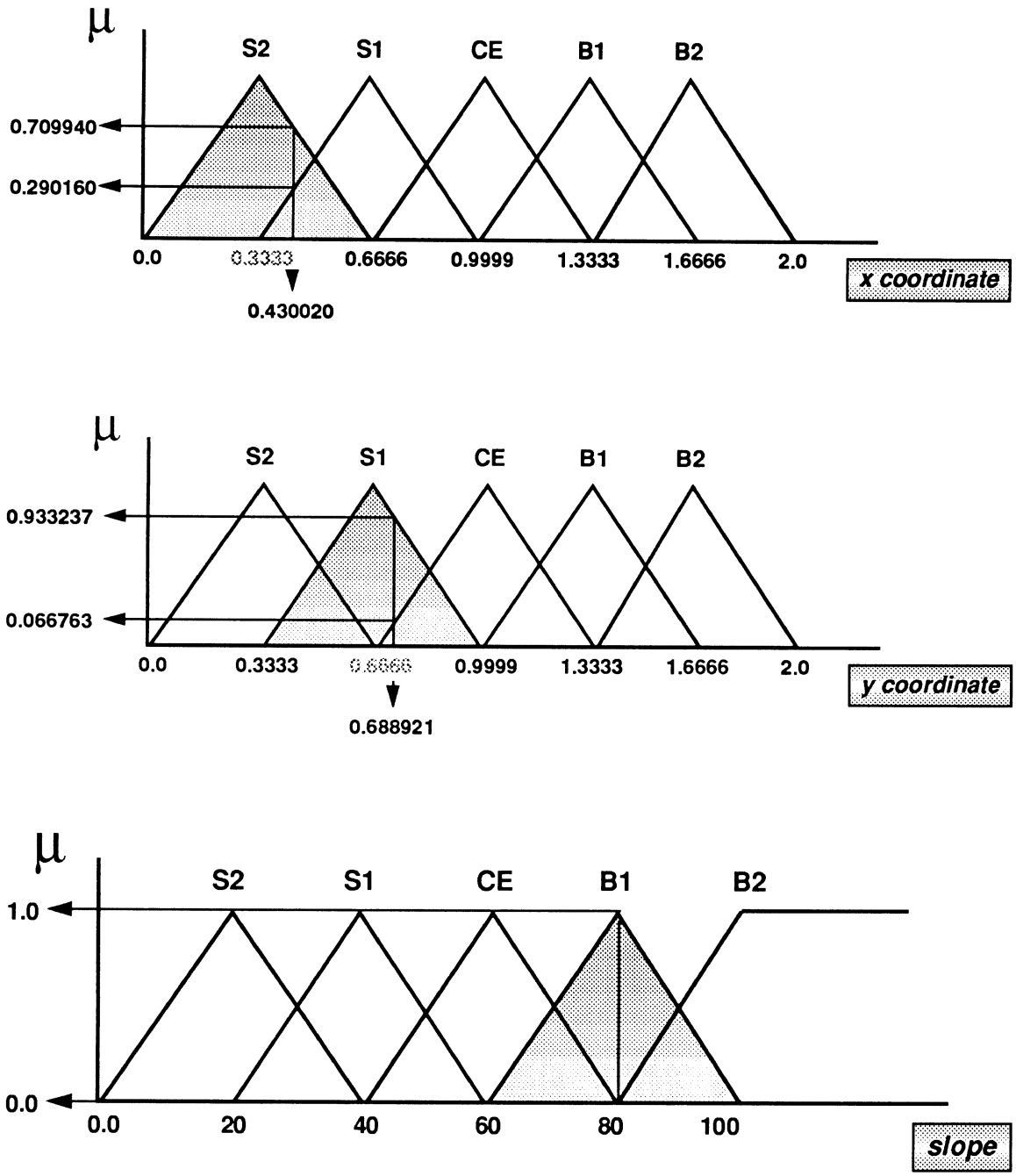

Figure 4.10. When the fuzzy controller receives new information, the inputs acting on the system will be fuzzified according to the degree of membership functions defined at the design stage. For the case study, notice that the inputs x coordinate, y coordinate, and slope, each "hit" two fuzzy values at a time (in reality, this will depend on the granularity of the system). Since each input can assume two values, the maximum degree of membership does not apply anymore, there will be a total of 8 rules that will get fired. These 8 rules will contribute to the final response control of the system.

These antecedents will be then mapped to their correspondent control actions in the FAM Bank. Referring to Appendix E, the antecedents above listed correspond to the following FAM entries:

S2 S1 B1
S2 B2
0.903761 0.597002 0.950000 0.902675 0.597429 Rule 18

S2 CE B1
S3 B2
0.722694 0.652132 0.700000 0.711064 0.807007 Rule 55

S2 CE B2
S3 B2
0.633215 0.636541 0.850000 0.521729 0.792331 Rule 59

S1 CE B2
S2 B2
0.873506 0.910627 0.500000 0.597705 0.732709 Rule 4

S1 S1 B1
S2 B2
0.503499 0.967721 0.700000 0.827206 0.510169 Rule 62

S1 S1 B2
S2 B2
0.912522 0.808292 0.750000 0.952327 0.512612 Rule 107

S2 S1 B2
S2 B2
0.695402 0.678764 0.700000 0.924284 0.580239 Rule 63

S1 CE B1
S2 B2
0.884166 0.744712 1.000000 0.611782 0.754855 Rule 16

Once all fired rules have been defined, the defuzzification strategy to obtain the output control $\Delta\theta_1$ and $\Delta\theta_2$ form the given inputs (*x, y, slope*) is calculated using equations (3.6) and (3.7) as listed in Chapter III. This gives:

$$\Delta\theta_1 = \frac{\sum_{i=1}^{K} m_{O^i}^i \overline{y^i}}{\sum_{i=1}^{K} m_{O^i}^i} \quad \text{(i)} \qquad \text{and} \qquad \Delta\theta_2 = \frac{\sum_{i=1}^{K} m_{O^i}^i \overline{y^i}}{\sum_{i=1}^{K} m_{O^i}^i} \quad \text{(ii)}$$

where $\overline{y^i}$ denotes the center value of each fuzzy region belonging to the output variables. Again, the center of a fuzzy region, is defined as the point that has the smallest absolute value among all the

points at which the membership function for this region has membership value equal to one. In this way, $\overline{y^i}$ would be:

$$\overline{y^i}_{\Delta\theta_1} = \{1.714285, 3.428571, 5.142857, 6.857142, 8.571428, 10.285714\}$$

$$\overline{y^i}_{\Delta\theta 2} = \{-25.714285, -21.428571, -17.142857, -12.857142, -8.571428, -4.285714\}$$

This center values can also be observed in Figure 4.7. Developing equations (i) and (ii), for the current data set, the following control outputs are obtained:

$$\Delta\theta_1 = \frac{m_{S2}m_{S1}m_{B1}y_{S2} + m_{S2}m_{CE}m_{B1}y_{S3} + m_{S2}m_{CE}m_{B2}y_{S3} + m_{S1}m_{CE}m_{B2}y_{S2}}{m_{S2}m_{S1}m_{B1} + m_{S2}m_{CE}m_{B1} + m_{S2}m_{CE}m_{B2} + m_{S1}m_{CE}m_{B2}} + \frac{m_{S1}m_{S1}m_{B1}y_{S2} + m_{S1}m_{S1}m_{B2}y_{S2} + m_{S2}m_{S1}m_{B2}y_{S2} + m_{S1}m_{CE}m_{B1}y_{S2}}{m_{S1}m_{S1}m_{B1} + m_{S1}m_{S1}m_{B2} + m_{S2}m_{S1}m_{B2} + m_{S1}m_{CE}m_{B1}}$$

$$\Delta\theta_2 = \frac{m_{S2}m_{S1}m_{B1}y_{B2} + m_{S2}m_{CE}m_{B1}y_{B2} + m_{S2}m_{CE}m_{B2}y_{B2} + m_{S1}m_{CE}m_{B2}y_{B2}}{m_{S2}m_{S1}m_{B1} + m_{S2}m_{CE}m_{B1} + m_{S2}m_{CE}m_{B2} + m_{S1}m_{CE}m_{B2}} + \frac{m_{S1}m_{S1}m_{B1}y_{B2} + m_{S1}m_{S1}m_{B2}y_{B2} + m_{S2}m_{S1}m_{B2}y_{B2} + m_{S1}m_{CE}m_{B1}y_{B2}}{m_{S1}m_{S1}m_{B1} + m_{S1}m_{S1}m_{B2} + m_{S2}m_{S1}m_{B2} + m_{S1}m_{CE}m_{B1}}$$

Notice that the above equations multiply the degree of membership of each input in the rule by the center value corresponding to the output response for the rule. The summation of these terms is then divided by the summation of the multiplication of the degree of membership of each input. The final angle configuration can be obtained by adding $\Delta\theta_1$ and $\Delta\theta_2$ respectively to the original angle configuration. Once the final angle configuration is obtained, the final coordinates in world space are found.

x final_fuzzy = 0.431244
y final_fuzzy = 0.715706

which are to be compared to:

x_o final = 0.430647
y_o final = 0.738916

It can be observed that the fuzzy controller gets very close to the desired position.

The remaining 999 data sets were tested in similar fashion. The total rms error for this case study was found to be 0.010486. Mathematically, this is:

# of data sets tested:	1000
Total error summation:	0.219907

$$rms_{error} = \sqrt{\frac{\sum_{i=1}^{1000}(real - fuzzy)^2}{(\#outputs)\times(\#datasets)}} = \sqrt{\frac{0.219907}{(2)\times(1000)}} = 0.010486$$

As it can be observed, the rms error obtained for this set of data is very encouraging. This indicates that with only 10% of the original number of rules generated, the FLC is capable of good generalization when new data is tested. Table 3 illustrates the results obtained for the initial 30 data sets of the testing file.

In the initial trials of the above methodology different configurations were tried for the 2N+1 number of fuzzy regions for each input and output. It was observed that a minimum of 5 fuzzy regions is required to start getting values that are closer to the target outputs.

In this case study, N=2 for the inputs and N=3 for the outputs was proposed. Although the results obtained were close to those generated with the inverse kinematic equations of Chapter III, it must be noted that there was no specific criteria to choose the value for N. As it is reported in most of the available literature, this is in essence a trial and error method that can take a lot of time when the control problem under consideration is not well known. In this specific case, the constraints assumed for the robot configuration allowed for a relatively easy deduction of N.

Furthermore, with the help of the *difference_x* and *difference_y* columns in Table 3, it can be observed that the differences in the values for the real and fuzzy coordinates, are a good tool to understand in which direction N should vary. For values of N=2 it was observed that there is a considerable difference between the fuzzy values and those from the testing set. As the granularity of the system is increased, each data input into the FLC will fire more rules in a parallel fashion. As a result, better centroid values will be obtained for the outputs and this in turn will approximate the fuzzy coordinate to the original coordinate.

Finally, the *difference_x* and *difference_y* columns can help define the range of the fuzzy interval values for each of the input and output variables. As mentioned before, it is possible to have values outside the fuzzy range. Although this was not considered in this example, the obtained differences indicate that this could indeed be possible.

Table 3.

The initial 30 sets of the testing file. Notice that the fuzzy results are very close to those obtained with the inverse kinematic equations of chapter III. The last two columns may give an understanding of how the fuzzy memberships can be modified.

x_final	y_final	x_final_fuzzy	y_final_fuzzy	difference_ x	difference_ y	rms error
0.430645	0.738917	0.431244	0.715706	-0.000599	0.023211	0.000539
0.370284	1.784337	0.378292	1.777003	-0.008008	0.007334	0.000118
1.859882	0.249997	1.857486	0.242693	0.002396	0.007304	0.000059
0.581046	0.730074	0.583589	0.719319	-0.002543	0.010755	0.000122
0.839633	0.574833	0.841077	0.563014	-0.001444	0.011819	0.000142
0.200833	0.718458	0.205668	0.712074	-0.004835	0.006384	0.000064
0.492657	1.801555	0.503171	1.794663	-0.010514	0.006892	0.000158
0.200588	1.354861	0.207147	1.346998	-0.006559	0.007863	0.000105
0.832408	1.305466	0.836422	1.291133	-0.004014	0.014333	0.000222
0.200893	1.775282	0.210958	1.768155	-0.010065	0.007127	0.000152
0.616599	1.27914	0.620089	1.264219	-0.00349	0.014921	0.000235
0.978791	1.194971	0.979657	1.176995	-0.000866	0.017976	0.000324
1.809014	0.249982	1.803	0.235955	0.006014	0.014027	0.000233
0.714401	1.478355	0.719068	1.461792	-0.004667	0.016563	0.000296
0.792871	0.72892	0.78814	0.707449	0.004731	0.021471	0.000483
0.400057	0.711288	0.398874	0.689981	0.001183	0.021307	0.000455
1.044901	0.248507	1.030895	0.232663	0.014006	0.015844	0.000447
0.200538	1.848126	0.21949	1.838857	-0.018952	0.009269	0.000445
0.891823	0.709971	0.887481	0.692391	0.004342	0.01758	0.000328
1.14345	0.801888	1.141481	0.783637	0.001969	0.018251	0.000337
0.201219	0.249985	0.193683	0.236042	0.007537	0.013943	0.000251
1.292813	1.2424	1.299004	1.226196	-0.006191	0.016204	0.000301
1.485864	0.507749	1.481573	0.491805	0.004291	0.015944	0.000273
1.20837	0.835217	1.208539	0.817084	-0.000169	0.018133	0.000329
0.200568	0.249997	0.194192	0.237255	0.006376	0.012742	0.000203
0.57688	1.56337	0.588709	1.550431	-0.011829	0.012939	0.000307
0.64804	0.249497	0.638024	0.238969	0.010016	0.010528	0.000211
0.39564	1.479369	0.406701	1.467046	-0.011061	0.012323	0.000274
0.807731	1.583846	0.818774	1.570982	-0.011043	0.012864	0.000287

Chapter V

Application of the Wang-Mendel Methodology in Job-Shop Scheduling

He that will not apply new
remedies must expect new evils;
for time is the greatest innovator
- Francis Bacon

5.1 Job-Shop Scheduling: Introduction

The most important characteristic of the job-shop scheduling problem is that the flow of work is not unidirectional. The elements of a problem are a set of machines and a collection of jobs to be scheduled (Baker 1975). Each job consists of several operations with the same linear precedence structure. Although it is possible to allow any number of operations in a given job, the most common formulation of the job-shop problem specifies that each job has exactly m operations, one on each machine. Because the work flow in a job shop is not unidirectional, each machine in the shop can be characterized by the input and output flows of work shown in Figure 5.1. In the job-shop problem, an operation is described with a triplet (i, j, k) in order to denote the fact that operation j of job i requires machine k.

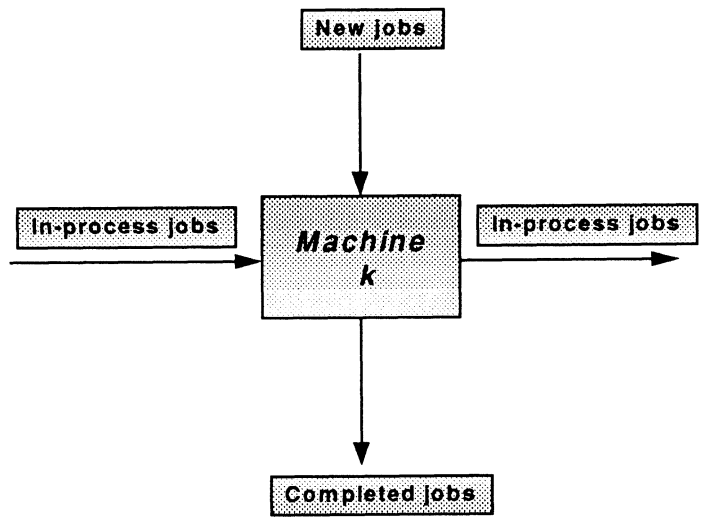

Figure 5.1. In the job-shop problem, an operation is described with a triplet (i, j, k) in order to denote the fact that operation j of job i requires machine k.

A graphical description of the job shop problem includes the jobs to be scheduled and a Gantt chart in which to fill those jobs. The illustration in Figure 5.2 consists of a collection of blocks, each of which is identified by a job-operation triplet. Using the scale of the Gantt chart, the length of the block is equal to the processing time of the associated operation. The sequential numbering of operations for a given job is a means of indicating the linear operation sequence. A schedule is a feasible resolution of the resource constraints when no two operations ever occupy the same machine simultaneously. A schedule is also a feasible resolution of the logical constraints when all operations of each given job can be placed on one time axis in precedence order and without overlap.

In principle, there are infinite number of feasible schedules for any job-shop problem because an arbitrary amount of idle time can be inserted at any machine between adjacent pairs of operations (French 1983). It should be clear, however, that once the operation sequence is specified for each machine, this kind of idle time cannot be helpful for any regular measure of performance. Rather, it is desirable that the operations be processed as compactly as possible.

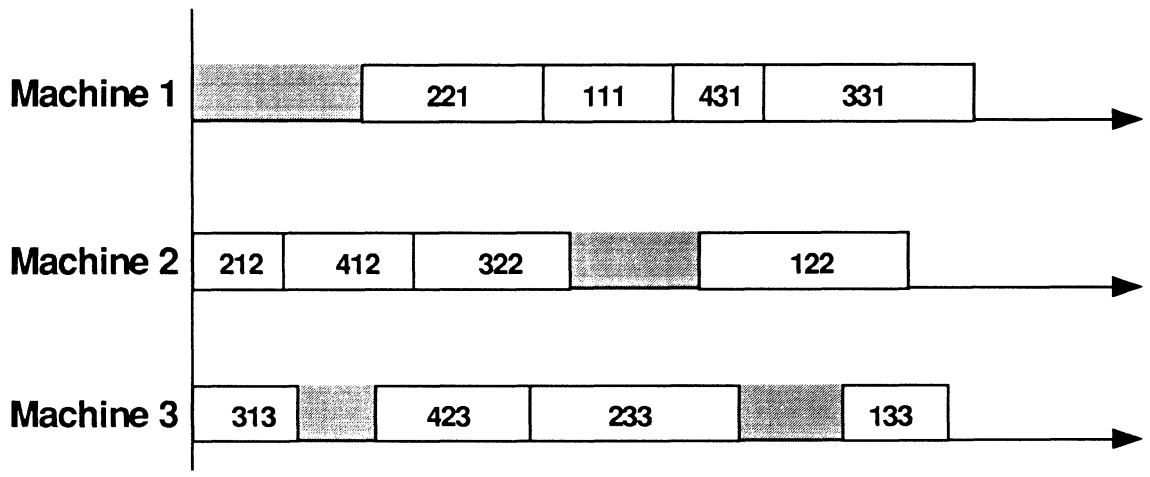

Figure 5.2. The sequential numbering of operations for a given job is a means of indicating the linear operation sequence. A schedule is a feasible resolution of the resource constraints when no two operations ever occupy the same machine simultaneously

Although the number of active schedules tends to be large, it is sometimes convenient to focus on a smaller subset called *nondelay* schedules. In a nondelay schedule no machine is kept idle at a time when it could begin processing some operation, and each operation is processed as soon as the machine is available (Baker 1974). As a result, a nondelay schedule can usually be expected to provide a very good solution, if not an optimum.

5.2 Case Study

Consider a simple manufacturing system production floor in which a number of products or units must be produced in response to customer orders in terms of process planning requirements. Jobs arrive over time at either specified or random arrival dates. Most jobs have similar priorities at the beginning, but the arrival of rush jobs that must be moved quickly through the line produces changes in the initial planning.

This simple manufacturing floor consists of one machine at which the required processing will be performed. The machine is capable of performing a set of different operations. Due to different product configurations, the machine will be assumed to require a different time interval to perform each operation.

The primary objective of this manufacturing schedule is to satisfy customer demands in terms of meeting due dates for each one of the jobs. Excessive earliness and lateness are equally poor practices. As a result, the choice of which job to process next is important because this will affect the overall processing or flow time of the system. The job must be allocated in the most appropriate order, chosen from alternate jobs available in the queue of operations. The job assignment must anticipate the resulting reaction of the machine, specially the expected completion time for the job. After this evaluation, the job

is sequenced as high priority or low priority. This characteristic allows the machine supervisor to process a job first even if its arrival is the last one in the system.

The sequencing priority is defined to be either FIFO (First In First Out) or LIFO (Last In First Out). Since both situations cannot occur simultaneously in one machine, it follows that one output should handle all possible outcomes. Nevertheless, in this job-shop there will be "two" outputs. The first one corresponds to the LIFO situation and a 1 means that the priority is LIFO, 0 means that it is FIFO. The second output corresponds to the FIFO situation and a 1 represents a FIFO priority while a 0 represents a LIFO priority. From this it is inferred that both outputs are opposite in value, that is:

Input	Input 1	Input 2	
Output	0	1	means FIFO priority
	1	0	means LIFO priority

In this case study, a singular machine is considered to receive n jobs. The jobs are processed one at a time; the remaining jobs form a processing queue for the machine. This queue could change when the dynamic rates of the environment are considered. The processing time for all jobs create a discrete mean-processing time distribution with the mean flow time as the performance criteria. The mean-processing time for each job is then taken into consideration to define the equation for the linear regression $a+b(t)$. Figure 5.3 illustrates the linear regression. The coefficients of this linear regression, a, b, become the inputs for the fuzzy controller. The outputs of the controller are two possible processing conditions: FIFO or LIFO. Depending on what type of decision is undertaken to process the job, the overall mean flow time performance of the queue will be affected. Note that this is a very simplified case of a job-shop scheduling problem. A more realistic formulation should consider factors such as the arrival time, the due date, the set-up time, bottlenecks, etc., each with its own particular regression (that is, different a, b coefficients) and a more complex performance criteria (Rabelo 1993). Also, as the

number of machines is increased in the system, other "scheduling" features, such as loading factors, should be accounted for (Rabelo 1993, Rabelo 1990).

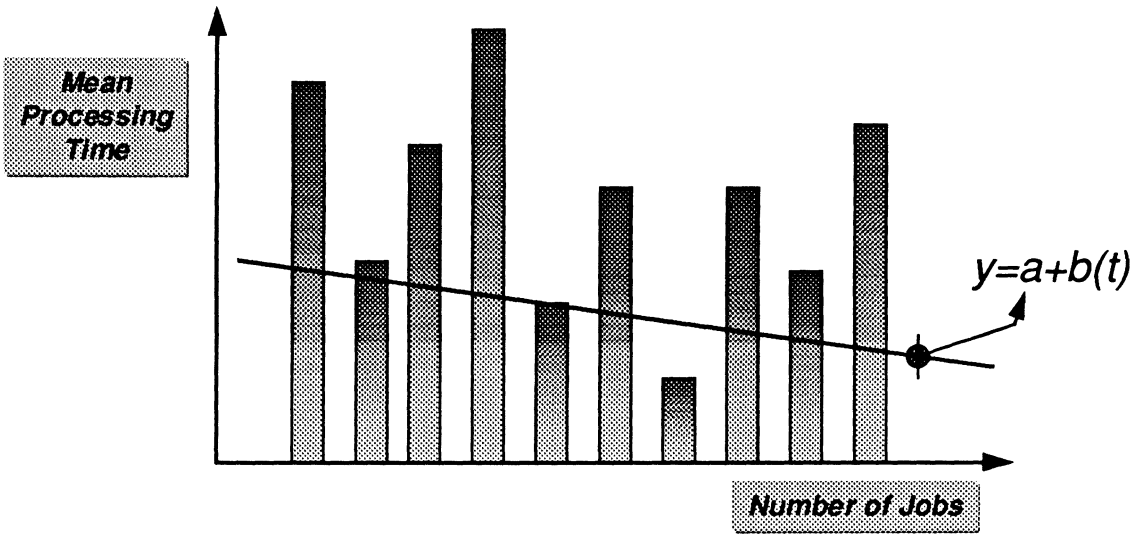

Figure 5.3 The coefficients of the linear regression $a+b(t)$ represent the mean-processing time distribution.

Table 4 indicates the vector value N for the problem, which is {2,2,1,1}, given the final 5 fuzzy regions for both inputs, and the 3 fuzzy regions for the outputs. Figure 5.3 shows the degree of membership functions for the problem. The domain interval corresponding to both inputs is divided into 5 regions (N=2, min=0.0, max=1.0), and the domain interval for both outputs is divided into 3 regions (N=1, min=0.0, max=0.0).

Table 4.
Fuzzy Ranges for inputs and outputs of the job-shop problem

	Lower Limit	Uper Limit	Fuzzy Regions
Input 1	0.0	1.0	5
Input 2	0.0	1.0	5
LIFO	0.0	1.0	3
FIFO	0.0	1.0	3

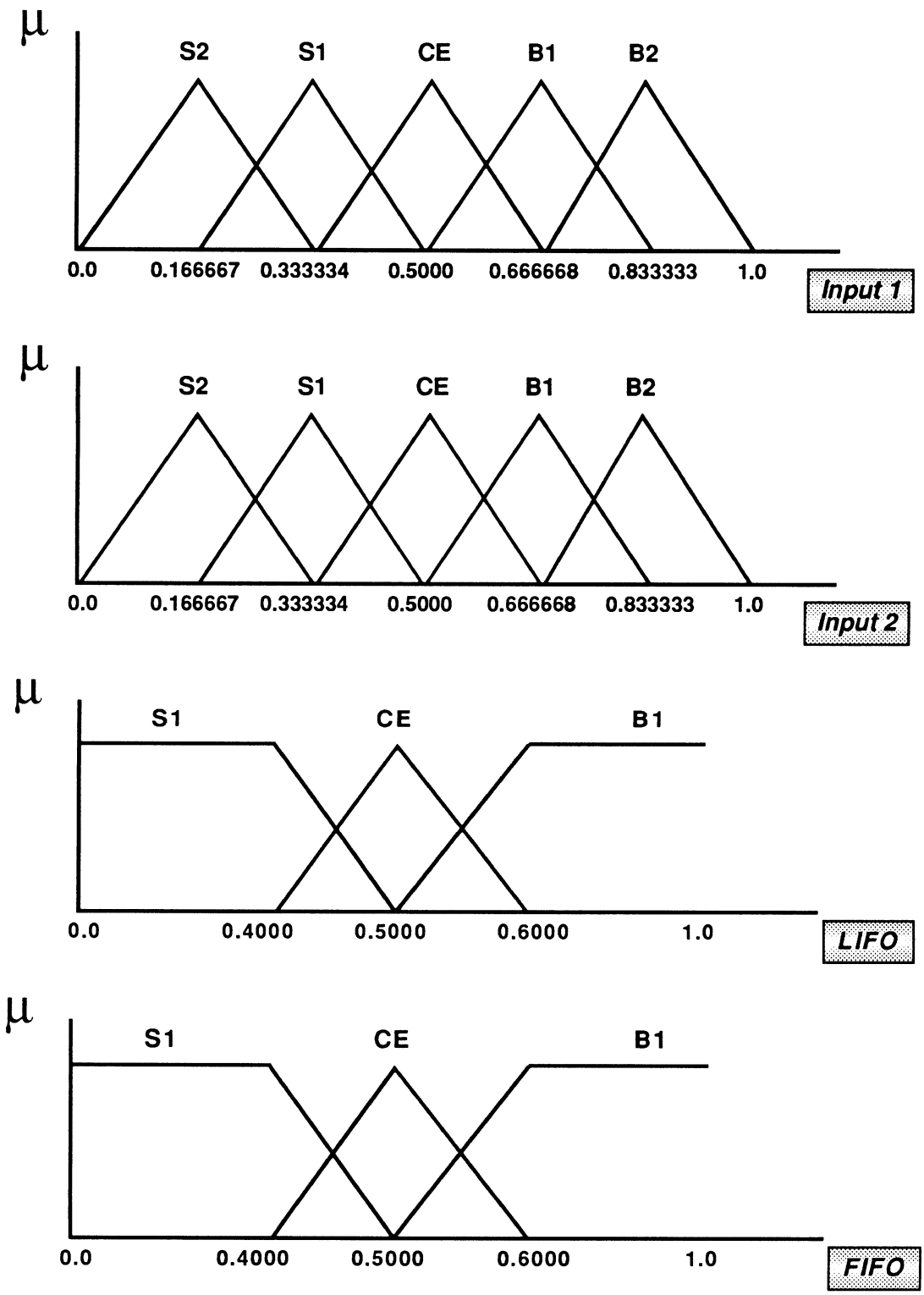

Figure 5.4. The degree of membership functions for the two inputs and the two outputs of the job-shop problem. These functions transform crisp values into degrees of membership.

The shape of each membership function for both outputs is both triangular and trapezoidal. For all practical purposes, the domain interval for the outputs could have been defined as a singleton, i.e. a membership allowing only two values, 0 or 1.

So as to be consistent with the Wang-Mendel methodology, the 2N+1 rule is maintained with the trapezoidal shapes having the whole weight in the final outcome, that is, the centers of gravity of these two regions are considered to be 0 and 1.

Once the degree of membership functions have been defined for all inputs and outputs, each fuzzy set is denoted by SN (Small N), ..., CE (Center), ..., BN (Big N). The degree of membership functions for all variables will transfer the measured crisp values into degrees of membership in fuzzy sets. This procedure is explained in the next section.

5.2.1 Training the Fuzzy Logic Controller

For each fuzzy set, a degree of membership is defined. Referring to Figure 5.4, and the input variable *Input 1*, the linguistic variable CE is defined by two straight lines crossing at (0.500,1.000), that is, the center value of the region denoted CE.

Therefore, given that two points of each line are known, the equation of each line is easily computed. Figure 5.5 illustrates this process. Mathematically the following values are of interest.

Point a = (0.33334,0.0000)
Point b = (0.50000,1.0000)

$$slope = m_1 = \frac{1.000}{0.1667} = 5.9999$$

$\therefore line_1 = y = 5.9999x - 2.0$, which corresponds to the first degree of membership function.

Similarly

Point b = (0.50000,1.0000)
Point c = (0.66668,0.0000)

$$slope = m_2 = \frac{-1.000}{0.1667} = -5.9999$$

$\therefore line_2 = y = -5.9999x + 4.0$, which corresponds to the second degree of membership function.

The above procedure is then applied to rest of the linguistic variables for all inputs and outputs. After assigning each region a fuzzy membership function, the variables of the job-shop problem look like those of *Input 1* in table 5.

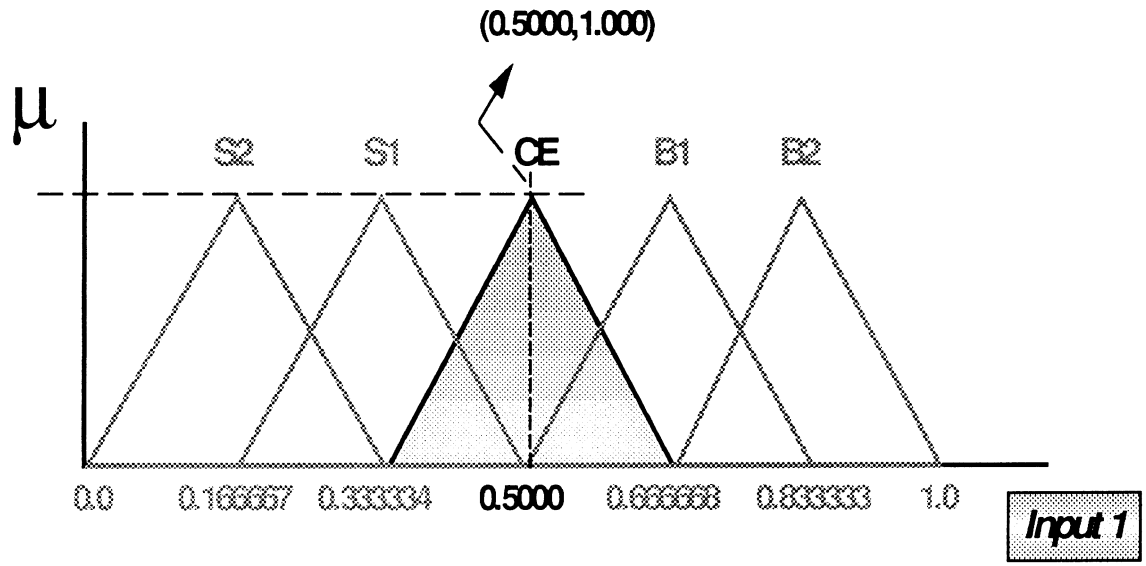

Figure 5.5. The fuzzy set corresponding to the linguistic variable CE is represented by the equations of two lines: $\therefore line_1 = y = 5.9999x - 2.0$ and $\therefore line_2 = y = -5.9999x + 4.0$.

Once the degree of membership functions have been declared, the fuzzification procedure will transform the crisp data into fuzzy values. Fuzzification proceeds as explained in section 4.4. Each member of the data set is assigned a degree of membership on its specific fuzzy region. Figure 5.6 is a graphical display of the fuzzification procedure. This fuzzification procedure will generate a rule for each data set belonging to

the training data. For this job-shop problem a training file of 358 data pairs was generated.

Table 5

Each input is divided into 2N+1 fuzzy regions. The degree of membership function is found for each region, and a fuzzy value is assigned

Interval	Membership Function	Fuzzy Value
$0.0000 \le x < 0.1667$	$y = 5.9999x$	S2
$0.1667 \le x < 0.3334$	$y = -5.9999x + 2.0$	S2
$0.1667 \le x < 0.3334$	$y = 5.9999x - 1.0$	S1
$0.3334 \le x < 0.5000$	$y = -5.9999x + 3.0$	S1
$0.3334 \le x < 0.5000$	$y = 5.9999x - 2.0$	CE
$0.5000 \le x < 0.6668$	$y = -5.9999x + 4.0$	CE
$0.5000 \le x < 0.6668$	$y = 5.9999x - 3.0$	B1
$0.6668 \le x < 0.8333$	$y = -5.9999x + 5.0$	B1
$0.6668 \le x < 0.8333$	$y = 5.9999x - 4.0$	B2
$0.8333 \le x < 1.0000$	$y = -5.9999x + 6.0$	B2

Referring always to section 4.4, the next step eliminates conflicting rules by assigning a degree or weight to each rule. Conflicting rules are then compared and that with the highest weight is conserved while the others are discarded. Appendix G presents a listing of the final 15 rules corresponding to this case study.

After the final set of rules has been defined, the next step in the FLC training calls for the formulation of the Fuzzy Associative Memory Bank (FAM). The FAM bank is a mapping of the inputs and outputs into a common framework and the one corresponding to this job-shop can be observed in figure 5.7. Formulation of the FAM bank concludes the training of the FLC. Section 5.2.2 will test the FLC.

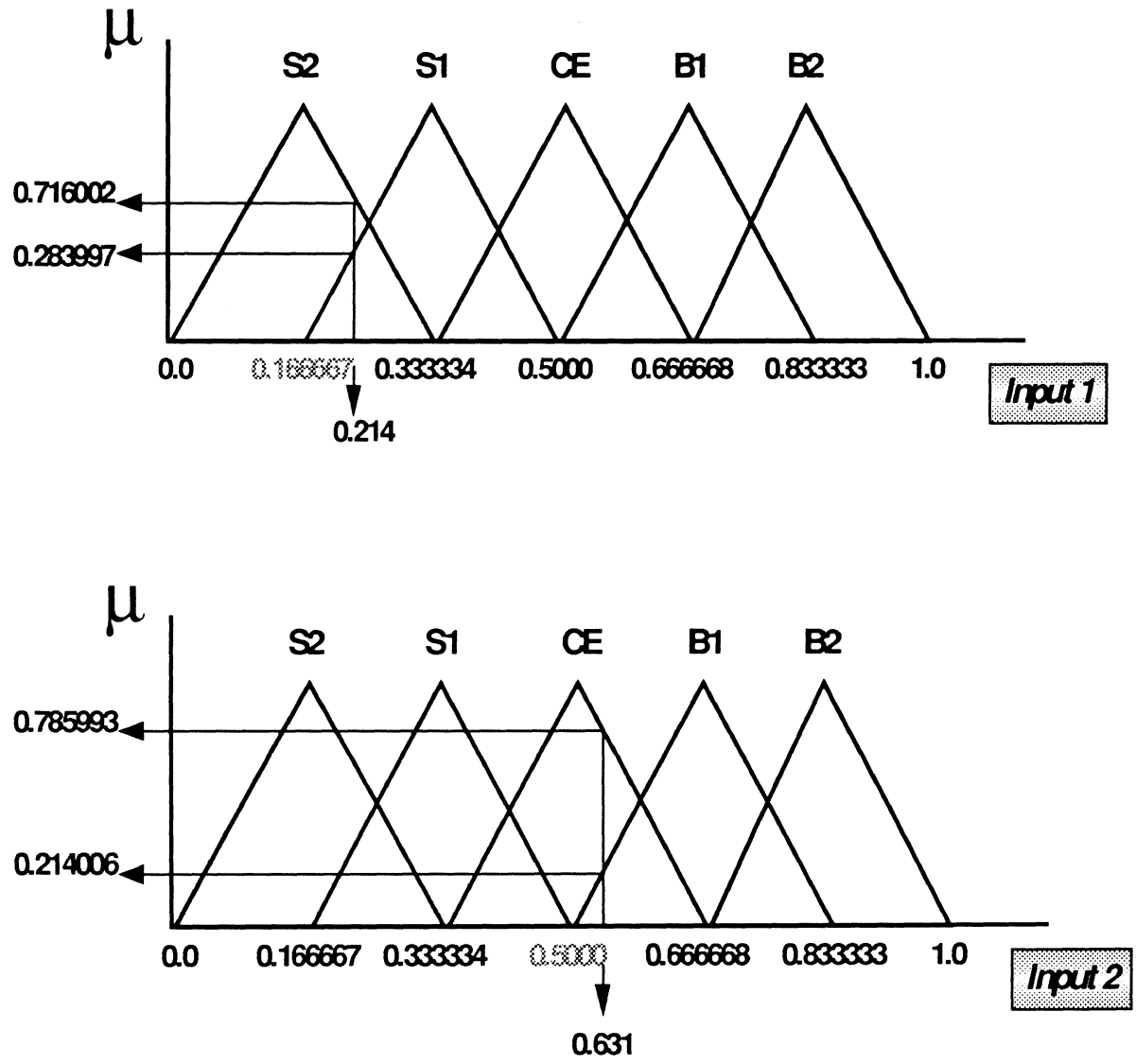

Figure 5.6. The degree of membership values are obtained by mapping the crisp input values with the corresponding degree of membership functions. The value that prevails is that corresponding to the highest degree of membership.

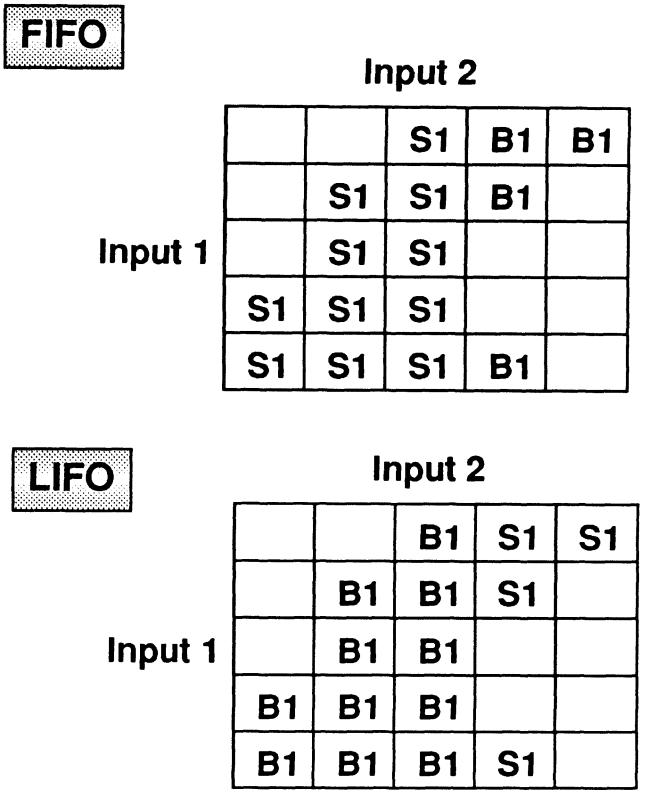

Figure 5.7. Generated FAM bank for the problem. From the original 358 rules, only 15 are considered for the bank. Given that only FIFO or LIFO can occur at one given time, note how the control outputs are opposite.

5.2.2 Job-Shop Scheduling: Fuzzy

In this section, the FLC's performance is demonstrated with a "walk-through" data pair from the testing file. The testing file has the same characteristics of the training file and it consists of 198 data sets. Considering the following data set, the testing procedure continues below.

	Input 1	Input 2
	0.214	0.631
Outputs	0.0	1.0

From the above data pair, the values corresponding to the inputs are fuzzified. Figure 5.7 depicts the procedure. Mathematically, this is:

Input 1 = 0.214

$$0.1667 \le x < 0.3334 \qquad \therefore y = -5.9999x + 2.0 = 0.716002 \quad \text{S2}$$
$$y = 5.9999x - 1.0 = 0.283997 \quad \text{S1}$$

Input 2 = 0.631

$$0.5000 \le x < 0.6668 \qquad \therefore y = -5.9999x + 4.0 = 0.214006 \quad \text{CE}$$
$$y = 5.9999x - 3.0 = 0.785993 \quad \text{B1}$$

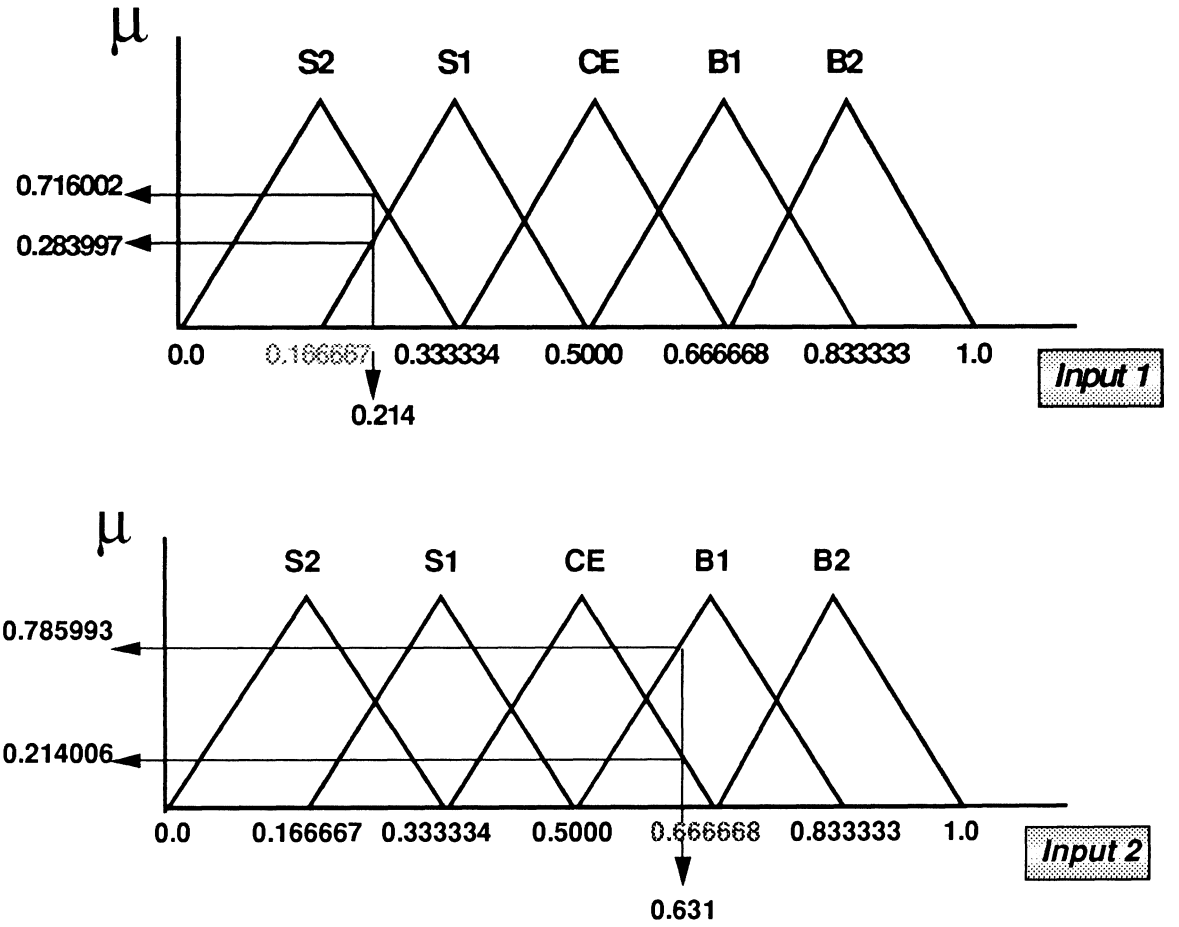

Figure 5.8. When the FLC receives new information, the inputs are fuzzified according to the degree of membership functions defined in 5.2.1. Given that each input "hits" two membership functions, the total number of rules fired simultaneously is 4.

Given that the FAM gets activated in parallel, the total possible number of rules that can fire simultaneously is 4, or 2^2, corresponding to the two inputs and the two fuzzy values activated at the same time. The antecedents of the 4 rules are:

Input 1	Input 2
S2	CE
S2	B1
S1	CE
S1	B1

These antecedents will be then mapped to their correspondent control actions in the FAM bank. The antecedents listed above correspond to the following FAM entries:

S1 B1
B1 S1
0.698000 0.677999 1.000000 0.000000 3

S1 CE
S1 B1
0.984000 0.922000 0.000000 1.000000 4

S2 B1
B1 S1
0.560000 0.557999 1.000000 0.000000 5

S2 CE
S1 B1
0.560000 0.658001 0.000000 1.000000 6

When all numerical values are found, the defuzzification strategy to obtain the output control FIFO or LIFO is calculated using equations 3.6 and 3.7. The procedure is also shown in section 4.4.

For this job-shop problem, the output is essentially binary since the occurrence of a FIFO situation implies the not occurrence of a LIFO situation (the reverse is also true). For this reason, the center of the fuzzy regions, which is the point that has the smallest absolute value among all the points at which the membership function for the specific region has membership value equal to one, takes only two possible values, 1 and 0. Having defined this, and in the same fashion as in the robot case study, the centroids are found to be:

Centroid 1 = 0.076752

Centroid 2 = 0.922034

which is Centroid 1 = 0.0

Centroid 2 = 1.0

giving a perfect match with the original testing set. Table 6 displays results for some other testing samples.

Table 6

Values obtained for the job-shop problem with and FLC of 15 rules. Note that the highest value corresponds to a 1.0 value while the lowest value is assigned a 0.0. The FAM satisfies most cases except for those listed in table 7

			FIFO	LIFO
Inputs	0.336	0.513	0	1
Centroids	0.076752	0.922		
			FIFO	LIFO
Inputs	0.214	0.631	1	0
Centroids	0.393	0.107		
			FIFO	LIFO
Inputs	0.257	0.519	0	1
Centroids	0.038	0.295334		
			FIFO	LIFO
Inputs	0.188	0.645	1	0
Centroids	0.2175	0.0325		
			FIFO	LIFO
Inputs	0.214	0.559	0	1
Centroids	0.0708	0.1292		
			FIFO	LIFO
Inputs	0.174	0.679	1	0
Centroids	0.166124	0		
			FIFO	LIFO
Inputs	0.076	0.818	1	0
Centroids	0.131944	0		
			FIFO	LIFO
Inputs	0.115	0.727	1	0
Centroids	0.166425	0		
			FIFO	LIFO
Inputs	0.184	0.589	1	0
Centroids	0.057469	0.050151		
			FIFO	LIFO
Inputs	0.352	0.583	1	0
Centroids	0.042968	0.048776		
			FIFO	LIFO
Inputs	0.428	0.371	0	1
Centroids	0	0.088558		
			FIFO	LIFO
Inputs	0.655	0.299	0	1
Centroids	0	0.080181		
			FIFO	LIFO
Inputs	0.243	0.848	1	0
Centroids	0.070035	0		

Examination of the final results indicate that the FLC was unable to map 5 data sets. The data sets are listed in Table 7. Considering that the total number of data sets tested was 198, the FLC has a 97% accurate performance.

Table 7

Mismatched data sets. The FLC failed to recognize 5 patterns observing a final controller performance of 97%.

			FIFO	LIFO
Inputs	0.352	0.583	1	0
Centroids	0.042968	0.048776		
			FIFO	LIFO
Inputs	0.352	0.517	0	1
Centroids	0.001327	0.013158		
			FIFO	LIFO
Inputs	0.234	0.583	1	0
Centroids	0.004963	0.005003		
			FIFO	LIFO
Inputs	0.224	0.571	1	0
Centroids	0.003271	0.004407		
			FIFO	LIFO
Inputs	0.211	0.575	1	0
Centroids	0.002239	0.002737		

5.3 Neural Networks approach: a comparison

A typical backpropagation neural network is applied to solve the same job-shop problem. Backpropagation is based on a simple concept: if the network gives the wrong answer, the weights are corrected so that the error is lessened and then future responses of the network are more likely to be correct.

Typically, backpropagation employs three or more layers of processing units. The initial layer is referred as the input layer the only units in the network that receive external input. The last layer is the output layer. Any layer in between the input and output layers

is referred to as a hidden layer in which the processing units are interconnected to the previous and later layers.

For the job-shop problem, the neural network is trained by supervised learning. The network was presented with pairs of patterns, an input pair corresponding to Input 1 and Input 2, paired with the LIFO/FIFO target output. As described in the previous section, each pattern is a vector of real numbers ranging from 0.0 to 1.0. The target output LIFO/FIFO pattern is defined as the desired response to the input pattern and it is used to determine the error values in the network when the weighs are adjusted.

An important concept of backpropagation refers to convergence. Convergence is a measure of learning expressed quantitatively through the root mean-squared (RMS) error. This measure reflects how close the network is to getting correct answers. As the network learns, its RMS error decreases.

The training data presented to the neural network had the same format as the one presented to the FLC, that is:

Input 1	Input 2
LIFO	FIFO

Just like in the FLC case, the output target can only be LIFO or FIFO, i.e. the output can only have value combinations of 1,0 or 0,1. Special cases lake 1,1, or 0,0 are not considered in this work.

Table 8 displays the final entries recorded on the history file corresponding to the learning rate of the neural network. For a total of 8600 iterations, the network reaches a rms error of 0.116.

Table 8.

History file corresponding to the job-shop problem. Notice that the network progressively learned from an initial rms error of 0.65 to a final rms error of 0.11604. Once this rms error is reached, the network is tested for generalization of new data

rms	max rms	epochs	LR 1	LR 2	Mom	H	DP	Generalization
0.112329	0.992438	7900	0.025000	0.025000	0.600000	4	360	0.168315
0.112329	0.992438	7900	0.025000	0.025000	0.600000	4	360	
0.112263	0.992724	8000	0.025000	0.025000	0.600000	4	360	0.168422
0.112263	0.992724	8000	0.025000	0.025000	0.600000	4	360	
0.112199	0.992996	8100	0.025000	0.025000	0.600000	4	360	0.168527
0.112199	0.992996	8100	0.025000	0.025000	0.600000	4	360	
0.112138	0.993254	8200	0.025000	0.025000	0.600000	4	360	0.168630
0.112138	0.993254	8200	0.025000	0.025000	0.600000	4	360	
0.112078	0.993501	8300	0.025000	0.025000	0.600000	4	360	0.168733
0.112078	0.993501	8300	0.025000	0.025000	0.600000	4	360	
0.112059	0.993578	8332	0.025000	0.025000	0.600000	4	360	
0.117849	0.986513	8400	0.030000	0.030000	0.700000	4	360	0.146131
0.117849	0.986513	8400	0.030000	0.030000	0.700000	4	360	
0.117302	0.996759	8453	0.030000	0.030000	0.700000	4	360	
0.116771	0.987668	8500	0.030000	0.030000	0.700000	4	360	0.148364
0.116771	0.987668	8500	0.030000	0.030000	0.700000	4	360	
0.116039	0.988824	8600	0.030000	0.030000	0.700000	4	360	0.150053
0.116039	0.988824	8600	0.030000	0.030000	0.700000	4	360	

**LR 1: learning rate 1*
**LR 2: learning rate 2*
***Mom: momentum*

For the rms error reached, generalization was performed to observe the level of learning. Out of 198 testing data pairs, 4 mismatches were encountered. Figure 5.9 displays the final configuration of the backpropagation neural network. The final weights of each link are also reported. Appendix G presents samples of the testing and training data.

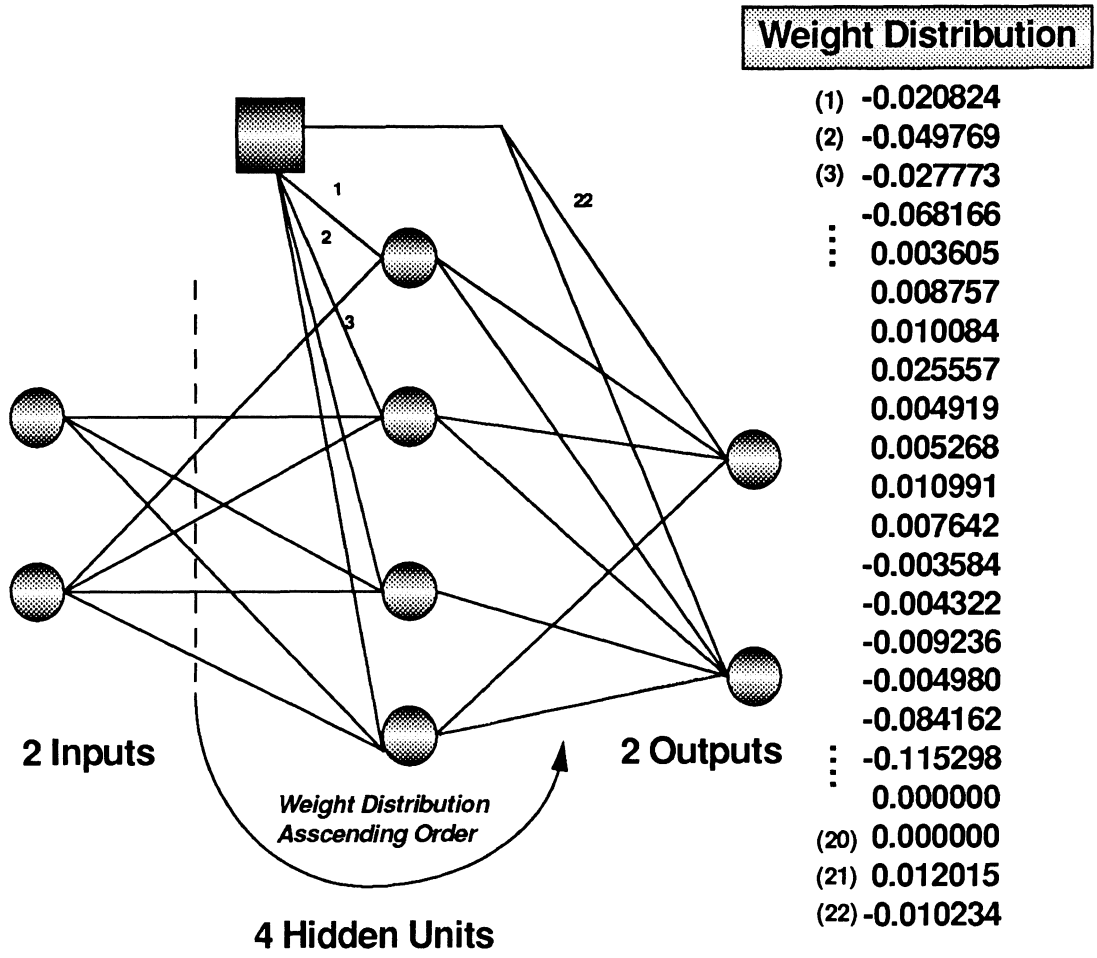

Figure 5.9. Final configuration for the back propagation neural network. The network has two inputs, two outputs, and 4 hidden units. Also, notice that the weights are distributed counterclockwise starting at the (1) link and finishing at number (22)

This neural network displayed a fairly large capability for general pattern mapping, since it only needed examples of the mapping to be learned. This flexibility is further enhanced by the large number of design choices available: choices for the number of layers, interconnections, processing units, the learning constants, and the data representations.

It was also concluded that the major drawback of this method is the amount of computational time required to reach the specific convergence level. This job-shop problem presented only two inputs and two outputs that oversimplify any real application. Real applications probably encompass thousands of examples for training sets and the constraints placed upon the network are much more demanding. This suggests a point at which the neural network approach starts decreasing in power. And this is where fuzzy logic becomes discernible to apply.

Training the Network with only Fuzzy Rules

The same network was then presented with only the fifteen rules obtained with the fuzzy machine. The network was able to reach an rms error of 0.1206 in a relatively short time, but generalization was not as good as desired. It was observed that the network was consistently able to perform satisfactorily 66% of the times (132 testing data were correctly recognized from a total possible of 198). The final configuration of the ANN architecture is presented in Figure 5.10.

5.4 Quinlan's ID3 Machine Learning : a comparison

The study and modeling of inductive learning is one of the central topics of machine learning. Inductive learning algorithms could provide both an improvement of the current techniques and a basis for developing alternative knowledge acquisition methods. These algorithms are used to detect and rectify inconsistencies, to remove redundancies, to cover gaps, and to simplify expert-derived decision rules. The rules could be incrementally improved with little or no human assistance by applying an inductive inference program to the data (Kinaci 1993).

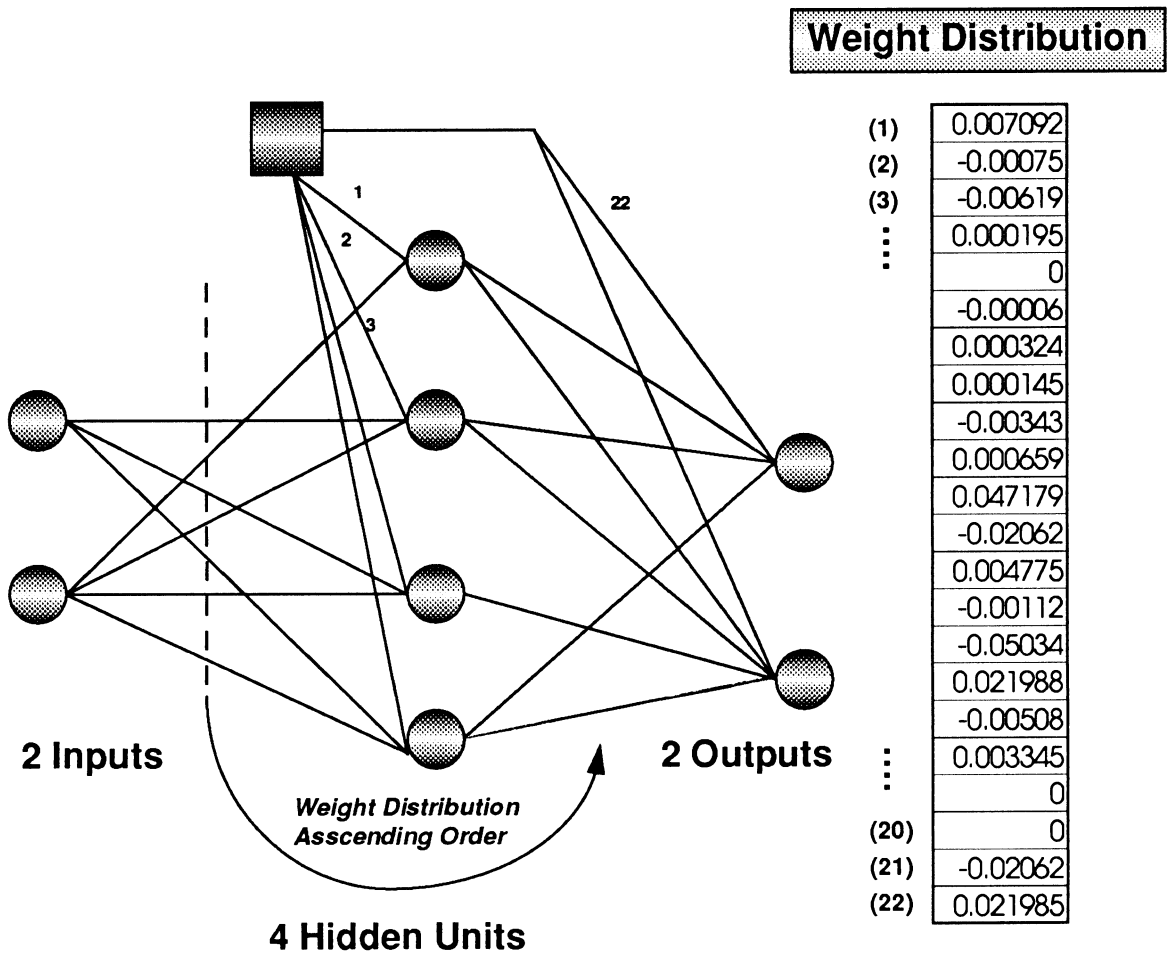

Figure 5.10 ANN architecture trained with only the 15 rules obtained with the fuzzy system. Note that the low weights values indicate overtraining.

Many inductive knowledge acquisition algorithms generate classifiers in the form of decision trees. A decision tree is a simple recursive structure for expressing a sequential classification process in which a *case*, described by a set of attributes, is assigned to one of a disjoint set of classes. Each leaf of the tree denotes a class. An interior node denotes a test on one or more of the attributes with a subsidiary decision tree for each possible outcome of the test. This is illustrated in Figure 5.11.

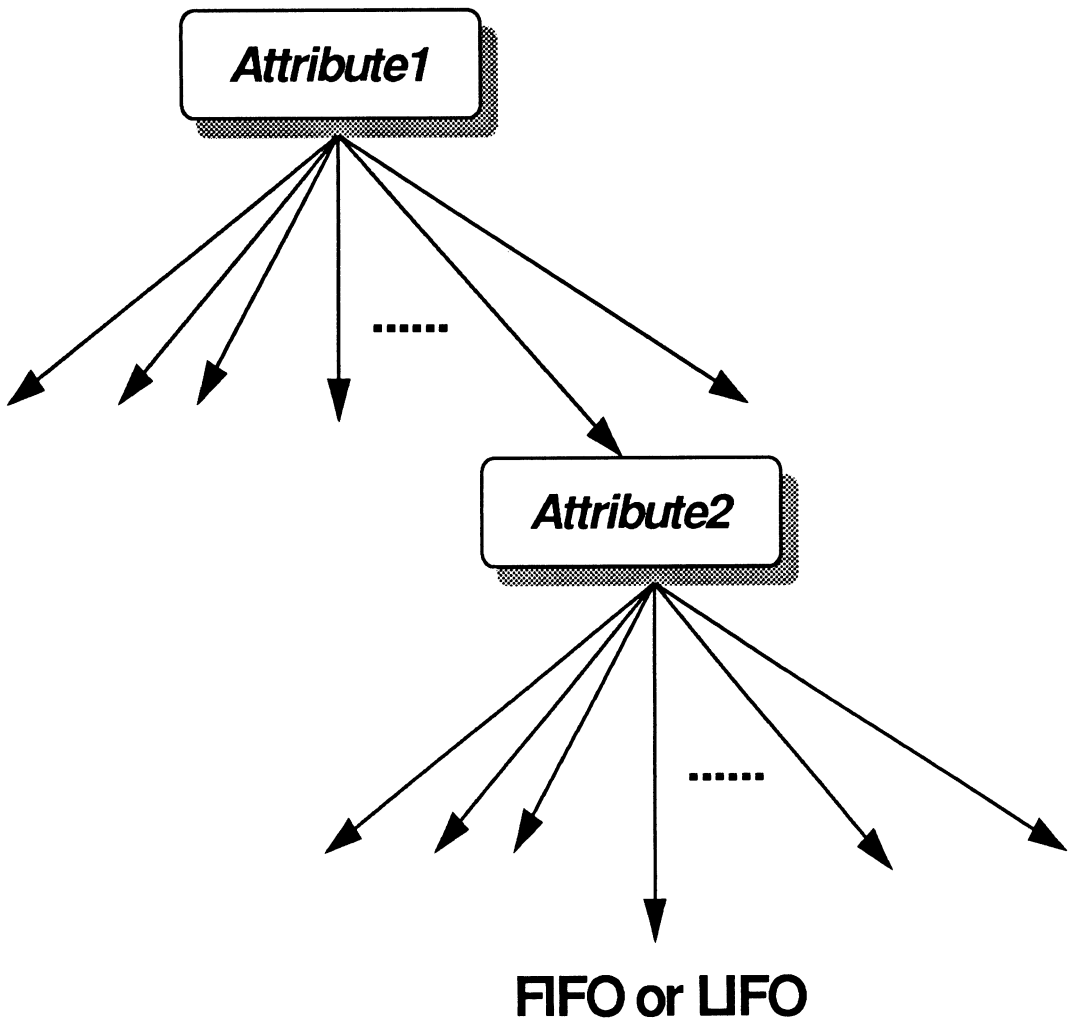

Figure 5.11 ID3 decision tree. Each leaf of the tree denotes a class. An interior node denotes a test on one or more of the attributes with a subsidiary decision tree for each possible outcome of the test.

Quinlan's ID3 takes objects of a known class described in terms of a fixed collection of attributes, and it builds a tree upon those attributes that correctly classify all the given objects. One of the characteristics which differentiates ID3 from other general purpose inference systems is that the effort required to accomplish an induction task grows with the number of objects and, in fact, its computation time increases only linearly as modeled by the product of:

1. The number of given exemplary objects,
2. The number of attributes used to describe objects,
3. The complexity of the concept to be developed as measured by the number of nodes in the decision tree.

ID3 is designed to solve the classification problems that contain many attributes and the training set that contains many objects. It produces a reasonably good decision tree without much computation. Although the ID3 method is used to generate simple decision trees, it does not guarantee that better decision trees have been evaluated.

In the job shop problem, the attributes for the problem are defined as *Input 1* and *Input 2*. In order to maintain the formulation similar to those developed for the FLC and the ANN machine, the value of the attributes is defined with numbers rather than linguistic descriptors. The possible values for both attributes is defined in the range [0.0,1.0]. Thus, both the training and testing data have the following format:

Input1 = Attribute1	*Input2* = Attribute 2
1.0 or 0.0 (LIFO)	0.0 or 1.0 (FIFO)

The outputs LIFO or FIFO define two possible classes for the system. In this manner, an object can be only LIFO or FIFO, which in real life would mean that if the class happens to be a 0.0 for the first attribute, then the task is given a FIFO priority. If the task happens to be 0.0 for the second attribute, then the task is assigned LIFO priority. Like in the previous two methods, objects with the same attributes can not belong to different class, that is, the class outcome can not be (1.0,1.0) or (0.0,0.0). The training file has 360 data pairs and the testing file has 198 data pairs as in Appendix G.

Since in this case study the attributes have continuous values both in the training and testing files, it is necessary to define ranges within the domain. This is done in order to account for different values that the attributes assume. For example, if the domain is

said to have 4 different values, the test will be performed based on the following distribution:

$$
\begin{aligned}
0.000 &\leq \textit{attribute} < 0.250 \\
0.250 &\leq \textit{attribute} < 0.500 \\
0.500 &\leq \textit{attribute} < 0.750 \\
0.750 &\leq \textit{attribute} < 1.000
\end{aligned}
$$

Once the values for the attributes are defined, the decision tree can be thought of an information source that produces a message which is the class of that object, in this case FIFO or LIFO. The selection of an attribute is defined on the assumption that the decision tree is associated to the amount of information passed on by this message.

For the job-shop problem, the near to optimal number of intervals was defined to be between 14 and 16. Although results can be improved by incrementing the total number of intervals in the domain, this does not necessarily mean that the new answer is the best. Table 9 shows the intervals corresponding to 14 leafs (the term "leaf" refers to each of the branches for an attribute in the decision tree). Table 10 displays the results obtained for the 14 leaf distribution. Notice that the number of successes is 334 in a total of 360 possible testing data sets.

Table 9

14 "leaf" intervals for the job shop problem.

$$0.000 \le input1 < 0.071$$
$$0.0714 \le input1 < 0.143$$
$$0.143 \le input1 < 0.214$$
$$0.214 \le input1 < 0.286$$
$$0.286 \le input1 < 0.357$$
$$0.357 \le input1 < 0.428$$
$$0.428 \le input1 < 0.500$$
$$0.500 \le input1 < 0.571$$
$$0.571 \le input1 < 0.643$$
$$0.643 \le input1 < 0.714$$
$$0.714 \le input1 < 0.786$$
$$0.786 \le input1 < 0.857$$
$$0.857 \le input1 < 0.928$$
$$0.928 \le input1 < 1.000$$

$$0.000 \le input2 < 0.071$$
$$0.0714 \le input2 < 0.143$$
$$0.143 \le input2 < 0.214$$
$$0.214 \le input2 < 0.286$$
$$0.286 \le input2 < 0.357$$
$$0.357 \le input2 < 0.428$$
$$0.428 \le input2 < 0.500$$
$$0.500 \le input2 < 0.571$$
$$0.571 \le input2 < 0.643$$
$$0.643 \le input2 < 0.714$$
$$0.714 \le input2 < 0.786$$
$$0.786 \le input2 < 0.857$$
$$0.857 \le input2 < 0.928$$
$$0.928 \le input2 < 1.000$$

Table 10

Output result for Quinlan's ID3 decision tree.
This output corresponds to 14 leafs on the tree.

NUMBER OF LEAF INTERVALS = 14
DATA PAIRS TO TEST = 360

RULES CREATED BY ID3

==========

IF 0.0000 < input1 <= 0.0714 **AND** 0.7857 < input2 <= 0.8571 **THEN** use FIFO.
IF 0.0000 < input1 <= 0.0714 **AND** 0.8571 < input2 <= 0.9286 **THEN** use FIFO.
IF 0.0714 < input1 <= 0.1429 **AND** 0.5714 < input2 <= 0.6429 **THEN** use FIFO.
IF 0.0714 < input1 <= 0.1429 **AND** 0.6429 < input2 <= 0.7143 **THEN** use FIFO.
IF 0.0714 < input1 <= 0.1429 **AND** 0.7143 < input2 <= 0.7857 **THEN** use FIFO.
IF 0.0714 < input1 <= 0.1429 **AND** 0.7857 < input2 <= 0.8571 **THEN** use FIFO.
IF 0.1429 < input1 <= 0.2143 **AND** 0.5000 < input2 <= 0.5714 **THEN** use LIFO.
IF 0.1429 < input1 <= 0.2143 **AND** 0.5714 < input2 <= 0.6429 **THEN** use FIFO.
IF 0.1429 < input1 <= 0.2143 **AND** 0.6429 < input2 <= 0.7143 **THEN** use FIFO.
IF 0.1429 < input1 <= 0.2143 **AND** 0.7143 < input2 <= 0.7857 **THEN** use FIFO.
IF 0.1429 < input1 <= 0.2143 **AND** 0.7857 < input2 <= 0.8571 **THEN** use FIFO.
IF 0.2143 < input1 <= 0.2857 **AND** 0.4286 < input2 <= 0.5000 **THEN** use LIFO.
IF 0.2143 < input1 <= 0.2857 **AND** 0.5714 < input2 <= 0.6429 **THEN** use FIFO.
IF 0.2143 < input1 <= 0.2857 **AND** 0.6429 < input2 <= 0.7143 **THEN** use FIFO.
IF 0.2143 < input1 <= 0.2857 **AND** 0.7857 < input2 <= 0.8571 **THEN** use FIFO.
IF 0.2857 < input1 <= 0.3571 **AND** 0.4286 < input2 <= 0.5000 **THEN** use LIFO.
IF 0.2857 < input1 <= 0.3571 **AND** 0.5714 < input2 <= 0.6429 **THEN** use FIFO.
IF 0.3571 < input1 <= 0.4286 **AND** 0.3571 < input2 <= 0.4286 **THEN** use LIFO.
IF 0.3571 < input1 <= 0.4286 **AND** 0.4286 < input2 <= 0.5000 **THEN** use LIFO.
IF 0.3571 < input1 <= 0.4286 **AND** 0.5000 < input2 <= 0.5714 **THEN** use LIFO.
IF 0.4286 < input1 <= 0.5000 **AND** 0.3571 < input2 <= 0.4286 **THEN** use LIFO.
IF 0.4286 < input1 <= 0.5000 **AND** 0.4286 < input2 <= 0.5000 **THEN** use LIFO.
IF 0.5000 < input1 <= 0.5714 **AND** 0.2857 < input2 <= 0.3571 **THEN** use LIFO.
IF 0.5000 < input1 <= 0.5714 **AND** 0.3571 < input2 <= 0.4286 **THEN** use LIFO.
IF 0.5000 < input1 <= 0.5714 **AND** 0.4286 < input2 <= 0.5000 **THEN** use LIFO.
IF 0.5714 < input1 <= 0.6429 **AND** 0.2143 < input2 <= 0.2857 **THEN** use LIFO.
IF 0.5714 < input1 <= 0.6429 **AND** 0.2857 < input2 <= 0.3571 **THEN** use LIFO.
IF 0.5714 < input1 <= 0.6429 **AND** 0.3571 < input2 <= 0.4286 **THEN** use LIFO.
IF 0.6429 < input1 <= 0.7143 **AND** 0.1429 < input2 <= 0.2143 **THEN** use LIFO.
IF 0.6429 < input1 <= 0.7143 **AND** 0.2143 < input2 <= 0.2857 **THEN** use LIFO.
IF 0.6429 < input1 <= 0.7143 **AND** 0.2857 < input2 <= 0.3571 **THEN** use LIFO.
IF 0.6429 < input1 <= 0.7143 **AND** 0.3571 < input2 <= 0.4286 **THEN** use LIFO.
IF 0.7143 < input1 <= 0.7857 **AND** 0.1429 < input2 <= 0.2143 **THEN** use LIFO.
IF 0.7143 < input1 <= 0.7857 **AND** 0.4286 < input2 <= 0.5000 **THEN** use LIFO.
IF 0.7857 < input1 <= 0.8571 **AND** 0.2857 < input2 <= 0.3571 **THEN** use LIFO.
IF 0.7857 < input1 <= 0.8571 **AND** 0.4286 < input2 <= 0.5000 **THEN** use LIFO.
IF 0.8571 < input1 <= 0.9286 **AND** 0.1429 < input2 <= 0.2143 **THEN** use LIFO.
IF 0.8571 < input1 <= 0.9286 **AND** 0.2143 < input2 <= 0.2857 **THEN** use LIFO.
IF 0.8571 < input1 <= 0.9286 **AND** 0.2857 < input2 <= 0.3571 **THEN** use LIFO.
IF 0.8571 < input1 <= 0.9286 **AND** 0.3571 < input2 <= 0.4286 **THEN** use LIFO.
IF 0.9286 < input1 <= 1.0000 **AND** 0.0000 < input2 <= 0.0714 **THEN** use LIFO.
IF 0.9286 < input1 <= 1.0000 **AND** 0.0714 < input2 <= 0.1429 **THEN** use LIFO.
IF 0.9286 < input1 <= 1.0000 **AND** 0.1429 < input2 <= 0.2143 **THEN** use LIFO.
IF 0.9286 < input1 <= 1.0000 **AND** 0.2143 < input2 <= 0.2857 **THEN** use LIFO.

Correctly classified data pairs : 334. Total tested data pairs : 360. ***Performance***: 0.927

[press any key to continue]

Chapter VI

Conclusions

The principle of the mind
is the Great Ultimate
- Zhu Xi

This work implemented a control strategy using a fuzzy methodology proposed by Wang and Mendel. The methodology was first applied to a simple robot motion control case study. Furthermore, the concepts of fuzzy logic were introduced in the field of manufacturing planning through an uncomplicated production floor scenario. Both cases confirmed that a fuzzy controller is relatively simple to construct and that a fuzzy model is in itself a plain structure. Involved mathematical insight was not required and the process relied more on intuition and experience.

Application of the Wang-Mendel methodology affirmed three important advantages. First, the use of the Fuzzy Associative Memory Bank allows the modeller to access and manipulate both numerical and linguistic information under one common framework. As opposed to other "learning from examples" techniques, the rules generated by the fuzzy inference are easy to extract and they are already in an understandable, human readable format.

Second, the design of the fuzzy controller architecture is flexible because membership functions can be defined in a large variety of shapes. In addition, and in order to obtain results that are closer to the desired control output, the modeller can adjust the number of fuzzy regions for each system variable with little effort. Although there is no method explaining what shape or how many fuzzy regions each system variable should have, FLC can perform with stability since both input and output ranges are known through their degree of membership in the interval [0,1].

Third, a FLC can be developed with the use of a straightforward one-pass build-up procedure. The FLC did not require an iterative process for the initial definition of the control rules. This one-pass procedure also eliminated conflicting or redundant rules from the FAM bank.

Calibration of the FLC relies more on intuition and experience and the definition of fuzzy membership functions is a trial and error task. Because of these two characteristics, the initial design and validation of a FLC could become a tedious and time consuming activity, especially for the novice fuzzy modeller.

Table 11 is a performance chart comparing the techniques surveyed in this work. All three techniques performed at levels above 90%. The back propagation neural network technique has a small advantage over the FLC. An important feature in the chart refers to the training and the execution time required to achieve control. The neural network technique is faster in training and overall system development. The FLC and ID3 approaches have acceptable developmental speed, with the FLC yielding better results.

The FLC and ID3 techniques generate rules that are understood intuitively and that can be readily changed as the system learns. This is very difficult in the back propagation neural network and it can only be obtained at high levels of abstraction. Because of this, the FLC and ID3 approaches are more desirable tools for system calibration.

Although not covered in this work, Table 11 presents some categories to complete the comparison among the studied techniques. Stability, for example, is difficult to attain with the back propagation neural network. The response of the system can be "wild data" and it would be very difficult to explain how it was generated. The other two techniques afford better stability since any response of the system can be traced through rules that have well defined input-output domain ranges. In addition, the FLC can handle incremental data (new characteristics fed into the system) and previous knowledge in a

Table 11

Comparison criteria for learning from examples techniques. The FLC and ID3 techniques generate rules that are understood intuitively and that can be readily changed as the system learns. This is very difficult in the back propagation neural network and it can only be obtained at high levels of abstraction. Because of this, the FLC and ID3 approaches are more desirable tools for system calibration.

	ANN Back-propagation	FLC	Machine Learning
Accuracy	98.8	98.1	93.3
Improve Accuracy	?	Increment Granularity	Increment Granularity
Incremental Learning	Difficult. Destroy & start over	Easy. Add	Difficult. Destroy & start over
Start with previous knowledge	Poor	Excellent	Average
Speed:Training & System Development	Fast	Medium	Medium
Execution, real time	Very Fast	Fast	Medium
Explanation of Rules	Very Difficult	Easy	Very Easy
Stability	Difficult	Satisfactory	Satisfactory
Incomplete Data	High	Medium	Low
Noise - Data	High	?	Low

better way than the other two methods which would have to destroy all previous control rules and start training from the beginning when faced with new information.

Implications for Further Research

This work can further be developed in the following areas:

(1) The Wang-Mendel methodology implies that the values of a fuzzy variable can lie outside its domain interval. As verified in the robot-motion case study, some of the initial training data was set to lie outside the domain interval. This apparently had no burden on the final control output. This calls for the re-formulation of the domain intervals to observe the effect on both the architecture of the controller (number of fuzzy regions and their shape) and the control response.

(2) A limitation of the Wang-Mendel methodology is that the memory or size of the FAM bank increases as the number of training data sets increases. Therefore, it is of interest to formulate a process where rules with higher generalization capability remain in the FAM bank, other rules are simply removed. This could be a iterative process that includes new rules into the FAM bank by checking conflicts between incoming and existing rules, assigning a degree to each rule, and discarding the one with the lowest degree, just as suggested by step number three of the methodology.

(3) Research on the applicability of fuzzy logic on job shop scheduling should consider a more complex scenario. The purpose should be such as to assign a set of machines to a set of jobs for the required processing. The jobs should consist of one or more operations that must follow a particular flow among the available machines. Due to different machine configurations, capabilities, and setups, each machine should be assumed to require a different time interval to perform any type of operation.

(4) Most research on job shop scheduling has considered only static problems. To handle jobs with variable arrival times, a dynamic scheduling system is needed. At this level, a fuzzy model becomes of interest because of its ability to handle incremental data. Also, the rules of a fuzzy model are intuitively understood and can be changed as the system learns; this can be an advantage when handling jobs with different arrival times, changes due to machines failures, and new processing requirements.

SELECTED REFERENCES

Afentakis, P., "Maximum Throughput in Flexible Manufacturing Systems," Proceedings of the Second ORSA/TIMS Conference on Flexible Manufacturing Systems, K. Stecke and R. Suri (Eds.), Elsevier, 1986, p. 509-520.

Bunnang, J. and Smith, S., "A Multifactor Priority Rule for Job Shop Scheduling Using Computer Search," Transactions: Industrial Engineering Research and Development, Vol. 17, No. 2, 1985, p. 141-146.

Brubaker, D. I., "Fuzzy Logic System solves control problem," EDN, June 1992, p. 121-127.

Brubaker, D. I., "Fuzzy-Logic basics: intuitive rules replace complex math," EDN, June 1992, p. 111-116.

Barron, J., "Putting Fuzzy Logic into Focus," Byte, April 1993, p. 111-118.

Cen, Y., Lin, K., and Hsu, S., "A self learning Fuzzy Controller," 1992 IEEE International Conference on Fuzzy Systems, San Diego, California, 1992, p. 189-196.

Chang, Y. and Sullivan, R., ""Real-Time Scheduling of FMS," presented at TIMS/ORSA San Francisco Meeting, May 1984.

Conway, R. and Maxwell, W., "Network Dispatching by Shortest Operation Discipline," Operations Research, Vol. 10, 1962, p. 51.

Cox, E., "Adaptive Fuzzy Systems," IEEE Spectrum, February 1993, p. 27-31.

Critchlow, A. J., Introduction to Robotics, Macmillan Publishing Company, New York, 1985.

Chiu, S. L., "In Control Version 1.4, User's Guide," Rockwell International Science Center, Rockwell International Corp. February 1991, p1.25.

Chryssolouris, G., Pierce, J. and Cobb, W., "A Decision Making Approach to the Operation of FMS," Proceedings of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Applications, Cambridge, Massachusetts, Elsevier Science Publishers B. V., 1989, p. 355-360.

Dummermuth, E. H., "Fuzzy Logic Controller Properties," Engineering Report No. PC6549, Allen-Bradley Control Systems, 1991.

Dummermuth, E. H., "Fuzzy Logic Controller," Engineering Report No. PC6547, Allen-Bradley Control Systems, 1991.

Dreyfus, H., What computers can't do: *a critique of AI*, Harper and Row, New York, 1979.

Daugherity, W., Rathakrishnan, B. and Yen, J., "Performance evaluation of a self-tuning Fuzzy Controller," 1992 IEEE International Conference on Fuzzy Systems, San Diego, California, 1992, p. 389-397.

"Fuzzy Models - What are they, and Why?," IEEE Transactions on Fuzzy Systems, Vol. 1, No. 1, 1993, p. 1-5.

Foo, Y. and Takefuji, Y., "Integer Linear Programming Neural Networks for Job-Shop Scheduling," Proceedings of the IEEE International Conference on Neural Networks, published by IEEE TAB, 1988, p. II341-II348.

Fox, M. and McDermott, J., "The Role of Databases in Knowledge-Based Systems," On Knowledge Base Management System, M. Brodie and J. Mylopoulos (Eds.), 1986, p. 407-444.

Forsyth, R. and Rada, R., "Machine Learning: *Applications in Expert Systems and Information retrieval*," Ellis Horwood Limited, England, 1986.

Groover, M. P., Weiss, M., Nagel, R. N. and Odrey, N. G., Industrial Robotics: *Technology Programming and Applications,* McGraw-Hill Book Company, New York, 1986.

Gere, W., "Heuristics in Job Scheduling," Management Science, Vol. 13, 1966, p. 167.

Gross, J., "Intelligent Feedback Control for Flexible Manufacturing Systems," Proceedings of the Second ORSA/TIMS Conference on Flexible Manufacturing Systems, K. Stecke and R. Suri (Eds.), 1989, p 145-156.

Gottwald, S. and Pedrycz, W., "Problems of the design of Fuzzy Controllers," Approximate Reasoning in Expert Systems, M.M. Gupta et al. (Eds.), 1985, p. 393-405.

Gupta, M. M. and Tsukamoto, Y., "Fuzzy Logic Controllers: A perspective," Proceedings of the Joint Automatic Control Conference, San Francisco, 1980, p. FA10-C.

Hershauer, J., and Ebert, J., "Search and Simulation Selection of a Job Shop Scheduling Rule," Management Science, Vol. 21, 1974, p. 883.

Hecht-Nielsen, R., "Applications of Counterpropagation Networks," Neural Networks, 1988, p 131-140.

Hutchinson, J., Leong, K., Snyder, D. and Ward, P., "Scheduling for Random Job Shop Flexible Manufacturing Systems," Proceedings of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems, K. Stecke and R. Suri (Eds.), Elsevier, 1989, p. 161-166.

Hirota, K., Arai, Y., and Hachisu, S., "Fuzzy controlled robot arm playing two-dimensional ping-pong game," Fuzzy Sets and Systems, Vol. 32, 1989, p. 159-169.

Jain, P. and Agonino, A.M., "Calibration of Fuzzy Linguistic Variables for Expert Systems," Expert Systems Technology Lab, University of California, Berkeley, Working Paper # 87-0803-3, 1988.

Kosko, B., Neural Networks and Fuzzy Systems: *A dynamical Systems Approach to Machine Intelligence*, Prentice Hall, Englewood Cliffs, New Jersey, 1992.

Koh, K., Cho, H. S., Kim, S. and Jeong, I., "Application of a self-organizing fuzzy control to the joint control of a puma-760 robot," IEEE International Workshop on Intelligent Robots and Systems, 1990, p. 537-542.

Kusiak, A., Intelligent Manufacturing Systems, Prentice Hall, 1990.

King, P. J. and Mamdani, E. H., "The Application of Fuzzy Control Systems to Industrial Processes," Round Table on Fuzzy Automata and Decision Processes, Proceedings of the Sixth Triennal IFAC World Congress, Cambridge/Boston, 1975.

Lippmann, R., Gold, B. and Malpass, M., "A Comparison of Hamming and Hopfield Neural Nets for Pattern Classification," Massachusetts Institute of Technology, Lincoln Laboratory, Technical Report 769, May 1987.

Lee, C.C., "Fuzzy Logic in Control Systems: Fuzzy Logic Controller - I, II," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 20, No. 2, 1990, p.404-435.

Lokov, D., "Adaptive robot under fuzzy control," Fuzzy Sets and Systems, Vol. 17, 1985, p. 1-8.

Meredith, D. L., Karr, C. L. and Krishna Kumar, K., "The use of Genetic Algorithms in the design of Fuzzy Logic Controllers," Second Workshop on Neural Networks: Academic/Industrial/NASA/Defense, WNN-AIND-91, 1991, p. 695-702.

Mamdani, E. H., "Applications of Fuzzy Algorithms for Simple Dynamic Plant," Proceedings of IEE, Vol. 121, No. 12, 1974, p. 1585-1588.

Mamdani, E. H., "The application of Fuzzy Control Systems to Industrial Processes," Automat, Vol. 13, No. 3, 1977, p. 235-242.

Mamdani, E. H. and Assilian, S., "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," International Journal of Man Machine Studies, Vol. 7, 1975, p 1-13.

McNeill, D. and Freiberger, P. , Fuzzy Logic, Simon&Schuster, New York, 1993.

Murakami, L., Takemoto, F. and Fujimura, H., and Ide, E., "Weld line tracking control of arch welding robot using fuzzy logic controller," Fuzzy Sets and Systems, Vol. 32, 1989, p 221-237.

Medungadi, A., "A fuzzy robot controller: hardware implementation," 1992 IEEE International Conference on Fuzzy Systems, San Diego, California, 1992, p. 1325-1331.

Nguyen, D. and Widrow, B., "The Truck Bucker-Upper: An example of Self-learning in Neural Networks," IEEE Control Systems Magazine, Vol. 10, No. 3, 1990, p. 18-23.

Norwich, A. M. and Turksen, I. B., "The Construction of Membership Functions," Fuzzy Set and Possibility Theory, R.R. Yayer (Ed), 1982.

Nomura, H., Hayashi, I. and Wakami, N., "A Learning Method of Fuzzy Inference Rules by Descent Method," 1992 IEEE International Conference on Fuzzy Systems, San Diego, California, 1992, p. 203-210.

Odrey, N. and Wilson, G., "Real Time Control and Coordination for Flexible Manufacturing Cells," 15th Conference on Production Research and Technology: Advances in Manufacturing Systems Integration and Processes, University of California at Berkeley, Berkeley, California, January 9-13, 1989, p. 445-453.

Pappis, C. C. and Mamdani, E. H., "A Fuzzy Logic Controller for a Traffic Junction," IEEE Transactions on Systems, Man, and Cybernetics, Vol. Snc-7, No. 10, 1977, p.707-716.

Rabelo, L., "A Hybrid Artificial Neural Networks and Knowledge-Based Expert Systems Approach to Flexible Manufacturing System Scheduling," Ph. D. Thesis, University of Missouri-Rolla, 1990.

Raman, N., Talbot, F. and Rachamadugu, R., "Simultaneous Scheduling of Machines and Material Handling Devices in Automated Manufacturing," Proceedings of the Second ORSA/TIMS Conference on Flexible Manufacturing System, K. Stecke and R. Suri (Eds.), Elsevier, 1986, p. 455-465.

Ralston, P. A. S. and Ward, T. L., "Fuzzy Control of Industrial Processes," Applications of Fuzzy Set Methodologies in Industrial Engineering, G. Evans, W. Karwowski, and W. Wilhelm (Eds.), 1989, p. 29-46.

Raju, G. V. S., and Zhou, J., "Fuzzy rule based approach for robot motion control," 1992 IEEE International Conference on Fuzzy Systems, San Diego, California, 1992, p. 1349-1356.

Sugeno, M., Yasukawa, T., "A Fuzzy Logic-based approach to Qualitative Modeling," IEEE Transactions on Fuzzy Systems, Vol. 1, No. 1, 1993, p. 7-26.

Sugeno, M. and Murakami, K., "An experimental study of fuzzy parking control using a model car," Industrial Applications of Fuzzy Control, M. Sugeno (Ed.), 1985, p. 125-138.

Sibightroth, J., "Creating Fuzzy Micros," Embedded Systems Programming, December 1991, p. 20-34.

Schwartz, D. G., Klir, G. J., "Fuzzy Logic flowers in Japan," IEEE Spectrum, July 1992, p. 32-35.

Simpson, P. K., Artificial Neural System, Pergamon Press, New York, 1990.

Tang, C., "A Job Scheduling Model for a Flexible Manufacturing Machine," Working Paper 2/85, Yale University, 1985.

Tang, C. and Denardo, E., "Models Arising from a Flexible Manufacturing Machine Part II: Minimization of the Number of Switching Instants," UCLA, Western Management Science Institute, Working Paper No. 342, 1986.

Tong, R. M., "The evaluation of Fuzzy Models derived from experimental data," Fuzzy Sets and System, Vol. 4, 1980, p 1-12.

Takagi, T. and Sugeno, M., "Derivation of Fuzzy Control Rules from Human Operator's Control Actions," IFAC Fuzzy Information, Marseille, France, 1983, p. 55-60.

Triantaphyllou, E. and Mann, S., "An Evaluation of the Eigenvalue Approach for determining the membership values in fuzzy sets," Fuzzy Sets and Systems, Vol. 35, 1990, p. 295-301.

Wiggins, R., "Docking a Truck: Fuzzy," AI Expert, May 1992, p. 28-35.

Wang, B. H. and Vachtsevanos, G., "Learning Fuzzy Logic Control: An Indirect Control Approach," 1992 IEEE International Conference on Fuzzy Systems, San Diego, California, 1992, p. 297-304.

Wang, L. X. and Mendel, J. M., "Generating Fuzzy Rules by Learning from Examples," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 6, 1992, p. 1414-1427.

Yasunobu, S., Miyamoto S. and Ihara, H., "Automatic Train Operation by Predictive Fuzzy Control," Industrial Applications of Fuzzy Control, M. Sugeno (Ed.), 1985, p. 1-18.

Yagishita, O., Itoh, O. and Sugeno, M., "Application of a fuzzy reasoning to the water purification process," Industrial Applications of Fuzzy Control, M. Sugeno (Ed.), 1985, p. 19-40.

Yasunobu, S. and Hasegawa, T., "Evaluation of an Automatic Container Crane operation system based on predictive fuzzy control," Control Theory Advance Technology, Vol. 2, No. 3, 1986, p 419-432.

Zadeh, L. A., "Fuzzy Sets," Information and Control, Vol. 8, 1965, p 338-353.

Zadeh, L. A., "The Concept of Linguistic Variable and its Application to Approximate Reasoning -I, II, III," Information Sciences, 1975, Vol. 8. p 199-249, Vol. 8, p 301-357, Vol. 9, p 43-80.

Zadeh, L. A., "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems," Fuzzy Sets and Systems, Vol. 11, 1983, p 199-227.

Zadeh, L. A., "The Calculus of Fuzzy IF/THEN Rules", AI Expert, March 1992, p. 23-27.

Zadeh, L. A., "Fuzzy Sets, Usuality and Commonsense Reasoning," EECS Technical Report, University of California, Berkeley, 1985.

APPENDIX A

"LEARNING FROM EXAMPLES" METHODS:

A. THE BACKPROPAGATION NEURAL NETWORK

B. QUINLAN'S ID3 MACHINE LEARNING

Learning from Examples

This section will briefly describe two control methods that have the ability of learning from examples: neural networks and machine learning. Since these two architectures are used in Chapter V for comparison against the FLC performance, a technical reference for clarity and understanding is necessary.

Artificial Neural Networks

An Artificial Neural Network (ANN), is a parallel, distributed information processing structure consisting of processing elements (which can posses a local memory and carry out localized information processing operations) interconnected together with unidirectional signal channels called connections. Each processing element has a single output connection which branches ("fans out") into as many collateral connections as desired (each carrying the same signal-the processing element output signal). The processing element output signal, can be of any mathematical type desired. All of the processing that goes on within each processing element must be completely local; i.e. it must depend only upon the current values of the input signal arriving at the processing element via impinging connections and upon values stored in the processing element's local memory (Hecht-Nielsen, 1988). The key elements of ANN are the distributed representation, the local operations, and the nonlinear processing. These attributes emphasize two of the primary applications of ANNs: situations where only a few decisions are required form a massive amount of data and situations where a complex nonlinear mapping must be learned.

ANN models are based on how the brain might process information. In reference to human information processing problems, the process performed in the mind is simulated in a conventional digital computer to gain an understanding of why the mind is so adaptive and resilient and the computer so rigid and precise. The processing elements (PEs), also referred to as nodes or neurons, are the ANN components where most, if not all, of the computing is done. Figure A displays the anatomy of a generic PE. The input signals come form either the environment or outputs of other PEs and form an input vector $A = (a_1, \ldots\ldots, a_i, \ldots\ldots a_n)$ where a_i is the activity level of the *ith* PE or input (Simpson 1990). Associated with each connected pair of PEs is an adjustable value (i.e., a system variable) called a weight (also referred to as long-term memory). The collection of weights that abbots the *jth* PE, b_j, forms a vector $W_j = (w_{1j}, \ldots\ldots, w_{ij}, \ldots\ldots, w_{nj})$, where the weight w_{ij} represents the connection strength form the input PE to the output PE. Sometimes there is an additional parameter Θ_j modulated by the weight w_{0j} that is associated with the inputs. This term is considered to be an internal threshold value that must be exceeded for there to be any PE activation. The weights W_j, their associated PE values A, and the possible extra parameter Θ_j are used to compute the output value b_j. This computation is typically performed by taking the dot product of A and W_j, subtracting the threshold, and passing the result through a threshold function f(). Mathematically this operation is defined as:

$$b_j = f(A \cdot W_j - w_{0j}\Theta_j)$$

or, in point-wise notation, as:

$$b_j = f(\sum_{i=1}^{n} a_i w_{ij} - w_{0j}\Theta_j)$$

ANN architectures are formed by organizing PEs into fields, also called layers, and linking them with weighted interconnections. Characteristics of these topologies include connection types, interconnection schemes, and field configurations (Simpson 1990).

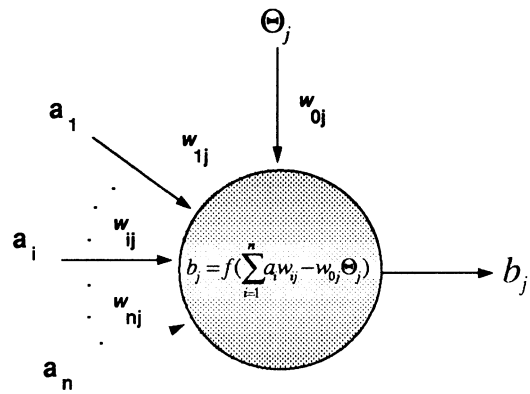

Figure A. Topology of the generic PE b_j. Inputs form the vector $A = (a_1, \ldots, a_i, \ldots a_n)$. The threshold is Θ_j. Each input and threshold has a corresponding weight value w_{ij} which represents the connection strength form the *ith* input a_i to the *jth* output b_j with the threshold being the weight w_{0j}. The collection of the weights forms the n+1 dimensional vector $W_j = (w_{1j}, \ldots, w_{ij}, \ldots, w_{nj})$. The output is shown as a function of its inputs, illustrating the local behavior of the PEs.

There are two types of connections. *Excitatory* connections increase a PE's activation and are typically represented by positive signals. *Inhibitory* connections decrease a PE's activation and are typically represented by negative signals. Interconnection schemes include *intra-field* connections which are connections between PEs in the same layer; *inter-field* connections which are connections between PEs in different layers(inter-field signals propagate in either feed forward or feedback fashion); and *recurrent* connections that loop and connect back to the same PE. Feed forward signals only allow information to flow amongst PEs in one direction. Feedback signals

allow information to flow in either direction. A field configuration is commonly composed by an *input layer*, or the field that receives input signals form the environment, and an *output layer*, or the field that emits signals to the environment. Any fields that lie between the input and output layers are called *hidden layers* and have no direct contact with the environment.

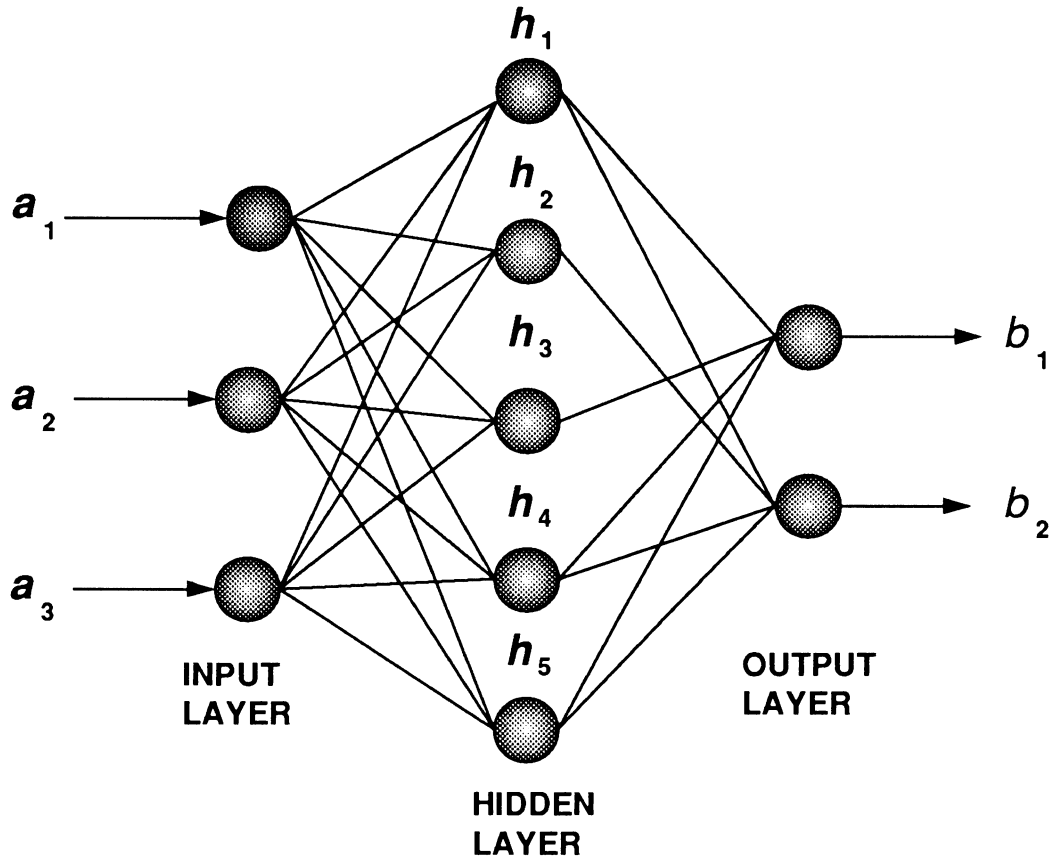

Figure B. Topology of the elementary backpropagation ANN, a supervised learning-feed forward recall ANN. This three layer ANN uses the backpropagation algorithm to adjust the connections between the input neurons and the neurons in the hidden layer and the later with the output neurons. Notice that each neuron presents a topology similar to the one in Figure A.

In this work, a backpropagation ANN will be used to compare its result with those obtained with the FLC. The elementary backpropagation is a three-layer ANN with feed forward connections. The backpropagation algorithm performs the input to output mapping by minimizing a cost function. The cost function is minimized by making weight

connection adjustments according to the error between the computed and the desired output. The cost function that is typically minimized is the squared error-the squared difference between the computed output and the desired output for each PE across all patterns in the data set. The weight adjustment procedure is derived by computing the change in the cost function with respect to the change in each weight. The element that makes the backpropagation algorithm so powerful is that this derivation is extended to find the equation for adapting the connections between the input and hidden layers of a multilayer ANN, as well as the penultimate layer to output layer adjustments The key element of the extension to the hidden layer adjustments is the realization that each intermediate layer PE's error is proportionally weighted sum of the errors produced at the output layer. Figure B illustrates the topology of the elementary backpropagation ANN.

Finally, the primary application of the backpropagation ANN is any situation that requires the acquisition of a complex non-linear mapping, provided that the system can wait long enough for the ANN to learn the mapping.

Machine Learning

Machine learning is the key to machine intelligence just as human learning is key to human intelligence. When a computer system improves its performance at a given task over time, without re-programming, it can be said to have learned something. Machine learning, investigates computer systems which model the process of learning in some way.

In a simple basis, all systems designed to modify and improve their performance share certain important common features. Figure C is a diagram of the four major

components of a typical learning system. essentially this sketches a pattern recognizer which learns to associate input descriptions with output categories.

There are two terms that are of interest in the examination of practical learning methods - "description language" and "training set" (Forsyth and Rada 1986). The description language is the notation or formalism in which the knowledge of the system is expressed. There are two kinds of description language which are important. The first is the notation used to represent the input examples. One of the simplest of input formats is the feature vector. Each aspect of the input example is measured numerically, and the vector of measurements defines the input situation. The second kind of description language is that chosen to represent the rules themselves. The expressiveness of the description language in which the rules are formulated is critical to the success of any learning algorithm. It also has a bearing on how readily the knowledge can be understood, and hence on whether it can be transferred to people.

The notion of a training set is important in understanding how a machine learning system is tested. Typically there is a database for examples for which the solutions are known. The system works through these instances and derives a rule or set of rules for associating input descriptions with output decisions. The rules, nevertheless, must be tested on cases other than those form which they were derived. Therefore, there should be another database, the test set, of the same kind but containing unseen data. If the rules also apply successfully to these fresh cases, the confidence in the system is also increased.

Assuming that the description language is adequate, the next problem is to automate the generation of useful descriptions in that language. One way of looking at this problem is to search through the space of all possible descriptions form those which are

valuable for the task in hand (Forsyth and Rada). The number of syntactically valid descriptions is astronomical; and the more expressive the description language, the more explosive this combinatorial problem is. Clearly, a way must be found of guiding the search and thereby ignoring the vast majority of potential descriptions, or concepts, which are useless for the current purpose.

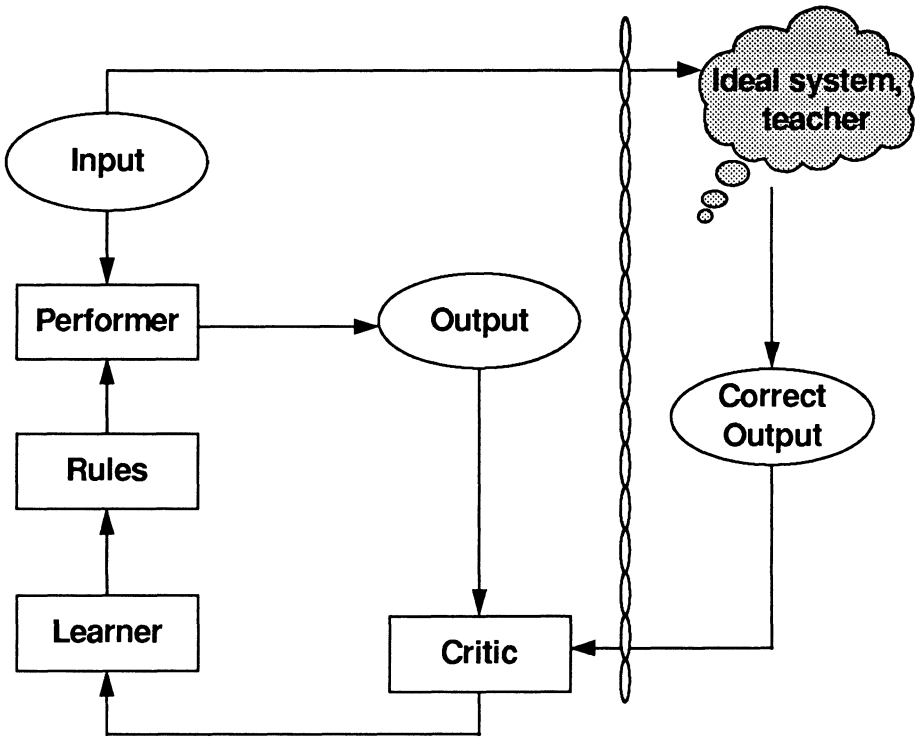

Figure C. This outline shows the logical layout of a learning system. To the left of the chain line is the *learning* system, to the right is the *ideal* system or teacher, whose behavior it is trying to match. The *critic* compares the actual with the desired output and passes on feedback to the *learner*. The learner attempts to modify the rules (or knowledge base) to improve the system's performance. The *performer* uses those rules to guide its performance at the task (but leaves the job of amending them to the learner module)

The learning system can start with an initial description (which may be randomly created) and apply transformation operators to generate its successors. The successors are new

descriptions for testing. The most important transformation operators are generalization and specialization. In this context, generalization of a rule means that it applies to more cases and specialization means that it applies to fewer. A number of searching methods have worked well with noise-free training instances. One of these methods is Quinlan's ID3.

Quinlan's ID3 (Interactive Dichotomizer 3) creates a decision tree that it is not particularly robust in the face of noisy data, though it could be improved in this respect if it did not always look for a "perfect" rule (Forsyth and Rada 1986). The algorithm works as follows. First, a random subset of size W form the training set (the "window") is selected. Second, the Concept Learning System (CLS) algorithm is applied is applied to form a rule for the current window. Then, the whole database is scanned to find exceptions to the latest rule. Finally, if there are exceptions, some of them are inserted into the window (possibly replacing existing examples) and the procedure is repeated form the second step forward; otherwise, stop and display the rule. This procedure, actually throws away the latest rule and starts again form scratch on each cycle.

The method of window selection is sometimes termed "exception-driven filtering". The need for windows arises because the main database may contain hundreds of thousands of cases, and hence be too large to be processed in an acceptable time. Figure D illustrates a typical ID3 discrimination tree.

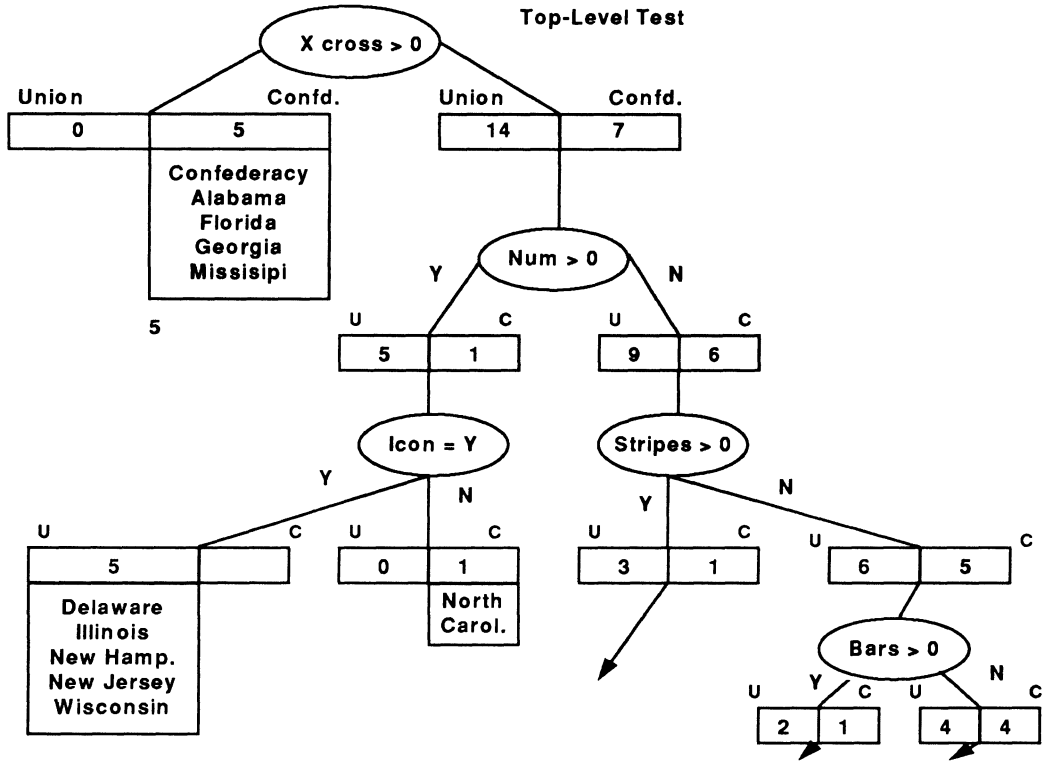

Figure D. The system is given data about the flags of various states of the United States. The objective is for it to learn how to distinguish states that joined the Confederacy in the Civil War, form those that stayed loyal to the union Terminal nodes are shown with the names of the states to which they apply hanging form them. terminal nodes contain only one type of data. Unfortunately, to achieve this often requires splitting of very small subsets. Groups 3:1, 2:1 and 4:4 still need to be further divided.

APPENDIX B

ROBOT MOTION CONTROL CASE STUDY

SAMPLE DATA SETS

Robot Motion Control Case Study: *Sample Data Sets*

Format: *slope*
x_ini y_ini angle1-ini Angle2_ini
x_end y_end angle1_end angle2_end

30.000000
1.746299 0.310937 -15.155231 53.642731
1.747965 0.360909 -17.419941 55.031830

24.000000
1.649381 0.498457 -11.163102 59.066528
1.651462 0.548413 -13.696254 61.023022

95.000000
1.429598 0.788287 -3.641915 68.038200
1.430125 0.838284 -6.414937 70.574928

90.000000
0.624502 1.029791 8.530807 102.807785
0.625058 1.079787 5.791884 105.947968

86.000000
0.200000 0.200000 -29.519428 161.556183
0.200581 0.249997 -36.869896 163.739792

9.000000
0.256073 1.627845 49.229664 63.814095
0.261595 1.677539 46.540508 69.039337

54.000000
1.252785 1.038486 7.078477 67.772392
1.253711 1.088478 4.108103 71.097328

24.000000
0.219550 1.772620 59.702328 46.728722
0.221631 1.822577 56.202698 53.473671

92.000000
1.602491 0.200000 -26.922092 71.572205
1.603035 0.249997 -29.037271 72.302658

98.000000
1.790977 0.404738 -8.216876 44.919048
1.791487 0.454735 -10.620233 46.708874

88.000000
1.236596 0.317599 -33.262871 99.622063
1.237164 0.367596 -35.925831 100.659966

28.000000
0.413214 0.424371 -22.814150 143.262955
0.414999 0.474339 -27.009935 145.546097

60.000000
0.200000 0.581168 1.689037 141.320175
0.200833 0.631161 -1.092933 144.205658

78.000000
1.434144 0.818075 -1.845947 66.041306
1.434785 0.868071 -4.655846 68.714943

46.000000
1.564907 0.705359 -3.869953 59.239998
1.565994 0.755347 -6.615048 61.755653

35.000000
1.320558 0.769169 -7.174772 77.917175
1.321986 0.819149 -9.952576 80.343155

49.000000
1.463243 1.139877 19.724304 38.746281
1.464264 1.189866 15.954353 43.928974

74.000000
0.682587 1.076726 9.979115 97.575104
0.683263 1.126721 7.227849 100.799232

35.000000
1.410548 0.783286 -4.392860 69.878998
1.411976 0.833266 -7.179693 72.446793

44.000000
0.893276 0.958445 0.802851 95.252510
0.894413 1.008432 -2.058076 98.147377

APPENDIX C

ROBOT MOTION CONTROL CASE STUDY

DEGREE OF MEMBERSHIP FUNCTIONS
FOR INPUTS AND OUTPUTS

Robot Motion Control Case Study: ***Degree of Membership Functions for Inputs and Outputs***

X Coordinate

Interval	Membership Function	Fuzzy Value
$0.000 \le x\_initial < 0.333$	$y = 3x$	*S2*
$0.333 \le x\_initial < 0.333$	$y = -3x + 2$	*S2*
$0.333 \le x\_initial < 0.666$	$y = 3x - 0.999$	*S1*
$0.666 \le x\_initial < 0.999$	$y = -3x + 3$	*S1*
$0.666 \le x\_initial < 0.999$	$y = 3x - 2$	*CE*
$0.999 \le x\_initial < 1.333$	$y = -3x + 3.999$	*CE*
$0.999 \le x\_initial < 1.333$	$y = 3x - 3$	*B1*
$1.333 \le x\_initial < 1.666$	$y = -3x + 5$	*B1*
$1.333 \le x\_initial < 1.666$	$y = 3x - 3.999$	*B2*
$1.666 \le x\_initial < 2.000$	$y = -3x + 6$	*B2*

Y Coordinate

Interval	Membership Function	Fuzzy Value
$0.000 \le y\_initial < 0.333$	$y = 3x$	*S2*
$0.333 \le y\_initial < 0.333$	$y = -3x + 2$	*S2*
$0.333 \le y\_initial < 0.666$	$y = 3x - 0.999$	*S1*
$0.666 \le y\_initial < 0.999$	$y = -3x + 3$	*S1*
$0.666 \le y\_initial < 0.999$	$y = 3x - 2$	*CE*
$0.999 \le y\_initial < 1.333$	$y = -3x + 3.999$	*CE*
$0.999 \le y\_initial < 1.333$	$y = 3x - 3$	*B1*
$1.333 \le y\_initial < 1.666$	$y = -3x + 5$	*B1*
$1.333 \le y\_initial < 1.666$	$y = 3x - 3.999$	*B2*
$1.666 \le y\_initial < 2.000$	$y = -3x + 6$	*B2*

SLOPE

Interval	Membership Function	Fuzzy Value
$0.00 \le slope < 20.0$	$y = 0.05x$	*S2*
$20.0 \le slope < 40.0$	$y = -0.05x + 2$	*S2*
$20.0 \le slope < 40.0$	$y = 0.05x - 1$	*S1*
$40.0 \le slope < 60.0$	$y = -0.05x + 3$	*S1*
$40.0 \le slope < 60.0$	$y = 0.05x - 2$	*CE*
$60.0 \le slope < 80.0$	$y = -0.05x + 4$	*CE*
$60.0 \le slope < 80.0$	$y = 0.05x - 3$	*B1*
$80.0 \le slope < 100.0$	$y = 0.05x + 5$	*B1*
$80.0 \le slope < 100.0$	$y = 0.05x - 4$	*B2*
$slope \ge 100.0$	$y = 1.0$	*B2*

DELTA-ANGLE1 (Difference between initial and final values for theta1)

Interval	Membership Function	Fuzzy Value
$0.000 \le \Delta\theta_1 < 1.714$	$y = 0.583x$	*S3*
$1.714 \le \Delta\theta_1 < 3.428$	$y = -0.583x + 1.999$	*S3*
$1.714 \le \Delta\theta_1 < 3.428$	$y = 0.583x - 0.999$	*S2*
$3.428 \le \Delta\theta_1 < 5.143$	$y = -0.583x + 2.999$	*S2*
$3.428 \le \Delta\theta_1 < 5.143$	$y = 0.583x - 1.999$	*S1*
$5.143 \le \Delta\theta_1 < 6.857$	$y = -0.583x + 3.999$	*S1*
$5.143 \le \Delta\theta_1 < 6.857$	$y = 0.583x - 2.999$	*CE*
$6.857 \le \Delta\theta_1 < 8.571$	$y = -0.583x + 4.999$	*CE*
$6.857 \le \Delta\theta_1 < 8.571$	$y = 0.583x - 3.999$	*B1*
$8.571 \le \Delta\theta_1 < 10.285$	$y = -0.583x + 5.999$	*B1*
$8.571 \le \Delta\theta_1 < 10.285$	$y = 0.583x - 4.999$	*B2*
$10.285 \le \Delta\theta_1 < 12.00$	$y = -0.583x + 6.999$	*B2*
$10.285 \le \Delta\theta_1 < 12.00$	$y = 0.583x - 5.999$	*B1*
$\Delta\theta_1 \ge 12.00$	$y = 1.0$	*B1*

DELTA-ANGLE2 (Difference between initial and final values for theta2)

Interval	Membership Function	Fuzzy Value
$-30.000 \le \Delta\theta_2 < -25.714$	$y = 0.233 + 6.999$	*S3*
$-25.714 \le \Delta\theta_2 < -21.428$	$y = -0.233x - 4.999$	*S3*
$-25.714 \le \Delta\theta_2 < -21.428$	$y = 0.233x + 5.999$	*S2*
$-21.428 \le \Delta\theta_2 < -17.143$	$y = -0.233x - 3.999$	*S2*
$-21.428 \le \Delta\theta_2 < -17.143$	$y = 0.233x + 4.999$	*S1*
$-17.143 \le \Delta\theta_2 < -12.857$	$y = -0.233x - 2.999$	*S1*
$-17.143 \le \Delta\theta_2 < -12.857$	$y = 0.233x + 3.999$	*CE*
$-12.857 \le \Delta\theta_2 < -8.571$	$y = -0.233x - 1.999$	*CE*
$-12.857 \le \Delta\theta_2 < -8.571$	$y = 0.233x + 2.999$	*B1*
$-8.571 \le \Delta\theta_2 < -4.285$	$y = -0.233x - 0.999$	*B1*
$-8.571 \le \Delta\theta_2 < -4.285$	$y = 0.233x + 1.999$	*B2*
$-4.285 \le \Delta\theta_2 < -0.000$	$y = -0.233x$	*B2*
$-4.285 \le \Delta\theta_2 < -0.000$	$y = 0.233x + 0.999$	*B3*
$\Delta\theta_2 \ge 0.000$	$y = 1.0$	*B3*

APPENDIX D

ROBOT MOTION CONTROL CASE STUDY

INITIAL FUZZY RULES: SAMPLE

Robot Motion Case Study: ***Rules generated for each data set***

Format: fuzzy values: slope x_ini y_ini
fuzzy values: delta_angle1 delta_angle2
degree of membership values for inputs and outputs

B2 S2 S1
S3 B3
0.761103 0.932811 0.500000 0.678919 0.675877

B2 S2 S2
S3 B3
0.948143 0.504629 0.800000 0.522328 0.543485

B1 S1 B2
S2 B2
0.711206 0.635139 0.750000 0.617597 0.591903

S1 CE B2
S2 B2
0.873506 0.910627 0.500000 0.597705 0.732709

S2 S2 B1
CE B2
0.600000 0.600000 0.700000 0.712227 0.509509

S2 B2 S2
S2 B2
0.768219 0.883535 0.450000 0.568675 0.780777

B1 CE CE
S2 B2
0.758355 0.884542 0.700000 0.732719 0.775818

S2 B2 S2
S2 B1
0.658650 0.682140 0.800000 0.958549 0.573821

B2 S2 B2
S3 B3
0.807473 0.600000 0.600000 0.766145 0.829561

B2 S2 B2
S3 B3
0.627069 0.785786 0.900000 0.598042 0.582374

B1 S2 B1
S2 B3
0.709788 0.952797 0.600000 0.553394 0.757823

S2 S2 S2
S2 B2
0.760358 0.726887 0.600000 0.552458 0.532733

S2 S1 CE
S2 B2
0.600000 0.743504 1.000000 0.622816 0.673279

B1 S1 B1
S2 B2
0.697568 0.545775 0.900000 0.639108 0.623849

B2 S1 S1
S2 B2
0.694721 0.883923 0.700000 0.601306 0.586986

B1 S1 S1
S2 B2
0.961674 0.692493 0.750000 0.620386 0.566062

B1 CE S1
S2 B2
0.610271 0.580369 0.550000 0.800862 0.790705

S1 CE B1
S2 B2
0.952239 0.769822 0.700000 0.604905 0.752297

B1 S1 S1
S2 B2
0.768356 0.650142 0.750000 0.625653 0.599152

CE CE S1
S2 B2
0.679828 0.875335 0.800000 0.668874 0.675469

APPENDIX E

ROBOT MOTION CONTROL CASE STUDY

111 CONTROL FUZZY RULES

Robot Motion Case Study: ***Final 111 control fuzzy rules***

Format: fuzzy values: slope x_ini y_ini
fuzzy values: delta_angle1 delta_angle2
degree of membership values for inputs and outputs Rule #

B2 S2 S1
S3 B3
0.598484 0.847200 0.650000 0.656099 0.757431 1

B2 S2 S2
S3 B3
0.979134 0.777449 0.700000 0.554022 0.575643 2

B1 S1 B2
S2 B3
0.900378 0.963696 0.650000 0.590483 0.518111 3

S1 CE B2
S2 B2
0.873506 0.910627 0.500000 0.597705 0.732709 4

S2 S2 B1
S2 B2
0.600000 0.714683 0.700000 0.878301 0.634247 5

S2 B2 S2
S2 B1
0.693023 0.798309 0.800000 0.875801 0.551528 6

B1 CE CE
S2 B2
0.703449 0.628063 0.900000 0.792864 0.863374 7

B2 S2 B2
S3 B3
0.831819 0.600000 0.800000 0.811789 0.811215 8

B1 S2 B1
S3 B3
0.952352 0.600000 0.850000 0.594979 0.837566 9

S2 S2 S2
CE B3
0.905657 0.600000 0.950000 0.695966 0.614663 10

S2 S1 CE
S2 B2
0.578597 0.937556 0.750000 0.902206 0.606216 11

B1 S1 B1
S2 B2
0.697568 0.545775 0.900000 0.639108 0.623849 12

B2 S1 S1
S2 B2
0.930320 0.941048 1.000000 0.591583 0.583858 13

B1 S1 S1
S2 B2
0.961674 0.692493 0.750000 0.620386 0.566062 14

B1 CE S1
S2 B2
0.788375 0.994159 0.550000 0.787280 0.817258 15

S1 CE B1
S2 B2
0.884166 0.744712 1.000000 0.611782 0.754855 16

CE CE S1
S2 B2
0.787081 0.921331 0.800000 0.668058 0.683271 17

S2 S1 B1
S2 B2
0.903761 0.597002 0.950000 0.902675 0.597429 18

S2 B1 CE
S3 B2
0.877547 0.979893 0.700000 0.564831 0.907478 19

S2 B2 CE
S2 B2
0.955583 0.830988 0.850000 0.983568 0.548142 20

S1 B1 B1
S2 B2
0.987549 0.852218 0.700000 0.661528 0.969211 21

S1 B1 S2
S2 B2
0.514116 0.551540 0.850000 0.899970 0.814082 22

S1 B1 B2
S2 B2
0.806162 0.974147 0.950000 0.596057 0.924185 23

S2 S1 S2
S2 B2
0.832886 0.877590 0.750000 0.665109 0.669811 24

B1 B1 B1
S1 B1
0.754460 0.872901 0.550000 0.890845 0.812725 25

B2 S2 B1
S3 B3
0.735758 0.667289 0.650000 0.594978 0.651259 26

S1 CE S1
S2 B2
0.629844 0.892444 0.750000 0.659069 0.686437 27

S1 B2 CE
S2 B2
0.965186 0.885161 0.850000 0.956388 0.631044 28

B1 S2 B2
S3 B3
0.878768 0.814394 0.700000 0.528036 0.709200 29

S2 S1 S1
S2 B2
0.600000 0.684461 1.000000 0.648117 0.670684 30

S2 B1 S1
S3 B2
0.818607 0.871785 0.950000 0.666638 0.887949 31

B2 S2 CE
S3 B3
0.921720 0.686358 0.950000 0.771432 0.787332 32

CE CE CE
S2 B2
0.883495 0.916849 0.600000 0.678421 0.720225 33

CE S1 CE
S2 B3
0.580999 0.533435 0.750000 0.910363 0.579940 34

CE S2 CE
S2 B3
0.875209 0.685296 0.650000 0.883684 0.789341 35

CE B2 B2
S1 B1
0.682222 0.975921 0.800000 0.910156 0.845857 36

CE B2 CE
S1 B1
0.862165 0.867440 0.550000 0.826068 0.917674 37

S1 B1 CE
S2 B2
0.820739 0.926270 0.600000 0.602520 0.941650 38

CE S1 S2
S2 B3
0.534976 0.723659 0.800000 0.867614 0.509274 39

S2 B2 B2
S2 B2
0.643536 0.930374 0.900000 0.652081 0.771771 40

S2 B1 S2
S3 B2
0.600000 0.866958 0.700000 0.746493 0.889033 41

S1 S1 S2
S2 B2
0.876678 0.749795 0.900000 0.945328 0.523195 42

CE B1 CE
S2 B2
0.737032 0.811596 0.550000 0.840353 0.975974 43

B1 CE S2
S2 B2
0.650583 0.669871 0.850000 0.779339 0.860287 44

S1 CE S2
S2 B2
0.663590 0.899119 0.750000 0.572242 0.723981 45

S1 B2 B2
S1 B1
0.957599 0.717192 0.800000 0.853105 0.952028 46

S2 CE S2
S3 B2
0.600000 0.758479 0.850000 0.795517 0.754639 47

S1 S2 S2
S2 B3
0.618429 0.948660 0.650000 0.857408 0.682609 48

B1 CE B1
S2 B2
0.937881 0.641737 0.750000 0.849486 0.907706 49

S2 B2 S1
S2 B1
0.743669 0.794334 0.800000 0.886861 0.539238 50

S1 S2 CE
S2 B3
0.922451 0.922172 0.550000 0.625117 0.620726 51

CE S1 B1
S2 B3
0.853288 0.963060 0.950000 0.700779 0.510520 52

S2 S2 S1
S1 B3
0.659258 0.600000 0.950000 0.650955 0.675204 53

S2 B1 B2
S3 B2
0.694503 0.765824 0.750000 0.624428 0.963609 54

S2 CE B1
S3 B2
0.722694 0.652132 0.700000 0.711064 0.807007 55

B1 S2 S1
S2 B3
0.654960 0.600000 0.800000 0.536787 0.826410 56

CE CE S2
S2 B2
0.598801 0.881110 0.800000 0.643214 0.747711 57

S1 S1 S1
S2 B2
0.925422 0.686021 0.700000 0.996081 0.501690 58

S2 CE B2
S3 B2
0.633215 0.636541 0.850000 0.521729 0.792331 59

B1 B1 S2
S1 B1
0.805271 0.895482 0.950000 0.856299 0.803972 60

CE S2 B1
S2 B3
0.710749 0.710865 0.750000 0.973597 0.780634 61

S1 S1 B1
S2 B2
0.503499 0.967721 0.700000 0.827206 0.510169 62

S2 S1 B2
S2 B2
0.695402 0.678764 0.700000 0.924284 0.580239 63

B2 S1 B2
S2 B2
0.694140 0.826062 0.850000 0.997989 0.963739 64

S1 S2 B1
S2 B3
0.516168 0.600000 0.800000 0.874522 0.797564 65

S2 B2 B1
S2 B2
0.728958 0.869324 0.850000 0.631311 0.801856 66

B1 S2 S2
S2 B3
0.825960 0.843461 0.950000 0.562246 0.683720 67

CE S1 B2
S2 B2
0.891607 0.913743 0.950000 0.738553 0.507723 68

S2 CE CE
S2 B2
0.719036 0.853768 0.650000 0.513869 0.729501 69

CE CE B1
S2 B2
0.889138 0.652396 0.900000 0.672574 0.617330 70

S1 B2 B1
S2 B2
0.662705 0.730151 0.750000 0.805565 0.803694 71

S2 B1 B1
S2 B2
0.526088 0.867525 0.950000 0.506778 0.882813 72

B1 S2 CE
S2 B3
0.972533 0.616601 0.850000 0.517777 0.660761 73

S1 CE CE
S2 B2
0.989688 0.737182 0.750000 0.595516 0.761164 74

S1 S1 CE
S2 B2
0.756500 0.950415 0.650000 0.872031 0.586967 75

B1 S1 CE
S3 B3
0.590249 0.614468 0.950000 0.509834 0.595686 76

B1 S1 S2
S2 B2
0.861680 0.909896 0.700000 0.591385 0.514682 77

B2 CE S1
S1 B1
0.694433 0.591217 0.950000 0.998664 0.858165 78

S2 S2 B2
CE B2
0.600000 0.741024 0.850000 0.587861 0.549773 79

CE S2 S2
S2 B3
0.741847 0.600000 1.000000 0.970106 0.782615 80

S2 CE S1
S2 B2
0.560117 0.703888 0.800000 0.579722 0.705287 81

S1 B1 S1
S2 B2
0.616947 0.851549 1.000000 0.739229 0.997155 82

B2 CE CE
S2 B2
0.674180 0.945577 0.650000 0.891063 0.897688 83

CE B1 S2
S2 B2
0.999319 0.989516 0.650000 0.850737 0.937833 84

S1 S2 B2
S2 B3
0.777086 0.722510 0.900000 0.675217 0.558231 85

S1 B2 S2
S2 B2
0.735977 0.621017 0.800000 0.752424 0.844751 86

B2 S1 CE
S2 B2
0.595896 0.992307 0.750000 0.986992 0.968990 87

S1 B2 S1
S2 B2
0.965675 0.509918 0.900000 0.770227 0.881356 88

S1 S2 S1
S2 B3
0.957173 0.780161 1.000000 0.729443 0.584560 89

B2 S1 S2
S2 B2
0.775809 0.527712 0.950000 0.604557 0.688573 90

CE B1 S1
S2 B2
0.721936 0.771573 0.800000 0.830828 0.966777 91

CE B2 B1
S1 B1
0.842422 0.721334 0.800000 0.919001 0.958036 92

B1 CE B2
S2 B2
0.742421 0.890068 0.600000 0.845759 0.877551 93

B1 B1 B2
S2 B2
0.602136 0.504354 0.550000 0.811326 0.894925 94

B1 B1 S1
S2 B2
0.619410 0.951117 1.000000 0.921814 0.818878 95

B2 S1 B1
S2 B2
0.658773 0.866196 0.600000 0.974079 0.982641 96

B1 B1 CE
S1 B1
0.845169 0.713795 0.850000 0.825037 0.907136 97

B2 CE S2
S2 B2
0.763307 0.906688 0.800000 0.909603 0.916207 98

S2 S2 CE
S1 B2
0.873561 0.885305 0.700000 0.666710 0.562847 99

B2 B1 S2
B1 S1
0.627332 0.647658 0.700000 0.890881 0.577798 100

CE B2 S1
CE CE
0.512305 0.683243 0.950000 0.503043 0.790260 101

CE B1 B2
S2 B2
0.903709 0.746753 0.500000 0.939564 0.870031 102

CE CE B2
S2 B2
0.753322 0.890995 0.600000 0.685652 0.725512 103

CE S2 S1
S2 B3
0.946507 0.600000 1.000000 0.845069 0.808512 104

CE B1 B1
S2 B2
0.894700 0.702165 1.000000 0.766956 0.900539 105

CE S1 S1
S2 B2
0.844234 0.951519 0.850000 0.750510 0.513658 106

S1 S1 B2
S2 B2
0.912522 0.808292 0.750000 0.952327 0.512612 107

B2 CE B1
S2 B2
0.585410 0.917677 0.750000 0.966889 0.961331 108

CE S2 B2
S2 B3
0.814156 0.908252 0.700000 0.893820 0.695325 109

B2 CE B2
CE CE
0.741374 0.524458 0.600000 0.943384 0.707772 110

B2 B1 B2
B1 CE
0.670187 0.598470 0.500000 0.944142 0.570554 111

APPENDIX F

JOB SHOP SCHEDULING CASE STUDY

TRAINING AND TESTING DATA SAMPLE SETS

Job Shop Scheduling Case Study: ***Sample data sets***

Format:	*Input1*	*Input2*
	FIFO	*LIFO*

0.336 0.513
0.000000 1.000000

0.214 0.631
1.000000 0.000000

0.257 0.519
0.000000 1.000000

0.188 0.645
1.000000 0.000000

0.214 0.559
0.000000 1.000000

0.174 0.679
1.000000 0.000000

0.076 0.818
1.000000 0.000000

0.115 0.727
1.000000 0.000000

0.184 0.589
1.000000 0.000000

0.352 0.583
1.000000 0.000000

0.428 0.371
0.000000 1.000000

0.655 0.299
0.000000 1.000000

0.243 0.848
1.000000 0.000000

0.401 0.439
0.000000 1.000000

0.599 0.391
0.000000 1.000000

0.530 0.391
0.000000 1.000000

0.477 0.401
0.000000 1.000000

0.411 0.499
0.000000 1.000000

0.579 0.331
0.000000 1.000000

0.438 0.431
0.000000 1.000000

0.125 0.679
1.000000 0.000000

0.095 0.727
1.000000 0.000000

0.164 0.601
1.000000 0.000000

0.099 0.711
1.000000 0.000000

0.122 0.677
1.000000 0.000000

0.132 0.701
1.000000 0.000000

APPENDIX G

JOB SHOP SCHEDULING CASE STUDY

15 CONTROL FUZZY RULES

Job Shop Scheduling Case Study: *15 Control Fuzzy Rules*

Format:	Fuzzy Input:	Input1	Input2	
	Fuzzy Output:	FIFO	LIFO	
		Degree of Membership		*Rule #*

B2 B1
B1 S1
0.541999 0.878001 1.000000 0.000000 1

B1 CE
S1 B1
0.644001 0.634001 0.000000 1.000000 2

S1 B1
B1 S1
0.698000 0.677999 1.000000 0.000000 3

S1 CE
S1 B1
0.984000 0.922000 0.000000 1.000000 4

S2 B1
B1 S1
0.560000 0.557999 1.000000 0.000000 5

S2 CE
S1 B1
0.560000 0.658001 0.000000 1.000000 6

CE S1
S1 B1
0.904001 0.954000 0.000000 1.000000 7

S1 S1
S1 B1
0.534000 0.618000 0.000000 1.000000 8

B1 S1
S1 B1
0.654000 0.626000 0.000000 1.000000 9

S2 B2
B1 S1
0.396000 0.691999 1.000000 0.000000 10

B2 S2
S1 B1
0.637999 0.572000 0.000000 1.000000 11

B1 S2
S1 B1
0.542001 0.752000 0.000000 1.000000 12

CE CE
S1 B1
0.802000 0.802000 0.000000 1.000000 13

B2 S1
S1 B1
0.552001 0.746000 0.000000 1.000000 14

B2 CE
S1 B1
0.973999 0.574000 0.000000 1.000000 15

APPENDIX H

PROGRAM SOURCE CODE

ROBOT MOTION CONTROL
CASE STUDY

SOURCE CODE

FILE NAMES:

Fuzzificztion procedure:	*steps1_3.cpp*
Defuzzification procedure:	*steps4_5.cpp*
Final rms_error computation:	*rms_cal.cpp*

Luis A. Quiroga *ISE - OHIO UNIVERSITY* *1993*

This program applies the Wang-Mendel methodology in the control of a two degree of freedom robot arm. The program is divided in three sections. The first section accomplishes the fuzzification procedure proposed by Wang and Mendel. The second section accomplishes the defuzzification step. Finally, the last part of this program computes the rns error for this procedure.
This section will perform the fuzzification step. A file containing the initial set of control rules is first generated. Since there will be conflicting rules, the program checks the degee of importance of each rule and eliminates those with lower degrees. The final set of rules is saved under the file Rulebase.

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <conio.h>

const int SIZE = 3;        //Defines size for linguistic values. User can change linguistic values according to
                           //specific application.
const int MAX = 1000;      //Defines size for original data file size. User can change size of raw data file according
                           //to specific appliaction.

struct dbase               //Defines structure of original data file
        {
        float slope;               //Defines the trajectory of the robot arm
        float a_coord;             //Defines the original x position
        float b_coord;             //Defines the original y position
        float theta1;              //Defines angle between the horizontal and the 1st. link
        float theta2;              //Defines angle between 1st. and 2nd. linnks.
        float x_final;             //Defines the final x position
        float y_final;             //Defines the final y position
        float phi1;                //Defines final angle between horizontal and 1st. link
        float phi2;                //Defines final angle between 1st. and 2nd. links.
        }db;

void membership1(float, float *, char *);     //Fuzzifies x position values
void membership2(float, float *, char *);     //Fuzzifies y position values
void membership3(float, float *, char *);     //Fuzzifies slope values
void membership4(float, float *, char *);     //Fuzzifies 1st. angle values
void membership5(float, float *, char *);     //Fuzzifies 2nd. angle values

void compare(float, float, char *, char *);   //Checks for conflicting rules

int rule_dec[MAX];             //Array to flag rules that are not conflicting
```

```
        //degree1, degree2: variables for the two possible degrees of membership in the fuzzification step.
        //degree_num, num_value, num: variables that carry the highest degree of membership after fuzzification.
        //literal1, literal2: variables for the two possible linguistic values in the fuzzification step.
        //degree_lit, lit_value, lit: variables that carry the highest linguistic value after fuzzification.
        //dummy1, dummy2: temporal variables

float degree1, degree2,degree_num, num_value,num;
char  literal1[SIZE],literal2[SIZE],degree_lit[SIZE],lit_value[SIZE],lit[SIZE];
float dummy1, dummy2;

                        // ''''''''''''''''''''''''''''''''  MAIN PROGRAM  ''''''''''''''''''''''''''''''''

void main()
{

        //degree_xlit, degree_ylit, degree_mlit: variables containing linguistic values for initial x's, y's, m's.
        //degree_a1lit, degree_a2lit, degree_lit: variables containing linguistic values for the initial angles.
        //temp1, temp2, temp3, temp4, temp5: temporal variables for linguistic values.
        //degree_x, degree_y, degree_m:variables containing degrees of membership for initial x's, y's, m's.
        //degree_a1, degree_a2:variables containing degrees of membership for initial angles.
        //d_angle1, d_angle2:variables containing the difference between initial and final angle values (raw data.)

char  filename[20];
char  degree_xlit[SIZE],degree_ylit[SIZE],degree_mlit[SIZE],\
      degree_a1lit[SIZE], degree_a2lit[SIZE],degree_lit[SIZE],\
      temp1[SIZE],temp2[SIZE],temp3[SIZE],temp4[SIZE],temp5[SIZE];
float degree_x, degree_y, degree_m, degree_a1, degree_a2, d_angle1, d_angle2;
float degree_num,temp6,temp7,temp8,temp9,temp10,temp11,temp22;
int   i,counter=0;
int   record_no=1;

FILE *fp;
FILE *rbase;
FILE *step3;
FILE *rulebase;

struct rulebase                 //Defines structure rulebase.  Rulebase is the file that will contain the final set of
                                //control rules.
        {
        float degree_x;         //Defines degree of memb. for x position
        char  degree_xlit[SIZE]; //Defines linguistic value for x position
        float degree_y;         //Defines degree of memb. for y position
        char  degree_ylit[SIZE]; //Defines linguistic value for y position
        float degree_m;         //Defines degree of memb. for the slope
        char  degree_mlit[SIZE]; //Defines linguistic value for the slope
        float degree_a1;        //Defines degree of memb. for the 1st. angle
        char  degree_a1lit[SIZE]; //Defines linguistic value for the 1st angle
        float degree_a2;        //Defines degree of memb. for the 2nd. angle
        char  degree_a2lit[SIZE]; //Defines linguistic value for the 2nd. angle
        }rubase;
```

```
/*
cout << "Enter the name of the input file:" ;//open database for reading
cin >> filename;
*/
        //Opens file "Rules". This file contains original set of rules
if ((rbase=fopen("rules","w")) == NULL)
  {
    cout << "File rules can not be open (a). Check for problems!!";
    exit(0);
  }

        //Opens original data file "Database". Database contains raw data
if ((fp=fopen("database","r")) == NULL)
  {
        cout <<"(i:\) Database file can not open (r). Check for problems!! ";
        exit(1);
  }

cout<<"   The program is finding the degree of membership"<<"\n"; //screen output comment
cout<<"   for each input and output.....please wait.   "<<"\n\n\n";     //screen output comment

                //Starts reading first record of original raw data file. Operation terminates for EOF
                //Note: "\n" is used through out the program to provide means for LF/CR.
while(fscanf(fp,"%f\n %f %f %f %f\n %f %f %f %f\n",\
                &db.slope,&db.a_coord,&db.b_coord,&db.theta1,&db.theta2,\
                &db.x_final,&db.y_final,&db.phi1,&db.phi2) != EOF){

        d_angle1 = (db.theta1 - db.phi1);  //Finds diffrence between initial and final values: 1st. angle
        d_angle2 = (db.theta2 - db.phi2);  //Finds diffrence between initial and final values: 2st. angle

        //Membership calls for fuzzification procedure. One subroutine per input/output is advised for modularity
        membership1(db.a_coord,&degree_num,degree_lit);
                rubase.degree_x = degree_num;           //Membership value is assigned for x coordinate var.
                strcpy(rubase.degree_xlit,degree_lit);  //Fuzzy linguistic value is assigned for x coordinate var.

        membership2(db.b_coord, &degree_num, degree_lit);
                rubase.degree_y = degree_num;           //Membership value is assigned for y coordinate var.
                strcpy(rubase.degree_ylit,degree_lit);  //Fuzzy linguistic value is assigned for y coordinate var.

        membership3(db.slope, &degree_num,degree_lit);
                rubase.degree_m = degree_num;           //Membership value is assigned for slope var.
                strcpy(rubase.degree_mlit,degree_lit);  //Fuzzy linguistic value is assigned for slope var.

        membership4(d_angle1,&degree_num,degree_lit);
                rubase.degree_a1 = degree_num;          //Membership value is assigned for 1st angle var.
                strcpy(rubase.degree_a1lit,degree_lit); //Fuzzy linguistic value is assigned for 1st angle var.

        membership5(d_angle2,&degree_num,degree_lit);
                rubase.degree_a2 = degree_num;          //Membership value is assigned for 2nd angle var.
                strcpy(rubase.degree_a2lit,degree_lit); //Fuzzy linguistic value is assigned for 2nd angle var.

        counter++;                                      //keeps count of total data sets fuzzified
```

```
            //Creates record in Rules file.  Saves both degree of membership and linguistic values
        fprintf(rbase,"%s  %s  %s\n",rubase.degree_xlit,rubase.degree_ylit,\
            rubase.degree_mlit);
        fprintf(rbase,"%s  %s\n",rubase.degree_a1lit,rubase.degree_a2lit);
        fprintf(rbase,"%f  %f  %f  %f  %f\n\n",rubase.degree_x,rubase.degree_y,\
            rubase.degree_m,rubase.degree_a1,rubase.degree_a2);

    }

        fclose(rbase);          //closes original raw data file
     fclose(fp);                //closes created rules file

            //Checks if the number of generated rules exceeeds the number of original data sets.  Can be deleted
        if(counter > MAX){
            cout << "The number of data sets in the database > 1000!";
            exit(1);
          }
            //Opens the created "Rules" file and inspects rules for possible conflicts

/*-------------- Opennig rulebase file to check for conflicting rules-----------*/
cout<<"    Successful creation of rules.  The program is now"<<"\n";        //screen output comment
cout<<"    checking for any conflicting rules.....Rulebase"<<"\n";          //screen output comment
cout<<"    file will contain only the strongest data pairs!"<<"\n\n\n";     //screen output comment

            //Opens "Rules" file.  This is for read only purposes
        if ((step3=fopen("rules","r")) == NULL)
          {
           cout <<"Rules file can not open (r). Check for problems!! ";
           exit(1);
          }

            //Opens "Rulebase" file.  This file contains final set of rules
        if ((rulebase=fopen("rulebase","w")) == NULL)
            {
             cout << "File rulebase can not be open (a). Check for problems!!";
         exit(0);
            }

        fpos_t pos;
        pos = 0;
        int j;

            //Main loop that will read each rule of the "Rules" file.  It advances forward one rule at a time
        for(i=0;i< counter;i++)
          {
          fsetpos(step3,&pos);                                    //Sets pointer to initial record position
          fscanf(step3,"%s  %s  %s\n %s %s\n %f %f %f %f %f",\
                rubase.degree_xlit,rubase.degree_ylit,rubase.degree_mlit,\
                rubase.degree_a1lit,rubase.degree_a2lit,&rubase.degree_x,\
                &rubase.degree_y,&rubase.degree_m,&rubase.degree_a1,\
                &rubase.degree_a2);
          fgetpos(step3,&pos);                                    //Assigns record position for reference
```

```
 if(rule_dec[i] == 1)     //Checks if rule for flag 1. Flag 1 means rule has been checked, advance to next
        continue;
                //Assigns read values to temporal vars. for swapping purposes
    strcpy(temp1,rubase.degree_xlit);
    strcpy(temp2,rubase.degree_ylit);
    strcpy(temp3,rubase.degree_mlit);
    strcpy(temp4,rubase.degree_a1lit);
    strcpy(temp5,rubase.degree_a2lit);
    temp6 = rubase.degree_x;
    temp7 = rubase.degree_y;
    temp8 = rubase.degree_m;
    temp9 = rubase.degree_a1;
    temp10 = rubase.degree_a2;
                //Finds degree of importance for the rule itself
        temp11 = (rubase.degree_x)*(rubase.degree_y)*(rubase.degree_m)*\
        (rubase.degree_a1)*(rubase.degree_a2);

    for(j=i+1; j<counter;j++)                                   //Internal loop to read next rule
     {
     fscanf(step3,"%s  %s  %s\n %s %s\n %f %f %f %f %f",\       //Reads next rule
              rubase.degree_xlit,rubase.degree_ylit,rubase.degree_mlit,\
              rubase.degree_a1lit,rubase.degree_a2lit,&rubase.degree_x,\
              &rubase.degree_y,&rubase.degree_m,&rubase.degree_a1,\
              &rubase.degree_a2);
    if(rule_dec[j] ==1)
    continue;

         //Checks if the antecedent part of the rules is the same.  If yes it is a conflict
     if(!strcmp(temp1,rubase.degree_xlit) && !strcmp(temp2,rubase.degree_ylit)&&\
          !strcmp(temp3,rubase.degree_mlit))
       {
          rule_dec[j] =1;           //Since rule is compared, assign Flag 1
          temp22 = (rubase.degree_x)*(rubase.degree_y)*(rubase.degree_m)*\//Find degree for rule
                    (rubase.degree_a1)*(rubase.degree_a2);

          if(temp22 > temp11)                    //Check which rule has higher degree.  If second rule
          {                                      //has higher degree than first, swap values and keep
             strcpy(temp1,rubase.degree_xlit);   //rule with higher degree.  Note closing brackets for
             strcpy(temp2,rubase.degree_ylit);   //acting loops: the process continues until first rule is
             strcpy(temp3,rubase.degree_mlit);   //compared against all other rules in "Rules" file.
             strcpy(temp4,rubase.degree_a1lit);
             strcpy(temp5,rubase.degree_a2lit);
             temp6 = rubase.degree_x;
             temp7 = rubase.degree_y;
             temp8 = rubase.degree_m;
             temp9 = rubase.degree_a1;
             temp10 = rubase.degree_a2;
       }
     }
}
    rule_dec[i] =1;          //If current rule has highest value, assign Flag 1 and save in main "Rulebase"
```

```
                                    //Create record in main "Rulebase" file
            fprintf(rulebase,"%s  %s  %s\n",temp1,temp2,temp3);
            fprintf(rulebase,"%s  %s\n",temp4,temp5);
            fprintf(rulebase,"%f  %f  %f  %f  %f     %d\n\n",temp6,temp7,temp8,\
                    temp9,temp10,record_no);
        record_no++;                //Assign rule number to record
          }
         int count=0;
                for(i=0;i< counter;i++)
                        if(rule_dec[i] ==1)
                        count++;
 fclose(rulebase);          //closes "rulebase" file
 fclose(step3);             //closes "rules" file
 cout <<"The program is terminated. Final set of rules is in Rulebase file.";     //screen output comment
 getch();
}
                        // """""""""""""""  END OF MAIN PROGRAM  """""""""""""""  //

                //MEMBERSHIP 1 - 5 subroutines perform the same function, i.e. fuzzification.
                //Refer to MEMBERSHIP 1 for comments

                // MEMBERSHIP1 defines the degree of membership for the initial x coordinate
void membership1(float a_coord, float *return_num, char  *return_lit)
{
  int interval;
                //Actual brake down of fuzzy regions for x coordinate var.
  if (a_coord >= 0 && a_coord < 0.33333)
     interval = 0;
  if (a_coord >= 0.33333 && a_coord < 0.66666)
     interval = 1;
  if (a_coord >= 0.66666 && a_coord < 0.99999)
     interval = 2;
  if (a_coord >= 0.99999 && a_coord < 1.33333)
     interval = 3;
  if (a_coord >= 1.33333 && a_coord < 1.66666)
     interval = 4;
  if (a_coord >= 1.66666 && a_coord < 2.)
     interval = 5;
                //According to raw data, obtained value is processed in one of cases below
  switch(interval){
                //Cases 0 - 5 simply find the degree of membership according to the degree of membership function.
                //Note that it is here that the crisp value is converted into a both a fuzzy linguistic value and its
                //specific degree of membership.
            case 0:
                    num = 3.0*a_coord;
                    strcpy(lit,"S2");
                    break;
            case 1:
                    degree1 = -3.0*a_coord + 2.0;
                    strcpy(literal1,"S2");
                    degree2 = 3.0*a_coord - 0.999999999;
                    strcpy(literal2,"S1");
                    break;
```

```
            case 2:
                    degree1 = -3.0*a_coord + 3.0;
                    strcpy(literal1,"S1");
                    degree2 = 3.0*a_coord - 2.0;
                    strcpy(literal2,"CE");
                    break;
            case 3:
                    degree1 = -3.0*a_coord + 3.999999999;
                    strcpy(literal1,"CE");
                    degree2 = 3.0*a_coord - 3.0;
                    strcpy(literal2,"B1");
                    break;
            case 4:
                    degree1 = -3.0*a_coord + 5.000000001;
                    strcpy(literal1,"B1");
                    degree2 = 3.0*a_coord - 3.999999999;
                    strcpy(literal2,"B2");
          break;
            case 5:
                    num = -3.0*a_coord + 6.0;
                    strcpy(lit,"B2");
                    break;
           default:
                 break;
      }

         if(interval && interval !=5)

                 //Since data generates two possible fuzzy values, subroutine compare will find highest value
         {
                 compare(degree1,degree2,literal1,literal2); //Call for compare subroutine
                 num = num_value;                            //Save highest degree of membership
                 strcpy(lit,lit_value);                      //Save specific linguistic value
         }
    *return_num = num;                                       //Returns degree of membership value
    strcpy(return_lit,lit);                                  //Returns linguistic value
}


                 //MEMBERSHIP2 defines the degree of membership for the initial y coordinate
void membership2(float b_coord, float *return_num, char  *return_lit)
{
  int interval;

  if (b_coord >= 0 && b_coord < 0.33333)
        interval = 0;
  if (b_coord >= 0.33333 && b_coord < 0.66666)
        interval = 1;
  if (b_coord >= 0.66666 && b_coord < 0.99999)
        interval = 2;
  if (b_coord >= 0.99999 && b_coord < 1.33333)
     interval = 3;
  if (b_coord >= 1.33333 && b_coord < 1.66666)
```

```
   interval = 4;
if (b_coord >= 1.66666 && b_coord < 2.0)
       interval = 5;

switch(interval){
                //Cases 0 - 5 simply find the degree of membership according to the degree of membership function.
                //Note that it is here that the crisp value is converted into a both a fuzzy linguistic value and its
                //specific degree of membership.

       case 0:
            num = 3.0*b_coord;
            strcpy(lit,"S2");
            break;

       case 1:
            degree1 = -3.0*b_coord + 2.0;
            strcpy(literal1,"S2");
            degree2 = 3.0*b_coord - 0.999999999;
            strcpy(literal2,"S1");
            break;

       case 2:
            degree1 = -3.0*b_coord + 3.0;
            strcpy(literal1,"S1");
            degree2 = 3.0*b_coord - 2.0;
            strcpy(literal2,"CE");
            break;
       case 3:
            degree1 = -3.0*b_coord + 3.99999999;
            strcpy(literal1,"CE");
            degree2 = 3.0*b_coord - 3.0;
            strcpy(literal2,"B1");
            break;
       case 4:
            degree1 = -3.0*b_coord + 5.000000001;
            strcpy(literal1,"B1");
            degree2 = 3.0*b_coord - 3.999999999;
            strcpy(literal2,"B2");
            break;
       case 5:
            num = -3.0*b_coord + 6.0;
            strcpy(lit,"B2");
            break;
       default:
            break;
       }

       if(interval && interval !=5)
        {
                compare(degree1,degree2,literal1,literal2);
                num = num_value;
                strcpy(lit,lit_value);
        }
```

```
    *return_num = num;
    strcpy(return_lit,lit);
}


                    // MEMBERSHIP3 defines the degree of membership for the slope
void membership3(float slope, float *return_num, char  *return_lit)
{
  int interval;

  if (slope >= 0 && slope < 20)
        interval = 0;
  if (slope >= 20 && slope < 40)
        interval = 1;
  if (slope >= 40 && slope < 60)
        interval = 2;
  if (slope >= 60 && slope < 80)
        interval = 3;
  if (slope >= 80 && slope < 100)
        interval = 4;

  if (slope >= 100)
        interval = 5;

  switch(interval){
                    //Cases 0 - 5 simply find the degree of membership according to the degree of membership function.
                    //Note that it is here that the crisp value is converted into a both a fuzzy linguistic value and its
                    //specific degree of membership.

        case 0:
            num = 0.05*slope;
            strcpy(lit,"S2");
            break;
        case 1:
            degree1 = -0.05*slope + 2;
            strcpy(literal1,"S2");
            degree2 = 0.05*slope -1;
            strcpy(literal2,"S1");
            break;
        case 2:
            degree1 = -0.05*slope + 3;
            strcpy(literal1,"S1");
            degree2 = 0.05*slope -2;
            strcpy(literal2,"CE");
            break;
        case 3:
            degree1 = -0.05*slope + 4;
            strcpy(literal1,"CE");
            degree2 = 0.05*slope -3;
            strcpy(literal2,"B1");
            break;
```

```
        case 4:
            degree1 = -0.05*slope + 5;
            strcpy(literal1,"B1");
            degree2 = 0.05*slope -4;
            strcpy(literal2,"B2");
            break;
        case 5:
            num = 1;
            strcpy(lit,"B2");
            break;
        default:
            break;
    }

        if(interval && interval !=5)
        {
                compare(degree1,degree2,literal1,literal2);
                num = num_value;
                strcpy(lit,lit_value);
        }
    *return_num = num;
    strcpy(return_lit,lit);
}
```

*// **MEMBERSHIP4** defines the degree of membership for the first angle variation*

```
void membership4(float d_angle1, float *return_num, char  *return_lit)
{
  int interval;

  //cout << "\nmem4:deltangle = "<< d_angle1;
  if (d_angle1 >= 0.0 && d_angle1 < 1.714285)
    interval = 0;
  if (d_angle1 >= 1.714285 && d_angle1 < 3.428571)
    interval = 1;
  if (d_angle1 >= 3.428571 && d_angle1 < 5.142857)
    interval = 2;
  if (d_angle1 >= 5.142857 && d_angle1 < 6.857142)
    interval = 3;
  if (d_angle1 >= 6.857142 && d_angle1 < 8.571428)
    interval = 4;
  if (d_angle1 >= 8.571428 && d_angle1 < 10.285714)
    interval = 5;
  if (d_angle1 >= 10.285714 && d_angle1 <= 12.0)
    interval = 6;
  if (d_angle1 > 12.0)
    interval = 7;

  switch(interval){
```

//Cases 0 - 5 simply find the degree of membership according to the degree of membership function.
//Note that it is here that the crisp value is converted into a both a fuzzy linguistic value and its
//specific degree of membership.

```
case 0:
     num = 0.58333*d_angle1;
     strcpy(lit,"S3");
     break;
case 1:
    degree1 = -0.5833333*d_angle1 + 1.9999997;
    strcpy(literal1,"S3");
    degree2 = 0.5833333*d_angle1 - 0.9999995;
    strcpy(literal2,"S2");
    break;
case 2:
    degree1 = -0.5833333*d_angle1 + 2.9999999;
    strcpy(literal1,"S2");
    degree2 = 0.5833333*d_angle1 - 1.9999997;
    strcpy(literal2,"S1");
    break;
case 3:
    degree1= -0.5833333*d_angle1 + 3.9999995;
    strcpy(literal1,"S1");
    degree2= 0.5833333*d_angle1 - 2.9999999;
    strcpy(literal2,"CE");
    break;
case 4:
    degree1 = -0.5833333*d_angle1 + 4.9999996;
    strcpy(literal1,"CE");
    degree2 = 0.5833333*d_angle1 - 3.9999996;
    strcpy(literal2,"B1");
    break;
case 5:
    degree1 = -0.5833333*d_angle1 + 5.9999998;
    strcpy(literal1,"B1");
    degree2 = 0.5833333*d_angle1 - 4.9999995;
    strcpy(literal2,"B2");
    break;
case 6:
    degree1 = -0.5833333*d_angle1 + 7.0;
    strcpy(literal1,"B2");
    degree2 = 0.5833333*d_angle1 - 5.9999998;
    strcpy(literal2,"B3");
    break;
case 7:
    num = 1.0;
    strcpy(lit,"B3");
    break;
default:
    break;
}
if(interval && interval !=7)
 {
        compare(degree1,degree2,literal1,literal2);
        num = num_value;
        strcpy(lit,lit_value);
 }
```

```
    *return_num = num;
    strcpy(return_lit,lit);
}

                //MEMBERSHIP5 defines the degree of mebership for the second angle variation
void membership5(float d_angle2, float *return_num, char  *return_lit)
{

  int interval;

  if (d_angle2 >= -30.0 && d_angle2 < -25.714285)
    interval = 0;
  if (d_angle2 >= -25.714285 && d_angle2 < -21.428571)
    interval = 1;
  if (d_angle2 >= -21.428571 && d_angle2 < -17.142857)
    interval = 2;
  if (d_angle2 >= -17.142857 && d_angle2 < -12.857142)
    interval = 3;
  if (d_angle2 >= -12.857142 && d_angle2 < -8.571428)
    interval = 4;
  if (d_angle2 >= -8.571428 && d_angle2 < -4.285714)
    interval = 5;
  if (d_angle2 >= -4.285714 && d_angle2 <= 0.0)
    interval = 6;
  if (d_angle2 > 0.0)
    interval = 7;
  switch(interval){
                //Cases 0 - 5 simply find the degree of membership according to the degree of membership function.
                //Note that it is here that the crisp value is converted into a both a fuzzy linguistic value and its
                //specific degree of membership.

        case 0:
             num = 0.2333333*d_angle2 + 6.999999;
             strcpy(lit,"S3");
             break;
        case 1:
            degree1 = -0.2333333*d_angle2 - 4.9999998;
            strcpy(literal1,"S3");
            degree2 = 0.2333333*d_angle2 + 5.9999998;
            strcpy(literal2,"S2");
            break;
        case 2:
            degree1 = -0.2333333*d_angle2 - 3.9999999;
            strcpy(literal1,"S2");
            degree2 = 0.2333333*d_angle2 + 4.9999998;
            strcpy(literal2,"S1");
            break;
        case 3:
            degree1= -0.2333333*d_angle2 - 2.9999998;
            strcpy(literal1,"S1");
            degree2= 0.2333333*d_angle2 + 3.9999999;
            strcpy(literal2,"CE");
            break;
```

```
        case 4:
            degree1= -0.2333333*d_angle2 - 1.9999998;
            strcpy(literal1,"CE");
            degree2= 0.2333333*d_angle2 + 2.9999998;
            strcpy(literal2,"B1");
            break;
        case 5:
            degree1= -0.2333333*d_angle2 - 0.9999999;
            strcpy(literal1,"B1");
            degree2= 0.2333333*d_angle2 + 1.9999998;
            strcpy(literal2,"B2");
            break;
        case 6:
            degree1 = -0.2333333*d_angle2;
            strcpy(literal1,"B2");
            degree2 = 0.2333333*d_angle2 + 0.9999999;
            strcpy(literal2,"B3");
            break;
        case 7:
            num = 1;
            strcpy(lit,"B3");
            break;

        default:
            break;

        }


        if(interval && interval !=7)
         {
                compare(degree1,degree2,literal1,literal2);
                num = num_value;
          strcpy(lit,lit_value);
         }
    *return_num = num;
    strcpy(return_lit,lit);
}

                //COMPARE will determine which degree of membership is higher

void compare(float degree1,float degree2,char *literal1,char *literal2)
{
        num_value=0;
        num_value= (degree1 > degree2) ? degree1 : degree2;          //Checks which degree of membership is higher
        if(degree1 > degree2)                          //Swaps values.  Saves higher numerical & linguistic values
                strcpy(lit_value,literal1);
        else
                strcpy(lit_value,literal2);
}
```

END OF FUZZIFICATION PROCEDURE

In this part of the program, a new set of raw data sets is used to check the accuracy of the rules stored in the Rulebase file. The program basically defines the interval in which the raw data lies, determines what rules in the rulebase are fired in parallel, and it applies the centroid rule to accomplish the defuzzification step and obtain crisp values for control purposes.

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <conio.h>

const int SIZE = 3;              //Maximum size for linguistic variable. User can redifine according to
                                 //specific application.
const int MAX = 1000;            //Maximum size for raw data sets. User can redifine according to specific
                                 //application
const int BOX_SIZE= 7;           //Size of mapping matrix. User redifines according to number of fuzzy regions
const int BOX_SIZE_2= 7;         //Size of mapping matrix. User redifines according to number of fuzzy regions

                //Array defining all possible linguistic values. User defines array according to number of fuzzy
                //regions in the fuzzy design.
char degrees[BOX_SIZE][3] ={"S3","S2","S1","CE","B1","B2","B3"};
char degrees_2[BOX_SIZE_2][3] ={"S3","S2","S1","CE","B1","B2","B3"};

                //Array defining center of gravity values of ech fuzzy region. User defines array according to
                //number of fuzzy regions in the fuzzy design.
float centers[BOX_SIZE]={1.714285,3.428571,5.142857,6.857142,8.571428,10.285714,12.0};
float centers_2[BOX_SIZE]={-25.714285,-21.428571,-17.142857,-12.857142,-8.571428,\
                           -4.285714,0.0};
int rule_dec[MAX];

char  angle1_box_lit[BOX_SIZE][BOX_SIZE][BOX_SIZE][3];   //Contains linguistic value for 1st FAM
char  angle2_box_lit[BOX_SIZE][BOX_SIZE][BOX_SIZE][3];   //Contains linguistic value for 2nd FAM
float angle1_box_val[BOX_SIZE][BOX_SIZE][BOX_SIZE];      //Contains numerical value for 1st FAM
float angle2_box_val[BOX_SIZE][BOX_SIZE][BOX_SIZE];      //Contains numerical value for 2nd FAM

void    mapping();                                       //Maps values to create FAM's. This fuction is used
                                                         //to create the proper mapping of inputs into
                                                         //outputs and present the map in matrix form.
float   match(char *,int);                               //Matches the value of incoming raw data with the
                                                         //center values of each fuzzy region. This function
                                                         //allows to find the multiplying factor of each
                                                         //member of the centroid rule equation.
void    membership1(float , float *, float *, int *, int *);  //Defines specific fuzzy region for incoming x coord
void    membership2(float, float *, float *, int *, int *);   //Defines specific fuzzy region for incoming y coord
void    membership3 (float, float *, float *, int *, int *);  //Defines specific fuzzy region for incoming slope
void    testing();                                       //Defines control output for raw data. This function
                                                         //will perform the deffuzification step by means of
                                                         //the centroid rule.
```

```
                    // """""""""""""""""  MAIN PROGRAM  """""""""""""""""""""

void main()
{
    int counter,i,j,k;

        mapping();                  //Call of mapping subroutine
 cout<<"   The program is executing the mapping function. Please wait...\n\n\n\n\n";//screen output output

                //Displays 1st FAM in matrix form.  Inactive to speed-up run-time
    /*  for(k=0;k<BOX_SIZE;k++){
                        printf("Value of m is %3s \n",degrees[k]);
                for(i=0;i<BOX_SIZE;i++){
                        for(j=0;j<BOX_SIZE;j++)
                                printf("%3s ",angle1_box_lit[i][j][k]);
                                 cout << "\t";
                        for(j=0;j<BOX_SIZE;j++)
                                printf("%5.3f  ",angle1_box_val[i][j][k]);
                    cout << "\n";
                //getchar();
                        }
        }
                //Displays 2nd. FAM in matrix form.  Inactive to speed-up run-time
        for(k=0;k<BOX_SIZE_2;k++){
                        printf("Value of m is %3s \n",degrees_2[k]);
                for(i=0;i<BOX_SIZE_2;i++){
                        for(j=0;j<BOX_SIZE_2;j++)
                                printf("%3s ",angle2_box_lit[i][j][k]);
                                 cout << "\t";
                        for(j=0;j<BOX_SIZE_2;j++)
                                printf("%5.3f  ",angle2_box_val[i][j][k]);
                    cout << "\n";
                // getchar();
                        }
        }*/
cout<<"   Mapping was successful. The program is now testing new data...\n\n\n\n\n";//screen comment output
        testing();
cout<<"   Testing is now completed.  Check final results in Results file.";       /screen comment output
        getchar();
}

                    // """""""""""""""""""""""""""""""""  END OF MAIN PROGRAM  """""""""""""""""""""""""

                    // ******************        MAPPING        ****************
void mapping()
{
int counter,i,j,k,l;
FILE *fp;
```

```
        //degree1, degree2: variables for the two possible degrees of membership in the fuzzification step
        //degree_num, num_value, num: variables that carry the highest degree of membership after fuzzification
        //literal1, literal2: variables for the two possible linguistic values in the fuzzification step
        //degree_lit, lit_value, lit: variables that carry the highest linguistic value after fuzzification

float degree1, degree2, slope, num_value, num;
char  literal1[SIZE],literal2[SIZE],degree_lit[SIZE],lit_value[SIZE],lit[SIZE];

struct rulebase                 //Defines structure Rulebase. This is the file containing final set of fuzzy rules
        {
        float degree_x;         //Degree of membership of x coordinate
        char  degree_xlit[SIZE]; //Linguistic value for x coordinate
        float degree_y;         //Degree of membership for y coordinate
        char  degree_ylit[SIZE]; //Linguistic value for y coordinate
        float degree_m;         //Degree of membership for slope path
        char  degree_mlit[SIZE]; //Linguistic value for slope path
        float degree_a1;        //Degree of membership for 1st. angle
        char  degree_a1lit[SIZE]; //Linguistic value for 1st. angle
        float degree_a2;        //Degree of membership for 2nd. angle
        char  degree_a2lit[SIZE]; //Linguistic value for 2nd angle
        }rubase;

                //Opens file Rulebase for read only.
if ((fp=fopen("rulebase","r")) == NULL)
  {
        cout <<"Rulebase file can not open (r). Check for problems!! ";
        exit(1);
  }
        //The following two loops organize the data in the form of a FAM bank. Inputs will be mapped into outputs
        //and presented in matrix form. There is always one FAM bank per output. FAMs are initialized to "0" for
        //the degree of membership, and to "xx" for linguistic values

        for(i=0;i<BOX_SIZE;i++)
                for(j=0;j<BOX_SIZE;j++)
                        for(k=0;k<BOX_SIZE;k++){
                            angle1_box_val[i][j][k]=0.;
                            strcpy(angle1_box_lit[i][j][k],"xx");
                          }

        for(i=0;i<BOX_SIZE_2;i++)
                for(j=0;j<BOX_SIZE_2;j++)
                        for(k=0;k<BOX_SIZE_2;k++){
                            angle2_box_val[i][j][k]=0.;
                            strcpy(angle2_box_lit[i][j][k],"xx");
                          }

                //Reads first rule from "Rulebase" file
while(fscanf(fp,"%s%s%s\n%s%s\n%f%f%f%f%f%d\n\n",
                rubase.degree_xlit,rubase.degree_ylit,rubase.degree_mlit,\
                rubase.degree_a1lit,rubase.degree_a2lit,&rubase.degree_x,\
                &rubase.degree_y,&rubase.degree_m,&rubase.degree_a1,\
                &rubase.degree_a2,&counter) != EOF){
```

```
                //The following two loops perform  mapping between the input and output parts of the current
                //rule.  Once the specific matrix position is determined, the corresponding degree of membership
                //and linguistic value are placed in the matrix.

        for(l=0;l<BOX_SIZE;l++){
                if(!strcmp(rubase.degree_xlit,degrees[l]))
                i=l;
                if(!strcmp(rubase.degree_ylit,degrees[l]))
                j=l;
                if(!strcmp(rubase.degree_mlit,degrees[l]))
                k=l;
        }
         if(!angle1_box_val[i][j][k]){
                 strcpy(angle1_box_lit[i][j][k],rubase.degree_a1lit);
                        angle1_box_val[i][j][k]=rubase.degree_a1;
        }

        for(l=0;l<BOX_SIZE_2;l++){
                if(!strcmp(rubase.degree_xlit,degrees_2[l]))
                i=l;
                if(!strcmp(rubase.degree_ylit,degrees_2[l]))
                j=l;
                if(!strcmp(rubase.degree_mlit,degrees_2[l]))
                k=l;
        }
         if(!angle2_box_val[i][j][k]){
                 strcpy(angle2_box_lit[i][j][k],rubase.degree_a2lit);
                        angle2_box_val[i][j][k]=rubase.degree_a2;
         }
   }

 fclose(fp);
}


                /*************** TESTING ***************/

void testing()
{
        //i, k, j: arrays to save the location of a specific value in the MAP matrix
        //count1, count2, count3: variables to carry location of specific value in the MAP matrix
        //degree_x, degree_y, degree_m:variables containing degrees of membership for initial x's, y's, m's.
        //degree_s, degree_a, degree_b:  arrays receiving the two possible linguistic values after fuzzification
        //centroid1, centroid2: variables that containig values for required angle adjustments: control output
        //map1, map1_2,:variables containing center values of fired fuzzy regions
        //denominator: variable containing denominator value for centroid rule
        //numerator, numerator2: variables containing numerator values for centroid1 and centroid2
        //map_lit, map_lit_2:variables containing linguistic values of fired fuzzy region

char filename[20];
int i[2],k[2],j[2];
int count1,count2,count3;
float degree_x, degree_y, degree_m, d_angle1, d_angle2;
```

```
float degree_s[2], degree_b[2], degree_a[2];
float centroid1,centroid2, map1,map1_2;
float denominator=0.,numerator=0.,numerator2=0.;
char map_lit[3],map_lit_2[3];
FILE *fp;
FILE *deffuzzi;
FILE *answer;

struct testbase          //Defines structure Testbase. This file contains test data and it is similar to Database
        {
        float slope              //Defines the trajectory of the robot arm
        float a_coord;           //Defines the original x position
        float b_coord;           //Defines the original y position
        float theta1;            //Defines angle between the horizontal and the 1st. link
        float theta2;            //Defines angle between 1st. and 2nd. linnks.
        float x_final;           //Defines the final x position
        float y_final;           //Defines the final y position
        float phi1;              //Defines final angle between horizontal and 1st. link
        float phi2;              //Defines final angle between 1st. and 2nd. links.
        }db;

struct answer                    //Defines structure Answer. This file contains control output results
        {
        float a_coord;           //Defines the original x position
        float b_coord;           //Defines the original y position
        float theta1;            //Defines angle between the horizontal and the 1st. link
        float theta2;            //Defines angle between 1st. and 2nd. linnks.
        float x_final;           //Defines the final x position
        float y_final;           //Defines the final y position
        float centroid1;         //Defines first centroid output.  Control output for 1st angle increment
        float centroid2;         //Defines first centroid output.  Control output for 2nd angle increment
        }anbase;

                //Opens "Dbase" file containing testing data.  Read only.
strcpy(filename,"dbase");
if ((fp=fopen(filename,"r")) == NULL)
  {
    cout << "Test file can not be open (r). Check for problems!!";
    exit(0);
  }

                //Opens "Results" file to record control output.  Write only.
if ((deffuzzi=fopen("results","w")) == NULL)
  {
    cout << "File Results can not open to write. Check for problems!!";
    exit(0);
  }

                //Reads records from testing data file.  Loops ends for EOF.
        while(fscanf(fp,"%f\n %f %f %f %f\n %f %f %f %f\n",\
                &db.slope,&db.a_coord,&db.b_coord,&db.theta1,&db.theta2,\
                &db.x_final,&db.y_final,&db.phi1,&db.phi2) != EOF){
```

```
        //These subroutines determine the specific fuzzy regions where test data lie
    membership1(db.a_coord,&degree_a[0],&degree_a[1], &i[0],&i[1]);
    membership2(db.b_coord,&degree_b[0],&degree_b[1], &j[0],&j[1]);
    membership3(db.slope,&degree_s[0],&degree_s[1], &k[0],&k[1]);

    int dummy1,dummy2,dummy3;
    float dummy_a,dummy_b,dummy_s;

        //This blocks determines the number of rules that have been activated in parallel. In this case
        //notice that there will be 8 rules fired at the same time. It is necessary to collect all those rules,
        //determine the degrees of membership of each input, multiply them by the central value of the
        //fuzzy region, and add them up to find the numerator of the centroid.
    for(count1=0;count1 < 2;count1++){
        dummy1=i[count1];
        dummy_a=degree_a[count1];
        for(count2=0;count2 < 2;count2++){
            dummy2=j[count2];
            dummy_b=degree_b[count2];
            for(count3=0;count3 < 2;count3++){
                dummy3=k[count3];
                dummy_s=degree_s[count3];
                    //Assigns linguistic value for input data
                strcpy(map_lit,angle1_box_lit[dummy1][dummy2][dummy3]);
                strcpy(map_lit_2,angle2_box_lit[dummy1][dummy2][dummy3]);
                    //Calls match subroutine to define central values for fuzzy regions
                map1 = match(map_lit,1);
                map1_2 = match(map_lit_2,2);
                    //Finds numerator and denominator values for both control variables
                denominator+= dummy_a*dummy_b*dummy_s;
                numerator+=dummy_a*dummy_b*dummy_s*map1;
                numerator2+=dummy_a*dummy_b*dummy_s*map1_2;
            }
        }
    }
        if(denominator){
        centroid1 =numerator/denominator;       //Determines first centroid control value
        centroid2 =numerator2/denominator;      //Defines second centroid control value
        }
    else
        cout <<"wild data, theta1\n";


        //Recors required data into "Results" file
    fprintf(deffuzzi,"%f %f  %f  %f\n",db.a_coord,db.b_coord,\
          db.x_final,db.y_final);
    fprintf(deffuzzi,"%f %f  %f  %f\n\n",db.theta1,db.theta2,centroid1,centroid2);
  }
   fclose(deffuzzi);
    fclose(fp);
}


        //MEMBERSHIP1 defines the fuzzy regions which are activated by initial x coordinate
```

```
void membership1(float a_coord, float *return_num1,float *return_num2, int *v1,int *v2)
{
   int interval;
   int int_v1, int_v2;      //variables containing the fuzzy interval which is fired by incoming data.
   float degree1, degree2;

  if (a_coord >= 0 && a_coord < 0.33333)
     interval = 0;
  if (a_coord >= 0.33333 && a_coord < 0.66666)
     interval = 1;
  if (a_coord >= 0.66666 && a_coord < 0.99999)
     interval = 2;
  if (a_coord >= 0.99999 && a_coord < 1.33333)
     interval = 3;
  if (a_coord >= 1.33333 && a_coord < 1.66666)
     interval = 4;
  if (a_coord >= 1.66666 && a_coord < 2.0)
     interval = 5;

  switch(interval){
                  //Cases 0-5 find the degree of membership according to the degree of membership function.
                  //Note that in this case we would like to know the fuzzy region in which the incoming data fires
                  //the fuzzy system.. The fuzzy intervals are saved under int_v1 and int_v2.
         case 0:
                    degree1 = 3.0*a_coord;
                    degree2 = degree1;
                    int_v1=1;
                    int_v2=1;
                    break;
         case 1:
                    degree1 = -3.0*a_coord + 2.0;
                    degree2 = 3.0*a_coord - 0.9999999;
                    int_v1=1;
                    int_v2=2;
                    break;
         case 2:
                    degree1 = -3.0*a_coord + 3.0;
                    degree2 = 3.0*a_coord - 2.0;
                    int_v1=2;
                    int_v2=3;
                    break;
         case 3:
                    degree1 = -3.0*a_coord + 3.999999;
                    degree2 = 3.0*a_coord - 3.0;
                    int_v1=3;
                    int_v2=4;
                    break;



         case 4:
```

```
                degree1 = -3.0*a_coord +  5.0000001;
                degree2 = 3.0*a_coord - 3.999999;
                int_v1=4;
                int_v2=5;
                break;
        case 5:
                degree2 = -3.0*a_coord + 6.0;
                degree1 =degree2;
                int_v1=5;
                int_v2=5;
                break;
        default:
                break;
     }

    *v1 = --int_v1;
    *v2 = --int_v2;
    float temp;int tempint;
        if(degree2 > degree1){
                temp = degree1;
                degree1 = degree2;
                degree2 = temp;
                tempint=  *v1;
                *v1 = *v2;
                *v2 =tempint;

     }
    *return_num1 = degree1;
    *return_num2 = degree2;
}
```

*//**MEMBERSHIP2** defines the fuzzy regions which are activated by initial y coordinate*

```
void membership2(float b_coord, float *return_num1,float *return_num2, int *v1,int *v2)
{
  int interval;
  int int_v1, int_v2;
  float degree1, degree2;

  if (b_coord >= 0 && b_coord < 0.33333)
     interval = 0;
  if (b_coord >= 0.33333 && b_coord < 0.66666)
     interval = 1;
  if (b_coord >= 0.66666 && b_coord < 0.99999)
     interval = 2;
  if (b_coord >= 0.99999 && b_coord < 1.33333)
     interval = 3;
  if (b_coord >= 1.33333 && b_coord < 1.66666)
     interval = 4;
  if (b_coord >= 1.66666 && b_coord < 2.0)
     interval = 5;
  switch(interval){
```

```
        //Cases 0-5 find the degree of membership according to the degree of membership function.
        //Note that in this case we would like to know the fuzzy region in which the incoming data fires
        //the fuzzy system.. The fuzzy intervals are saved under int_v1 and int_v2.
    case 0:
            degree1 = 3.0*b_coord;
            degree2 = degree1;
            int_v1=1;
            int_v2=1;
            break;
    case 1:
            degree1 = -3.0*b_coord + 2.0;
            degree2 = 3.0*b_coord - 0.9999999;
            int_v1=1;
            int_v2=2;
            break;
    case 2:
            degree1 = -3.0*b_coord + 3.0;
            degree2 = 3.0*b_coord - 2.0;
            int_v1=2;
            int_v2=3;
            break;
    case 3:
            degree1 = -3.0*b_coord + 3.999999;
            degree2 = 3.0*b_coord - 3.0;
            int_v1=3;
            int_v2=4;
            break;
    case 4:
            degree1 = -3.0*b_coord + 5.0000001;
            degree2 = 3.0*b_coord - 3.999999;
            int_v1=4;
            int_v2=5;
            break;
    case 5:
            degree2 = -3.0*b_coord + 6.0;
            degree1 =degree2;
            int_v1=5;
            int_v2=5;
            break;
    default:
            break;
  }
*v1 = --int_v1;
*v2 = --int_v2;
      float temp;int tempint;
    if(degree2 > degree1){
            temp = degree1;
            degree1 = degree2;
            degree2 = temp;
            tempint=  *v1;
            *v1 = *v2;
            *v2 =tempint;
    }
```

```
    *return_num1 = degree1;
    *return_num2 = degree2;
}

                //MEMBERSHIP3 defines the fuzzy regions which are activated by initial slope path
void membership3(float slope, float *return_num1,float *return_num2, int *v1,int *v2)
{
  int interval;
  int int_v1, int_v2;
  float degree1, degree2;

  if (slope >= 0 && slope < 20)
        interval = 0;
  if (slope >= 20 && slope < 40)
        interval = 1;
  if (slope >= 40 && slope < 60)
        interval = 2;
  if (slope >= 60 && slope < 80)
        interval = 3;
  if (slope >= 80 && slope < 100)
        interval = 4;
  if (slope >= 100)
        interval = 5;
  switch(interval){
                //Cases 0-5 find the degree of membership according to the degree of membership function.
                //Note that in this case we would like to know the fuzzy region in which the incoming data fires
                //the fuzzy system.. The fuzzy intervals are saved under int_v1 and int_v2.
        case 0:
                degree1 = 0.05*slope;
                degree2 = degree1;
                int_v1=1;
                int_v2=1;
                break;
        case 1:
                degree1 = -0.05*slope + 2;
                degree2 = 0.05*slope -1;
                int_v1=1;
                int_v2=2;
                break;
        case 2:
                degree1 = -0.05*slope + 3;
                degree2 = 0.05*slope -2;
                int_v1=2;
                int_v2=3;
                break;
        case 3:
                degree1 = -0.05*slope + 4;
                degree2 = 0.05*slope -3;
                int_v1=3;
                int_v2=4;
                break;
        case 4:
```

```
                degree1 = -0.05*slope + 5;
                degree2 = 0.05*slope -4;
                int_v1=4;
                int_v2=5;
                break;
        case 5:
                degree2 = 1;
                degree1 =degree2;
                int_v1=5;
                int_v2=5;
                break;
        default:
                break;
        }
        float temp;int tempint;
        if(degree2 > degree1){
                temp = degree1;
                degree1 = degree2;
                degree2 = temp;
                tempint=  *v1;
                *v1 = *v2;
                *v2 =tempint;
        }
    *v1 = --int_v1;
    *v2 = --int_v2;
    *return_num1 = degree1;
    *return_num2 = degree2;
}
                /****************** MATCH ****************/
        //This subroutine the values corresponding to the fuzzy regions activated by the incoming data. The
        //function will determine the center value of the fuzzy region as specified in the array degrees[i]
loat match(char *string,int theta_index)
{

float value=0;
int i,upper_limit;
int result;

        if(theta_index==1)
                upper_limit=BOX_SIZE;
        if(theta_index==2)
                upper_limit=BOX_SIZE_2;

        for(i=0;i<upper_limit;i++){
                if(theta_index==1){
                if(!strcmp(string,degrees[i])){
                  value = centers[i];
                  break;
                }
             }


                if(theta_index==2){
```

```
                if(!strcmp(string,degrees_2[i])){
                  value = centers_2[i];
                  break;
                }
            }
        }
        return value;

}
```

END OF DEFUZZIFICATION PROCEDURE

This final part of the program will convert the obtained angle adjustments into final angle configurations. Upon obtaining the final angle configuration, the final (x,y) positions are computed and differences are found between calculated and original (x,y) positions.

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <conio.h>
#include <math.h>
FILE *deffuzzi;
FILE *finished;
void main()
{
        //final_angle1, final_angle2:variables containing final decimal values for 1st and 2nd angle
        //end_x, end_y:variables containing final values for x and y coordinates
        //diff_x, diff_y:variables containing the diffrence between calculated and original final (x,y) coordinates
        //error: variable containing the squared differential between calculated and original coordinate values
        //link1, link2: variables defining the robot arm lenght
 float final_angle1,final_angle2,end_x,end_y,diff_x,diff_y,error;
 float  link1=1.,link2=1.;
struct answer     //Definition of structure Answer. These file contains the generated control output from
                  //deffuzification step.
        {
        float a_coord;
        float b_coord;
        float theta1;
        float theta2;
        float x_final;
        float y_final;
        float centroid1;
        float centroid2;
        }anbase;
struct final      //Definition of structure Final. Final contains the final differences between the original and
                  //calculated (x, y) coordinates, the calculated (x,y) coordinates, and the final rms error
        {
        float x_final;
        float y_final;
        float x_end;
        float y_end;
        float diff_x;
        float diff_y;
        float error;
        }end;
        //Opens fiel Results. This file contains results from deffuzification step. Read only.
 if ((deffuzzi=fopen("results","r")) == NULL)
  {
    cout << "File Results can not open to read. Check for problems!!";
    exit(0);
  }
```

```
        //Opens file Final. This file contains final x, y coordinates. Write only.
if ((finished=fopen("final","w")) == NULL)
  {
    cout << "File Final can not open to write. Check for problems!!";
    exit(0);
  }
                //angle_rad1, angle_rad2: variables containing angle values in radian units
                //min: variable containing the minimum difference between calculated and original (x,y)'s
                //max: variable containing the maximum difference between calculated and original (x,y)'s
                //average:variable containing rms error value
                //sum: variable containing cummulative summ of squared differentials
                //counter:counter for the number of sets tested
        int i1,i2,j1,j2,k1,k2;
        float angle_rad1,angle_rad2;
        const float PI=3.141592654;
        float min,max,average,sum=0.;
        int counter=0;

        //Starts reading first record of original raw data file. Operation terminates for EOF
while(fscanf(deffuzzi,"%f %f  %f %f\n%f %f  %f %f\n\n",\
                &anbase.a_coord,&anbase.b_coord,&anbase.x_final,&anbase.y_final,\
                &anbase.theta1,&anbase.theta2,&anbase.centroid1,&anbase.centroid2)\
                != EOF){

        //Finds the angle adjustment. Computes angle configuration in radian units.
        final_angle1 = anbase.theta1 - anbase.centroid1;
        final_angle2 = anbase.theta2 - anbase.centroid2;
        angle_rad1 = PI*final_angle1/180.;
        angle_rad2 = PI*final_angle2/180.;
        //Determines the final (x,y) coordinate according to fuzzy angle configuration
  end_x = link1*cos(angle_rad1)+link2*cos(angle_rad1)*cos(angle_rad2)-\
                link2*sin(angle_rad1)*sin(angle_rad2);
  end_y = link1*sin(angle_rad1)+link2*sin(angle_rad1)*cos(angle_rad2)+\
                link2*cos(angle_rad1)*sin(angle_rad2);

        //Finds the difference between calculated and original (x,y) coordinates
  error =(((anbase.x_final-end_x)*(anbase.x_final-end_x))+\
          ((anbase.y_final-end_y)*(anbase.y_final-end_y)));

        //Finds the difference by horizontal and vertical positions
  diff_x = anbase.x_final-end_x;
  diff_y = anbase.y_final-end_y;

  /*
  printf("\n    x_initial= %f        y_initial= %f\n",anbase.a_coord,anbase.b_coord);
  printf("theta1_initial=%f   theta2_initial=%f\n",anbase.theta1,anbase.theta2);
  printf("*****************************************************\n");
  printf("   centroid1=%f         centroid2=%f\n",anbase.centroid1,anbase.centroid2);
  printf("final_angle1=%f      final_angle2=%f\n\n",final_angle1,final_angle2);
  printf("****************** Comparison **********************\n");

  printf("    x_final= %f          y_final = %f\n",anbase.x_final,anbase.y_final);
  printf("end_x_fuzzy = %f \    end_y_fuzzy = %f\n",end_x,end_y);
```

```
  printf("---------------------------------------------------\n");
  printf("differences x =%f \t    y=%f\n",anbase.x_final-end_x,anbase.y_final-end_y);
  cout<<"error is "<<error;
  printf("\n---------------------------------------------------\n");
  getchar();*/
    if (!counter){
            min=error;
            max=error;
      }
    else
          {
            if(error > max)max=error;
            if(error < min)min=error;
          }
   sum+=error;
   counter++;

   fprintf(finished,"%f  %f   %f  %f  %f  %f  %f \n",anbase.x_final,\
                        anbase.y_final,end_x,end_y,diff_x,diff_y,error);

/*
  printf("\nx error is %f \%\n",fabs(100.*(anbase.x_final-end_x)/anbase.x_final));
  printf("y error is %f \%\n",(float)fabs(100.*(anbase.y_final-end_y)/anbase.y_final));
  getchar(); */
    }

  average = sqrt(sum/(counter*2));            //Finds rms error
  cout<<"#######################################"<<"\n\n";
  cout<<"        The number of data sets tested is: "<<counter<<"\n\n";
  cout<<"            The total error summation is: "<<sum<<"\n\n";
  //cout<<"                     Minimun error is: "<<min<<"\n";
  //cout<<"                     Maximun error is: "<<max<<"\n\n";
  cout<<"                   Total rms error is: "<<average;
}
```

END OF PROGRAM

JOB-SHOP SCHEDULING
CASE STUDY

SOURCE CODE

FILE NAMES:

Fuzzificztion procedure:	*steps13j.cpp*
Defuzzification procedure:	*steps45j.cpp*

Luis A. Quiroga *ISE - OHIO UNIVERSITY* *1993*

This first part of the program will perform the initial generation of fuzzy control rules for the job-shop scheduling case study. Raw data is first fed to the system in order to perform the initial fuzzification procedure. Upon fuzzification, the generated rules are saved to later be tested with new data sets. The program is divided into MAIN and MEMBERSHIP modules. Each input and output has an independent membersship subroutine to accomplish fuzzification. Like for the robot arm control problem, conflicting rules are checked for and eliminated from the rulebase. The final set of control rules is saved in the file Rbasejs

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <conio.h>

const int SIZE = 3;             //Defines size for linguistic values. User can changhe linguistic values
                                //according to specific application
const int MAX = 360;            //Defines size for original data file size. User can change size of raw data file
                                //according to specific application
struct dbase                    //Defines structure Dbase, the original data file.
        {
        float input1;           //Defines the first coefficient of linear regression describing the mean processing
                                //time distribution
        float input2;           //Defines the second coefficient of linear regression describing the mean
                                //processing time distribution
        float output1;          //Defines first output decision. Output 1 represents a LIFO priority
        float output2;          //Defines second output decision. Output 2 represents a FIFO priority
        }db;

void membership1(float, float *, char *);   //Fuzzifies first coefficient values of the linear regression distribution
void membership2(float, float *, char *);   //Fuzzifies second coefficient values of the linear regression distribution
void membership4(float, float *, char *);   //Fuzzifies first output values of the linear regression distribution
void membership5(float, float *, char *);   //Fuzzifiessecond coefficient values of the linear regression distribution
void compare(float, float, char *, char *); //Checks for conflicting rules

int rule_dec[MAX];              //Array to flag rules that are not conflicting

        //degree1, degree2: variables for the two possible degrees of membership in the fuzzification step.
        //degree_num, num_value, num: variables that carry the highest degree of membership after fuzzification.
        //literal1, literal2: variables for the two possible linguistic values in the fuzzification step.
        //degree_lit, lit_value, lit: variables that carry the highest linguistic value after fuzzification.
float degree1, degree2, degree_num,num_value,num;
char  literal1[SIZE],literal2[SIZE],degree_lit[SIZE],lit_value[SIZE],lit[SIZE];
float dummy1, dummy2;
```

```
// """""""""""""""""  MAIN PROGRAM  """""""""""""""""

void main()
{

        //degree_xlit, degree_ylit: variables containing linguistic values for initial inputs of the system
        //degree_a1lit, degree_a2lit:variables containing linguistic values for outputs of the system
        //degree_num: variable containing highest degree of membership after fuzzification
        //degree_lit: variable containing final linguistic value after fuzzification
        //degree_x, degree_y:variables containing degree of membership for the inputs
        //degree_a1, degree_a2:variables containing degrees of membership for the outputs
        //temp1-5:temporal variables for linguistic values
        //temp6-22:temporal variables for degree of membership
char  filename[20];
char  degree_xlit[SIZE],degree_ylit[SIZE],\
     degree_a1lit[SIZE], degree_a2lit[SIZE],degree_lit[SIZE],\
     temp1[SIZE],temp2[SIZE],temp3[SIZE],temp4[SIZE],temp5[SIZE];
float degree_x, degree_y, degree_a1, degree_a2, degree_num;
float temp6,temp7,temp8,temp9,temp10,temp11,temp22;
int  i,counter=0;
int  record_no=1;

FILE *fp;
FILE *rbase;
FILE *step3;
FILE *rulebase;

struct rbasejs          //Defines structure Rbasejs. This file contains the final set of fuzzy rules for control
        {
        float degree_x;             //Contains the degree of membership for the first coefficient
        char  degree_xlit[SIZE];    //Contains linguistic value for first coefficient
        float degree_y;             //Contains the degree of membership for second coefficient
        char  degree_ylit[SIZE];    //Contains linguistic value for second coefficient
        float degree_a1;            //Contains the degree of membership for the first output
        char  degree_a1lit[SIZE];   //Contains linguistic value for first output
        float degree_a2;            //Contains the degree of membership for the second output
        char  degree_a2lit[SIZE];   //Contains linguistic value for second output
        }rubase;

        //Opens file "Rulesjs". This file contains the initial set of rules. Write only.
if ((rbase=fopen("rulesjs","w")) == NULL)
  {
    cout << "File rules can not be open (a). Check for problems!!";
    exit(0);
  }

        //Opens file "n32train". This file contains raw data. Read only.
if ((fp=fopen("n32train","r")) == NULL)
  {
        cout <<"(i:\) n32train file can not open (r). Check for problems!! ";
        exit(1);
  }
```

```
                    //Starts reading first record of original data file.  Operation terminates for EOF
while(fscanf(fp,"%f %f\n %f %f\n",\
                    &db.input1,&db.input2,&db.output1,&db.output2) != EOF){
/*
printf("\nRECORD %d \nslope = %f ",counter,db.slope);
printf("input1 = %f ",db.input1);
printf("input2 = %f ",db.input2);
cout << "\n";
*/
                    //Membership calls for fuzzification procedure.  There is one subroutine for each input/output
          membership1(db.input1,&degree_num,degree_lit);
                    rubase.degree_x = degree_num;
                    strcpy(rubase.degree_xlit,degree_lit);

          membership2(db.input2, &degree_num, degree_lit);
                    rubase.degree_y = degree_num;
                    strcpy(rubase.degree_ylit,degree_lit);

          membership4(db.output1,&degree_num,degree_lit);
                    rubase.degree_a1 = degree_num;
                    strcpy(rubase.degree_a1lit,degree_lit);

          membership5(db.output2,&degree_num,degree_lit);
                    rubase.degree_a2 = degree_num;
                    strcpy(rubase.degree_a2lit,degree_lit);
          counter++;

                              /************ debugging break ***********/
          //if(counter ==50)
          //        break;

          //cout<<"lit: x, y, m, a1, a2 are:  "<< rubase.degree_xlit<<"  "<<\
          //        rubase.degree_ylit<<"  "<<rubase.degree_mlit<<"  "<<rubase.degree_a1lit<<\
          //        "  "<<rubase.degree_a2lit<<"\n";

          //cout<<"mem x, y, m, a1, a2 are:  "<< rubase.degree_x<<"  "<<\
          //        rubase.degree_y<<"  "<<rubase.degree_m<<"  "<<rubase.degree_a1<<\
          //        "  "<<rubase.degree_a2;
          //getchar();

                    //Creates record in Rulesjs file.  Saves both the degree of membership and linguistic values
          fprintf(rbase,"%s  %s\n",rubase.degree_xlit,rubase.degree_ylit);
          fprintf(rbase,"%s  %s\n",rubase.degree_a1lit,rubase.degree_a2lit);
          fprintf(rbase,"%f  %f  %f  %f\n\n",rubase.degree_x,rubase.degree_y,\
                    rubase.degree_a1,rubase.degree_a2);
    }
          fclose(rbase);
     fclose(fp);
```

```
        //cout << "# of records is "<< counter;
        if(counter > MAX){
                cout << "The number of data sets in the database > 360!";
                exit(1);
         }

                //Opens created Rulesjs file and inspects rules for possible conflicts
/*---------- Opennig rulebase file to check for conflicting rules ---------*/

        if ((step3=fopen("rulesjs","r")) == NULL)
          {
           cout <<"Rules file can not open (r). Check for problems!! ";
           exit(1);
          }
        if ((rulebase=fopen("rbasejs","w")) == NULL)
           {
            cout << "File rulebase can not be open (a). Check for problems!!";
         exit(0);
           }

        fpos_t pos;
        pos = 0;
        int j;

        for(i=0;i< counter;i++)
          {

          fsetpos(step3,&pos);                    //Sets pointer to initial record position
          fscanf(step3,"%s  %s\n %s %s\n %f %f %f %f %f",\
                    rubase.degree_xlit,rubase.degree_ylit,\
                    rubase.degree_a1lit,rubase.degree_a2lit,&rubase.degree_x,\
                    &rubase.degree_y,&rubase.degree_a1,&rubase.degree_a2);
          fgetpos(step3,&pos);                    //Assigns record position for reference
          if(rule_dec[i] == 1)
                continue;
                        //Assigns read values to temporal variables for swapping purposes
          strcpy(temp1,rubase.degree_xlit);
          strcpy(temp2,rubase.degree_ylit);
          strcpy(temp4,rubase.degree_a1lit);
          strcpy(temp5,rubase.degree_a2lit);
          temp6 = rubase.degree_x;
          temp7 = rubase.degree_y;
          temp9 = rubase.degree_a1;
          temp10 = rubase.degree_a2;
                        //Finds degree of current rule
          temp11 = (rubase.degree_x)*(rubase.degree_y)*\
                   (rubase.degree_a1)*(rubase.degree_a2);
          for(j=i+1; j<counter;j++)
            {           //Reads next rule to campare againts current one
           fscanf(step3,"%s  %s\n %s %s\n %f %f %f %f %f",\
                    rubase.degree_xlit,rubase.degree_ylit,\
                    rubase.degree_a1lit,rubase.degree_a2lit,&rubase.degree_x,\
                    &rubase.degree_y,&rubase.degree_a1,&rubase.degree_a2);
```

```
            if(rule_dec[j] ==1)
            continue;

             if(!strcmp(temp1,rubase.degree_xlit) && !strcmp(temp2,rubase.degree_ylit))
             {
                    rule_dec[j] =1;
                          //Finds degree of importance for the rule
                    temp22 = (rubase.degree_x)*(rubase.degree_y)*\
                              (rubase.degree_a1)*(rubase.degree_a2);
                          //                                  //Check which rule has higher degree. If second rule
                    if(temp22 > temp11)                       //has higher degree than the first one, swap values
                    {                                         //and keep rule with higher degree. Note closing
                      strcpy(temp1,rubase.degree_xlit);       //brackets for loops: the process continues until first
                      strcpy(temp2,rubase.degree_ylit);       //rule is compared against all other rules in Rulesjs
                      strcpy(temp4,rubase.degree_a1lit);
                      strcpy(temp5,rubase.degree_a2lit);
                      temp6 = rubase.degree_x;
                      temp7 = rubase.degree_y;
                      temp9 = rubase.degree_a1;
                      temp10 = rubase.degree_a2;
                  }
             }
        }
            rule_dec[i] =1;
            //cout<<"(a)="<<temp6<<"   (b)="<<temp7<<"   (c)="<<temp9<<"  (d)="<<temp10;
            //getchar();
                  //Creates record in main Rbasejs file
            fprintf(rulebase,"%s  %s\n",temp1,temp2);
            fprintf(rulebase,"%s  %s\n",temp4,temp5);
            fprintf(rulebase,"%f  %f  %f  %f   %d\n\n",temp6,temp7,\
                  temp9,temp10,record_no);
            record_no++;

          }
         int count=0;
                  for(i=0;i< counter;i++)
                            if(rule_dec[i] ==1)
                            count++;

                  //    cout << "\nrules to write=" << count;
 fclose(rulebase);
 fclose(step3);
 cout << "The program is terminated. Final set of rules is in Rbasejs file.";
 getch();
}

                  // """""""""""""""""  END OF MAIN PROGRAM  """""""""""""""""
```

//MEMBERSHIP1-5 subroutines perform the fuzzification step for each input and output variable

*// **MEMBERSHIP1** defines the degree of membership for the first coefficient of linear regression*

```
void membership1(float input1, float *return_num, char  *return_lit)
{
 // cout<<"input1 is: "<<input1<<"    ";
  int interval;
                    //Breakdown of fuzzy regions for the first coefficient of linear regression
  if (input1 >= 0 && input1 < 0.166667)
     interval = 0;
  if (input1 >= 0.166667 && input1 < 0.333334)
     interval = 1;
  if (input1 >= 0.333334 && input1 < 0.500)
     interval = 2;
  if (input1 >= 0.500 && input1 < 0.666668)
     interval = 3;
  if (input1 >= 0.666668 && input1 < 0.833333)
     interval = 4;
  if (input1 >= 0.833333 && input1 < 1.)
     interval = 5;

  switch(interval){
                    //Cases 0-5 find the degree of membership according to the degree of membership function.
                    //Note that it is here that the crisp value is converted into both a fuzzy linguistic value and its
                    //specific degree of membership.
          case 0:
                      num = 5.999999*input1;
                      strcpy(lit,"S2");
                      break;
          case 1:
                      degree1 = -5.999999*input1 + 2.0;
                      strcpy(literal1,"S2");
                      degree2 = 5.999999*input1 - 1.0;
                      strcpy(literal2,"S1");
                      break;
          case 2:
                      degree1 = -5.999999*input1 + 3.0;
                      strcpy(literal1,"S1");
                      degree2 = 5.999999*input1 - 2.0;
                      strcpy(literal2,"CE");
                      break;
          case 3:
                      degree1 = -5.999999*input1 + 4.0;
                      strcpy(literal1,"CE");
                      degree2 = 5.999999*input1 - 3.0;
                      strcpy(literal2,"B1");
                      break;
          case 4:
                      degree1 = -5.999999*input1 + 5.0;
                      strcpy(literal1,"B1");
                      degree2 = 5.999999*input1 - 4.0;
                      strcpy(literal2,"B2");
                       break;
```

```
            case 5:
                    num = -5.999999*input1 + 5.9999999;
                    strcpy(lit,"B2");
                    break;
            default:
                 break;
        }

          if(interval && interval !=5)
          {
                  compare(degree1,degree2,literal1,literal2); //Call for Compare subroutine
                  num = num_value;                            //Save highest degree of membership
                  strcpy(lit,lit_value);                      //Save specific linguistic value
          }
      *return_num = num;                                      //Returns degree of membership function
      strcpy(return_lit,lit);                                 //Returns linguistic value
//      cout<<"the value of input1 is:"<<num<<"  "<<lit<<"\n";
}
          //MEMBERSHIP2 defines the degree of membership for the second coefficient of linear regression
void membership2(float input2, float *return_num, char  *return_lit)
{
//   cout<<"input2 is: "<<input2<<"   ";
   int interval;
                  //Breakdown of fuzzy regions for the second coefficient of linear regression
   if (input2 >= 0 && input2 < 0.166667)
          interval = 0;
   if (input2 >= 0.166667 && input2 < 0.333334)
          interval = 1;
   if (input2 >= 0.333334 && input2 < 0.500)
          interval = 2;
   if (input2 >= 0.500 && input2 < 0.666668)
      interval = 3;
   if (input2 >= 0.666668 && input2 < 0.833333)
      interval = 4;
   if (input2 >= 0.833333 && input2 < 1.0)
          interval = 5;

   switch(interval){
                  //Cases 0-5 find the degree of membership according to the degree of membership function.
                  //Note that it is here that the crisp value is converted into both a fuzzy linguistic value and its
                  //specific degree of membership.

          case 0:
               num = 5.999999*input2;
               strcpy(lit,"S2");
               break;

          case 1:
               degree1 = -5.999999*input2 + 2.0;
               strcpy(literal1,"S2");
               degree2 = 5.999999*input2 - 1.0;
               strcpy(literal2,"S1");
               break;
```

```
            case 2:
                degree1 = -5.999999*input2 + 3.0;
                strcpy(literal1,"S1");
                degree2 = 5.999999*input2 - 2.0;
                strcpy(literal2,"CE");
                break;
        case 3:
                degree1 = -5.999999*input2 + 4.0;
                strcpy(literal1,"CE");
                degree2 = 5.999999*input2 - 3.0;
                strcpy(literal2,"B1");
                break;
            case 4:
                degree1 = -5.999999*input2 + 5.0;
                strcpy(literal1,"B1");
                degree2 = 5.999999*input2 - 4.0;
                strcpy(literal2,"B2");
                break;
            case 5:
                num = -5.999999*input2 + 5.999999;
                strcpy(lit,"B2");
                break;
            default:
                 break;
            }
       //cout << "\nmem2:interval = " << interval << "\n degree1 = ";
            //cout << degree1 << "and degree2 = " << degree2;

            if(interval && interval !=5)
             {
                    compare(degree1,degree2,literal1,literal2);  //Call for Compare subroutine
                    num = num_value;                             //Save highest degree of membership
                    strcpy(lit,lit_value);                       //Save specific linguistic value

             }
       *return_num = num;                                        //Returns degree of membership value
       strcpy(return_lit,lit);                                   //Returns linguistic value
       //cout<<"the value of input2 is:"<<num <<"  "<<lit<<"\n";
}

          // MEMBERSHIP3 defines the degree of membership for the first output
void membership4(float output1, float *return_num, char  *return_lit)
{

  int interval;
  //cout << "\nmem4:output1 = "<< output1;
                    //Breakdown of fuzzy regions for the first output
  if (output1 >= 0.0 && output1<0.4)
     interval = 0;
  if (output1 >= 0.4 && output1<0.5)
     interval = 1;
  if (output1 >= 0.5 && output1<0.6)
     interval = 2;
```

```
if (output1 >= 0.6)
  interval = 3;

switch(interval){
                //Cases 0-5 find the degree of membership according to the degree of membership function.
                //Note that it is here that the crisp value is converted into both a fuzzy linguistic value and its
                //specific degree of membership.

        case 0:
             num = 0.0;
             strcpy(lit,"S1");
             break;
        case 1:
            degree1 = -10*output1 + 5.0;
            strcpy(literal1,"S1");
            degree2 = 10*output1 - 4.0;
            strcpy(literal2,"CE");
            break;
        case 2:
            degree1 = -10*output1 + 6.0;
            strcpy(literal1,"CE");
            degree2 = 10*output1 - 5.0;
            strcpy(literal2,"B1");
            break;
        case 3:
            num = 1.0;
            strcpy(lit,"B1");
            break;

        default:
            break;
        }
        //cout <<"\nmem4:interval = " << interval << "\n degree1 = ";
        //cout << degree1 <<"   and degree2 = " << degree2;


        if(interval && interval !=3)
         {
                compare(degree1,degree2,literal1,literal2);  //Call for Compare subroutine
                num = num_value;                             //Save highest degree of membership
                strcpy(lit,lit_value);                       //Save specific linguistic value

         }
   *return_num = num;                                        //Returns degree of membership value
   strcpy(return_lit,lit);                                   //Returns linguistic value
   //cout<<"\nnum ="<<num<<"    lit="<<lit;
   //getchar();
}
```

```
                    //MEMBERSHIP4 defines the degree of mebership for the second ouput
void membership5(float output2, float *return_num, char  *return_lit)
{
  int interval;
  //cout << "\nmem5:output2 = "<< output2;
                    //Breakdown of fuzzy regions for the second output
  if (output2 >= 0.0 && output2<0.4)
    interval = 0;
  if (output2 >= 0.4 && output2<0.5)
    interval = 1;
  if (output2 >= 0.5 && output2<0.6)
    interval = 2;
  if (output2 >= 0.6)
    interval = 3;

  switch(interval){
                    //Cases 0-5 find the degree of membership according to the degree of membership function.
                    //Note that it is here that the crisp value is converted into both a fuzzy linguistic value and its
                    //specific degree of membership.
          case 0:
               num = 0.0;
               strcpy(lit,"S1");
               break;
          case 1:
              degree1 = -10*output2 + 5.0;
              strcpy(literal1,"S1");
              degree2 = 10*output2 - 4.0;
              strcpy(literal2,"CE");
              break;
          case 2:
              degree1 = -10*output2 + 6.0;
              strcpy(literal1,"CE");
              degree2 = 10*output2 - 5.0;
              strcpy(literal2,"B1");
              break;
          case 3:
             num = 1.0;
             strcpy(lit,"B1");
             break;

          default:
              break;
          }
          //cout << "\nmem4:interval = " << interval << "\n degree1 = ";
          //cout << degree1 << "and degree2 = " << degree2;
          //getchar();

          if(interval && interval !=3)
           {
                    compare(degree1,degree2,literal1,literal2);  //Call for Compare subroutine
                    num = num_value;                             //Save highest degree of membership
                    strcpy(lit,lit_value);                       //Save specific linguistic value
           }
```

```
        *return_num = num;              //Returns degree of membership value
        strcpy(return_lit,lit);         //Returns linguistic value
    //cout<<"\nnum ="<<num<<"    lit="<<lit;
    //getchar();
}
```

*// **COMPARE** will determine which degree of membership is higher*

```
void compare(float degree1,float degree2,char *literal1,char *literal2)
{
        num_value=0;
        num_value= (degree1 > degree2) ? degree1 : degree2;     //Checks which degree of membership is higher
        if(degree1 > degree2)           //Swaps values. Saves higher numerical & linguistic values
                strcpy(lit_value,literal1);
        else
                strcpy(lit_value,literal2);
}
```

END OF FUZZIFICATION STEP

In this part of the program, a new set of raw data is used to check the accuracy of the rules generated in the previous steps. The program defines the interval in which the raw data lies, determines what rules in the rulebase are fired in parallel, an it applies the centroid rule to accomplish the defuzzification step and obtain values for control purposes

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <conio.h>
#include <math.h>

const int SIZE = 3;                    //Maximum size for linguistic variable. User can redifine according to
                                       //specific application
const int MAX = 360;                   //Maximum size raw data sets. User can redifine according to specific
                                       //application
const int BOX_SIZE= 5;                 //Size of mapping matrix. User redifines according to number of fuzzy
                                       //regions
const int BOX_SIZE_2= 5;               //Size of mapping matrix. User redifines according to number of fuzzy
                                       //regions
const int OUTPUT_SIZE= 3;              //Size of output control. User can redifine according to specific
                                       //application
                //Array defining all possible linguistic values for each fuzzy region. User defines array according
                //to number of fuzzy regions in the fuzzy design
char degrees[BOX_SIZE][3] ={"S2","S1","CE","B1","B2"};
char degrees_2[BOX_SIZE_2][3] ={"S2","S1","CE","B1","B2"};
char output_degrees[OUTPUT_SIZE][3] ={"S1","CE","B1"};
                //Array defining center of gravity values of each fuzzy region. User defines array according to
                //number of fuzzy regions in the fuzzy design.
float centers[OUTPUT_SIZE]={0.0,0.5,1.0};
float centers_2[OUTPUT_SIZE]={0.0,0.5,1.0};
int rule_dec[MAX];
char  angle1_box_lit[BOX_SIZE][BOX_SIZE][3]; //Contains linguistic value for 1st FAM
char  angle2_box_lit[BOX_SIZE][BOX_SIZE][3]; //Contains linguidtic value for 2nd FAM
float angle1_box_val[BOX_SIZE][BOX_SIZE];    //Contains numerical value for 1st FAM
float angle2_box_val[BOX_SIZE][BOX_SIZE];    //Contains numerical value for 2nd FAM

void  mapping();                       //Maos values to create FAM's. This function is used to create
                                       //the proper mapping of inputs into outputs and present the
                                       //map in matrix form
float match(char *,int);               //Matches the value of incoming raw data with the center
                                       //values of each fuzzy region. This function allows to find the
                                       //multiplying factor of each member of the centroid rule
                                       //equation
void  membership1(float , float *, float *, int *, int *);  //Defines specific fuzzy region for first coefficient
void  membership2(float, float *, float *, int *, int *);   //Defines specific fuzzy region for second coefficient
void  testing();                       //Defines control output for raw data. This function will
                                       //perform the defuzzification step by means of the centroid rule
```

```
                    // """""""""""""""""  MAIN PROGRAM  """""""""""""""""""""
void main()
{
    int counter,i,j;

        mapping();              //Call for Mapping subroutine

            //Displays 1st FAM in matrix form.
            for(i=0;i<BOX_SIZE;i++){
                for(j=0;j<BOX_SIZE;j++)
                    printf("%3s ",angle1_box_lit[i][j]);
                     cout << "\t";
                for(j=0;j<BOX_SIZE;j++)
                    printf("%5.3f  ",angle1_box_val[i][j]);
                cout << "\n";
            //getchar();
                    }
                cout<<"\n";

            //Displays 2nd FAM in matrix form
            for(i=0;i<BOX_SIZE_2;i++){
                for(j=0;j<BOX_SIZE_2;j++)
                    printf("%3s ",angle2_box_lit[i][j]);
                     cout << "\t";
                for(j=0;j<BOX_SIZE_2;j++)
                    printf("%5.3f  ",angle2_box_val[i][j]);
                cout << "\n";
                // getchar();
                    }
        testing();              //Call for testing subroutine
        cout<<"The program is over.  Check results in Results file";
        getchar();
}

            // """""""""""""""  END OF MAIN PROGRAM  """""""""""""""


            // ************        MAPPING        ***********
void mapping()
{
int counter,i,j,k,l;
FILE *fp;
        //degree1, degree2:variables for the two possible degrees of membership in the fuzzification step
        //degree_num, num_value, num:: variables that carry the highest degree of membership after fuzzification
        //literal1, literal2:variables for the two possible linguistic values in the fuzzification step
        //degree_lit, lit_value, lit: variables that carry the highest linguistic value after fuzzification
float degree1, degree2, slope,degree_num,num_value,num;
char  literal1[SIZE],literal2[SIZE],degree_lit[SIZE],lit_value[SIZE],lit[SIZE];
```

```
struct rbasejs                  //Defines structure Rbasejs. This is the file containing final set of fuzzy rules
        {
        float degree_x;         //Degree of membership of the first coefficient
        char  degree_xlit[SIZE]; //Linguistic value for the first coefficient
        float degree_y;         //Degree of membership for the second coefficient
        char  degree_ylit[SIZE]; //Linguistic value for the second coefficient
        float degree_a1;        //Degree of membership for the first output
        char  degree_a1lit[SIZE]; //Linguistic value for the first output
        float degree_a2;        //Degree of membership for the second output
        char  degree_a2lit[SIZE]; //Linguistic value for the second output
        }rubase;
                //Opens file Rbasejs for read only
if ((fp=fopen("rbasejs","r")) == NULL)
  {
        cout <<"Rulebase file can not open (r). Check for problems!! ";
        exit(1);
  }
        //The following loops organize the data in the form of a FAM bank. Inputs will be mapped into outputs
        //and presented in matrix form. There is always one FAM bank per output. FAM's are initialized to "0"
        //for the degree of membership, and to "xx"for the linguistic values
        for(i=0;i<BOX_SIZE;i++)
                for(j=0;j<BOX_SIZE;j++){
                        angle1_box_val[i][j]=0.;
                        strcpy(angle1_box_lit[i][j],"xx");
                      }

        for(i=0;i<BOX_SIZE_2;i++)
                for(j=0;j<BOX_SIZE_2;j++){
                        angle2_box_val[i][j]=0.;
                        strcpy(angle2_box_lit[i][j],"xx");
                      }
                //Reads first rule from "Rbasejs" file
while(fscanf(fp,"%s%s\n%s%s\n%f%f%f%f%d\n\n",
                rubase.degree_xlit,rubase.degree_ylit,\
                rubase.degree_a1lit,rubase.degree_a2lit,&rubase.degree_x,\
                &rubase.degree_y,&rubase.degree_a1,\
                &rubase.degree_a2,&counter) != EOF){

        //The following two loops perform the mapping between the input and output parts of the current rule.
        //Once the specific matrix posiiton is determined, the corresponding degree of membership and linguistic
        //value are placed in the matrix
        for(l=0;l<BOX_SIZE;l++){
                if(!strcmp(rubase.degree_xlit,degrees[l]))
                i=l;
                if(!strcmp(rubase.degree_ylit,degrees[l]))
                j=l;
        }
         if(!angle1_box_val[i][j]){
                strcpy(angle1_box_lit[i][j],rubase.degree_a1lit);
                angle1_box_val[i][j]=rubase.degree_a1;
                 cout<<"angle1 lit:"<<angle1_box_lit[i][j]<<"\n";
                 cout<<"angle1 mem:"<<angle1_box_val[i][j]<<"\n";
        }
```

```
        for(l=0;l<BOX_SIZE_2;l++){
                if(!strcmp(rubase.degree_xlit,degrees_2[l]))
                i=l;
                if(!strcmp(rubase.degree_ylit,degrees_2[l]))
                j=l;
        }
        if(!angle2_box_val[i][j]){
                strcpy(angle2_box_lit[i][j],rubase.degree_a2lit);
                        angle2_box_val[i][j]=rubase.degree_a2;
        }
}

 fclose(fp);
}


                        //*************  TESTING  ************
void testing()
{

        //i, j: arrays to save the location of a specific value in the MAP matrix
        //count1, count2: variables to carry location of specific value in the MAP matrix
        //degree_x, degree_y: variables containing degrees of membership for coefficients of linear regression
        //degree_a, degree_b:arrays receiving the two possible linguistic values after fuzzification
        //centroid1, centroid2: variables that contain values for required control action
        //map1, map1_2:variables containing center values for fired fuzzy regions
        //denominator: variable containing denominator value for centroid rule
        //numerator, numerator2: variables containing numerator values for centroid1 and centroid2
        //map_lit, map_lit_2: variables containing linguistic values of fired fuzzy regions
        //summ, error, rms_error: variables used to compute final rms error of fuzzy system.
char  filename[20];
int i[2],j[2];
int count1=0,count2=0,counter=0;
float degree_x, degree_y;
degree_b[2], degree_a[2];
float centroid1,centroid2, map1,map1_2,summ=0.,error=0.,rms_error=0.;
float denominator=0.,numerator=0.,numerator2=0.;
char map_lit[3],map_lit_2[3];
FILE *fp;
FILE *deffuzzi;
FILE *answer;

struct testbase         //Definition of structure testbase. This file contains data sets to test the control rules
        {
        float input1;   //Defines the first coefficient of linear regression
        float input2;   //Defines the second coefficient of linear regression
        float output1;  //Defines the first output
        float output2;  //Defines the second output
        }db;
```

```
struct answer           //Definition of structure Answer. This file contains the final control output
        {
        float input1;           //Defines the first coefficient of linear regression
        float input2;           //Defines the second coefficient of linear regression
        float output1;          //Defines the first output
        float output2;          //Defines the second output
        float centroid1;        //Defines first control output
        float centroid2;        //Defines second control output
        }anbase;
/*
cout << "Enter the name of the input file:" ;  //open database for reading
cin >> filename;
*/
                //Opens file "n32test1" containing testing data.  Read only
strcpy(filename,"n32test1");
if ((fp=fopen(filename,"r")) == NULL)
  {
    cout << "Test file can not be open (r). Check for problems!!";
    exit(0);
  }
                //Opens file "nnanswer" to record control output.  Write only
if ((deffuzzi=fopen("nnanswer","w")) == NULL)
  {
    cout << "File Results can not open to write. Check for problems!!";
    exit(0);
  }
        int dummy1,dummy2,adjust_1,adjust_2,inconsistent=0;

                //Reads record from testing file.  Operation ends for EOF.
        while(fscanf(fp,"%f %f\n %f %f\n",\
                &db.input1,&db.input2,&db.output1,&db.output2) != EOF){

                //These subroutines will determine specific fuzzy regions where test data lies
        membership1(db.input1,&degree_a[0],&degree_a[1], &i[0],&i[1]);
        membership2(db.input2,&degree_b[0],&degree_b[1], &j[0],&j[1]);

                //This blocks determines the number of rules that have been activated in parallel.  In this case
                //notice that there will be 4 rules fired at the same time.  It is necessary to collect all those rules,
                //determine the degrees of membership of each input, multiply them by the central value of the
                //fuzzy region, and add them up to find the numerator af the centroid.

        float dummy_a,dummy_b;
          for(count1=0;count1 < 2;count1++){
                dummy1=i[count1];
                dummy_a=degree_a[count1];
        for(count2=0;count2 < 2;count2++){
                dummy2=j[count2];
                dummy_b=degree_b[count2];
                                //Assigns linguistic value for input data
                        strcpy(map_lit,angle1_box_lit[dummy1][dummy2]);
                        strcpy(map_lit_2,angle2_box_lit[dummy1][dummy2]);
```

```
                    //Calls match subroutine to define central value for fuzzy regions
                map1 = match(map_lit,1);
                map1_2 = match(map_lit_2,2);
                    //Finds numerator and denominator values for both control variables
                denominator+= dummy_a*dummy_b;
                numerator+=dummy_a*dummy_b*map1;
                numerator2+=dummy_a*dummy_b*map1_2;
        }
      }
          if(denominator){
          centroid1 =numerator/denominator;        //Determines first centroid control value
          centroid2 =numerator2/denominator;       //Determines second centroid control value
          }
    else
          cout <<"wild data, theta1\n";
    if(centroid1>centroid2){
          adjust_1=1;
          adjust_2=0;
          }
    else{
          adjust_1=0;
          adjust_2=1;
    }
    if(!(((int)db.output1 == adjust_1) &&((int)db.output2 == adjust_2))){
                inconsistent++;
          //If control output is different from the one expected, send it to screen
    printf("%f  %f\n",db.input1,db.input2);
          printf("%f  %f   %d  %d\n",db.output1,db.output2,adjust_1,adjust_2);
          printf("centroids %f  %f\n",centroid1,centroid2);
          printf("map1= %s map2 = %s \n",map_lit,map_lit_2);
          cout << "inconsistent = " << inconsistent << "\n";
    getchar();
    }
          //Create record in final "nnanswer" file
    fprintf(deffuzzi,"%f  %f   %f  %f\n",db.input1,db.input2,\
          db.output1,db.output2);
    fprintf(deffuzzi,"%f  %f\n\n",centroid1,centroid2);

    error=((db.output1-adjust_1)*(db.output1-adjust_1)+\
          (db.output2-adjust_2)*(db.output2-adjust_2));
    summ+=error;
  counter++;
    cout<<"conter is"<<counter<<"\n";
    numerator=0.;
    numerator2=0.;
}
cout << "inconsistent = " << inconsistent << "\n";
rms_error=sqrt(summ/(counter*2));
cout<<"counter="<<counter<<"\n";
cout<<"rms_error = "<<rms_error<<"\n";
 fclose(deffuzzi);
    fclose(fp);
}
```

```
                //MEMBERSHIP1 defines the fuzzy regions which are activated by first coeffificent
void membership1(float input1, float *return_num1,float *return_num2, int *v1,int *v2)
{
   int interval;
   int int_v1, int_v2;       //Variables containing the fuzzy interval which is fired by incoming data
   float degree1, degree2;

  if (input1 >= 0 && input1 < 0.166667)
     interval = 0;
  if (input1 >= 0.166667 && input1 < 0.333334)
     interval = 1;
  if (input1 >= 0.333334 && input1 < 0.500)
     interval = 2;
  if (input1 >= 0.500 && input1 < 0.666668)
     interval = 3;
  if (input1 >= 0.666668 && input1 < 0.833333)
     interval = 4;
  if (input1 >= 0.833333 && input1 < 1.0)
     interval = 5;

  switch(interval){
                //Cases 0-5 find the degree of membership according to the degree of membership function. Note
                //that in this case we would like to know the fuzzy region in which the incoming data fires the
                //fuzzy system. The fuzzy intervals are saved under int_v1 and int_v2
        case 0:
                  degree1 = 5.999999*input1;
                  degree2 = degree1;
                  int_v1=1;
                  int_v2=1;
                  break;
        case 1:
                  degree1 = -5.999999*input1 + 2.0;
                  degree2 = 5.999999*input1 - 1.0;
                  int_v1=1;
                  int_v2=2;
                  break;
        case 2:
                  degree1 = -5.999999*input1 + 3.0;
                  degree2 = 5.999999*input1 - 2.0;
                  int_v1=2;
                  int_v2=3;
                  break;
        case 3:
                  degree1 = -5.999999*input1 + 4.0;
                  degree2 = 5.999999*input1 - 3.0;
                  int_v1=3;
                  int_v2=4;
                  break;
```

```
        case 4:
                degree1 = -5.999999*input1 +  5.0;
                degree2 = 5.999999*input1 - 4.0;
                int_v1=4;
                int_v2=5;
                break;

        case 5:
                degree2 = -5.999999*input1 + 5.999999;
                degree1 =degree2;
                int_v1=5;
                int_v2=5;
                break;
        default:
                break;
      }
    *v1 = --int_v1;
    *v2 = --int_v2;
    float temp;int tempint;
        if(degree2 > degree1){
                temp = degree1;
                degree1 = degree2;
                degree2 = temp;
                tempint=  *v1;
                *v1 = *v2;
                *v2 =tempint;

     }
}
```

//MEMBERSHIP2 defines the fuzzy regions which are activated by the second coefficient

```
void membership2(float input2, float *return_num1,float *return_num2, int *v1,int *v2)
{
  int interval;
  int int_v1, int_v2;
  float degree1, degree2;

  if (input2 >= 0 && input2 < 0.166667)
     interval = 0;
  if (input2 >= 0.166667 && input2 < 0.333334)
     interval = 1;
  if (input2 >= 0.333334 && input2 < 0.500)
     interval = 2;
  if (input2 >= 0.500 && input2 < 0.666668)
     interval = 3;
  if (input2 >= 0.666668 && input2 < 0.833333)
     interval = 4;
  if (input2 >= 0.833333 && input2 < 1.0)
     interval = 5;

  switch(interval){
                //Cases 0-5 find the degree of membership according to the degree of membership function.  Note
                //that in this case we would like to know the fuzzy region in which the incoming data fires the
                //fuzzy system.  The fuzzy intervals are saved under int_v1 and int_v2
```

```
    case 0:
                degree1 = 5.999999*input2;
                degree2 = degree1;
                int_v1=1;
                int_v2=1;
                break;
    case 1:
                degree1 = -5.999999*input2 + 2.0;
                degree2 = 5.999999*input2 - 1.0;
                int_v1=1;
                int_v2=2;

                break;
    case 2:
                degree1 = -5.999999*input2 + 3.0;
                degree2 = 5.999999*input2 - 2.0;
                int_v1=2;
                int_v2=3;
                break;
    case 3:
                degree1 = -5.999999*input2 + 4.0;
                degree2 = 5.999999*input2 - 3.0;
                int_v1=3;
                int_v2=4;
                break;
    case 4:
                degree1 = -5.999999*input2 + 5.0;
                degree2 = 5.999999*input2 - 4.0;
                int_v1=4;
                int_v2=5;
                break;
    case 5:
                degree2 = -5.999999*input2 + 5.999999;
                degree1 =degree2;
                int_v1=5;
                int_v2=5;
                break;
    default:
                break;
  }
*v1 = --int_v1;
*v2 = --int_v2;
        float temp;int tempint;
    if(degree2 > degree1){
            temp = degree1;
            degree1 = degree2;
            degree2 = temp;
            tempint=  *v1;
            *v1 = *v2;
            *v2 =tempint;
    }
```

```
    *return_num1 = degree1;
    *return_num2 = degree2;
    //cout<<"degree1(2)="<<degree1<<"     "<<"degree2="<<degree2<<"\n";
}

                /*************** MATCH ***************/

        //This function finds the values corresponding to the fuzzy regions activated by the incoming data. The
        //subroutine will determine the center value of the fuzzy region as specified in the array degrees[i]
float match(char *string,int theta_index)
{
float value=0;
int i,upper_limit;
int result;

upper_limit=OUTPUT_SIZE;

        for(i=0;i<upper_limit;i++){
                if(theta_index==1){
                if(!strcmp(string,output_degrees[i])){
                value = centers[i];
                break;
                }
            }
                if(theta_index==2){
                if(!strcmp(string,output_degrees[i])){
                value = centers_2[i];
                break;
                }
            }
        }
    return value;

}
```

END OF DEFUZZIFICATION PROCEDURE.

END OF PROGRAM

ABSTRACT

Traditional manufacturing schemes, thereon referred as manufacturing planning, have not been able to provide tools with enough flexibility to adapt to the many constraints of every day production. These approaches are quantitative in nature and their application to control problems fails to perform satisfactorily when the information required to answer a query is non-linear or ill-defined. With the advent of high quality requirements, conventional manufacturing systems have been forced to explore innovative alternatives to cope with the pressure of global competitiveness. Neural Networks and Machine Learning, with their ability to learn from examples, have been proposed early on for solving non-linear control problems adaptively. More recently, Fuzzy Logic has emerged as a promising approach for controlling processes.

This work studies the application of Fuzzy Logic through a methodology proposed by Wang and Mendel from the University of Southern California. This methodology is of relevant interest because it provides fuzzy model builders with a unique tool to generate rules from fuzzy sets which combine both numerical and linguistic information.

The methodology is explained in detail and is applied to two uncomplicated case studies. The first case study is a typical control problem an it relates to robot motion control. A simulation is developed to obtain various information describing the movement of a two-degree-of-freedom robot arm. The collected pairs are used for the training of the fuzzy controller which is then tested with new data.

Although Fuzzy Logic is now quite popular in commercial control applications, there is little reported in the area of manufacturing planning and control. In the second case study, the Wang-Mendel methodology is applied to a basic job shop scheduling problem. Performance of the fuzzy machine is compared against a traditional backpropagation neural network and Quinlan's ID3 machine learning technique.